Università degli Studi di Palermo
Corso di Laurea in Informatica Magistrale (LM 18)

Relazione tecnica di progetto
“Hyperledger Energy System”

Studente:
Giulia Maraventano

Docente:
Prof. Pierluigi Gallo

A.A. 2020/2021

PREMESSA

Il progetto descritto in questa relazione ha l'obiettivo di simulare, su piccola scala, la gestione di una domotica basata sul monitoraggio e il controllo dei carichi elettrici.
Il contesto considerato è quello del trading energetico, ovvero lo scambio di energia elettrica prodotta da privati (prosumers).
Utilizzando la tecnologia blockchain oggi è possibile che la comunicazione tra produttore e consumatore avvenga integralmente in sicurezza.
Nella presente simulazione il fornitore di energia ha il controllo della soglia di energia dei carichi del consumatore.
Superata la soglia di un carico il produttore può inviare un segnale al dispositivo del consumatore per spegnerlo, e un segnale per riaccenderlo se il valore torna sotto la soglia.

HYPERLEDGER FABRIC

Hyperledger Fabric è una piattaforma DLT (distributed ledger technology) opensource, creata dalla Linux-Foundation, per lo sviluppo di applicazioni software all'interno di una rete protetta e sicura, perché basata sulla tecnologia Blockchain.
In questo contesto, viene utilizzata una rete distribuita e "permissioned", ovvero una rete composta di nodi, ognuno dei quali possiede le stesse caratteristiche e gli stessi permessi. Si tratta quindi di una rete privata e sicura, che garantisce privacy, integrità e confidenzialità, a cui possono partecipare solo coloro che si autenticano con un certificato.
Questa rete possiede un'architettura modulare e dettagliatamente configurabile secondo le esigenze del caso.

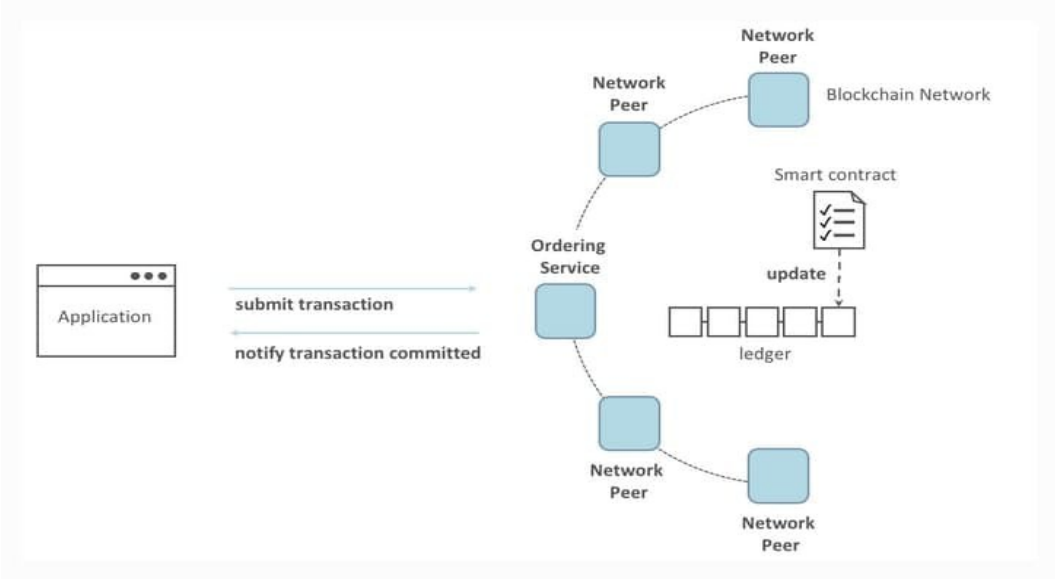

Figura 1: Struttura della Hyperledger Fabric.

I componenti della Hyperledger Fabric sono i seguenti:

- Servizio di Ordine, che si occupa di ordinare le transazioni e stabilire il consenso, inviando i blocchi ai peer.
- MSP (Membership Service Provider), che associa le entità della rete a identità digitali cifrate.
- Contratti Intelligenti, che girano all'interno di un Docker a parte per sicurezza.
- Libro mastro (Ledger), definito da istruzioni per la gestione (lettura/scrittura) degli asset.

