UNIVERSITÀ DEGLI STUDI DI CAMERINO

CORSO DI LAUREA TRIENNALE IN INFORMATICA [CORSO L-31]

ANNO ACCADEMICO 2021/2022

PROGETTO:

HYPERLEDGER FABRIC: FABNET 2.0

CANDIDATO:

COLUCCI MATTEO [Mat. 106597]

RELATORE

ANDREA MORICHETTA

CORRELATORE

ALESSANDRO MARCELLETTI

INTRODUZIONE **2**
SCOPO DEL PROGETTO 2
TECNOLOGIE UTILIZZATE **3**
Kotlin 3
Springboot 3
Angular 3
Docker 4
BACKGROUND **5**
Blockchain 5
Struttura 5
Immutabilità 5
Decentralizzazione e trasparenza 5
Consenso 6
Prime applicazioni 6
Hyperledger Fabric **7**
Architettura 7
Com'è fatta una rete Fabric? 8
Canali 8
Ledger 8
Chaincode 8
Il servizio di ordinamento 9
Identità: Certificate Authority e Membership Service Provider 9
IMPLEMENTAZIONE **10**
Enrollment CA 11
TLS CA 11
Registering and Enrolling tramite CA 11
Orderer/Peer 12
Struttura dopo la registrazione 14
Creazioni canali 14
DEMO **16**
1. Network 16
2. Organizzazione 17
3. Consortium 18
4. Channel 18
5. Download network 18
STRUTTURA ZIP **19**
Creazione della rete 21
CONCLUSIONI 23

INTRODUZIONE

Oggi giorno un tema molto discusso è quello delle Blockchain.
La prima blockchain nasce nel 2008 ad opera di **Satoshi Nakamoto**, implementata l'anno seguente, con l'obiettivo di fungere da "libro mastro" della nascente valuta digitale **Bitcoin**.
Negli anni a seguire abbiamo assistito alla crescita smisurata della criptovaluta che ha portato Bitcoin ad affermarsi al vertice sul mercato delle blockchain.
Fino a poco tempo fa Bitcoin godeva di una pessima fama infatti il grande pubblico la associava al "dark web" e quindi ad affari illegali.
Ultimamente grazie all'adozione da parte di **Istituzioni** anche governative Bitcoin sta iniziando a acquisire sempre maggior fiducia. Ma Bitcoin è solo una applicazione che sfrutta Blockchain!
La vera rivoluzione, per molti di pari portata dell'avvento di internet, è Blockchain.
Infatti, si sente sempre più parlare di blockchain, inizialmente l'attenzione e la conoscenza verso questo mondo riguardava prevalentemente gli sviluppatori, oggi la blockchain fa sempre più parte delle nostre vite, poiché capace di dare risposte nuove a tanti e diversi bisogni d'impresa, organizzazioni, cittadini e consumatori.
La blockchain che verrà analizzata durante il progetto sarà la blockchain Hyperledger Fabric.
Hyperledger Fabric è un progetto open source della Linux Foundation.
Progettato come base per sviluppare applicazioni di livello aziendale e soluzioni di settore.
Un esempio dell'utilizzo della blockchain Hyperledger Fabric è "**IBM FOOD TRUST**", sviluppata insieme a Walmart, gigante statunitense della grande distribuzione alimentare.
Food Trust si occupa di fornire una fonte affidabile di informazioni e tracciabilità per migliorare la trasparenza e l'efficienza nella rete alimentare.
Reso possibile attraverso la condivisione di un registro per l'archiviazione della documentazione di **conformita digitale** e della rete dei **certificati audit**. (L'audit di rete è la misura collettiva effettuata per analizzare, studiare e raccogliere dati su una rete allo scopo di accertarne lo stato di salute in conformità con i requisiti della rete/organizzazione.)
In primis aumenta **l'affidabilità** sulla provenienza del cibo grazie all'accesso istantaneo ai dati di tracciabilità end-to-end per verificare la cronologia nella rete alimentare e nella catena di approvvigionamento ma soprattutto ridurrebbe anche il numero di decessi che ogni anno avvengono per malattie di origine alimentare.
Hyperledger Fabric pur mettendo a disposizione degli applicativi per realizzare una rete, non predispone di un tool che si occupi di generare e configurare a partire dalle entità, una rete.
Nelle varie community di Fabric, ricevere supporto per la configurazione della rete è stato a tratti impossibile, poiché durante la configurazione di una rete entrano in gioco l'utilizzo di diverse tecnologie come ad esempio l'uso dei container docker, con le relative dipendenze e variabili ambientali che a loro volta potrebbero causare errori in background.
Essendo Fabric una tecnologia ancora in sviluppo il supporto alla creazione della rete è ancora molto debole.
A questo problema si è cercato di porre rimedio sviluppando il wizard FabNet.

SCOPO DEL PROGETTO

Lo scopo del progetto è lo studio del framework Hyperledger Fabric, con focus sul setup di una rete di produzione. Lo studio ha coinvolto anche l'implementazione della rete di produzione partendo da un wizard preesistente.

Il wizard in questione è chiamato FabNet. Poiché Fabric non predispone un tool che si occupi di generare e configurare una rete a partire dalle entità, il wizard FabNet permette di configurare i componenti minimi per generare una "First-network", ovvero una semplice rete privata.

La prima fase è stata lo studio della tecnologia permissioned Hyperledger Fabric, successivamente si è proseguito con lo studio della struttura del wizard, dei vari framework utilizzati e della rete che veniva generata dal wizard.

La rete generata da FabNet risultava obsoleta per la versione 2.2.x e quindi non compatibile con lo sviluppo di una rete di produzione. E' stato dunque necessario l'adattamento del wizard FabNet per tal fine.

TECNOLOGIE UTILIZZATE

Kotlin

Kotlin è un linguaggio di programmazione general purpose, multi-paradigma, open source sviluppato dall'azienda di software JetBrains.

Kotlin è strutturato per interoperare con la piattaforma Java Runtime Environment come target principale, il che garantisce il funzionamento delle applicazioni in ogni ambiente che accetti la JVM, ivi compreso Android, ma il compilatore è in grado anche di emettere codice JavaScript. È inoltre presente la possibilità di compilare il linguaggio Kotlin direttamente in linguaggio macchina tramite il compilatore Kotlin per l'ambiente di riferimento.

Springboot

Spring Boot è un framework open-souce per lo sviluppo di applicazioni web basato su Java. È un'estensione di Spring e permette di raggruppare tanti framework in un unico

pacchetto pronto all'uso consentendo al programmatore di gestire in modo veloce ed efficiente autenticazione, persistenza dei dati e server web.

Per fornire un servizio restful, Spring Boot necessità dell'annotazione @RestController sulla classe controller, in modo da comunicare che la essa conterrà i metodi usati come endpoint.
I metodi della classe controller saranno i metodi per ottenere ed inviare dati che dovranno necessariamente avere l'annotazione @RequestMapping("/index") contenente il path al quale l'endpoint fa riferimento.

Angular

Angular è uno dei framework **Typescript** open-source più popolari disponibili sul mercato. Esso permette di creare applicazioni web e mobile.

È stato sviluppato da Google e viene spesso utilizzato per lo sviluppo di applicazioni web lato client.

Grazie a una serie di funzionalità e strumenti, Angular semplifica lo sviluppo delle applicazioni, garantendo ottimi risultati in termini di prestazioni.

Esso infatti permette la creazione di applicazioni dinamiche, garantendo all'utente fruitore dell'applicazione un'esperienza d'uso di alto livello.

