

Sistemi Cooperativi e Reti Sociali

Daniele Margiotta, Gabriele Prestifilippo, Luca Squadrone

Progetto 2019

Indice

1 Idea Progetto	3
1.1 Premesse	3
2 Software/Linguaggi utilizzati	4
2.1 Docker	4
2.1.1 Docker-Swarm	4
2.2 Golang	5
2.2.1 Chaincode	5
2.3 Hyperledger	6
2.3.1 Permissioned and Permissionless Blockchain	6
2.3.2 Hyperledger Fabric	7
2.3.3 Transazione	7
3 Network Topology	8
4 Realizzazione Network	9
4.1 Inizializzazione Swarm	9
4.2 Creazione Network	9
4.3 Multi-Host-Network	9
5 Configurazione della Rete	11
5.1 PC1	11
5.1.1 Server CA	11
5.1.2 Orderer	12
5.1.3 CouchDB per PC1	13
5.1.4 Peer0	14
5.2 PC2	15
5.2.1 CouchDB per PC2	15
5.2.2 Peer1	15
5.2.3 CLI	16
5.2.4 script.sh	17
6 Smart Contract	18
6.1 Codice	18
6.1.1 Struct e Librerie	18
6.1.2 Init	19
6.1.3 Invoke	20
6.1.4 main	20

6.1.5	aggiungi	20
6.1.6	aggiungiMarito / aggiungiMoglie	23
6.1.7	divorzio	24
6.1.8	queryMoglie / queryMarito	27
7	Test Rete	28
7.1	CLI bin/bash su PC2	28
7.2	Istanziare il Chaincode su Peer0	28
7.3	Interrogare come Peer	29
7.3.1	aggiungi	30
7.3.2	aggiungiMarito (errore)	30
7.3.3	divorzio Marito	31
7.3.4	divorzio Moglie	32
7.3.5	aggiungiMarito	32
7.3.6	queryMoglie	33
8	Conclusione	34
8.1	Vantaggi	34
8.2	Svantaggi	34
8.3	Svantaggi / Vantaggi	34

Capitolo 1

Idea Progetto

Il matrimonio prevede degli accordi che riguardano anche i rapporti patrimoniali tra i coniugi. Dal 1975, il regime patrimoniale stabilito dalla legge, durante la celebrazione del matrimonio, è quello della comunione dei beni. L'idea di questo progetto è quella di creare uno Smart Contract da eseguire sulla Blockchain, che sia in grado di gestire la comunione dei beni di due coniugi senza l'uso di un intermediario umano come un giudice o un avvocato.

1.1 Premesse

La realizzazione dello smart contract di base su queste premesse:

- Una coppia sposata con la comunione dei beni deve utilizzare un solo conto in comune;
- Se una persona chiede il divorzio non deve poter utilizzare (aggiungere o togliere) i soldi sul conto in comune, ma in quel caso può utilizzare il conto separato privato (vale anche per il coniuge che magari non vuole ancora divorziare).
- Se tutti e due chiedono il divorzio allora i beni presenti nel conto in comune dovranno essere divisi a metà in ogni caso.

Capitolo 2

Software/Linguaggi utilizzati

Per il progetto sono stati utilizzati diversi software e linguaggi di programmazione che elencheremo di seguito.

2.1 Docker

Docker è un software opensource per il deployment e la gestione dei container ovvero dei contenitori all'interno dei quali vengono inserite ed eseguite applicazioni specifiche. Lo strumento utilizza delle "immagini" che facilitano enormemente la distribuzione delle applicazioni da eseguire in forma containerizzata e ciascun file d'immagine è strutturato su più livelli o strati in modo che ogni modifica applicata sul contenuto generi un nuovo livello. Una soluzione intelligente che permette di preservare la configurazione dei livelli sottostanti pur consentendo possibilità di personalizzazione illimitate (è ovviamente possibile tornare alla versione precedente in qualunque momento e senza difficoltà).

2.1.1 Docker-Swarm

La modalità Swarm è una funzione di Docker che fornisce funzionalità di orchestrazione dei contenitori predefinite, inclusi il clustering nativo di host Docker e la pianificazione dei carichi di lavoro dei contenitori. Un gruppo di host Docker forma un cluster "swarm" quando i relativi motori Docker vengono eseguiti insieme nella "modalità swarm".

Uno swarm è costituito da due tipi di host contenitore: nodi di gestione e nodi del ruolo di lavoro. Ogni swarm viene inizializzato mediante un nodo di gestione e tutti i comandi dell'interfaccia della riga di comando di Docker per il controllo e il monitoraggio di uno swarm devono essere eseguiti dai uno dei relativi nodi di gestione. I nodi di gestione possono essere considerati come dei "guardiani" dello stato Swarm; insieme formano un gruppo di consenso che mantiene il riconoscimento dello stato dei servizi in esecuzione nello swarm e il relativo compito è quello di garantire che lo stato effettivo dello swarm corrisponda sempre a quello previsto che viene definito dallo sviluppatore o dall'amministratore.

I nodi del ruolo di lavoro vengono orchestrati dallo swarm di Docker tramite nodi di gestione. Per aggiungere uno swarm, un nodo del ruolo di lavoro deve utilizzare un "token di aggiunta" che è stato generato dal nodo di gestione quando lo swarm è stato inizializzato.

2.2 Golang

Go è un linguaggio di programmazione open source sviluppato da Google. Golang è compilabile, anche se si è concentrato sin dall'inizio nel garantire un'elevata velocità di conversione. Inoltre il linguaggio di programmazione dispone di una modalità automatica per la pulizia della memoria (in inglese Garbage Collection o abbreviato in GC), che si occupa in background di una gestione ottimale delle risorse della memoria disponibili e in questo modo impedisce l'insorgere di relativi problemi.

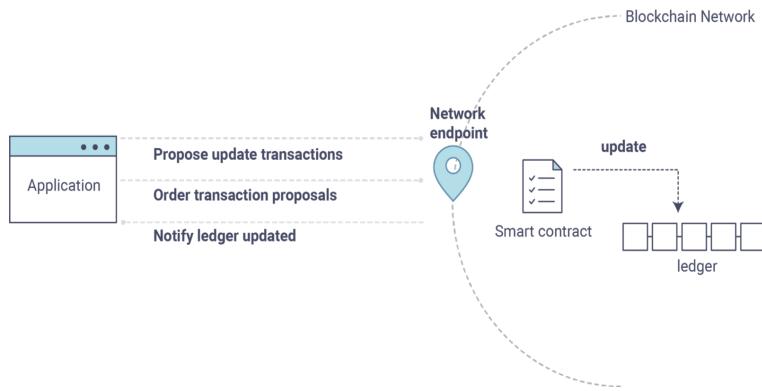
Alcune delle caratteristiche di markup decisive di Golang sono:

- Un sistema di tipi con una forte proprietà di espressione ma leggero per una classificazione e differenziazione ottimale dei diversi oggetti (variabili, funzioni, ecc.);
- La concorrenza (programmazione parallela) per un'esecuzione più veloce del programma;
- La già menzionata pulizia automatica (GC) per garantire un utilizzo ottimale della memoria disponibile e per evitare problemi correlati;
- Una specificazione rigida di dipendenze senza una sintassi dichiarativa impegnativa;
- L'uso su tutte le piattaforme, consentendo così di utilizzare le applicazioni sviluppate su tutti i sistemi comuni.

In questo progetto è stato utilizzato per la scrittura del Chaincode.

2.2.1 Chaincode

Chaincode (o smart contract) è un programma che gestisce la logica aziendale concordata dai membri della rete fabric e gestisce lo stato del ledger condiviso attraverso la transazione inviata dall'applicazione-client.



2.3 Hyperledger

Hyperledger è un progetto open source per lo sviluppo collaborativo di blockchain nato all'interno della Linux Foundation alla fine del 2015. Hyperledger Fabric 1.0 è il primo, dopo un anno di test, sperimentazioni, progetti pilota e proof of concept, ad essere stato reso disponibile, nell'aprile 2017 (e per maggio 2018 è prevista la release 2.0), per l'offerta di servizi basati sulla tecnologia blockchain. Per prima cosa bisogna specificare che il progetto nasce per abilitare la costruzione di **blockchain permissioned** (ossia “private” e che richiedono, per potervi aderire, specifici permessi definiti dall'owner della blockchain che può essere un consorzio, un'associazione, un comitato direttivo ecc.).

2.3.1 Permissioned and Permissionless Blockchain

Tutte le blockchain possono essere categorizzate rispetto a chi può contribuire al consenso del sistema. le due categorie principali sono le **permissioned blockchain** e le **Permissionless(o Public)**

In una Permissioned blockchain, come hyperledger, gli attori che possono contribuire al consenso del sistema sono un insieme ristretto di nodi appartenenti a un insieme di organizzazioni autorizzate dette **consortium**. In questo tipo di blockchain esistono due particolari tipi di nodi:

- I nodi di tipo **orderer** con il compito di ordinare e dividere in blocchi univoci e finali le transazioni in accordo con gli altri nodi di tipo orderer, mantenere i canali dei diversi **orderer system channel(s)** ognuno di cui ospita la lista delle organizzazioni appartenenti al consortium, gestire i permessi di creazione di nuovi canali nella rete, fornire un controllo di accesso ai canali di base.
- I nodi di tipo **administrator** con il compito di modificare gestire la lista del consortium, i permessi e quali canali siano un orderer system channel e gestire il set di politiche di accesso ai canali attualmente in uso.

Si noti che in una permissioned blockchain, a seconda delle politiche in atto, la creazione di smart contract può essere ristretta ad un sottinsieme dei nodi tra quelli approvati(o che possono validare blocchi). Inoltre in una Permissioned blockchain, poichè i blocchi sono organizzati dai nodi ordered, le transazioni sono deterministicamente accettate o rifiutate per tutti i nodi, ed inoltre è molto ridotto il fenomeno di ledger fork.

Tutte queste proprietà hanno lo svantaggio di richiedere un'autenticazione dei nodi che interagiscono con la rete della blockchain, poichè ogni nodo(inclusi i nodi orderer) per autenticarsi deve fornire un **Membership Service Provider** (MSP), ovvero una autenticazione di appartenenza a un'organizzazione. Richiedere un'autenticazione automaticamente richiede la presenza di una C.A., indipendentemente da come essa sia implementata.

