

Feuille de route pour créer un smart contract

Laurent Garnier

September 5, 2019

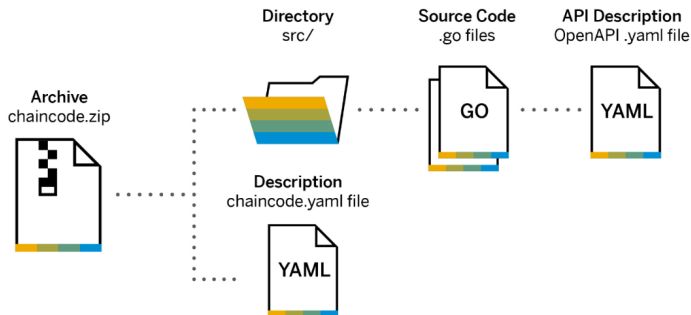
Outline

Structure générale du dossier

Dans tout dossier chaincode il doit y avoir :

- ▶ un fichier nommé `chaincode.yaml`
- ▶ un dossier `src`
- ▶ dans le dossier `src` il doit y avoir :
 1. Le code source `nom_du_fichier.go`
 2. L'API de description `openapi.yaml`

Voir la figure ci-dessous :



Le fichier `chaincode.yaml`

Dans ce fichier nous fournirons les méta-données de notre chaincode.

Les balises `Id` et `Version` sont importantes ici.

Chaque fois qu'un chaincode est appelé, soit depuis une API REST soit depuis un autre chaincode, l'identifiant (ID) du chaincode doit être connu.

Cet ID doit aussi être fourni lorsque le chaincode est déployé.

Nous recommandons de spécifier tous les IDs de chaincode dans le format DNS inverse, comme pour les classes Java.

Par exemple, l'Id de notre chaincode est

`blockchain-example-chaincode_test`.

Cette syntaxe pour les IDs de chaincode : caractères alphanumérique et les tirets - et `_`.

De la même manière, chaque chaincode se voit attribué un numéro de version.

Puisqu'un chaincode est déployé "pour toujours" sur la blockchain et ne peut être effacé, pour remplacer un chaincode on ré-utilise son numéro de version.

Code du fichier chaincode.yaml

```
# THIS SAMPLE CODE MAY BE USED SOLELY AS PART OF A TEST  
# BLOCKCHAIN SERVICE (THE "SERVICE") AND IN ACCORDANCE  
# WITH THE TERMS OF THE AGREEMENT FOR THE SERVICE.  
# THIS SAMPLE CODE PROVIDED "AS IS", WITHOUT ANY WARRANTY,  
# ESCROW, TRAINING, MAINTENANCE, OR SERVICE OBLIGATIONS  
# WHATSOEVER ON THE PART OF ALMERYYS/BE|YS.
```

```
Id:          blockchain-example-chaincode_test  
Version:    1
```

Création du fichier de description API

ALMERYYS/BE|YS fournit une porte d'entrée qui expose toutes les fonctions chaincode comme des APIs REST normales.

Vous pouvez accomplir cela en fournissant une description OpenAPI de l'API REST qui est liée aux fonctions du chaincode. Vous faites cela avec le document YAML qui est écrit avec le chaincode en golang.

Le document YAML décrit exactement comment chaque fonction peut accéder via un appel REST, quels paramètres sont disponibles et comment les paramètres doivent être transmis au chaincode.

Dans le dossier src créer un fichier `hello_world.yaml` :

Code du fichier hello_world.yaml

```
swagger: "2.0"
info:
  description: |
    The Hello World! chain code shows the first steps
    in developing a chaincode that can read/write
    strings onto the blockchain and can expose these
    functions as REST API.
    THIS SAMPLE CODE MAY BE USED SOLELY AS PART OF
    THE TEST AND EVALUATION OF THE ALMERYYS/BE|YS
    BLOCKCHAIN SERVICE (THE "SERVICE") AND IN
    ACCORDANCE WITH THE AGREEMENT FOR THE SERVICE.
    THIS SAMPLE CODE PROVIDED "AS IS", WITHOUT ANY
    WARRANTY, ESCROW, TRAINING, MAINTENANCE, OR
    SERVICE OBLIGATIONS WHATSOEVER ON THE PART OF
    ALMERYYS/BE|YS.
  version: "1.0"
  title: "Hello World!"
```

Création du fichier du code source (en Golang)

Lors du développement d'un chaincode, l'étape suivante est d'écrire le programme GO lui-même (ce qui est notre chaincode).

Dans le programme, il peut y avoir un nombre quelconque de fonctions, chaque fonction ayant un nombre quelconque de paramètres "non nommés".

L'appelant de toute fonction doit connaître le nom de la fonction et de la séquence exacte de paramètres.

Dans les configurations Hyperledger Fabric standards, l'accès au chaincode se fait uniquement via un SDK ce qui requiert un accès du chaincode au HTTPS/gRPC.