Angular usa il pattern Model-View-Controller (MVC) che permette la separazione di codice tra i vari componenti creati. I componenti sono moduli riutilizzabili che comprendono una parte grafica (file html) ed una parte logica (file typescript).

Tutte le applicazioni Angular di default hanno un component predefinito (root component) nel quale saranno poi referenziati altri component.

Angular permette inoltre la possibilità di definire nuove direttive in modo semplice e veloce. Spesso integrato come **frontend** nelle applicazioni server-side di tipo restful, integra delle api di facile utilizzo per la manipolazione delle richieste **HTTP**.

Docker

Docker è un progetto open-source nato con lo scopo di automatizzare la distribuzione di applicazioni sotto forma di ”**container**” leggeri, portatili e autosufficienti che possono essere eseguiti su di un cloud (pubblico o privato) o in locale.

Questi container sono **autosufficienti** perché contengono già tutte le dipendenze dell’applicazione, sono portabili perché sono distribuiti in un formato standard leggibile da qualsiasi server Docker.

I container sono inoltre leggeri perché sfruttano i servizi offerti dal kernel del sistema operativo ospitante, invece di richiedere l’esecuzione di un kernel virtualizzato come avviene per la maggior parte delle Virtual Machine.

Docker implementa **API** di alto livello per la gestione dei container eseguendo i processi in ambienti isolati.

Creare e gestire container tramite Docker è di grande necessità durante la creazione di sistemi distribuiti. Esso infatti permette a diverse applicazioni o processi di lavorare autonomamente sulla stessa macchina fisica o su differenti macchine virtuali.

In Docker è possibile la definizione e l’esecuzione di più container insieme, adoperando il tool ”Compose”.

Docker Compose per l’appunto, permette allo sviluppatore di raggruppare un numero arbitrario di container modellando un file di tipo **YAML** che permetterà la loro configurazione, l’avvio e la loro interruzione in modo simultaneo.

BACKGROUND

Blockchain

Struttura

La blockchain, come la parola stessa comunica, si può pensare semplicemente come una **catena di blocchi**, consequenziali e incatenati.

Ogni blocco è costituito da un informazione o meglio un dato. Ogni blocco ha una capienza massima, una volta che il blocco raggiunge la capienza massima, il blocco viene crittograficamente sigillato tramite la **funzione di hash**, essa permette di trasformare qualsiasi tipo di dato in una stringa alfanumerica, in grado di riassumere tutte le informazioni nel blocco.

La funzione di hash ha un requisito fondamentale, è **univoca**, ovvero posso trasformare qualsiasi informazione in una stringa alfanumerica, univocamente, ma tramite quella stringa non potrò mai risalire alle informazioni che lo hanno generato.

Immutabilità

Al momento della creazione del blocco successivo, la primissima informazione che il nuovo blocco acquisisce è l'hash del blocco precedente, ogni blocco contiene il **riassunto criptografico** del blocco precedente.

Il codice rimane univoco, fin quando le informazioni nel blocco rimangono inalterate, nel caso in cui, l'informazione contenuta in quel blocco dovesse essere alterata, il codice univoco cambia e si ottiene il cosiddetto "Butterfly effect", andando ad alterare il codice tutti i blocchi a esso collegati.

Questo meccanismo ci permette di capire se nella blockchain c'è un **intrusione** e quindi il sistema estrometterebbe questa blockchain.

Decentralizzazione e trasparenza

Questo meccanismo è possibile grazie ad un'altra caratteristica fondamentale delle blockchain, essa è condivisa e decentralizzata.

Ovvero, la blockchain non è presente su un computer, non esiste da una parte, ma è una copia infinita della catena di blocchi, chiunque può avere una copia della blockchain sul proprio pc.

E nel caso venga alterata, non viene fatto alcun danno alla blockchain, poiché è solo una copia e inoltre il sistema è capace di riconoscere la copia alterata che a sua volta non sarà più valida.

Consenso

Il sistema però a sua volta ha bisogno dei "**mining**", ovvero dei controllori, connessi alla rete, che verificano i nuovi blocchi e autorizzano le transazioni, per permettere il funzionamento del

sistema.
Per ottenere ciò, il sistema deve incentivare i mining offrendo una paga direttamente proporzionale alla potenza computazionale che i mining mettono a disposizione.
Ad esempio con Bitcoin viene utilizzata la cripto bitcoin come pagamento.

Prime applicazioni

La prima applicazione riconosciuta della blockchain è la criptovaluta **Bitcoin**, anche se altri ne hanno seguito le orme.

Ethereum, una criptovaluta alternativa, ha adottato un approccio diverso, integrando molte delle stesse caratteristiche di Bitcoin ma aggiungendo smart contract per creare una piattaforma per applicazioni distribuite.

Uno **smart contract** è un programma che consente di automatizzare delle operazioni al verificarsi di determinate condizioni. Il meccanismo con cui avvengono i pagamenti nella blockchain può essere considerato uno smart contract.

Bitcoin ed Ethereum rientrano in una classe di blockchain che classificheremmo come tecnologia **blockchain pubblica senza autorizzazione**.

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.

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.

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 s**ervizio 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.

Arichitettura

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.
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.

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.

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**.
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.

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.

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.

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.

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, ognuno dei tre linguaggi sopracitati è pienamente supportato da numerose API e dalla community di sviluppatori.

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.

In Fabric anche l'ordinamento è un servizio pluggable 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.
- 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.

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.

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.
Inoltre l’MSP trasforma un’identità in un ruolo identificando specifici privilegi che un attore ha su un nodo o un canale.

IMPLEMENTAZIONE

La fase implementativa è partita con lo studio delle tecnologie analizzate.

In primis, sono partito con il download dei binari, sample e immagini che fabric richiede, successivamente è stato testato la corretta installazione dei componenti attraverso gli esempi che Fabric mette a disposizione.

Al primo avvio della rete generata con il wizard FabNet, è stato segnalato il seguente errore:

```
2022-01-10 11:43:50.013 CET 0001 ERRO [main] InitCmd -> Cannot run peer because error when setting up MSP of type bccsp from directory /home/matteo/Docume
nti/prova/test-network/orgs/mychannel/../org1.example.com/entities/org1admin/msp: Setup error: nil conf reference
2022-01-10 11:43:50.060 CET 0001 ERRO [main] InitCmd -> Cannot run peer because error when setting up MSP of type bccsp from directory /home/matteo/Docume
nti/prova/test-network/orgs/mychannel/../org2.example.com/entities/org2admin/msp: Setup error: nil conf reference
2022-01-10 11:43:50.108 CET 0001 ERRO [main] InitCmd -> Cannot run peer because error when setting up MSP of type bccsp from directory /home/matteo/Docume
nti/prova/test-network/orgs/mychannel/../org1.example.com/entities/org1admin/msp: Setup error: nil conf reference
2022-01-10 11:43:50.153 CET 0001 ERRO [main] InitCmd -> Cannot run peer because error when setting up MSP of type bccsp from directory /home/matteo/Docume
nti/prova/test-network/orgs/mychannel/../org2.example.com/entities/org2admin/msp: Setup error: nil conf reference
```

Tale errore segnala la mancanza di certificati/directory necessari per il corretto funzionamento della rete.
Per questo motivo ho avuto la necessità di riconfigurare l’intera rete, attraverso la documentazione fornita da Fabric, per la rete di produzione.
Si è partiti dalla configurazione e dalla registrazione delle Certification Authority (CA), per poi proseguire con la registrazione delle entità tramite le rispettive CA.
Successivamente sono stati corretti sia gli errori relativi ai container e sia gli errori generati dalle strutture limitate di peer e orderer.
Per poi infine correggere gli errori relativi alla creazione e all'aggiunta di un'entità al canale.

