POR-FESR EMILIA ROMAGNA 2014-2020

Asse 1 - Ricerca e innovazione

Azione 1.2.2 - Supporto alla realizzazione di progetti complessi di attività di ricerca e sviluppo su poche aree tematiche di rilievo e all'applicazione di soluzioni tecnologiche funzionali alla realizzazione della strategia di S3

Bando 2018

Progetti di ricerca industriale strategica rivolti agli ambiti prioritari della Strategia di Specializzazione Intelligente

Sistemi interoperabili ed efficienti per la gestione sicura di filiere industriali

Deliverable D4.2:
Descrizione del software e manuale utente

Data di consegna prevista:	31 Gennaio 2022
Autori:	MECHLAV, CIRI ICT, CRIS, CROSSTEC, INFN-TTLab
Versione:	1

Indice

1. Introduzione

In questo allegato tecnico, che rientra nell'ambito della Fase 4 "Sperimentazione e verifica dei risultati", sotto forma di manuale utente si descrivono i passi principali da seguire per far sì che possano essere installate e configurate correttamente le rispettive reti blockchain utilizzate per i due casi d'uso (Carpigiani e Bianco Accessori) secondo il modello strutturale del tipo IaaS privato. Il contenuto di questo documento consentirà a un utente, che possiede delle conoscenze adeguate a questo contesto, di poter replicare una versione base delle stesse reti blockchain descritte nell'allegato tecnico 3.2.

Il documento è strutturato nelle seguenti modalità: nella Sezione 2 si descrive il manuale utente inerente al caso d'uso Bianco Accessori; nella Sezione 3 si descrive il manuale utente per la creazione di una rete Fabric inerente al caso d'uso Carpigiani, partendo dai prerequisiti necessari all'installazione fino allo sviluppo della rete vera e propria.

2. Caso d'uso Bianco Accessori basato su Ethereum

Si prende in considerazione la procedura di installazione del sistema SmartChain per il caso d'uso Bianco Accessori a partire dai repository di progetto tramite il sistema *Docker.* In particolare, si propone la procedura di installazione per un ambiente di test locale. La procedura può essere facilmente adattata per essere impiegata al fine di installare il sistema in un ambiente distribuito.

In Sezione 2.1 si descrive l'ambiente di test ottenibile seguendo la procedura. In Sezione …

2.1. Descrizione dell'ambiente di test

L'ambiente di test iniziale prevede l'esposizione dei seguenti servizi:

- Il servizio di Identity and Access Control Management (IAM) istanziato con IndigoIAM presso lo URI https://iam.local:443;
- Il servizio di Key Management (KMS) istanziato con Hashicorp Vault istanziato presso lo URI https://vault.local:8200;
- Il servizio di gestione confidenziale della blockchain (CBM) istanziato tramite l'implementazione realizzata nell'ambito di questo stesso progetto, disponibile presso lo URI http://cbm.local:9090.
- Il servizio blockchain compatibile con i linguaggi di Ethereum e istanziata tramite il software Ganache, raggiungibile presso lo URI http://ganache.local:8545

L'ambiente di test esegue anche ulteriori sottoservizi impiegati da uno o più dei componenti appena descritti, che non verranno descritti in modo dettagliato in questo documento.

2.2. Generazione delle chiavi crittografiche e dei certificati digitali

L'esecuzione richiede la generazione di chiavi crittografiche asimmetriche per la corretta generazione di certificati digitali e la corretta gestione della logica di rilascio di strutture dati attestate crittograficamente per il rilascio e la verifica delle informazioni di autenticazione e autorizzazione nell'ambito dei protocolli di gestione federata delle identità (OpenID Connect) e di delega di autorizzazioni (OAuth2). Il sistema di autenticazione e di gestione delle identità Indigo IAM che viene utilizzato richiede l'impiego di Json Web Keys (JWK) per la firma e la verifica di JSON Web Token (JWT).

All'interno dei file del codice fornito sono incluse alcune chiavi utilizzabili in ambienti di test. In caso di utilizzo in ambienti accessibili pubblicamente è necessario rigenerare le chiavi, ad esempio seguendo le seguenti procedure. Le chiavi di test differiscono dalle chiavi utilizzabili in ambiente reale unicamente per l'autorita' che li firma: in questo ambiente di test abbiamo utilizzato certificati x509 self-signed, mentre in ambiente di produzione è necessario generare certificati firmati da un'autorita' pubblicamente riconosciuta.

Generazione delle chiavi e dei certificati self-signed x509

È necessario creare certificati digitali per i servizi IndigoIAM e Vault.

```
./create-x509.sh iam_local
./create-x509.sh vault_local
chmod go+r ssl-*/*.key
```

Notare che la modifica dei permessi di accesso alle chiavi segrete deve essere impostata in modo più oculato in un ambiente di produzione.

Generazione Json Web Keys (JWK)

Per generare il materiale crittografico necessario alla generazione si impiega un software esterno pubblicamente disponibile su github e ottenibile tramite git:

```
git clone https://github.com/mitreid-connect/json-web-key-generator
cd json-web-key-generator
```

É necessario compilare il software utilizzando *maven*:

```
mvn package
```

In questa istanza si genera il file con la json Web key nel file *iam-keystore.jwks*:

```
java -jar target/json-web-key-generator-0.9-SNAPSHOT-jar-with-
dependencies.jar -t RSA -s 2048 -S -i rsa1 | tail -n +2 | tee iam-
keystore.jwks
```

Si sposta il file all'interno della dovuta directory per l'accesso da parte di IndigoIAM:

```
cd ..
mkdir -p jwk-iam
mv json-web-key-generator/iam-keystore.jwks -t jwk-iam/
```

Infine, si esporta la sola chiave pubblica che dovrà utilizzare il KMS ed eventualmente altri servizi applicativi per la verifica dell'autenticità dei token rilasciati da IndigoIAM:

```
utils/jwk_to_public_pem.py jwk-iam/iam-keystore.jwks -o ssl-vault_local/iam-jwt-pk.pem
```

2.3. Esecuzione e configurazione di IndigoIAM

Eseguire i servizi associati a IndigoIAM tramite il seguente comando

```
docker-compose up --build --force-recreate --remove-orphans -d iam
```

Accedere alla homepage di IndigoIAM allo URI https://iam.local e loggarsi con utente **admin** e password **password** (che dovrà essere modificata al primo accesso). Tramite l'interfaccia utente eseguire le seguenti operazioni:

- Registrare un nuovo client OAuth2 andando nella voce **MitreID Dashboard / Manage Client.**
    - Nella scheda **Main tab** importare il nome del client come **vault** e aggiungere i seguenti Redirect URIs:
        - https://vault.local:8200/ui/vault/auth/oidc/oidc/callback
        - https://vault.local:8200/v1/auth/oidc/oidc/callback
    - Nella scheda **Access tab** definire le seguenti impostazioni:
        - Selezionare gli scope **openid, profile, email, address, phone, offline_access**
        - Selezionare i grant type **authorization code, password, device, token exchange**
        - Selezionare il response type **code**
    - Prendere nota del valore del parametro **ClientSecret** mostrato dall'interfaccia grafica e impostarlo come valore del campo **OIDC_CLIENT_SECRET** all'interno del file **vault-env** presente nella directory **vault-config**
- Creare i seguenti utenti tramite la dashboard di IndigoIAM: **user1-company, user1-company2, user2-company1, user2-company2, vault-admin, manager-companies, webapp-admin**. Nella modalità di debug impiegata in questo contesto, è stato scelto di non impostare alcuna password predefinita per gli utenti creati e quindi di non inviare alcuna email per l'impostazione delle password da parte degli utenti, che avverrebbe in contesti di utilizzo in ambienti reali di produzione. Al fine di impostare la password di default per ogni utente è necessario accedere accedere ai link per l'impostazione delle password presenti nei log generati da IndigoIAM, che sono visualizzabili tramite il comando **docker logs indigoiam**. È possibile impostare password arbitrariamente complesse per ogni utente.

- Creare quattro gruppi e assegnare ad ognuno di essi gli utenti indicati:
    - gruppo **company-c1**, con utenti **user1-company1** e **user2-company1**;
    - gruppo **company-c2**, con utenti **user1-company2** e **user2-company2**;
    - gruppo **admin-vault**, con utente **admin-vault**;
    - gruppo **manager-companies**, con utente **manager-companies**.

Vincoli di configurazione e utilizzo

Si nota che data l'attuale configurazione delle policy di autorizzazione, è estremamente importante rispettare i seguenti vincoli:

- tutti e soli i gruppi associati a delle imprese devono impiegare il prefisso **company-**
- ogni utente associato con un gruppo con prefisso **company-** non deve associato ad alcun altro gruppo

Il non rispetto di questa configurazione può provocare malfunzionamenti e problemi di sicurezza. Questi vincoli sono un trade-off individuato nell'ambito dell'integrazione di IndigoIAM e Hashicorp Vault per consentire una gestione flessibile delle policy di autorizzazione. All'interno del progetto è stata individuata anche una variante che non ha questi vincoli, ma che richiede l'impiego di un maggior numero di policy di autorizzazione all'interno di Vault e di operazioni necessarie per la configurazione di Vault stesso (questa variante non viene descritta in questo documento). In particolare, attualmente il numero di policy è costante e indipendente rispetto al numero di imprese registrate nel sitema, mentre la variante richiedebbe un numero di policy che dipende linearmente rispetto al numero di imprese registrate.

Una soluzione alternativa che non incorra in questi trade-off richiederebbe richiederebbe la modifica di alcuni aspetti di configurazione avanzata di Indigo IAM nell'ambito della codifica dei gruppi gerarchici all'interno dei token JWT rilasciati. L'approfondimento di questo approccio è al di fuori dello scope del progetto.

2.4. Esecuzione e configurazione del servizio KMS

Prima di eseguire il servizio KMS è necessario aver impostato il valore della variabile **OIDC_CLIENT_SECRET** all'interno del file **vault-config/vault-env** come descritto nella sezione precedente. L'esecuzione delle seguenti operazioni senza aver configurato correttamente questo parametro provoca malfunzionamenti all'interno del sistema.