Code du fichier hello_world.go :

```
// DISCLAIMER:  
// THIS SAMPLE CODE MAY BE USED SOLELY AS PART OF THE TEST  
// AND EVALUATION OF THE ALMERYYS/BE|YS BLOCKCHAIN SERVICE  
// (THE "SERVICE") AND IN ACCORDANCE WITH THE TERMS OF THE  
// AGREEMENT FOR THE SERVICE. THIS SAMPLE CODE PROVIDED  
// "AS IS", WITHOUT ANY WARRANTY, ESCROW, TRAINING,  
// MAINTENANCE, OR SERVICE OBLIGATIONS WHATSOEVER ON THE  
// PART OF ALMERYYS/BE|YS.
```

Comprendre le fichier de description API

Le fichier de description API `hello_world.yaml` est utilisé pour décrire l'interface HTTP exacte pour le chaincode.

C'est important de comprendre que le fichier `.yaml` est ensuite utilisé dans deux contextes différents :

- ▶ Pour générer la page de test API depuis laquelle les APIs de chaincode peuvent être testées directement
- ▶ La passerelle d'API utilise des aspects spécifiques du fichier YAML pour décider de la méthode d'extraction des paramètres de la requête HTTP entrante. Ensuite, ils associent ces éléments à une fonction du chaincode à appeler.

Comprendre la correspondance verbes HTTP et appels de chaincode

Un ensemble riche de verbes HTTP est utilisé de manière spécifique pour que les appels REST imitent les opérations CRUD (Create Read Update Delete) typiques des bases de données.

Du côté Hyperledger Fabric, les fonctions chaincode peuvent être appelées comme suit :

- ▶ Par un appel `Invoke` qui écrit une transaction avec des ensembles lecture/écriture dans la blockchain
- ▶ Par un appel `Query` pour un type de fonction en lecture seule

Pour toutes les demandes HTTP entrantes, chaque verbe HTTP spécifique correspond à un appel `Invoke` ou `Query` de Hyperledger Fabric.

Dans cet exemple de chaincode, nous utiliserons des appels `POST` et `GET`.

Tableau des correspondances HTTP/Chaincode

Verbe HTTP	Action type	Correspond à	Effet sur la Blockchain
POST	Créer	Invoke	Ecrit une transaction
GET	Lire	Query	Aucun

Comprendre les chemins de chaincode

La section des chemins de chaincode est utilisée pour définir une définition enrichie de l'API basée sur REST et toutes les fonctionnalités de Swagger peuvent être utilisées pour décrire l'API. Deux cas spéciaux s'appliquent :

- ▶ Pour chaque chemin, vous devez préciser l'`operationId`. C'est le nom de direct de la fonction chaincode qui doit être appelée pour ce chemin.
- ▶ Pour les paramètres, les cinq emplacements de paramètres sont pris en charge. Les paramètres peuvent être `path`, `query`, `header`, `form`, ou `body`. Pour les types de paramètres, vous pouvez utiliser uniquement des paramètres simples qui peuvent être mis en correspondance avec des type de chaîne de caractères en entrée de chaincode. Les types acceptés sont : `string`, `number`, `integer`, `boolean`, et `file`.

Définir le chemin du chaincode de POST

Pour définir le chemin d'accès au chaincode (utilisé pour appeler le chaincode), ouvrir le fichier `hello_world.yaml` avec un éditeur de texte et copiez-collez les lignes suivantes :

Code du fichier hello_world.yaml

```
consumes:
  - application/x-www-form-urlencoded

paths:

  /{id}:

    post:
      operationId: write
      summary: Write a text (once) by ID
      parameters:
        - name: id
          in: path
          required: true
          type: string
        - name: text
          in: formData
          required: true
          type: string
      responses:
        200:
          description: Text Written
        500:
          description: Failed
```

Explications

Ce chemin de publication (POST) inclut la fonction `write` (`operationID: write`), deux paramètres pouvant être écrits (`id` et `text`) et deux codes de réponse (200 pour la réussite de la publication et 500 pour l'échec).

Notez que cet exemple inclut également la section `consumes`.

Ceci définit les types de contenu par défaut qui seront acceptés pour tous les appels d'API, s'ils ne sont pas définis spécifiquement.

Par défaut, il doit être défini sur

`application/x-www-form-urlencoded` pour signaler que les demandes HTTP entrantes auront des paramètres au format `nom / valeur`.

Définir le chemin GET

Pour définir le chemin de chaincode GET (utilisé pour interroger le chaincode), copiez et collez les lignes suivantes sous le code précédent :

