

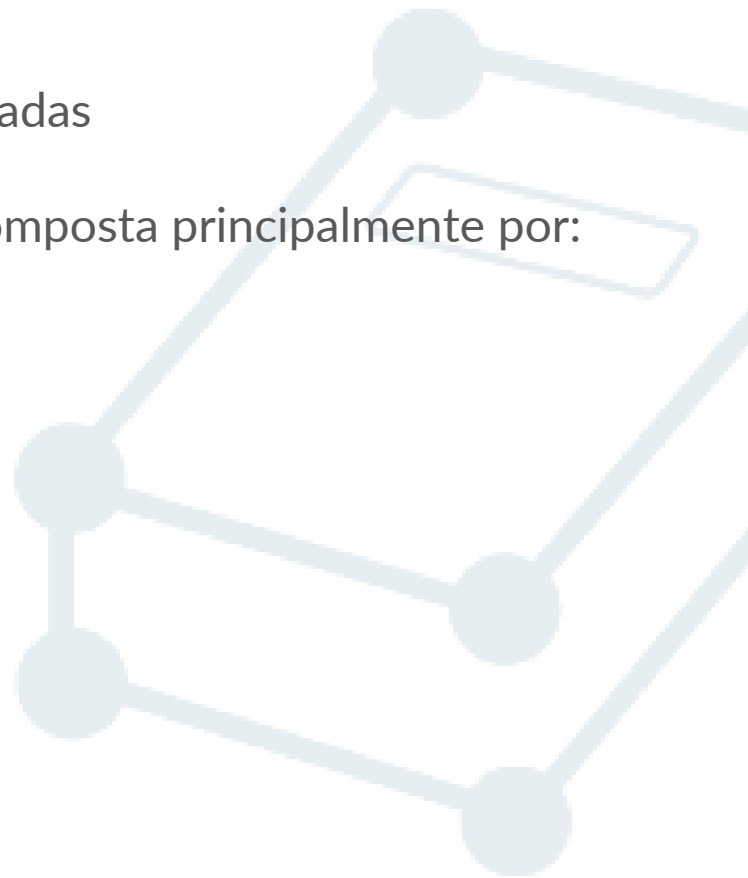
# **Desenvolvendo Dapps em plataforma Hyperledger Fabric**

# O que é uma Dapp

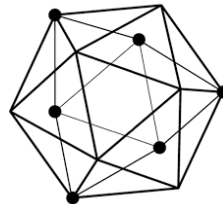
Aplicações baseadas em plataformas descentralizadas

Compõe a arquitetura de uma rede Blockchain composta principalmente por:

- Smart contract
- Middleware
- App (mobile/Web)



# Hyperledger Foundation



## Código Aberto

esforço colaborativo com o objetivo de melhorar as **tecnologias em blockchain** entre diversos segmentos empresariais

Administrado pela **Linux Foundation**, sendo o projeto de maior velocidade de crescimento na história da fundação.

**Colaboração global** entre empresas financeiras, bancos, IoT, logística, indústrias e tecnológicas

# Utilizando o readthedocs



latest

Search docs

 **HYPERLEDGER**

Read the Docs v: latest

Idiomas

en es fa fr ko it ja ml ru

vi zh\_CN pt

Versões

latest release-2.4 release-2.3

release-2.2 release-2.1 release-2.0

release-1.4 release-1.3 release-1.2

release-1.1

Downloads

HTML

No Read the Docs

Início do projeto Compilações

Downloads

No GitHub

Visualização Editar

Pesquisar

Pesquisar nos docs

Hospedado por Read the Docs · Política de privacidade

Docs » A Blockchain Platform for the Enterprise

[Edit on GitHub](#)

## Note

Please make sure you are looking at the documentation that matches the version of the software you are using. See the version label at the top of the navigation panel on the left. You can change it using selector at the bottom of that navigation panel.

## A Blockchain Platform for the Enterprise



Enterprise grade permissioned distributed ledger platform that offers modularity and versatility for a broad set of industry use cases.

- [Introduction](#)
- [What's new in Hyperledger Fabric v2.x](#)
- [Release notes](#)
- [Key Concepts](#)
- [Getting Started - Install](#)
- [Getting Started - Run Fabric](#)
- [Tutorials](#)
- [Deploying a production network](#)
- [Operations Guides](#)
- [Upgrading to the latest release](#)

# Configuração inicial

Esse curso irá utilizar o ambiente Linux para desenvolvimento.

- Git
- Curl
- Docker
- Docker images do Fabric (na versão desejada)
- GoLang
- NodeJs



# Verificando o docker

```
sudo systemctl enable docker
```

Limpando o docker

```
docker stop $(docker ps -a -q)
```

```
docker rm $(docker ps -a -q)
```

```
docker rmi -f $(docker images)
```

```
docker volume prune
```

```
docker system prune
```



# Fabric-samples e Docker images v2.2



Instalando tudo

```
curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.2.9 1.5.5
```

Repositório fabric-samples

<https://github.com/hyperledger/fabric-samples.git>

Imagens oficiais

<https://hub.docker.com/r/hyperledger/fabric-peer>

<https://hub.docker.com/r/hyperledger/fabric-orderer>

<https://hub.docker.com/r/hyperledger/fabric-orderer>

<https://hub.docker.com/r/hyperledger/fabric-tools>



# Test-network v2.2

Dentro do diretório *fabric-samples/test-network*

Utilizar o script *network network.sh*

Se não for a 1ª vez limpar o ambiente

*./network.sh down*

Subir os containers com o comando

*./network.sh up*





# Ferramenta *cryptogen*

Ferramenta de geração de MSPs simulados para ambiente de desenvolvimento

Utiliza arquivos de configuração (yaml)

Ex:

*PeerOrgs:*

- *Name: Org1*

  - Domain: org1.example.com*

  - EnableNodeOUs: true*

*Template:*

  - Count: 1*

  - SANS:*

    - *localhost*

*Users:*

  - Count: 1*



# Utilizando Fabric-CA

Se não for a 1ª vez limpar o ambiente

```
./network.sh up -ca
```

```
tree organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/
```

```
organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/  
├── msp  
│   ├── cacerts  
│   │   └── localhost-7054-ca-org1.pem  
│   ├── config.yaml  
│   ├── IssuerPublicKey  
│   ├── IssuerRevocationPublicKey  
│   ├── keystore  
│   │   └── 868e452a1fa47d12bfefbe306f7f2e9ebf383599c339914541b688b6ad5a3b6a_sk  
│   ├── signcerts  
│   │   └── cert.pem  
└── user
```



# Certificados gerados



Os certificados gerados para uso na rede Hyperledger Fabric são identificados:

**Common Name (CN):** nome e domínio do certificado

**Organization (O):** organização do certificado

**Organization Unit (OU):** peer, orderer, client, admin

```
openssl x509 -in localhost-7054-ca-org1.pem -text -noout
```

Certificate:ls

Data:

Version: 3 (0x2)

Serial Number:

5f:d1:6a:5f:a4:5f:0a:0f:a9:cd:1b:1c:d8:90:ee:38:9d:d3:b5:37

Signature Algorithm: ecdsa-with-SHA256

Issuer: C = US, ST = North Carolina, L = Durham, O = org1.example.com, CN = ca.org1.example.com

Validity

Not Before: Nov 7 17:15:00 2022 GMT

Not After : Nov 3 17:15:00 2037 GMT

Subject: C = US, ST = North Carolina, L = Durham, O = org1.example.com, CN = ca.org1.example.com

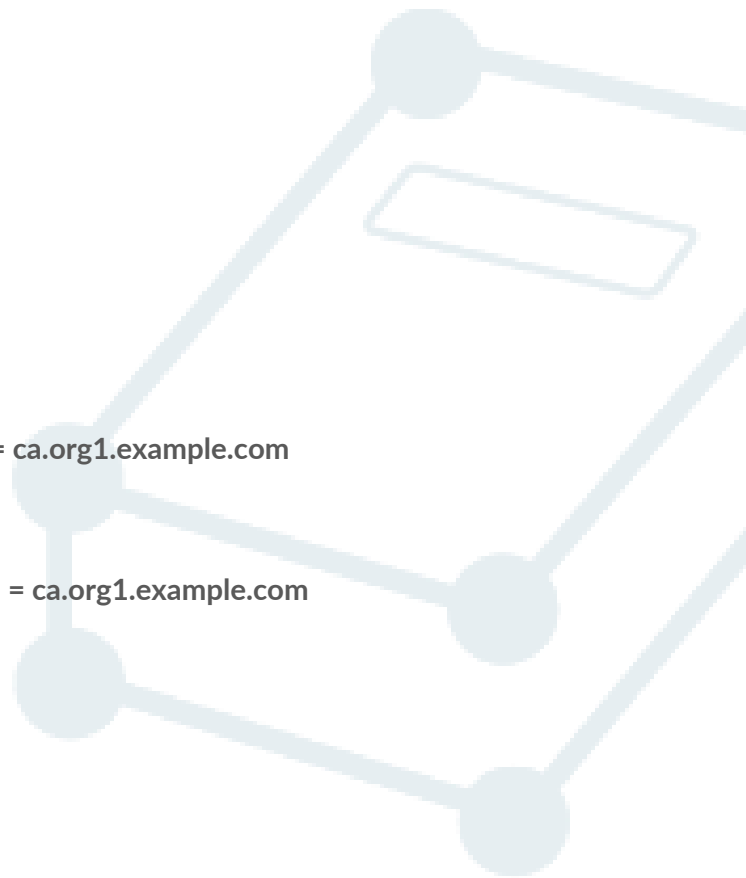
Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:c9:3a:2f:1f:65:2b:4f:b8:cb:8d:d9:52:63:36:...



# Arquitetura proposta

Rede – *blockchain business network* - a ser criada

3 orgs

- **org1**
- **org2**
- **orderer**

1 channel – *mychannel*

1 chaincode - *fabcar*



# Ferramenta *configtxgen*



Gera artefatos de configuração da rede e inspeção da rede

Os artefatos de configuração são propostos ao orderer para poder realizar ações tais como:

- Criar *genesis block*
- Criar novo channel
- Adicionar peer
- Adicionar nova organização

Configura o arquivo em *configtx.yaml*

Deve estar em *FABRIC\_CFG\_PATH* ou utilizar o parâmetro *-configPath*

No diretório *test-network*:

```
export FABRIC_CFG_PATH=$PWD/configtx
```



# Configtx.yaml

```
Welcome  configtx.yaml.txt  ! configtx.yaml X
C: > Users > Adm > Desktop > ! configtx.yaml
157 #####
158 #
159 #   SECTION: Application
160 #
161 #   - This section defines the values to encode into a config transaction or
162 #   genesis block for application related parameters
163 #
164 #####
165 Application: &ApplicationDefaults
166
167     # Organizations is the list of orgs which are defined as participants on
168     # the application side of the network
169     Organizations:
170
171     # Policies defines the set of policies at this level of the config tree
172     # For Application policies, their canonical path is
173     #   /Channel/Application/<PolicyName>
174     Policies:
175         Readers:
176             Type: ImplicitMeta
177             Rule: "ANY Readers"
178         Writers:
179             Type: ImplicitMeta
180             Rule: "ANY Writers"
181         Admins:
182             Type: ImplicitMeta
183             Rule: "MAJORITY Admins"
184         LifecycleEndorsement:
185             Type: ImplicitMeta
186             Rule: "MAJORITY Endorsement"
187         Endorsement:
188             Type: ImplicitMeta
189             Rule: "MAJORITY Endorsement"
190
191     Capabilities:
192         <<: *ApplicationCapabilities
193 #####
```

# Arquivo configtx.yaml

O arquivo configtx.yaml possui diversas sessões para realizar a criação de artefatos de configuração.

- **Organizations**
- **Capabilities (Channel, Orderer, Application)**
- **Application**
- **Orderer**
- **Channel**
- **Profiles**



# Criando o bloco gênese



O *genesis block* possui as regras iniciais de uma Blockchain business network. No exemplo ele é criado no profile *TwoOrgsOrdererGenesis*:

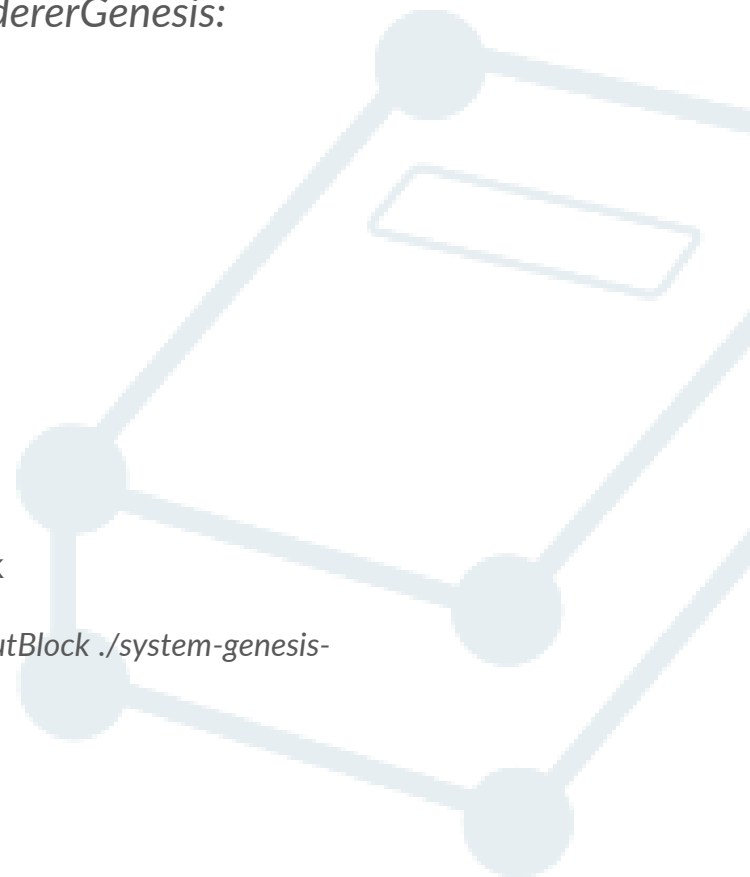
```
TwoOrgsOrdererGenesis:  
  <<: *ChannelDefaults  
  Orderer:  
    <<: *OrdererDefaults  
    Organizations:  
      - *OrdererOrg  
    Capabilities:  
      <<: *OrdererCapabilities  
  Consortiums:  
    SampleConsortium:  
      Organizations:  
        - *Org1  
        - *Org2
```

Comando para gerar o artefato de configuração do genesis block

```
configtxgen -profile TwoOrgsOrdererGenesis -channelID system-channel -outputBlock ./system-genesis-block/genesis.block
```

Inspecionando o bloco gênese

```
configtxgen -inspectBlock ./system-genesis-block/genesis.block
```





# Iniciando a Business Network v2.2



Inspecionando o bloco gênese

```
configtxgen -inspectBlock ./system-genesis-block/genesis.block
```

O orderer vai utilizar o *genesis block* para iniciar a rede.