- API (Application Programming Interface), che permettono di interfacciare alla rete diverse applicazioni proprietarie, che utilizzano altri linguaggi di programmazione.

La H.F. creata in questo progetto si chiama **energy** ed è all'interno dell'omonima cartella, che è strutturata nel seguente modo:

- **energy**
  - **/organizations**
    - **/produttore**
      - **/application**
        - package.json (file di configurazione)
        - script.js (applicazione che invoca i chaincode)
      - **/contract**
        - **/lib (chaincodes)**
        - index.js (main script)
        - package.json (file di configurazione)
    - **/consumatore**
      - **/application**
        - package.json (file di configurazione)
        - script.js (applicazione che invoca i chaincode)
      - **/contract**
        - **/lib (chaincodes)**
        - index.js (main script)
        - package.json (file di configurazione)
  - net-start.sh (avvio della catena)
  - net-stop.sh (stop della catena)

BLOCKCHAIN NETWORK

Quando si crea una nuova rete, al suo interno viene configurato un canale, in base agli accordi e alle regole stabilite dalle organizzazioni che ne devono far parte. Questo canale è contenuto nel "configuration block".
Successivamente, le organizzazioni (produttore e consumatore) che si collegheranno a questo canale, devono essere identificate con la creazione di un CA (Certificate Authority). In questo modo avranno il permesso di collegare ognuno il proprio peer o il nodo del servizio di ordine. Tutti i nodi conservano una copia del libro mastro (Ledger) del canale che viene ogni volta aggiornato con un nuovo blocco di Assets (insieme di coppie chiave-valore che identificano lo stato di un'entità).
Per avviare l'intera rete, configurando i certificati CA e le identità delle due organizzazioni (produttore, consumatore), bisogna lanciare il seguente script:

net-start.sh

All'interno di questo script bash vengono lanciati i due comandi fondamentali per avviare la rete blockchain, con la creazione dei docker containers delle due organizzazioni e dell'orderer service, la generazione dei rispettivi CA, la

creazione di un canale per le transazioni e il successivo collegamento dei peer di ogni organizzazione al canale:

./network.sh up createChannel -ca

Con il comando ***docker ps -a*** si può visualizzare la struttura della rete.
All'interno delle rispettive cartelle delle due organizzazioni (produttore e consumatore) ci sono due sottocartelle (contract e application), una contenente lo smart-contract (chaincode), comune ad entrambe le organizzazioni, e l'altra contenente l'applicazione privata che si interfaccia con la rete invocando i comandi stabiliti nel chaincode.

SMART-CONTRACT

Una volta avviata e configurata la rete, bisogna creare uno smart-contract (chaincode), definito dalle regole e dalle funzionalità stabilite a priori, in modo che ogni organizzazione possa effettuare le transazioni invocando ed eseguendo i comandi sul proprio peer, dove viene successivamente firmato l'output della transazione e distribuito agli altri peer, i quali, se lo considerano attendibile, lo confermano sul registro.
La policy che specifica le organizzazioni impostate sul canale che devono eseguire lo smart contract viene definita policy di approvazione.
I chaincode creati sono: **energycontract.js**, contenente tutte le policy generali, e **energy.js**, dove viene gestito lo stato dell'entità Energia (EROGATA, INTERROTTA).
Gli asset principali sono i seguenti:

```
async CreateAsset(ctx, id, color, size, owner, power) {
        const asset = {
            ID: id,
            Owner: owner,
            power: power,
        };
        await ctx.stub.putState(id, Buffer.from(JSON.stringify(asset)));
        return JSON.stringify(asset);
    }

async Eroga(ctx, id, data, power) {

        let energy = Energy.createInstance(id, data, parseInt(power));
        energy.setErogata();
        if (power > 3000) {
            energy.setInterrotta();
        }
        return energy;
    }
```

In questa parte di codice c'è il comando che viene invocato per dell'erogazione di energia ai carichi elettrici del consumatore. Se la potenza richiesta è inferiore a 3000 Watt, lo stato di erogazione rimane attivo, altrimenti l'erogazione di energia viene interrotta.

