

Programmation d'une blockchain à l'aide de la plateforme HyperLedger

Auteur : Amine Haoues Boussoualim



Encadrants :

Claude Duvallet, Cyrille Bertelle

Année : 2019/2020

Introduction	3
État de l'art	5
La blockchain	5
Hyperledger Fabric	6
Le cas de vente de voiture de l'UN/CEFACT	8
Travaux réalisés	9
Les outils utilisés	9
Le réseau de test	10
Le fonctionnement des smart contracts	11
Un smart contract pour gérer une vente de voiture	14
Une application pour utiliser le smart contract	16
Les résultats	18
Conclusion	21
Bibliographie	22

Introduction

Depuis son apparition en 2008, la blockchain ne cesse de faire parler. Elle connaît un véritable engouement depuis 2017, notamment suite à l'explosion des crypto monnaies, en particulier le bitcoin. Son champ d'application est immense. On pense le plus souvent justement aux crypto monnaies comme le bitcoin, mais il y a en réalité beaucoup d'autres domaines qui peuvent en bénéficier : banques, assurances, industrie musicale, immobilier.

À la vue de cet engouement, beaucoup de plateformes sont apparues afin de développer et de gérer des blockchains. Au cours de ce projet, nous allons nous concentrer sur la plateforme Hyperledger Fabric.

L'objectif de ce projet est d'expérimenter la création d'une blockchain à l'aide de la plateforme Hyperledger Fabric, et de se concentrer plus précisément sur l'écriture de contrat intelligent, afin de gérer par exemple un achat de voiture.

État de l'art

La blockchain

La blockchain est une liste d'enregistrements appelés blocs qui sont liés entre eux de manière cryptographique. Chaque bloc contient son hash, le hash du bloc précédent et une ou plusieurs transactions. De cette manière, une blockchain est immuable et infalsifiable. C'est donc une base de données distribuée sans organe de contrôle qui permet l'enregistrement de transactions entre deux parties. La blockchain est généralement gérée par un réseau pair à pair qui adhère à un type de protocole. Quand une transaction est enregistrée, elle l'est par tous les pairs du réseau. Ainsi, si on veut modifier une information sur la blockchain, il faut la modifier sur une majorité des pairs du réseau.

Il existe trois types de blockchain :

- La blockchain publique : les utilisateurs sont contrôlés par un réseau pair à pair. Tout le monde participe au processus de création de consensus. Chaque noeud du réseau va valider les choix initiés par les développeurs (Bitcoin ou Ethereum par exemple).
- La blockchain privée : l'accès à la blockchain est géré par un administrateur qui peut en modifier le protocole à tout moment.
- La blockchain de consortium : seulement certains noeuds du réseau vont gérer le consensus. On peut la qualifier d'hybride puisque certains noeuds peuvent être privés et d'autre publics.

La blockchain possède donc trois propriétés :

- La transparence : chaque participant à la blockchain peut accéder à l'historique des transactions depuis sa création.
- La sécurité : grâce au hachage, il est presque impossible de corrompre les données.
- La décentralisation : elle fonctionne grâce à un réseau pair à pair.

Hyperledger Fabric

La fondation Hyperledger est l'un des principaux acteurs de la blockchain dans le monde. C'est un projet open source lancé par la Linux Foundation afin de faire avancer la collaboration intersectorielle sur les technologies blockchain. Hyperledger offre à ses membres une structure technologique pour développer un projet reposant sur la blockchain. Pour cela, elle met à disposition différents frameworks et outils présentés sur la figure ci-dessous.