Enrollment CA

Il primo componente ad essere sviluppato in una rete Fabric è il CA. I certificati associati a un nodo devono essere creati prima che il nodo stesso possa essere distribuito.
I certificati utilizzati dalla rete sono generati con il Certificate Authority proprietario di Fabric.

Il Fabric CA oltre a generare i certificati per eventuali utenti aggiuntivi, crea anche le strutture MSP necessarie per definire correttamente i componenti e le organizzazioni.

TLS CA

Essa genera i certificati utilizzati per proteggere le comunicazioni su Transport Layer Security (TLS).
Poiché è necessario specificare che la rete utilizzerà TLS prima di distribuire la CA di "registrazione" (il file YAML che specifica la configurazione di questa CA ha un campo per abilitare TLS), è necessario distribuire prima la CA TLS e utilizzare il suo certificato radice durante il bootstrap della CA di registrazione.
Questo certificato TLS verrà utilizzato anche dal client fabric-ca durante la connessione alla CA di registrazione per registrare le identità per utenti e nodi.

Il Fabric CA è un server istanziabile utilizzando un container Docker con l'immagine **hyperledger/fabrica-ca.**

Il file Docker permette di personalizzare il numero di porta su cui verrà istanziato il server e il path del file di configurazione fabric-ca-server-config.yaml che conterra le personalizzazioni aggiuntive che possono essere eseguite.

Alla precedente configurazione di fabric-ca-server.yaml è stata aggiunta la parte relativa al bccps, in questa sezione si controlla dove verrà archiviata la chiave privata per le CA.

Registering and Enrolling tramite CA

Un'entità viene prima registrata con una CA da un amministratore della CA.

fabric-ca-client register --id.name org1admin --id.secret org1adminpw --caname ca-org1 --id.type admin --tls.certfiles ../fabric-ca-server/tls-cert.pem

id.name: L'ID di registrazione dell'identità
id.secret: password
id.type: Tipo (admin, client, peer o orderer)
tls.certfiles: Il percorso relativo al certificato firmato radice della CA TLS (generato durante la creazione della CA TLS

Una volta che la CA di registrazione è stata impostata e le identità sono state registrate, l'amministratore della CA dovrà contattare l'utente che si iscriverà fuori banda per fornire loro l'ID di registrazione e password che hanno utilizzato durante la registrazione dell'identità.
Utilizzando questo ID e password, l'utente può registrare l'identità utilizzando la propria copia del client Fabric CA per contattare la CA pertinente.
Se il TLS è stato abilitato, questo utente dovrà acquisire il certificato firmato radice della CA TLS da includere durante la registrazione.

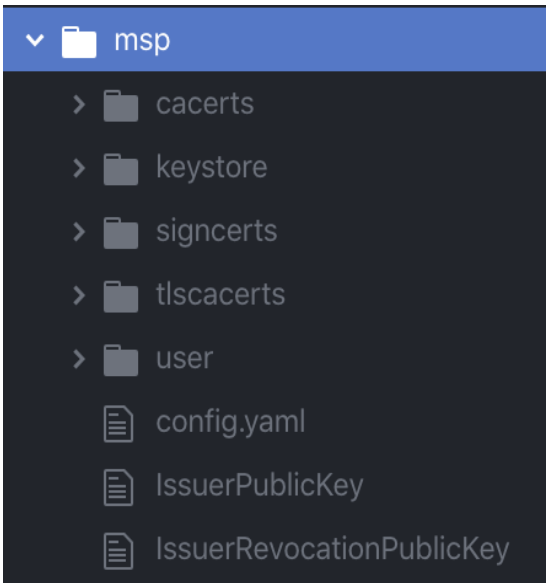
fabric-ca-client enroll -u https://org1admin:org1adminpw@localhost:7054 -M ../entities/org1admin/msp --tls.certfiles ../fabric-ca-server/tls-cert.pem

ca_url: Url del CA che include la porta

fabric-ca-client enroll -u https://<ENROLL_ID>:<ENROLL_SECRET><@CA_URL>:<PORT>

msp_folder: il percorso relativo all'msp
tls.certfile: Il percorso relativo al certificato firmato radice della CA TLS della CA TLS associata a questa organizzazione.

La figura in basso mostra le sottocartelle restituite da una registrazione avvenuta correttamente:



cacerts: contiene il certificato radice della CA dell'organizzazione in cui è stata registrata 'identità dell'amministratore.

keystore: contiene la chiave privata del nodo

signcerts: contiene la chiave pubblica del nodo

tlscacerts: contiene il certificato radice della CA TLS che ha emesso certificati alle CA o ai nodi associati a questa organizzazione.