Per eseguire i servizi associati a Vault, che istanzia il KMS del sistema, eseguire il seguente comando:

```
docker-compose up --build --force-recreate --remove-orphans -d vault
```

Al fine di inizializzare le funzionalità di Vault è necessario ottere una shell all'interno del container appena eseguito tramite il seguente comando:

```
docker exec -it vault.local sh
```

Tramite la linea di comando appena ottenuta eseguire i seguenti comandi:

- Inizializzare le chiavi di sicurezza di Vault tramite il seguente comando

  ```
  ./vault_init.sh
  ```

  - Nota: nel caso non sia la prima volta che viene eseguito questo servizio è necessario eseguire invece il comando `./vault_unseal.sh`
- Eseguire il comando `./setup_oidc.sh` per abilitare l'autenticazione OpenID Connect (OIDC) di IndigoIAM sull'interfaccia grafica di Vault. Lo script abilita tre ruoli:
  - **company (default)**: gestisce il login da parte di utenti che rappresentano imprese. Tutti gli utenti associati a un gruppo che ha nel suo nome il prefisso **company-** possono effettuare il login con questo ruolo (nota: verificare il rispetto dei vincoli descritti nella sezione precedente per la corretta gestione di questi gruppi);
  - **admin**: permette l'esecuzione di operazioni di amministrazione all'interno di Vault stesso. Possono accedere con questo ruolo tutti gli utenti di IndigoIAM che sono stati associati al gruppo **admin-vault**;
  - **manager**: permette l'attivazione e la disattivazione dei keyring delle imprese (necessario, ad esempio, nel momento in cui viene registrata a sistema una nuova impresa). Permette inoltre l'inserimento di nuove policy per la condivisione di contenuti fra utenti del sistema. Tutti gli utenti di IndigoIAM che sono associati al gruppo **manager-companies** possono effettuare il login con questo ruolo.
- Attivare due imprese **company-c1** e **company-c2** di test eseguendo i seguenti comandi:
  - `./activate_company.sh company-c1`
  - `./activate_company.sh company-c2`

2.5. Esecuzione e configurazione del servizio blockchain

Eseguire i servizi associati alla blockchain, istanziata tramite il software Ganache, tramite il seguente comando

```
docker-compose up –build –force-recreate –remove-orphans -d
```

Alla prima esecuzione è necessario effettuare il deployment sulla blockchain di Ganache tramite le seguenti operazioni:

- accedere al terminale del container tramite il seguente comando:

  ```
  docker exec -it ganache.local bash
  ```
- compilare lo smart contract ed entrare nella console di brownie:

```
brownie networks add smartchain bc-localhost host=http://127.0.0.1:8545 \
```

```
    chainid=1337
brownie console --network bc-localhost
```

- effettuare il deployment dello smart contract compilato:

```
instance = SupplierChain.deploy({'from': accounts[0].address, 'gasPrice': 0})
```

- annotarsi l'indirizzo di deployment dello smart contract per configurare in modo appropriato gli altri servizi che dovranno utilizzarlo.

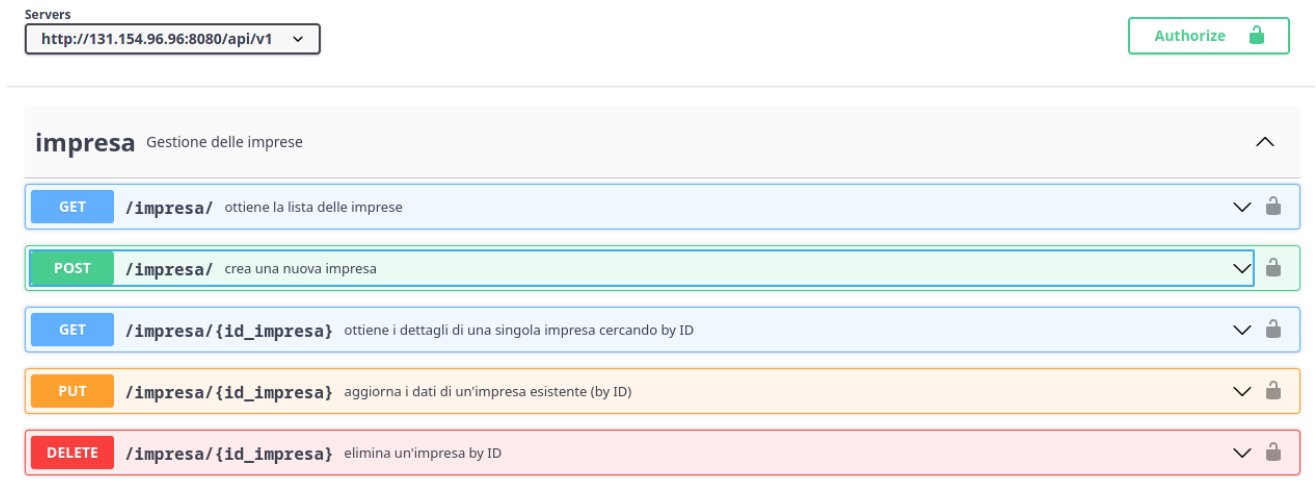
2.6. Esecuzione del CBM

Il servizio CBM è un middleware che non conserva lo stato di alcuna informazione e di default inoltra i token di autorizzazione da IndigoIAM a Vault senza effettuare alcuna verifica. Per questo motivo la sua esecuzione non richiede alcuna operazione di configurazione aggiuntiva.

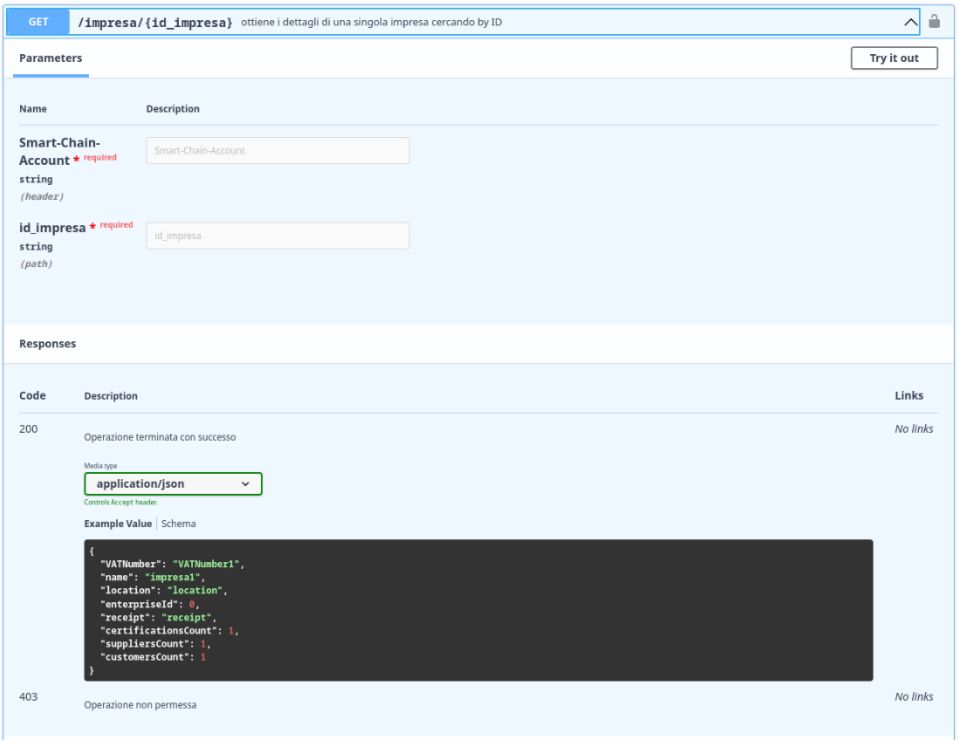
L'esecuzione del CBM richiede l'esecuzione del seguente comando:

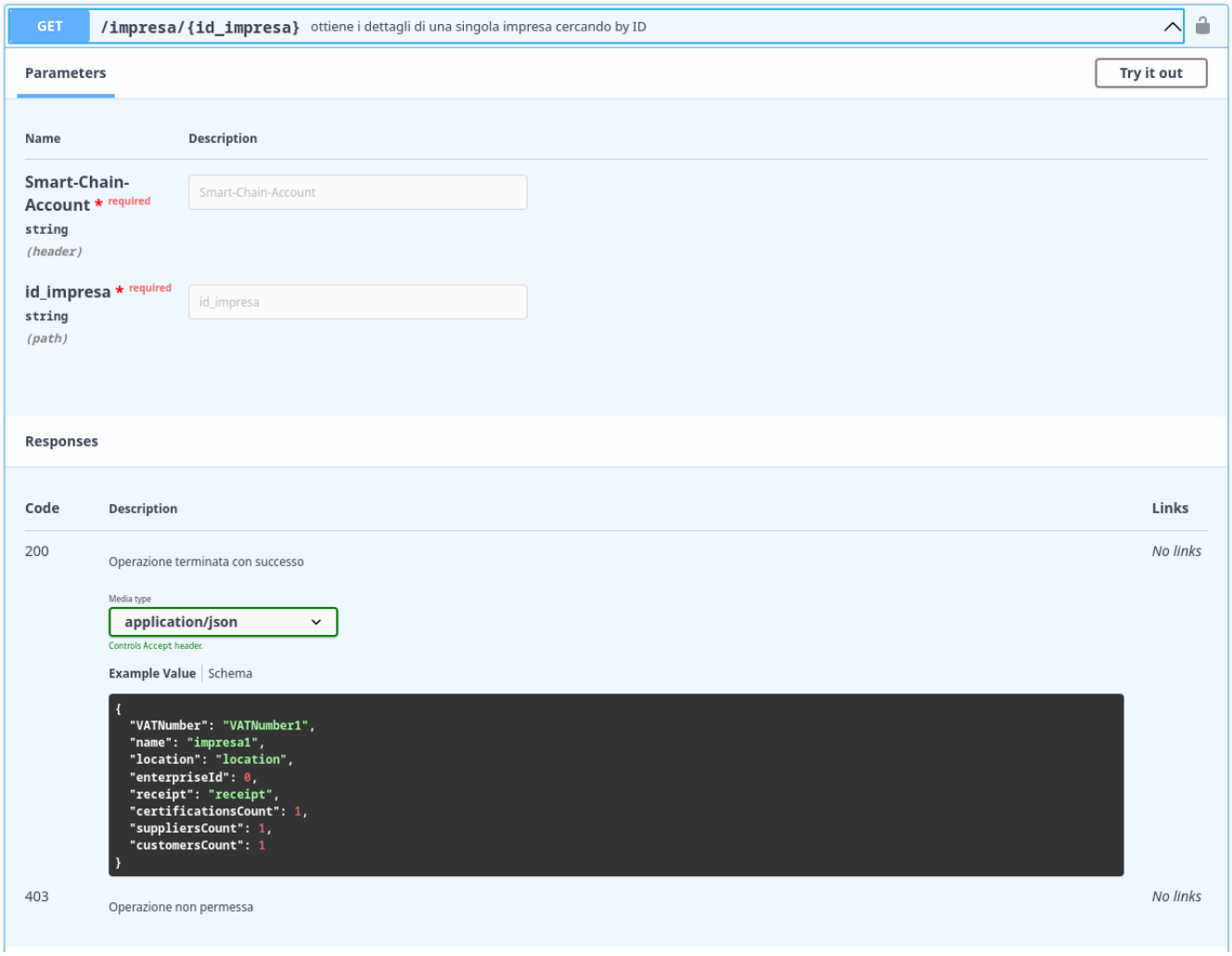
```
podman-compose up --build --force-recreate --remove-orphans -d
```

Una volta deployata, l'applicazione espone una Web API RESTful che permette di interagire interagire con i servizi blockchain rispettando le policy. Di seguito si mostrano alcuni screen shot di esempio nell'ambito di utilizzo di un server di test installato presso le infrastrutture INFN.