Configuração do docker:

*orderer.example.com:*

*environment:*

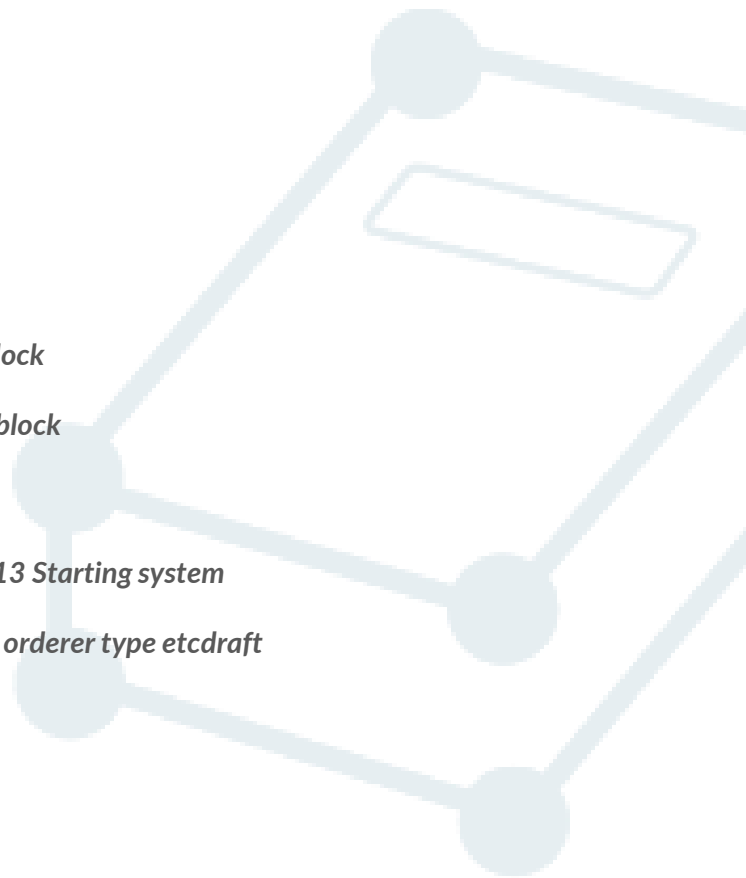
- *ORDERER\_GENERAL\_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block*

*volumes:*

- *../system-genesis-block/genesis.block:/var/hyperledger/orderer/orderer.genesis.block*

Configuração do docker:

2022-11-07 20:37:39.800 UTC [orderer.commmmon.multichannel] Initialize -> INFO 013 Starting system channel 'system-channel' with genesis block hash *ca7bb35a2fd70f7c4c86b8f0b778d45758e249f93ab8d47b8621dbe2a009e174* and orderer type etcdraft



# Criando um channel na rede



Utilizando o comando `configtxgen` para o profile **TwoOrgsChannel** para gerar a transação de criação de channel.

```
TwoOrgsChannel:  
  Consortium: SampleConsortium  
  <<: *ChannelDefaults  
  Application:  
    <<: *ApplicationDefaults  
    Organizations:  
      - *Org1  
      - *Org2  
    Capabilities:  
      <<: *ApplicationCapabilities
```

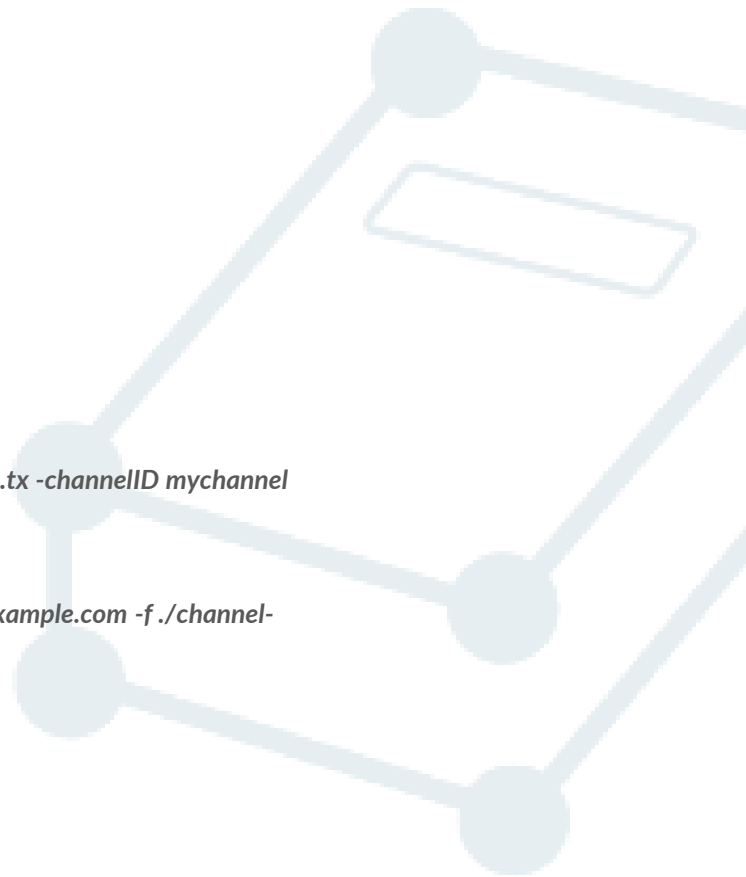
Criação da transação *mychannel.tx*

```
configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/mychannel.tx -channelID mychannel
```

Criando o channel na rede

```
peer channel create -o localhost:7050 -c mychannel --ordererTLSHostnameOverride orderer.example.com -f ./channel-artifacts/mychannel.tx --outputBlock ./channel-artifacts/mychannel.block --tls --cafile ...
```

Bloco de configuração *mychannel.block* criado.



# Entrada de um peer no channel

Entrada de um peer de uma org no channel através da aplicação do bloco de configuração.

```
setGlobals $ORG  
peer channel join -b ./channel-artifacts/mychannel.block
```

O peer dentro do channel representa a org dentro do channel

```
2022-11-07 22:52:52.152 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join  
channel
```



# Utilizando o container *cli*



Container cli é para da imagem hyperledger/fabric-tools

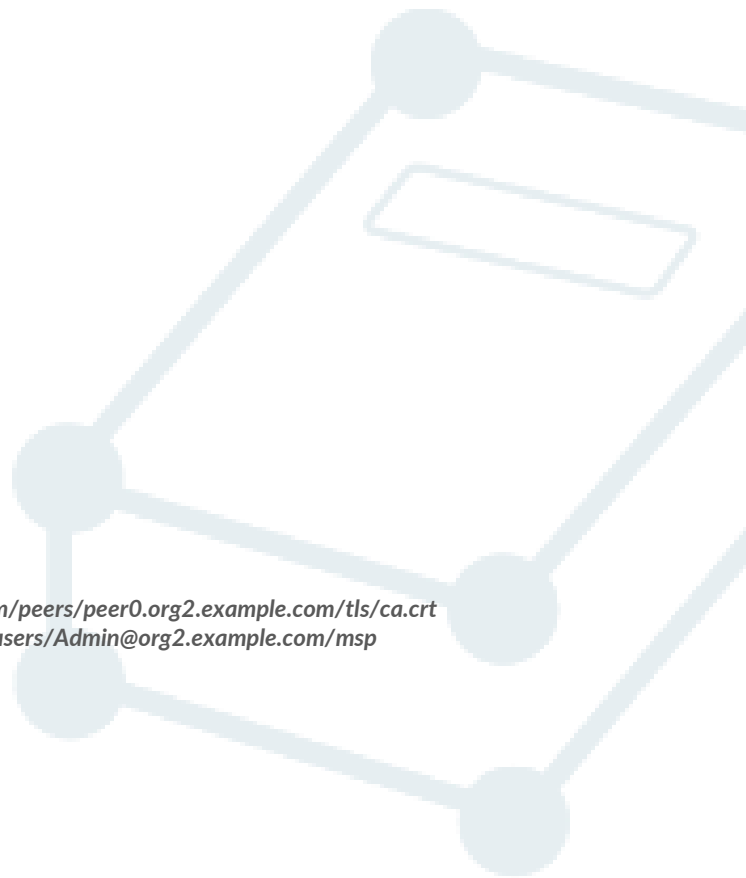
Possibilita a operação entre peers sem a mudança de containers

Variáveis de ambiente utilizadas pelo comando *peer*

**CORE\_PEER\_TLS\_ENABLED**  
**CORE\_PEER\_LOCALMSPID**  
**CORE\_PEER\_TLS\_ROOTCERT\_FILE**  
**CORE\_PEER\_MSPCONFIGPATH**  
**CORE\_PEER\_ADDRESS**

Exemplo

```
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
export CORE_PEER_ADDRESS=localhost:9051
```



# Atualizando blocos de configuração

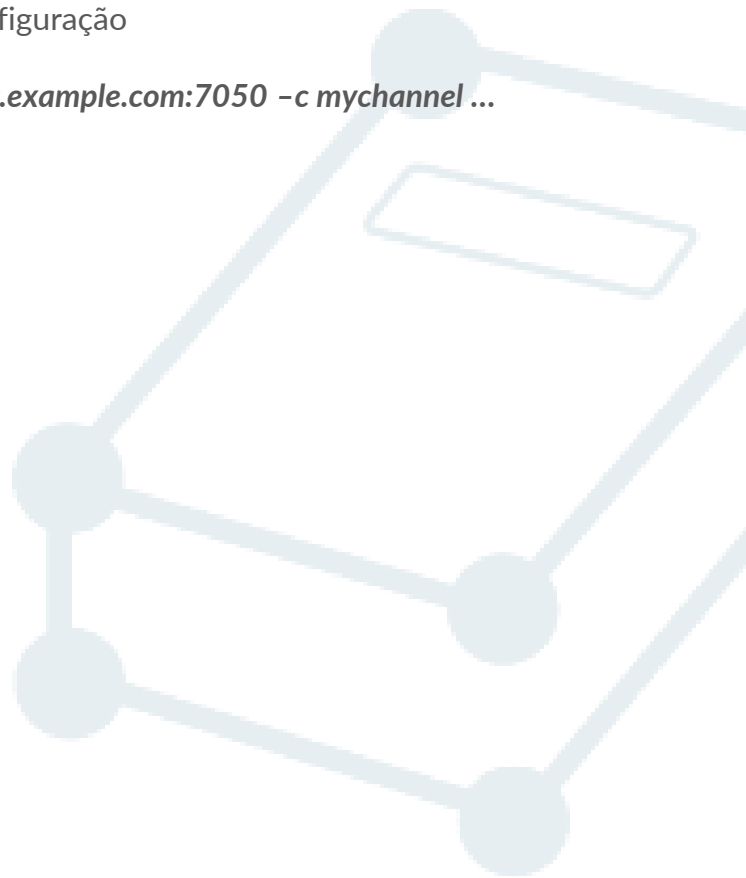


Algumas operações são realizadas através da atualização dos blocos de configuração

```
peer channel fetch config | newest | oldest | blockNumber outputFile -o orderer.example.com:7050 -c mychannel ...
```

O bloco de configuração é modificado e atualizado no channel

```
peer channel update -o orderer.example.com:7050 -c mychannel -f txFile
```



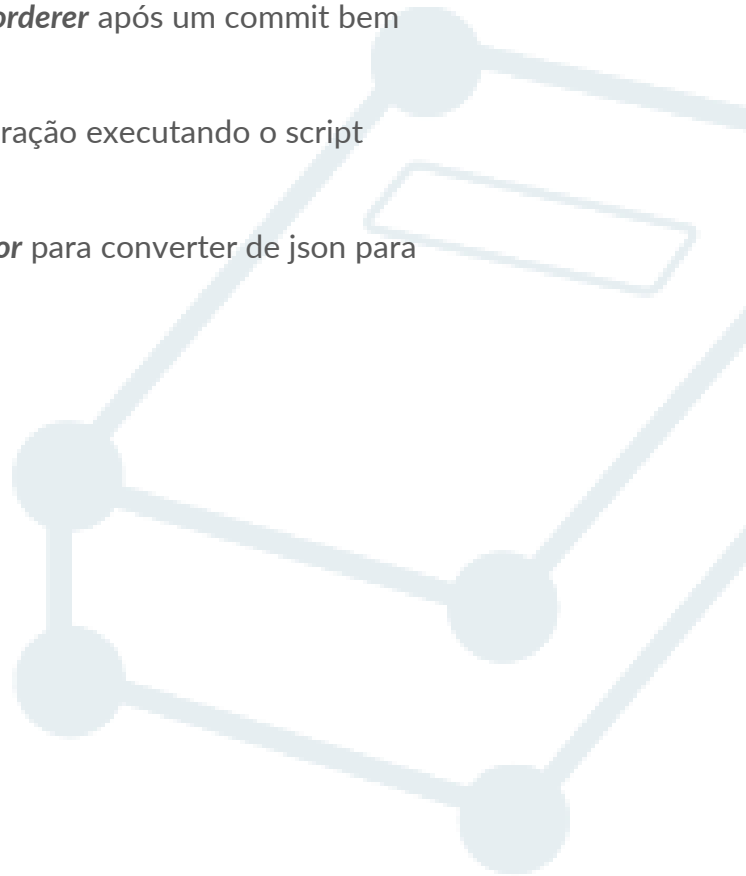
# Configuração do Anchor Peer



Os **anchor peers** representam os *peers* que vão receber os novos blocos do **orderer** após um commit bem sucedido.