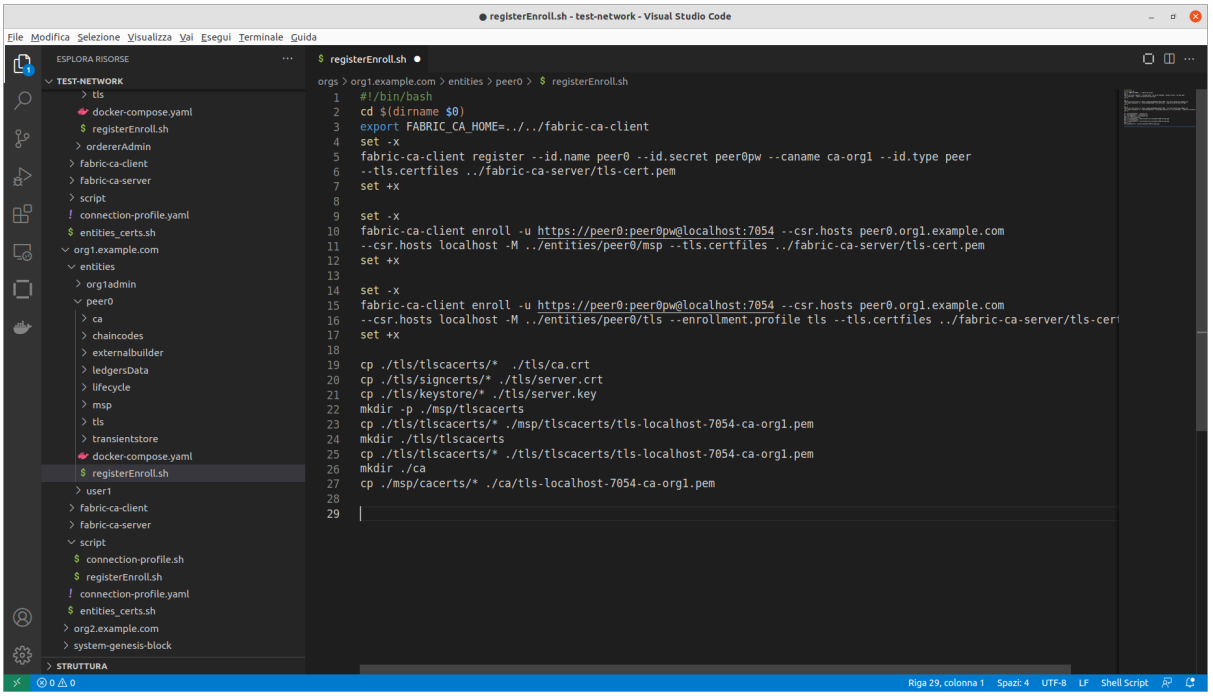
Orderer/Peer

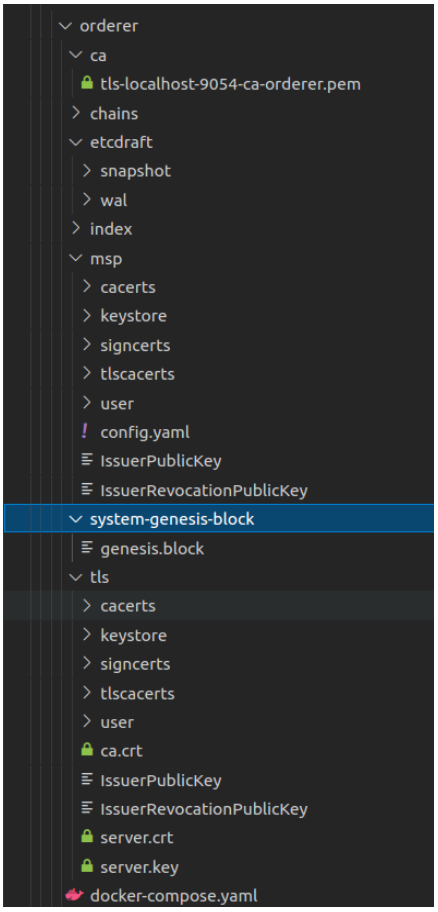
Per creare peer e orderer, dopo aver ottenuto i loro certificati è necessario definire ulteriori file docker-compose.yaml ciascuno relativo alle entità peer e orderer delle organizzazioni.

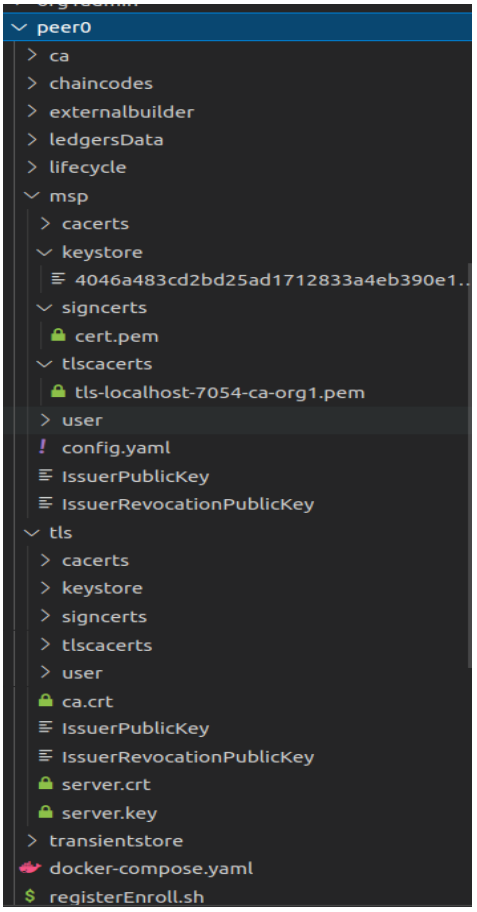
Essi sono istanziati come container ciascuno con la sua rispettiva immagine:

Hyperledger/fabric-peer
Hyperledger/fabric-orderer.

Orderer e peer in fase di registrazione hanno bisogno di determinati certificati e percorsi che potranno essere generati o copiati come nell'esempio sotto riportato.


registerEnroll.sh - test-network - Visual Studio Code
File Modifica Selezione Visualizza Vai Esegui Terminale Guida
ESPLORA RISORSE
TEST-NETWORK
tls
docker-compose.yaml
registerEnroll.sh
ordererAdmin
fabric-ca-client
fabric-ca-server
script
connection-profile.yaml
entities_certs.sh
org1.example.com
entities
org1admin
peer0
ca
chaincodes
externalbuilder
ledgersData
lifecycle
msp
tls
transientstore
docker-compose.yaml
registerEnroll.sh
user1
fabric-ca-client
fabric-ca-server
script
connection-profile.sh
registerEnroll.sh
connection-profile.yaml
entities_certs.sh
org2.example.com
system-genesis-block
STRUTTURA
registerEnroll.sh
orgs > org1.example.com > entities > peer0 > registerEnroll.sh
#!/bin/bash
cd $(dirname $0)
export FABRIC_CA_HOME=../../fabric-ca-client
set -x
fabric-ca-client register --id.name peer0 --id.secret peer0pw --caname ca-org1 --id.type peer
--tls.certfiles ../fabric-ca-server/tls-cert.pem
set +x
set -x
fabric-ca-client enroll -u https://peer0:peer0pw@localhost:7054 --csr.hosts peer0.org1.example.com
--csr.hosts localhost -M ../entities/peer0/msp --tls.certfiles ../fabric-ca-server/tls-cert.pem
set +x
set -x
fabric-ca-client enroll -u https://peer0:peer0pw@localhost:7054 --csr.hosts peer0.org1.example.com
--csr.hosts localhost -M ../entities/peer0/tls --enrollment.profile tls --tls.certfiles ../fabric-ca-server/tls-cert
set +x
cp ./tls/tlscacerts/* ./tls/ca.crt
cp ./tls/signcerts/* ./tls/server.crt
cp ./tls/keystore/* ./tls/server.key
mkdir -p ./msp/tlscacerts
cp ./tls/tlscacerts/* ./msp/tlscacerts/tls-localhost-7054-ca-org1.pem
mkdir ./tls/tlscacerts
cp ./tls/tlscacerts/* ./tls/tlscacerts/tls-localhost-7054-ca-org1.pem
mkdir ./ca
cp ./msp/cacerts/* ./ca/tls-localhost-7054-ca-org1.pem
Riga 29, colonna 1
Spazi: 4
UTF-8
LF
Shell Script


orderer
ca
tls-localhost-9054-ca-orderer.pem
chains
etcdraft
snapshot
wal
index
msp
cacerts
keystore
signcerts
tlscacerts
user
config.yaml
IssuerPublicKey
IssuerRevocationPublicKey
system-genesis-block
genesis.block
tls
cacerts
keystore
signcerts
tlscacerts
user
ca.crt
IssuerPublicKey
IssuerRevocationPublicKey
server.crt
server.key
docker-compose.yaml


peer0
ca
chaincodes
externalbuilder
ledgersData
lifecycle
msp
cacerts
keystore
4046a483cd2bd25ad1712833a4eb390e1..
signcerts
cert.pem
tlscacerts
tls-localhost-7054-ca-org1.pem
user
config.yaml
IssuerPublicKey
IssuerRevocationPublicKey
tls
cacerts
keystore
signcerts
tlscacerts
user
ca.crt
IssuerPublicKey
IssuerRevocationPublicKey
server.crt
server.key
transientstore
docker-compose.yaml
registerEnroll.sh

Struttura dopo la registrazione

Nella cartella orderer si può notare la cartella **etcdraft**, indica il **meccanismo di consenso utilizzato dal orderer**, è basato su un sistema leader-follower in cui un leader su un canale decide l'ordinamento e i follower seguono il suo comportamento.

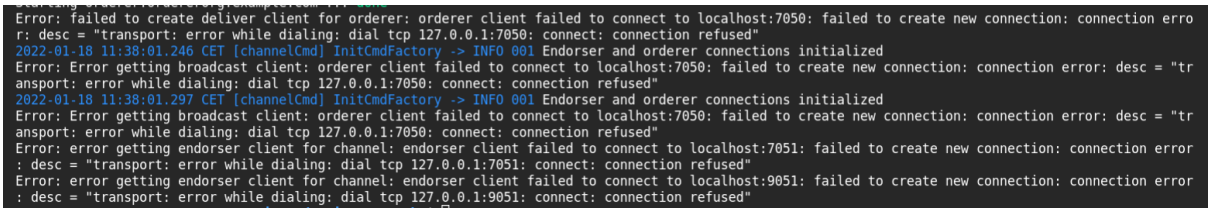
(Nelle versioni successive a Fabric 2.2 i meccanismi di consenso **Kaftka e Solo** potrebbero essere deprecati.)