In una Permissionless blockchain, come Bitcoin o Ethereum, chiunque può partecipare alla rete per partecipare al consenso e creare smart contract. Ciò implica che l'accettazione di un blocco non è più deterministica, ed inoltre non vi è alcun vincolo sul fenomeno di ledger fork .

2.3.2 Hyperledger Fabric

Hyperledger Fabric offre un'architettura modulare che rende disponibili componenti (meccanismo di consenso, servizi per l'adesione e la gestione dei membri della blockchain ecc.) che, con la logica del plug and play, possono essere attivati nell'ambito di una blockchain.

2.3.3 Transazione

Prima di dire con quale algoritmo di consenso operano i nodi e come è stata realizzata la struttura, vediamo come una transazione viene realmente effettuata in hyperledger.

Ogni transazione passa tre fasi:

1. Proposal:

In questa prima fase un nodo autorizzato manda una proposta di transazione ad un insieme di peer a cui è collegato, e riceve da essi una risposta. In questa fase nessuno dei nodi interessati apporta modifiche al proprio ledger

2. Ordering and Packaging:

In questa fase il nodo autorizzato manda la transazione, i dati ad essa associati e le risposte ricevute dai suoi peer ad un nodo di tipo orderer, che provvederà a raggruppare le transazioni in blocchi. L'ordine in cui sono ordinate le transazioni è deterministico in base ai dati associati ad ogni transazione

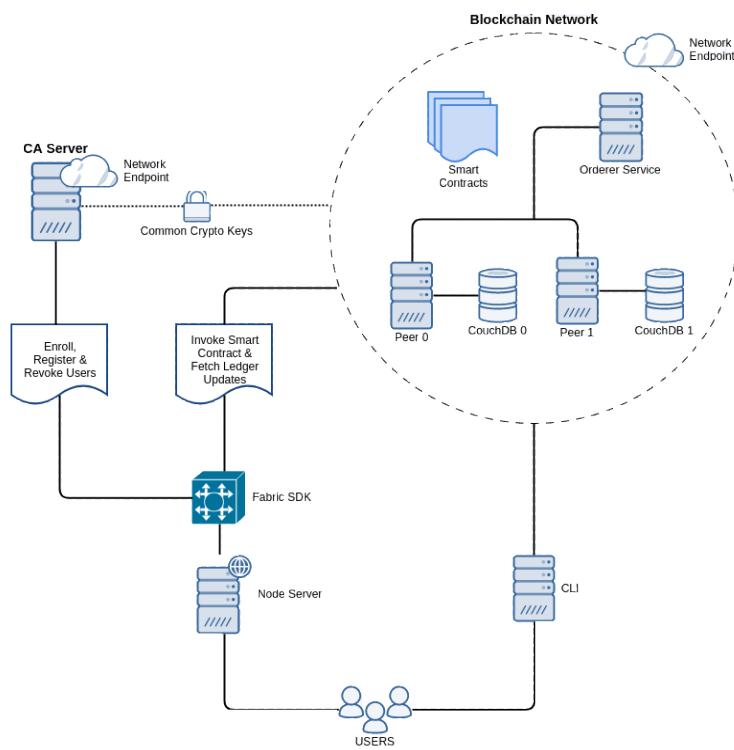
3. Validation and Commit Nell'ultima fase i blocchi sono distribuiti ad ogni peer della rete. Le singole transazioni del blocco appena ricevuto quindi sono dunque deterministicamente validate in base alle transazioni validate fino a quel momento. Le transazioni non valide sono segnate come tali, ma l'ordine delle transazioni nel blocco non viene modificato. Il blocco viene dunque eseguito e le transazioni valide modificano quindi le copie locali dei ledger.

Capitolo 3

Network Topology

La rete creata per il progetto è costituita da 2 host (PC1 e PC2) che forniscono i servizi come di seguito descritti:

- Certificate Authority (CA) su PC1
- Orderer su PC1
- Peer0 su PC1
- Peer1 su PC2
- CLI su PC2



Capitolo 4

Realizzazione Network

4.1 Inizializzazione Swarm

Dopo aver installato docker nelle rispettive macchine, possiamo inizializzare una Swarm sul PC1 tramite il comando:

```
1 $ docker swarm init
```

Adesso generiamo il token necessario per aggiungere il PC2 e quindi sul PC1 usiamo il comando:

```
1 $ docker swarm join --token manager
```

Che restituirà un comando (come nell'esempio dopo) da usare su PC2 per aggiungersi alla swarm appena creata:

```
1 $ docker swarm join --token SWMTKN-1-3  
as8cvf3yxk8e7zj98954jhjza3w75mngmxh543llgpo0c8k7z -61  
zyibtaqjffeess <IP-PC1:PORTA>
```

4.2 Creazione Network

Sul PC1 creeremo una network dentro al docker che andremo ad usare per la condivisione del progetto tra i due peer. Tramite il seguente comando relaziamo la rete "my-net":

```
1 $ docker network create --attachable --driver overlay my-net
```

4.3 Multi-Host-Network

Siccome il nostro progetto consiste nel creare uno smart contract e far comunicare due PC su di esso, useremo una Network pre-esistente che è possibile reperire su Github, infatti con il seguente comando la cloniamo su PC1 e PC2.

```
1 $ git clone https://github.com/wahabjawed/Build-Multi-Host-Network-Hyperledger.git
```

Una volta scaricata la cartella notiamo lo script **bmhn.sh** che è analogo a **./byfn.sh** della "firsts-network" di fabric. Questo script genera le Network Artifacts (Crypto Material). È necessario copiare queste cartelle su PC2 in modo

che entrambi i progetti abbiano lo stesso materiale crittografico.

N.B. È importante che entrambi i PC debbano avere lo stesso materiale crittografico altrimenti la rete non comunicherà.

Capitolo 5

Configurazione della Rete

5.1 PC1

Di seguito descriviamo i comandi da eseguire sul PC1 per configurare la rete, da notare che ogni comando deve essere eseguito su un terminale diverso in quanto i comandi **"docker run"** avvieranno processi in container isolati. Un contenitore è un processo che viene eseguito su un host. L'host può essere locale o remoto. Quando viene eseguito un comando docker run, il processo contenitore in esecuzione è isolato in quanto ha il proprio file system, la propria rete e la propria struttura di processo isolata dall'host.

N.B. Assicurarsi inoltre di essere nella cartella **"Build-Multi-Host-Network-Hyperledger /"** prima di eseguire qualsiasi script. Gli script utilizzano i file nella cartella "Build-Multi-Host-Network-Hyperledger" e generano un errore se non riescono a trovarla.

5.1.1 Server CA

Per avviare il server CA sul PC1, abbiamo bisogno di eseguire questo comando, ma prima di farlo dobbiamo sostituire **"inserire il nome della chiave segreta"** con il nome della **private key** che si trova nella cartella **"/crypto-config/peerOrganizations/org1.example.com/ca"**.

```
1 docker run --rm -it --network="my-net" --name ca.example.com -p  
    7054:7054 -e FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server -  
    e FABRIC_CA_SERVER_CA_NAME=ca.example.com -e  
    FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-ca-server-  
    config/ca.org1.example.com-cert.pem -e  
    FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-  
    config/{inserire il nome della chiave segreta} -v $(pwd)/crypto-  
    config/peerOrganizations/org1.example.com/ca/:/etc/hyperledger/  
    /fabric-ca-server-config -e  
    CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=hyp-net hyperledger/  
    fabric-ca sh -c 'fabric-ca-server start -b admin:adminpw -d'
```

```

root@daniele-Parallels-Virtual-Platform: /home/daniele/fabric-samples/Build-Multi-Host
mples.com'
2019/06/25 12:17:44 [DEBUG] Initialize Idemix issuer revocation key material
2019/06/25 12:17:44 [INFO] Idemix issuer revocation public and secret keys were
generated for CA 'ca.example.com'
2019/06/25 12:17:44 [INFO] The revocation key was successfully stored. The publ
ic key is at: /etc/hyperledger/fabric-ca-server/IssuerRevocationPublicKey, priv
ate key is at: /etc/hyperledger/fabric-ca-server/nsp/keystore/IssuerRevocationP
rivateKey
2019/06/25 12:17:44 [DEBUG] Intializing nonce manager for issuer 'ca.example.co
m'
2019/06/25 12:17:44 [INFO] Home directory for default CA: /etc/hyperledger/fab
ric-ca-server
2019/06/25 12:17:44 [DEBUG] 1 CA instance(s) running on server
2019/06/25 12:17:44 [INFO] Operation Server Listening on 127.0.0.1:9443
2019/06/25 12:17:44 [INFO] Listening on http://0.0.0.0:7054

```

5.1.2 Orderer

Cosa sono?

Gli "orderers" (o "ordering service nodes"): funzionano da canale di comunicazione tra i clients ed i peers, offrendo un servizio di broadcast per i messaggi contenenti le transazioni che avvengono in un network, è possibile specificare se tutti i client ricevano le stesse informazioni oppure no.