Le chiamate sono diversificate cambiando il metodo http (GET, POST, PUT, e DELETE) e variando la parte finale dell'URI. in particolare, nel caso si voglia interagire con la risorsa impresa, è sufficiente aggiungere /impresa all'uri di base passando, nel caso di GET, POST e DELETE, l'id dell'impresa.


GET
/impresa/{id_impresa}
ottiene i dettagli di una singola impresa cercando by ID
Parameters
Try it out
Name
Description
Smart-Chain-Account * required
string
(header)
Smart-Chain-Account
id_impresa * required
string
(path)
id_impresa
Responses
Code
Description
Links
200
Operazione terminata con successo
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
{
"VATNumber": "VATNumber1",
"name": "impresa1",
"location": "location",
"enterpriseId": 0,
"receipt": "receipt",
"certificationsCount": 1,
"suppliersCount": 1,
"customersCount": 1
}
403
Operazione non permessa
No links



Nel caso in cui l'operazione vada a buon fine si riceverà un una risposta HTTP con status 200 mentre, nel caso l'operazione non sia autorizzata, si riceverà il codice 403.

2.7. Configurazione garanzie di confidenzialità dei dati

Il prototipo realizzato offre la possibilità di applicare delle modifiche a file di configurazione del CBM e di alcuni campi delle strutture dati dello smart contract al fine di offrire dirrenti trade-off in termini di confidenzialità dei dati memorizzati su blockchain. A questo fine, l'applicazione CBM può essere configurata per indicare quali campi delle strutture dati memorizzate su smart contract. Nel seguente esempio si mostrano tre variabili di configurazione che mostrano i campi cifrati per le strutture dati relative alle imprese (CBM_ENCRYPTED_FIELDS_COMPANY), ai fornitori registrati (CBM_ENCRYPTED_FIELDS_REGSUPPLIER) e ai fornitori non registrati (CBM_ENCRYPTED_FIELDS_UNREGSUPPLIER).

```
CBM_ENCRYPTED_FIELDS_COMPANY = 'vat_number, name, location'
CBM_ENCRYPTED_FIELDS_REGSUPPLIER = 'role'
CBM_ENCRYPTED_FIELDS_UNREGSUPPLIER = 'vat_number, name, location, role'
```

La configurazione mostrata nell'esempio implica che il CBM cifrerà i dati relativi alla partita IVA delle imprese (`vat_number`), al loro nome (`name`) e ai dettagli della loro posizione (`location`). Campi analoghi vengono cifrati anche nel caso di fornitori non registrati presso il sistema, i cui dati vengono inseriti solo all'interno dei servizi blockchain, e dei dettagli dei ruoli di fornitura. Il sistema consente di modificare quali campi devono essere cifrati con una semplice modifica delle variabili di configurazione e con modifiche minime agli smart contract per rispettare i corretti tipi di dati (in particolare, tutti i dati cifrati devono avere tipo *bytes* all'interno degli smart contract).

2.8. Installazione e accesso all'Applicazione Web

Per eseguire i servizi associati alla webapp, impostare il valore della variabile **CLIENT_SECRET** all'interno del file webapp-config/webapp-env con il segreto annotato in precendenza per il client IAM "webapp". Dopodichè, eseguire il seguente script:

```
./webapp_init.sh
```

Una volta che l'esecuzione dell'Applicazione Web è andata a buon fine, è possibile raggiungere la pagina Home dell'applicazione.

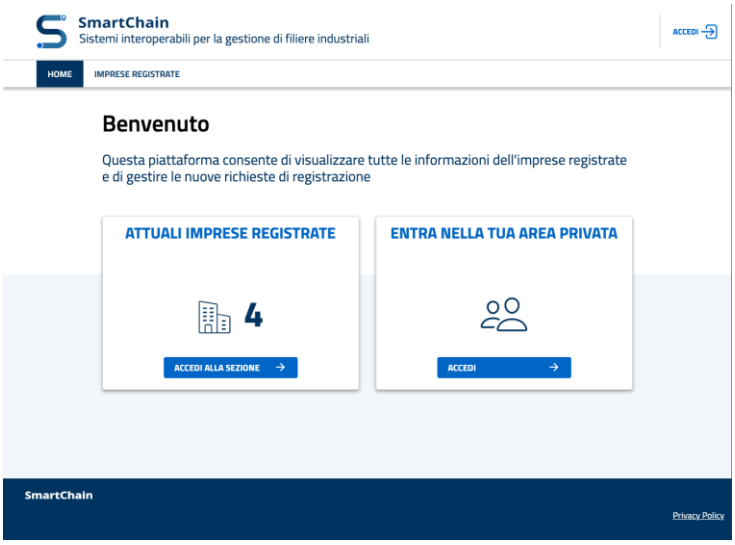


Figura 1: Schermata Home della webapp

Eseguendo il login come admin o come utente impresa si viene reindirizzati alla rispettiva area privata dove usufruire delle specifiche funzionalità. In caso di assenza di login, è possibile accedere alle funzionalità previste per l'utenza ospite riguardanti i dati di natura pubblica.

La schermata Home con l'interfaccia grafica per effettuare il login è illustrata in Figura 1. Di seguito si descrive l'interfacciamento con il sistema a seconda dei differenti profili di utenza supportati.

Funzionalità utenza Amministrazione

Se in possesso di credenziali di amministrazione si viene reindirizzati nella relativa Home Page, composta come di seguito:

- nella parte centrale una dashboard che mostra il numero di richieste di registrazione pendenti (da gestire) e il numero di imprese già registrate su SmartChain;
- funzione di accesso all'elenco delle richieste di registrazione da parte degli utenti;
- funzione di accesso all'elenco delle imprese già registrate.

L'amministratore può accedere alle pagine con l'elenco delle richieste di registrazione pendenti, strutturata con un'area di ricerca e una griglia dei risultati. Qui è possibile ricercare un'azienda con i seguenti parametri:

- Ragione Sociale;
- Partita IVA;
- Nome responsabile compilazione;
- Cognome responsabile compilazione;
- Indirizzo legale.

Tutti i parametri di ricerca sono a testo libero ed utilizzati per filtrare i risultati che contengono la porzione di testo indicato. Le richieste sono consultabili in modalità read-only e l'amministratore non può modificarle in alcun modo. L'amministratore può decidere se accettare una specifica richiesta di iscrizione dell'azienda o se respingerla:

- In caso l'azienda non venga autorizzata, l'utente riceve un messaggio via email in cui sarà indicato che la sua azienda non è stata accettata.
- In caso l'azienda venga autorizzata, l'utente può accedere alle funzionalità dedicate.

Funzionalità utenza Impresa

Se l'utente non ha ancora associata alcuna impresa, troverà sulla pagina principale di SmartChain un tasto funzione per registrare la propria impresa dove gli saranno richiesti i seguenti dati come da Figura 2:

- informazioni sull'impresa;
- dati non critici di impresa: il responsabile, gli stabilimenti dell'impresale nformazioni di contatto dell'impresa e la documentazione non strutturata in formato PDF, JPG, GIF o XML.

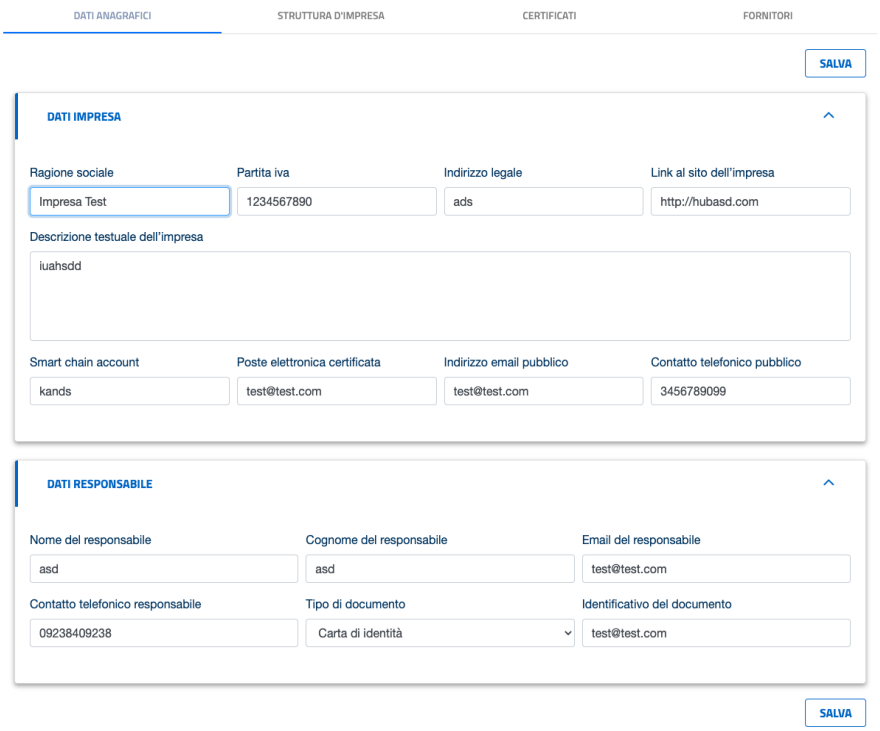


Figura 2: inserimento dati registrazione impresa

Al termine dell'inserimento delle informazioni necessarie, l'utente potrà richiedere al sistema di registrare la propria impresa premendo un apposito pulsante. Una volta che la propria richiesta di registrazione viene appovata, l'utente può accedere alla schermata Home riservata, come in Figura 3.

