ESCOLA SUPERIOR DE REDES

Módulo 4
Hyperledger Fabric

GoLedger

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

Analisando 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)
}
```

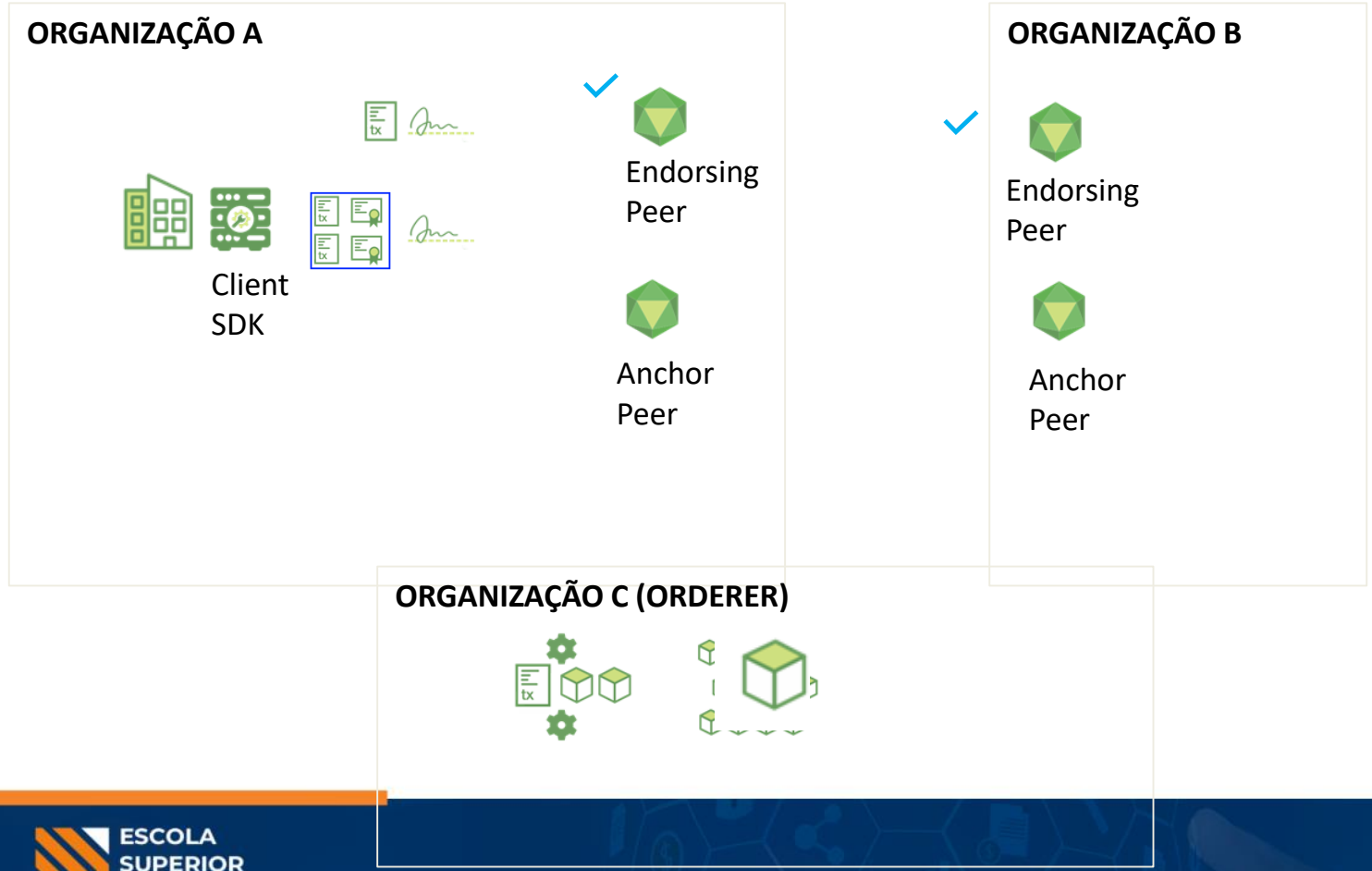
A função PutState

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
Client
SDK
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/m
sp
export CORE_PEER_ADDRESS=localhost:7051
```

Chamando a função CreateCar

```
peer chaincode invoke \
  -o localhost:7050 \
  --ordererTLSHostnameOverride orderer.example.com \
  --tls --cafile
"${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" \
  -C mychannel -n fabcar \
  --peerAddresses localhost:7051 \
  --tlsRootCertFiles
"${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":"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.

Desenvolvendo chaincodes

Mapeamento de dados

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

Cadastramento de transações

Biblioteca CC-Tools

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.ICCError) {
      ..........
      ..........
      ..........
    return bookJSON, nil
  },
}
```

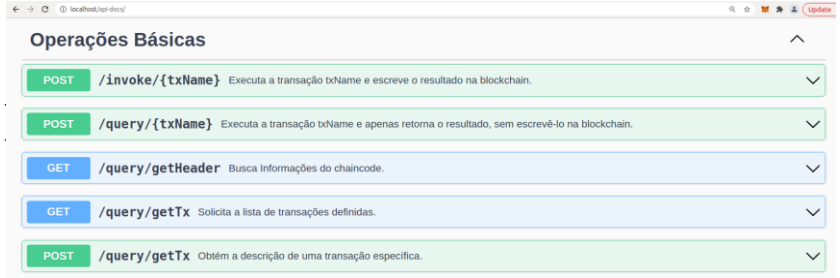
Ferramentas

Testes unitários

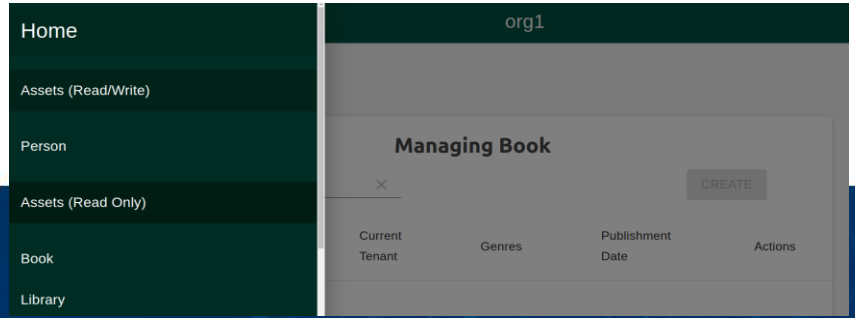
```
func TestCreateNewLibrary(t *testing.T) {
  stub := mock.NewMockStub("org3MSP", new(cc.CCDemo))

  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*

./npmInstall.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

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

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 idenficador 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

```
string      # texto livre
number      # numero flutuante
datetime    # data e hora
boolean     # true / false

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

->[asset]   # referencia a outro asset
[]->[asset] # array de referencias a assets
```

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.

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

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",                      // Idenficação 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,
}
```

ESCOLA SUPERIOR DE REDES

Tarefa

PRÓXIMO MÓDULO:
5
Hyperledger Fabric
(parte 4)

GoLedger