**ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA**

**SCUOLA DI INGEGNERIA E ARCHITETTURA**

*CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA*

**TESI DI LAUREA**

in

Data Mining M

# Analisi e sviluppo di una piattaforma blockchain per la supply chain

CANDIDATO:

Riccardo Fiorini

RELATORE:

Chiar.mo Prof. Claudio Sartori

CORRELATORE:

Leonardo Bruni

Anno Accademico 2018-2019

Sessione III

# Abstract

Il termine blockchain ha origine nel 2009 con la nascita di Bitcoin[1]: come il nome suggerisce, si tratta di una sequenza di blocchi di dati collegati a catena. Questa sequenza è immutabile ed ogni blocco è legato al precedente e al successivo grazie a un sistema di codici hash, sviluppato per impedire operazioni fraudolente. Questa tecnologia disruptive ha subito suscitato interesse, principalmente per il proprio modello d'interazione basato su un sistema di fiducia distribuito, caratterizzato dall'assenza di un'autorità centrale. L'ambito di questa tecnologia era inizialmente solo quello delle criptovalute, ma ben presto si è allargato ad altri settori, come ad esempio la gestione della supply chain, che sarà presa in analisi in questo progetto di tesi.

Perché gestire la supply chain con la blockchain? Ci sono alcune necessità di gestione comuni a tutte le aziende quando si parla di rapporto con i fornitori: visione comune del prodotto, certificati, tracciabilità, ordini d'acquisto, sviluppo collaborativo… L'analisi di questi requisiti, unita alla necessità di avere attori ben noti con cui comunicare, porta inevitabilmente alla necessità di abbandonare il modello iniziale di blockchain pubblica *permissionless* e alla migrazione verso le cosiddette blockchain private *permissioned*. Le blockchain private provano a superare alcune delle limitazioni delle blockchain pubbliche, introducendo un'autorità che restringe l'accesso alla rete ad una cerchia ristretta di soli peer autorizzati. La piattaforma utilizzata in questa tesi è Hyperledger Fabric, le cui caratteristiche ben si sposano con i requisiti di uno scenario business-to-business con aziende che interagiscono tra loro in una blockchain privata. L'obiettivo finale della tesi è quello di sviluppare, con Hyperledger Fabric[2], una simulazione di quello che potrebbe essere il modello d'interazione tra un'azienda e un suo fornitore, con questi due attori che effettuano transazioni regolate da un'entità centrale, chiamata *orderer*, la quale produce gli effettivi blocchi che saranno inseriti in blockchain. Chiaramente il modello può essere espanso anche ad altri fornitori, così che tutta la filiera possa essere gestita tramite blockchain.

[1] La dicitura "Bitcoin" si utilizza per indicare l'architettura, "bitcoin" per la criptovaluta.

[2] https://www.hyperledger.org/projects/fabric

*Grazie…*

*Al mio relatore, professor Claudio Sartori, e al professor Federico Ravaldi,*
*per avermi dato la possibilità di svolgere questa tesi.*

*Ai miei team leader Leonardo e Francesco, al mio tutor Fabio e a tutti i membri dei team Shark e Gamble,*
*per avermi accolto e guidato come tesista prima, come collega poi.*

*Alla mia famiglia che mi ha supportato ogni giorno della mia vita.*

*Ma il grazie più grande va a Federica, che mi ha supportato e sopportato negli ultimi quattro anni,*
*insegnandomi il valore dell'impegno come studente, come lavoratore, come uomo e come compagno.*
*A due che come noi non si son persi mai.*

*"Ringrazio il cielo di essere su questo palco…"*

# Sommario

# Capitolo 1

# Introduzione

## *1.1 Scenario analizzato nella tesi*

Questa tesi si occupa di analizzare come la gestione della supply chain possa essere effettuata da un'azienda sfruttando la tecnologia blockchain. In particolare, per permettere solo ad attori autorizzati di partecipare alla rete, sarà utilizzato il paradigma delle private blockchain. Immedesimandoci in un'azienda che si occupa di progettare occhiali, potremmo pensare che essa collabori con un'altra azienda, la quale sceglie il miglior materiale per le parti del design fornito. La seconda azienda avrà anch'essa la sua rete di fornitori di materiali, che potrebbe gestire allo stesso modo, creando un fitto intreccio di scambi tra attori diversi all'interno della stessa blockchain. Il framework scelto per sviluppare l'architettura, Hyperledger Fabric, permette di gestire all'interno della stessa rete blockchain canali privati di comunicazione tra le aziende: in questo modo ogni azienda vede solo i dati a cui è autorizzata ad accedere, in modo da eliminare eventuali problemi legati alla trasparenza dei dati a possibili competitor.

Il progetto di tesi è stato svolto in azienda, presso Iconsulting Spa. Iconsulting è un'azienda nata nel 2001 che si occupa di consulenza informatica in numerosi ambiti, quali Advisory, Big Data Platform, Business Analytics, Location Analytics, Machine Learning e, appunto, Blockchain. Per Iconsulting «i dati sono le scintille che innescano la conoscenza» e il loro lavoro mira proprio ad aiutare i clienti a «sviluppare la loro relazione con i dati e ad acquisire consapevolezza nel processo decisionale»[1].

Come punto di partenza di questa tesi, è stato analizzato un progetto recentemente sviluppato (e già entrato in funzione) per la gestione della supply chain di una grossa azienda di moda che si occupa di occhialeria. Hanno aderito al progetto numerose aziende, tra cui: 6 produttori di montature, un produttore di lenti, 3 produttori di astucci, 2 fornitori di materie prime, 3 produttori di componenti e un istituto di certificazione.

[1] Sito web di Iconsulting: https://www.iconsulting.biz/

Gestire la supply chain con blockchain ha evidenti vantaggi in fatto di controllo dell'integrità dei dati, infatti ogni azienda ha interesse che questi non vengano cambiati in quanto possono essere state già state prese decisioni importanti sulla base di essi e non può accettare cambiamenti non desiderati. In questo senso la blockchain è perfetta in quanto ogni operazione che viene eseguita è registrata in modo permanente e verificabile in caso di dispute, perché le transazioni sono firmate e registrate, senza la possibilità di modifiche retroattive. Un altro vantaggio deriva dalla semplificazione delle comunicazioni tra le aziende: spesso gli scambi di richieste, documenti, ecc., possono avvenire tramite e-mail o per contratto scritto, con possibilità di smarrire informazioni e quindi dover ripetere il tutto. Gestendo in blockchain tutti questi aspetti, non è possibile perdere informazioni, in quanto tutto è tracciato e verificabile.

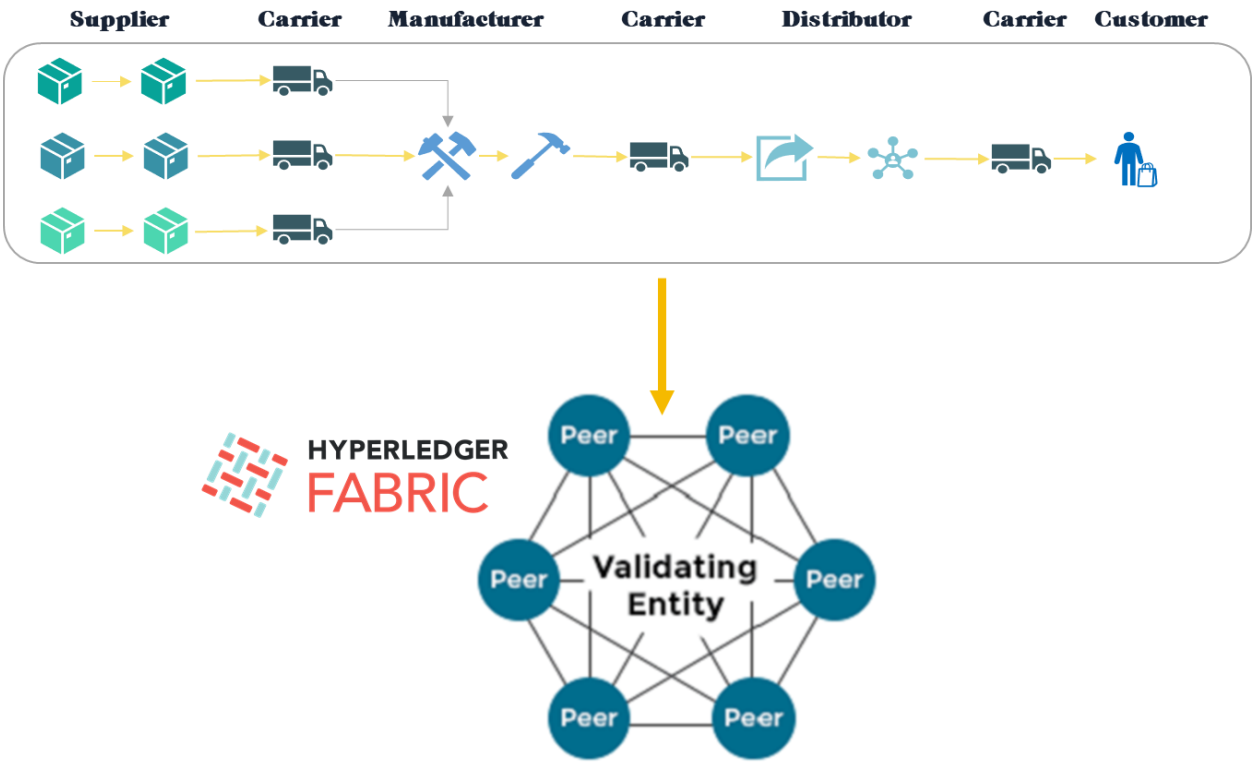



**Figura 1: gestione della supply chain con blockchain**

## *1.2 Obiettivi della tesi*

La prima parte della trattazione ha il compito di introdurre le caratteristiche fondamentali di questa tecnologia, analizzando punti di forza e di debolezza delle varie soluzioni. Il

passo successivo descrive le decisioni prese per il design della parte progettuale che mira a ottimizzare il funzionamento dell'architettura già esistente prima introdotta: sarà costruita una rete blockchain con più nodi, scritto il codice per effettuare le transazioni necessarie alla gestione di un insieme di eyewear e infine sviluppata un'applicazione applicazione web di test per fornire agli utenti la possibilità di interagire con la blockchain.

## *1.3 Contenuto della tesi*

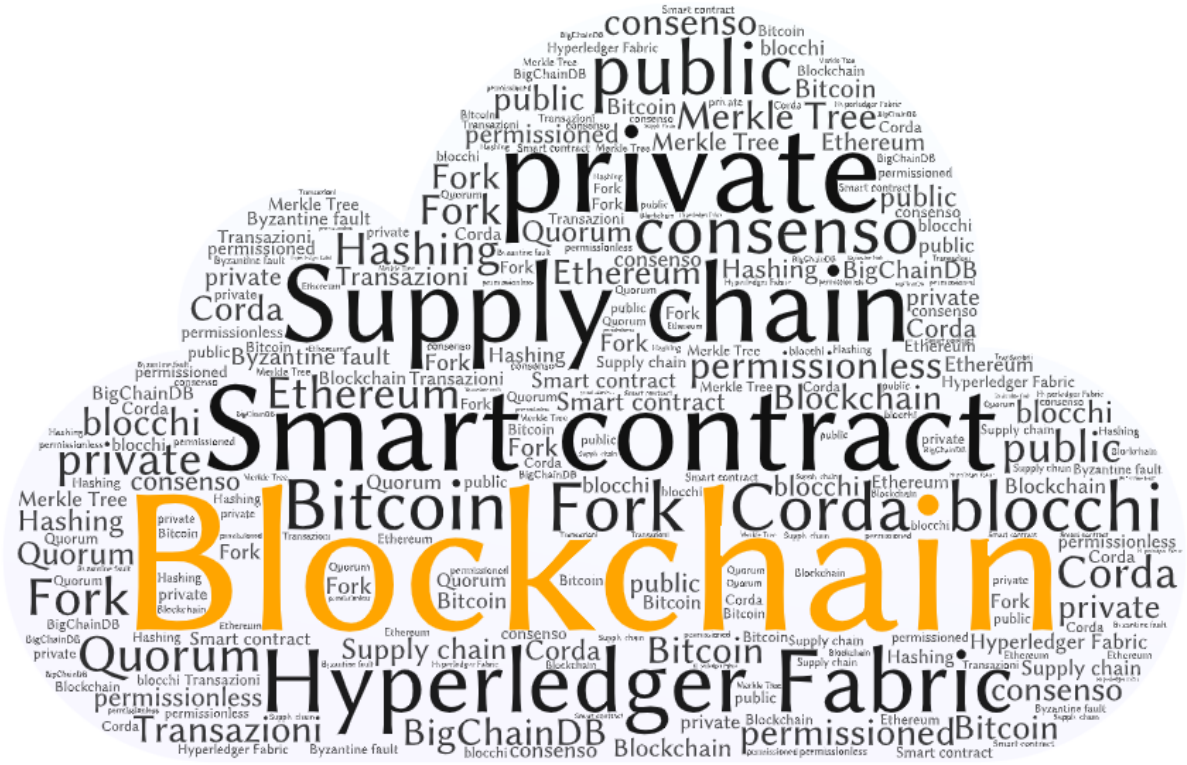
La tesi è strutturata come segue:

- Il capitolo 2 dà una visione generale della blockchain e del problema del consenso distribuito, focalizzando la trattazione sull'analisi delle varie categorie di blockchain.
- Il capitolo 3 descrive la piattaforma di blockchain privata Hyperledger Fabric.
- Il capitolo 4 descrive la parte progettuale riguardante il caso d'uso analizzato. Saranno forniti dettagli d'implementazione e spiegazioni sui principali passi da percorrere per costruire una rete con Hyperledger Fabric. Inoltre, sarà mostrato il funzionamento dell'applicazione sviluppata per interagire con la blockchain.
- Il capitolo 5 conclude la tesi facendo un riassunto della teoria e della parte progettuale, analizzando l'architettura ottenuta e proponendo direzioni di ricerca future riguardo Hyperledger Fabric.

# Capitolo 2

## Blockchain: a new business ecosystem

Questo capitolo tratta la tecnologia Blockchain sotto i suoi vari aspetti e ne spiega il funzionamento. La sezione 2.1 introduce velocemente i concetti fondamentali che saranno poi affrontati nelle successive sezioni. La sezione 2.2 illustra brevemente la storia dello sviluppo della blockchain. La sezione 2.3 verranno descritte più in dettaglio le caratteristiche fondamentali di questa tecnologia, tra cui: blocchi, blockchain pubbliche e private, algoritmo di consenso e *smart contract*. La sezione 2.4 conclude il capitolo, analizzando le principali caratteristiche di alcune delle soluzioni blockchain più interessanti.



## 2.1 Introduzione

Una blockchain è una lista concatenata di record, chiamati blocchi, collegati crittograficamente. Ogni blocco è formato da un'intestazione, contenente l'hash del blocco precedente, e un corpo, che contiene i dati transazionali. Una blockchain rappresenta il "libro mastro" di tutte le transazioni registrate su una rete, le quali sono memorizzate in maniera permanente. Per evitare operazioni fraudolente, ogni record in un blocco non può essere retroattivamente alterato senza la conseguente modifica di tutti i blocchi successivi, il che richiederebbe il consenso di più della metà dei nodi nella rete. L'uso della blockchain rimuove la caratteristica di infinita riproducibilità di un asset digitale[2], confermando che ogni unità di valore è stata trasferita solo una volta e risolvendo il cosiddetto "problema della doppia spesa"[3].

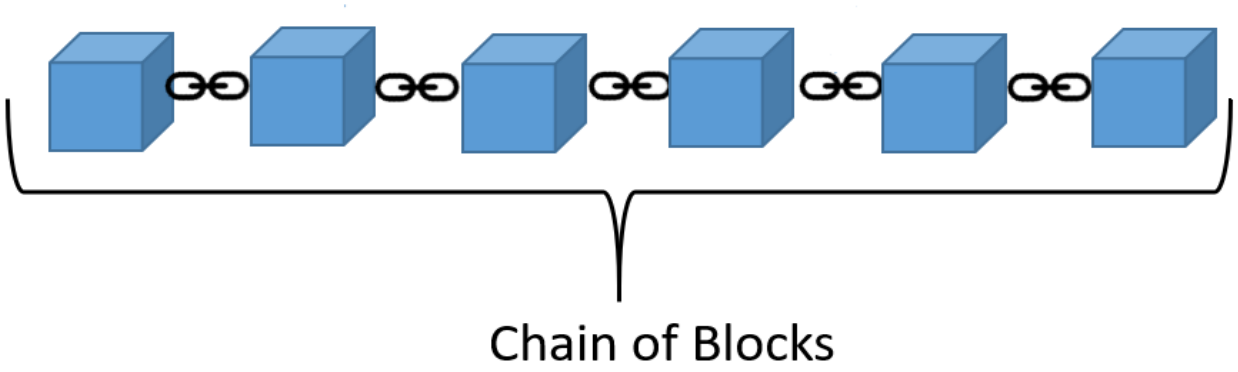



**Figura 2: rappresentazione di una catena di blocchi.**

Una blockchain è tipicamente realizzata su di una rete peer-to-peer, che aderisce collettivamente a un protocollo per la comunicazione tra i nodi e la validazione dei blocchi. In questo senso, il sistema blockchain fa parte delle cosiddette Distributed

[2] Un asset digitale è una qualsiasi risorsa rappresentabile tramite bit a cui è associata una proprietà o un diritto d'uso.

[3] Il problema della doppia spesa consiste nell'uso dello stesso titolo per effettuare due o più acquisti.

Ledger Technologies (DLT)[4]. Questo sistema permette ai partecipanti di verificare le transazioni senza la presenza di una terza parte, in modo relativamente semplice.

Attualmente sono presenti numerose tecnologie blockchain, differenti per uso e implementazione, tutte sviluppate in seguito alla nascita della Bitcoin Blockchain. Una prima classificazione che può essere fatta su queste tecnologie è tra blockchain pubbliche o private, che si differenziano per la politica con cui un nodo può unirsi alla rete. È importante osservare che il concetto di blockchain non si lega solo a quello di criptovaluta, come erroneamente parte dell'opinione pubblica immagina, ma si riferisce a un concetto di molto più ampio respiro, un paradigma di consenso distribuito e di registrazione di transazioni che può essere applicato a tanti ambiti (supply chain, registrazione dei territori, strumenti di governance...). Questi concetti saranno approfonditi nel seguito della tesi, in modo che le caratteristiche principali di questa tecnologia siano chiarite prima di affrontare la specifica tematica progettuale.

## *2.2 Storia*

Il primo lavoro su una catena di blocchi resa crittograficamente sicura è stato descritto nel 1991 da Stuart Haber e W. Scott Stornetta[5], i quali volevano implementare un sistema in cui i timestamp dei documenti non potessero essere manomessi. Nel 1992, Bayer, Haber e Stornetta inserirono i Merkle Tree al design[6], i quali ne migliorarono l'efficienza permettendo a più documenti di essere incorporati in un blocco.

La prima blockchain è stata concettualizzata da una persona nota come Satoshi Nakamoto (la cui identità è ancora ignota) nel 2008, il quale progettò un metodo *Hashcash-like* per il timestamp dei blocchi senza il bisogno che essi fossero firmati da una parte fidata prima di essere aggiunti alla catena. Il progetto fu implementato l'anno

[4] Un *distributed ledger* (letteralmente "libro mastro distribuito") è un database replicato e condiviso, i cui dati sono sincronizzati correttamente su più peer grazie a un algoritmo di consenso, senza alcuna forma di centralizzazione.