A definição de anchor peer acontece com a atualização do bloco de configuração executando o script **setAnchorPeer.sh** dentro do container *cli*.

Atualização de blocos de configuração com o uso da ferramenta **configtxlator** para converter de json para protobuf.



# Chaincode lifecycle



Ciclo de vida dos chaincodes instanciados no *channel*

O channel possui uma política de endosso (*endorsing policy*) para a instanciação e atualização do chaincode no channel.

Cada chaincode possui uma *endorsing policy* para as suas transações.

Cada *endorsing policy* representa o conjunto mínimo de assinaturas para para completar a transação com sucesso. Ex:

*Maioria simples*

*Todos*

*Qualquer um*

*Regra lógica, ex: org1 and [org2 or org3]*

A instalação e instanciação do chaincode no channel requer as seguintes etapas:

**Package:** empacotamento do chaincode em um arquivo tar.

**Install:** instalação do chaincode nos endorsing peers.

**Approve:** aprovação do chaincode pelas orgs para validar o lifecycle endorsing policy

**Commit:** instanciar o chaincode no channel

**Init** (opcional): realizar uma transação inicial.



# Instalando um chaincode no channel (FabCar-go) GoLedger

As seguintes etapas serão realizadas após o comando:

```
./network.sh deployCC -ccn fabcar -ccp ../chaincode/fabcar/go -ccl go
```

Preparando o chaincode

```
go mod vendor
```

Criando um pacote para o peer

```
peer lifecycle chaincode package fabcar.tar.gz --path ../fabcar/go --lang golang --label fabcar_1.0
```

Instalando o chaincode no peer

```
setGlobals $ORG
```

```
peer lifecycle chaincode install fabcar.tar.gz
```

Verificando os chaincodes instalados

```
peer lifecycle chaincode queryinstalled
```

*Installed chaincodes on peer:*

*Package ID: fabcar\_1.0:6c5c429e8a6734ff978f54d12a1d9e5e5296663e20dcacff6dd2276bb0e8b12b, Label: fabcar\_1.0*

Um chaincode instalado precisa ser instanciado para transformar o **peer** em **endorsing peer**





# Instanciando um chaincode



Aprovando o chaincode

```
setGlobals $ORG
```

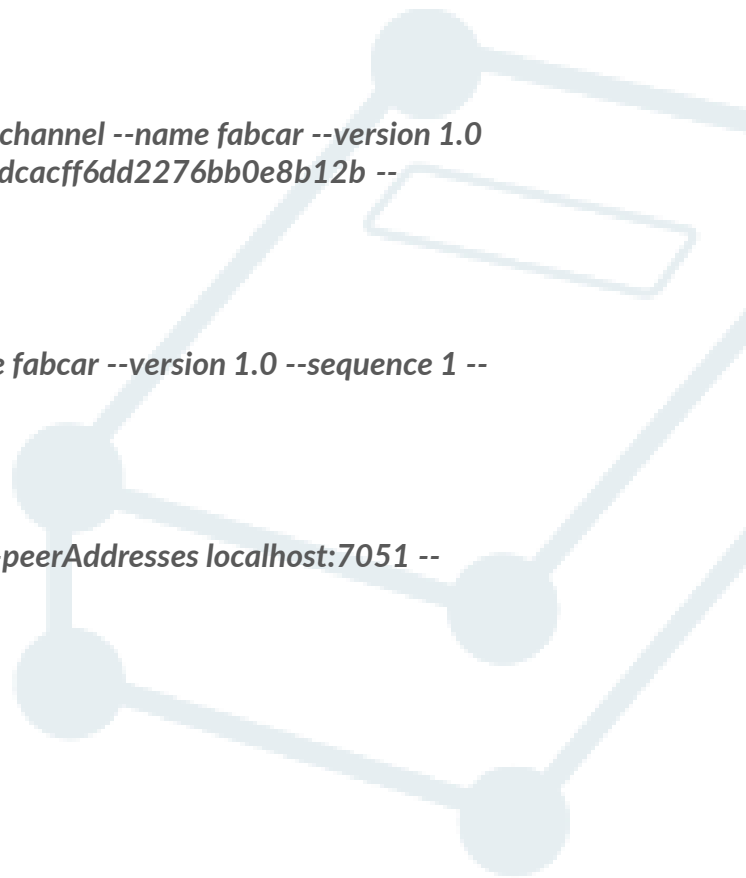
```
peer lifecycle chaincode approveformyorg -o localhost:7050 ... --channelID mychannel --name fabcar --version 1.0  
--package-id fabcar_1.0:6c5c429e8a6734ff978f54d12a1d9e5e5296663e20dcacff6dd2276bb0e8b12b --  
sequence 1
```

Aprovar em todas as orgs e verificar com a função:

```
peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name fabcar --version 1.0 --sequence 1 --  
output json
```

Comitar o chaincode

```
peer lifecycle chaincode commit -o localhost:7050 ... --channelID mychannel --peerAddresses localhost:7051 --  
peerAddresses localhost:9051 --name fabcar --version 1.0 --sequence 1
```



# Inicializando um chaincode

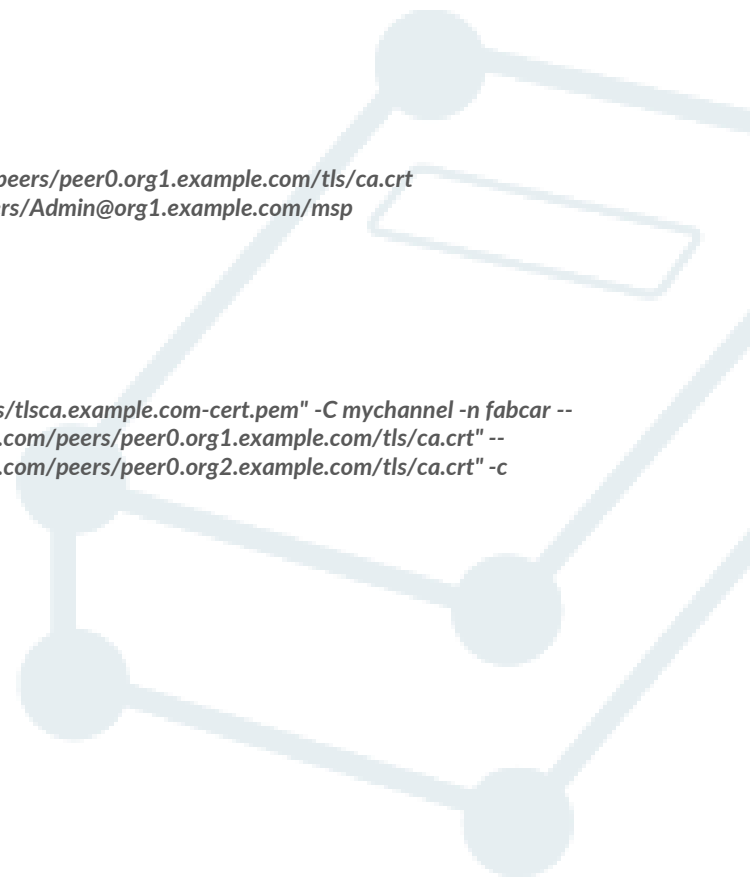


Configuração das variáveis de ambiente para inicializar o chaincode

```
export FABRIC_CFG_PATH=$PWD/../config/  
export CORE_PEER_TLS_ENABLED=true  
export CORE_PEER_LOCALMSPID="Org1MSP"  
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt  
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp  
export CORE_PEER_ADDRESS=localhost:7051
```

Chamando a primeira transação

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile  
"${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n fabcar --  
peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --  
peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c  
'{"function": "InitLedger", "Args": []}'
```



# Testando o chaincode

Verificando o estado do peer

```
peer chaincode query -C mychannel -n fabcar -c '{"Args":["QueryAllCars",""]}'
```

```
[{"Key": "CAR0", "Record": {"make": "Toyota", "model": "Prius", "colour": "blue", "owner": "Tomoko"}}, {"Key": "CAR1", "Record": {"make": "Ford", "model": "Mustang", "colour": "red", "owner": "Brad"}}, {"Key": "CAR2", "Record": {"make": "Hyundai", "model": "Tucson", "colour": "green", "owner": "JinSoo"}}, {"Key": "CAR3", "Record": {"make": "Volkswagen", "model": "Passat", "colour": "yellow", "owner": "Max"}}, {"Key": "CAR4", "Record": {"make": "Tesla", "model": "S", "colour": "black", "owner": "Adriana"}}, {"Key": "CAR5", "Record": {"make": "Peugeot", "model": "205", "colour": "purple", "owner": "Michel"}}, {"Key": "CAR6", "Record": {"make": "Chery", "model": "S22L", "colour": "white", "owner": "Aarav"}}, {"Key": "CAR7", "Record": {"make": "Fiat", "model": "Punto", "colour": "violet", "owner": "Pari"}}, {"Key": "CAR8", "Record": {"make": "Tata", "model": "Nano", "colour": "indigo", "owner": "Valeria"}}, {"Key": "CAR9", "Record": {"make": "Holden", "model": "Barina", "colour": "brown", "owner": "Shotaro"}}]
```

O operação query retorna o transaction proposal response.



# Chaincode FabCar

Contrato inteligente padrão *fabric-samples*

Gerenciamento de um conjunto de carros.  
Ativo principal compartilhado na rede.

```
// Car describes basic details of what makes up a car
type Car struct {
    Make string `json:"make"`
    Model string `json:"model"`
    Colour string `json:"colour"`
    Owner string `json:"owner"`
}
```



# Funções principais

*InitLedger* – inicia o channel com um conjunto de 10 carros

*CreateCar* – Cria um novo carro

*QueryCar* – Leitura dos dados de um carro

*QueryAllCars* – Retorna as informações de todos os carros do channel/chaincode.

*ChangeCarOwner* – Muda o proprietário de um carro

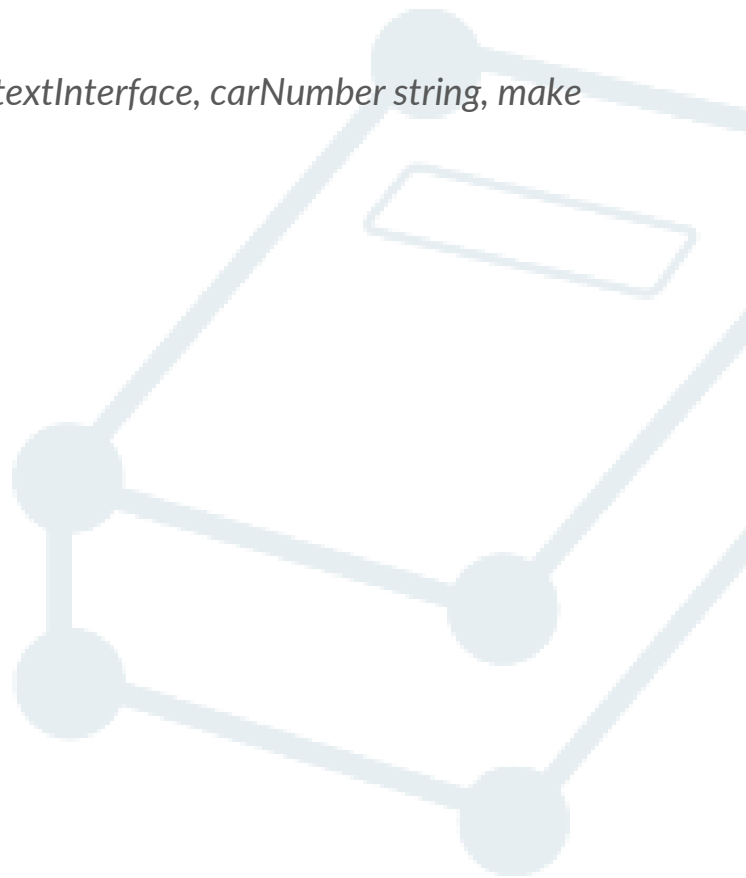


# Analizando a função CreateCar