Consenso

Andiamo quindi a dare una breve definizione di quali tipi di consenso possiamo chiedere ai nodi di tipo orderer:

- Solo: Esiste un solo nodo di tipo orderer che decide l'ordinamento dei blocchi. questo tipo di consenso è adatto solamente a progetti di testing o piccole reti locali, in quanto rende la blockchain network non fault tollerant.
- Raft: Tra tutti i nodi di tipo orderer viene scelto dinamicamente un nodo di tipo Leader, mentre gli altri nodi diventano di tipo Follower. In questo tipo di consenso il leader ha il compito di creare il blocco a partire dalle transazioni e lo replica a tutti i follower. questo tipo di consenso è crash tollerant, poichè questo tipo di consenso è definito in modo che sia sempre possibile selezionare un nodo leader finché è possibile stabilire un quorum tra i nodi, ovvero finchè almeno la metà dei nodi +1 è attiva.
- Kafka: Kafka concettualmente utilizza la stessa idea di nodi follower e leader, ma include funzionalità come coordinamento dei nodi, cluster membership, access control, controller election ed altre. Impostare un consenso di tipo Kafka è notoriamente difficile a causa della sua complessità e numero di opzioni disponibili. Ciononostante Kafka fornisce alla rete un livello di fault tolerance molto più alto di Raft se impostato correttamente

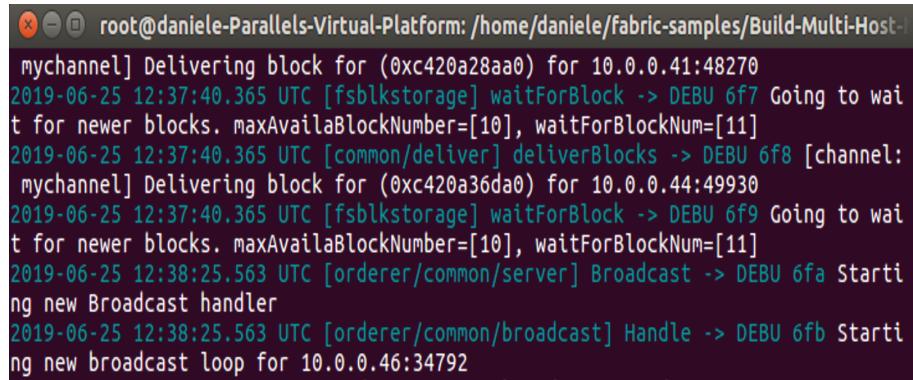
Eseguirlo su PC1

Per eseguirlo lanciamo questo comando su un altro terminale del PC1:

```

1 docker run --rm -it --network="my-net" --name orderer.example.com --
  p 7050:7050 -e ORDERER_GENERAL_LOGLEVEL=debug -e
  ORDERER_GENERAL_LISTENADDRESS=0.0.0.0 -e
  ORDERER_GENERAL_LISTENPORT=7050 -e
  ORDERER_GENERAL_GENESISMETHOD=file -e
  ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.
  genesis.block -e ORDERER_GENERAL_LOCALMSPID=OrdererMSP -e
  ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp -e
  ORDERER_GENERAL_TLS_ENABLED=false -e
  CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net -v $(pwd)/channel-
  artifacts/genesis.block:/var/hyperledger/orderer/orderer.
  genesis.block -v $(pwd)/crypto-config/ordererOrganizations/
  example.com/orderers/orderer.example.com/msp:/var/hyperledger/
  orderer/msp -w /opt/gopath/src/github.com/hyperledger/fabric
  hyperledger/fabric-orderer orderer

```



The terminal window displays the logs of an orderer node. It shows the node delivering blocks, waiting for newer blocks, and performing broadcasts. The logs are timestamped and include file names like fsblkstorage and common/deliver.

```

root@daniele-Parallels-Virtual-Platform:/home/daniele/fabric-samples/Build-Multi-Host-
mychannel] Delivering block for (0xc420a28aa0) for 10.0.0.41:48270
2019-06-25 12:37:40.365 UTC [fsblkstorage] waitForBlock -> DEBU 6f7 Going to wait for newer blocks. maxAvailableBlockNumber=[10], waitForBlockNum=[11]
2019-06-25 12:37:40.365 UTC [common/deliver] deliverBlocks -> DEBU 6f8 [channel: mychannel] Delivering block for (0xc420a36da0) for 10.0.0.44:49930
2019-06-25 12:37:40.365 UTC [fsblkstorage] waitForBlock -> DEBU 6f9 Going to wait for newer blocks. maxAvailableBlockNumber=[10], waitForBlockNum=[11]
2019-06-25 12:38:25.563 UTC [orderer/common/server] Broadcast -> DEBU 6fa Starting new Broadcast handler
2019-06-25 12:38:25.563 UTC [orderer/common/broadcast] Handle -> DEBU 6fb Starting new broadcast loop for 10.0.0.46:34792

```

5.1.3 CouchDB per PC1

Cos'è?

Con il nome CouchDB, indichiamo un elemento dello **State database**, che implementa a tutti gli effetti un ledger con proprietà aggiuntive rispetto ad un ledger classico. Ogni State database è composto da due elementi, un **LevelDB**, che è il database standard chiave-valore, che mantiene i dati validati del sistema, e un **CouchDB** che è un "optional alternative external state database", ovvero un database esterno di stati alternativi del database principale.

CouchDB è utilizzato per semplificare la generazione di proposte di transazioni (in particolar modo di transazioni complesse che potrebbero richiedere delle modifiche alle chiavi e quindi influire sulle transazioni successive) e dei commit delle transazioni.

Per esempio, se avessimo due agenti Bob e Alice, e volessimo trasferire tutti gli oggetti di proprietà di Bob ad Alice, quello che vorremmo fare è generare una transazione che sposta tutti gli oggetti di Bob ad Alice, e mandarla ad un nodo orderer per essere inserita in un blocco. Nel tempo tra la generazione della transazione ed il suo commit, Alice potrebbe ricevere un commit per l'acquisizione di un ulteriore oggetto da un altro agente. Al momento del commit della transazione tra Alice e Bob, il nuovo oggetto di Alice non era previsto in questa transazione, quindi si potrebbe generare un oggetto orfano di proprietario, o peggio, l'oggetto potrebbe scomparire.

Utilizzando CouchDB ci si può astrarre da queste problematiche di commit delle transazioni, poichè CouchDB permette di mantenere tutti gli stati temporanei delle transazioni, e quindi gestire ogni conflitto, cambio di chiave o oggetto mancante prima di eseguire la scrittura sul ledger con procedure automatiche.

Si noti che CouchDB è un componente opzionale, esterno al peer, poichè se si prevede che il numero di transazioni contemporanee sia relativamente basso, l'utilizzo di CouchDB potrebbe non essere effettivamente necessario. A livello implementativo CouchDB, proprio perchè opzionale, è un componente esterno al peer, ma che lavora a stretto contatto con il peer al quale è associato (e con il quale ha un canale privato di comunicazione).

Eseguirlo su PC1

Questo comando genererà un'istanza couchDB che verrà utilizzata da peer0 (PC1) per l'archiviazione del ledger.

```
1 docker run --rm -it --network="my-net" --name couchdb0 -p 5984:5984
-e COUCHDB_USER= -e COUCHDB_PASSWORD= -e
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net hyperledger/fabric
--couchdb
```

```
[root@daniele-Parallels-Virtual-Platform: /home/daniele/fabric-samples/Build-Multi-Host] [notice] 2019-06-25T12:37:54.767359Z nonode@nohost <0.2276.0> b8174ab869 couchdb0:5984 10.0.0.41 undefined GET /mychannel_mycc/renzo?attachments=true 200 ok 6
[notice] 2019-06-25T12:37:54.776454Z nonode@nohost <0.2276.0> 17c8f1feb1 couchdb0:5984 10.0.0.41 undefined GET /mychannel_mycc/lucia?attachments=true 200 ok 2
[notice] 2019-06-25T12:37:58.975397Z nonode@nohost <0.2276.0> 0f38bc821b couchdb0:5984 10.0.0.41 undefined GET /mychannel_lscc/mycc?attachments=true 200 ok 1
[notice] 2019-06-25T12:37:58.978537Z nonode@nohost <0.2276.0> 2e4f1e28e7 couchdb0:5984 10.0.0.41 undefined GET /mychannel_mycc/renzo?attachments=true 200 ok 1
[notice] 2019-06-25T12:37:58.983390Z nonode@nohost <0.2276.0> b2ef9c193 couchdb0:5984 10.0.0.41 undefined GET /mychannel_mycc/ContoInComune?attachments=true 200 ok 4
[notice] 2019-06-25T12:37:59.000412Z nonode@nohost <0.2276.0> c095309f24 couchdb0:5984 10.0.0.41 undefined GET /mychannel_mycc/ContoMarito?attachments=true 200 ok 14
```

5.1.4 Peer0

Infine con il seguente comando associamo i servizi prima avviati e generiamo ("startiamo") il Peer0 sul PC1:

```
1 docker run --rm -it --link orderer.example.com:orderer.example.com
--network="my-net" --name peer0.org1.example.com -p 8051:7051 -p 8053:7053 -e CORE_LEDGER_STATE_STATEDATABASE=CouchDB -e
CORE_LEDGER_STATE_COUCHDBCONFIG.COUCHDBADDRESS=couchdb0:5984 -e
CORE_LEDGER_STATE_COUCHDBCONFIG.USERNAME= -e
CORE_LEDGER_STATE_COUCHDBCONFIG.PASSWORD= -e
CORE_PEER_ADDRESSAUTODETECT=true -e CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock -e CORE_LOGGING_LEVEL=DEBUG -e
CORE_PEER_NETWORKID=peer0.org1.example.com -e CORE_NEXT=true -e
CORE_PEER_ENDORSER_ENABLED=true -e CORE_PEER_ID=peer0.org1.example.com -e
CORE_PEER_PROFILE_ENABLED=true -e
CORE_PEER_COMMITTER_LEDGER_ORDERER=orderer.example.com:7050 -e
CORE_PEER_GOSSIP_IGNORESECURITY=true -e
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net -e
CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.com:7051 -e
CORE_PEER_TLS_ENABLED=false -e
```

```
CORE_PEER_GOSSIP_USELEADERELECTION=false -e
CORE_PEER_GOSSIP_ORGLEADER=true -e CORE_PEER_LOCALMSPID=Org1MSP
-v /var/run/:/host/var/run/ -v $(pwd)/crypto-config/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/
/msp:/etc/hyperledger/fabric/msp -w /opt/gopath/src/github.com/
hyperledger/fabric/peer hyperledger/fabric-peer peer node start
```

```
root@daniele-Parallels-Virtual-Platform:/home/daniele/fabric-samples/Build-Multi-Host
2 Sleeping 5s
2019-06-25 14:03:32.240 UTC [gossip/discovery] periodicalSendAlive -> DEBU 1005
3 Sleeping 5s
2019-06-25 14:03:34.362 UTC [gossip/comm] Probe -> DEBU 10054 Entering, endpoint: 10.0.0.44:7051 PKIID: [221 20 3 98 99 139 116 5 131 103 166 181 80 193 248 2
21 90 229 105 54 145 93 204 179 159 5 204 81 13 193 154 86]
2019-06-25 14:03:37.242 UTC [gossip/discovery] periodicalSendAlive -> DEBU 1005
4 Sleeping 5s
2019-06-25 14:03:37.364 UTC [gossip/comm] Probe -> DEBU 10056 Returning context
deadline exceeded
2019-06-25 14:03:37.364 UTC [gossip/discovery] func1 -> DEBU 10057 {peer1.org1.
```

5.2 PC2

Tutti i seguenti comandi ora verranno eseguiti su terminali diversi del PC2 posizionandosi ogni volta nella cartella **”Build-Multi-Host-Network-Hyperledger”**.