[5] Staurt Haber & W. Scott Stornetta, *How to Time-Stamp a Digital Document,* 1991

[6] Staurt Haber, W. Scott Stornetta & Dave Bayer*, Improving the Efficiency and Reliability of Digital Time-Stamping,* 1992

seguente da Nakamoto come componente fondamentale della criptovaluta bitcoin, per la quale funge da libro mastro pubblico per tutte le transazioni della rete. Le parole *block* e *chain* erano usate separatamente nel paper originale di Nakamoto, ma acquisirono popolarità come parola singola negli anni seguenti. Il primo acquisto effettuato con bitcoin fu una pizza, nel 2009.

Bitcoin fu così la prima moneta digitale a risolvere il problema della doppia spesa senza il bisogno di un'entità centrale che fungesse da garante. La sua realizzazione ha ispirato numerose altre criptovalute, sempre basate su blockchain pubblica, il cui numero si stima sia superiore alle 2000 (effettivamente acquistabili). Nell'Agosto 2014, la dimensione totale di tutti i blocchi di bitcoin raggiunse i 20 GB. Negli anni successivi la crescita è stata continua, arrivando alla dimensione attuale che si attesta sui circa 260 GB (a gennaio 2020)[7].

Nel 2014 si iniziò a usare il termine "blockchain 2.0", per indicare un nuovo modo d'utilizzo della blockchain. L'obiettivo è quello di estendere il concetto ben oltre le criptovalute, principalmente con l'integrazione di *smart contract* e la creazione di blockchain private; tale svolta tecnologica ha già iniziato a rivoluzionare il modo in cui le aziende concepiscono le loro attività, infatti è sempre più comune un utilizzo *enterprise* della blockchain, spostando l'attenzione dalle solo criptovalute a tanti altri ambiti applicativi e scenari commerciali. Secondo quanto emerge da una ricerca presentata al convegno *Blockchain Business Revolution* dell'Aprile 2019, in Italia l'85% dei manager d'azienda intervistati dichiara di conoscere la blockchain e il 68% ritiene che sia una tecnologia importante per lo sviluppo economico del Paese[8].

Numerosi stati si stanno muovendo nella direzione della blockchain, investendo e stimolando le imprese a sviluppare nuove soluzioni, valutando la creazione di criptovalute nazionali e riorganizzando la gestione tramite blockchain di vari settori dell'amministrazione pubblica. Dagli Stati Uniti agli Emirati Arabi, passando per la Cina, i paesi a maggior sviluppo tecnologico si sono mossi per non rimanere indietro su un

[7] Statistiche disponibili alla pagina: https://www.blockchain.com/charts/blocks-size

[8] Articolo completo: https://www.zerounoweb.it/software/blockchain/blockchain-cresce-la-conoscenza-per-il-20-degli-italiani-e-l85-delle-imprese/

campo che sarà certamente decisivo in futuro. Il Parlamento europeo con la Risoluzione del 3 ottobre 2018 “sulle tecnologie di registro distribuito e blockchain: creare fiducia attraverso la disintermediazione”, ha riconosciuto la rilevanza della blockchain come strumento «che può democratizzare i dati e rafforzare la fiducia e la trasparenza», in quanto «rafforza l’autonomia dei cittadini» e migliora «l’efficienza dei costi delle transazioni eliminando intermediari e costi di intermediazione, oltre ad aumentare la trasparenza delle transazioni». Per quanto riguarda l’Italia, Il Ministero dello Sviluppo Economico ha selezionato nel mese di dicembre 2018 trenta membri del “Gruppo di Esperti di alto livello per l’elaborazione della strategia nazionale sulle tecnologie basate sui registri distribuiti e blockchain”, nell’ambito di un processo diretto a favorire lo sviluppo della tecnologia Blockchain a livello nazionale. Nel febbraio 2019 è stato approvato il DL semplificazioni, che riconosce, tra le varie cose, la validità degli *smart contract*, elevando l’Italia allo status di primo stato europeo a porne una legislazione specifica.

### 2.2.1 Internet of value

Tutte le tecnologie rivoluzionarie hanno avuto una storia travagliata contrassegnata da un lento inizio e poi da una rapida adozione grazie ad una "*killer application*" in grado di sfruttarne appieno le capacità. Nel caso di Distributed Ledger basati su crittografia, il ruolo di killer lo ha svolto Bitcoin, rivoluzionando la concezione di criptovalute e di scambio di valore online in un sistema distribuito senza intermediari. Fortunatamente il destino delle blockchain non è legato solo a Bitcoin e alle criptovalute, altrimenti si parlerebbe solo di un sistema di valuta fortemente influenzato dagli andamenti del mercato e alle cosiddette “bolle”, come quella del 2018 che causò una svalutazione massiccia del valore dei bitcoin. Con il tempo sono nate tecnologie come Ethereum[9] che hanno come scopo non tanto la gestione delle criptovalute, quanto quella dello sviluppo di *smart contract* per far fronte a casi d’uso più complessi. La vera novità che scaturirà dall’uso dalla blockchain nei prossimi anni sarà l’abilitazione del cosiddetto Internet of Value.

[9] https://ethereum.org/it/

Per anni si è sentito parlare di "Web 2.0" e delle meravigliose possibilità che avrebbe offerto ai propri utenti. Grandi aziende quali Google e Facebook hanno prosperato seguendo un approccio da centralizzatori e acquisendo sempre più dati dai propri utenti. L'Internet of Value si pone come alternativa a questo approccio, adottando la decentralizzazione in un sistema distribuito grazie alla partecipazione attiva degli utenti e prevedendo la possibilità di spostare valore in maniera semplice, efficace e sicura tramite una rete peer-to-peer ad incentivi. La blockchain potrà essere la tecnologia per abilitare questo switch, con la possibilità di rappresentare asset, di produrre e gestire *coin,* di proteggere la privacy degli utenti, ecc.…

## *2.3 Descrizione*

Una blockchain può essere considerata come un database distribuito *append-only,* con i dati transazionali memorizzati in blocchi. Quando si vuole proteggere l'integrità dei dati memorizzati, che possono essere minacciati dall'esistenza di numerosi lettori e scrittori, la blockchain è una soluzione tecnologica per lo storage da valutare alternativamente a un database tradizionale. L'incertezza dei partecipanti alla rete riguardo alla sicurezza dei dati è minima, in quanto i meccanismi di timestamp e di consenso collaborativo da parte dei nodi distribuiti in rete assicura robuste garanzie sull'integrità dei dati. Nonostante i record in blockchain non siano immodificabili, le blockchain possono essere considerate sicure per design in quanto un'alterazione retroattiva di dati richiede che ci sia almeno il 50% + 1 del consenso, questo permette di avere anche un sistema di computazione distribuita con alta tolleranza ai guasti.

Quando in questa sezione si parla di blockchain in termini generali, si fa riferimento alle sue caratteristiche fondamentali, concepite inizialmente da Nakamoto per la Bitcoin Blockchain. Quando si descrive la caratteristica di una particolare implementazione di blockchain, verrà specificato.

### 2.3.1 I blocchi

La blockchain prende il nome dalla sua unità di memorizzazione di base, il blocco. Il primo blocco della catena è chiamato *genesis block* ed è il punto di partenza di una lista

in continua crescita di blocchi collegati, che può essere rappresentata come una catena. Un blocco è formato da un'intestazione e un corpo, più propriamente header e body. L'header può contenere vari campi, in particolare in Bitcoin abbiamo:

- Versione: indica la versione del software utilizzato.
- Hash del blocco precedente: funge da puntatore tra il blocco corrente e il blocco precedente della catena.
- Merkle root: è un hash che riassume la lista delle transazioni del blocco.
- Timestamp: indica l'istante in cui il blocco è stato creato; la lista di blocchi è costruita cronologicamente ordinata.
- Bits: rappresenta il corrente valore target.
- Nonce: è un numero casuale o pseudo-casuale, utilizzato per aumentare l'entropia della funzione hash.
- Numero di transazione: identifica il numero della transazione.

Il body contiene invece una lista di transazioni validate, ognuna delle quali è composta a sua volta da diversi campi. Ogni blocco può contenere più transazioni, le quali sono validate al momento della creazione del blocco.

Il tempo medio che una rete impiega per generare un blocco extra nella blockchain è chiamato *block time*. In generale possiamo pensare che la creazione di un blocco corrisponda al momento in cui avviene la transazione, quindi un minore *block time* significa transazioni più veloci. Prendendo come esempio le due piattaforme blockchain più celebri, Bitcoin ed Ethereum, abbiamo che il *block time* per il primo è in media di 10 minuti, mentre per il secondo è tra 10 e 20 secondi.

### Hashing e Merkle Tree

L'hash del blocco corrente è dunque inserito nell'intestazione del blocco successivo della catena e lavora come un puntatore hash. I puntatori, in programmazione, sappiamo essere variabili che memorizzano l'indirizzo di un'altra variabile. Un puntatore hash è simile a un puntatore, ma invece che contenere semplicemente l'indirizzo del blocco precedente, ne contiene anche l'hash dei dati. Per ottenere gli hash, è usata una funzione hash crittografica, cioè una funzione hash H che rispetta alcune proprietà:

- è deterministica
- è rapida da calcolare
- deve essere impossibile risalire ad A dato H(A).

La concatenazione dei blocchi permette alla blockchain di essere robusta di fronte a possibili manomissioni perché modificando i dati memorizzati dentro un blocco, anche il suo digest varia. Questo significa che è sufficiente modificare il contenuto di un singolo blocco per invalidare tutti i successivi blocchi della catena.

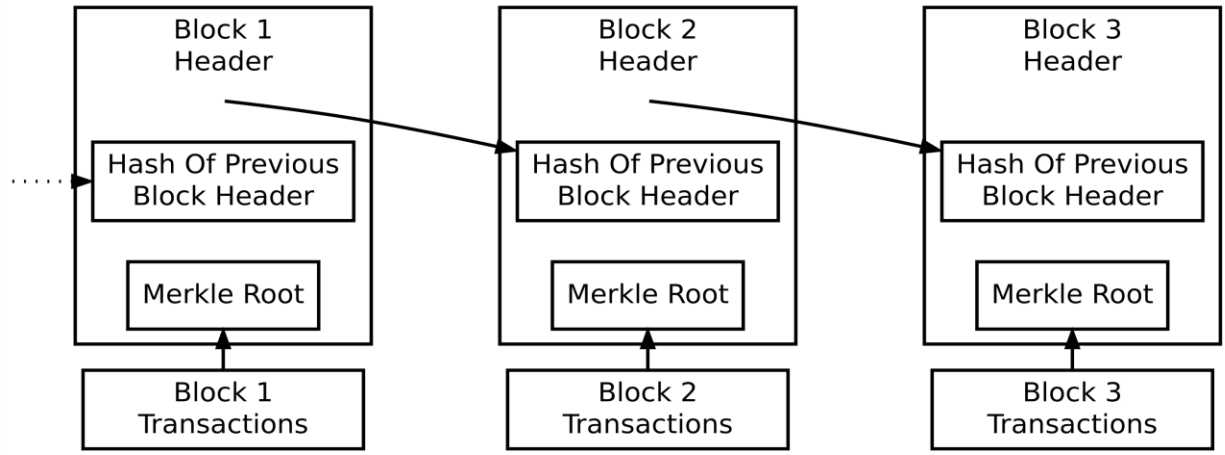



**Figura 3: rappresentazione di una catena di blocchi logicamente collegati dall'hash del blocco precedente**

Nella Bitcoin blockchain è usata una struttura dati chiamata Merkle Tree, che permette di riassumere il contenuto di più transazioni in un singolo hash, il Merkle tree root; questo permette di ridurre la dimensione dei dati da memorizzare. La generazione del Merkel tree consiste nell'applicazione ripetuta della funzione hash. Una volta che gli hash di tutte le transazioni sono stati calcolati, essi vengono raggruppati in coppie, concatenati e usati di nuovo come input della funzione hash. Questo comportamento è ripetuto fino a che rimane un solo valore, appunto il Merkle tree root. La struttura generata è un albero binario in cui ogni nodo padre è semplicemente l'hash della concatenazione dei due figli. In Fig. 4, il nodo etichettato come "Top Hash" è il *Merkle root*.

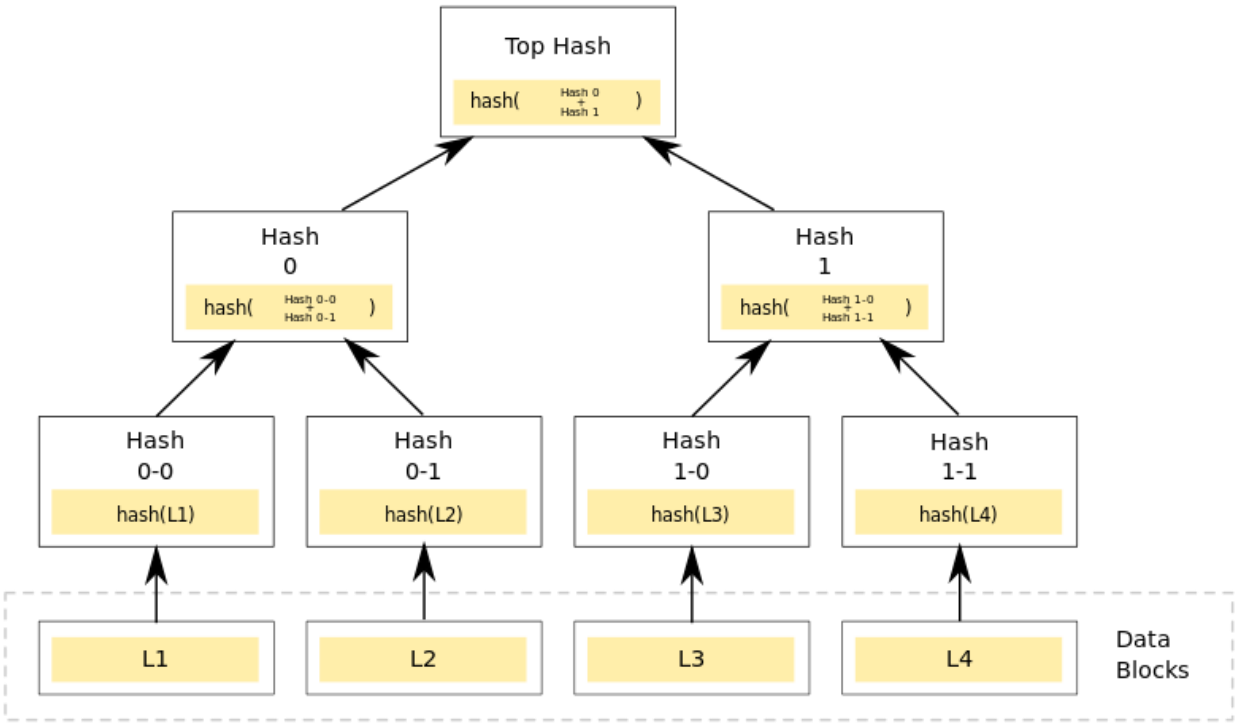



**Figura 4: rappresentazione grafica di un Merkle tree**

### Transazioni

Le transazioni sono combinate in blocchi e rappresentano la più piccola unità di dati che può essere memorizzata in blockchain. Le transazioni non sono standardizzate e il loro contenuto dipende strettamente dalle applicazioni per le quali la blockchain è stata concepita. Molte piattaforme di blockchain considerano una transazione come un semplice trasferimento di proprietà di valuta digitale. In questo caso, i membri della rete possono verificare se una transazione è valida o meno verificando la proprietà dell'asset e l'autorizzazione al trasferimento tramite firma digitale. È riduttivo pensare a una transazione semplicemente come un trasferimento di valuta perché anche applicazioni *general-purpose* possono essere costruite sopra una piattaforma di blockchain. Per questa ragione, le transazioni possono essere considerate come generiche operazioni di lettura/scrittura che innescano cambiamenti di stato in un database distribuito.

### Fork

A volte blocchi indipendenti possono essere prodotti concorrentemente, creando una biforcazione (*fork*) temporanea. Oltre a una storia affidabile basata su hash, ogni

blockchain ha uno specifico algoritmo per valutare differenti versioni della storia così che possa essere selezionata quella con punteggio più alto. I blocchi che non sono selezionati per l'inclusione nella catena sono chiamati blocchi orfani. I peer si possono trovare così ad avere occasionalmente diverse versioni della storia, ma mantengono in memoria solo quella con punteggio più alto. Quando ricevono una versione con punteggio più alto, estendono o sovrascrivono il loro database e ritrasmettono ad altri peer le modifiche. Non c'è mai una garanzia assoluta che una particolare entry rimarrà per sempre nella miglior versione della storia. Tipicamente, le blockchain sono costruite in modo da incentivare l'estensione con nuovi blocchi piuttosto che sovrascrivere i vecchi. Per questa ragione, la possibilità che una entry possa essere sostituita decresce esponenzialmente con l'aggiunta di nuovi blocchi, tendendo allo 0.

In seguito a modifiche importanti al protocollo, generalmente per modifiche a regole di sicurezza, può essere necessario un *hard fork*, cioè un cambio radicale della rete che rende validi blocchi prima non validi o viceversa. Un *hard fork* richiede che tutti i nodi degli utenti si aggiornino all'ultima versione del software. Se un gruppo di nodi continua a usare la vecchia versione del software mentre altri usano quella nuova, può capitare una divisione permanente, come capitato per Ethereum, determinando la nascita di Ethereum ed Ethereum Classic nel 2016[10]. Alternativamente, per evitare ciò, i nodi che usano il nuovo software possono ritornare alle vecchie regole, come è avvenuto per bitcoin in seguito allo split del 12 marzo 2013[11].

### 2.3.2 Blockchain pubbliche e private, permissioned e permissionless.

Una prima suddivisione fra le categorie di possibili blockchain è tra blockchain pubbliche e blockchain private, dipendentemente dal fatto che si possa unire chiunque a una rete oppure che l'accesso sia ristretto. Esse possono inoltre caratterizzarsi dal fatto di essere *permissionless* o *permissioned*, cioè se richiedono o meno il possesso di un permesso per unirsi alla rete. Generalmente le reti pubbliche sono *permissionless*, mentre quelle

[10] https://en.wikipedia.org/wiki/Ethereum_Classic

[11] https://freedom-to-tinker.com/2015/07/28/analyzing-the-2013-bitcoin-fork-centralized-decision-making-saved-the-day/

private sono *permissioned*: infatti il caso d'uso tipico di ognuna di loro è concedere, rispettivamente, l'accesso a tutti i peer interessati (public blockchain) o solo a una ristretta cerchia di peer autorizzati (private blockchain). Ciò non significa che non siano presenti soluzioni di reti pubbliche *permissioned* o private *permissionless*, ma sono di certo una minoranza nel mercato. Nel seguito, spesso saranno usati quindi *permissionless* e public come sinonimi; allo stesso modo sarà fatto con private e *permissioned*.

