Documentazione

Questo capitolo descrive in modo tecnico e dettagliato la funzione e l'interazione tra i file di configurazione, gli artefatti crittografici e gli script di automazione utilizzati per orchestrare una rete Hyperledger Fabric dockerizzata. Vengono inoltre indicati i criteri per la personalizzazione modulare dell'infrastruttura.
La documentazione è strutturata in questo modo:
Tipo: Indica se il file è di input (file modificabile e configurabile) o output (generato da script *bash*)
Descrizione: Breve descrizione del file: a cosa serve e come usarlo.
Path: Posizione all'interno del tree del progetto
Trigger per rigenerazione: Specifica quando deve essere rigenerato dagli script (solitamente in caso di modifica)
Script associato: Script che permette di generare il documento
Per configurare i file per la personalizzazione, è possibile consultare la documentazione ufficiale di *Hyperledger Fabric*

File statici e generabili

`crypto-config.yaml`

Tipo: input
Descrizione: definisce le entità organizzative della rete, inclusi MSP ID, domini e struttura dei peer. Viene elaborato da `cryptogen` per generare la gerarchia di certificati X.509.
Path: radice progetto o ./artifacts/
Trigger per rigenerazione: modifica nella topologia (nuove organizzazioni, peer, utenti).
Script associato: `generate-artifacts.sh`

`configtx.yaml`

Tipo: input
Descrizione: struttura le definizioni logiche della rete: capabilities, policies di

consenso, profili di generazione dei blocchi e transazioni di canale. Necessario per `configtxgen`.
Path: radice progetto o ./config/
Trigger per rigenerazione: cambio policy, aggiornamento capabilities, aggiunta organizzazioni o canali.
Script associato: `generate-artifacts.sh`

`genesis.block`

Tipo: output
Descrizione: blocco di sistema generato da `configtxgen`, utilizzato dal nodo orderer in fase di bootstrap.
Path: ./channel-artifacts/
Trigger per rigenerazione: cambio `configtx.yaml` o rigenerazione full-stack.
Script associato: `generate-artifacts.sh`

`mychannel.tx`

Tipo: output
Descrizione: transazione di creazione del canale con riferimento a uno o più MSP autorizzati. Input per il comando `peer channel create`.
Path: ./channel-artifacts/
Trigger per rigenerazione: cambio nome canale o MSP.
Script associato: `generate-artifacts.sh`

`docker-compose.yaml`

Tipo: input runtime
Descrizione: file di orchestrazione Docker che definisce container peer, orderer, CLI e CA. Ogni servizio è associato a un set di volumi, env vars e porte esposte.
Path: root progetto
Trigger per rigenerazione: variazione container, nomi host, mapping porte o volumi.
Script associato: Configurabile in uno script es:`start-network.sh`

`mychannel.block`

Tipo: output
Descrizione: primo blocco del canale, restituito da `peer channel create`. Serve ai peer per effettuare il join su canale.
Path: `channel-artifacts/` nel container CLI
Trigger per rigenerazione: dopo nuova invocazione del comando `peer channel`

`create`
Script associato: `createChannel.sh`

Script automatizzati

`generate-artifacts.sh`

Funzione: orchestration iniziale della generazione di certificati e artefatti logici di rete. Esegue in sequenza:

1. `cryptogen generate -config=crypto-config.yaml`
2. `configtxgen -profile -channelID system-channel -outputBlock genesis.block`
3. `configtxgen -profile -channelID mychannel -outputCreateChannelTx mychannel.tx`

Precondizione: file di configurazione validi e directory target pulite.

`createChannel.sh`

Funzione: crea un canale su ordine dell'organizzazione CLI e registra i peer all'interno del canale.

1. `peer channel create` con `-outputBlock`
2. `peer channel join` su ciascun peer
3. `peer channel update` per gli anchor peers (opzionale)

Precondizione: container CLI attivo con MSP montato correttamente.

`deploy-chaincode.sh`

Funzione: pipeline per lifecycle del chaincode (Fabric v2+):

1. `peer lifecycle chaincode package`
2. `install` su ogni peer
3. `queryinstalled`, `approveformyorg`, `checkcommitreadiness`
4. `commit` finale sul canale
5. `querycommitted`

Precondizione: canale attivo, peer in stato "joined", env vars corretti.

Avvio completo della rete

Per avviare la rete, è necessario eseguire sequenzialmente i seguenti script nella root del progetto:

```
./generate-artifacts.sh
```

Questo script prepara tutto il materiale crittografico e di configurazione necessario:

- Genera i certificati (con cryptogen)
- Crea il blocco di genesi (genesis.block)
- Genera la transazione per la creazione del canale (mychannel.tx)

```
docker compose up -d
```

Avvia tutti i container Docker definiti nel file docker-compose.yaml.
La rete parte con i peer, gli orderer e (se previsto) il container CLI o CA.
Successivamente controllare l'esecuzione del dei container con il comando:

```
docker ps
```

Continuando:

```
./createChannel.sh
```

Questo passaggio ha necessità di essere revisionato dal momento che possono verificarsi diversi errori circa la mancanza del percorso MSP.

Infine,

```
./deploy-chaincode.sh
```

Esegue il ciclo completo di packaging, installazione, approvazione e commit del chaincode. Al termine, lo smart contract sarà pronto per essere invocato o interrogato. In questa struttura creata, il chaincode è un semplice file di esempio che, al momento dell'integrazione con il progetto finale, deve essere sostituito con tutte le componenti adatte.

Sviluppi futuri

Per migliorare e ottimizzare il progetto per garantire ancora più scalabilità, ma sopratutto resilienza a guasti, bisogna implementare meccanismi che garantiscano flessibilità e robustezza: se qualcosa dovesse andare storto durante il deploy del chaincode, il sistema dovrà essere in grado di riportarsi automaticamente in uno stato consistente attraverso il rollback.