5.2.1 CouchDB per PC2

Questo comando genererà un’istanza couchDB che verrà utilizzata da Peer1 per l’archiviazione del ledger.

```
1 docker run --rm -it --network="my-net" --name couchdb1 -p 6984:5984
-e COUCHDB_USER= -e COUCHDB_PASSWORD= -e
COREVMDOCKER.HOSTCONFIG.NETWORKMODE=my-net hyperledger/fabric
-couchdb
```

```
root@daniele-Parallels-Virtual-Platform:/home/daniele/fabric-samples/Build-Multi-Host
[notice] 2019-06-25T12:38:18.772936Z nonode@nohost <0.445.0> cbada40bab couchdb1:5984 10.0.0.44 undefined GET /mychannel_mycc/lucia?attachments=true 200 ok
5
[notice] 2019-06-25T12:38:22.509698Z nonode@nohost <0.445.0> 0b09dece53 couchdb1:5984 10.0.0.44 undefined GET /mychannel_lscc/mycc?attachments=true 200 ok 1
[notice] 2019-06-25T12:38:22.526003Z nonode@nohost <0.445.0> fd3e89a2c1 couchdb1:5984 10.0.0.44 undefined GET /mychannel_mycc/renzo?attachments=true 200 ok
2
[notice] 2019-06-25T12:38:22.530089Z nonode@nohost <0.445.0> 4ecc8119e9 couchdb1:5984 10.0.0.44 undefined GET /mychannel_mycc/ContoInComune?attachments=true 200 ok 3
[notice] 2019-06-25T12:38:22.536238Z nonode@nohost <0.445.0> 62cb334630 couchdb1:5984 10.0.0.44 undefined GET /mychannel_mycc/ContoMarito?attachments=true 200 ok 3
```

5.2.2 Peer1

Con il seguente comando generiamo (“startiamo”) il Peer1 sul PC2:

```

1 docker run --rm -it --network="my-net" --link orderer.example.com:
  orderer.example.com --link peer0.org1.example.com:peer0.org1.
  example.com --name peer1.org1.example.com -p 9051:7051 -p
  9053:7053 -e CORE_LEDGER_STATE_STATEDATABASE=CouchDB -e
  CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb1:5984 -e
  CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME= -e
  CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD= -e
  CORE_PEER_ADDRESSAUTODETECT=true -e CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock -e
  CORE_LOGGING_LEVEL=DEBUG -e
  CORE_PEER_NETWORKID=peer1.org1.example.com -e CORE_NEXT=true -e
  CORE_PEER_ENDORSER_ENABLED=true -e CORE_PEER_ID=peer1.org1.example.com -e
  CORE_PEER_PROFILE_ENABLED=true -e
  CORE_PEER_COMMITTER_LEDGER_ORDERER=orderer.example.com:7050 -e
  CORE_PEER_GOSSIP_ORGLEADER=true -e
  CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1.org1.example.com:7051 -e
  CORE_PEER_GOSSIP_IGNORESECURITY=true -e CORE_PEER_LOCALMSPID=Org1MSP -e
  CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net -e
  CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org1.example.com:7051 -e
  CORE_PEER_GOSSIP_USELEADERELECTION=false -e
  CORE_PEER_TLS_ENABLED=false -v /var/run/:/host/var/run/ -v $(pwd)/crypto-config/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/msp:/etc/hyperledger/fabric/msp -w /opt/gopath/src/github.com/hyperledger/fabric/peer hyperledger/fabric-peer peer node start

```

```

root@daniele-Parallels-Virtual-Platform:/home/daniele/fabric-samples/Build-Multi-Host
2019-06-25 14:01:06.426 UTC [gossip/comm] Probe -> DEBU e7d3 Returning context deadline exceeded
2019-06-25 14:01:06.426 UTC [gossip/discovery] func1 -> DEBU e7d4 {peer0.org1.example.com:7051 [] [102 98 19 152 186 134 105 75 143 125 211 12 78 34 172 90 62 8 4 125 251 29 192 224 85 249 237 216 140 106 52 138] 10.0.0.41:7051 <nil> <nil>} is still dead
2019-06-25 14:01:06.427 UTC [gossip/discovery] periodicalReconnectToDead -> DEBU e7d5 Sleeping 25s
2019-06-25 14:01:09.284 UTC [gossip/discovery] periodicalSendAlive -> DEBU e7d6 Sleeping 5s
2019-06-25 14:01:14.286 UTC [gossip/discovery] periodicalSendAlive -> DEBU e7d7 Sleeping 5s
2019-06-25 14:01:19.288 UTC [gossip/discovery] periodicalSendAlive -> DEBU e7d8 Sleeping 5s

```

5.2.3 CLI

Cos'è?

Con CLI intendiamo una **command line interface** che ci permetterà di interagire con la nostra rete blockchain, istanziare il chaincode sui peer, creare canali e inviare richieste alla blockchain.

Eseguirlo su PC2

Il seguente comando avvia sul PC2 la CLI:

```

1 docker run --rm -it --network="my-net" --name cli --link orderer.example.com:orderer.example.com --link peer0.org1.example.com:peer0.org1.example.com --link peer1.org1.example.com:peer1.org1.example.com -p 12051:7051 -p 12053:7053 -e GOPATH=/opt/gopath -e CORE_PEER_LOCALMSPID=Org1MSP -e CORE_PEER_TLS_ENABLED=false -e CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock -e CORE_LOGGING_LEVEL=DEBUG -e CORE_PEER_ID=cli -e CORE_PEER_ADDRESS=peer0.org1.example.com:7051 -e

```

```
CORE_PEER_NETWORKID=cli -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp -e CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net -v /var/run/:/host/var/run/ -v $(pwd)/chaincode:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode/go -v $(pwd)/crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ -v $(pwd)/scripts:/opt/gopath/src/github.com/hyperledger/fabric/peer/scripts/ -v $(pwd)/channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts -w /opt/gopath/src/github.com/hyperledger/fabric/peer hyperledger/fabric-tools /bin/bash -c '/scripts/script.sh'
```

Se alla fine del comando si vede la scritta **All GOOD** allora la rete è stata installata correttamente.

Il comando precedente alla fine avvia lo script `./scripts/script.sh`

5.2.4 script.sh

Questo script genera i seguenti punti:

1. Crea un canale (mychannel nel nostro caso);
 2. Fai in modo che peer0 e peer1 si uniscano al canale;
 3. Una volta entrato con successo nel canale, lo script aggiornerà l'anchor peer (peer di ancoraggio, peer0 nel nostro caso);
 4. Installa il chaincode su entrambi i peer.

Ora che la nostra rete è funzionante, vediamo come è fatto il **chaincode** creato appositamente per il progetto e invochiamo/interroghiamo il chaincode su entrambi i peer da PC2.

Capitolo 6

Smart Contract

Gli smart contract di Hyperledger Fabric sono scritti in **chaincode** e chiamati da un'applicazione esterna alla blockchain quando quella applicazione ha bisogno di interagire col ledger. Nella maggior parte dei casi, chaincode interagisce solo con la componente database del ledger, il world state (eseguendo una query, per esempio) e non con il transaction log.

6.1 Codice

Analizziamo il codice dello smart contract realizzato per vedere come è strutturato le operazioni che rende possibili eseguire.

6.1.1 Struct e Librerie

La struct **SCRSChaincode** contiene tutto il necessario per poter descrivere una coppia sposata e verrà utilizzata nelle successive funzioni.

```
1 package main
2
3 import (
4     "fmt"
5     "strconv"
6     "strings"
7
8     "github.com/hyperledger/fabric/core/chaincode/shim"
9     pb "github.com/hyperledger/fabric/protos/peer"
10 )
11
12 // SCRSChaincode implementazione
13 type SCRSChaincode struct {
14     Marito          string
15     Moglie          string
16     MaritoStatoCivile string
17     MoglieStatoCivile string
18     ContoInComune   string
19 }
```

La librera **Shim** funziona come intermediario e intercetta in modo trasparente un'API, modifica i parametri passati, gestisce l'operazione stessa o reindirizza l'operazione altrove.

6.1.2 Init

La funzione **Init** viene chiamata ogni volta che viene inizializzato un chaincode. Quindi in questo punto, utilizzando i parametri passati attraverso l'array *Args* (5 parametri), possiamo inizializzare le variabili contenute nella struct (*t *SCRSCChaincode*) sopra descritta.

```
24 func (t *SCRSCChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
25     fmt.Println("Progetto Matrimonio Init")
26     _, args := stub.GetFunctionAndParameters()
27     var err error
28
29     if len(args) != 5 {
30         return shim.Error("Incorrect number of arguments. Expecting 5")
31     }
32
33     // Initialize the chaincode
34     t.Marito = args[0]
35     if strings.Compare(args[1], "sposato") != 0 {
36         return shim.Error("Devi inserire la stringa 'sposato' per la variabile Marito.")
37     }
38     t.MaritoStatoCivile = args[1]
39
40     t.Moglie = args[2]
41     if strings.Compare(args[3], "sposata") != 0 {
42         return shim.Error("Devi inserire la stringa 'sposata' per la variabile Donna.")
43     }
44     t.MoglieStatoCivile = args[3]
45
46     t.ContoInComune = args[4]
47
48     fmt.Printf("MaritoStatoCivile = %s, MoglieStatoCivile = %s, ContoInComune = %s, ContoMarito = %s, ContoMoglie = %s\n", t.MaritoStatoCivile, t.MoglieStatoCivile, t.ContoInComune, t.ContoMarito, t.ContoMoglie)
49
50     // Write the state to the ledger
51     err = stub.PutState(t.Marito, []byte(t.MaritoStatoCivile))
52     if err != nil {
53         return shim.Error(err.Error())
54     }
55
56     err = stub.PutState(t.Moglie, []byte(t.MoglieStatoCivile))
57     if err != nil {
58         return shim.Error(err.Error())
59     }
60
61     err = stub.PutState("ContoInComune", []byte(t.ContoInComune))
62     if err != nil {
63         return shim.Error(err.Error())
64     }
65
66     err = stub.PutState("ContoMarito", []byte("0"))
67     if err != nil {
68         return shim.Error(err.Error())
69     }
70
71     err = stub.PutState("ContoMoglie", []byte("0"))
72     if err != nil {
73         return shim.Error(err.Error())
74     }
75
76     return shim.Success(nil)
77 }
```