In una blockchain pubblica, chiunque abbia un accesso a Internet può unirsi alla rete per effettuare e validare transazioni. Generalmente, queste reti prevedono incentivi economici per gli utenti che partecipano al sistema di verifica delle transazioni, tipicamente sotto forma di creazione di asset digitali come *token* o *coins*. Le due più famose piattaforme di blockchain, Bitcoin ed Ethereum, sono pubbliche. Il grande vantaggio di una rete aperta e *permissionless* è che non necessita di controllo dell'accesso: ciò significa che le applicazioni possono essere aggiunte alla rete senza l'approvazione di altri e senza avere bisogno di essere un nodo fidato. Gli algoritmi più utilizzati per il consenso sono il Proof of Stake e il Proof of Work, necessari per rendere sicura la rete. Più utenti sono presenti in rete, maggiore è la sicurezza e il valore della tecnologia; in economia, questo è descritto come "effetto rete"[12].

Le blockchain private, al contrario, non si appoggiano a nodi anonimi e non beneficiano dell'effetto rete. Partecipanti e validatori non possono liberamente unirsi alla rete ma devono essere autorizzati da un amministratore dell'organizzazione proprietaria di essa. Queste reti sacrificano decentralizzazione, anonimato e *openness* in cambio di velocità d'esecuzione, controllo sui partecipanti e riduzione dei costi. L'organizzazione proprietaria della rete inoltre, ha il potere di modificare le regole di funzionamento della blockchain stessa, rifiutando determinate transazioni in base alle regole e alle normative stabilite. La tecnologia attualmente più utilizzata, in ambito blockchain private, è Hyperledger Fabric.

[12] È l'effetto di incremento del valore di un bene dovuto al fatto che esso sia utilizzato da molte persone.

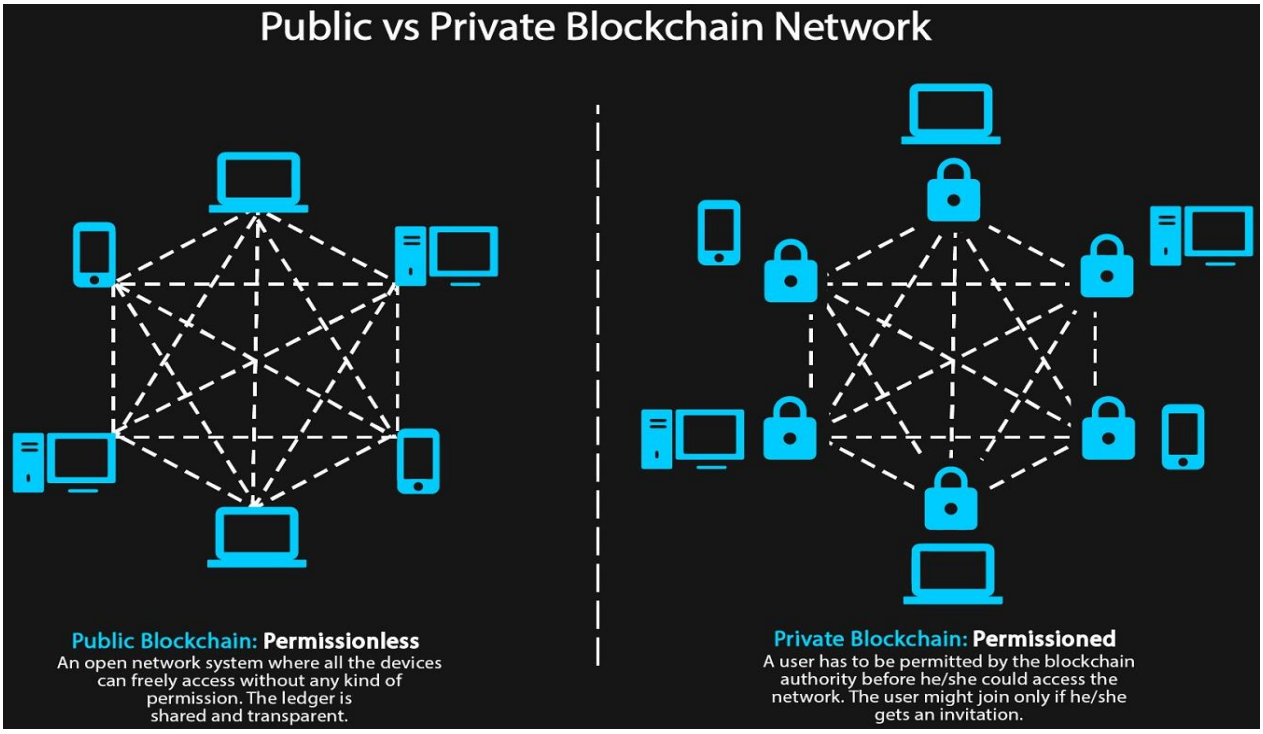



**Figura 5: public vs private blockchain**

### 2.3.3 Algoritmi di consenso

In un sistema distribuito più parti interagiscono tra loro per mantenere uno stato coerente del *ledger*. Molti utenti possono simultaneamente eseguire transazioni o accedere al *ledger* in lettura e scrittura. Il problema del consenso distribuito richiede che tutti i partecipanti siano in accordo su un ordinamento totale degli eventi e mantengano una replica dei dati memorizzati, così che possano verificare la validità delle nuove transazioni prima di aggiungerle al *ledger* e che siano a conoscenza dello stato corrente della blockchain e degli eventi accaduti nel tempo. La completa replicazione dei dati è necessaria negli ambienti di blockchain pubblica perché i partecipanti possono unirsi o lasciare la rete ad ogni momento. Usando una replicazione parziale dei dati, infatti, se ogni nodo che memorizza una replica della stessa porzione del dataset andasse offline, sarebbe impossibile recuperare lo stato finale del *ledger*. Una replicazione parziale sarebbe invece fattibile in ambiente privato, dove l'identità degli utenti è ben conosciuta. La replicazione parziale è comunque una soluzione adottata in pochissime piattaforme blockchain (per esempio BigchainDB); la maggior parte delle piattaforme preferisce una

replicazione totale. È importante che in un sistema distribuito siano gestiti comportamenti errati e crash accidentali dei nodi; questi in particolare possono subire attacchi informatici o avere problemi di comunicazione che compromettono la corretta interazione tra i peer.

### Consenso nelle blockchain pubbliche

Le piattaforme di blockchain pubbliche richiedono l'uso di algoritmi di consenso che non necessitano dell'identificazione degli utenti; i più diffusi sono Proof of Work (PoW) e Proof of Stake *(*PoS*)*. Bitcoin ed Ethereum utilizzano PoW.

Un *proof of work* può essere visto come un pezzo di dato difficile da produrre ma facile da verificare, come per esempio un processo casuale con bassa probabilità, così che siano necessari molti tentativi in media prima che sia generato un valido *proof of work*. Affinché un blocco sia accettato, i *miners* della rete devono completare un *proof of work* che copre tutti i dati del blocco. La difficoltà di questo lavoro regola la velocità a cui i blocchi sono creati. Ad esempio, l'algoritmo utilizzato da bitcoin per creare blocchi si chiama Hashcash, un sistema sviluppato inizialmente per limitare attacchi di *denial-of-service* e spam: lo schema utilizzato è basato su SHA-256, funzioni hash crittografiche utilizzate per verificare l'integrità di un dato confrontando l'hash previsto con quello calcolato. Il primo *miner* che riesce ad arrivare alla soluzione ottiene un premio in bitcoin, dunque maggiore è la potenza computazionale che si presta al sistema, maggiore è la probabilità di essere il vincitore del premio. La principale criticità di questo sistema è l'enorme consumo energetico che richiede la validazione di un blocco, che diventerà insostenibile in breve tempo con i ritmi di crescita attuali.

Proof of Stake è un algoritmo che si basa su un concetto diverso, in quanto non premia la potenza computazionale, ma bensì il possesso del maggior numero di "capitale in gioco". In un sistema PoS non ci sono *miner*, bensì validatori che si alternano per votare e proporre il prossimo blocco; il voto di ognuno di essi avrà un peso in accordo con la quantità di credito posseduto. Le principali critiche rivolte a questo sistema si basano sul fatto che c'è un maggiore rischio di manomissione rispetto a Proof of Work: in PoW, un *miner* malevolo che vuole appropriarsi indebitamente di denaro che non gli spetta, avrà bisogno di spende più di quanto riuscirà a rubare per riuscire a manomettere la rete,

mentre in PoS il validatore non deve prestare potenza computazionale e per questo sono necessari meccanismi diversi per impedire comportamenti malevoli. Il fondatore di Ethereum, Vitalik Buterin, ha annunciato nel 2017 di voler effettuare in futuro un *fork* da PoW verso PoS. Il rilascio di Ethereum 2.0, che implementa il protocollo Casper per PoS, era stata prevista per gennaio 2020, ma al momento è stata posticipata (indiscrezioni della community parlano di luglio 2020). Il protocollo prevede che ogni validatore piazzi una "scommessa" su quello che pensa essere il prossimo blocco e, in caso esso sia validato, riceverà un premio proporzionale alla scommessa. In caso il validatore provi ad agire in modo malevolo, sarà punito con la perdita di parte del credito posseduto.

Esistono molti altri algoritmi di consenso, come Proof of Authority (PoA) e Proof of Space (PoSpace). Proof of Authority permette di avere transazioni veloci basate su un meccanismo di validazione in base all'identità. I partecipanti di una rete possono avere il diritto di essere validatori, in un sistema basato sulla reputazione legata all'identità: per conservare la loro posizione di validatori, gli utenti devono mantenere una certa reputazione e ciò li scoraggia da comportamenti malevoli. PoSpace è molto simile al PoW, con la differenza che invece di potenza computazionale, l'utente mette a disposizione un certo ammontare di memoria o spazio su disco per risolvere una sfida. Grazie al minore consumo di energia, dovuto all'utilizzo della memoria invece che della potenza computazionale, è un'alternativa più "green" al PoW.

### Consenso nelle blockchain private

In ambiente privato, dove gli utenti possono essere identificati, ci si può slegare dai meccanismi di consenso prima introdotti, in favore di altri più efficienti. In questo modello i blocchi sono validati solo da alcune entità validatrici, nelle quali gli utenti ripongono fiducia. Il mining viene quindi eliminato, introducendo una o più autorità centrali che si occupano della creazione/validazione dei blocchi.

In questo sistema è importante implementare meccanismi che permettano di gestire il consenso in presenza di nodi crashati o difettosi. Ci sono due principali gruppi di algoritmi che sono stati sviluppati per raggiungere tale obiettivo:

- Algoritmi *crash fault-tolerant*. Gestiscono nodi crashati o irraggiungibili. Per raggiungere il consenso deve esser rispettata la disuguaglianza $n >= 2f + 1$, con *n* numero totale dei nodi nella rete e *f* numero dei nodi crashati. Un esempio di algoritmo di consenso *crash fault-tolerant* è Raft, utilizzabile in Hyperledger Fabric.
- Algoritmi *Byzantine fault-tolerant*. Gestiscono nodi crashati, irraggiungibili e con comportamenti malevoli. Per raggiungere il consenso in un ambiente privato, deve essere rispettata la disequazione $n >= 3f + 1$, con n numero totale dei nodi e *f* il numero dei nodi *faulty*[13]. Un esempio è BFT-SMaRT.

### 2.3.4 Smart contract

In seguito all'avvento della blockchain 2.0, "*smart contract*" è diventata sicuramente una delle parole d'ordine: uno *smart contract* non è altro che un codice che può essere parzialmente o interamente eseguito senza l'intervento umano, tramite la garanzia automatica che le condizioni pattuite nel contratto siano effettivamente verificate. La blockchain rappresenta una forma di memorizzazione di dati sicura e certificata, che fornisce la sicurezza e la tracciabilità necessaria per contratti che abbiano un valore legale, aprendo quindi a scenari di contratti automatici ad esempio per pagamenti, rimborsi o snellimento di iter burocratici. Al momento della stipulazione del contratto, le due parti devono mettere per iscritto ogni requisito in modo chiaro, così che tutto possa essere tradotto poi in codice per implementare perfettamente le condizioni dell'accordo. Uno *smart contract* quindi non è altro che un programma che utilizza la blockchain per la tracciabilità e la memorizzazione dei dati e al verificarsi di determinate condizioni adotta il comportamento deciso nel contratto. Per fare *ingestion* di dati e controllare particolari condizioni esterne, uno *smart contract* ha bisogno di un componente chiamato oracolo, il quale fa da tramite tra la blockchain e il mondo esterno.

È importante in tutto ciò il tema della fiducia: *lo smart contract* deve fornire una serie di garanzie a tutte le parti coinvolte e primariamente deve garantire che il codice con cui è

[13] Un nodo *faulty* è un nodo "intenzionalmente difettoso" che si comporta in modo diverso in base all'interlocutore.

stato scritto non possa essere modificato, che le fonti di dati che determinano le condizioni di applicazione siano certificate e affidabili, che le modalità di lettura e controllo di queste fonti siano a loro volta certificate. La sicurezza e la validità legale è data dalla possibilità di tracciare ogni blocco, ogni transazione e dunque ogni dato che sia mai esistito in blockchain.

Per esempio, possiamo pensare a una compagnia aerea che vuole implementare un sistema di rimborso automatico del biglietto per i ritardi superiori a una certa soglia: un programma terrà traccia in blockchain di tutti gli aerei, con relativi orari di arrivo e partenza, e dei passeggeri che si sono registrati al servizio. All'arrivo e alla partenza di ogni aereo un oracolo farà *ingestion* dei dati in blockchain, quindi il contratto controllerà gli orari previsti e in modo automatico si occuperà di garantire rimborsi agli utenti registrati.

Inizialmente il termine *smart contract* faceva riferimento solo a veri e propri contratti digitali, la cui attuazione automatica non era legata obbligatoriamente alla blockchain, garantiti tramite firme (digitali o meno) delle parti coinvolte. Con il tempo si è poi molto allargato il significato del termine, arrivando a indicare genericamente un'applicazione distribuita basata su blockchain. In Hyperledger Fabric il termine *smart contract* indica i *chaincode*, cioè i programmi utilizzati per effettuare operazioni sui *ledger*. Anche in Ethereum un generico programma che utilizza la blockchain viene definito *smart contract* e in particolare l'esecuzione di tali contratti è fortemente legata all'utilizzo di Ether, i token di Ethereum.

## 2.4 Quando usare una blockchain?

A questo punto della trattazione, si può analizzare in quali casi può essere utile l'utilizzo della blockchain e quando invece non è raccomandata. Essa può essere la soluzione giusta quando abbiamo necessità di:

- Log *append-only* per le transazioni: è possibile solo aggiungere nuovi dati in fondo alla catena;
- Integrità dei dati: il contenuto dei blocchi non può essere modificato;

- Creazione di applicazioni distribuite: ogni nodo contiene il codice dell'applicazione e lo stato del *ledger*;
- In particolare, è raccomandata una blockchain pubblica quando si vuole trasparenza, in quanto ogni utente ha visione di tutti i dati;
- Una blockchain privata è invece necessaria quando si vuole restringere l'accesso ai dati a sole alcuni attori autorizzati.

Al contrario, l'uso della blockchain non è consigliato nel caso serva:

- Alto throughput in termini di transazioni eseguite per secondo. Le piattaforme private hanno un throughput più alto di quelle pubbliche, ma comunque minore di quello dei database tradizionali.
- Bassa latenza transazionale. Questa è legata al fatto che ogni transazione debba essere verificata e inserita in un nuovo blocco, dunque il blocco deve essere validato e distribuito agli altri partecipanti della rete.
- Parziale replicazione dei dati. La maggior parte delle piattaforme richiede che ogni utente abbia lo spazio necessario per la replicazione completa, dunque ciò a lungo andare potrebbe essere insostenibile per alcuni. Sono poche le piattaforme che permettono una replicazione parziale.

## 2.5 Stato dell'arte

Spesso nella trattazione ho fatto riferimento a Bitcoin, in quanto rappresenta la prima blockchain e quindi il punto di partenza di tutte le altre. Dal white paper di Nakamoto si sono poi sviluppate tutte le altre blockchain, utilizzate sia nell'ambito delle criptovalute che in altri ambiti. Un'altra tecnologia a cui ho fatto riferimento è Ethereum, seconda solo a Bitcoin in quanto a importanza sul mercato. Nei prossimi sotto paragrafi andrò ad introdurre alcune delle tecnologie più interessanti attualmente disponibili, descrivendo le loro principali caratteristiche.

Ad Hyperledger Fabric sarà dedicato per intero il terzo capitolo, in modo da dare quanti più dettagli possibili prima della parte progettuale.

### 2.5.1 Bitcoin

Bitcoin è la tecnologia utilizzata per scambiare criptovalute bitcoin su una rete di blockchain pubblica. Utilizza un database distribuito su più peer per tenere traccia delle transazioni, sfruttando la crittografia per gestire gli aspetti funzionali, come la generazione di nuova moneta e l'attribuzione della proprietà dei bitcoin. Alcuni aspetti tecnologici sono già stati discussi in precedenza: consenso, blocchi, tipologia di rete, ecc., ma è comunque importante approfondire ulteriormente gli aspetti non ancora introdotti di questa importante tecnologia.

La rete Bitcoin consente il possesso e il trasferimento anonimo di monete virtuali, le quali possono essere salvate su più dispositivi elettronici sotto forma di "portafoglio" digitale o mantenuti presso terze parti che svolgono funzioni simili a una banca. La struttura peer-to-peer della rete Bitcoin e la mancanza di un ente centrale rende impossibile a qualunque autorità, governativa o meno, il blocco dei trasferimenti, il sequestro di bitcoin senza il possesso delle relative chiavi o la svalutazione dovuta all'immissione di nuova moneta. Il client ufficiale, Bitcoin Core, deriva direttamente dal codice scritto da Satoshi Nakamoto per implementare il protocollo di comunicazione e la rete peer-to-peer che ne risulta. In virtù della caratteristica open source del software, nel tempo sono stati creati anche altri programmi che implementano il protocollo Bitcoin in modo indipendente.

Bitcoin utilizza uno schema di crittografia a chiave pubblica per implementare lo scambio di valuta. Ogni utente che partecipa alla rete possiede un portafoglio contenente un numero arbitrario di coppie di chiavi crittografiche, dove la chiave privata serve ad apporre una firma digitale a ogni transazione facendo in modo che sia autorizzato al pagamento solo l'utente proprietario di quella moneta. La rete verifica poi la firma utilizzando la chiave pubblica. L'algoritmo utilizzato da Bitcoin per generare le chiavi è l'Elliptic Curve Digital Signature Algorithm. Se la chiave privata viene smarrita, la rete Bitcoin non potrà riconoscere in alcun altro modo la proprietà del denaro: il caso più eclatante di perdita patrimoniale risale al 2013, quando un utente si sbarazzò per sbaglio di un hard disk contenente la sua chiave privata, perdendo conseguentemente 7500 bitcoin per un valore di circa 7,5 milioni di dollari.

Per impedire la possibilità di utilizzare più volte la stessa moneta, la rete implementa un sistema di *timestamping* che assegna identificatori sequenziali a ognuna delle transazioni, le quali sono poi sottoposte al protocollo di consenso *proof-of-work*. Ogni volta che viene effettuata una transazione, essa parte dallo stato di "non confermata" e diventerà poi "confermata" solo quando sarà verificata attraverso il sistema di marcatura temporale e consenso della blockchain.

