

Módulo 5

Hyperledger Fabric

(parte 4)



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.

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"
}
```

Execute Clear

Responses

Request URL
http://localhost/api/query/getSchema

Server response

Code	Details
200	<div>Response body</div> <pre>{ "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 }] }</pre> <div>Response headers</div>

 Download **ESCOLA
SUPERIOR
DE REDES**

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" }  
}
```

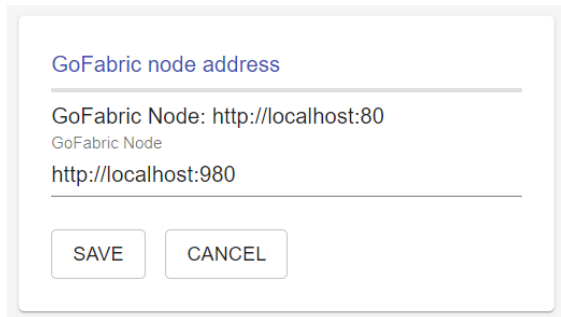
Utilizando a ferramenta Web App (*Golnitus*)

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)

A screenshot of a configuration dialog box titled "GoFabric node address". It contains two input fields. The first field is labeled "GoFabric Node:" and has the value "http://localhost:80". The second field is labeled "GoFabric Node" and has the value "http://localhost:980". Below the input fields are two buttons: "SAVE" and "CANCEL".

GoFabric node address

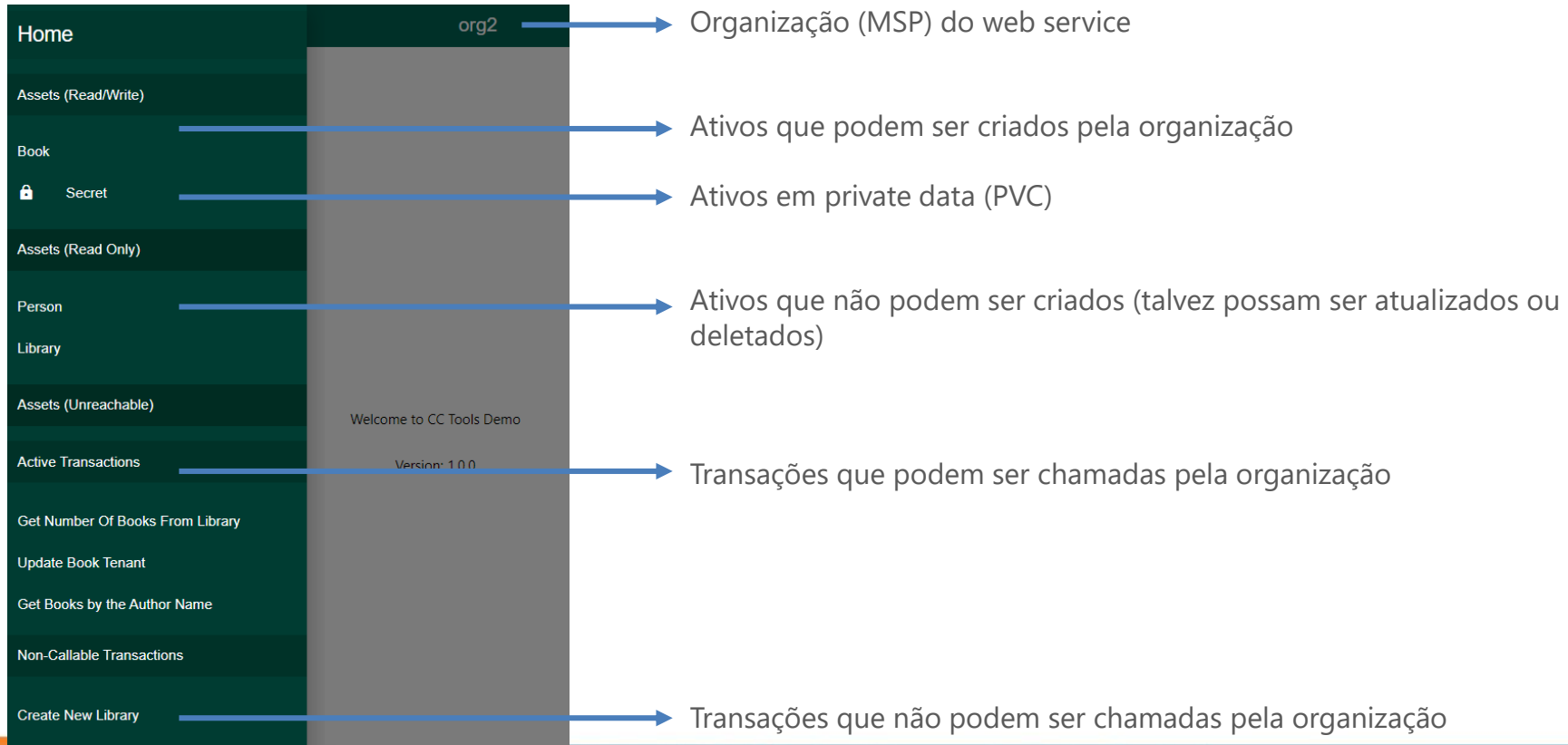
GoFabric Node: http://localhost:80

GoFabric Node

http://localhost:980

SAVE CANCEL

Usando o Golnitus



Trabalhando com ativos

The screenshot shows a web application interface for 'Managing Person'. At the top, there is a green header bar with a hamburger menu, navigation arrows, a refresh icon, a settings gear, and the text 'org1'. Below the header, there is a 'CURL' button. The main content area is titled 'Managing Person' and includes a search bar with a magnifying glass icon and a close 'X' icon. To the right of the search bar is a green 'CREATE' button. Below these elements is a table with the following columns: 'CPF (Brazilian ID)', 'Name of the person', 'Date of Birth', 'Person's height', and 'Actions'. The table contains three rows of data. The first row has CPF '31820792048', Name 'Keyla', Date of Birth '11/11/2000, 0:00:00 GMT-2', Height '1.6', and Actions icons (refresh, view, edit, delete). The second row has CPF '12345678909', Name 'Maria', Height '1.66', and Actions icons. The third row has CPF '65852796115', Name 'Carlos', Date of Birth '11/11/1990, 0:00:00 GMT-2', Height '1.8', and Actions icons. Blue arrows point from the 'CURL' button to a separate box, from the 'CREATE' button to a list of operations, and from the Actions icons of the first row to the same list of operations.

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://goledger-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

Links

[Report a Vulnerability](#) [Open Source Insights](#)

f

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 utilizando 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

Tarefa

PRÓXIMO MÓDULO:

6

**Hyperledger Fabric
(parte final)**



GoLedger