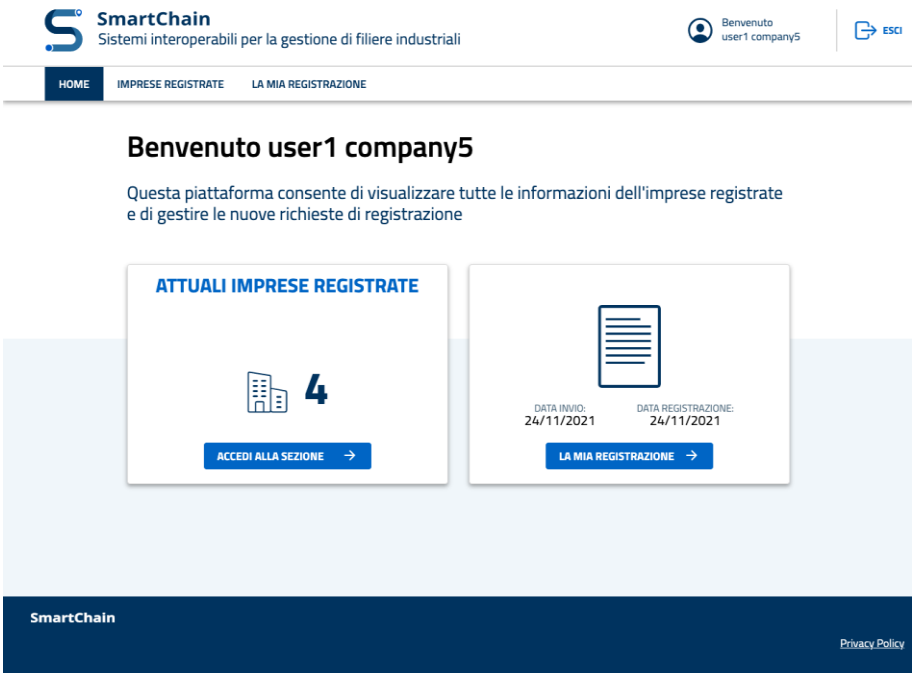


Figura 3: Schermata Home impresa registrata

La schermata Home di un'utenza Impresa permette l'accesso alle seguenti funzioni:

- accesso alle informazioni pubbliche delle altre imprese registrate;
- accesso alla propria registrazione, dove è possibile rivedere, modificare, inserire ed eliminare i dati anagrafici, i dati di struttura dell'impresa, i certificati e i fornitori.

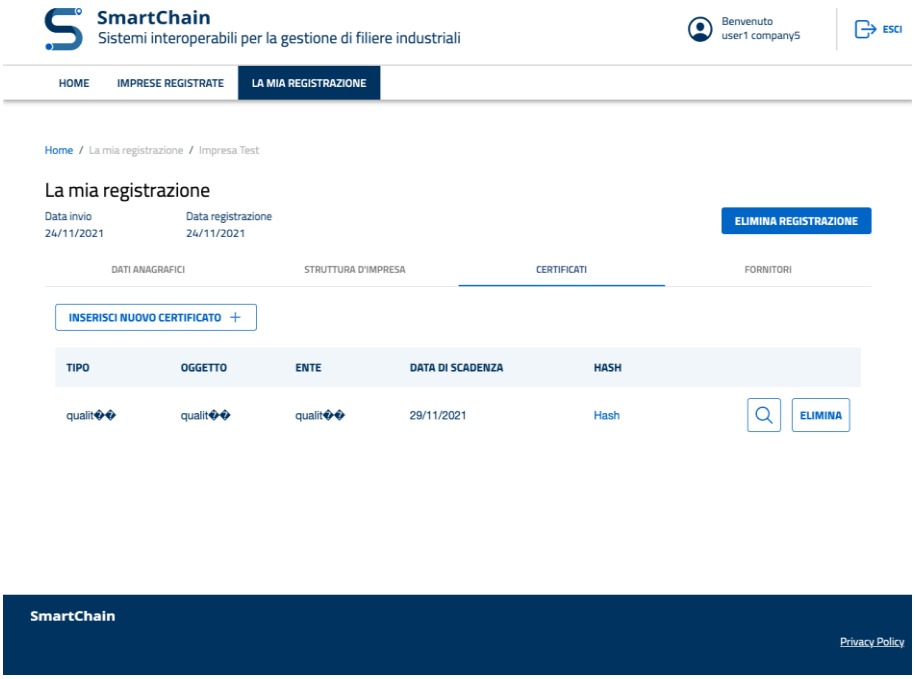


Figura 4: Schermata gestione dei certificati

In particolare, accedendo sezione "certificati", l'utente impresa troverà di fronte la schermata di gestione dei propri certificati (come illustrato in Figura 4). Qui può inserire un nuovo certificato cliccando sul pulsante dedicato e consultare la panoramica dei certificati inseriti in precedenza. Per ogni certificato della panoramica, può ottenere i dettagli specifici del certificato o eliminarlo. Accedendo invece alla sezione "fornitori", l'utente impresa troverà di fronte la schermata di gestione dei propri certificati (come illustrato in Figura 5). Analogamente alla sezione precedente, qui può inserire un nuovo fornitore cliccando sul pulsante dedicato e consultare la panoramica dei fornitori inseriti in precedenza. Per ogni fornitore della panoramica, può ottenere i dettagli specifici di fornitura o eliminarlo.

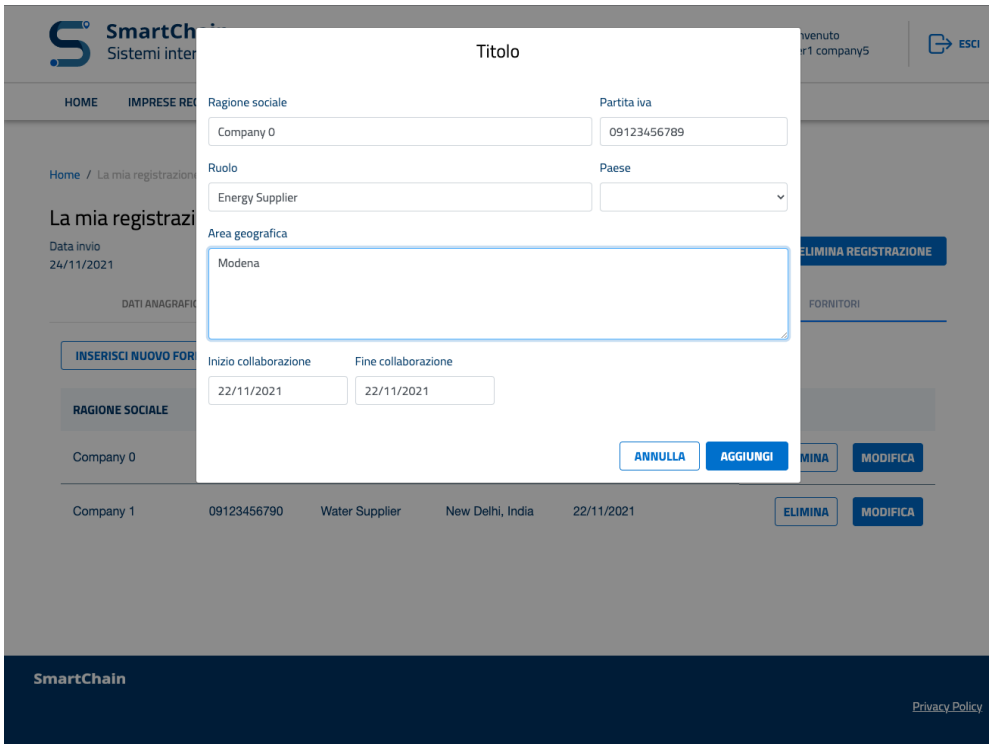


Figura 5: Schermata inserimento fornitore

Funzionalità utenza Ospite

Un utente non in possesso di credenziali, può comunque accedere alle informazioni pubbliche della piattaforma. Nello specifico, dalla schermata Home di Figura 1, cliccando sull'apposita sezione ha accesso alla lista delle imprese registrate, come illustrato in Figura 6. Qui può utilizzare il pulsante di ricerca per filtrare la lista delle impresee utilizzando la ricerca per keyword e per filtri su un set di campi (nome impresa, partita iva, provincia della sede legale).

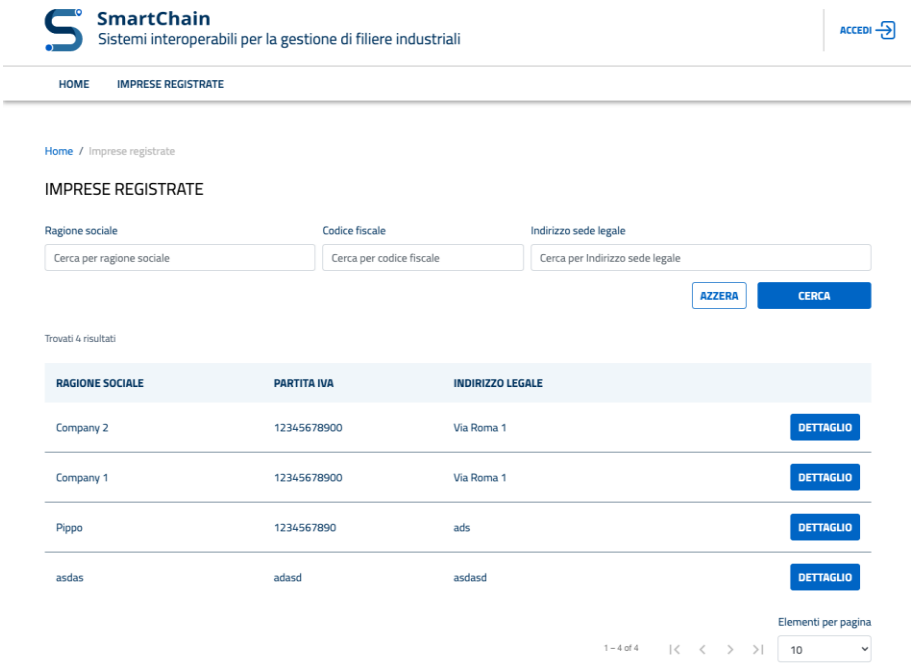


Figura 6: Lista delle informazioni pubbliche di imprese

Cliccando su una specifica impresa, verrà mostrata una maschera di dettaglio con le informazioni pubbliche delle imprese che si sono registrate, cioè le informazioni non-critiche e la lista dei certificati (come illustrato in Figura 7).