L'attività di generazione di bitcoin viene spesso definita come "mining", cioè “estrazione”, per dare l’idea del valore che si vuole estrarre dalla blockchain. Inizialmente il client stesso si occupava di svolgere i calcoli necessari all'estrazione dei bitcoin, sfruttando la sola CPU. Oggigiorno esistono programmi specializzati che utilizzano hardware dedicato basato su processori progettati per questo utilizzo. Data l’enorme mole di potenza computazionale richiesta, spesso i *miner* si uniscono in mining pool dove i partecipanti mettono in comune le proprie risorse, spartendosi poi i blocchi generati in funzione del contributo di ognuno. Il numero di bitcoin creati per blocco era inizialmente 50, tale quantità è stata programmata per diminuire nel tempo secondo una progressione geometrica con un dimezzamento del premio ogni 4 anni circa. Così dimensionata, questa serie comporta che in totale verranno creati circa 21 milioni di bitcoin nel giro di 130 anni circa. Dal punto di vista tecnico il processo di mining non è altro che un'operazione di *hashing* inverso: determinare un numero tale per cui l'hash SHA-256 di un insieme di dati rappresentante il blocco sia inferiore a una soglia data. Questa soglia, chiamata difficoltà, è ciò che determina la natura concorrenziale del mining di bitcoin: più potenza di calcolo viene aggiunta alla rete bitcoin e più questo parametro aumenta, aumentando di conseguenza il numero di calcoli mediamente necessari.

### 2.5.2 Ethereum

Ethereum è una piattaforma open source di calcolo distribuito basata su blockchain pubblica, il cui principale obiettivo è quello di fornire ai propri utenti la possibilità di sviluppare *smart contract*. Ethereum è stato proposto nel tardo 2013 da Vitalik Buterin, un programmatore russo appassionato di criptovalute. Lo sviluppo è partito in seguito a una raccolta di fondi online nell’estate del 2014 ed è stato completato nel luglio 2015.

Ethereum fornisce una macchina virtuale, l'Ethereum Virtual Machine (EVM), che può eseguire script usando una rete pubblica di nodi distribuiti. Il set d'istruzioni della macchina virtuale è Turing completo, cioè dà la possibilità di eseguire ogni tipo di operazione, incluse le istruzioni che modificano l'esecuzione del flusso di informazioni (come i cicli). Bitcoin al contrario permette solo l'esecuzione di script molto più semplici. La possibilità di ripetere porzioni di codice espone il sistema al problema dei cicli infiniti, che potrebbero sovraccaricare la rete rendendo impossibili altre transazioni. Per superare questo problema e mitigare la spam, viene usato un meccanismo interno di prezzatura per le transazioni, chiamato Gas. L'Ether è la criptovaluta generata dalla piattaforma come premio per la computazione effettuata dai nodi *miner* ed è l'unica valuta accettata come pagamento per le transazioni. Come già introdotto precedentemente, il protocollo di consenso utilizzato è PoW, ma si sta discutendo concretamente sulla possibilità di spostarsi verso il PoS.

Uno *smart contract* è definito come un pezzo di software che è eseguito in blockchain e che può contenere istruzioni arbitrarie. Ogni transazione di Ethereum contiene due campi in più rispetto a Bitcoin: STARTGAS rappresenta quante istruzioni saranno potenzialmente eseguite dal codice innescato dalla transazione stessa, GASPRICE indica il prezzo che deve essere pagato per un'istruzione. Il proposito principale di Ethereum è la creazione di Decentralized Autonomous Organizations (DAO), controllate dall'esecuzione di *smart contract*. Gli utenti possono interagire per eseguire compiti complicati senza l'intervento di un'autorità centrale, in un'organizzazione in cui le decisioni sono prese dalla maggioranza dei membri. Una possibile implementazione di DAO può essere lo scambio di energia elettrica eseguito direttamente tra utenti che la producono (per esempio con pannelli fotovoltaici) e altri interessati ad acquistarla, il tutto senza il bisogno dell'intermediazione di compagnie elettriche. Legato a *smart contract* e DAO è il concetto di Dapp, cioè *distributed application*: un'applicazione distribuita ha il suo codice di back-end ospitato ed eseguito su una rete peer-to-peer, rendendo l'applicazione più robusta, flessibile, trasparente e resiliente. Esempi di Dapp, ma non basate su blockchain, sono bitTorrent[14] e Tor[15].

[14] www.bittorrent.com

Altro concetto fondamentale in Ethereum è quello di stato: in ogni momento, la blockchain è caratterizzata da uno stato corrente, dato da tutto l'insieme di transazioni avvenute precedentemente. La visione dello stato attuale della rete permette di conoscere la quantità di Ether posseduti dagli utenti, le informazioni memorizzate e i pezzi di software che possono essere fatti partire con l'esecuzione di transazioni. Tutte queste informazioni sono associate ai partecipanti e raccolte in account. Una caratteristica interessante degli account consiste nella possibilità di scambiarsi messaggi. In questo modo è implementato un meccanismo di intercomunicazione tra account, così da ottenere anche comportamenti complessi.

### 2.5.3 Quorum

Quorum[16] è una versione *permissioned* di Ethereum costruita per applicazioni *enterprise*. Sviluppato da J.P. Morgan, Quorum è una *fork* 100% open source basata su Go Ethereum. È l'ideale per qualsiasi applicazione che richiede alto throughput e alta velocità di processamento di transazioni private all'interno di un gruppo *permissioned* di partecipanti noti. Il sistema di permessi può essere definito in modo flessibile, decidendo quali entità e utenti inserire nella rete usando il modello di *smart contract permissioned* di Quorum. Il fatto di avere una version *permissioned* di Ethereum è utile alle aziende per gestire finanze, supply chain, vendita al dettaglio, immobili, ecc. Particolare attenzione è posta per una forte privacy: le informazioni private non sono mai mandate ai partecipanti della rete; i dati privati sono criptati e condivisi solo direttamente con le parti interessate. Per dettagliare meglio la struttura della propria organizzazione è possibile mappare sotto-entità a entità genitore.

In base al caso d'uso specifico può essere scelto l'algoritmo di consenso che meglio risponde alle necessità, tra: Raft, IBFT o Clique PoA.

Quorum supporta due tipi di stato:

- Stato pubblico, accessibile a tutti i nodi della rete;
- Stato privato, accessibile solo dai nodi con i permessi corretti.

---

[15] www.torproject.org

[16] https://www.goquorum.com/

La differenziazione avviene tramite l'uso di payload criptato o meno nelle transazioni. I nodi possono determinare tramite un valore della firma se la transazione è privata o no. Se la transazione è privata, il nodo può eseguire la transazione solo se ha l'abilità di accedere e decriptare il payload. Le transazioni private contribuiscono a creare uno stato private unico di ogni nodo, mentre quelle pubbliche compongono uno stato pubblico condiviso da tutti i nodi della rete.

### 2.5.4 Corda

Corda è una piattaforma blockchain *permissioned* open source scritta in Kotlin, il cui codice è disponibile su GitHub[17] sotto licenza Apache 2. È stata pensata fin dall'inizio per il business: permette di eseguire transazioni dirette con forte privacy usando *smart contract*, riduce i costi di transazioni e memorizzazione dei record e razionalizza le operazioni di business. Le applicazioni sviluppate su questa piattaforma prendono il nome di CorDapps e possono essere sviluppate in Java e altri linguaggi JVM. Il suo design è scalabile per incontrare le necessità di ogni attività ed è supportato da una grande comunità di sviluppatori che lavorano continuamente per aggiungere nuove feature e funzionalità.

Una rete Corda è fatta da nodi, ognuno dei quali ospita un'istanza di Corda e una o più CorDapps, in un tipico design peer-to-peer. La comunicazione è *point-to-point* e ogni nodo ha la possibilità potenziale di comunicare con gli altri. Ogni nodo può accedere alla rete solo previa autorizzazione da parte di un operatore di rete, il quale firma un certificato che mappa l'identità di rete del nodo a un'identità del mondo reale. La comunicazione è resa possibile tramite un sistema di *discovery* basato su un servizio di mapping di rete: una lista di peer, simile a un elenco telefonico, con i propri metadati utili all'identificazione e alla scoperta di servizi.

Il *ledger* è mantenuto consistente tra i nodi in modo che ad ogni transazione i peer abbiano la garanzia di vedere la stessa versione del *ledger*, benché con "prospettive diverse": la base di dati distribuita tra i nodi è univoca, ma ognuno ne ha una visione ristretta e non c'è un'entità o peer che ne ha la visione totale. Ogni nodo conosce i fatti

[17] https://github.com/corda/corda

in cui è coinvolto, mentre non avrà mai visibilità di quelli che non lo coinvolgono. Gli stati rappresentano un fatto noto ad un certo momento sul *ledger*, non possono quindi essere modificati per riflettere un cambio nello stato del mondo. Il ciclo di vita di un fatto condiviso è caratterizzato da una sequenza di stati: quando un fatto ha bisogno di essere aggiornato, creiamo una nuova versione di esso e marchiamo lo stato preesistente come "storico".

Le transazioni sono operazioni atomiche di aggiornamento del *ledger*, rappresentano i collegamenti tra le varie sequenze di stato. Una proposta di transazione è *committed* solo se:

- Non contiene doppie spese;
- È contrattualmente valida;
- È firmata dalle parti richieste.

Una transazione è contrattualmente valida se, oltre a presentare tutte le firme necessarie, tutti i suoi stati in input e output sono accettabili in accordo al contratto specificato. La verifica di una transazione deve essere deterministica: un contratto dovrebbe sempre accettare o rifiutare una data transazione. Per verificare il determinismo di un contratto, è possibile utilizzare moduli appositi. Dato che un contratto non ha accesso alle informazioni del mondo esterno, non può sapere se una transazione è conforme a quanto inizialmente accordato tra le parti. Per questo i peer dovrebbero sempre verificare i contenuti di una transazione prima di firmarla, anche se la transazione è contrattualmente valida, per vedere se è in accordo con l'aggiornamento proposto del *ledger*. A volte, la validità di una transazione dipenderà da alcune informazioni esterne, come un tasso di cambio. In questi casi è richiesta la presenza di un oracolo che si occupi di passare tali informazioni al contratto.

### 2.5.5 BigchainDB

BigchainDB[18] è un progetto open source di database distribuito basato su blockchain. È pensato sia per essere integrato in *stack* di architetture già esistenti, sostituendo di fatto

[18] https://www.bigchaindb.com/

il database relazionale, sia per la progettazione di uno *stack* distribuito pensato apposta per la blockchain.

I database relazionali organizzano i dati in tabelle, i database NoSQL li organizzano in JSON o coppie chiave-valore, mentre BigchainDB struttura i dati in asset. Un asset può rappresentare una qualsiasi risorsa digitale o fisica, come l'ordine di un cliente o un'auto. Oltre alle informazioni descrittive immutabili, è possibile specificare anche metadati che possono essere aggiornati, come per esempio lo stato di temperatura e umidità di un sensore. Un asset può avere uno o più proprietari, ma può anche essere sé stesso il proprietario (si pensi a un'auto a guida autonoma o un sensore IoT che effettua transazioni da solo), inoltre può rappresentare la prova di una qualsiasi proprietà di un particolare oggetto, per esempio rappresenta la prova che l'utente ABC possiede la bicicletta con codice XYZ.

Tradizionalmente, le applicazioni sono progettate focalizzandosi sui processi di business, BigchainDB invece pone al centro gli asset. Questo cambio di prospettiva da una visione *process-centric* ad una *asset-centric,* influenza fortemente lo sviluppo delle applicazioni.

Gli asset possono essere registrati su blockchain in due modi:

- Dagli utenti in transazioni CREATE, che rappresentano l'inserimento di una nuova risorsa.
- Trasferiti ad altri utenti in transazioni TRANSFER.

Una transazione contiene un *transaction ID,* cioè un hash di tutte le informazioni contenute nella stessa, che viene usato per creare riferimenti alla transazione in transazioni future.

Ogni transazione è caratterizzata da un input e un output. Un input è il puntatore all'output di una precedente transazione; specifica a chi apparteneva un asset in precedenza e fornisce la prova che le condizioni per l'operazione di trasferimento sono soddisfatte. In una transazione CREATE non c'è un proprietario precedente, quindi l'input qui rappresenta semplicemente chi sta registrando l'oggetto. L'output di una transazione specifica le condizioni che è necessario soddisfare per cambiare la proprietà di uno specifico asset. Per esempio, per trasferire una bicicletta, una persona

ha bisogno di firmare la transazione con la sua chiave privata. Ciò contiene implicitamente anche l'informazione che la chiave pubblica associata con quella chiave privata rappresenti il proprietario dell'asset.

Ad una transazione possono essere aggiunti ulteriori metadati per rappresentare ogni tipo di dato, come l'età di una bicicletta o i chilometri percorsi. I metadati possono essere aggiornati ad ogni transazione, in contrasto con quelli che sono i dati inseriti nei campi dell'asset. È inoltre possibile effettuare query sui metadati.

Capitolo 3

# Hyperledger Fabric

Hyperledger Fabric è un'innovativa piattaforma di blockchain privata *permissioned* sviluppata usando il linguaggio di programmazione Go. Diversamente da altre piattaforme blockchain, non è basata su una criptovaluta nativa e permette l'esecuzione di *chaincode*, gli *smart contract* nel gergo di Hyperledger, senza alcun costo. La caratteristica fondamentale di Hyperledger Fabric è la modularità: alcune funzionalità, come il meccanismo di consenso o l'identificazione degli utenti, non sono standardizzate e possono essere sostituite dagli amministratori della blockchain in base alle necessità delle loro applicazioni.

Nella sezione 3.1 andrò ad introdurre le caratteristiche chiave della piattaforma. La sezione 3.2 introdurrà poi i principali componenti dell'architettura.

## *3.1 Introduzione*

In seguito alla diffusione e al successo di tecnologie come Bitcoin, Ethereum ed altre derivate, è cresciuto rapidamente l'interesse industriale verso la blockchain. Si è cominciato così a pensare di allargare l'ambito di applicazione di questa tecnologia anche a casi d'uso specifici *enterprise* più innovativi, come assicurazioni, sanità, risorse umane, supply chain e tanti altri. Per un uso *entreprise* abbiamo necessità di:

- reti *permissioned* in cui gli utenti possano essere identificati
- alto throughput e bassa latenza per le transazioni
- privacy e confidenzialità dei dati privati scambiati tra le aziende.

Hyperledger Fabric è stato pensato fin dall'inizio con lo scopo di supportare applicazioni *entreprise*. Ha un'architettura modulare e configurabile che permette innovazione, versatilità e ottimizzazione per un ampio range di casi d'uso. È supportato dalla Linux Foundation, la quale ha una lunga storia di successo nel sostenere progetti open source che crescono grazie al contributo di nutrite communities.

Uno dei punti fondamentali di Fabric è il supporto per protocolli di consenso configurabile che permettono alla piattaforma di essere efficientemente personalizzata per soddisfare al meglio le necessità e il modello di fiducia. I protocolli di consenso di Fabric non richiedono una criptovaluta nativa per incentivare i *miner* o per permettere l'esecuzione di *smart contract,* ma sono completamente liberi da costi.

Essendo una piattaforma *permissioned*, i partecipanti sono noti. Questo significa che essi possono anche non fidarsi completamente l'uno dell'altro (per esempio essere competitor), ma il modello funziona lo stesso in quanto la governance si basa sul presupposto che esista un accordo tra le parti, un contratto ad esempio, che regoli le possibili dispute che potrebbero nascere. Hyperledger Fabric fornisce confidenzialità principalmente grazie alla sua architettura basata su canali e al supporto per i dati privati.

La combinazione di queste caratteristiche rende Fabric una delle piattaforme più performanti disponibili oggi sia in termini di processamento che di latenza delle transazioni.

## *3.2 Modularità*

Ad alto livello, Fabric è composto dai seguenti componenti modulari:

- Un servizio di ordinamento che stabilisce il consenso sull'ordine delle transazioni e quindi fa broadcast dei blocchi verso i peer;
- Un membership service provider responsabile dell'associazione delle entità nella rete con identità crittografiche;
- Un servizio di gossip peer-to-peer che dissemina ai vari peer i blocchi in output dal servizio di ordinamento;
- *Smart contract*, eseguiti dentro un ambiente containerizzato per maggiore isolamento;
- Un *ledger* che può essere configurato per supportare differenti DBMS;
- Una policy di approvazione e validazione che può essere indipendentemente configurata in base all'applicazione.

Questi punti saranno poi affrontati in dettaglio nel proseguimento della trattazione.

## 3.3 L'approccio execute-order-validate

La maggior parte delle piattaforme blockchain che supportano *smart contract* seguono un'architettura *order-execute* nella quale il protocollo di consenso valida e ordina le transazioni, poi le propaga a tutti i nodi, infine i peer eseguono le transazioni sequenzialmente. L'architettura *order-execute* può essere ritrovata in quasi tutti i sistemi blockchain esistenti, dalle piattaforme pubbliche/*permissionless* quali Ethereum a piattaforme private/*permissioned* come Corda.

Fabric introduce una nuova architettura per le transazioni chiamata *execute-order-validate*, la quale si prefigge come obiettivi: resilienza, flessibilità, scalabilità, prestazioni e confidenzialità. Il flusso delle transazioni è diviso in tre passi:

- Esecuzione di una transazione e controllo della sua correttezza.
- Ordinamento delle transazioni tramite un protocollo di consenso.
- Validazione delle transazioni, in accordo a una specifica politica di approvazione, prima di farne il *commit* nel *ledger*.

In Fabric, ogni transazione necessita di essere eseguita solo dal sottoinsieme di nodi necessari per soddisfare la propria specifica politica di approvazione. Questo permette l'esecuzione di transazioni in parallelo, migliorando le prestazioni e la scalabilità del sistema. La prima fase elimina anche il non determinismo, in quanto risultati inconsistenti possono essere filtrati prima di essere ordinati.

## 3.4 Com'è fatta una rete Fabric?

Hyperledger Fabric permette di creare reti blockchain private *permissioned* alle quali gli utenti possono accedere solo dopo autorizzazione da parte di un Membership Service Provider noto. Ruoli chiave sono svolti dall'*orderer* e dai canali: tramite i canali si può accedere ai *chaincode*, i quali permettono di effettuare operazioni sul *ledger* associato a ogni canale; ogni transazione viene approvata dai peer che ospitano i *chaincode* e poi inviata all'*orderer* per essere validata.