6.1.3 Invoke

Invoke viene chiamato ogni volta che una transazione viene inviata al chaincode, smistando la richiesta a seconda del valore passatogli come "funzione", richiamando quella giusta presente nello smart contract oppure ritornando un errore con la descrizione di quelle che possono essere richiamare.

```
82 func (t *SCRSChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
83     fmt.Println("Progetto Matrimonio Invoke")
84     function, args := stub.GetFunctionAndParameters()
85     if function == "aggiungi" {
86         // richama la funzione aggiungi
87         return t.aggiungi(stub, args)
88     } else if function == "divorzia" {
89         // richama la funzione divorzia
90         return t.divorzia(stub, args)
91     } else if function == "queryMarito" {
92         // richama la funzione queryMarito
93         return t.queryMarito(stub, args)
94     } else if function == "queryMoglie" {
95         // richama la funzione queryMoglie
96         return t.queryMoglie(stub, args)
97     } else if function == "aggiungiMarito" {
98         // richama la funzione aggiungiMarito
99         return t.aggiungiMarito(stub, args)
100    } else if function == "aggiungiMoglie" {
101        // richama la funzione aggiungiMoglie
102        return t.aggiungiMoglie(stub, args)
103    }
104
105    return shim.Error("Invalid invoke function name. Expecting \\\"aggiungi\\\" \\\"aggiungiMa
106 }
```

6.1.4 main

Abbiamo bisogno di avviare il processo del chaincode in modo che possa ascoltare le richieste in entrata e questo è possibile farlo mettendo main tramite il metodo "Start" della libreria shim.

```
528 func main() {
529     err := shim.Start(new(SCRSChaincode))
530     if err != nil {
531         fmt.Printf("Error starting SCRS chaincode: %s", err)
532     }
533 }
```

6.1.5 aggiungi

La funzione aggiungi prende come parametro 3 valori di tipo string, che stanno a rappresentare:

1. Nome del marito;
2. Nome della moglie;
3. Valore da aggiungere (sottrarre se < 0) al Conto in Comune.

Possiamo notare l'uso della funzione **GetState** e **PutState** della libreria shim, che servono rispettivamente per recuperare dalla blockchain il valore associato al parametro passato e aggiornare il valore associato al parametro passato.

```
231 func (t *SCRSChaincode) aggiungi(stub shim.ChaincodeStubInterface, args []string) pb.Response {
232     var X string // Transaction value
233     var Conto int
234     var err error
235
236     if len(args) != 3 {
237         return shim.Error("Incorrect number of arguments. Expecting 3")
238     }
239
240     t.Marito = args[0]
241     t.Moglie = args[1]
242
243     // Get the state from the ledger
244     Maritovalbytes, err := stub.GetState(t.Marito)
245     if err != nil {
246         return shim.Error("Failed to get state")
247     }
248     if Maritovalbytes == nil {
249         return shim.Error("Entity not found")
250     }
251     t.MaritoStatoCivile = string(Maritovalbytes)
252
253     Moglievalbytes, err := stub.GetState(t.Moglie)
254     if err != nil {
255         return shim.Error("Failed to get state")
256     }
257     if Moglievalbytes == nil {
258         return shim.Error("Entity not found")
259     }
260
261     t.MoglieStatoCivile = string(Moglievalbytes)
262
263     if strings.Compare(t.MaritoStatoCivile, "sposato") != 0 || strings.Compare(t.MoglieStatoCivile, "sposata") != 0 {
264         return shim.Error("Invalid transaction, tutte e due devono essere sposati!")
265     }
266
267     X = args[2]
268
269     Contovalbytes, err := stub.GetState("ContoInComune")
270     if err != nil {
271         return shim.Error("Failed to get state")
272     }
273     if Contovalbytes == nil {
274         return shim.Error("Entity not found")
275     }
276
277     Somma, err := strconv.Atoi(X)
278     Conto, err = strconv.Atoi(string(Contovalbytes))
279
280     if Somma < 0 {
281         if Conto < Somma {
282             return shim.Error("Impossibile completare operazione, Conto inferiore alla somma da voler sottrarre.")
283         }
284
285     Conto = Conto + Somma
286
287     t.ContoInComune = strconv.Itoa(Conto)
288 }
```

```

289     fmt.Printf("MaritoStatoCivile = %s, MoglieStatoCivile = %s, ContoInComune = %s\n", t.MaritoStatoCivile, t.MoglieStatoCivile, t.ContoInComune)
290
291     // Write the state back to the ledger
292     err = stub.PutState(t.Marito, []byte(t.MaritoStatoCivile))
293     if err != nil {
294         return shim.Error(err.Error())
295     }
296
297     err = stub.PutState(t.Moglie, []byte(t.MoglieStatoCivile))
298     if err != nil {
299         return shim.Error(err.Error())
300     }
301
302     err = stub.PutState("ContoInComune", []byte(t.ContoInComune))
303     if err != nil {
304         return shim.Error(err.Error())
305     }
306
307     Risposta := "MaritoStatoCivile = " + t.MaritoStatoCivile + ", MoglieStatoCivile = " + t.MoglieStatoCivile + ", ContoInComune = " + t.ContoInComune
308
309     return shim.Success([]byte(Risposta))
310 }
```

”aggiungi” recupera lo *StatoCivile* associato al *Marito* passatogli come parametro (analogamente per la *Moglie*) e controlla se tutti e due sono ancora sposati (non hanno chiesto il divorzio), se si allora recupera il valore del *ContoInComune* dalla blockchain e ne aggiunge (sottrae se valore < 0) il valore ”X” passatogli come terzo parametro. Infine esegue un PutState di tutti i dati prelevati in precedenza.

6.1.6 aggiungiMarito / aggiungiMoglie

N.B. la seguente funzione ha un suo corrispondente femminile, per evitare rigidità ne descriviamo solo uno.

La funzione aggiungiMarito (aggiungiMoglie) richiede solo due parametri

1. nome del Marito (Moglie);
2. Valore da aggiungere (sottrarre se < 0) al ContoMarito (ContoMoglie).

```
111 func (t *SCRSChaincode) aggiungiMarito(stub shim.ChaincodeStubInterface, args []string) pb.Response {  
112     var ContoMarito string  
113     var X string  
114  
115     if len(args) != 2 {  
116         return shim.Error("Incorrect number of arguments. Expecting 2")  
117     }  
118  
119     t.Marito = args[0]  
120  
121     Maritovalbytes, err := stub.GetState(t.Marito)  
122     if err != nil {  
123         return shim.Error("Failed to get state")  
124     }  
125     if Maritovalbytes == nil {  
126         return shim.Error("Entity not found")  
127     }  
128     t.MaritoStatoCivile = string(Maritovalbytes)  
129  
130     if t.MaritoStatoCivile != "divorzio" {  
131         return shim.Error("Devi aver richiesto il divorzio per poter usufruire del conto separato")  
132     }  
133  
134     X = args[1]  
135  
136     ContoMaritovalbytes, err := stub.GetState("ContoMarito")  
137     if err != nil {  
138         return shim.Error("Failed to get state")  
139     }  
  
140     if ContoMaritovalbytes == nil {  
141         return shim.Error("Entity not found")  
142     }  
143  
144     Somma, err := strconv.Atoi(X)  
145     Conto, err := strconv.Atoi(string(ContoMaritovalbytes))  
146  
147     if Somma < 0 {  
148         if Conto < Somma {  
149             return shim.Error("Impossibile completare operazione, Conto inferiore alla somma da voler  
150         }  
151     }  
152  
153     Conto = Conto + Somma  
154  
155     ContoMarito = strconv.Itoa(Conto)  
156  
157     err = stub.PutState("ContoMarito", []byte(ContoMarito))  
158     if err != nil {  
159         return shim.Error(err.Error())  
160     }  
161  
162     Risposta := "ContoMarito = " + ContoMarito  
163     fmt.Printf("ContoMarito = %s\n", ContoMarito)  
164  
165     return shim.Success([]byte(Risposta))  
166 }
```

”aggiungiMarito” (aggiungiMoglie) controlla se il Marito (Moglie) ha come *StateCivile* quello di ”divorzio” e in caso affermativo prima di aggiungere la quantità passatagli come parametro nel ContoMarito (ContoMoglie) fa un ulteriore controllo che riguarda il valore < 0, perchè se l’utente ha intenzione di prelevare allora la quantità richiesta non deve superare quella disponibile.

6.1.7 divorzio

La funzione ”divorzio” richiede 2 parametri:

1. nome del Coniuge A;
2. nome del Coniuge B.