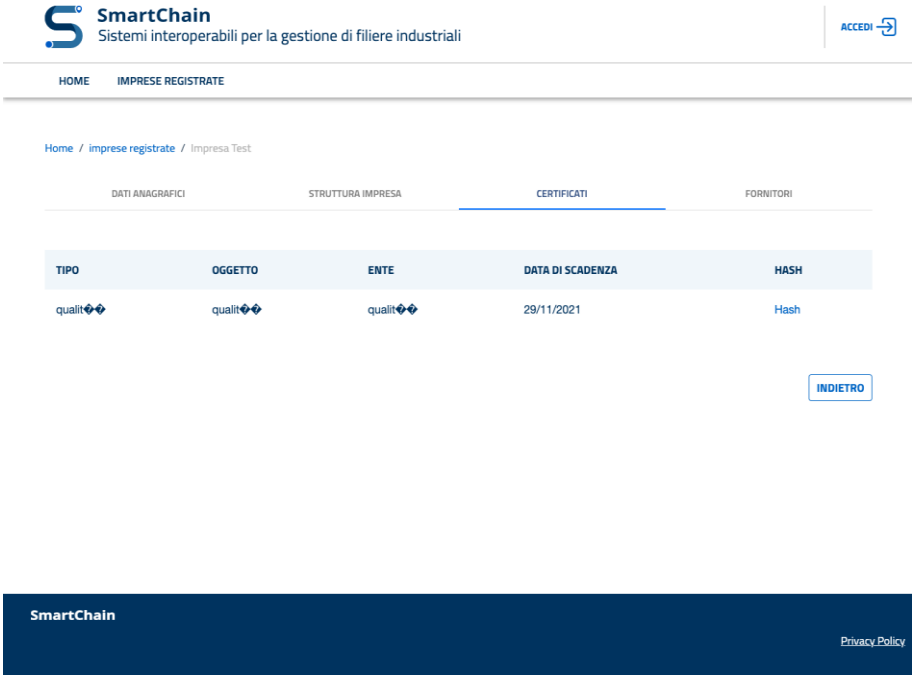


Figura 7: Schermata visualizzazione informazioni pubbliche certificati

3. Caso d'uso Carpigiani basato su Hyperledger Fabric

In questa sezione viene esposta l'implementazione base dal quale il gruppo di ricerca è partito per creare l'architettura della rete blockchain pensata per il caso d'uso Carpigiani. Nella prima parte vengono indicati tutti i prerequisiti software necessari per poter eseguire correttamente la rete Fabric e come impostare tutte le variabili d'ambiente necessarie per il corretto uso degli strumenti. Successivamente vengono indicate le immagini Fabric da installare, compresi tutti i binari utili per la configurazione della rete, lo sviluppo degli artefatti e i certificati. Infine, nel paragrafo 2.3 viene illustrato come sviluppare la rete in base alla configurazione dei file *.yaml* e l'utilizzo di tutti i binari citati precedentemente. Inoltre, seppur la rete blockchain realizzata con questa guida rappresenta una versione base rispetto a quella definita nell'allegato D3.2 per il caso Carpigiani, risulta comunque possibile ricrearla apportando le dovute modifiche alle configurazioni nei file indicati. Ad esempio, come verrà mostrato successivamente, si preferisce rappresentare il servizio di ordering di tipo "**Solo"** per una maggiore chiarezza e semplicità, piuttosto che il servizio di tipo Raft usato per Carpigiani.

3.1. Prerequisiti Fabric

Per la creazione della rete Fabric, è necessario dapprima installare tutti i software necessari per il corretto funzionamento del software Hyperledger. Nello specifico, per l'ambiente Linux, dovranno essere installati:

- l'ultima versione di git;
- l'ultima versione di cURL;
- la versione 17.06.2-ce o successiva di docker e docker-compose;
- la versione 1.11.x di Go o successiva, se si desidera scrivere il chaincode nel medesimo linguaggio di programmazione di Hyperledger;
- l'ultima release stabile di Node.JS Runtime e di Npm.

Oltre all'installazione, è necessaria la corretta configurazione dei suddetti tool affinché possano essere utilizzati da Fabric. Di seguito verranno quindi esposte le procedure per l'installazione e la configurazione di Go, poiché i passaggi da seguire risultano più complessi. Per quanto riguarda gli altri tool, invece, è sufficiente installarli mediante apt.

Per installare Go è necessario scaricare l'ultima versione dei file compilati dal sito ufficiale ed estrarli dall'archivio compresso con il comando:

```
$ tar xvzf Gofile.tar.gz
```

nella cartella di destinazione: */usr/local/go*; in seguito, è necessario impostare la variabile di ambiente PATH nel file *~/.bashrc* aggiungendo, in fondo al contenuto presente nel file, la riga:

```
$ export PATH=$PATH:/usr/local/go/bin
```

3.2. Installazione binari e immagini docker dei componenti Fabric

In questo paragrafo si procede con l'installazione dei file binari utili nelle fasi di implementazione della rete. Per queste operazioni è possibile utilizzare lo strumento Open Source cURL al fine di scaricare i file di esempio, disponibili sul repository di Hyperledger Fabric, eseguendo il seguente comando sul terminale:

```
$ curl -sSL http://bit.ly/2ysbOFE | bash -s
```

Se alternativamente si desidera scaricare una versione specifica, è possibile passare come parametro del comando un identificatore di versione per le immagini docker di Fabric e Fabric-CA (Certification Authority) come mostrato nel seguente comando di esempio:

```
$ curl -sSL http://bit.ly/2ysbOFE | bash -s -- 2.2.4 1.5.2
```

Se si dovesse riscontrare un errore durante l'esecuzione del comando precedente, la causa potrebbe essere dovuta all'utilizzo di una versione di cURL troppo vecchia che non è in grado di gestire i reindirizzamenti, oppure può essere dovuto a un ambiente non supportato.

Per poter utilizzare i binari forniti da Fabric, senza dover specificare completamente il percorso di ciascun binario, è possibile aggiungere alla variabile di ambiente "PATH" la loro posizione nel file system (path), eseguendo sul terminale il seguente comando:

```
$ export PATH=<percorso del download>/bin:$PATH
```

Altrimenti sarebbe possibile aggiungere il suddetto comando nel file *.bashrc,* presente nella directory *home,* e, per rendere effettivo i cambiamenti, eseguire il comando:

```
$ exec $SHELL
```

Questo permette (anche dopo il riavvio del dispositivo) di eseguire i file binari senza che vengano indicati i relativi percorsi.

3.3. Sviluppo della rete

Raggiunto questo punto dell'installazione, si procede con lo sviluppo della rete vera e propria.

Il primo passo consiste in alcune procedure di configurazione inerenti la struttura della rete e la gestione dei certificati, che richiedono rispettivamente la modifica dei file **configtx.yaml** e **crypto-gen.yaml**.

Nel file *configtx.yaml* vengono inserite le configurazioni inerenti ai seguenti artefatti di Fabric:

- orderer genesis block;
- channel configuration transaction;
- anchor peer transaction.

Il *blocco di genesis*, utilizzato dal servizio di ordinamento e il file di *transazione di configurazione del canale*, verranno trasmessi agli orderer nel momento in cui avviene la creazione del canale.

Le *anchor peer transaction*, specificano i dettagli del peer di ancoraggio di ogni organizzazione sul canale appena creato e sono indispensabili per fornire visibilità alla rete formata dai peer appartenenti all'organizzazione.

Successivamente si procede con la creazione dei file crittografici, come ad esempio certificati e chiavi. Per questo scopo viene utilizzato il file binario chiamato **cryptogen**. Quest'ultimo prende come parametro d'ingresso il file *crypto-config.yaml* che contiene la descrizione della topologia della rete; tramite l'invocazione del binario si effettua la generazione dei certificati x.509 e delle chiavi private per le varie entità che partecipano alla rete. Nel medesimo file vengono anche indicati gli hostname e i domini delle due organizzazioni e dell'orderer.

Nella sezione mostrata in Figura 1, vengono indicati: il peer dell'organizzazione, l'orderer e l'utente. Inoltre, anche se non è specificato nel file, viene sempre creato un utente amministratore (Admin). La chiave **EnableNodeOUs,** viene invece utilizzata per assegnare differenti permessi ai membri dell'organizzazione. Nella sezione **Template** viene indicato il numero di peer da creare per l'organizzazione, partendo dal primo che verrà nominato **peer0** e incrementando, sulla base dello stesso formato identificativo, il numero fino alla quantità richiesta. Esiste anche un parametro facoltativo, chiamato **Start**, che permette di definire l'indice da cui partire per assegnare i nomi ai peer (es. peer5, peer6, etc).

```
# ORDERER
OrdererOrgs:
  - Name: Orderer
    Domain: example.com
    Specs:
      - Hostname: orderer
# PEERS
PeerOrgs:
  - Name: SampleOrg
    Domain: sampleorg.example.com
    EnableNodeOUs: true
    Template:
      Count: 1
      SANS:
        - localhost
    Users:
      Count: 1
```

Figura 1 - Esempio del file crypto-config.yaml per un orderer e l'organizzazione SampleOrg.

Al termine della configurazione del file, viene eseguito il binario *cryptogen* per ottenere i certificati. Un esempio di come invocare il binario tramite console è mostrato nello snippet di codice sottostante:

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

Al termine dell'esecuzione del comando, sul terminale dovrebbe essere mostrato il seguente risultato:

```
$ sampleorg.example.com
```

I file relativi ai certificati e alle chiavi, cioè il materiale MSP (Membership Service Provider), sono collocati nella directory **/crypto-config**.

In seguito alla generazione dei materiali crittografici, risulta necessario eseguire il tool **configtxgen** al fine di generare il contenuto degli artefatti sulla base delle configurazioni presenti nel file configtx.yaml che viene fornito al binario in input.