Nella cartella del peer invece si possono notare le cartelle relative ai chaincodes, lifecycle, externalBuiler.

Chaincodes: utilizzato per il supporto degli smart contract;
Lifecycle: rappresenta il ciclo di vita del chaincode di Fabric è un processo che consente a più organizzazioni di concordare come verrà gestito un chaincode prima che possa essere utilizzato su un canale.
ExternalBuilder: Rappresenta un elenco di directory da considerare come builder e lanciatori esterni chaincode. L'elaborazione del rilevamento del generatore esterno eseguirà un'iterazione su costruttori nell'ordine specificato di seguito.

Inoltre è possibile notare le directory ***ledgersData,*** per il supporto del ledger.

Per il deploy del peer e del orderer in una rete di produzione, occorrerà configurare i rispettivi file **core.yaml** (peer) **orderer.yaml** (orderer).



Nella seconda fase dello sviluppo dopo aver generato i percorsi e certificati necessari per la registrazione dell'identità nella rete e dopo aver creato i container per i relativi peer e orderer, si è verificato un errore nella connessione e nell'aggiunta al canale.

Creazioni canali

Per la creazione dei canali sulla rete il primo step è quello di definire un file chiamato **configtx.yaml** nel quale viene descritta l'architettura della rete che verrà definita.
Il file conterrà la lista delle organizzazioni, la lista dei consorzi, la lista dei canali e le politiche interne.
Successivamente è necessario creare il blocco genesi a partire da questo file tramite il tool **configtxgen**.
Il **blocco genesi** contiene tutta l'architettura di rete e deve essere inviato agli orderer prima del loro avvio.
Dopo aver creato e distribuito agli orderer il blocco genesi, la creazione dei canali descritti nel

file è di competenza delle entità admin di ogni organizzazione.
Il canale deve essere creato da uno degli admin delle organizzazioni che sono indicate nel consorzio amministratore dello stesso.

mychannel > $ create.sh

```
#!/bin/bash
cd $(dirname $0)
export CORE_PEER_TLS_ENABLED=true
export FABRIC_CFG_PATH=$PWD/../../
export CORE_PEER_LOCALMSPID="org1-example-com"
export CORE_PEER_MSPCONFIGPATH="$PWD/../org1.example.com/entities/org1admin/msp"
peer channel create -o localhost:7050 -c mychannel --ordererTLSHostnameOverride orderer.ordererOrg.example.com
-f ./mychannel.tx --outputBlock ./mychannel.block
--tls --cafile $PWD/../ordererOrg.example.com/entities/orderer/tls/ca.crt
#### Adding org1.example.com anchor peers definition to channel block
export CORE_PEER_LOCALMSPID="org1-example-com"
export CORE_PEER_MSPCONFIGPATH="$PWD/../org1.example.com/entities/org1admin/msp"
peer channel update -o localhost:7050 -c mychannel --ordererTLSHostnameOverride orderer.ordererOrg.example.com
-f ./org1.example.com-anchor.tx --tls
--cafile $PWD/../ordererOrg.example.com/entities/orderer/tls/ca.crt
#### Adding org2.example.com anchor peers definition to channel block
export CORE_PEER_LOCALMSPID="org2-example-com"
export CORE_PEER_MSPCONFIGPATH="$PWD/../org2.example.com/entities/org2admin/msp"
peer channel update -o localhost:7050 -c mychannel
--ordererTLSHostnameOverride orderer.ordererOrg.example.com
-f ./org2.example.com-anchor.tx --tls --cafile $PWD/../ordererOrg.example.com/entities/orderer/tls/ca.crt
```

> mychannel > $ join.sh

```
#!/bin/bash
cd $(dirname $0)
#### Joining peers org1.example.com
export CORE_PEER_TLS_ENABLED=true
export FABRIC_CFG_PATH=$PWD/../../
export CORE_PEER_LOCALMSPID="org1-example-com"
export CORE_PEER_MSPCONFIGPATH="$PWD/../org1.example.com/entities/org1admin/msp"
export CORE_PEER_TLS_ROOTCERT_FILE="$PWD/../org1.example.com/entities/peer0/tls/ca.crt"
export CORE_PEER_ADDRESS=localhost:7051
peer channel join -b ./mychannel.block
# -------------------------------------------
#### Joining peers org2.example.com
export CORE_PEER_TLS_ENABLED=true
export FABRIC_CFG_PATH=$PWD/../../
export CORE_PEER_LOCALMSPID="org2-example-com"
export CORE_PEER_MSPCONFIGPATH="$PWD/../org2.example.com/entities/org2admin/msp"
export CORE_PEER_TLS_ROOTCERT_FILE="$PWD/../org2.example.com/entities/peer0/tls/ca.crt"
export CORE_PEER_ADDRESS=localhost:9051
peer channel join -b ./mychannel.block
# -------------------------------------------
```

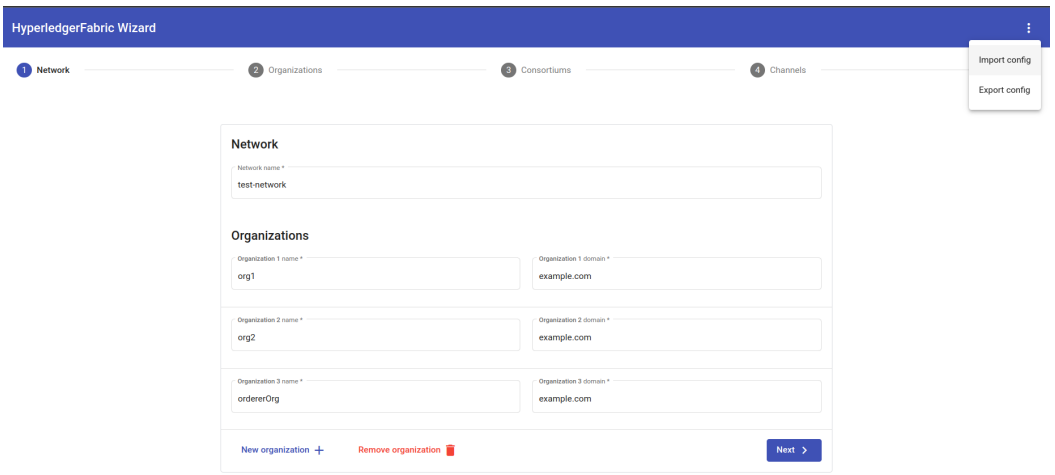
create.sh e join.sh, durante la configurazione di questi script è fondamentale settare correttamente le rispettive variabili d'ambiente.