L’intendo di questa funzione è quella di dare la possibilità all’utente di chiedere il divorzio (A chiede divorzio da B), quindi poter bloccare il ”ContoInComune” (in attesa di richiesta del divozio anche da parte di B verso A) e passare all’uso del ”ContoMarito/ContoMoglie”.

```

314 func (t *SCRSChaincode) divorzia(stub shim.ChaincodeStubInterface, args []string) pb.Response {
315     if len(args) != 2 {
316         return shim.Error("Incorrect number of arguments. Expecting 2")
317     }
318
319     ConiugeA := args[0]
320     ConiugeB := args[1]
321
322     err := stub.PutState(ConiugeA, []byte("divorzio"))
323     if err != nil {
324         return shim.Error(err.Error())
325     }
326
327     //
328     // controllo se tutte e due adesso sono 'divorzio'          *1*
329     // e in caso prendo il ContoInComune, lo divido in 2          *2*
330     // lo metto metà in ContoMarito e metà in ContoMoglie      *2*
331     // azzero ContoInComune                                     *3*
332     // faccio putState di tutto quanto                         *4*
333     //
334
335     ConiugeAvalbytes, err := stub.GetState(ConiugeA)
336     if err != nil {
337         return shim.Error("Failed to get state")
338     }
339     if ConiugeAvalbytes == nil {
340         return shim.Error("Entity not found")
341     }

```

```

342 ConiugeAStatoCivile := string(ConiugeAValbytes)
343 //
344
345 ConiugeBValbytes, err := stub.GetState(ConiugeB)
346 if err != nil {
347     return shim.Error("Failed to get state")
348 }
349 if ConiugeBValbytes == nil {
350     return shim.Error("Entity not found")
351 }
352 ConiugeBStatoCivile := string(ConiugeBValbytes)
353 //
354 // *1*
355 if ConiugeAStatoCivile == ConiugeBStatoCivile {
356
357     Contovalbytes, err := stub.GetState("ContoInComune")
358     if err != nil {
359         return shim.Error("Failed to get state")
360     }
361     if Contovalbytes == nil {
362         return shim.Error("Entity not found")
363     }
364     Conto, err := strconv.Atoi(string(Contovalbytes))
365
366     //
367     ContoMaritovalbytes, err := stub.GetState("ContoMarito")
368     if err != nil {
369         return shim.Error("Failed to get state")
370     }
371
372     if ContoMaritovalbytes == nil {
373         return shim.Error("Entity not found")
374     }
375     ContoMa, err := strconv.Atoi(string(ContoMaritovalbytes))
376
377     //
378     ContoMoglievalbytes, err := stub.GetState("ContoMoglie")
379     if err != nil {
380         return shim.Error("Failed to get state")
381     }
382     if ContoMoglievalbytes == nil {
383         return shim.Error("Entity not found")
384     }
385     ContoMo, err := strconv.Atoi(string(ContoMoglievalbytes))
386
387     // *2*
388     ContoMo = ContoMo + (Conto / 2)
389     ContoMa = ContoMa + (Conto / 2)
390
391     // *3*
392     Conto = 0
393
394     ContoInComune := strconv.Itoa(Conto)
395     ContoMarito := strconv.Itoa(ContoMa)
396     ContoMoglie := strconv.Itoa(ContoMo)
397

```

```

398     // *4*
399     err = stub.PutState("ContoInComune", []byte(ContoInComune))
400     if err != nil {
401         return shim.Error(err.Error())
402     }
403     err = stub.PutState("ContoMarito", []byte(ContoMarito))
404     if err != nil {
405         return shim.Error(err.Error())
406     }
407     err = stub.PutState("ContoMoglie", []byte(ContoMoglie))
408     if err != nil {
409         return shim.Error(err.Error())
410     }
411 }
412
413 fmt.Printf("ConiugeAStatoCivile = %s, ConiugeBStatoCivile = %s\n", ConiugeAStatoCivile, ConiugeBStatoCivile)
414 Risposta := "ConiugeAStatoCivile = " + ConiugeAStatoCivile + "ConiugeBStatoCivile = " + ConiugeBStatoCivile
415
416 return shim.Success([]byte(Risposta))
417 }
```

Dopo aver modificato lo *StatoCivile* del ”ConiugeA” viene controllato se tutti e due i ”Coniugi” hanno come *StatoCivile* ”divorzio” (nel caso fosse il secondo a chiederlo) e in caso affermarivo viene prelevato dalla blockchain il valore del *ContoInComune* che viene spartito a metà, inserito nei rispettivi *ContoMarito/ContoMoglie* e successivamente settato a 0. Infine una *PutState* di tutti i valori utilizzati.

6.1.8 queryMoglie / queryMarito

N.B. la seguente funzione ha un suo corrispondente maschile, per evitare ridondanza ne descriviamo solo uno.

La funzione "queryMoglie" (queryMarito) richiede 1 solo parametro:

1. nome della Moglie (Marito);

```
477 func (t *SCRSChaincode) queryMoglie(stub shim.ChaincodeStubInterface, args []string) pb.Response {  
478     var err error  
479  
480     if len(args) != 1 {  
481         return shim.Error("Incorrect number of arguments. Expecting name of the person to query")  
482     }  
483  
484     t.Moglie = args[0]  
485  
486     // Get the state from the ledger  
487     Moglievalbytes, err := stub.GetState(t.Moglie)  
488     if err != nil {  
489         jsonResp := "{\"Error\":\"Failed to get state for " + t.Moglie + "\"}"  
490         return shim.Error(jsonResp)  
491     }  
492  
493     if Moglievalbytes == nil {  
494         jsonResp := "{\"Error\":\"Nil for " + t.Moglie + "\"}"  
495         return shim.Error(jsonResp)  
496     }  
497  
498     ContoInComunevalbytes, err := stub.GetState("ContoInComune")  
499     if err != nil {  
500         jsonResp := "{\"Error\":\"Failed to get state for ContoInComune\"}"  
501         return shim.Error(jsonResp)  
502     }  
503  
504     if ContoInComunevalbytes == nil {  
505         jsonResp := "{\"Error\":\"Nil for ContoInComune\"}"  
506         return shim.Error(jsonResp)  
507     }  
508  
509     ContoMoglievalbytes, err := stub.GetState("ContoMoglie")  
510     if err != nil {  
511         jsonResp := "{\"Error\":\"Failed to get state for ContoMoglie \\"}"  
512         return shim.Error(jsonResp)  
513     }  
514  
515     if ContoMoglievalbytes == nil {  
516         jsonResp := "{\"Error\":\"Nil for ContoMoglie\"}"  
517         return shim.Error(jsonResp)  
518     }  
519  
520     jsonResp := "{\"Name\":\"" + t.Moglie + "\",\"State\":\"" + string(Moglievalbytes) + "\",\"ContoInComune\":\"" + string(ContoInComunevalbytes) + "\",\"ContoMoglie\":\"" + string(ContoMoglievalbytes) + "\"}"  
521     fmt.Printf("QueryMoglie Response:%s\n", jsonResp)  
522  
523     Risposta := "Nome = " + t.Moglie + ", Stato = " + t.MoglieStatoCivile + ", ContoComune = " + t.  
524     return shim.Success([]byte(Risposta))  
525 }
```

La funzione serve a prelevare dalla blockchain tutte le informazioni relative alla Moglie (Marito) passatagli come parametro, formattarle in una stringa per poterle mostrare all'utente come risultato quella query.

Capitolo 7

Test Rete

7.1 CLI bin/bash su PC2

Per poter testare la rete con lo smart contract abbiamo bisogno di un container con una CLI, quindi ci posizioniamo su un terminale del PC2 e inseriamo il seguente comando:

```
1 docker run --rm -it --network="my-net" --name cli --link orderer.example.com:orderer.example.com --link peer0.org1.example.com:peer0.org1.example.com --link peer1.org1.example.com:peer1.org1.example.com -p 12051:7051 -p 12053:7053 -e GOPATH=/opt/gopath -e CORE_PEER_LOCALMSPID=Org1MSP -e CORE_PEER_TLS_ENABLED=false -e CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock -e CORE_LOGGING_LEVEL=DEBUG -e CORE_PEER_ID=cli -e CORE_PEER_ADDRESS=peer0.org1.example.com:7051 -e CORE_PEER_NETWORKID=cli -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp -e CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net -v /var/run/:/host/var/run/ -v $(pwd)/chaincode:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode/go -v $(pwd)/crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ -v $(pwd)/scripts:/opt/gopath/src/github.com/hyperledger/fabric/peer/scripts/ -v $(pwd)/channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts -w /opt/gopath/src/github.com/hyperledger/fabric/peer/hyperledger/fabric-tools /bin/bash
```

Ora che siamo dentro la CLI, possiamo inserire i comandi per istanziare, richiamare e interrogare il chaincode.

7.2 Istanziare il Chaincode su Peer0

Per istanziare il chaincode su peer0 avremo bisogno di impostare prima alcune variabili d'ambiente.

```
1 # Variabili d'ambiente per PEER0
2
3 CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
4
```

```

5 CORE_PEER_LOCALMSPID="Org1MSP"
6
7 CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/
   fabric/peer/crypto/peerOrganizations/org1.example.com/peers/
   peer0.org1.example.com/tls/ca.crt
8
9 CORE_PEER_ADDRESS=peer0.org1.example.com:7051

```

Adesso stiamo impersonando il peer0 e possiamo inizializzare lo smart contract (che è stato installato precedentemente attraverso script.sh) tramite il comando:

```

1 $ peer chaincode instantiate -o orderer.example.com:7050 -C
   mychannel -n mycc -v 1.0 -c '{"Args":["init","renzo","sposato",
   "lucia","sposata","100"]}' -P "OR ('Org1MSP.member','Org2MSP.
   member')"

```

Questo comando inizializzerà i parametri della struct SCRSChaincode dello smart contract come:

- Marito = "renzo";
- Moglie = "lucia";
- StatoCivilMarito = "sposato";
- StatoCivilMoglie = "sposata";
- ContoInComune = "100".

7.3 Interrogare come Peer

N.B. tutte le operazioni possono essere intercambiate ed eseguite dai due peer contenuti nella rete, per provarlo basta cambiare le variabili d'ambiente nel modo giusto (come descritto sopra) e aspettare il tempo necessario per le convalidazioni delle transazioni.

Per prendere le parti del peer1 bisogna impostare le variabili d'ambiente nel seguente modo:

```

1 # Environment variables for PEER1
2
3 CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/
   fabric/peer/crypto/peerOrganizations/org1.example.com/users/
   Admin@org1.example.com/msp
4
5 CORE_PEER_LOCALMSPID="Org1MSP"
6
7 CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/
   fabric/peer/crypto/peerOrganizations/org1.example.com/peers/
   peer0.org1.example.com/tls/ca.crt
8
9 CORE_PEER_ADDRESS=peer1.org1.example.com:7051

```

Una volta impostate le variabili, proviamo ad interrogare lo smart contract creato precedentemente dal peer0.