Nella Figura 2 viene mostrato un esempio del file *configtx.yaml* configurato per un'organizzazione chiamata **SampleOrg** e che fa uso di un solo nodo per il servizio di ordering.

Con il simbolo "&" viene indicata la sezione relativa alle organizzazioni inerenti a **SampleOrg**. In questa vengono inseriti i seguenti parametri:

- il nome;
- l'id;
- la cartella in cui ci sono collocati tutti i certificati inerenti all'organizzazione (MSPDir);
- la sezione delle policy in cui viene specificato chi può firmare in caso di lettura, scrittura, approvazione e per tutte le operazioni da amministratore;
- le informazioni relative all'orderer di riferimento;
- le informazioni relative all'anchor peer per la comunicazione tra i nodi della stessa organizzazione e tra organizzazioni diverse.

La parola chiave *SkipAsForeign* risulta invece utile quando un amministratore appartenente a una singola organizzazione, senza il permesso di accedere alle directory MSP di altre organizzazioni, desidera creare un canale; questa impostazione non risulta però utile nel nostro caso poiché non ci sono organizzazioni con questa limitazione.

```
Organizations:
    - &SampleOrg
        Name: SampleOrg
        SkipAsForeign: false
        ID: SampleOrg
        # MSPDir is the filesystem path which contains the MSP configuration.
        MSPDir: <./crypto-config/path_to_msp_folder>
        #   /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
        Policies: &SampleOrgPolicies
            Readers:
                Type: Signature
                Rule: "OR('SampleOrg.member')"
            Writers:
                Type: Signature
                Rule: "OR('SampleOrg.member')"
            Admins:
                Type: Signature
                Rule: "OR('SampleOrg.admin')"
            Endorsement:
                Type: Signature
                Rule: "OR('SampleOrg.member')"
        OrdererEndpoints:
            - "127.0.0.1:7050"
        # AnchorPeers used for cross-org Gossip communication.
        AnchorPeers:
            - Host: 127.0.0.1
              Port: 7051
```

Figura 2 - Esempio del file `configtx.yaml` inerente alla sezione delle Organizations.

Successivamente, come mostrato in Figura 3, risulta necessario indicare le Capabilities, cioè le versioni dei componenti compatibili con la piattaforma. A riguardo, poiché l'unico nodo di ordinazione esegue Fabric in versione 2.x, viene configurata la capacità del canale del sistema di ordinazione su **V2_0**. Il canale di sistema utilizza i criteri predefiniti presenti nella sezione Channel e stabilisce a **V2_0** il livello di capacità del canale. Quest'ultimo utilizza quindi le funzionalità **V2_0** dell'applicazione e si avvale dei criteri predefiniti presenti nella sezione **Application** per stabilire le modalità di interazione tra i peer delle organizzazioni e il canale.

In Figura 4, viene mostrata la sezione inerente le Policy a livello applicativo. In questa sezione vengono definiti i permessi per: il ciclo di vita della fase di approvazione (LifecycleEndorsement), approvazione, lettura, scrittura e amministrazione. Con il termine **Majority** si rappresenta la necessità di raggiungere la maggioranza assoluta per l'approvazione delle operazioni richieste. Il termine **Any**, invece, esprime la necessità di una sola firma di qualsiasi organizzazione come minimo necessario per l'approvazione.

La sottosezione delle **Capabilities** descrive il livello delle capacità dell'applicazione, in questo caso vengono utilizzate le ApplicationCapabilities configurate precedentemente (Figura 3).

```
Capabilities:
    # Channel capabilities apply to both the orderers and the peers
    Channel: &ChannelCapabilities
        # Prior to enabling V2.0 channel capabilities, ensure that all orderers and
        # peers on a channel are at v2.0.0 or later.
        V2_0: true
    # Orderer capabilities apply only to the orderers, and may be safely used with prior
    # release peers.
    Orderer: &OrdererCapabilities
        # Prior to enabling V2.0 orderer capabilities, ensure that all orderers on a
        # channel are at v2.0.0 or later.
        V2_0: true
    # Application capabilities apply only to the peer network, and may be safely used
    # with prior release orderers.
    Application: &ApplicationCapabilities
        # V2.0 for Application enables the new non-backwards compatible features and
        # fixes of fabric v2.0. Prior to enabling V2.0 orderer capabilities, ensure that
        # all orderers on a channel are at v2.0.0 or later.
        V2_0: true
```

Figura 3 -Esempio del file configtx.yaml inerente alla sezione delle versioni delle Capabilities.

```
    #   /Channel/Application/<PolicyName>
    Policies: &ApplicationDefaultPolicies
        LifecycleEndorsement:
            Type: ImplicitMeta
            Rule: "MAJORITY Endorsement"
        Endorsement:
            Type: ImplicitMeta
            Rule: "MAJORITY Endorsement"
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"
    # Capabilities describes the application level capabilities, see the
    # dedicated Capabilities section elsewhere in this file for a full description
    Capabilities:
        <<: *ApplicationCapabilities
```

Figura 4 - Esempio del file configtx.yaml inerente alla sezione delle Policies delle Capabilities.

Nella sezione relativa agli orderer, analogamente a quanto fatto per i peer, vengono inserite le informazioni per la configurazione del servizio di ordering. Come già discusso nei documenti tecnici D3.1 e D3.2, Fabric supporta tre tipologie di servizio di ordering; per questo esempio viene però mostrata solamente la tipologia **SOLO**, come visibile in Figura 5.

Nella sottosezione **Addresses** è possibile non inserire alcun dato e specificare le informazioni da utilizzare durante la fase di definizione dell'organizzazione, come mostrato in precedenza in Figura 2.

La parola chiave *BachTimeout* indica dopo quanti secondi avviene la creazione del blocco. Se durante questo intervallo di tempo stabilito vengono ricevute poche transazioni, il blocco viene comunque chiuso allo scadere del tempo stabilito; se invece il sistema dovesse ricevere troppe transazioni, chiuderebbe il blocco prima del termine del timeout, ovvero appena viene raggiunto il limite impostato nel parametro *BatchSize* che rappresenta il numero massimo di messaggi o la quantità in byte massima raggiungibile.

Il campo *MaxChannels* è impostato al valore zero al fine di indicare la possibilità per gli orderer di risiedere in numero illimitato di canali.

```
Orderer: &OrdererDefaults
    # Available types are "solo", "kafka" and "etcdraft".
    OrdererType: solo
    Addresses:
        # - 127.0.0.1:7050
    BatchTimeout: 2s
    BatchSize:
        MaxMessageCount: 500
        AbsoluteMaxBytes: 10 MB
        PreferredMaxBytes: 2 MB
    MaxChannels: 0
```

Figura 5 - Esempio del file configtx.yaml inerente alla sezione dell'Orderer.

Nella Figura 6 viene rappresentata la sezione inerente alle politiche degli orderer. Oltre ai casi di lettura, scrittura e amministrazione, vengono indicate le politiche di validazione dei blocchi. In questo esempio viene mostrato che gli orderer accettano la firma di qualsiasi scrittore.

OrdererCapabilities, come già descritto in Figura 3, indica invece le capacità a livello degli orderer.

```
    # /Channel/Orderer/<PolicyName>
    Policies:
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"
        BlockValidation:
            Type: ImplicitMeta
            Rule: "ANY Writers"
    Capabilities:
        <<: *OrdererCapabilities
```

Figura 6 - Esempio del file configtx.yaml inerente alla sezione delle versioni delle Policies dell'Orderer.

Nella Figura 7 vengono rappresentate le politiche del canale con la stessa semantica utilizzata per la figura precedente.

```
Channel: &ChannelDefaults
    # /Channel/<PolicyName>
    Policies:
        # Who may invoke the 'Deliver' API
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        # Who may invoke the 'Broadcast' API
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        # By default, who may modify elements at this config level
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"
    Capabilities:
        <<: *ChannelCapabilities
```

Figura 7 - Esempio del file configtx.yaml inerente alla sezione delle Policies del canale.

Nella Figura 8 sono indicati tutti i profili necessari per l'impostazione finale della rete riportando le politiche e le configurazioni descritte in precedenza. Il profilo, in sintesi, indica quale configurazione considerare tra quelle disponibili. In questo esempio viene descritto il profilo **SampleSingleMSPSolo** che fa riferimento alle politiche di:

- **ChannelDefaults**, per quanto riguarda i canali;
- **OrdererDefaults**, inerente agli orderer;
- **SampleOrg**, per quanto riguarda l'organizzazione.

Il consorzio rappresenta l'intera rete in cui, inizialmente, è presente la sola organizzazione descritta in precedenza.

Nell'esempio mostrato il profilo **SampleSingleMSPSolo** viene quindi utilizzato per la creazione del system channel e del genesis block.

Il system channel definisce i nodi appartenenti al servizio di ordinamento e l'insieme di organizzazioni che risultano amministratori sempre di tale servizio. Il canale di sistema include anche un insieme di organizzazioni peer che appartengono al consorzio blockchain. Il canale **MSP** di ciascun membro del consorzio viene quindi incluso nel canale di sistema, consentendo a ciascun componente di creare nuovi canali di applicazione e aggiungere altri partecipanti, facenti parte del consorzio, al nuovo canale.

Il blocco di genesi dell'orderer e i successivi artefatti creati in un secondo momento, sono memorizzati nella cartella */channel-artifacts*.

L'uso del profilo **SampleSingleMSPSolo** prevede la creazione di un consorzio denominato **SampleConsortium**, questo contiene l'organizzazione peer **SampleOrg** che era stata descritta in precedenza all'interno del file *configtx.yaml.*

La sezione **Orderer** del profilo, specifica di utilizzare il servizio di ordinazione Solo a nodo singolo, come definito nella sezione Orderer.

La sezione **Orderer → Organization → SampleOrg** specifica quale organizzazione è definita come amministratore del servizio di ordinazione.

L'inserimento del profilo **SampleOrgChannel** è invece necessario per permettere la creazione del canale mediante il file di configurazione e in questo viene indicato il nome del consorzio.

Nell'esempio mostrato viene chiamato **SampleConsortium**, ospitato dal canale di sistema della rete. Di conseguenza, il servizio di ordinazione definito nel profilo **SampleSingleMSPSolo** diventa il gruppo di consenso del canale.