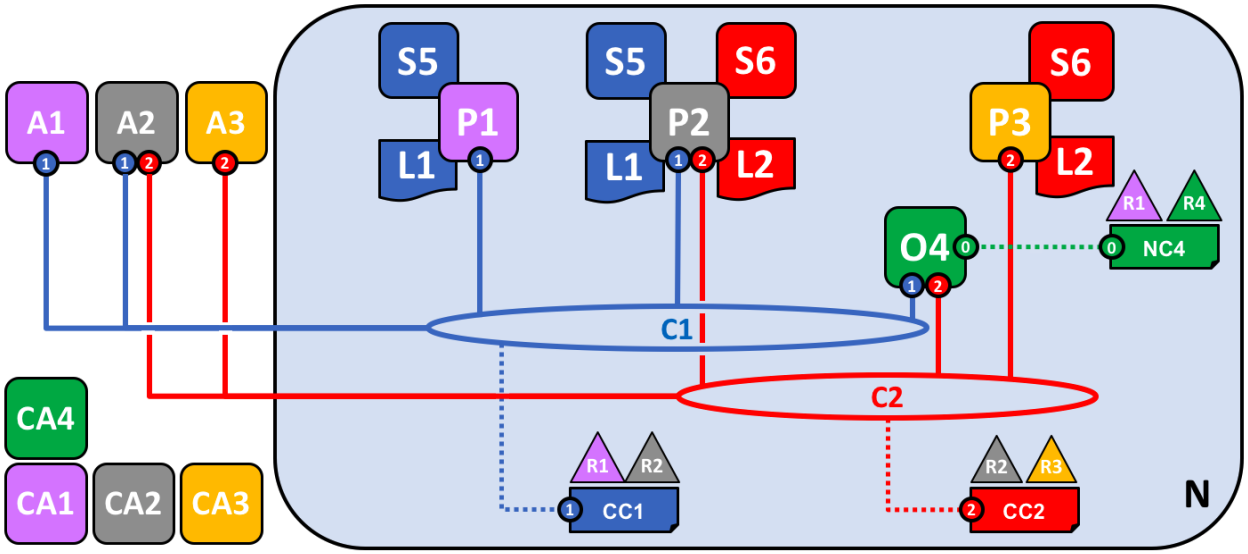



**Figura 6: rete sample di Hyperledger Fabric**

In Fig. 6, le organizzazioni R1, R2, R3 e R4 hanno deciso congiuntamente di mettere per iscritto un accordo per organizzare e utilizzare una rete N sviluppata con Hyperledger Fabric. R4 è l'iniziatore della rete (cioè colui che ha costruito la versione iniziale della rete) e non ha intenzione di effettuare transazioni di business su di essa; non avendo dunque "interessi" sulla rete, è l'organizzazione perfetta per gestire l'*orderer*. Ognuna delle organizzazioni ha una Certificate Authority preferita (CA1, CA2, CA3, CA4).

R1 e R2 hanno bisogno di comunicazioni private all'interno della rete, come pure R2 e R3, le quali sono rese possibili grazie alla creazione dei canali C1 e C2. L'organizzazione R2 ha un'applicazione client che può lavorare su C1 e C2, mentre l'organizzazione R3 ha un'applicazione che invece lavora solo su C2. Il nodo peer P1 mantiene una copia del *ledger* L1 associato a C1 e ospita il *chaincode* S5 per le invocazioni su C1. Il nodo peer P2 mantiene una copia del *ledger* L1 e una copia del *ledger* L2 associata a C2, oltre ai *chaincode* S5 e S6 per C1 e C2. Il nodo peer P3 mantiene una copia del *ledger* L2 e ospita il *chaincode* S6. La rete è governata in accordo a policy specificate nella configurazione di rete NC4, la quale specifica che il controllo sulla rete è nelle mani delle organizzazioni R1 e R4. Il canale C1 è governato in accordo alle policy specificate nella configurazione di canale CC1 ed è sotto il

controllo delle organizzazioni R1 e R2. Il canale C2 è governato in accordo alle policy specificate nella configurazione di canale CC2 ed è sotto il controllo delle organizzazioni R2 e R3. Il servizio di ordinamento O4 funziona da punto di amministrazione per N e utilizza il canale di sistema per mantenere le impostazioni di rete. Il servizio di ordinamento supporta anche i canali applicativi C1 e C2 al fine di ordinare le transazioni in blocchi da distribuire ai peer.

Sul sito di Hyperledger è possibile accedere a molti progetti sample per testare reti già definite e pronte per la sperimentazione.

### 3.4.1 I canali

Fabric offre la possibilità di creare canali, che permettono a un gruppo di partecipanti della rete di comunicare e condividere informazioni in modo esclusivo con i soli iscritti allo stesso canale. Un canale è creato da un'organizzazione e può avere più proprietari, cioè organizzazioni che possiedono sullo stesso canale gli stessi diritti. L'insieme delle organizzazioni che possono creare canali è detto consorzio. A un canale possono essere collegati un numero arbitrario di componenti, quali applicazioni client oppure nodi peer. La presenza di canali permette alle organizzazioni di formare una cosiddetta "rete di reti": i canali permettono di mantenere la privacy di dati e comunicazioni, in un ambiente la cui infrastruttura è però condivisa. I canali permettono l'accesso agli *smart contract*, i quali sono istanziati su un canale e rendono disponibili interfacce per i metodi di business chiamabili dagli utenti.

I canali sono fondamentali per scenari in cui nella rete sono presenti competitors che non vogliono far conoscere a ogni partecipante il contenuto delle loro transazioni. Per esempio, potremmo avere A che vende prodotti a B e a C: A non vuole fare sapere a B che C sta pagando di meno, altrimenti richiederebbe di avere lo stesso prezzo. Potendo disporre di canali separati per la comunicazione con B e con C, A è sicuro che le informazioni saranno mantenute separate per ciascuno dei due clienti. Ogni peer infatti mantiene una copia separata del *ledger* relativo a ogni canale a cui è iscritto, il quale sarà visibile solo ai peer iscritti a tale canale.

Oltre ai canali “regolari” appena descritti, spesso definiti *application channels,* è presente nella rete anche un canale chiamato *system channel,* utilizzato dal servizio di ordinamento per mantenere le configurazioni di sistema.

### 3.4.2 Il ledger

Il *ledger* in Fabric è formato da due componenti: il *world state* e il *transaction log*. Il world state descrive lo stato del *ledger* in un determinato momento nel tempo: è il database del *ledger*. Il transaction log si occupa di registrare tutte le transazioni, la sequenza delle quali porta allo stato attuale del ledger: è l’update history del world state.

Il *datastore* del *ledger* per il world state è di default LevelDB, un database *key-value*, ma essendo un componente *pluggable*, è possibile utilizzare altri *datastore* (al momento solo CouchDB è supportato). Il *transaction log* non ha invece necessità di essere *pluggable*, in quanto semplicemente registra i *before* e *after values* relativi alle transazioni in un file append-only. Il log è anche chiamato semplicemente “blockchain”, infatti è qui che sono contenuti i blocchi, compreso il *genesis block*, tra loro incatenati tramite codici hash.

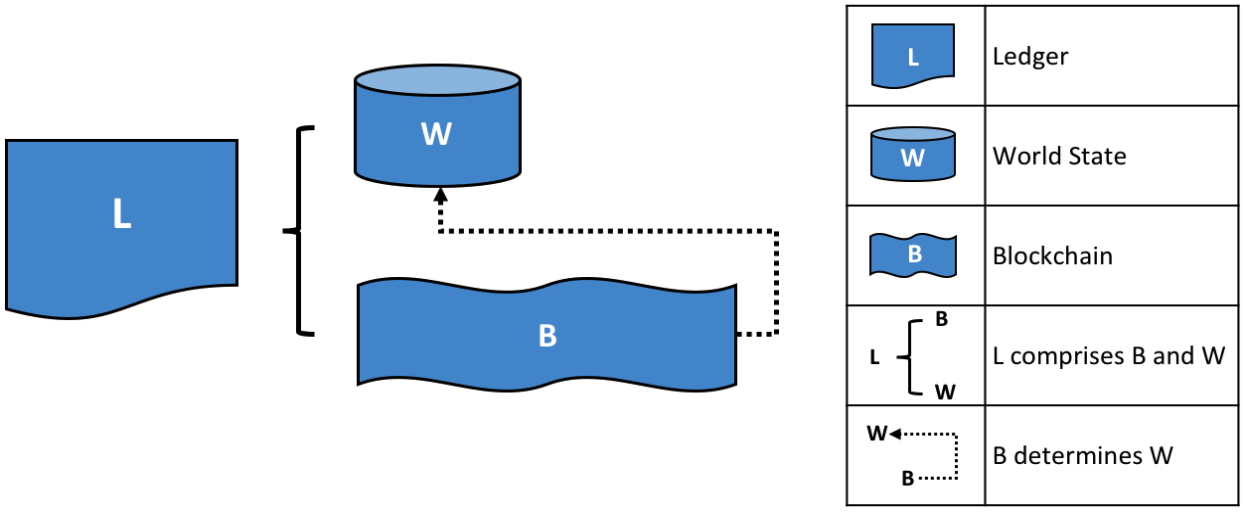



**Figura 7: *ledger* composto da world state e log (qui chiamato blockchain)**

Un blocco è formato da header, dati e metadati. L’header è formato da tre campi:

- Numero del blocco: un intero il cui valore è incrementato di 1 ad ogni creazione di un blocco. Il *genesis block* ha il numero 0.

- Hash del blocco corrente: l'hash di tutte le transazioni contenute nel blocco corrente.
- Hash dell'header del blocco precedente.

La sezione di metadati contiene invece:

- il certificato e la firma del creatore del blocco;
- un hash cumulativo di tutti gli update di stato avvenuti nei blocchi precedenti;
- un bitmap contenente un indicatore *valid/invalid* per ogni transazione.

Infine, la parte di dati contiene le transazioni ordinate, con una struttura ben definita.

Oltre ai due componenti appena descritti, potrebbe essere necessario anche un side database utilizzato per memorizzare dati privati, cioè dati che sono visibili solo ai membri delle organizzazioni definite in un file specificato al momento dell'istanziazione di un *chaincode*. Ad esempio, un oggetto rappresentante un'auto potrebbe avere una parte di dati pubblica, come il prezzo, e una parte privata, come la percentuale di guadagno. In tal caso, è possibile salvare lo stato "pubblico" dell'auto nel world state database e il suo stato privato in un database a parte. Le transazioni relative a questo side database sono visibili da tutte le entità sullo stesso canale, ma il contenuto è criptato e decifrabile solo dalle organizzazioni abilitate a vederlo. Anche il side database, come il log e il world state, contribuisce a formare il *ledger*.

### 3.4.3 Chaincode

Gli *smart contract* in Fabric sono chiamati *chaincode* e sono invocati da un'applicazione esterna alla blockchain quando essa necessita di interagire con il *ledger*. Nella maggior parte dei casi, il *chaincode* interagisce solo con il world state database e non con il transaction log. I linguaggi supportati per l'implementazione di *chaincode* sono Java, Go e Node. La scelta di un linguaggio piuttosto che un altro non comporta alcuna differenza a livello di capacità espressiva, infatti ognuno dei tre linguaggi sopracitati è pienamente supportato da numerose API e dalla community di sviluppatori. Nel prossimo capitolo, sarà analizzato nello specifico il caso di un *chaincode* Java, utilizzato per la parte progettuale della tesi.

Un *chaincode* contiene la definizione dei metodi di business per modificare lo stato degli oggetti gestiti e per condividere dati tra le varie organizzazioni che collaborano su blockchain. I metodi definiti servono per leggere e modificare dati nel *ledger*, con operazioni put, get e delete sul world state: put per inserire un oggetto, get per effettuare una query e delete per eliminare un oggetto; l'eliminazione qui è una semplice eliminazione dal world state e non dal log, i cui record sono di fatto immutabili.

Ogni *chaincode* ha una policy di endorsement associata, che indica quali organizzazioni devono firmare una transazione generata da uno *smart contract* affinché essa sia considerata valida. Ogni transazione, valida o meno, è aggiunta al *ledger*, ma solo quelle valide concorrono a modificare il world state. Il *chaincode* prende in ingresso un set di parametri di input chiamati *transaction proposal* e li usa in combinazione alla sua logica di programma per legger e scrivere il *ledger*. La transazione viene distribuita ai vari peer nella rete e ad essa è associata una *transaction response* contenente il *read-write* set degli stati letti da modificare e gli stati da scrivere se la transazione è valida. La validità della transazione viene verificata in due fasi: prima si controlla che essa abbia ricevuto le firme necessarie, poi che il *read-write* set non sia cambiato dal momento in cui è stato firmato. Se passa entrambi i set è marcata come valida e può quindi modificare il world state.

### 3.4.4 Il servizio di ordinamento

L'ordinamento delle transazioni, la loro approvazione, la creazione dei blocchi e la successiva diffusione di essi verso i vari peer sono compiti svolti dall'*orderer*, il nodo che si occupa del meccanismo di consenso nella rete. La definizione di un *orderer* è il primo passo per la definizione di una rete: un solo *orderer* è sufficiente per una piena funzionalità del sistema, tuttavia in scenari reali generalmente sono presenti più nodi *orderer*, appartenenti a una stessa organizzazione o a più organizzazioni, per avere maggiore *fault tolerance;* il nodo (o i più nodi) *orderer* rappresenta il servizio di ordinamento per la blockchain.

Spesso le blockchain pubbliche si basano su algoritmi di consenso probabilistici, i quali garantiscono la consistenza del *ledger* solo "*eventually*"[19], ma soffrono la possibilità di *fork* dovuti alla visione divergente che i peer possono avere riguardo all'ordinamento delle transazioni e quindi del *ledger*.

La presenza di un'entità ordinatrice implica la perdita della caratteristica di decentralizzazione tipica delle reti blockchain, ma nelle reti private questa è una caratteristica non fondamentale in quanto l'accento è posto più sulle prestazioni e sulla sicurezza che un meccanismo di questo tipo riesce a garantire. Inoltre, considerando una rete con più *orderer*, in realtà ci si trova di nuovo in uno scenario di consenso distribuito in quanto ci sarà necessità di un accordo fra le varie entità ordinatrici sulla successione delle transazioni.

Come già ampiamente detto, anche l'ordinamento è un servizio *pluggable* in Fabric e ciò implica che in base alle esigenze sia possibile utilizzare meccanismi diversi:

- Nel caso di un solo nodo *orderer*, il meccanismo è chiamato *Solo orderer*. Chiaramente, esso non presenta alcuna caratteristica di *fault tolerance*, ma è utile per scenari di test.
- Per ottenere caratteristiche di *fault tolerance*, è possibile utilizzare Raft, un servizio di ordinamento basato sull'implementazione del protocollo omonimo. Esso è basato su un sistema leader-follower in cui un leader su un canale decide l'ordinamento e i follower seguono il suo comportamento.
- L'ultima possibilità è utilizzare Apache Kafka in un sistema leader-follower simile a quello di Raft. Presenta molto overhead di configurazione e per questo motivo è spesso preferito Raft.

### Il flusso delle transazioni

Il flusso delle transazioni è diviso in tre fasi:

- Proposta;
- Ordinamento e packaging delle transazioni in blocchi;

[19] L'eventual consistency è un modello di consistenza che permette che lo stato di una base di dati passi per una finestra di inconsistenza, effettuando poi la conciliazione alla fine.

- Validazione e *commit*

Nella prima fase, un'applicazione client manda una proposta di transazione a un subset di peer che prima invocano un *chaincode* per produrre una proposta di update del *ledger* e poi approvano i risultati. I peer non applicano la proposta di update alla loro copia del *ledger* in questa fase, ma ritornano all'applicazione client una proposta di update. Le proposte delle transazioni approvate saranno ordinate in blocchi solo nella fase due e poi distribuite a tutti i peer per la validazione finale e il *commit* nella fase tre.

Nella seconda fase, le applicazioni mandano proposte di transazioni approvate al nodo di un servizio di ordinamento. Il servizio di ordinamento crea blocchi di transazioni che saranno infine distribuiti a tutti i peer sul canale per la validazione finale e il *commit* nella fase tre. Il primo compito del servizio di ordinamento è quindi quello di ricevere insiemi di transazioni in una sequenza ben definita e "impacchettarli" nei blocchi che formeranno la blockchain. In Hyperledger Fabric, i blocchi generati dal servizio di ordinamento sono finali, ciò implica che non ci possono essere *fork*, in quanto transazioni validate non saranno mai respinte o scartate.

La fase tre comincia con la distribuzione dei blocchi dall'*orderer* a tutti i peer connessi. Vale la pena notare che non tutti i peer necessitano di essere connessi a un *orderer*: i blocchi possono essere passati da un peer all'altro usando un protocollo di gossip. Il secondo compito di un nodo *orderer* è dunque distribuire i nodi ai peer. Una volta completata questa fase il *ledger* è aggiornato con successo.

### 3.4.5 Identità: Certificate Authority e Membership Service Provider

Ogni attore nella rete blockchain ha un'identità incapsulata in un certificato digitale rilasciato da una Certificate Authority. Queste identità servono a determinare gli esatti permessi sulle risorse e sull'accesso alle informazioni dei vari attori. I certificati sono digitalmente firmati dalla CA e legati alla chiave pubblica dell'attore. All'interno di una stessa organizzazione possono essere usate più di una Certificate Authority.

Per unirsi a una specifica rete, c'è necessità di convertire la propria identità digitale in qualcosa di riconosciuto all'interno di essa. È quindi necessario:

1. Avere un'identità rilasciata da una CA fidata.

2. Diventare membro di un’organizzazione che è riconosciuta e approvata dai membri della rete. L’MSP si occupa di collegare l’identità all’appartenenza a un’organizzazione.
3. Aggiungere l’MSP a un consorzio sulla rete o a un canale.
4. Assicurarsi che l’MSP sia incluso nelle definizioni di policy sulla rete.

Nonostante il nome, il Membership Service Provider non si occupa di “fornire membership”: l’implementazione di un MSP è infatti un set di cartelle che sono aggiunte alla configurazione della rete e che sono usate per definire un’organizzazione sia dall’interno (le organizzazioni decidono chi sono i propri admin) che dall’esterno (permettendo ad altre organizzazioni di validare che le entità hanno l’autorità per ciò che stanno tentando di fare). Mentre una CA genera i certificati che rappresentano le identità, la MSP contiene una lista di identità permesse. Ma il potere di una MSP va oltre il solo listare chi è un partecipante della rete o un membro di un canale. Essa trasforma un’identità in un ruolo identificando specifici privilegi che un attore ha su un nodo o un canale.

# Capitolo 4

# Approfondimento progettuale

Questo capitolo tratta il caso d'uso analizzato della tesi: la gestione della supply chain da parte di una grande azienda in ambito fashion. Illustrerò come la blockchain può essere utilizzata per costruire un'architettura che si occupi di svolgere tale compito e come poterla implementare grazie ad Hyperledger Fabric.

La sezione 4.1 illustra lo scenario della tesi; la sezione 4.2 descrive le scelte progettuali che sono state prese, in accordo alle quali si è deciso di utilizzare Hyperledger Fabric. La sezione 4.3 spiega il lavoro che è stato fatto per costruire la rete, la 4.4 illustra l'implementazione del *chaincode*, infine la sezione 4.5 mostra la costruzione di un'applicazione web di prova per testare la soluzione sviluppata.

## *4.1 Blockchain per supply chain*

La blockchain ha buone probabilità di diventare la tecnologia sulla quale le aziende saranno maggiormente portate a investire in futuro per la gestione della supply chain. Il motivo? Perché semplifica lo scambio dei dati e potrebbe avere un forte impatto a livello di vantaggi per tutta la filiera coinvolta nel progetto. Possiamo pensare che i vantaggi si possano estendere anche ai clienti finali nel caso si voglia estendere la visibilità della "storia" di ogni prodotto (o parte di esso). In questa direzione vanno ad esempio iniziative in ambito agroalimentare che utilizzano la blockchain per garantire al consumatore la tracciabilità dei prodotti, si veda il caso di Spinosa Spa[20] che ha implementato una blockchain per garantire la provenienza del latte per le loro mozzarelle di bufala oppure di Carrefour per la sua "Filiera qualità Carrefour"[21]. Un altro esempio può essere la tracciabilità dei componenti di un'auto, come in caso di un

[20] https://www.spinosaspa.com/blockchain/

[21] https://www.mark-up.it/carrefour-il-pollo-filiera-qualita-tracciato-con-tecnologia-blockchain/

progetto di BMW[22] per fornire agli acquirenti i dati sulle raffinerie e miniere di provenienza delle materie prime.