7.3.1 aggiungi

Con questa query aggiungiamo un valore di 110 al *ContoInComune*.

```
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode invoke -C mychannel -n mycc -c '{"Args":["aggiungi","renzo","lucia","110"]}'"
2019-06-25 10:57:33.228 UTC [viperutil] getKeysRecursively -> DEBU 001 Found map
[string]interface{} value for peer.BCCSP
2019-06-25 10:57:33.228 UTC [viperutil] unmarshalJSON -> DEBU 002 Unmarshal JSON
: value cannot be unmarshalled: invalid character 'S' looking for beginning of v
alue
2019-06-25 10:57:33.228 UTC [viperutil] getKeysRecursively -> DEBU 003 Found rea
l value for peer.BCCSP.Default setting to string SW
2019-06-25 10:57:33.228 UTC [viperutil] getKeysRecursively -> DEBU 004 Found map
[string]interface{} value for peer.BCCSP.SW
2019-06-25 10:57:33.229 UTC [viperutil] getKeysRecursively -> DEBU 005 Found map
[string]interface{} value for peer.BCCSP.SW.FileKeyStore
2019-06-25 10:57:33.229 UTC [viperutil] unmarshalJSON -> DEBU 006 Unmarshal JSON
: value cannot be unmarshalled: unexpected end of JSON input

root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer
pNT+xFhpNBV5XT84wCgYIKoZIzj0EAwIwcZEL\nnMAKGA1UEBhMCVVMxEzARBgNVBAgTCKNhGlmB3Juaw
ExFjAUBgNVBAcTDVNhbLBG\ncmFuY2lZy28xGTAXBgNVBAoTEG9yZzEuZXhbxBsZ5jb20xHDAaBgNV
BAMTE2Nh\lnLm9yZzEuZXhhbxBsZ5jb20wHhcNMTkwNjI1MTAwNDAwWhcNMjkwnjIyMTAwNDAw\lnWjbB
M0swQYDVQ0GEwJVUzETMBEGA1UECBMKQ2FsawZvcn5pYTEWMBoGA1UEBxMN\lnU2FuIEZyVW5jaXNjbz
EfMB0GA1UEAxMWcGVlcjEub3JnMS5leGFtcGxlLmNvbTBZ\lnMBMGByqGSM49AgEGCCqGSM49AwEHA0IA
BHftQua+rV7DqkPnykGemirYMStpZozG\ncwwphqA6ywt7dL0wx183UYTfT4eb30Fby12M1nGUTE3FsD
zQX20jirSjTTBLMA4G\na1UdDwEB/WQEAwIHgDAMbgNVHRMBAf8EAjAAACsGA1UdIwQkMCKAIBg8ktn
wybKnjYohnhutminzHcNUpcgnTVumtProf6SeMa0GCCqGSM49BAMCA0gAMEUCIQct2Fro\,nv7jUY/dn
QbJcfcaBokDnS4p7jTvKV03QUlowIgRfHyAmoRosbuusZjdIrmcaa\ntBGSABDIflshkTMrJvs=\n
-----END CERTIFICATE-----\n" signature="0E\002\000\222\303\242\265u\207\034~\2
55>r\275\363\277.\014U\303#\203\037[\241\310\007%\316\177C\307\002\,\t\377\357>\2
0200\340\023\262\022Mg1xj\224\324\006N0^\025\320\223Q\257\377r\001\354#" >
2019-06-25 10:57:33.312 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 0aa Ch
aincode invoke successful. result: status:200 payload:"MaritoStatoCivile = sposa
to, MoglieStatoCivile = sposata, ContoInComune = 210"
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

Come possiamo osservare il *ContoInComune* aumenta da 100 a 210.

Ovviamente sarebbe possibile sottrarre inserendo nella query un valore negativo
(es. -100).

7.3.2 aggiungiMarito (errore)

Con questa query proviamo ad aggiungere un valore di 100 al *ContoMarito*.

```
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C mychannel -n mycc -c '{"Args":["aggiungiMarito","renzo","100"]}'"
2019-06-25 10:55:22.263 UTC [viperutil] getKeysRecursively -> DEBU 001 Found map
[string]interface{} value for peer.BCCSP
2019-06-25 10:55:22.263 UTC [viperutil] getKeysRecursively -> DEBU 002 Found map
[string]interface{} value for peer.BCCSP.PKCS11
2019-06-25 10:55:22.263 UTC [viperutil] unmarshalJSON -> DEBU 003 Unmarshal JSON
: value is not a string: <nil>
2019-06-25 10:55:22.263 UTC [viperutil] getKeysRecursively -> DEBU 004 Found rea
l value for peer.BCCSP.PKCS11.Hash setting to <nil> <nil>
2019-06-25 10:55:22.263 UTC [viperutil] unmarshalJSON -> DEBU 005 Unmarshal JSON
: value is not a string: <nil>
2019-06-25 10:55:22.264 UTC [viperutil] getKeysRecursively -> DEBU 006 Found rea
l value for peer.BCCSP.PKCS11.Security setting to <nil> <nil>
2019-06-25 10:55:22.264 UTC [viperutil] getKeysRecursively -> DEBU 007 Found map
[string]interface{} value for peer.BCCSP.PKCS11.FileKeyStore
```

```

root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer
cer to "pick_first"
2019-06-25 10:55:22.285 UTC [grpc] Printf -> DEBU 041 pickfirstBalancer: Handles
ubConnStateChange: 0xc4204369e0, CONNECTING
2019-06-25 10:55:22.285 UTC [grpc] Printf -> DEBU 042 pickfirstBalancer: Handles
ubConnStateChange: 0xc4204369e0, READY
2019-06-25 10:55:22.286 UTC [msp] GetDefaultSigningIdentity -> DEBU 043 Obtainin
g default signing identity
2019-06-25 10:55:22.286 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 044 java cha
incode disabled
2019-06-25 10:55:22.286 UTC [msp/identity] Sign -> DEBU 045 Sign: plaintext: 0AA
B070A6708031A0C089AFBC7E80510...7269746F0A0572656E7A6F0A03313030
2019-06-25 10:55:22.286 UTC [msp/identity] Sign -> DEBU 046 Sign: digest: 074B49
C99D318CF9A15D87D0DBA1B2C866778DEBE87B957E899E158DA179B3F0
Error: endorsement failure during query, response: status:500 message:"Devi aver
richiesto il divorzio per poter usufruire del conto separato"
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer#

```

Ma essendo il Marito ancora nello *StatoCivileMarito* = "sposato" non può farlo. Per abilitare la funzione dovrebbe richiedere il divorzio.

7.3.3 divorzio Marito

Adesso interpretremo il caso in cui il Marito voglia divorziare dalla Moglie.

```

root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chain
code query -C mychannel -n mycc -c '{"Args":["divorzia","renzo","lucia"]}'
2019-06-25 11:02:07.468 UTC [viperutil] getKeysRecursively -> DEBU 001 Found map
[string]interface{} value for peer.BCCSP
2019-06-25 11:02:07.468 UTC [viperutil] unmarshalJSON -> DEBU 002 Unmarshal JSON
: value cannot be unmarshalled: invalid character 'S' looking for beginning of v
alue
2019-06-25 11:02:07.468 UTC [viperutil] getKeysRecursively -> DEBU 003 Found rea
l value for peer.BCCSP.Default setting to string SW
2019-06-25 11:02:07.468 UTC [viperutil] getKeysRecursively -> DEBU 004 Found map
[string]interface{} value for peer.BCCSP.SW
2019-06-25 11:02:07.468 UTC [viperutil] getKeysRecursively -> DEBU 005 Found map
[string]interface{} value for peer.BCCSP.SW.FileKeyStore
2019-06-25 11:02:07.468 UTC [viperutil] unmarshalJSON -> DEBU 006 Unmarshal JSON
: value cannot be unmarshalled: unexpected end of JSON input
2019-06-25 11:02:07.468 UTC [viperutil] getKeysRecursively -> DEBU 007 Found rea

root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer
2019-06-25 11:02:07.490 UTC [grpc] Printf -> DEBU 040 ClientConn switching balan
cer to "pick_first"
2019-06-25 11:02:07.490 UTC [grpc] Printf -> DEBU 041 pickfirstBalancer: Handles
ubConnStateChange: 0xc42042c9d0, CONNECTING
2019-06-25 11:02:07.491 UTC [grpc] Printf -> DEBU 042 pickfirstBalancer: Handles
ubConnStateChange: 0xc42042c9d0, READY
2019-06-25 11:02:07.491 UTC [msp] GetDefaultSigningIdentity -> DEBU 043 Obtainin
g default signing identity
2019-06-25 11:02:07.491 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 044 java cha
incode disabled
2019-06-25 11:02:07.491 UTC [msp/identity] Sign -> DEBU 045 Sign: plaintext: 0AA
B070A6708031A0C08AFFEC7E80510...69610A0572656E7A6F0A056C75636961
2019-06-25 11:02:07.491 UTC [msp/identity] Sign -> DEBU 046 Sign: digest: 4C2EE2
B4628E0F20C482A1A9ABCCFEB48AE3D6C853CE17F5DCBB1605CE29FFA2
ConiugeAStatoCivile = divorzioConiugeBStatoCivile = sposata
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer#

```

Adesso che lo *StatoCivileMarito* è stato cambiato da "sposato" a "divorzio" gli sarà possibile accedere al *ContoMarito*, ma non più al *ContoInComune* in attesa del divorzio dell'altro coniuge.

7.3.4 divorzio Moglie

Adesso interpretremo il caso in cui la Moglie voglia divorziare dal Marito dopo che lui ha richiesto in precedenza il divorzio.