Infine, nella sezione **Application**, l'organizzazione relativa al consorzio viene inclusa come **membro** del canale.

```
Profiles:
    # SampleSingleMSPSolo defines a configuration which uses the Solo orderer,
    # and contains a single MSP definition (the MSP sampleconfig).
    # The Consortium SampleConsortium has only a single member, SampleOrg.
    SampleSingleMSPSolo:
        <<: *ChannelDefaults
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *SampleOrg
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *SampleOrg
    SampleOrgChannel:
        Consortium: SampleConsortium
        <<: *ChannelDefaults
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *SampleOrg
            Capabilities:
                <<: *ApplicationCapabilities
```

Figura 8 - Esempio del file configtx.yaml inerente alla sezione dei Profiles.

Al termine della configurazione del file è necessario invocare il file binario **cryptogen** per creare il blocco di genesi. L'invocazione del binario prevede il passaggio obbligatorio come parametro della posizione sul disco del file *configtx.yaml* appena modificato, risulta quindi utile impostare la variabile d'ambiente di Fabric eseguendo sul terminale il seguente comando:

```
$ export FABRIC_CFG_PATH=$PWD
```

Nel comando appena mostrato, *$PWD* rappresenta la cartella di lavoro in cui è presente anche il file binario da invocare.

Una volta eseguita l'istruzione precedente, si invoca il tool eseguendo a terminale il seguente comando:

```
$ ../bin/configtxgen -profile <nameOfProfile> -channelID <name-of-
sys-channel> -outputBlock ./channel-artifacts/genesis.block
```

Il parametro *<nameOfProfile>* è il nome del profilo creato nella sezione Profiles del file *configtx.yaml*. Nel caso specifico di questo esempio il valore è pari a **SampleSingleMSPSolo**. Il parametro *<name-of-sys-channel>* è invece il nome che si vuole assegnare al canale di sistema.

L'esecuzione del precedente comando dovrebbe fornire il seguente output:

```
2021-10-26 19:21:56.301 EDT [common/tools/configtxgen] main ->
INFO 001 Loading configuration
2021-10-26 19:21:56.309 EDT [common/tools/configtxgen]
doOutputBlock -> INFO 002 Generating genesis block
2021-10-26 19:21:56.309 EDT [common/tools/configtxgen]
doOutputBlock -> INFO 003 Writing genesis block
```

In seguito è necessario creare l'artefatto della transazione del canale, questo avviene invocando il binario **configtxgen** e fornendogli alcuni parametri in input quali il nome del profilo creato nella sezione Profiles del file *configtx.yaml*, il nome del file di output e l'ID relativi al canale. Il seguente snippet di codice mostra un esempio di esecuzione del comando:

```
$ export CHANNEL_NAME=channel  && ../bin/configtxgen -profile
SampleOrgChannel -outputCreateChannelTx ./channel-
artifacts/channel.tx -channelID channel1
```

L'esecuzione del precedente comando dovrebbe fornire il seguente output:

```
2017-10-26 19:24:05.324 EDT [common/tools/configtxgen] main ->
INFO 001 Loading configuration
2017-10-26 19:24:05.329 EDT [common/tools/configtxgen]
doOutputChannelCreateTx -> INFO 002 Generating new channel
configtx
2017-10-26 19:24:05.329 EDT [common/tools/configtxgen]
doOutputChannelCreateTx -> INFO 003 Writing new channel tx
```

Successivamente è necessario eseguire il comando mostrato nello snippet seguente per tutte le organizzazioni presenti a sistema, al fine di creare gli anchor peer necessari al corretto funzionamento della rete; l'esecuzione del binario prevede di fornire alcuni parametri in input come il nome del profilo, il file di output degli anchor peer, l'id del canale e il nome dell'organizzazione:

```
$ ../bin/configtxgen -profile SampleOrgChannel -
outputAnchorPeersUpdate ./channel-artifacts/SampleOrgMSPanchors.tx
-channelID channel1 -asOrg SampleOrgMSP
```

Infine è possibile avviare la rete mediante docker-compose come mostrato nel seguente comando:

```
$ docker-compose -f docker-compose-cli.yaml up -d
```

Nel comando appena mostrato il parametro **-d** permette di evitare che venga stampato sul terminale tutto il log relativo ai nodi.

Il file *docker-compose-cli.yaml* presente nel comando precedente, viene utilizzato per configurare: le variabili d'ambiente, i volumi, i container, le immagini docker da usare e altri parametri inerenti ai componenti Fabric che costituiscono la rete. Un esempio di tale file è mostrato in Figura 9 in cui, inizialmente, viene indicata la versione del linguaggio *yaml* che si vuole utilizzare per il resto del file. Difatti, è necessario effettuare tale operazione perché differenti versioni usano sintassi diverse. In questo manuale è stata è stata utilizzata la versione 3.7.

Nella sezione *volumes* del medesimo file sono indicati i nomi dei file system che verranno salvati sulla memoria non volatile del dispositivo che ospita i container docker, così da rendere persistenti i componenti di Fabric anche in caso di crash o riavvio della macchina sulla quale sono ospitati.

Infine la sezione *networks* viene utilizzata per specificare il nome della rete Fabric a cui i container dovranno agganciarsi per poter comunicare correttamente.

```
version: '3.7'
volumes:
  orderer.example.com:
  peer0.sampleorg.example.com:
networks:
  test:
    name: fabric_test
```

Figura 9 - Esempio del file docker-compose-cli.yaml (parte 1 - volumi e reti).

In Figura 10 viene illustrato il prosieguo del file *docker-compose-cli.yaml*. Nello specifico viene mostrata la sezione dei servizi, i quali rappresentano i nodi Fabric (orderer e peer) partecipanti alla rete. Come primo elemento della sottosezione viene indicato l'orderer esplicitando alcuni parametri che lo caratterizzano come:

- il nome del container;
- la versione dell'immagine docker da utilizzare (latest in questo caso);
- il servizio;
- le variabili d'ambiente;
- la directory di lavoro principale;
- il comando per eseguire l'orderer;
- i volumi (cioè i file da caricare nel container);
- le porte in cui l'orderer rimane in ascolto;
- la rete a cui l'orderer deve agganciarsi.

Riguardo la sezione inerente alle variabili d'ambiente, queste sono necessarie per effettuare la configurazione del nodo. Nel caso specifico di questo esempio si impostano:

- l'MSP per l'identità;
- l'indirizzo e la porta di ascolto;
- viene abilitato il TLS per la comunicazione cifrata tra i nodi;
- le variabili inerenti all'amministrazione;
- la variabile operations per le RESTful API.

Come già esposto per l'orderer, anche per il peer risulta necessario esplicitare le opportune variabili d'ambiente per la loro configurazione. La Figura 11 espone in dettaglio il file utilizzato per questo esempio e per il peer viene richiesto di specificare anche altre informazioni come il **CORE_PEER_ID**, il **CORE_PEER_CHAINCODEADDRESS** e il **CORE_PEER_GOSSIP** per la comunicazione tra i nodi della stessa organizzazione e/o canale.

```
services:
  orderer.example.com:
    container_name: orderer.example.com
    image: hyperledger/fabric-orderer:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_LOGGING_SPEC=INFO
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_LISTENPORT=7050
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
      # enabled TLS
      - ORDERER_GENERAL_TLS_ENABLED=true
      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
      - ORDERER_GENERAL_CLUSTER_CLIENTCERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      - ORDERER_GENERAL_CLUSTER_CLIENTPRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      - ORDERER_GENERAL_CLUSTER_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
      - ORDERER_GENERAL_BOOTSTRAPMETHOD=none
      - ORDERER_CHANNELPARTICIPATION_ENABLED=true
      - ORDERER_ADMIN_TLS_ENABLED=true
      - ORDERER_ADMIN_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      - ORDERER_ADMIN_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      - ORDERER_ADMIN_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
      - ORDERER_ADMIN_TLS_CLIENTROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
      - ORDERER_ADMIN_LISTENADDRESS=0.0.0.0:7053
      - ORDERER_OPERATIONS_LISTENADDRESS=0.0.0.0:17050
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
    command: orderer
    volumes:
        - ./channel-
artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block
        - ./crypto-
config/ordererOrganizations/example.com/orderers/orderer.example.com/msp:/var/hyperledger
/orderer/msp
        - ./crypto-
config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/:/var/hyperledge
r/orderer/tls
        - orderer.example.com:/var/hyperledger/production/orderer
    ports:
      - 7050:7050
      - 7053:7053
      - 17050:17050
    networks:
      - test
```

Figura 10 - Esempio del file docker-compose-cli.yaml (parte 2 - peer).

```
 peer0.sampleorg.example.com:
    container_name: peer0.sampleorg.example.com
    image: hyperledger/fabric-peer:latest
    labels:
      service: hyperledger-fabric
    environment:
      #Generic peer variables
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=fabric_test
      - FABRIC_LOGGING_SPEC=INFO
      #- FABRIC_LOGGING_SPEC=DEBUG
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_PROFILE_ENABLED=false
      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
      - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
      # Peer specific variables
      - CORE_PEER_ID=peer0.sampleorg.example.com
      - CORE_PEER_ADDRESS=peer0.sampleorg.example.com:7051
      - CORE_PEER_LISTENADDRESS=0.0.0.0:7051
      - CORE_PEER_CHAINCODEADDRESS=peer0.sampleorg.example.com:7052
      - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
      - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.sampleorg.example.com:7051
      - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.sampleorg.example.com:7051
      - CORE_PEER_LOCALMSPID=SampleOrgMSP
      - CORE_OPERATIONS_LISTENADDRESS=0.0.0.0:17051
    volumes:
        - /var/run/docker.sock:/host/var/run/docker.sock
        - ./crypto-
config/peerOrganizations/sampleorg.example.com/peers/peer0.sampleorg.example.com/msp:/etc
/hyperledger/fabric/msp
        - ./crypto-
config/peerOrganizations/sampleorg.example.com/peers/peer0.sampleorg.example.com/tls:/etc
/hyperledger/fabric/tls
        - peer0.sampleorg.example.com:/var/hyperledger/production
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: peer node start
    ports:
      - 7051:7051
      - 17051:17051
    networks:
      - test
```

Figura 11 - Esempio del file docker-compose-cli.yaml (parte 3 - orderer).

Infine, nella Figura 12 viene rappresentato il tool "*Command Line Interface*" (CLI), utile per effettuare operazioni comunicando direttamente con il peer al quale si desidera agganciare. Per completare la configurazione della CLI e abilitare la comunicazione, è sufficiente definire le variabili di ambiente in modo corretto. La Figura 12 mostra il file utilizzato per questo esempio.