```
func (s *SmartContract) CreateCar(ctx contractapi.TransactionContextInterface, carNumber string, make
string, model string, colour string, owner string) error {
    car := Car{
        Make:  make,
        Model: model,
        Colour: colour,
        Owner: owner,
    }

    carAsBytes, _ := json.Marshal(car)

    return ctx.GetStub().PutState(carNumber, carAsBytes)
}
```



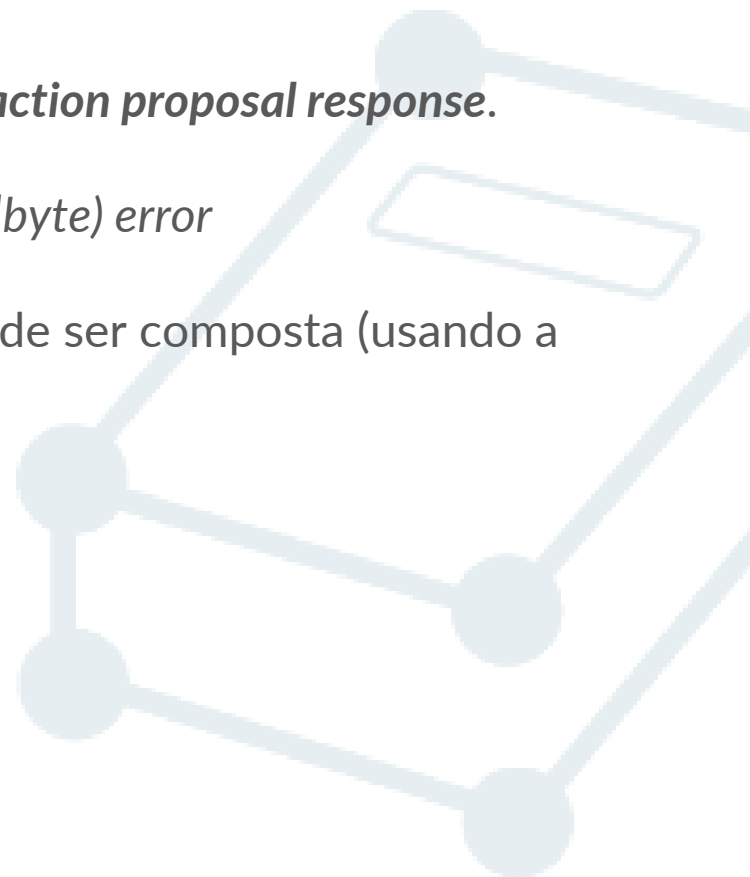
# A função PutState

A função **PutState** realiza uma alteração no *transaction proposal response*.

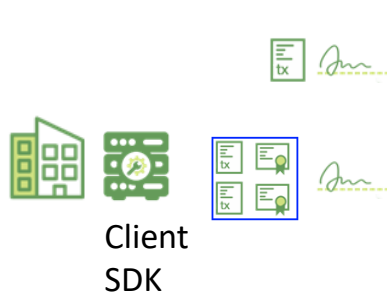
```
func (s *ChaincodeStub) PutState(key string, value []byte) error
```

**key**: chave única referência no channel. Chave pode ser composta (usando a função CreateCompositeKey)

**value**: dados a serem gravados no channel



## ORGANIZAÇÃO A



Endorsing  
Peer



Anchor  
Peer

## ORGANIZAÇÃO B



Endorsing  
Peer



Anchor  
Peer

## ORGANIZAÇÃO C (ORDERER)

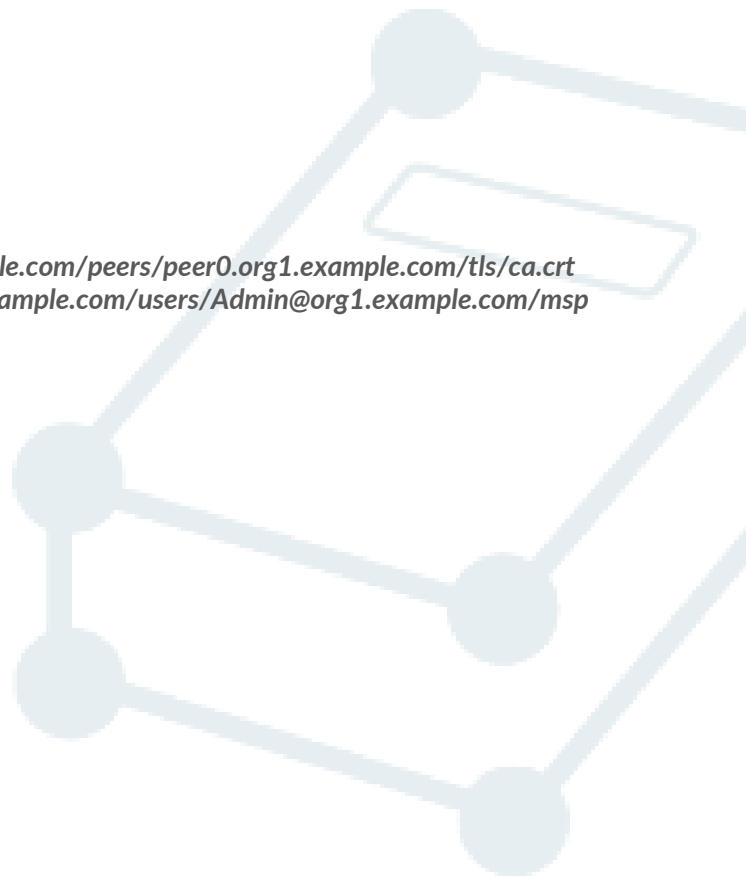




# Configuração de variáveis

Configuração das variáveis de ambiente para inicializar o chaincode

```
export FABRIC_CFG_PATH=$PWD/../config/  
export CORE_PEER_TLS_ENABLED=true  
export CORE_PEER_LOCALMSPID="Org1MSP"  
export  
CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt  
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp  
export CORE_PEER_ADDRESS=localhost:7051
```



# Chamando a função CreateCar



```
peer chaincode invoke \  
-o localhost:7050 \  
--ordererTLSHostnameOverride orderer.example.com \  
--tls --cafile  
"${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlsacerts/tlsca.example.com-cert.pem" \  
-C mychannel -n fabcar \  
--peerAddresses localhost:7051 \  
--tlsRootCertFiles  
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" \  
--peerAddresses localhost:9051 --tlsRootCertFiles  
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" \  
-c '{"function":"CreateCar","Args":["CAR10", "Gurgel", "Mini", "branco", "Marcos"]}'  
2022-11-09 18:40:39.119 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful.  
result: status:200
```

```
peer chaincode query -C mychannel -n fabcar -c '{"Args":["QueryCar","CAR10"]}'  
{ "make": "Gurgel", "model": "Mini", "colour": "branco", "owner": "Marcos" }
```

# Chamando a função QueryCar

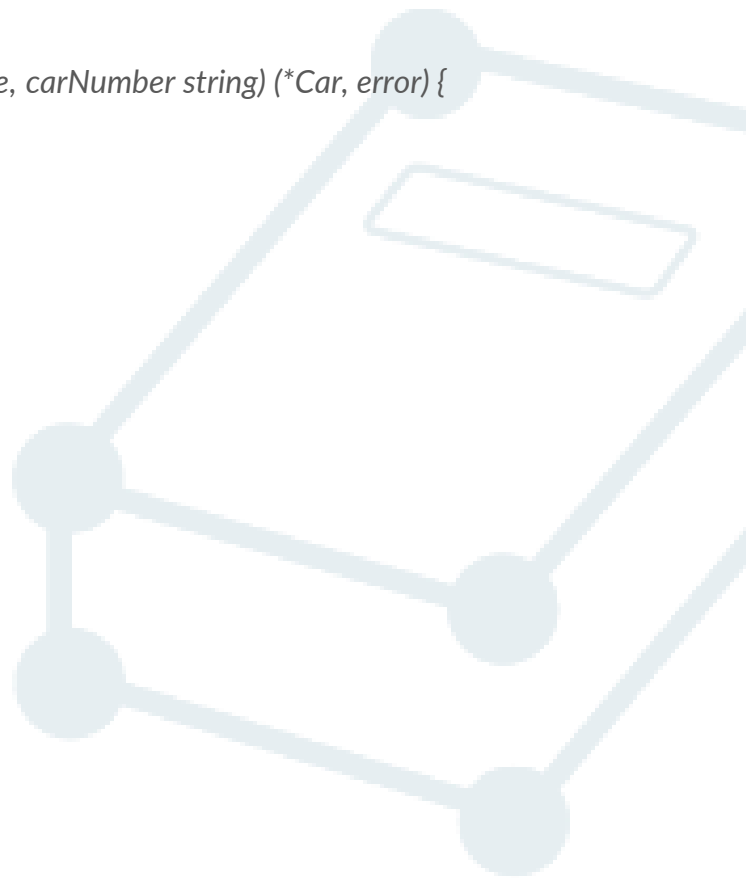
```
// QueryCar returns the car stored in the world state with given id
func (s *SmartContract) QueryCar(ctx contractapi.TransactionContextInterface, carNumber string) (*Car, error) {
    carAsBytes, err := ctx.GetStub().GetState(carNumber)

    if err != nil {
        return nil, fmt.Errorf("Failed to read from world state. %s", err.Error())
    }

    if carAsBytes == nil {
        return nil, fmt.Errorf("%s does not exist", carNumber)
    }

    car := new(Car)
    _ = json.Unmarshal(carAsBytes, car)

    return car, nil
}
```



# A função **GetState**

A função **GetState** retorna o world state do ativo presente no peer.

```
func (s *ChaincodeStub) GetState(key string) ([]byte, error)
```

key: chave única referência no channel.



# Lendo a informação de um carro

Verificando o estado do peer

```
peer chaincode query -C mychannel -n fabcar -c '{"Args":["QueryCar", "CAR2"]}'
```

O operação query retorna o transaction proposal response.



# Atualizando um chaincode no channel



Para atualizar um chaincode deve-se executar os passos similares a instanciação do chaincode.

```
./network.sh deployCC -ccn fabcar -ccp ../chaincode/fabcar/go/ -ccl go -ccv 2.0 -ccs 2
```

*Verificar a sintaxe GoLang do chaincode usando o comando*

***go vet***

Os chaincodes atualizados devem:

- Possuir versão inédita
- Sequenciado em relação ao chaincode anterior (1, 2, 3...)

Fluxo igual ao do instanciar.

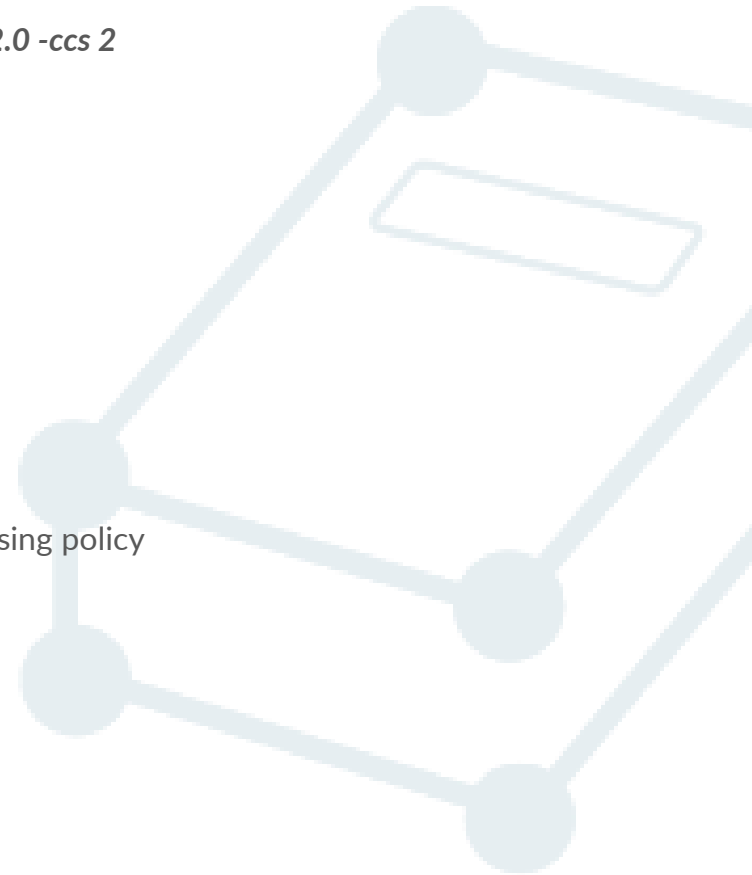
**Package:** empacotamento do chaincode em um arquivo tar.

**Install:** instalação do chaincode nos endorsing peers.

**Approve:** aprovação do chaincode pelas orgs para validar o lifecycle endorsing policy

**Commit:** instanciar o chaincode no channel

**Init** (opcional): realizar uma transação inicial.



# Biblioteca CC-Tools

# *Desenvolvendo chaincodes*

Mapeamento de dados

Gerenciamento de direito de escrita e leitura entre organizações (MSP)

Cadastramento de transações





# ***Biblioteca CC-Tools***

A biblioteca CC-Tools é desenvolvida em GoLang e possui diversas características que facilitam a jornada de aprendizado, desenvolvimento e deployment em produção de um chaincode para Hyperledger Fabric.

# ***Biblioteca CC-Tools - características***

Open source

Padronização de assets, keys e referências de assets (asset dentro de asset)

Tipos de dados padrão e customizáveis

Gerenciamento de organizações (MSP)

Cadastramento de Transações

Gerenciamento de permissões de escrita por propriedade de assets por MSP

Gerenciamento de private data.

# *Biblioteca CC-Tools - características*

Transações embedded:

- Create
- Read
- Update
- Delete
- Search (paginated)
- ReadHistory

Transações customizáveis

Integração das transações com a Rest API (GET, PUT, POST, DELETE)

Permissionamento de chamadas de transações por MSP

# Exemplo de Asset