Tuttavia, il caso studiato non ha come finalità quella di certificare al cliente la provenienza di un prodotto, ma l'iniziativa mira piuttosto a impiegare la blockchain per rafforzare la rete di scambi e la fiducia che c'è fra fornitori, richiedenti, enti di certificazione, ecc., i quali convivono all'interno di un ecosistema con regole e garanzie ben definite.

L'azienda in questione si occupa interamente di eyewear di lusso e vuole porsi come la prima nel settore a implementare un approccio innovativo basato su Distributed Ledger Technology. Il flow del prodotto, brevemente, si può descrivere così: si parte da una fase preliminare in cui la direzione aziendale decide i disegni degli occhiali da produrre, quindi decide i fornitori ai quali affidarne la produzione, i quali preparano l'industrializzazione dell'oggetto e avvisano a loro volta i loro sub-fornitori; un ulteriore attore della catena è l'ente di certificazione che si occupa di fornire alle aziende i certificati richiesti relativi a un occhiale.

Il segreto del funzionamento della blockchain, in questo caso, è che "l'unione fa la forza", cioè l'alleanza delle varie aziende coinvolte nel processo permette di ottenere risultati vantaggiosi per ognuno degli attori all'interno della rete, che vanno ben oltre il semplice tracciamento dello stato di un prodotto o della qualità di esso. Gli obiettivi fondamentali che si vogliono perseguire sono:

- La trasparenza: le parti coinvolte nella catena devono poter registrano i dati rendendoli disponibili agli altri player con cui comunicano, in modo che siano certificati e che la rappresentazione di essi sia univoca tra i vari attori, superando così l'opacità delle supply chain. Questo deve garantire tempi rapidi di identificazione nel caso di errori o problematiche relative ai dati scambiati.
- La rapidità e la protezione degli scambi: una piattaforma blockchain che permetta transazioni abbastanza rapide e garantisca l'affidabilità dei dati. Inoltre, essi devono essere resi visibili solo agli utenti autorizzati.

[22] https://www.bmw.com/it/innovation/blockchain-automotive.html

- Lo sviluppo collaborativo: una modalità di funzionamento che favorisca la collaborazione tra i vari anelli della supply chain. Le informazioni scambiate devono poter essere infatti consultate e verificate dai player, rendendo quindi possibile la nascita di diverse forme di collaborazione.

## 4.2 *Scelte progettuali*

Quali scelte implementative prendere per poter realizzare un progetto che risponda alle necessità descritte? Innanzitutto, bisogna cercare di capire se è meglio una blockchain pubblica o privata: qui la scelta è facile in quanto emerge chiara e forte la necessità di avere attori ben noti ed autorizzati ad agire nella rete. Soluzioni quindi basate ad esempio su Ethereum non sono da considerarsi e non sono neppure molto sensate in un ambiente di questo tipo. Le piattaforme blockchain disponibili sono numerose e ognuna con le proprie caratteristiche, ma quale risulta essere la migliore? Tra le piattaforme già citate nel capitolo 2 - Quorum, Corda e Hyperledger Fabric - proprio su quest'ultima è ricaduta la scelta, per via di alcune sue caratteristiche quali:

- Il più alto numero di transazioni per secondo, fino a 110000[23];
- Modello di comunicazione tra organizzazioni basato su canali, il quale ricalca perfettamente lo scenario di interazione desiderato;
- Possibilità di affiancare dati privati a dati pubblici all'interno dello stesso canale;
- Architettura scalabile e fortemente personalizzabile;
- Grande disponibilità di documentazione online, codice open source e possibilità di sviluppare *chaincode* in linguaggi non *domain-specific* (Go, Java e Node.js).

Ogni organizzazione può disporre di vari peer nella rete, sui quali sono installati i *chaincode*. Per questioni di sicurezza e isolamento, un'organizzazione non può ospitare sulle proprie macchine, dove è fatto il *deploy* dei peer, uno o più *orderer*: per evitare comportamenti malevoli indesiderati, non è sicuro che un'organizzazione "possegga" un *orderer*, dunque è consigliato che sia un'organizzazione esterna ad occuparsi dei meccanismi di ordinamento delle transazioni e creazione dei blocchi.

[23] https://webthesis.biblio.polito.it/7436/1/tesi.pdf

Il modello di test da costruire sarà basato su due organizzazioni che collaborano per la progettazione di occhiali. Un'organizzazione si occupa di inserire la definizione di un eyewear in blockchain e l'altra collegherà all'eyewear il codice di una lente, che si presuppone precaricata in blockchain.

## *4.3 Costruzione della rete*

Come ambiente di sviluppo sono state scelte macchine Linux montanti Ubuntu Server 18.04 e la versione 1.4 di Hyperledger Fabric. Per testare il funzionamento di un'applicazione che risponda alle scelte progettuali prese, è opportuno partire con il codice di base fornito da Hyperledger Fabric[24]. Questa rete sample è utile per prendere dimestichezza con l'architettura e il funzionamento della rete. Sul sito di Fabric sono disponibili numerosi tutorial ed esempi di applicazioni dai quali poter partire per sviluppare le proprie soluzioni.

Una volta capito il funzionamento dei vari componenti e del flow di esecuzione della piattaforma, è tempo di passare a un *deploy* distribuito per provare le prestazioni e il funzionamento in un contesto reale. La soluzione si basa sulla costruzione di una rete composta da due organizzazioni, Org1 e Org2, ognuna delle quali possiede due peer. Per l'ordinamento la rete si affida a una configurazione basata su Solo Orderer, chiamato Orderer0. I peer delle organizzazioni possono comunicare tra loro facendo join sul canale *mychannel.* Tutti i peer connessi al canale avranno la stessa rappresentazione del *ledger* e una copia installata del *chaincode*. La scelta di installare il *chaincode* su tutti i peer è stata presa per rendere possibile la chiamata da parte di un utente verso qualsiasi peer, ciò anche a favore di una migliore *fault-tolerance* e replicazione dei servizi. Come database è stato scelto LevelDB, che fornisce migliori prestazioni rispetto a CouchDB se non sono richieste query complesse.

In Fig. 6 è mostrata una rappresentazione grafica che riassume i componenti della rete: nell'immagine, L1 e S1 rappresentano rispettivamente il *ledger* e il *chaincode*; i peer sono chiamati Peer0Org1, Peer1Org1, Peer0Org2 e Peer1Org2, in modo molto esplicativo.

[24] Istruzioni alla pagina: https://hyperledger-fabric.readthedocs.io/en/latest/build_network.html

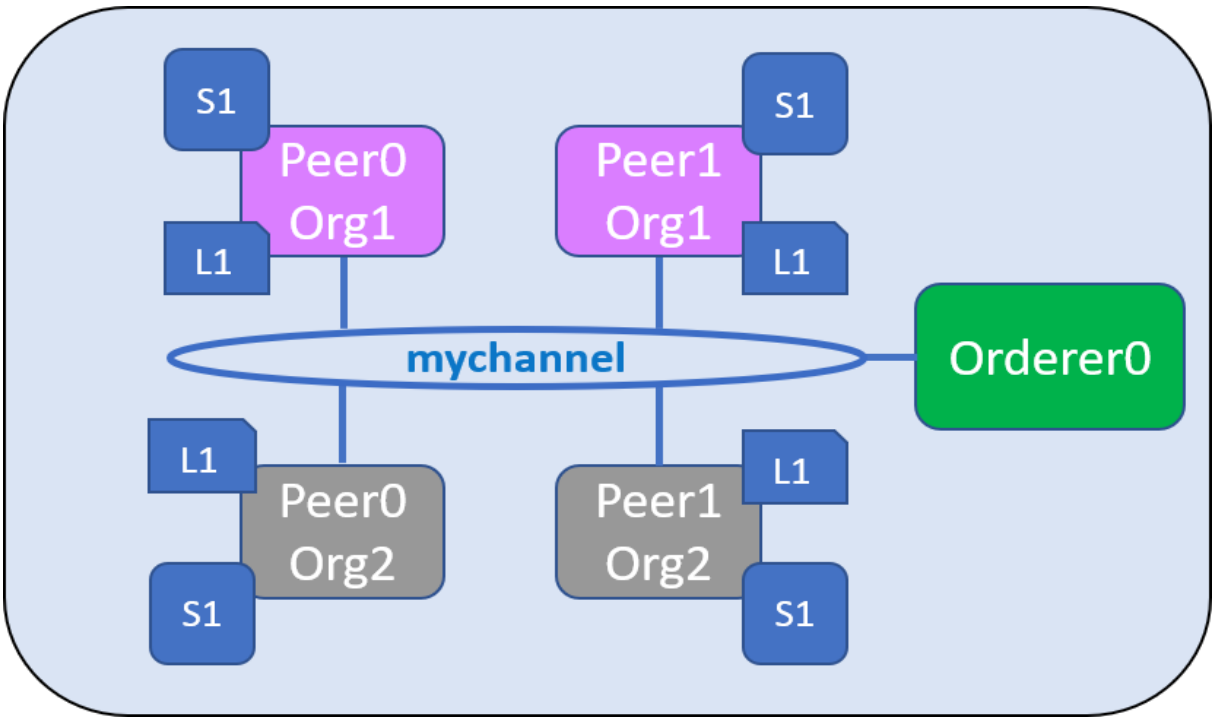



**Figura 8: illustrazione dei principali componenti della rete**

Il *deploy* di peer, *orderer* e *chaincode* è effettuato tramite container con Docker[25]. Il *deploy* dell'*orderer,* per semplicità, può essere effettuato sullo stesso nodo di una delle due organizzazioni, in quanto in ambiente di test non abbiamo particolari necessità di prevenire manomissioni dell'ordinamento da parte di un'organizzazione. La macchina che ospiterà *l'orderer* e l'organizzazione 1 verrà indicata come devlkey01, mentre la macchina per l'organizzazione 2 come devlkey02. Per abilitare la comunicazione fra i nodi delle due organizzazioni, la soluzione scelta è quella di creare uno Swarm Docker.

Per la costruzione di una rete blockchain funzionante, Fabric richiede l'esecuzione di una lunga sequenza di comandi. Il primo passo per lo sviluppo è clonare su tutti i nodi la repository sample di Hyperledger e installare *dependecy* e *binary* relativi. Per generare i dati crittografici, Fabric mette a disposizione il tool `cryptogen` che genera i certificati e gli artefatti necessari per la gestione del canale. Il comando, da chiamare all'interno della cartella clonata, è:

```
cryptogen generate --config=./crypto-config.yaml
```

[25] https://www.docker.com/

dove `cryptoconfig.yaml` è il file di configurazione che contiene la definizione dei partecipanti e dei componenti della rete per cui creare i certificati.

Un altro tool è `configtxgen`, che consuma il file `configtx.yaml` contenente la topologia della rete; la sequenza di comandi da eseguire, è:

```
configtxgen -profile TwoOrgsOrdererGenesis -channelID $SYS_CHANNEL -outputBlock ./channel-artifacts/genesis.block
```

per generare il genesis block; poi:

```
configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $SYS_CHANNEL
```

per generare la transazione della creazione del canale.

Infine, si genera l'update per gli anchor peer (entità presenti su un canale che permettono *discovery* dei peer tramite protocollo di gossip) per entrambe le organizzazioni:

```
configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate <OrgMSPanchors.txt> -channelID $SYS_CHANNEL -asOrg <orgName>
```

Le cartelle create dai precedenti comandi, `crypto-config` e `channel-artifacts`, andranno copiate con tutto il loro contenuto su entrambi i nodi.

Quindi è necessario creare uno Swarm utilizzando il comando

```
docker swarm init --advertise-addr <host-1 ip address>
```

e poi fare join dell'altro nodo allo swarm:

```
docker swarm join -token <token> <host-1 Ip Address> ––advertise-addr <host-2 Ip Address>
```

Il successivo passo è la creazione di una rete overlay, utile quando si vogliono collegare due container facenti parte di due *host* diversi:

```
docker network create --attachable --driver overlay fabric
```

A questo punto è possibile procedere con la vera costruzione della rete blockchain, effettuando il *deploy* dei container per i peer, *l'orderer* e il CLI[26] container fornito da

[26] Command Line Interface

Hyperledger Fabric, utile per effettuare operazioni sui peer senza dover accedere a essi. Per lanciare i container, è necessario eseguire, per ognuno di essi:

```
docker stack deploy --compose-file <entry> $DOCKER_STACK
```

Eseguiti questi passaggi, abbiamo una situazione come in figura 7, verificabile su ogni *host* con il comando `docker ps`.

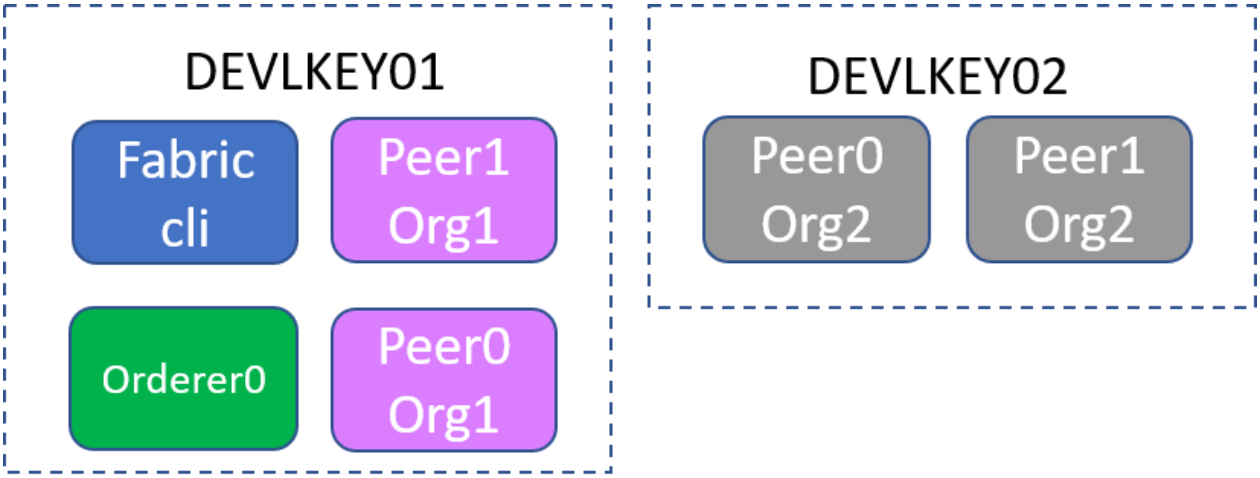



**Figura 9: *deploy* dei componenti sulle macchine**

Per terminare la costruzione della rete è necessario creare il canale, unire i peer ad esso e aggiornare gli anchor peer. Accedendo al container della CLI si può eseguire il seguente comando:

```
peer channel create -o orderer0.example.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/channel.tx
```

per creare il canale; dunque:

```
peer channel join -b $CHANNEL_NAME.block
```

avendo cura di modificare ogni volta le variabili d'ambiente della CLI per indicare i diversi peer da aggiungere.

Infine, per aggiornare gli anchor peer:

```
peer channel update -o orderer0.example.com:7050 -c <channel-name> -f ./channel-artifacts/${CORE_PEER_LOCALMSPID}anchors.tx
```

Ora è possibile per gli utenti accedere ai peer tramite il canale, sul quale invocare le operazioni di lettura/scrittura sul *ledger,* le quali sono definite sul *chaincode* da implementare.

## *4.4 Implementazione e installazione chaincode*

Per l'implementazione del *chaincode* è stato scelto Java come linguaggio di programmazione e IntelliJ IDEA Community come IDE. Le librerie da utilizzare per poter implementare il chaincode sono fabric-protos[27] e fabric-chaincode-shim[28], disponibili su Maven. Esse offrono le API di basso livello per poter interagire con il *ledger*.

Occhiali e lenti sono rappresentati da semplici classi "*bean*", rispettivamente Eyewear e Lens. La classe astratta ChaincodeBase fornita da fabric-chaincode-shim è estesa dalla classe SimpleChaincode. Essa contiene la definizione di tutti i metodi invocabili da un utente e in particolare fa *l'override* delle funzioni `invoke` e `init`:

- `init` è chiamata automaticamente quando il *chaincode* viene istanziato; di solito contiene la definizione di operazioni utili per inizializzare dati in blockchain. A titolo esemplificativo, `init` nel codice è usata per creare un eyewear.
- `invoke` è usata per ricevere tutte le chiamate alle funzioni definite nel *chaincode*, funzionando quindi come "router" per instradare le richieste verso differenti *path*.

Sotto, il codice del metodo `init` e di parte del metodo `invoke`.

```
@Override
public Response init(ChaincodeStub stub) {
    Eyewear eyewear = new Eyewear("1", "yellow");
    ObjectMapper objectMapper = new ObjectMapper();
    try {
        String ID = "1";
        stub.putState(ID, objectMapper.writeValueAsBytes(eyewear));
    } catch (JsonProcessingException e) {
        e.printStackTrace();
        return ResponseUtils.newErrorResponse("Initialization error");
    }
    return ResponseUtils.newSuccessResponse("Init");
}
```

[27] https://mvnrepository.com/artifact/org.hyperledger.fabric-chaincode-java/fabric-chaincode-protos/1.4.4

[28] https://mvnrepository.com/artifact/org.hyperledger.fabric-chaincode-java/fabric-chaincode-shim

```
@Override
public Response invoke(ChaincodeStub stub) {
    String func = stub.getFunction();
    List<String> params = stub.getParameters();
    switch (func) {
        case "createEyewear":
            return createEyewear(stub, params);
        case "getEyewear":
            return getEyewear(stub, params);
        …
            }
    return ResponseUtils.newErrorResponse("Unsupported method");
}
```

Tutti i metodi per comunicare con il *ledger* hanno come input un oggetto ChaincodeStub e come output un oggetto Response:

- Il ChaincodeStub è passato alle chiamate del *chaincode* dalla piattaforma di Fabric. Lo *stub* incapsula le API tra l'implementazione del *chaincode* e il peer.
- Response rappresenta l'oggetto ritornato da un'invocazione del *chaincode*.

Tra i metodi che lo *stub* fornisce, ci sono:

- `stub.putState(String key, Byte[] value)` che è usato per inserire un oggetto in blockchain in una forma <Key, Value>;
- `stub.getFunction()` per ottenere il nome della funzione richiesta dall'invocazione del client;
- `stub.getParameter()` per ottenere i parametri passati nella chiamata, da passare alla funzione richiesta.
- `stub.getStringState(String key)` per ricercare un oggetto per chiave nel *ledger* e ottenerne la rappresentazione sotto forma di stringa.
- `stub.delState(String key)` per eliminare un'oggetto per chiave.