Code du fichier hello_world.yaml

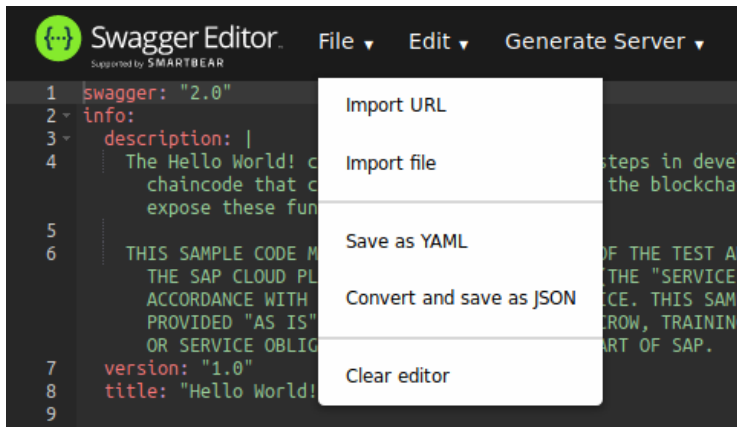
```
get:
  operationId: read
  summary: Read text by ID
  parameters:
    - name: id
      in: path
      required: true
      type: string
  produces:
    - text/plain
  responses:
    200:
      description: OK
    500:
      description: Failed
```

Explications

Ce chemin pour GET comprend les fonctions de lecture (`operationID: read`), un paramètre à lire (`id`) et deux codes de réponse (200 pour une lecture réussie et 500 pour un échec de lecture).

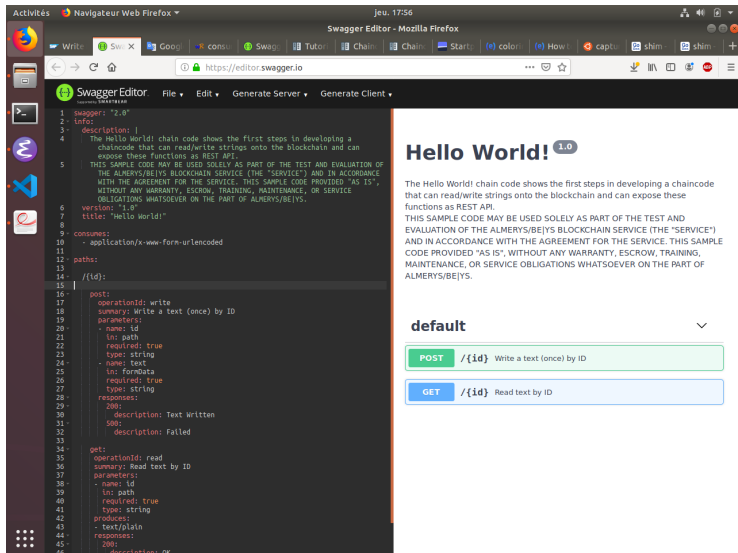
Valider le `hello_world.yaml` avec Swagger.io

Ouvrir un navigateur web et naviguer jusqu'à l'éditeur Swagger.io
Cliquer sur File > Clear Editor



Puis copier le code du fichier `hello_world.yaml` à l'intérieur

Swagger complet



Swagger Editor - Mozilla Firefox

https://editor.swagger.io

Swagger Editor

File Edit Generate Server Generate Client

```
1 swagger: "2.0"
2 info:
3   description: |
4     The Hello World! chain code shows the first steps in developing a
5     chaincode that can read/write strings onto the blockchain and can
6     expose these functions as REST API.
7     THIS SAMPLE CODE MAY BE USED SOLELY AS PART OF THE TEST AND EVALUATION OF
8     THE ALMERY/BEIYS BLOCKCHAIN SERVICE (THE "SERVICE") AND IN ACCORDANCE
9     WITH THE AGREEMENT FOR THE SERVICE, THIS SAMPLE CODE PROVIDED "AS IS",
10    WITHOUT ANY WARRANTY, ESCROW, TRAINING, MAINTENANCE, OR SERVICE
11    OBLIGATIONS WHATSOEVER ON THE PART OF ALMERY/BEIYS.
12  version: "1.0"
13  title: "Hello World!"
14  consumes:
15    - application/x-www-form-urlencoded
16  paths:
17    /{id}:
18      post:
19        operationId: write
20        summary: Write a text (once) by ID
21        parameters:
22          - name: id
23            in: path
24            required: true
25            type: string
26          - name: text
27            in: formData
28            required: true
29            type: string
30        responses:
31          200:
32            description: Text Written
33          500:
34            description: Failed
35      get:
36        operationId: read
37        summary: Read text by ID
38        parameters:
39          - name: id
40            in: path
41            required: true
42            type: string
43        produces:
44          - text/plain
45        responses:
46          200:
47            description: OK
```

Hello World! ^{1.0}

The Hello World! chain code shows the first steps in developing a chaincode that can read/write strings onto the blockchain and can expose these functions as REST API.

THIS SAMPLE CODE MAY BE USED SOLELY AS PART OF THE TEST AND EVALUATION OF THE ALMERY/BEIYS BLOCKCHAIN SERVICE (THE "SERVICE") AND IN ACCORDANCE WITH THE AGREEMENT FOR THE SERVICE, THIS SAMPLE CODE PROVIDED "AS IS", WITHOUT ANY WARRANTY, ESCROW, TRAINING, MAINTENANCE, OR SERVICE OBLIGATIONS WHATSOEVER ON THE PART OF ALMERY/BEIYS.

default

POST /{id} Write a text (once) by ID

GET /{id} Read text by ID