La Figura 13 illustra invece un esempio di variabili d'ambiente appositamente configurate per abilitare la comunicazione con il **peer0** appartenente all'organizzazione **SampleOrg**.

```
  cli:
    container_name: cli
    image: hyperledger/fabric-tools:latest
    labels:
      service: hyperledger-fabric
    tty: true
    stdin_open: true
    environment:
      - GOPATH=/opt/gopath
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - FABRIC_LOGGING_SPEC=INFO
      #- FABRIC_LOGGING_SPEC=DEBUG
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: /bin/bash
    volumes:
        - ./crypto-
config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto-config
    depends_on:
      - peer0.sampleorg.example.com
    networks:
      - test
```

Figura 12 - Esempio del file docker-compose-cli.yaml (parte 4 - cli).

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypt
o/peerOrganizations/sampleorg.example.com/users/Admin@sampleorg.example.com/msp
CORE_PEER_ADDRESS=peer0.sampleorg.example.com:7051
CORE_PEER_LOCALMSPID="SampleOrgMSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/c
rypto/peerOrganizations/sampleorg.example.com/peers/peer0.sampleorg.example.com/
tls/ca.crt
```

Figura 13 - Esempio di variabili d'ambiente impostate nella CLI per comunicare con il peer0 di SampleOrg.

Terminati tutti i passaggi descritti fino ad ora, è necessario eseguire il comando *docker-compose,* indicato precedentemente, per avviare la rete. In questo caso, verranno attivati: il peer, l'orderer e il CLI. Per verificare che tutti i container relativi ai componenti precedentemente indicati siano stati configurati e avviati correttamente, eseguire il comando *docker ps* come mostrato in Figura 14 in cui viene mostrato l'elenco dei componenti.

```
NAMES                       IMAGE                               COMMAND             STATUS
cli                         hyperledger/fabric-tools:latest     "/bin/bash"         Up 5 seconds
orderer.example.com         hyperledger/fabric-orderer:latest   "orderer"           Up 7 seconds
peer0.sampleorg.example.com hyperledger/fabric-peer:latest      "peer node start"   Up 7 seconds

...
```

Figura 14 - Risultato ottenuto eseguendo il comando "docker ps".

3.4. Esempio di utilizzo della piattaforma

In questa sezione vengono mostrati due esempi operativi su come poter utilizzare la rete appena creata. Il primo mostra come ottenere le informazioni dal ledger inerenti a tutte le operazioni svolte dalle macchine da gelato, mentre il secondo riguarda l'installazione di uno smart contract.

3.4.1. Visualizzazione dei refill di una macchina

Nel menù laterale cliccare sul simbolo "Refills" (Figura 15);

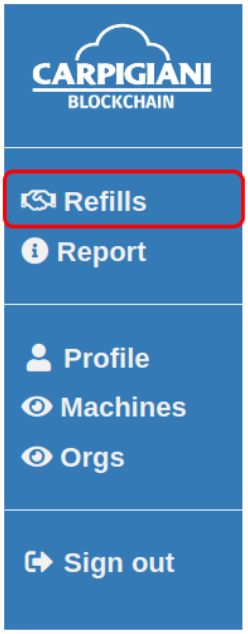


Figura 15 - Click su "Refills" della sidebar dell'utente standard.

1. selezionare la macchina di cui si vuole visionare i refill scrivendo il nome nel campo "Choose a machine" (sarà sufficiente inserire le prime lettere e automaticamente verranno suggerite le corrispondenze presenti nella rete);
2. selezionare il canale di appartenenza della macchina di cui si vuole visionare i refill scrivendo il nome del canale nel campo "Choose a channel" (anche in questo caso sarà sufficiente inserire le prime lettere e automaticamente verranno suggerite le corrispondenze presenti nella rete);
3. selezionare una finestra temporale di dati nel menu a tendina "Choose how many days" (i valori possibili sono All/Today/Week/Month);
4. selezionare il giorno di inizio e di fine della selezione dei refill per la macchina desiderata inserendo la data rispettivamente nei campi "Start calendar" e "End calendar";
5. cliccare sul pulsante "Search" (Figura 16);

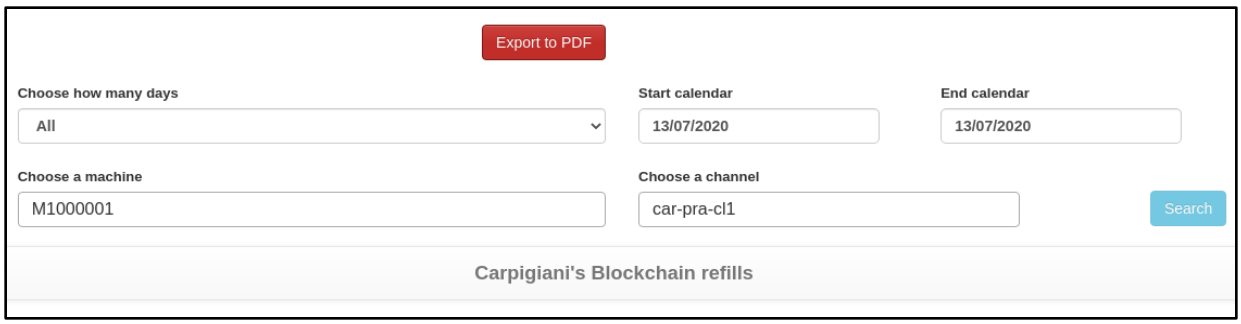


Figura 16 - Schermata Refill compilata per ottenere i refill della macchina selezionata

6. nella parte inferiore della pagina (sotto al titolo "Carpigiani's Blockchain refills") compariranno i refill che corrispondono ai parametri impostati (Figura 17);

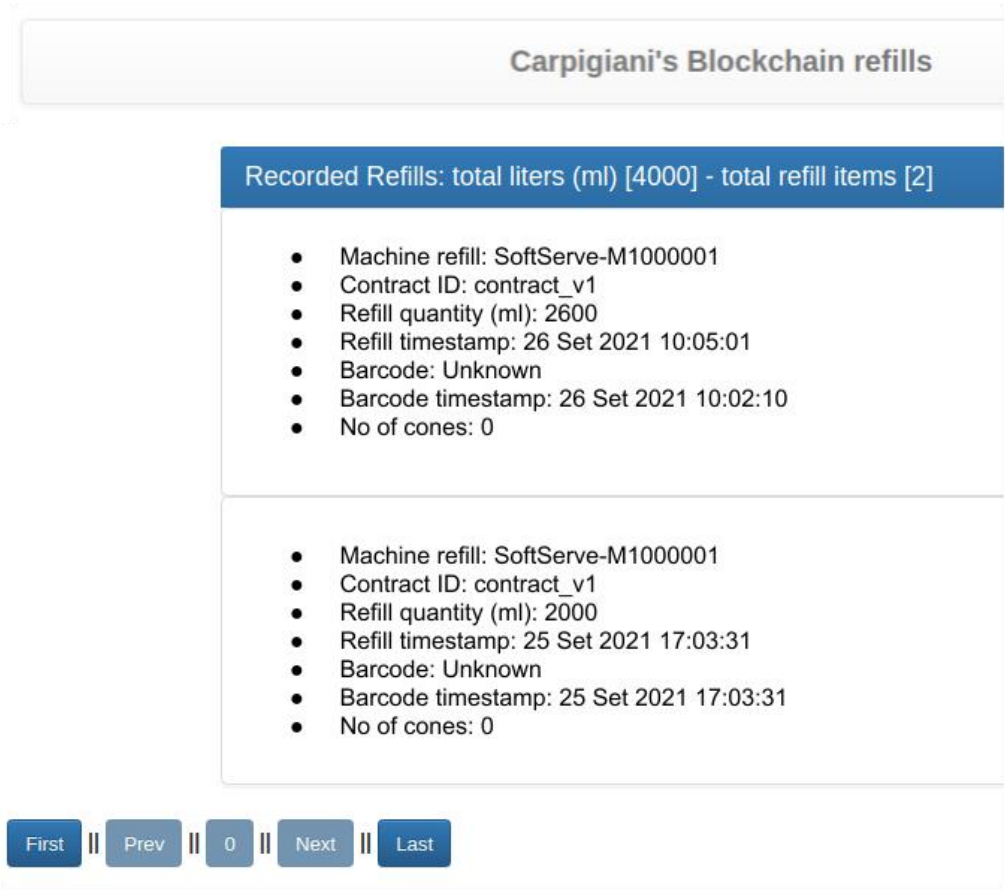


Figura 17 - Refill della macchina SoftServe-M1000001 ottenuti.

7. Se il risultato dovesse contenere troppi refill, è possibile cliccare sui pulsanti appositi per cambiare pagina e visionare le successive transazioni.

3.4.2. [solo amministratore] Installare o aggiornare un nuovo smart contract su un canale

1. nel menu laterale dell'amministratore cliccare sul simbolo della blockchain (Figura 18);

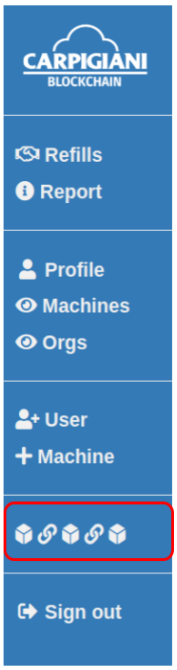


Figura 18 - Click sul tasto blockchain della sidebar dell'amministratore.

2. cliccare nuovamente sul menu laterale di Fabric, sul simbolo "Install SC" (Figura 19);

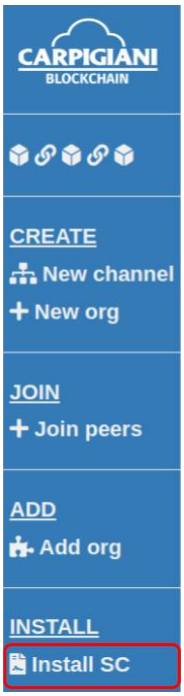


Figura 19 - click su "Install SC" della sidebar di Fabric.

3. selezionare il canale, inserire il nome del chaincode e la versione (Figura 20);
4. cliccare sul pulsante “Create”.

Figura 20 - Form compilato per l’installazione dello smart contract.