```
// Description of a book
var Book = assets.AssetType{
  Tag: "book",
  Label: "Book",
  Description: "Book",
  Props: []assets.AssetProp{
    {
      IsKey: true, // Primary Key
      Tag: "title",
      Label: "Book Title",
      DataType: "string",
      Writers: []string{"org2MSP"}, // This means only org2 can create the asset (others can edit)
    },
    {
      Tag: "currentTenant",
      Label: "Current Tenant",
      DataType: "->person", /// Reference to another asset
    },
    {
      Tag: "genres",
      Label: "Genres",
      DataType: "[]string", // String list
    },
    {
      Tag: "published",
      Label: "Publishment Date",
      DataType: "datetime", // Date property
    },
  },
}
```

# Exemplo de Transação

```
// Updates the tenant of a Book
// POST Method
var UpdateBookTenant = tx.Transaction{
    Tag: "updateBookTenant",
    Label: "Update Book Tenant",
    Description: "Change the tenant of a book",
    Method: "PUT",
    Callers: []string{ $org\dMSP }, // Any orgs can call this transaction

    Args: []tx.Argument{
        {
            Tag: "book",
            Label: "Book",
            Description: "Book",
            DataType: "->book",
            Required: true,
        },
        {
            Tag: "tenant",
            Label: "tenant",
            Description: "New tenant of the book",
            DataType: "->person",
        },
    },
},
Routine: func(stub *sw.StubWrapper, req map[string]interface{}) ([]byte, errors.ICCErrors) {
    .....
    .....
    .....
    return bookJSON, nil
},
}
```

# Ferramentas

Testes unitários

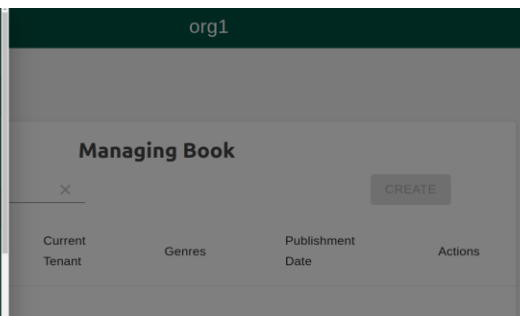
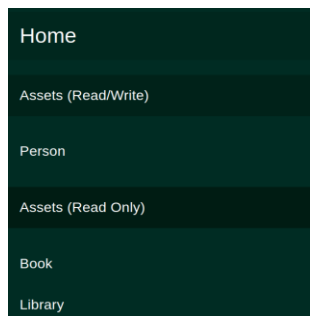
```
func TestCreateNewLibrary(t *testing.T) {
    stub := mock.NewMockStub("org3MSP", new(cc.CCDemo))

    expectedResponse := map[string]interface{}{
        "@key": "library:3cab201f-9e2b-579d-b7b2-72297ed17f49",
        "@lastTouchBy": "org3MSP",
        "@lastTx": "createNewLibrary",
        "@assetType": "library",
        "name": "Maria's Library",
    }
}
```

Web service (Rest API)  
integrado



Aplicação Web



# Repositório cc-tools-demo

Baixar o repositório

```
git clone https://github.com/goledgerdev/cc-tools-demo.git
```

Vendorar o chaincode no diretório *chaincode*

```
go mod vendor
```

Vendorar o web service no *diretório rest-server*

```
./npmlInstall.sh
```

# Iniciando o ambiente

Executar o script para HL Fabric 1.4 com 3 orgs

*./startDev.sh*

Executar o script para HL Fabric 2.2 com 3 orgs

*./startDev2.sh*

Executar o script para HL Fabric 1.4 com 1 org

*./startDev.sh -n 1*

Para subir a aplicação Web

*./run-web-cc.sh*



# cc-tools-demo – diretórios e scripts

<code>-- chaincode</code>	<i># chaincode source code</i>
<code> -- assettypes</code>	<i># definições dos assets</i>
<code> -- datatypes</code>	<i># tipos de dados customizados</i>
<code> -- txdefs</code>	<i># transações customizadas</i>
<code>-- rest-server</code>	<i># web service (REST API)</i>
<code>npmInstall.sh</code>	<i># vendorar web service</i>
<code>startDev.sh</code>	<i># reiniciar apis (1.4)</i>
<code>startDev2.sh</code>	<i># reiniciar apis (2.2)</i>
<code>-- fabric</code>	<i># artefatos HL Fabric 1.4</i>
<code>startDev.sh</code>	<i># iniciar rede HL 1.4</i>
<code>-- fabric2</code>	<i># artefatos HL Fabric 2.2</i>
<code>startDev.sh</code>	<i># iniciar rede HL 2.2</i>
<code>renameProject.sh</code>	<i># renomear nome do chaincode</i>
<code>startDev.sh</code>	<i># iniciar rede HL Fabric 1.4</i>
<code>startDev2.sh</code>	<i># iniciar rede HL Fabric 2.2</i>

# Definição de um asset

Definição dos assets dentro da pasta *assettypes*

```
var Person = assets.AssetType{
  Tag:      "person",           // Identificação do ativo (JSON)
  Label:    "Person",          // Texto identificador do ativo
  Description: "Personal data", // Texto identificador detalhado do ativo
  Props: []assets.AssetProp{
    {
      IsKey: true,              // Propriedade faz parte da chave primária/composta
      Tag:   "id",              // Identificação da propriedade (JSON/interface{})
      Label: "CPF (Brazilian ID)", // Texto identificador da propriedade
      DataType: "cpf",          // Tipo da propriedade (embedded ou custom)
      Writers: []string{`org1MSP`}, // Organização que pode criar/alterar a propriedade
    },
    {
      // Mandatory property
      Required: true,
      Tag:      "name",
      Label:    "Name of the person",
      DataType: "string",
      Validate: func(name interface{}) error { // Função de validação (criação ou alteração da propriedade)
        nameStr := name.(string)
        if nameStr == "" {
          return fmt.Errorf("name must be non-empty")
        }
      },
      return nil
    },
  },
}
```

# Lista de assets do chaincode

Os assets devem estar explicitamente definidos no arquivo *assetTypeList.go* na pasta *chaincode*

```
var assetTypeList = []assets.AssetType{  
    assettypes.Person,  
    assettypes.Book,  
    assettypes.Library,  
    assettypes.Secret,  
}
```

# Datatypes embedded

CC-Tools possui os seguintes *datatypes* padrão

<i>string</i>	<i># texto livre</i>
<i>number</i>	<i># numero flutuante</i>
<i>datetime</i>	<i># data e hora</i>
<i>boolean</i>	<i># true / false</i>

<i>[]string</i>	<i># array de texto livre</i>
<i>[]number</i>	<i># array numero flutuante</i>
<i>[]datetime</i>	<i># array data e hora</i>
<i>[]boolean</i>	<i># array de true / false</i>

<i>-&gt;[<a href="#">asset</a>]</i>	<i># referencia a outro asset</i>
<i>[]-&gt;[<a href="#">asset</a>]</i>	<i># array de referencias a assets</i>

# Definição de um Datatype custom

Definição dos *custom datatypes* dentro da pasta *datatypes*

```
var cpf = assets.DataType{
  Parse: func(data interface{}) (string, interface{}, errors.ICCError) {
    cpf, ok := data.(string)
    if !ok {
      return "", nil, errors.NewCCError("property must be a string", 400)
    }

    cpf = strings.ReplaceAll(cpf, ".", "")
    cpf = strings.ReplaceAll(cpf, "-", "")
    if len(cpf) != 11 {
      return "", nil, errors.NewCCError("CPF must have 11 digits", 400)
    }

    ...
    return cpf, cpf, nil
  },
}
```

# Lista de custom datatypes do chaincode

Os custom datatypes devem estar explicitamente definidos no arquivo ***datatypes.go*** na pasta *chaincode/datatypes*

```
var CustomDataTypes = map[string]assets.DataType{
    "cpf":    cpf,
    "bookType": bookType,
}
```

# Transações embedded

CC-Tools possui os seguintes transações embutidas automaticamente para uso.

<code>Tx.ReadAsset</code>	<i>// Ler ativo no world state</i>
<code>tx.CreateAsset</code>	<i>// Criar no ativo no channel</i>
<code>tx.UpdateAsset</code>	<i>// Atualizar ativo no channel</i>
<code>tx.DeleteAsset</code>	<i>// Deletar ativo no channel</i>
<code>tx.Search</code>	<i>// Procurar ativos com rich query no world state</i>
<code>tx.ReadAssetHistory</code>	<i>// Histórico de um ativo no ledger</i>

# Definição de uma transação custom

```
var CreateNewLibrary = tx.Transaction{
  Tag:      "createNewLibrary",           // Identificação da transação (API endpoint)
  Label:    "Create New Library",        // Texto identificador da transação
  Description: "Create a New Library",    // Texto identificador detalhado da transação
  Method:   "POST",                      // Método do web service
  Callers:  []string{"$org3MSP", "$orgMSP"}, // Organizações (MSP) que podem chamar essa transação
  Args: []tx.Argument{
    {
      Tag:      "name",                  // Identificação do argumento
      Label:    "Name",                  // Texto identificador do argumento
      Description: "Name of the library", // Texto identificador detalhado do argumento
      DataType:  "string",               // Tipo de dados do argumento
      Required:  true,                   // Argumento obrigatório (default=false)
    },
  },
},
Routine: func(stub *sw.StubWrapper, req map[string]interface{}) ([]byte, errors.ICCError) {
  name, _ := req["name"].(string) // Argumento do chamada
  libraryMap := make(map[string]interface{})
  libraryMap["@assetType"] = "library"
  libraryMap["name"] = name

  libraryAsset, err := assets.NewAsset(libraryMap) // Preparação do ativo para gravação
  if err != nil {...}
  _, err = libraryAsset.PutNew(stub) // Criação do transaction proposal response
  if err != nil {...}
  libraryJSON, nerr := json.Marshal(libraryAsset)
  if nerr != nil {...}
  return libraryJSON, nil // Retorna resultado para a API
},
```



# Lista de transações do chaincode

As transações, inclusive as embedded devem estar explicitamente definidas no arquivo ***txList.go*** na pasta ***chaincode***

```
var txList = []tx.Transaction{
    tx.CreateAsset,
    tx.UpdateAsset,
    tx.DeleteAsset,
    txdefs.CreateNewLibrary,
    txdefs.GetNumberOfBooksFromLibrary,
    txdefs.UpdateBookTenant,
    txdefs.GetBooksByAuthor,
}
```

# Utilizando o Web Service padrão (API)

Web service padronizado para realizar chamadas na rede Blockchain.

Modelo padrão:

Invoke (realiza alteração no channel)

Métodos: *POST, PUT, DELETE*

*api/invoke/:tx*

Query (não altera o channel):

Métodos: *GET, POST*

*api/query/:tx*

# Endpoints de apoio

## ***query/getHeader***

Método GET

Retorna versão do CC-Tools e dados do header para visualização na interface

## ***query/getSchema***

Métodos GET e POST

Retorna informações dos assets ou de um asset específico

## ***query/getTx***

Métodos GET e POST

Retorna informações das transações ou de uma transação específica

# Exemplo de getSchema

POST

/query/getSchema

Obtém a descrição de um tipo de asset específico.

Parameters

Cancel

No parameters

Request body

application/json

O assetType deve conter um tipo de asset definido pelo chaincode.  
Examples:

person

```
{  "assetType": "person"}
```

ExecuteClear

Responses

Request URL

http://localhost/api/query/getSchema

Server response

Code

Details

200

Response body

```
{ "tag": "person", "label": "Person", "description": "Personal data of someone", "props": [{"tag": "id", "label": "CPF (Brazilian ID)", "description": "", "isKey": true, "required": true, "readOnly": false, "dataType": "string", "writers": [{"orgMSP": "orgMSP"}]}, {"tag": "name", "label": "Name of the person", "description": "", "isKey": false, "required": true, "readOnly": false, "dataType": "string", "writers": null}, {"tag": "dateOfBirth", "label": "Date of Birth", "description": "", "isKey": false, "required": false, "readOnly": false, "dataType": "datetime", "writers": null}, {"tag": "height", "label": "Person's height", "description": "", "isKey": false, "required": false, "readOnly": false, "defaultValue": 0, "dataType": "number", "writers": null}]} 
```

Download

Response headers

# Endpoints para as tx embedded

*invoke/createAsset*

Método POST

*{"asset": [ {JSON do ativo}, {JSON do ativo}, ... ] }*

*invoke/readAsset*

Método POST

*{“key”: {“@assetType”: Tipo do ativo, Dados da chave } }*

*invoke/updateAsset*

Método PUT

*{“update”: {“@assetType”: Tipo do ativo, Dados da chave, Propriedades a serem alteradas } }*

*invoke/deleteAsset*

Método DELETE

*{“key”: {“@assetType”: Tipo do ativo, Dados da chave } }*

# Identificação de chaves dentro do asset

Propriedade da chave

*@key*

Asset deve ter a identificação do tipo do asset

*@assetType*

Para algumas operações a chave é calculada automaticamente.

Exemplo:

```
{ "key": { "@assetType": "person", "id": "318.207.920-48" } }
```

```
{ "key": { "@assetType": "person", "@key": "person:47061146-c642-51a1-844a-bf0b17cb5e19" } }
```

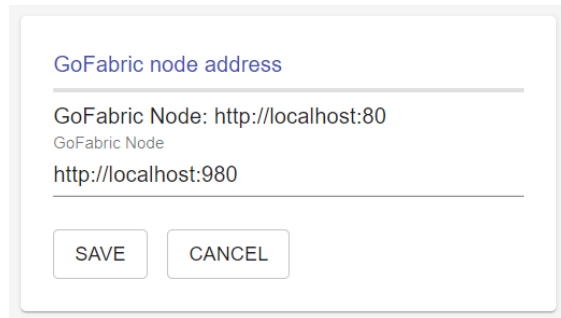
# Utilizando a ferramenta Web App (Golnitus)

Dentro do diretório raiz

`./run-cc-web.sh`

Acessar o browser <http://localhost:8080>

Configurar a interface para acessar a api correta (icone de ferramenta do header)



The image shows a configuration dialog box for a GoFabric node. It has a title bar that says "GoFabric node address". Below the title bar, there is a text input field containing "http://localhost:80". Below this, there is a label "GoFabric Node" and another text input field containing "http://localhost:980". At the bottom of the dialog, there are two buttons: "SAVE" and "CANCEL".

# Usando o Golnitus





The image shows a screenshot of the Golnitus web interface. On the left is a dark green sidebar with a list of menu items. On the right is a grey main content area. Blue arrows point from specific menu items to explanatory text on the right.

Menu Item	Description
org2	Organização (MSP) do web service
Assets (Read/Write)	Ativos que podem ser criados pela organização
Secret	Ativos em private data (PVC)
Assets (Read Only)	
Person	Ativos que não podem ser criados (talvez possam ser atualizados ou deletados)
Library	
Assets (Unreachable)	
Active Transactions	Transações que podem ser chamadas pela organização
Get Number Of Books From Library	
Update Book Tenant	
Get Books by the Author Name	
Non-Callable Transactions	
Create New Library	Transações que não podem ser chamadas pela organização

Additional text visible in the interface includes: "Welcome to CC Tools Demo" and "Version: 1.0.0".





# Trabalhando com ativos















org1

CURL

### Managing Person

 Search 

CREATE

CPF (Brazilian ID)	Name of the person	Date of Birth	Person's height	Actions
31820792048	Keyla	 11/11/2000, 0:00:00 GMT-2	1.6	   
12345678909	Maria		1.66	   
65852796115	Carlos	 11/11/1990, 0:00:00 GMT-2	1.8	   

## CURL

```
curl -X POST "http://localhost:80/api/query/search" -H 'Content-Type: application/json' -H 'cache-control: no-cache' -d '{"query":{"selector":{"@assetType":"person"},"limit":11,"bookmark":"","resolve":true}}
```




CANCEL

Chamada para operações de

- CreateAsset
- ReadAssetHistory
- ReadAsset
- UpdateAsset
- DeleteAsset

# Documentação CC-Tools

<https://golegger-cc-tools.readthedocs.io/en/latest/>



Search docs

Home

Welcome to CC-Tools documentation 🙌

Features

Getting Started

Tutorials

Assets

Transactions

External Tools

Testing

Functions

```
# .readthedocs.yaml
build:
  tools:
    python: "3.11"
sphinx:
  configuration: conf.py
python.install:
  - requirements: pip.in
```

Supercharge your Sphinx docs with deployment on Read the Docs. **Sign up today!**

Ad by EthicalAds · Host ads

Next »

[Docs](#) » [Home](#)

## Welcome to CC-Tools documentation 🙌

The **GoLedger CC-Tools** library is a collaborative effort from GoLedger to provide developers a powerful and easy-to-use library for creating Hyperledger Fabric chaincodes.

Developed using GoLang, the **GoLedger CC-Tools** library has several features that facilitate the journey of learning, development and deployment in production of a chaincode.

**GoLedger CC-Tools** is an open-source project lead by **GoLedger** and open for use to the Hyperledger development community.

Here's where you can start:

- [Getting Started](#)
- [Tutorials](#)
- Key concepts
  - [Assets](#)
  - [Transactions](#)
- [Reference guides](#)

*This documentation page is in constant development...*


Join our [Discord!](#)

## Features


- Standard asset data mapping (and their properties)
- Encapsulation of Hyperledger Fabric chaincode sdk interface functions

# Documentação Biblioteca GoLang

<https://pkg.go.dev/github.com/goledgerdev/cc-tools@v0.7.5>

 Search packages or symbols


Why Go ▾ Learn Docs ▾ Packages C


Discover Packages > github.com/goledgerdev/cc-tools > assets 


**assets** package


Version: v0.7.5 Latest | Published: Sep 14, 2022 | License: Apache-2.0 | Imports: 16 | Imported by: 4

Details

 Valid go.mod file ?

 Redistributable license ?

 Tagged version ?


 Stable version ?


[Learn more](#)

Repository

[github.com/goledgerdev/cc-tools](https://github.com/goledgerdev/cc-tools)

Links

 Report a Vulnerability

 Open Source Insights



Documentation

Index

Constants

Variables

▸ Functions

▸ Types

Source Files

<> Documentation

Index

```
func CustomDataTypes(m map[string]DataType) error
func DataTypeMap() map[string]DataType
func GenerateKey(asset map[string]interface{}) (string, errors.ICCError)
func InitAssetList(l []AssetType)
func PutNewRecursive(stub *sw.StubWrapper, object map[string]interface{}) (map[string]interface{}, errors.ICCError)
func PutRecursive(stub *sw.StubWrapper, object map[string]interface{}) (map[string]interface{}, errors.ICCError)
func StartupCheck() errors.ICCError
func UpdateRecursive(stub *sw.StubWrapper, object map[string]interface{}) (map[string]interface{}, errors.ICCError)
```

# Pacotes

O pacote *asset* é responsável pelas funções relativas ao gerenciamento de ativos no channel.

O pacote *transactions* é responsável pelas funções relativas as transações.

# Pacotes

O pacote ***asset*** é responsável pelas funções relativas ao gerenciamento de ativos no channel.

Todas as funções do pacote ***asset*** validam as definições dos ativos criados no diretório *assettypes*.

Tipos (type) do pacote *asset*:

***Asset*** : objeto *map[string]interface{}* de um ativo completo com as suas propriedades.

***Key*** : objeto *map[string]interface{}* das informações da chave do asset.

O pacote ***transactions*** é responsável pelas funções relativas as transações.

O pacote *transaction* possui as transações embedded.

# Criando/alterando ativos no channel

As principais funções para criar/alterar um *asset* (variação do *PutState*) são:

***func NewAsset(m map[string]interface{}) (a Asset, err errors.ICCError)***

Prepara o objeto tipo Asset para ser utilizando em operações futuras. Esperar

***func (a \*Asset) PutNew(stub \*sw.StubWrapper) (map[string]interface{}, errors.ICCError)***

Cria novo asset. Se asset já existir, retorna erro.

***func (a \*Asset) Put(stub \*sw.StubWrapper) (map[string]interface{}, errors.ICCError)***

Cria asset. Se asset já existir, atualiza

***func (a \*Asset) Update(stub \*sw.StubWrapper, update map[string]interface{}) (map[string]interface{}, errors.ICCError)***

Atualiza asset. Se asset não existir, retorna erro

Todas as transações de alteração de ativos validam as regras registradas na definição dos *assets* (*assettypes*)

# Lendo ativos

As principais funções para ler um **asset** são (variações do `GetState`):

***func NewAsset(m map[string]interface{}) (a Asset, err errors.ICCError)***

Prepara o objeto tipo `Asset` para ser utilizado em operações futuras.

***func (k \*Key) ExistsInLedger(stub \*sw.StubWrapper) (bool, errors.ICCError)***

Verifica a existência de um `asset`.

***func (a \*Asset) Get(stub \*sw.StubWrapper) (\*Asset, errors.ICCError)***

Busca um **asset**, caso exista, retorna em um objeto `Asset`

***func (k \*Key) GetMap(stub \*sw.StubWrapper) (map[string]interface{}, errors.ICCError)***

Busca um **asset**, caso exista, retorna em um objeto `map[string]interface{}`

***func (k \*Key) GetBytes(stub \*sw.StubWrapper) ([]byte, errors.ICCError)***

Busca um **asset**, caso exista, retorna em um objeto `[]byte`

***func (a \*Asset) GetRecursive(stub \*sw.StubWrapper) (map[string]interface{}, errors.ICCError)***

Busca um **asset**, caso exista, retorna em um objeto `map[string]interface{}` com todas as referências resolvidas.

# Transação CC-Tools

Uma transação definida no CC-Tools possui as seguintes características.

Identificação para chamadas Rest API

Tag: *“nome da transação”*

Método da Rest API

Method: *“POST” | “GET” | “PUT” | “DELETE”*

Argumentos da transação

Args: *[]tx.Argument{*  
    {  
        Tag: *“nome do argumento”,*  
        Label: *“Descrição do argumento”,*  
        Description: *“Descrição detalhada do argumento”,*  
        DataType: *“tipo do argumento”,*  
        Required: *true,*  
    },...

Função para executar a transação

Routine: *func(stub \*sw.StubWrapper, req map[string]interface{}) ([]byte, errors.ICCError) {...*



# Lendo os argumentos recebidos na transação

Os argumentos enviados pela API podem ser lidos da seguinte forma:

```
variavel, _ := req["nome do argumento"].(tipo do argumento)
```

Exemplo:

```
name, _ := req["name"].(string) // argumento name de tipo string
```

```
name, ok := req["name"].(string) // argumento name verificando se está é string
```

```
limit, _ := req["limit"].(float64) // argumento do tipo number
```

```
birthDate, _ := req["birthdate"].(time.Time) // argumento do tipo datetime
```

```
check, _ := req["check"].(bool) // argumento do tipo boolean
```

```
libraryKey, _ := req["library"].(assets.Key) // argumento do tipo referencia a um asset (->asset)
```

```
names, _ := req["names"].([]string) // argumento name de tipo []string
```

```
limits, _ := req["limits"].([]float64) // argumento do tipo []number
```

```
birthDate, _ := req["birthdate"].([]time.Time) // argumento do tipo []datetime
```

```
librarys, _ := req["librarys"].([]interface{}) // argumento do tipo array de referencias a um asset ([]->asset)
```

# Retornando um resultado da transação

Caso a transação falhe, deve-se retornar um erro usando o pacote *errors*

Exemplos:

```
if !ok { errors.NewCCError("type parameter is missing or in wrong format", 400) }
```

```
if err { return nil, errors.WrapError(err, " parameter missing") }
```

Caso a função não tenha apresentado falhas, deve-se retornar um *[]byte*.

Exemplo:

```
ret, _ := json.Marshal(asset) // Retornando um json  
return ret, nil
```

Uma transação bem sucedida NÃO garante a atualização do channel.

# Datatypes customizados

Datatypes podem ter tipos específicos para os chaincodes. São definidos na pasta datatypes

Um datatype custom possui as seguintes características:

*AcceptedFormats: []string{"tipos de datatypes"} // opcional*

*DropDownValues: map[string]interface{}{lista do dropdown} //opcional*

*Description: // descrição do datatype custom*

*Parse: função de validação do datatype*

# Validando um valor com datatype custom

Dentro da função *Parse*, caso o valor tenha alguma inconsistência deve-se retornar erro utilizando o pacote *errors*

Exemplo:

```
return "", nil, errors.NewCCError("asset property must be an integer", 400)
```

Em caso de sucesso, deve retornar o valor em formato string e o valor real

Exemplo:

```
return fmt.Sprintf(retVal), retVal, err
```

# Atualizando o chaincode

Para atualizar o chaincode em ambiente de validação deve usar o script:

*upgradeCC.sh <version> # para HL Fabric 1.4*

*upgradeCC2.sh <version> <sequence> # para HL Fabric 2.2*

# Queries no channel

As buscas são feitas dentro do peer através de operações de pesquisa.

As pesquisas são realizadas nas bases de *state* escolhida (**LevelDb** ou **CouchDb**)

**LevelDb** é a base default e roda dentro do container peer

Pesquisas em **LevelDb** são realizadas para as chaves (primárias ou compostas)

**CouchDb** é um modelo mais poderoso e roda em um container separado do *peer*.

*Rich queries* e indexação são permitidas para **CouchDb**

# Trabalhando com CouchDb

O uso do CouchDb pode ser realizado com a configuração dos endorsing peer

Váriáveis de ambiente:

`CORE_LEDGER_STATE_STATEDATABASE=CouchDB`

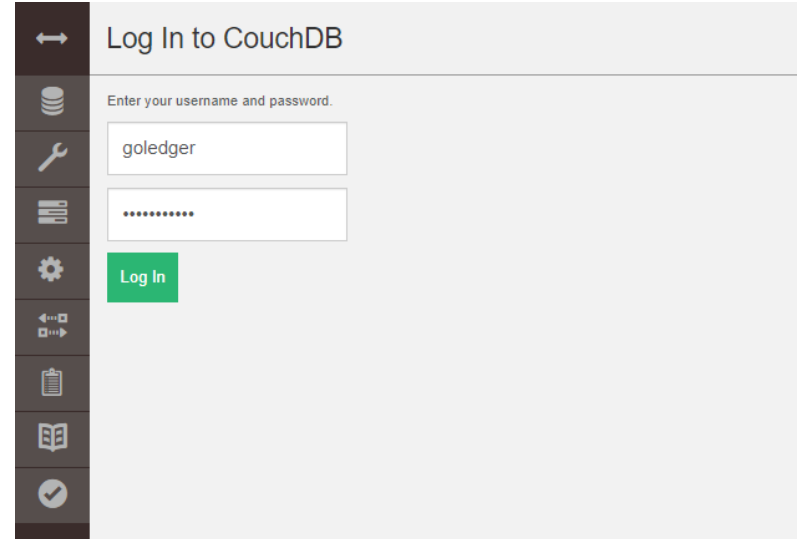
`CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couch.peer0.org.example.com:5984`


`CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin`

`CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw`

Exemplo de acesso ao couchdb via interface Web (*Fauxion*)

[http://localhost:5984/\\_utils/#login](http://localhost:5984/_utils/#login)





## Log In to CouchDB

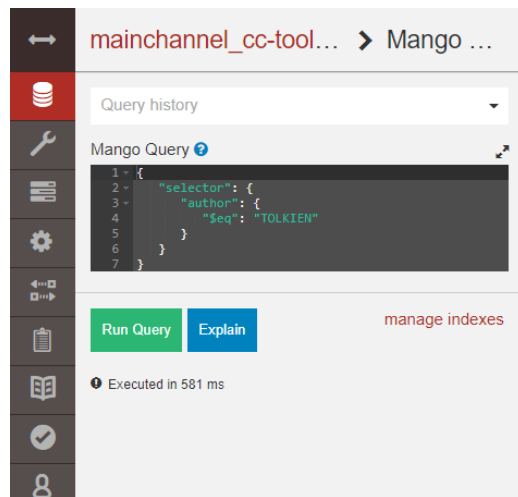
Enter your username and password.

Log In

# Usando a interface do CouchDB

O base do *state* é identificada pelo nome do *channel* junto com o nome do *chaincode*.

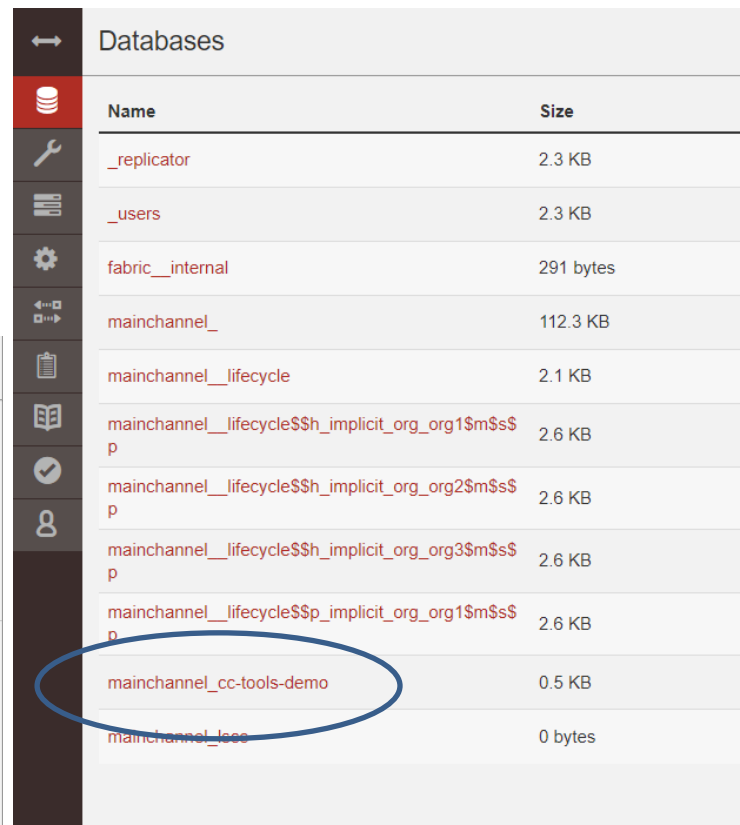
Queries podem ser executadas da interface através do item *Run Query with Mango*



The screenshot shows the Mango query interface. The breadcrumb path is 'mainchannel\_cc-tool... > Mango ...'. The 'Query history' dropdown is set to 'Mango Query'. The query editor contains the following JSON:

```
1 {
2   "selector": {
3     "author": {
4       "$eq": "TOLKIEN"
5     }
6   }
7 }
```

Below the editor are 'Run Query' and 'Explain' buttons. A status message indicates 'Executed in 581 ms'. A 'manage indexes' link is visible on the right.



The screenshot shows the 'Databases' interface with a list of databases. The 'mainchannel\_cc-tools-demo' database is circled in blue.

Name	Size
<a href="#">_replicator</a>	2.3 KB
<a href="#">_users</a>	2.3 KB
<a href="#">fabric__internal</a>	291 bytes
<a href="#">mainchannel__</a>	112.3 KB
<a href="#">mainchannel__lifecycle</a>	2.1 KB
<a href="#">mainchannel__lifecycle\$\$h_implicit_org_org1\$m\$s\$p</a>	2.6 KB
<a href="#">mainchannel__lifecycle\$\$h_implicit_org_org2\$m\$s\$p</a>	2.6 KB
<a href="#">mainchannel__lifecycle\$\$h_implicit_org_org3\$m\$s\$p</a>	2.6 KB
<a href="#">mainchannel__lifecycle\$\$p_implicit_org_org1\$m\$s\$p</a>	2.6 KB
<a href="#">mainchannel_cc-tools-demo</a>	0.5 KB
<a href="#">mainchannel__logs</a>	0 bytes



# Tabela de Query CouchDb

Queries podem ser realizadas através de uma sintaxe. Maiores detalhes na documentação oficial do CouchDb

<https://docs.couchdb.org/en/3.2.2-docs/api/database/find.html>

Exemplo:

Busca documentos com status “draft”

```
"selector": {  
  "status": { "$eq": "draft" }  
}
```

Busca asset com year maior ou igual a 2020

```
"selector": {  
  "year": { "$gte": 1900 }  
}
```

# Função Search

Função Search utilizando pesquisas realizadas no CouchDB.

Exemplo:

```
// Prepare couchdb query
```

```
query := map[string]interface{}{  
    "selector": map[string]interface{}{  
        "@assetType": "book",  
        "author":    authorName,  
    },  
}
```

```
response, err := assets.Search(stub, query, "", true)
```

```
...
```

# Utilizando funções de pesquisa

O Hyperledger Fabric possui diversas funções para realizar queries dentro do chaincode. Essas funções trabalham com iterators dentro de loops.

Exemplos:

*GetQueryResult*

*GetQueryResultWithPagination*

*GetStateByPartialCompositeKey*

*GetHistoryForKey*

Exemplo:

```
resultsIterator, _ = stub.GetQueryResult(queryString)
for resultsIterator.HasNext() {
    queryResponse, err := resultsIterator.Next()

```

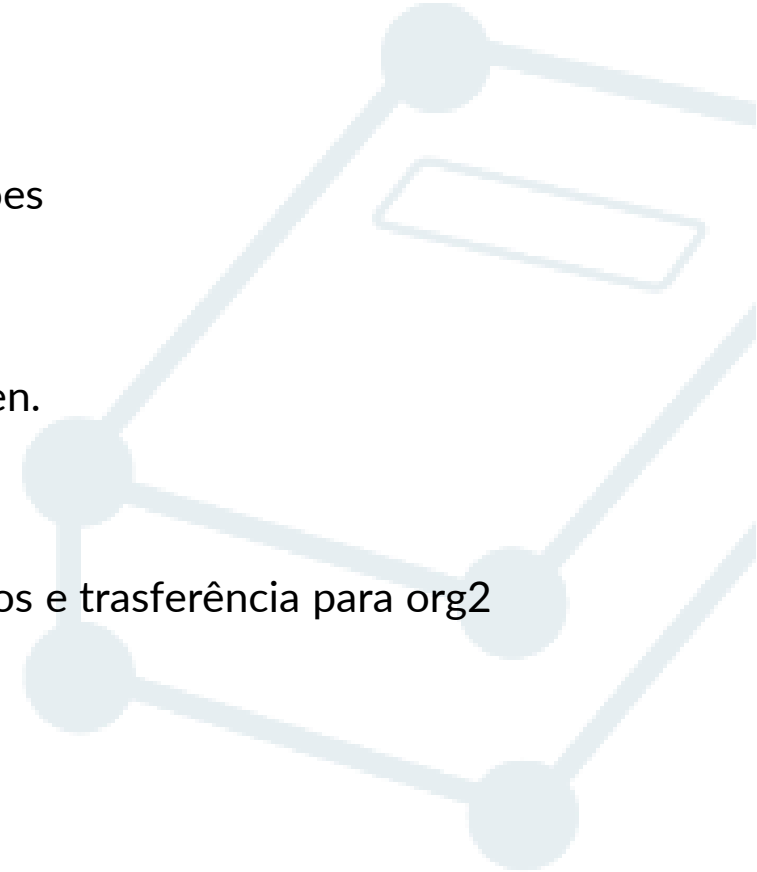
...

# Tarefa



# Tarefa Criando um Token

- Objetivo
- Começar com 1 org (`./startDev.sh -n 1`)
- Criar um token para transferencia entre organizações
  - Definitions do token
  - Método de criação de token
  - Método de transferência particionada do token.
  - Função de contabilidade do token.
  - Refazer a rede com 3 orgs
  - Limitar método de criação para org1 e métodos e transferência para org2 (contabilidade sem restrições)



## Definição do token

- Ativo Token
- Deve possuir os campos:
  - id (string) -> chave
  - proprietário (string) -> obrigatório
  - quantidade (number) -> default 0

## Método criar token

- Método POST
- Argumentos -> iguais a definição do token
- Validação:
- Quantidade deve ser maior ou igual a 0



## Método transferir token

- Método PUT
- Argumento destinatario (string)
- NÃO VERIFICAR quantidade
- Criar um novo token para o destinatário.
- Reduzir a quantidade do token origem

## Método de contabilidade

- Método GET
- Argumento proprietario
- Usar a função Search



# Private Data Collections

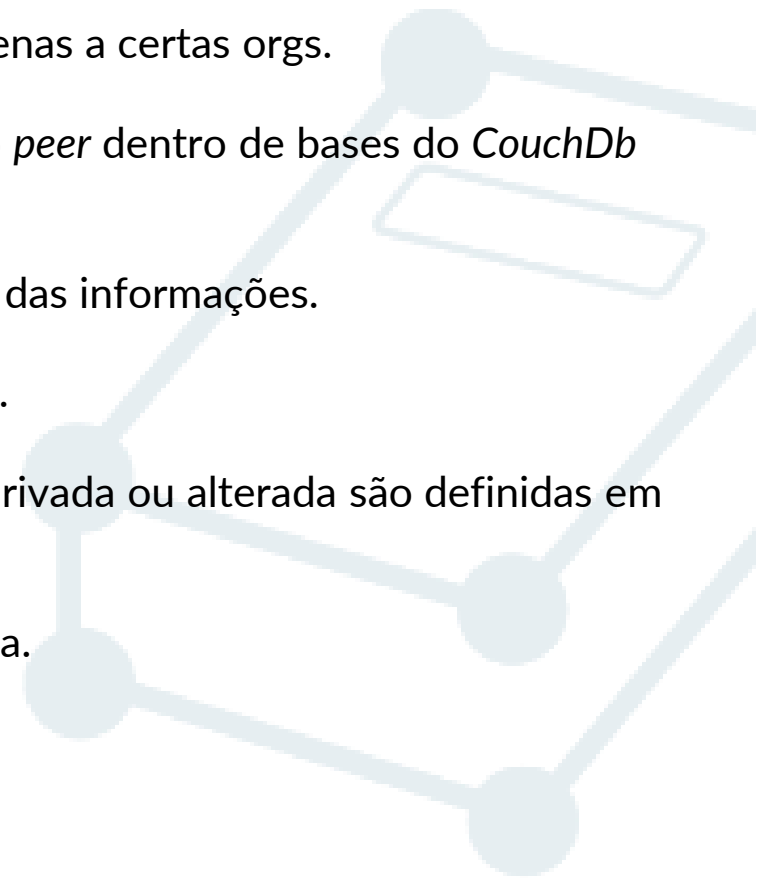
A biblioteca CC-Tools possui mecanismos para trabalhar com PDCs.

Assets que terão as informações gravadas em PDCs devem ter o operador *Readers*.



# Private Data Collection

- Um *channel* pode conter informações acessíveis apenas a certas orgs.
- Essas informações ficam gravadas externamente ao *peer* dentro de bases do *CouchDb* em um banco de dados transiente.
- Dentro do *ledger* ficam registradas apenas os hashes das informações.
- Esse conceito é chamado de ***Private Data Collection***.
- As organizações que podem acessar a informação privada ou alterada são definidas em uma *Private Data Collection Policy*.
- Uma Private Data Collection pode ter tempo de vida.



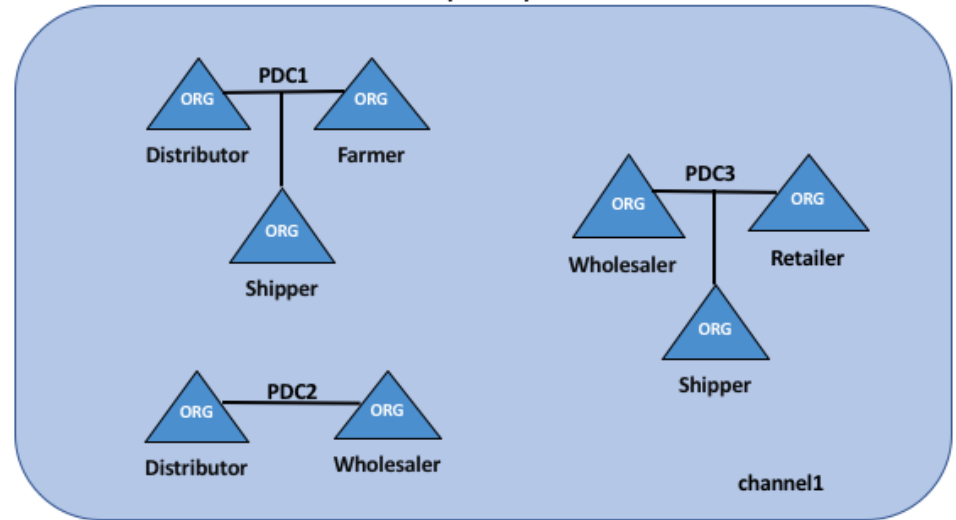
# Private Data Collections

- Caso o transação de um *chaincode* faça uso de PDC, o fluxo da transação possui etapa extra para geração de um TRANSACTION PROPOSAL RESPONSE.
- Transação possui leitura ou escrita em um PDC então:
- Gravação da informação privada gravada em base de *state* separada dentro do *CouchDB*.
- Replicação das informações privadas entre *peers* das organizações.
- PDC possui um Policy para definir as orgs que podem ter acesso a informação privada.
- Possibilidade de utilização de Transient Data para que os argumentos da transação não sejam gravados no bloco.
- *Transaction flow* de uma transação que usa PDC é mais complexo.

# Exemplo de um channel com PDCs

- PDC1: Distributor, Farmer and Shipper
- PDC2: Distributor and Wholesaler
- PDC3: Wholesaler, Retailer and Shipper

Private data collections  
(PDC)



PVC Policy (A or B; mínimo 2 peers)



### ORGANIZAÇÃO A



Endorsing Peer

Método Chaincode  $f(x)$

...

`PutPrivateData(asset)`



~~TransientData~~  
**World State**  
[asset]



### ORGANIZAÇÃO B



Peer



~~TransientData~~  
**World State**  
[asset]

### ORGANIZAÇÃO C

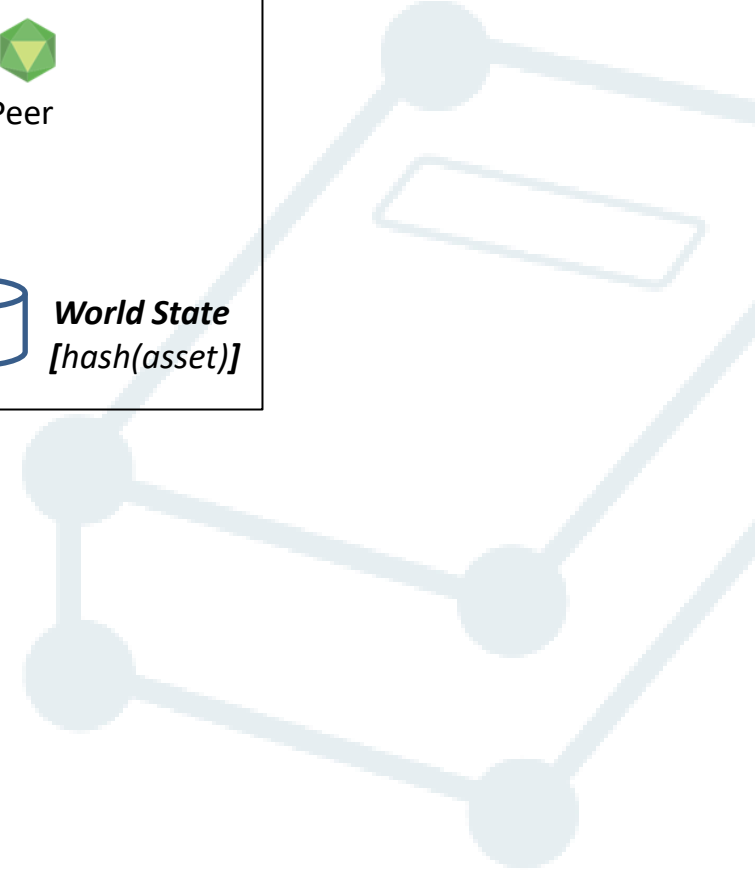


Peer



**World State**  
[hash(asset)]

### ORGANIZAÇÃO ORDERER



# Exemplo de um asset em PDC