È quindi necessario implementare tutta la serie di funzioni chiamabili dall'utente necessarie a creare, modificare, eliminare e ricercare oggetti nella blockchain. A titolo esemplificativo, nella pagina accanto è presente il codice per l'aggiunta, eliminazione e modifica di eyewear.

```
private Response deleteEyewear(ChaincodeStub stub, List<String> args) {
    if (args.size() != 1) {
        return ResponseUtils.newErrorResponse("Incorrect number of arguments. Expecting
1");
    }
    String key = args.get(0);
    if (!checkString(stub.getStringState(key)))
        return ResponseUtils.newErrorResponse("Eyewear not found");
    stub.delState(key);
        return ResponseUtils.newSuccessResponse("Eyewear deleted");
}

private Response getEyewear(ChaincodeStub stub, List<String> args) {
    if (args.size() != 1)
        return ResponseUtils.newErrorResponse("Incorrect number of arguments, expecting
1");
    String Id = args.get(0);
    if (!checkString(Id))
        return ResponseUtils.newErrorResponse("Invalid argument");
    try {
        String eyewearString = stub.getStringState(Id);
        if (!checkString(eyewearString))
            return ResponseUtils.newErrorResponse("Nonexistent eyewear");
        return ResponseUtils.newSuccessResponse(eyewearString);
    } catch (Throwable e) {
        return ResponseUtils.newErrorResponse(e.getMessage());
    }
}

private Response modifyEyewear(ChaincodeStub stub, List<String> args) {
    if (args.size() != 2)
        return ResponseUtils.newErrorResponse("Incorrect number of arguments, expecting
2");
    String ID = args.get(0);
    String lensDescr = args.get(1);
    if (!checkString(ID) || !checkString(lensDescr))
        return ResponseUtils.newErrorResponse("Invalid argument(s)");
    if (!checkString(stub.getStringState(ID)))
        return ResponseUtils.newErrorResponse("Eyewear not found");
    ObjectMapper objectMapper = new ObjectMapper();
    try {
        Eyewear eyewear = objectMapper.readValue(stub.getStringState(ID),
Eyewear.class);
        if (eyewear.getLensID().equals("not linked")) {
            eyewear.setLensDescr(lensDescr);
            stub.putState(ID, objectMapper.writeValueAsBytes(eyewear));
        }
        …
    }
    return ResponseUtils.newSuccessResponse("Modified");
}
```

Ulteriore decisione implementativa, necessaria da spiegare a questo punto della trattazione, è la rappresentazione usata per gestire gli oggetti Lens nel sistema: a differenza di un Eyewear, che è visibile a ogni entità nella rete blockchain, Lens si considera come un dato privato della seconda organizzazione. Org1 deve poter vedere solo l'ID della lente associata e non ulteriori dettagli che possono essere dati sensibili che Org2 non vuole mostrare. Si è quindi deciso di trattare gli oggetti Lens come "private data", gestiti in Fabric tramite un side database, quindi non appartenenti al database principale in cui sono memorizzati gli eyewear. Allo stesso modo ORG1 può aver bisogno di dati privati alla sua organizzazione, come il prezzo di un Eyewear, che rappresenteremo con la classe Price. Un oggetto Price è creato per essere associato a un eyewear e la sua "vita" nel sistema è legata a quella di un eyewear, dunque al momento dell'eliminazione di un Eyewear anche il suo Price viene eliminato.

La definizione di una collezione di dati privati è possibile tramite un file di configurazione da passare al momento dell'istanziazione del *chaincode*. Si veda il codice sotto, contenuto nel file `collections.json`.

```
{
    "name": "Lenses",
    "policy": "OR('Org2MSP.member')",
    "requiredPeerCount": 1,
    "maxPeerCount": 1,
    "blockToLive":0
},
{
    "name": "Prices",
    "policy": "OR('Org1MSP.member')",
    "requiredPeerCount": 1,
    "maxPeerCount": 1,
    "blockToLive":0
}
```

In questo file sono indicate le organizzazioni autorizzate a vedere le collezioni Prices e Lenses, con le policy e le configurazioni necessarie.

Per poter interagire con i dati privati, ChiancodeStub mette a disposizione i metodi:

- `stub.putPrivatedata(String collection, String key, Byte[] value)` per inserire un oggetto in una collezione di dati privati (es. “Lenses”);
- `stub.getPrivateDataUTF8(String collection, String key)` per ottenere la rappresentazione in forma di stringa di un oggetto privato.

Si veda come esempio il codice sotto per aggiungere ed eliminare un Price:

```
private Response addPrice(ChaincodeStub stub, List<String> args) {
    if (args.size() != 2) {
        return ResponseUtils.newErrorResponse("Incorrect number of arguments. Expecting
2");
    }
    String ID = args.get(0);
    String price = args.get(1);
    if (!checkString(ID) || !checkDouble(price))
        return ResponseUtils.newErrorResponse("Invalid argument(s)");

    Price pr = new Price(Double.parseDouble(price));
    try {
        if (!checkString(stub.getStringState(ID)))
            return ResponseUtils.newErrorResponse("Nonexistent eyewear");
        if (checkString(stub.getPrivateDataUTF8("Prices", ID)))
            return ResponseUtils.newErrorResponse("Price already present");
        ObjectMapper objectMapper = new ObjectMapper();
        stub.putPrivateData("Prices", ID, objectMapper.writeValueAsBytes(pr));
        …
        }
    return ResponseUtils.newSuccessResponse("Price added");
}


private Response deletePrice(ChaincodeStub stub, List<String> args) {
    if (args.size() != 1) {
        return ResponseUtils.newErrorResponse("Incorrect number of arguments. Expecting
1");
    }
    String key = args.get(0);
    if (!checkString(stub.getPrivateDataUTF8("Prices", key)))
        return ResponseUtils.newErrorResponse("Price not found");
    if (checkString(stub.getPrivateDataUTF8("Prices", key)))
        stub.delPrivateData("Prices", key);
    return ResponseUtils.newSuccessResponse("Price deleted");
}
```

Ulteriore importante funzione da implementare è `linkEyewear`, che permette agli utenti di Org2 di collegare una lente a un eyewear. Una volta effettuato il link, l'eyewear non è più modificabile. Sotto, il codice di `linkEyewear`:

```
private Response linkEyewear(ChaincodeStub stub, List<String> args) {
    if (args.size() != 2)
        return ResponseUtils.newErrorResponse("Incorrect number of arguments, expecting
2");
    String ID = args.get(0);
    String lensID = args.get(1);
    ObjectMapper objectMapper = new ObjectMapper();

    String eyewearString = stub.getStringState(ID);
    try {
        Eyewear eyewear = objectMapper.readValue(eyewearString, Eyewear.class);
        eyewear.setlensID(lensID);
        stub.putState(ID, objectMapper.writeValueAsBytes(eyewear));
    } catch (Exception e) {
        return ResponseUtils.newErrorResponse(e.getMessage());
    }
    return ResponseUtils.newSuccessResponse("linked");
}
```

La classe SimpleChaincode include in totale 21 metodi, non elencati qui per ragioni di spazio, che permettono tutte le operazioni per creazione, aggiunta, modifica e ricerca (per chiave e ricerca per intervallo) di Eyewear, Lens e Price, oltre all'operazione di link e altri metodi accessori.

Le operazioni per rendere il *chaincode* accessibile prevedono prima l'installazione di esso, cioè la distribuzione del codice sui peer che intendono utilizzarlo, e l'istanziazione sul canale, cioè la compilazione del codice del *chaincode* e la creazione di un container dedicato per l'esecuzione dello stesso su ognuno dei peer che lo ha installato. La combinazione di installazione e istanziazione consente a un peer di utilizzare un singolo *chaincode* su molti canali.

Dalla CLI, è necessario eseguire il comando per l'installazione del *chaincode* "eyewear1":

```
peer chaincode install -n eyewear1 -v ${VERSION} -l ${LANGUAGE} -p
${CC_SRC_PATH}
```

su ogni peer.

Quindi un solo peer effettua l'istanziazione sul canale, ad esempio peer0Org2:

```
peer chaincode instantiate -o orderer0.example.com:7050 -C $CHANNEL_NAME -n eyewear1 -l ${LANGUAGE} -v ${VERSION} -c '{"Args":["init"]}' -P "OR ('Org1MSP.peer','Org2MSP.peer')" --collections-config $GOPATH/src/github.com/chaincode/chaincode_example/collections.json
```

Si noti che in coda a questo comando è passato anche il file di configurazione per le collezioni di dati privati.

A questo punto l'ambiente è pronto per essere testato ed ogni peer è pronto per eseguire e processare le richieste di transazioni degli utenti. Dalla CLI è possibile quindi provare a richiamare i metodi esposti dal *chaincode*; la sintassi per un'invocazione è come segue:

```
peer chaincode invoke -o orderer0.example.com:7050 -C mychannel -n eyewear5 --peerAddresses peer0.org1.example.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt -c '{"Args":["createEyewear","2", "light grey"]}'
```

Con il comando appena illustrato è possibile invocare una transazione per l'inserimento di un eyewear in blockchain. L'invocazione contiene i certificati necessari all'identificazione: si noti che essa è essenziale, infatti una transazione potrebbe richiedere dati privati e solo presentando il certificato di appartenenza a un'organizzazione è possibile accedere a tali dati. Si confronti a tale proposito l'output su riga di comando di un'operazione di `getRangeLenses()`, la quale ritorna le lenti il cui ID è in un certo range. L'operazione, effettuata da un utente di Org2, ha output:

```
Chaincode invoke successful. result: status:200 message:"Lenses initialized"
```

**Figura 10: output di `initLenses` (Org1)**

La stessa operazione, invocata da un utente di Org1, restituirà invece:

```
endorsement failure during invoke. response: status:500 message:"failed to execute transaction
01: error sending: chaincode stream terminated"
```

**Figura 11: output di `initLenses` (Org2)**

Come già detto, il world state è gestito tramite database, mentre la blockchain è un file di log: entrambi sono salvati nel filesystem dei peer e si possono visualizzare anche da linea di comando navigando alle directory apposite (/var/hyperledger/production/ledgersData/chains/chains/mychannel è il percorso di default per la blockchain, /var/hyperledger/production/ledgersData/stateLeveldb per il world state).

Un ultimo step necessario per poter passare alla fase di test tramite applicazione web è l'inizializzazione di due server che fungano da Certificate Authority; anche le CA saranno gestite tramite container con Docker:

```
docker stack deploy --compose-file docker-compose-ca.yaml fabric
```

Come implementazione si è scelto la Certificate Authority default di Fabric, Fabric-ca: fabric-ca1 sarà il server di cui verrà fatto il *deploy* su devlkey01, mentre fabric-ca2 è il server per devlkey02.

## *4.5 Applicazione web di test*

Per testare il funzionamento dello *smart contract* in modo più intuitivo che da terminale, è stato deciso di sviluppare un'applicazione web di test che permetta a un utente di mandare richieste di transazioni sul *ledger* in modo semplice, gestendo gli oggetti e interrogando il *ledger*. Il *deploy* dell'applicazione è stato effettuato utilizzando Apache Tomcat[29] come web server, con l'accesso possibile dalla rete aziendale di Iconsulting.

In particolare, il design si basa su due pagine jsp sviluppate per simulare lo scenario di un'applicazione di Org1 e di un'applicazione di Org2, con le relative operazioni disponibili per i propri utenti. A queste pagine si accede tramite una pagina di login che riconosce le credenziali degli utenti autorizzati ad accedervi, inoltrando l'utente alla pagina relativa alla propria organizzazione. Sotto, in Fig. 12 e 13, sono mostrate le grafiche per le pagine jsp di Org1 e Org2.

[29] https://tomcat.apache.org/download-80.cgi

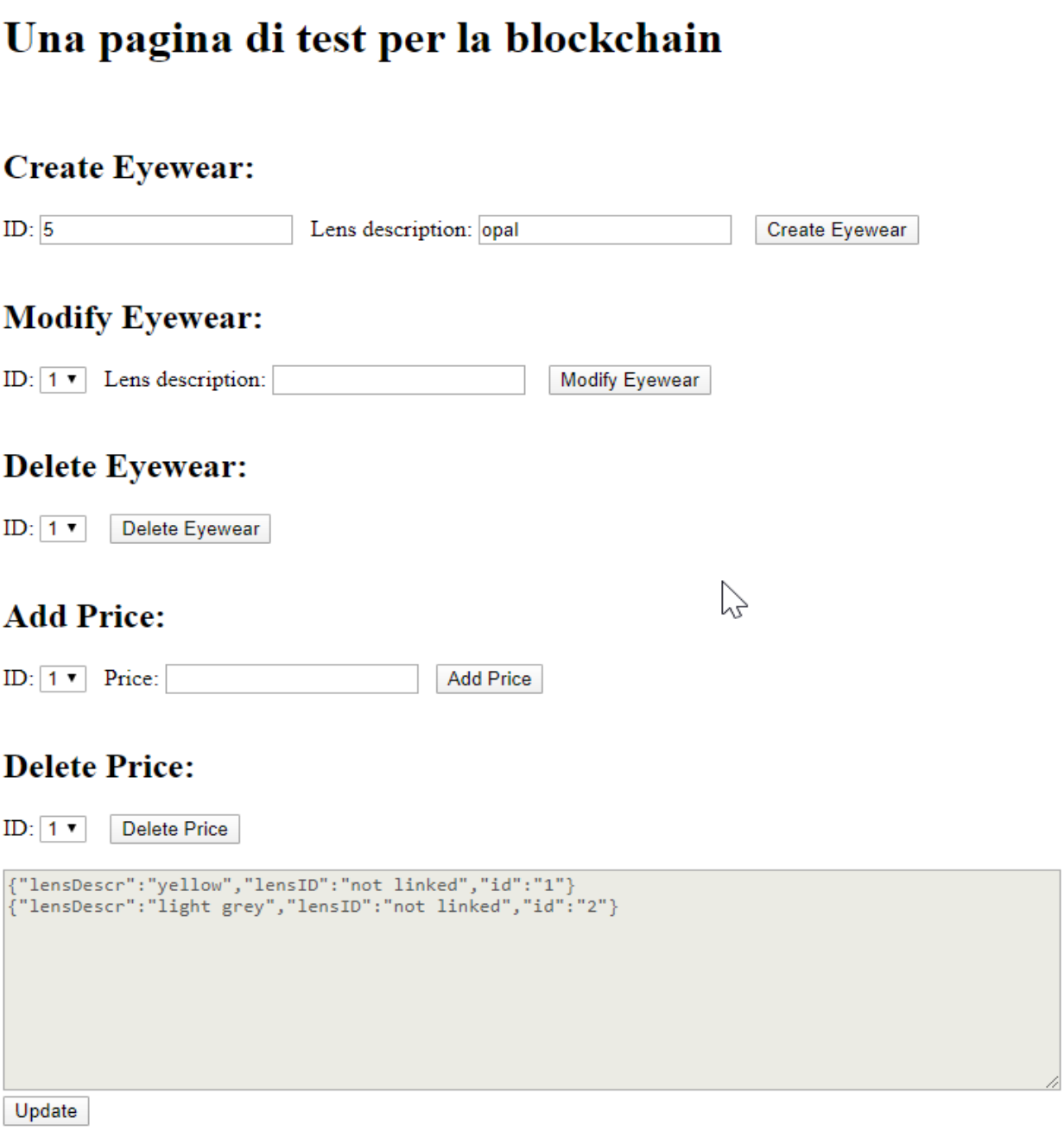



**Figura 12: pagina Org1.jsp**


**Figura 13: pagina Org2.jsp**

Come si nota, nella pagina della prima organizzazione sono disponibili i metodi per creare, modificare ed eliminare un Eyewear, come pure quelli per gestire un Price. Nella pagina della seconda organizzazione è disponibile solo il metodo per effettuare il link di un Eyewear a una Lens. Questo per ricalcare il business flow delle due organizzazioni. A fondo pagina è disponibile un'area di testo in cui sono riportati gli eyewear presenti dal sistema.

### 4.5.1 Hyperledger Fabric Gateway Java SDK

Il funzionamento dell'applicazione è reso possibile dal Fabric Gateway Java SDK[30], il quale implementa le API di alto livello per comunicare con una rete Fabric, permettendo dunque a un'applicazione di sottomettere transazioni per interagire con il *chaincode*. Il progetto comprende una classe astratta BCConnector per inizializzare le informazioni "base" di connessione al *chaincode:* le classi che la estendono, BCConnectorOrg1 e BCConnectorOrg2, adattano la superclasse alla specifica applicazione per la loro organizzazione. Altre classi sono UserContext per rappresentare le informazioni di un utente e Util che contiene metodi di utilità.

[30] https://github.com/hyperledger/fabric-gateway-java

Il primo passo è creare un cliente per il fabric-ca server e fare *enroll* di un admin. Il procedimento non è necessario se già si dispone di chiavi e certificato, ma è utile mostrarlo per ricalcare un caso d'uso reale. Lo step successivo è il settaggio dell'utente admin all'interno di uno UserContext, la classe che plasma il concetto di utente dell'applicazione (con nome, ruolo, account, affiliazione, ecc.…). Successivamente è necessario creare gli oggetti le cui classi ricalcano le entità della rete, in particolare Peer, Orderer e Channel. Sotto è mostrato parte del codice per le operazioni appena descritte.

```
public BCConnectorOrg1() throws Exception {
	…
	setCaClient(HFCAClient.createNewInstance("https://localhost:7054", getProps()));

	setAdminEnrollment(getCaClient().enroll("admin", "adminpw"));
	getAdminUserContext().setEnrollment(getAdminEnrollment());

	Util.writeUserContext(getAdminUserContext());
	…
	setPeer_url("grpc://localhost:7051");

	setProps(new Properties());
	getProps().put("pemFile", getCertPath());
	setPeer(getHfClient().newPeer(getPeer_name(), getPeer_url(), getProps()));

	setOrderer(getHfClient().newOrderer(getOrderer_name(), getOrderer_url(),
getOrderer_properties()));

	setChannel(getHfClient().newChannel("mychannel"));
	getChannel().addPeer(getPeer());
	getChannel().addOrderer(getOrderer());
	getChannel().initialize();
}
```

Infine, si possono invocare le operazioni sul *chaincode*, come si può vedere nell'estratto di codice sottostante, in cui è mostrato come viene chiamata la funzione `createEyewear`.

```
public String operation(String func, String[] args) throws Exception {

	TransactionProposalRequest request =
getHfClient().newTransactionProposalRequest();
	request.setChaincodeID(getCcid());

	if (func.equals("createEyewear")) {
		request.setFcn(func); // Chaincode invoke function name
		String[] arguments = { args[0], args[1] };
		request.setArgs(arguments);

		request.setProposalWaitTime(10000);
		Collection<ProposalResponse> responses =
	getChannel().sendTransactionProposal(request);
		for (ProposalResponse res : responses) {
			System.out.println(res.getMessage());
		}
		getChannel().sendTransaction(responses);
	}

	…
```

Le transazioni che sono state *committed* portano a una modifica dello stato della rete, il quale può essere interrogato tramite query. Il procedimento per proporre una QueryByChaincodeRequest è il seguente:

```
	QueryByChaincodeRequest queryRequest =
getHfClient().newQueryProposalRequest();
	queryRequest.setChaincodeID(getCcid());
	queryRequest.setFcn("getRangeEyewear");
			// Query the chaincode
	Collection<ProposalResponse> queryResponse =
getChannel().queryByChaincode(queryRequest);

	for (ProposalResponse pres : queryResponse) {
		// Do something
	}
```

Le funzioni chiamabili dall'utente sul *chaincode*, come `getRangeEyewear` e `createEyewear`, sono gestite dalle *form* delle jsp, le quali effettuano le chiamate alla funzione `operation` di BCConnector.