Demo

Il wizard è stato realizzato in due moduli separati, un frontend scritto in **TypeScript** con framework **Angular** e un backend scritto in **Kotlin** con framework Spring.
Il wizard è composto da cinque interfacce relative alle seguenti funzionalità:

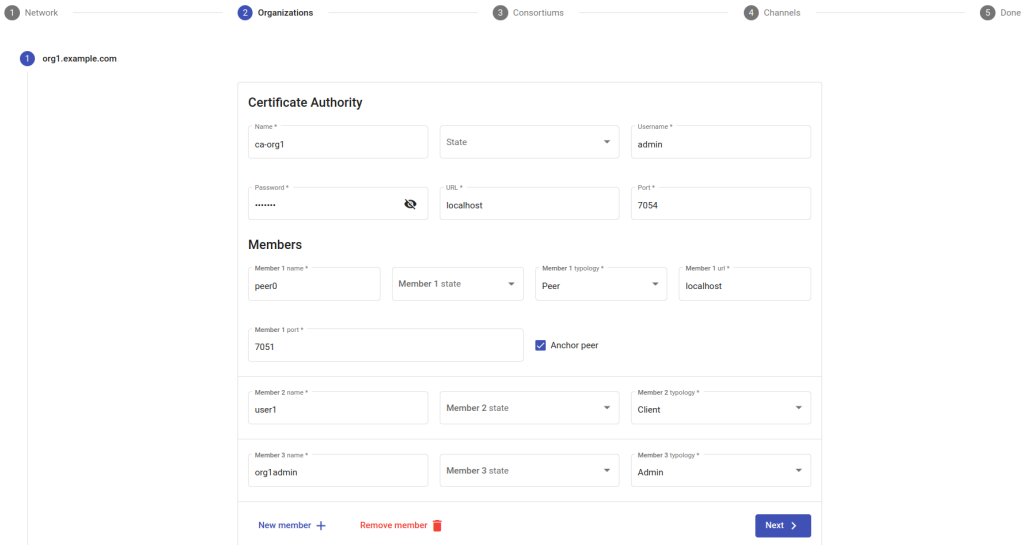
1. Network
2. Organizations
3. Consortium
4. Channel
5. Done.

In questa sezione verrà mostrata in dettaglio il wizard realizzato e la costruzione di una rete step by step.



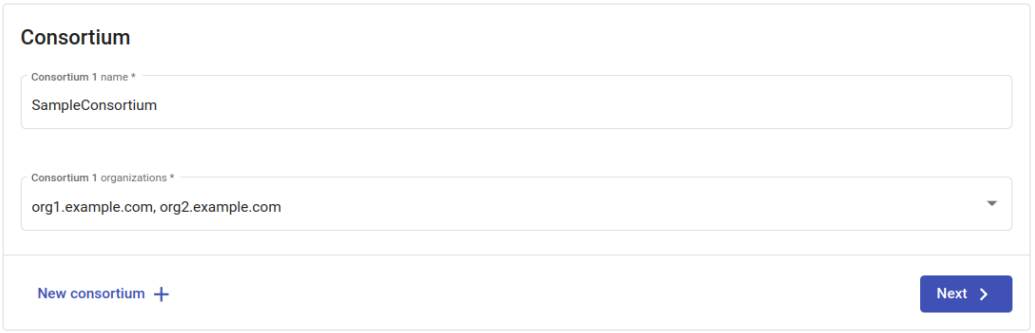
1. Network

Nel primo step è necessario inserire il nome della rete, il numero e il nome delle organizzazioni che la comporranno.
È fondamentale che la coppia nome e dominio identifica l'organizzazione univocamente sulla rete.
È possibile importare o esportare una configurazione:
È disponibile un file "test-network.json", sul sito https://pros.unicam.it/fabnet/, con una configurazione semplice, permette di testare la rete.



2. Organizzazione

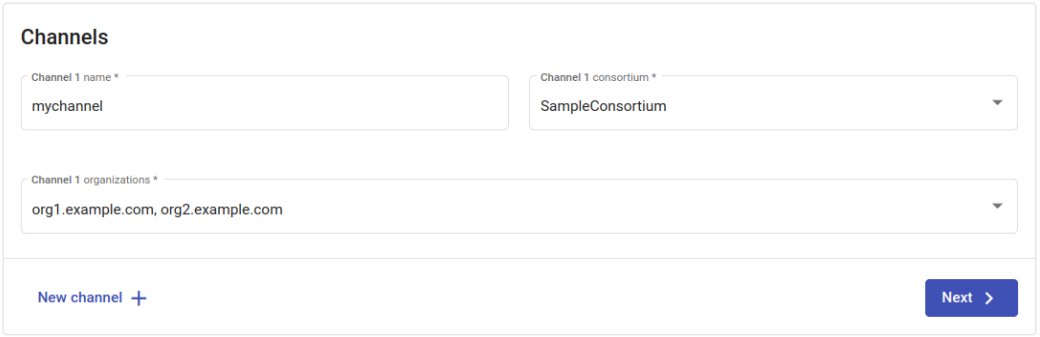
Il secondo step consente di andare a definire nel dettaglio i membri di ogni organizzazione precedentemente specificata.
La prima parte è dedicata al Certificate Authority. È richiesto url e numero di porta dove verrà eseguito e opzionalmente lo stato che figurerà nel suo certificato.
La seconda parte dello step riguarda i membri. È possibile definire fino a n membri a scelta tra peer, orderer, admin, client.
È tuttavia obbligatorio che ogni organizzazione definisca almeno un membro admin e che sulla rete vi sia tra i membri delle organizzazioni almeno un membro orderer.
Se viene selezionata la tipologia di membro peer o orderer è necessario inserire, oltre al nome, url e indirizzo di porta.



3.Consortium

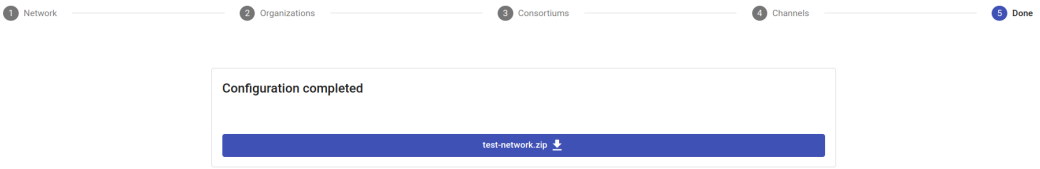
Il terzo step permette di definire i consorzi, ovvero un insieme di organizzazioni che saranno necessarie in quanto ogni canale è amministrato da un consorzio.

Sarà necessario inserire il nome da dare al consorzio, e selezionare le organizzazioni che debbano far parte del consorzio.
È possibile definire più consorzi usando il pulsante ”New consortium”.



4.Channel

Il quarto step consente di definire i canali sulla rete. Per ogni canale è necessario scegliere un nome univoco e il consorzio che ne farà parte e lo amministrerà.
Le organizzazioni del consorzio selezionato saranno quelle che andranno a comporre il canale.



5. Download network

il quinto step consente il download della network

STRUTTURA ZIP

Completando la configurazione di una rete, il server genera e mantiene in persistenza degli zip. Le sottocartelle sono state pensate per racchiudere tutto il necessario affinché sia possibile distribuire ogni membro azzerando problemi di interdipendenza.