```
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chain
code query -C mychannel -n mycc -c '{"Args":["divorzia","lucia","renzo"]}'
2019-06-25 11:56:32.542 UTC [viperutil] getKeysRecursively -> DEBU 001 Found map
[string]interface{} value for peer.BCCSP
2019-06-25 11:56:32.542 UTC [viperutil] getKeysRecursively -> DEBU 002 Found map
[string]interface{} value for peer.BCCSP.SW
2019-06-25 11:56:32.542 UTC [viperutil] unmarshalJSON -> DEBU 003 Unmarshal JSON
: value cannot be unmarshalled: invalid character 'S' looking for beginning of v
alue
2019-06-25 11:56:32.543 UTC [viperutil] getKeysRecursively -> DEBU 004 Found rea
l value for peer.BCCSP.SW.Hash setting to string SHA2
2019-06-25 11:56:32.543 UTC [viperutil] unmarshalJSON -> DEBU 005 Unmarshal JSON
: value is not a string: 256
2019-06-25 11:56:32.543 UTC [viperutil] getKeysRecursively -> DEBU 006 Found rea
l value for peer.BCCSP.SW.Security setting to int 256
2019-06-25 11:56:32.543 UTC [viperutil] getKeysRecursively -> DEBU 007 Found map

root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer
2019-06-25 11:56:32.572 UTC [grpc] Printf -> DEBU 040 ClientConn switching balan
cer to "pick_first"
2019-06-25 11:56:32.573 UTC [grpc] Printf -> DEBU 041 pickfirstBalancer: Handles
ubConnStateChange: 0xc420442840, CONNECTING
2019-06-25 11:56:32.573 UTC [grpc] Printf -> DEBU 042 pickfirstBalancer: Handles
ubConnStateChange: 0xc420442840, READY
2019-06-25 11:56:32.573 UTC [msp] GetDefaultSigningIdentity -> DEBU 043 Obtainin
g default signing identity
2019-06-25 11:56:32.573 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 044 java cha
incode disabled
2019-06-25 11:56:32.574 UTC [msp/identity] Sign -> DEBU 045 Sign: plaintext: 0AA
B070A6708031A0C08F097C8E80510...69610A056C756369610A0572656E7A6F
2019-06-25 11:56:32.574 UTC [msp/identity] Sign -> DEBU 046 Sign: digest: 631333
A564D5EE3E976E5AA43688C32C34551F0BF64821B870D87ED5EFA434E4
ConiugeAStatoCivile = divorzioConiugeBStatoCivile = divorzio
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

Adesso che lo *StatoCivileMarito* è stato cambiato da "sposata" a "divorzio" le sarà possibile accedere al *ContoMoglie* e siccome tutti e due i coniugi sono nello stato "divorzio" il *ContoInComune* sarà stato diviso nei due "conti separati" e azzerato.

7.3.5 aggiungiMarito

Adesso che tutti e due i coniugi hanno chiesto il divorzio attraverso l'apposita funzione, proviamo ad aggiungere un valore di 100 al *ContoMarito*.

```
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chain
code invoke -C mychannel -n mycc -c '{"Args":["aggiungiMarito","renzo","100"]}'
2019-06-25 11:58:45.775 UTC [viperutil] getKeysRecursively -> DEBU 001 Found map
[string]interface{} value for peer.BCCSP
2019-06-25 11:58:45.775 UTC [viperutil] unmarshalJSON -> DEBU 002 Unmarshal JSON
: value cannot be unmarshalled: invalid character 'S' looking for beginning of v
alue
2019-06-25 11:58:45.775 UTC [viperutil] getKeysRecursively -> DEBU 003 Found rea
l value for peer.BCCSP.Default setting to string SW
2019-06-25 11:58:45.775 UTC [viperutil] getKeysRecursively -> DEBU 004 Found map
[string]interface{} value for peer.BCCSP.SW
2019-06-25 11:58:45.776 UTC [viperutil] getKeysRecursively -> DEBU 005 Found map
[string]interface{} value for peer.BCCSP.SW.FileKeyStore
2019-06-25 11:58:45.776 UTC [viperutil] unmarshalJSON -> DEBU 006 Unmarshal JSON
: value cannot be unmarshalled: unexpected end of JSON input
2019-06-25 11:58:45.776 UTC [viperutil] getKeysRecursively -> DEBU 007 Found rea
```

```

root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer
0070rg1MSP\022\226\006----BEGIN CERTIFICATE-----\nMIICGjCCAcCgAwIBAgIRANMt2wQkp
NT+xHpNBV5xt84wCgYIKoZIzj0EAwIwczEL\nnMAkGA1UEBhMCVVMxEzARBgNVBAgTCKNhbgLmb3JuawE
xFjAUBgNVBAcTDVNhb1BG\nncmFuY2lzb28xGTAXBgNVBAoTEG9yZzEuZXhhbXBsZS5jb20xHDAaBgNVB
AMTE2Nh\nl.Lm9yZzeuZXhhbXBsZS5jb20wHhcNMkWnjI1MTAwNDAwWhcNMkWnjIyMTAwNDAw\nnWjbBm
QswCQYDVQQGEwJVUzETMBEGAIUECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMN\nU2FuIEZyYW5jaXNjbzE
fMB0GA1UEAxMWcGVlcjEub3JnMS5leGftcGxlLmNvbTBZ\nnMBMGByqGSM49AgEGCCqGSM49AwEHA0IA8
HftQua+rv7DqkPnykGemirY5tpZozG\nncwphqA6ywt7dL0wx183UYtT4eb30FbY12M1nGUTE3fsDz
QX20jirsjTTBLMA4G\nnA1UDwEB/wQEAWIHgDAMBgNVHRMBAf8EAjAAAMCsGA1UDIwQkMCKAIBgB8ktNw
ybK\njYohnhutninzHCNUpcgnTVumtProf6SeMAoGCCqGSM49BAMCA0gAMEUCIQct2Fr\n\nv7jUY/dnQ
bJCfcab0KDnS4Qp7jTvKV03QULomWigRfHyhAmo1RosbuusZjdIrmxaa\nntBGSa8DIflshkTMrJvs=\n
----END CERTIFICATE----\n" signature:"0D\\002 \$\\362\\262\\346w\\351\\245r\\266?\\377\\0
05&\\231\\177\\263}\\341\\Q\\320?\\274\\362\\r\$\\223\\223#R\\327\\002 :fo3[\\237\\333\\337z\\346
\\000\\322\\332\\024\\217\\227n\\3236\\251\\240v\\321%Vs:M\\257\\226!\\277" >
2019-06-25 11:58:45.835 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 0aa Ch
aincode invoke successful. result: status:200 payload:"ContoMarito = 100"
root@c074ec78148b:/opt/gopath/src/github.com/hyperledger/fabric/peer#

```

Questa volta come si può vedere è stato aggiunto correttamente il valore passato come parametro al *ContoMarito*.

7.3.6 queryMoglie

N.B. la seguente funzione ha un suo corrispondente maschile, per evitare ridondanza ne descriviamo solo uno. Adesso andiamo a vedere una situazione in cui la moglie, attraverso un altro peer (peer1) e prima di aver chiesto il divorzio vuole controllare il l'ammontare del *ContoInComune*.

```

root@652fcf14e523:/opt/gopath/src/github.com/hyperledger/fabric/peer
root@652fcf14e523:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chain
code query -C mychannel -n mycc -c '{"Args":["queryMoglie","lucia"]}' 
2019-06-25 12:34:46.107 UTC [viperutil] getKeysRecursively -> DEBU 001 Found map
[string]interface{} value for peer.BCCSP
2019-06-25 12:34:46.107 UTC [viperutil] unmarshalJSON -> DEBU 002 Unmarshal JSON
: value cannot be unmarshalled: invalid character 'S' looking for beginning of v
alue
2019-06-25 12:34:46.107 UTC [viperutil] getKeysRecursively -> DEBU 003 Found rea
l value for peer.BCCSP.Default setting to string SW
2019-06-25 12:34:46.107 UTC [viperutil] getKeysRecursively -> DEBU 004 Found map
[string]interface{} value for peer.BCCSP.SW
2019-06-25 12:34:46.107 UTC [viperutil] unmarshalJSON -> DEBU 005 Unmarshal JSON
: value cannot be unmarshalled: invalid character 'S' looking for beginning of v
alue
2019-06-25 12:34:46.107 UTC [viperutil] getKeysRecursively -> DEBU 006 Found rea
l value for peer.BCCSP.SW.Hash setting to string SHA2

```

```

root@652fcf14e523:/opt/gopath/src/github.com/hyperledger/fabric/peer
2019-06-25 12:34:46.131 UTC [grpc] Printf -> DEBU 040 ClientConn switching balan
cer to "pick_first"
2019-06-25 12:34:46.131 UTC [grpc] Printf -> DEBU 041 pickfirstBalancer: Handles
ubConnStateChange: 0xc42043c9b0, CONNECTING
2019-06-25 12:34:46.131 UTC [grpc] Printf -> DEBU 042 pickfirstBalancer: Handles
ubConnStateChange: 0xc42043c9b0, READY
2019-06-25 12:34:46.132 UTC [msp] GetDefaultSigningIdentity -> DEBU 043 Obtainin
g default signing identity
2019-06-25 12:34:46.132 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 044 java cha
incode disabled
2019-06-25 12:34:46.133 UTC [msp/identity] Sign -> DEBU 045 Sign: plaintext: 0AA
A070A6608031A0B08E6A9C8E80510...6572794D6F676C69650A056C75636961
2019-06-25 12:34:46.133 UTC [msp/identity] Sign -> DEBU 046 Sign: digest: 90DDE0
53AF180F7BF733FD22A36CD626EB7A31E3A52F08D6C7CDFAA1D3CA113B
Nome = lucia, Stato = sposata, ContoComune = 210, ContoSeparato = 0
root@652fcf14e523:/opt/gopath/src/github.com/hyperledger/fabric/peer#

```

Il cambio di peer (effettuato tramite il cambio delle variabili descritto in precedenza) lo si può notare dal nome del container diverso.

Capitolo 8

Conclusione

In questo capitolo si riassumono i principali vantaggi e svantaggi dell'utilizzo dello smart contract implementato in questo progetto.

8.1 Vantaggi

1. Risparmio dei costi per l'intermediario (giudice, avvocato) che gestisce la divisione dei beni durante un divorzio;
2. Separazione del conto in comune immediata e generazione di 2 nuovi conti separati automatica senza ulteriori spese amministrative;
3. Il blocco del conto in comune automatico dopo che un coniuge ha chiesto il divorzio incentiva l'altro coniuge a chiederlo a sua volta senza allungare i tempi burocratici del divorzio legale.

8.2 Svantaggi

1. Divisione dei beni non relativa alla percentuale dei beni immessi nel conto condiviso nella durata del matrimonio;
2. Il blocco automatico del conto rende immediatamente inaccessibili i beni contenuti nel conto in comune.

8.3 Svantaggi / Vantaggi

1. Sistema rigido rispetto alle problematiche umane.