Il flow di esecuzione e di utilizzo della rete è dunque completo: un utente può accedere alla pagina web della sua organizzazione per invocare operazioni sul *chaincode* sottostante, il quale implementa i metodi per l'accesso al *ledger* e per la comunicazione tra i peer. Tutte le operazioni saranno registrare in blockchain e in particolare quelle *committed* andranno a modificare il world state. Lo scopo era quello di mostrare la

gestione della supply chain da parte di un’azienda di moda e un suo partner, tramite una rete sample e un’applicazione di test che potessero mettere in luce in particolare i particolari progettuali relativi alle operazioni svolte in blockchain.

# Capitolo 5

# Conclusioni

Questo capitolo riassume i risultati della tesi, analizzando quanto visto riguardo ad Hyperledger Fabric e alla parte progettuale. La tesi si conclude identificando alcune future possibili direzioni di ricerca per la blockchain blockchain, in particolare per Hyperledger Fabric, e sugli impatti che l'utilizzo di essa può avere nella supply chain.

L'analisi della tecnologia blockchain ha evidenziato le caratteristiche principali e le varie tipologie che si possono distinguere. La caratteristica fondamentale della blockchain è la memorizzazione di dati transazionali in blocchi immutabili collegati tra loro da codici hash. Le reti blockchain si possono distinguere in pubbliche o *permissionless* e private o *permissioned*.

La parte progettuale di queste tesi è stata svolta seguendo un progetto reale in cui è stata sviluppata una piattaforma blockchain, per la gestione della supply chain di una grande azienda in ambito fashion, usando la piattaforma di blockchain *permissioned* Hyperledger Fabric. Le funzionalità principali per cui è stato scelto Hyperledger Fabric sono la presenza di canali per comunicazioni private e l'elevato throughput delle transazioni dovuto al paradigma *execute-order-validate*. Fabric è una piattaforma molto flessibile, che fornisce componenti *pluggable* per le diverse esigenze, come ad esempio il meccanismo di consenso o le policy di approvazione delle transazioni. Il suo design si basa su un *ledger* formato da due componenti principali: il *transaction log* e il *world state database*.

Per testare la piattaforma, è stata sviluppata una rete composta da due macchine fisiche sulle quali è stato fatto il *deploy* dei peer che contengono il codice Java del *chaincode*, cioè lo *smart contract* di Fabric. I peer appartengono a due organizzazioni: la prima si occupa di creare il design di un eyewear e la seconda collega al design fornito il codice della lente da utilizzare per la produzione. Le richieste di transazioni sono possibili tramite il canale *mychannel* che permette l'accesso al *chaincode* da parte delle

applicazioni delle due organizzazioni. Come meccanismo di consenso è stato utilizzato il Solo Orderer e come database per il world state LevelDB.

Per fornire un'interfaccia grafica al sistema, è stata sviluppata un'applicazione web di test basata su jsp: ogni organizzazione ha la propria pagina dedicata, dalla quale effettua le richieste per le transazioni di cui ha necessità. La connessione alla blockchain è possibile grazie al Java SDK, il quale fornisce le API di alto livello per richiamare il *chaincode*.

Questa tesi si conclude analizzando alcuni punti importanti che dovrebbero essere presi in considerazione in future attività di ricerca nel campo delle piattaforme blockchain e in particolare Hyperledger Fabric. Innanzitutto, è sicuramente da aspettarsi uno sviluppo molto forte nei prossimi anni nel campo delle *permissioned* blockchain in quanto l'interesse industriale si sta intensificando proprio in questa direzione perché ci sono necessità a livello *enterprise* che una rete *permissionless* non può garantire, quali controllo dei partecipanti e riservatezza delle informazioni. Considerando il caso specifico analizzato, la gestione della supply chain in uno scenario business-to-business con blockchain può portare grandi vantaggi nella direzione della trasparenza, dello sviluppo collaborativo e dell'accrescimento di fiducia tra le parti; inoltre, la tracciatura di ogni step di una catena produttiva può portare alla riduzione della contraffazione e dell'evasione. Grandi vantaggi saranno poi possibili anche in tanti campi in cui invece le blockchain pubbliche basate su token sono più adatte, come il nascente mercato dello scambio di energia P2P basato su smart contract.

Un altro punto su cui sicuramente ci sarà un avanzamento tecnologico è il throughput transazionale: benché le reti *permissionless* permettano prestazioni migliori di quelle *permissioned,* la velocità di un database tradizionale risulta molto maggiore. Per quanto riguarda le reti *permissionless*, c'è da aspettarsi la nascita di nuovi protocolli di consenso e la migrazione da parte delle principali piattaforme verso algoritmi meno *energy-consuming* (forse il 2020 sarà l'anno del passaggio di Ethereum da PoW a PoS).

È interessante a questo proposito l'analisi fatta da Gartner nel 2019 in cui si suddivide l'avanzamento evolutivo della tecnologia blockchain in quattro step, come si vede nella figura sottostante.

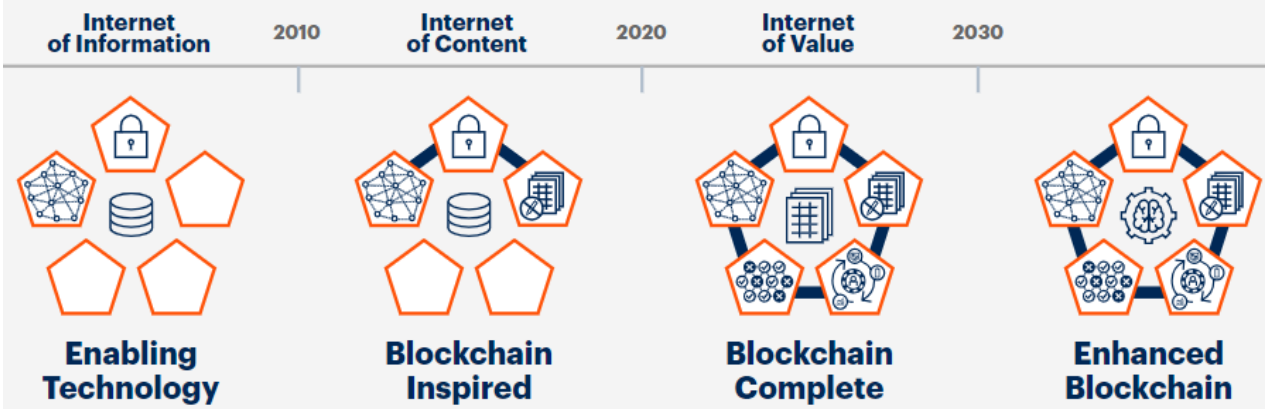



**Figura 14: Gartner evolutionary steps of blockchain.**

In accordo a questa analisi, siamo in una fase di passaggio da soluzioni *blockchain inspired* che utilizzano crittografia, distribuzione e immutabilità dei dati, a soluzioni blockchain complete che aggiungono anche la gestione degli asset tramite token e la completa decentralizzazione del sistema. Il prossimo step, che porterà alla maturità della tecnologia, sarà l'integrazione della blockchain con Internet of Things e Intelligenza Artificiale per abilitare nuovi scenari di business.

Per quanto riguarda Hyperledger Fabric, si sta assistendo al suo progressivo imporsi come standard industriale, almeno per il settore della supply chain. Le sue performance e la sua adattabilità alle esigenze degli utenti sono fondamentali e sicuramente saranno aggiunte altre feature *pluggable* come meccanismi di consenso *full byzantine fault tolerant.* Attualmente il supporto per i data source è limitato ai soli LevelDB e CouchDB, quindi potrebbe essere utile disporre di una scelta più ampia per poter ampliare le scelte a disposizione degli sviluppatori. Inoltre, una direzione di ricerca fondamentale dovrebbe essere quella dell'integrazione della blockchain con sistemi informativi già esistenti, in modo da poter sincronizzare dati ed effettuare analisi tramite API messe a disposizione degli utenti. Ai tre linguaggi di programmazione *general-purpose* (Go, Node.js e Java) supportati potrebbero affiancarsene anche altri, per allargare ancora di più la comunità di sviluppatori (magari Python). L'ultimo punto, forse quello più importante, riguarda la tokenizzazione: la mancanza di una valuta nativa per rappresentare asset è di certo un punto a sfavore di Hyperledger, che se non verrà risolto nelle prossime release potrebbe far perdere parte dell'interesse che questa piattaforma ha suscitato. La versione alpha 2.0 prevedeva la presenza dei FabToken, ma la funzionalità è stata ritirata perché era

stata riscontrata una falla nel suo funzionamento. La versione 2.0, rilasciata a febbraio 2020, non comprende i token, il cui inserimento in future release non è stato ancora confermato.

[1] “Fabric,” [Online]. Available: https://www.hyperledger.org/projects/fabric. [Accessed 20 11 2019].

[2] M. Bellini, “DLT,” 26 Settembre 2019. [Online]. Available: https://www.blockchain4innovation.it/tag/dlt/. [Accessed 20 11 2019].

[3] W. S. S. Stuart Haber, “How to Time-Stamp a Digital Document,” 1991. [Online]. Available: https://www.anf.es/pdf/Haber_Stornetta.pdf. [Accessed 20 11 2019].

[4] S. H. W. S. S. Dave Bayer, “Improving the Efficiency and Reliability of Digital Time-Stamping,” 1992. [Online]. Available: http://www.math.columbia.edu/~bayer/papers/Timestamp_BHS93.pdf. [Accessed 20 11 2019].

[5] S. Nakamoto, “bitcoin.pdf,” 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed 20 11 2019].

[6] R. Alyoshkin, “Blockchain 2.0. The Purpose of Blockchain,” 3 Ottobre 2017. [Online]. Available: https://medium.com/polys-blog/blockchain-2-0-the-purpose-of-blockchain-e84e5a95cdd9. [Accessed 20 11 2019].

[7] E. M. Rogers, Diffusion of Innovations, 1962.

[8] A. Alù, “Blockchain, le principali normative nazionali al mondo,” 19 Febbraio 2019. [Online]. Available: https://www.agendadigitale.eu/documenti/normativa-blockchain-le-principali-iniziative-nazionali/. [Accessed 20 11 2019].

[9] *Risoluzione del Parlamento europeo del 3 ottobre 2018 sulle tecnologie di registro distribuito e blockchain: creare fiducia attraverso la disintermediazione,* 2018.

[10] “Blockchain: lanciata call per esperti,” Roma, 2018.

[11] “Decreto semplificazioni: la legge di conversione in Gazzetta,” 11 Febbraio 2019. [Online]. Available: https://www.altalex.com/documents/leggi/2019/02/07/semplificazioni. [Accessed 20 11 2019].

[12] V. AA., “Satoshi Nakamoto,” [Online]. Available: https://it.bitcoin.it/wiki/Satoshi_Nakamoto. [Accessed 20 11 2019].

[13] P. Siriwardena, “The Mystery Behind Block Time,” 15 Maggio 2017. [Online]. Available: https://medium.facilelogin.com/the-mystery-behind-block-time-63351e35603a. [Accessed 20 11 2019].

[14] B. Curran, “What is a Merkle Tree? Beginner’s Guide to this Blockchain Component,” 9 Luglio 2018. [Online]. Available: https://blockonomi.com/merkle-tree/. [Accessed 20 11 2019].

[15] A. VV., “Ethereum Classic,” [Online]. Available: https://en.wikipedia.org/wiki/Ethereum_Classic. [Accessed 20 11 2019].

[16] A. Narayanan, “Analyzing the 2013 Bitcoin fork: centralized decision-making saved the day,” 28 Luglio 2015. [Online]. Available: https://freedom-to-tinker.com/2015/07/28/analyzing-the-2013-bitcoin-fork-centralized-decision-making-saved-the-day/. [Accessed 20 11 2019].

[17] “bigchaindb.com,” [Online]. Available: https://www.bigchaindb.com/. [Accessed 20 11 2019].

[18] “La Byzantine Fault Tolerance Spiegata,” 18 Novembre 2019. [Online]. Available: https://www.binance.vision/it/blockchain/byzantine-fault-tolerance-explained. [Accessed 20 11 2019].

[19] A. Collini, “Blockchain: modello generale e tassonomia delle componenti chiave,” Cesena, 2017.

[20] F. Provenzani, “Cos’è il Proof Of Work (PoW) e Proof Of Stake (PoS)?,” 26 Aprile 2019. [Online]. Available: https://www.money.it/Cos-e-la-Proof-Of-Work-PoW-e-Proof-of-Stake. [Accessed 20 11 2019].

[21] L. L. R. S. M. Pease, “The Byzantine Generals Problem,” Luglio 1992. [Online]. Available: https://www.microsoft.com/en-us/research/uploads/prod/2016/12/The-Byzantine-Generals-Problem.pdf. [Accessed 16 Gennaio 2020].

[22] “Come Bitcoin risolve il problema della doppia spesa,” 2018 Luglio 25. [Online].

Available: https://it.ihodl.com/tutorials/2018-07-25/bitcoin-doppia-spesa/. [Accessed 16 Gennaio 2020].

[23] “hashcash.org,” Novembre 2003. [Online]. Available: http://www.hashcash.org/. [Accessed 16 Gennaio 2020].

[24] “Blockchain Size,” [Online]. Available: https://www.blockchain.com/charts/blocks-size. [Accessed 16 Gennaio 2020].

[25] “Blockchain: cresce la conoscenza per il 20% degli italiani e l’85% delle imprese,” Aprile 2019. [Online]. Available: https://www.zerounoweb.it/software/blockchain/blockchain-cresce-la-conoscenza-per-il-20-degli-italiani-e-l85-delle-imprese/. [Accessed 16 Gennaio 2020].

[26] S. Stimolo, “Cina: la strategia blockchain e le risposte dal mondo crypto,” [Online]. Available: https://cryptonomist.ch/2019/11/23/cina-strategia-blockchain-mondo-crypto/. [Accessed 16 Gennaio 2020].

[27] “Blockchain - Membri del Gruppo di esperti,” 2018. [Online]. Available: https://www.mise.gov.it/index.php/it/10-istituzionale/ministero/2039024-blockchain-membri-del-gruppo-di-esperti. [Accessed 16 Gennaio 2020].

[28] S. E. M. S. J. Liebowitz, “Network Externalities (Effects),” [Online]. Available: https://personal.utdallas.edu/~liebowit/palgrave/network.html. [Accessed 23 gennaio 2020].

[29] V. Valsecchi, “La classificazione delle Blockchain: pubbliche, autorizzate e private,” 20 giugno 2018. [Online]. Available: https://www.spindox.it/it/blog/la-classificazione-delle-blockchain/. [Accessed 23 gennaio 2020].

[30] A. Back, “Hashcash - A Denial of Service Counter-Measure,” 1 agosto 2002. [Online]. Available: http://www.hashcash.org/papers/hashcash.pdf. [Accessed 23 gennaio 2020].

[31] VV.AA., “Proof of work,” [Online]. Available: https://en.bitcoin.it/wiki/Proof_of_work. [Accessed 23 gennaio 2020].

[32] “SHA-256,” [Online]. Available: https://en.bitcoin.it/wiki/SHA-256. [Accessed 23

gennaio 2020].

[33] N. Reiff, “What is Casper, the Latest Ethereum Upgrade?,” 25 giugno 2019. [Online]. Available: https://www.investopedia.com/news/what-casper-latest-ethereum-upgrade/. [Accessed 23 gennaio 2020].

[34] M. Porta, “Dapp: cosa sono e come funzionano le applicazioni decentralizzate,” 17 agosto 2019. [Online]. Available: https://cryptonomist.ch/2019/08/17/dapp-cosa-sono-come-funzionano/. [Accessed 24 gennaio 2020].

[35] V. Buterin, “A Next-Generation Smart Contract and Decentralized Application Platform,” 2013. [Online]. Available: https://whitepaper.io/document/5/ethereum-whitepaper. [Accessed 24 gennaio 2020].

[36] “torproject.org,” [Online]. Available: https://www.torproject.org/. [Accessed 24 gennaio 2020].

[37] “bittorrent.com,” [Online]. Available: https://www.bittorrent.com/. [Accessed 24 gennaio 2020].

[38] “ripple.com,” [Online]. Available: https://ripple.com/. [Accessed 29 gennaio 2020].

[39] D. O. a. J. Ousterhout, “In Search of an Understandable Consensus Algorithm,” 2014. [Online]. Available: https://raft.github.io/raft.pdf. [Accessed 30 gennaio 2020].

[40] [Online]. Available: https://github.com/bft-smart/library. [Accessed 30 gennaio 2020].

[41] “goquorum.com,” [Online]. Available: https://www.goquorum.com/. [Accessed 30 gennaio 2020].

[42] “IBFT Consensus,” [Online]. Available: https://whitepaper.ledgerium.io/architecture-blockchain/ibft. [Accessed 30 gennaio 2020].

[43] “Ethereum Validation Clique PoA,” [Online]. Available: https://github.com/clearmatics/ion/wiki/Ethereum-Validation---Clique-PoA. [Accessed 30 gennaio 2020].

[44] [Online]. Available: https://github.com/corda/corda. [Accessed 30 gennaio 2020].

[45] M. Bellini, “La Blockchain come garanzia della filiera alimentare,” 15 gennaio 2017.

[Online]. Available: https://www.blockchain4innovation.it/iot/la-blockchain-come-garanzia-della-filiera-alimentare/. [Accessed 4 febbraio 2020].

[46] C. M. Ingo Wilhelm, “blockchain-automotive.html,” 11 novembre 2019. [Online]. Available: https://www.bmw.com/it/innovation/blockchain-automotive.html. [Accessed 4 febbraio 2020].

[47] “Spinosa, mozzarella Dop ad alta tecnologia,” 23 giugno 2019. [Online]. Available: https://distribuzionemoderna.info/intervista/spinosa-mozzarella-dop-ad-alta-tecnologia-1. [Accessed 4 febbraio 2020].

[48] M. Bonini, “Un modello di Blockchain per la tracciabilità dei prodotti alimentari, basata su Ethereum,” *Sanità Pubblica Veterinaria,* no. 13, 2019.

[49] M. Macagno, Evaluation of Blockchain Technologies for Vehicular Applications, Torino, 2018.

[50] M. Sarvepalli, “Hyperledger Fabric 1.4 on Multiple Hosts using Docker Swarm and Compose,” 7 settembre 2019. [Online]. Available: https://medium.com/@malliksarvepalli/hyperledger-fabric-1-4-on-multiple-hosts-using-docker-swarm-and-compose-ec668db0bad5. [Accessed 05 febbraio 2020].

[51] “Installazione, istanziazione e aggiornamento di un chaincode,” 31 maggio 2019. [Online]. Available: https://cloud.ibm.com/docs/services/blockchain?topic=blockchain-install-instantiate-chaincode&locale=it. [Accessed 07 gennaio 2020].

[52] “Hyperledger Fabric Account-based Wallet Java Chaincode,” 9 febbraio 2019. [Online]. Available: https://medium.com/coinmonks/hyperledger-fabric-account-based-wallet-java-chaincode-8cbf80a6fb82. [Accessed 07 febbraio 2020].

[53] C. Ferranti, “Blockchain & Internet of Value,” [Online]. Available: https://magazine.euclidea.com/blockchain-internet-of-value. [Accessed 22 febbraio 2020].