Il comando per installare il contratto (chaincode) sui due peer e per distribuirlo sul canale, è contenuto nello script ***cc.sh,*** *che si occupa anche di configurare le variabili di ambiente*:

./network.sh deployCC -ccn energy -ccp ${DIR}/organization/produttore/contract -ccl javascript

Questo comando utilizza il ciclo di vita del chaincode per impacchettare, installare, interrogare il chaincode installato, approvare il chaincode per entrambe le organizzazioni e infine eseguire il commit del chaincode.

APPLICATION

L'applicazione Javascript che si interfaccia con il chaincode (app.js), invocandone i comandi per le transazioni, si trova sia nella directory del produttore, sia in quella del consumatore, poiché ognuno utilizza la propria interfaccia per interagire con il sistema.
Guardando l'**app.js** del produttore, il chaincode importante che viene invocato è il seguente:

```
await contract.evaluateTransaction('eroga', '00001', '07/07/2021', '350');
```

Conoscendo il rispettivo chaincode, descritto precedentemete, il valore della potenza inviato nella transazione (350) è inferiore a 3000, pertanto l'entità energia continua a essere nello stato **EROGATA (1)**.

```
*** Result: {
  "class": "org.energynet.Energy",
  "key": "600:00001",
  "currentState": 1,
  "energyNumber": "00001",
  "data": "07/07/2021",
  "potenza": 600
}
```

Quando superiore, lo stato dell'entità energia si aggiorna in **INTERROTTA (2)**.

```
*** Result: {
  "class": "org.energynet.Energy",
  "key": "3200:00001",
  "currentState": 2,
  "energyNumber": "00001",
  "data": "07/07/2021",
  "potenza": 3200
}
```

L'interfaccia web del sistema domotico del consumatore consiste in un semplice pannello per gestire l'accensione e lo spegnimento di alcuni carichi elettrici. Ogni carico elettrico consuma una quantità di energia, richiedendo una potenza ben precisa che si somma agli altri carichi accesi.

Appena il consumatore accende o spegne un carico, il valore della potenza richiesta cambia e viene effettuata una richiesta asincrona al produttore, il quale evoca il chaincode eroga per verificare il valore della potenza richiesta e stabilire se continuare o meno a erogare l'energia.

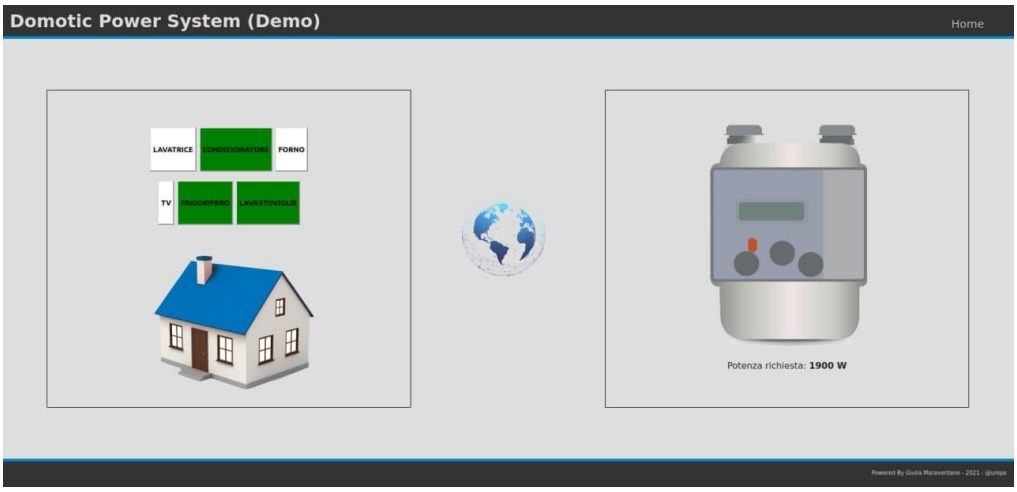

Figura 2: Interfaccia web che chiama l'applicazione .js ogni volta che varia il valore della somma delle potenze dei carichi elettrici.

RIFERIMENTI

https://hyperledger-fabric.readthedocs.io