```
var Secret = assets.AssetType{
    Tag:      "secret",
    Label:    "Secret",
    Description: "Secret between Org2 and Org3",
    Readers: []string{"org2MSP", "org3MSP"},
    Props: []assets.AssetProp{
        {
            IsKey: true,
            Tag:   "secretName",
            Label: "Secret Name",
            DataType: "string",
            Writers: []string{"org2MSP"}, // This means only org2 can create the asset (org3 can edit)
        },
        {
            Tag:   "secret",
            Label: "Secret",
            DataType: "string",
        },
    },
}
```

# Arquivo collections2.json

Cada *asset PDC (Readers)* deve ter um elemento correspondente dentro do arquivo *collections2.json*

```
[  
  {  
    "name": "secret",  
    "requiredPeerCount": 0,  
    "maxPeerCount": 3,  
    "blockToLive": 1000000,  
    "memberOnlyRead": true,  
    "policy": "OR('org2MSP.member', 'org3MSP.member')"  
  }  
]
```

D

# **Orquestradores Blockchain**

# Orquestradores Hyperledger Fabric

Estão disponíveis diversos orquestrados para facilitar a operação em redes Hyperledger Fabric

- Orquestradores open source
- Orquestradores em nuvem
- Orquestradores licenciados



# Orquestradores open-source

Disponibilizados pela Hyperledger Foundation

Hyperledger Cello

<https://www.hyperledger.org/use/cello>

Hyperledger Bevel

<https://www.hyperledger.org/use/bevel>

Fabric Operator (Hyperledger Labs)

<https://labs.hyperledger.org/labs/hlf-operator.html>

# Orquestradores em nuvem e licenciados

Disponíveis na nuvens públicas

Amazon Managed Blockchain

<https://aws.amazon.com/pt/managed-blockchain/>

IBM Blockchain Platform

<https://www.ibm.com/products/blockchain-platform-hyperledger-fabric>

Oracle Blockchain Platform Service

<https://www.oracle.com/br/blockchain/cloud-platform/>

Binario Cloud Blockchain (GoFabric engine)

<https://binario.cloud/produtos/blockchain>

# GoFabric

Orquestrador disponibilizado pela empresa GoLedger

<https://gofabric.io/>

- Lista de redes Hyperledger Fabric
- Dashboard de rede
- Gestão multichannel
- Operações de escalabilidade da rede (*addPeer*, *AddOrg*, etc)
- Integração automática com chaincodes desenvolvidos com a biblioteca CC-Tools
- Atualização de chaincodes
- Atualização de APIs
- Mapeamento de dados dos chaincodes utilizando *Templates* de dados


# GoFabric

Orquestrador disponibilizado pela empresa GoLedger

<https://gofabric.io/>

- Lista de redes Hyperledger Fabric
- Dashboard de rede
- Gestão multichannel
- Operações de escalabilidade da rede (*addPeer*, *AddOrg*, etc)
- Integração automática com chaincodes desenvolvidos com a biblioteca CC-Tools
- Atualização de chaincodes
- Atualização de APIs
- Mapeamento de dados dos chaincodes utilizando *Templates* de dados

# GoFabric – Dashboard

GOFABRIC

DASHBOARD


NETWORK


NODES


CHAINCODES

API


SETUP MACHINES







MSARRES



GoFabric networks

✓ Networks loaded

digital-certificate-id-

cert-channel

id-channel

New network

DIGITAL-CERTIFICATE-ID-

CHANNELS

cert-channel

Orgs

3

Peers

4

Orderers

5

id-channel

Orgs

3

Peers

4

Orderers

5

ORGANIZATIONS

international-student-id.org

Peers

2

Orderers

1

APIs

2

Joined Channels

- cert-channel
- id-channel

social-admin.gov

Peers

1

Orderers

1

APIs

1

Joined Channels

- id-channel

## CERT-CHANNEL

4

ACTIVE PEERS

3

ORGS

5

ORDERERS

FABRIC VERSION

2.2

CHAINCODE

certificates

VERSION

1.0

international-student-id.org

ca: 45.225.25.191

• orderer0: 45.225.25.191

• peer0: 45.225.25.191

• peer1: 45.225.25.240

ccapi:  
none  
[45.225.25.240](#)

ENTER GRAFANA

stanford-university.edu

ca: 45.225.25.74

• orderer0: 45.225.25.74

• peer0: 45.225.25.74

ccapi:  
[45.225.25.74](#)

ENTER GRAFANA

ucla.edu

ca: 45.225.25.218

• orderer0: 45.225.25.218

• peer0: 45.225.25.218

ccapi:  
[45.225.25.218](#)

# GoFabric – Data Mapping Templates

The screenshot displays the GoFabric web application interface for managing data mapping templates. The top navigation bar includes links to DASHBOARD, NETWORK, NODES, CHAINCODES, API, SETUP MACHINES, and a user profile icon. The left sidebar shows the 'GoFabric networks' section with a dropdown menu currently set to 'digital-certificate-id-'. Below this, there are links for 'cert-channel', 'id-channel', and a '+ New network' button. The main content area is titled 'TEMPLATES' and is divided into two panels: 'TEMPLATE LIST' and 'EDIT TEMPLATE'.

**TEMPLATE LIST**

Template Name	Edit	Delete
certificateTemplate		
certificates		
idTemplate		

**EDIT TEMPLATE**

**Name**  
idTemplate

**Description**

**Assets** CREATE NEW

Label	Tag	
Passport	passport	

**Fields**

Label	Tag	Is key	Required	Read only	
Passport ID	id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Name	name	<input type="checkbox"/>	<input checked="" type="checkbox"/>		

**Data type**  
String

**Dropdown menu:**

- Number
- String
- Boolean

**Field types:**

- List of Number
- List of String
- List of Boolean

# GoFabric – deployment na nuvem



## ORGANIZATIONS

international-student-id × social-admin × stanford-university × state-department × >

ADD MORE

ORGANIZATION BASIC INFO

Organization name: international-student-id Domain name: org

CA INFO

CA address: 45.225.25.191 CA user: admin

IP or domain address for CA: You must save this

CA password: \*\*\*\*\* Repeat password: Must be equal to CA password

INSTANCES

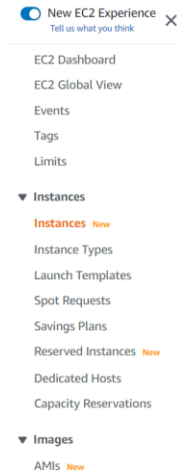
Machine configuration

Instance address: +

☒ Peer ☐ Orderer

Instance address: 45.225.25.191 ✓ Peer ✓ Orderer

Instance address: 45.225.25.240 ✓ Peer



Instances (3) info

Search

Instance state = running X Clear filters

<input type="checkbox"/>	Name	Instance ID	Instance state	Public IPv4 ...
<input type="checkbox"/>	-	i-0009f0a932eddf9c5	Running	3.129.44.196
<input type="checkbox"/>	goprocess-audit	i-02cdb10f594c407be	Running	3.135.62.89
<input type="checkbox"/>	goprocess-regi...	i-077650e507dbb8c94	Running	18.221.179.150

Select an instance

# GoFabric – multichannel

**GoFabric** DASHBOARD NETWORK NODES CHAINCODES API SETUP MACHINES ? MSARRES M

GoFabric networks  
✓ Networks loaded

digital-certificate-id-  
cert-channel  
id-channel  
+ New network

Define Organizations ✓ 2 Define Channels 3 Define Chaincodes 4 Start network

CHANNELS

cert-channel x id-channel x

ADD CHANNEL

CHANNEL BASIC INFO

Channel name  
cert-channel

Peers Available


- international-student-id
- social-admin
- 45.225.25.120 +
- stanford-university
- state-department

Peers in Channel

- international-student-id
- 45.225.25.191 ✗
- 45.225.25.240 ✗
- stanford-university
- 45.225.25.74 ✗



# GoFabric – Chaincode marketplace

 GOFABRIC

GoFabric networks  
✓ Networks loaded

digital-certificate-id- ▾

cert-channel

id-channel

+ New network

cert-channel id-channel

certificates ×

ADD CHAINCODE

Chaincode name  
certificates

TEMPLATE

CLOUD CHAINCODES

CHAINCODE FILE

Select the template chaincode ⓘ

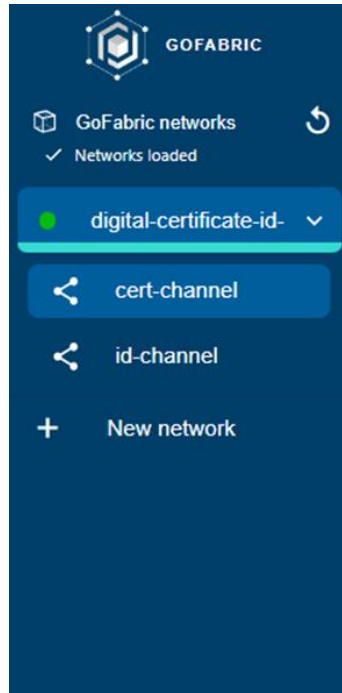
GOSHARE

Select the user template chaincode ⓘ

GOBIO

GOPROCESS

# GoFabric – Private Chaincode



## PERMISSIONS

**i** If no permissions are set, every organization in the network will be able to read and write to every asset

Passport

Asset

international-student-id, stanford-university

☒ Private data **i**

☒ international-student-id

☐ social-admin

☒ stanford-university

☐ state-department

☐ ucla

Private data

Passport ID string

Prop

All enabled

# GoFabric – Atualizar Chaincode

**GoFABRIC** DASHBOARD NETWORK NODES CHAINCODES API SETUP MACHINES

GoFabric networks  
✓ Networks loaded

digital-certificate-id-  
cert-channel  
id-channel  
+ New network

## UPGRADE CHAINCODE

certificates

Chaincode version  
2.0

☒ AUTO VERSION UPGRADE

CHANNEL PEERS UPGRADE

international-student-id

- ☒ peer0.international-student-id.org - 45.225.25.191
- ☒ peer1.international-student-id.org - 45.225.25.240

stanford-university

Select the template chaincode ⓘ

GOSHARE

Select the user template chaincode ⓘ

GOBIO GOPROCESS

Add your template

certificates

PERMISSIONS

ⓘ If no permissions are set, every organization in the network will be able to read and write to every asset

Document Hash

Asset

Course

Private data

Private