Nella directory radice troviamo, i file configurazione di **peer e orderer,** rispettivamente core.yaml e order.yaml, il file config.json in cui viene salvata **l'architettura** della rete.
Lo script **prereq.sh,** si occupa del download dei file binari di Fabric, contenuti dentro la directory **bin**.

test-network

├── **bin**

├── **orgs**

├── config.json

├── core.yaml

├── orderer.yaml

├── prereq.sh

All'interno della cartella **orgs**, sono memorizzate le organizzazioni definite in fase di configurazione.
Per ogni organizzazione, Il nome delle sottocartelle è il nome dell'entità insieme al nome completo dell'organizzazione (nome e dominio).
Nella cartella **mychannel** sono presenti gli script create.sh e join.sh relativi alla creazione e l'aggiunta dell'entità al canale.
Inoltre sono presenti gli script **artfacts.sh** e **configtx.sh** che creano a loro volta i relativi componenti di rete necessari per il corretto funzionamento della rete.
All'interno del file **configtx.yaml** è memorizzata l'intera struttura di rete.

orgs

├── mychannel

├── ordererOrg.example.com

├── org1.example.com

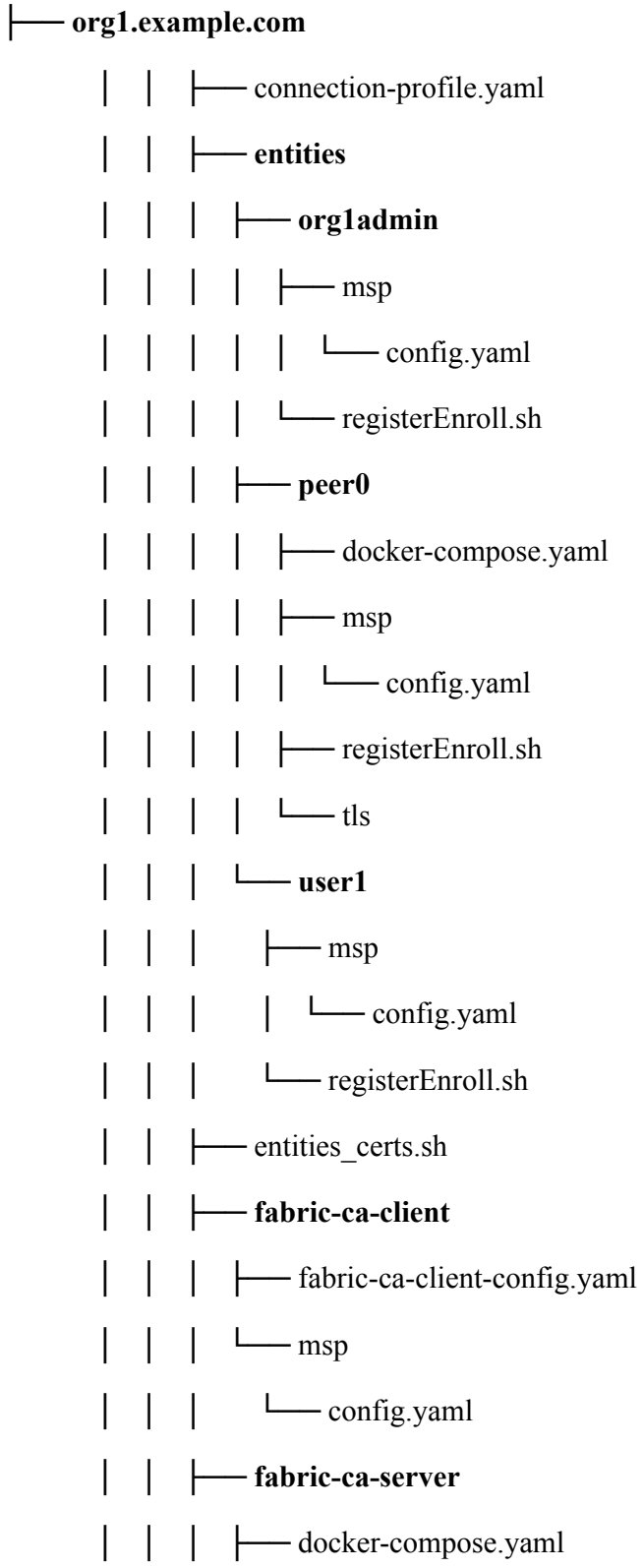
├── org2.example.com

├── artfacts.sh

├── configtx.sh

├── **configtx**.yaml

Per ogni organizzazione, vi è una directory “**entities**” contenente l’entità appartenenti, all’interno di ogni entità è presente la cartella **msp**, che racchiude i materiali crittografici che servono per identificarla e lo script **registerEnroll.sh**, per la registrazione dell'entità.
All’interno dell’organizzazione inoltre sono contenute le directory fabric-ca-server, fabric-ca-client, relative al Certification Authority dell’organizzazione.



```
│   │   │   └── fabric-ca-server-config.yaml
│   │   └── script
│   │       ├── connection-profile.sh
│   │       └── registerEnroll.sh
```

La rete a sua volta è formata da due tipi di file, configurazione e script.

I file di configurazione specificano le informazioni in generale della rete.

Gli script servono per creare i relativi componenti di rete a partire dalle configurazioni.

Lo script che permette la creazione della rete è start.sh, il file va scaricato da https://pros.unicam.it/fabnet/ e aggiunto allo zip.

Creazione della rete

Durante l'esecuzione dello script **start.sh**, vengono scaricati in primis i binari tramite lo script prereq.sh, successivamente la directory contenente i binari viene aggiunta al path delle variabili d'ambiente.

bash **prereq.sh**

export PATH=$PWD/bin:$PATH

```
matteo@matteo-VLT-WX0:~/Scaricati/test-network$ bash start.sh
Downloading HyperledgerFabric binaries
===> Downloading from:  https://github.com/hyperledger/fabric/releases/download/v2.2.0/hyperledger-fabric-linux-amd64-2.2.0.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   680  100   680    0     0    955      0 --:--:-- --:--:-- --:--:--   953
100 72.7M  100 72.7M    0     0  3746k      0  0:00:19  0:00:19 --:--:-- 5199k
==> Done.
===> Downloading from:  https://github.com/hyperledger/fabric-ca/releases/download/v1.4.8/hyperledger-fabric-ca-linux-amd64-1.4.8.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   683  100   683    0     0    472      0  0:00:01  0:00:01 --:--:--   472
100 23.6M  100 23.6M    0     0  2607k      0  0:00:09  0:00:09 --:--:-- 3958k
==> Done.
Remember to add the binaries path to your PATH environment variable otherwise scripts won't be usable.
```

Dopo di che vengono avviate CAs e vengono creati gli artefatti per le rispettive organizzazioni. Tramite l'esecuzione di **artifacts.sh**

bash $PWD/orgs/artifacts.sh

```
Creating ca-org1 ... done
+ fabric-ca-client enroll -u https://admin:adminpw@localhost:7054 --caname ca-org1 --tls.certfiles ../fabric-ca-server/tls-cert.pem
2022/02/03 10:20:20 [INFO] TLS Enabled
2022/02/03 10:20:20 [INFO] generating key: &{A:ecdsa S:256}
2022/02/03 10:20:20 [INFO] encoded CSR
2022/02/03 10:20:21 [INFO] Stored client certificate at /home/matteo/Scaricati/test-network/orgs/org1.example.com/fabric-ca-client/
msp/signcerts/cert.pem
2022/02/03 10:20:21 [INFO] Stored root CA certificate at /home/matteo/Scaricati/test-network/orgs/org1.example.com/fabric-ca-client
/msp/cacerts/localhost-7054-ca-org1.pem
2022/02/03 10:20:21 [INFO] Stored Issuer public key at /home/matteo/Scaricati/test-network/orgs/org1.example.com/fabric-ca-client/m
sp/IssuerPublicKey
2022/02/03 10:20:21 [INFO] Stored Issuer revocation public key at /home/matteo/Scaricati/test-network/orgs/org1.example.com/fabric-
ca-client/msp/IssuerRevocationPublicKey
+ set +x
+ fabric-ca-client register --id.name peer0 --id.secret peer0pw --caname ca-org1 --id.type peer --tls.certfiles ../fabric-ca-server
/tls-cert.pem
2022/02/03 10:20:21 [INFO] Configuration file location: /home/matteo/Scaricati/test-network/orgs/org1.example.com/fabric-ca-client/
fabric-ca-client-config.yaml
2022/02/03 10:20:21 [INFO] TLS Enabled
2022/02/03 10:20:21 [INFO] TLS Enabled
```

Successivamente viene eseguito lo script **configtx.sh**, crea il genesis block e i channels txs

bash $PWD/orgs/configtx.sh

```
2022-02-03 10:43:12.558 CET [common.tools.configtxgen] main -> INFO 001 Loading configuration
2022-02-03 10:43:12.576 CET [common.tools.configtxgen.localconfig] completeInitialization -> INFO 002 orderer type: etcdraft
2022-02-03 10:43:12.576 CET [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 Orderer.EtcdRaft.Options unse
t, setting to tick_interval:"500ms" election_tick:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
2022-02-03 10:43:12.576 CET [common.tools.configtxgen.localconfig] Load -> INFO 004 Loaded configuration: /home/matteo/Scaricati/te
st-network/test-network/orgs/configtx.yaml
2022-02-03 10:43:12.579 CET [common.tools.configtxgen] doOutputBlock -> INFO 005 Generating genesis block
2022-02-03 10:43:12.579 CET [common.tools.configtxgen] doOutputBlock -> INFO 006 Writing genesis block
2022-02-03 10:43:12.603 CET [common.tools.configtxgen] main -> INFO 001 Loading configuration
2022-02-03 10:43:12.618 CET [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/matteo/Scaricati/te
st-network/test-network/orgs/configtx.yaml
2022-02-03 10:43:12.618 CET [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channel configtx
2022-02-03 10:43:12.620 CET [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writing new channel tx
2022-02-03 10:43:12.641 CET [common.tools.configtxgen] main -> INFO 001 Loading configuration
2022-02-03 10:43:12.656 CET [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/matteo/Scaricati/te
st-network/test-network/orgs/configtx.yaml
2022-02-03 10:43:12.656 CET [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer update
2022-02-03 10:43:12.658 CET [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer update
2022-02-03 10:43:12.686 CET [common.tools.configtxgen] main -> INFO 001 Loading configuration
2022-02-03 10:43:12.704 CET [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/matteo/Scaricati/te
st-network/test-network/orgs/configtx.yaml
2022-02-03 10:43:12.704 CET [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer update
2022-02-03 10:43:12.706 CET [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer update
```

Successivamente vengono avviati i container dei peer/ orderer tramite il relativo file di **docker.compose.yaml**

docker-compose -f $PWD/orgs/org1.example.com/entities/peer0/docker-compose.yaml up -d

```
Creating network "peer0_default" with the default driver
Creating volume "peer0_peer0.org1.example.com" with default driver
Creating peer0.org1.example.com ... done
Creating volume "peer0_peer0.org2.example.com" with default driver
WARNING: Found orphan containers (peer0.org1.example.com) for this project. If you removed or renamed this service in your compose
file, you can run this command with the --remove-orphans flag to clean it up.
Creating peer0.org2.example.com ... done
Creating network "orderer_default" with the default driver
Creating volume "orderer_orderer.ordererOrg.example.com" with default driver
Creating orderer.ordererOrg.example.com ... done
2a1c0770f7442bf5928a74cbf8372a28fc8854911cbd689ff36f2455f0375bfa
```

Una volta avviati i container si passa alla creazione delle rete e alla connessione tra i seguenti container creati

docker network create test-network

docker network connect test-network orderer.ordererOrg.example.com

docker network connect test-network peer0.org1.example.com

docker network connect test-network peer0.org2.example.com

Infine tramite gli scrpit mychannel/ create.sh e join.sh si procede all'inserimento delle rispettive entità nei rispettivi canali.

```
2022-01-29 19:41:00.960 CET [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-01-29 19:41:00.975 CET [cli.common] readBlock -> INFO 002 Expect block, but got status: &{NOT_FOUND}
2022-01-29 19:41:00.977 CET [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2022-01-29 19:41:01.179 CET [cli.common] readBlock -> INFO 004 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2022-01-29 19:41:01.182 CET [channelCmd] InitCmdFactory -> INFO 005 Endorser and orderer connections initialized
2022-01-29 19:41:01.384 CET [cli.common] readBlock -> INFO 006 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2022-01-29 19:41:01.390 CET [channelCmd] InitCmdFactory -> INFO 007 Endorser and orderer connections initialized
2022-01-29 19:41:01.592 CET [cli.common] readBlock -> INFO 008 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2022-01-29 19:41:01.602 CET [channelCmd] InitCmdFactory -> INFO 009 Endorser and orderer connections initialized
2022-01-29 19:41:01.805 CET [cli.common] readBlock -> INFO 00a Expect block, but got status: &{SERVICE_UNAVAILABLE}
2022-01-29 19:41:01.812 CET [channelCmd] InitCmdFactory -> INFO 00b Endorser and orderer connections initialized
2022-01-29 19:41:02.015 CET [cli.common] readBlock -> INFO 00c Expect block, but got status: &{SERVICE_UNAVAILABLE}
2022-01-29 19:41:02.022 CET [channelCmd] InitCmdFactory -> INFO 00d Endorser and orderer connections initialized
2022-01-29 19:41:02.229 CET [cli.common] readBlock -> INFO 00e Received block: 0
2022-01-29 19:41:02.301 CET [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-01-29 19:41:02.311 CET [channelCmd] update -> INFO 002 Successfully submitted channel update
2022-01-29 19:41:02.354 CET [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-01-29 19:41:02.364 CET [channelCmd] update -> INFO 002 Successfully submitted channel update
2022-01-29 19:41:02.475 CET [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-01-29 19:41:02.830 CET [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2022-01-29 19:41:02.889 CET [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-01-29 19:41:03.068 CET [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
matteo@matteo-VLT-WX0:~/Documenti/ultima/test-network$ docker ps -a
CONTAINER ID   IMAGE                               COMMAND             CREATED         STATUS         PORTS
 NAMES
6a64606ad9ea   hyperledger/fabric-orderer:2.2.0    "orderer"           7 minutes ago   Up 7 minutes   0.0.0.0:7050->7050/tcp, :::7050->7050/tcp
 orderer.ordererOrg.example.com
23c2f1091ec7   hyperledger/fabric-peer:2.2.0       "peer node start"   7 minutes ago   Up 7 minutes   7051/tcp, 0.0.0.0:9051->9051/tcp, :::9051->9051/tcp
 peer0.org2.example.com
0a47ac492053   hyperledger/fabric-peer:2.2.0       "peer node start"   7 minutes ago   Up 7 minutes   0.0.0.0:7051->7051/tcp, :::7051->7051/tcp
 peer0.org1.example.com
```

CONCLUSIONI

In uno scenario **b2b** la blockchain Fabric 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.**

Essendo un argomento ancora in aggiornamento, la documentazione relativa a Fabric e tutti i suoi servizi è ancora poca e non ben definita. Nelle community di supporto Stack Overflow (consigliata all'interno documentazione Fabric) o Reddit trovare una soluzione, ricevere una risposta oppure trovare una domanda inerente a un mio dubbio, che poi fosse inerente anche alla mia versione di Fabric è stato spesso impossibile.

Sviluppi futuri per l'utilizzo della rete in campo aziendali riguardano il **deploy di peer e orderer**, attraverso i relativi file di configurazione e l'implementazione della gestione delle **policies.**

In Fabric le **policies** rappresentano il modo in cui vengono prese le decisioni nella rete, il modo in cui i membri raggiungono un accordo sull'accettazione o il rifiuto delle modifiche alla rete, a un canale o a uno **smart contract**.