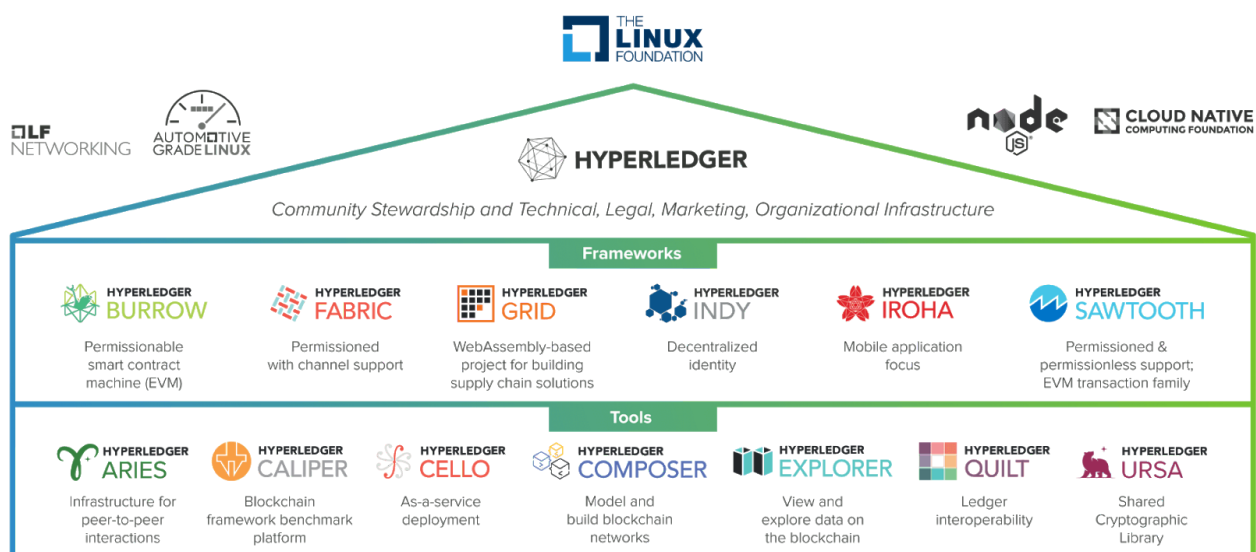


Figure 1 : Les différents composants d'hyperledger

Nous allons plus précisément parler d'Hyperledger Fabric, car c'est le sujet de ce TPE.

Hyperledger Fabric est censé être une fondation pour le développement d'applications ou de solutions avec une architecture modulaire. Il permet de gérer de manière modulaire et versatile les différents composants de la blockchain telle que la privatisation et le consensus afin de faciliter l'accès aux entreprises. Les entreprises peuvent alors très facilement gérer les différentes permissions au sein d'une blockchain. Hyperledger Fabric permet la création d'un "réseau de réseaux", les entreprises peuvent alors gérer qui au sein de leur réseau accèdent à quels types de données.

Le modèle d'Hyperledger Fabric est constitué de plusieurs éléments que je vais décrire :

- Les assets : il s'agit de n'importe quelle ressource ayant une valeur monétaire, que ce soit quelque chose de physique ou bien d'une propriété intellectuelle. Ils sont représentés comme une collection de paires de clefs-valeurs dont le changement d'état est enregistré comme une transaction sur un canal. Ils sont modifiables grâce à des "chaincodes" (smart contract).
- Les chaincodes : ils permettent de définir un asset sous la forme de code (dans mon cas javascript) et un ensemble de règles qui vont permettre de gérer les transactions et modifications d'un asset.
- Le ledger : il s'agit de l'enregistrement immuable de toutes les modifications d'états dans la "fabric". Une modification d'état sera le résultat d'une transaction "chaincode" envoyée par l'un des pairs. Chaque transaction va donner lieu à une paire de clef-valeur qui sera envoyée au ledger. Le ledger est composé d'une blockchain et chaque canal en possède un. Chaque paire conserve une copie du ledger.

En résumé, on a donc un ledger immuable qui est propre à chaque canal, et un chaincode qui va modifier les états d'un asset et tout ceci fait partie d'un réseau (cf. Figure 2). Un ledger peut être partagé à tous les utilisateurs d'un canal ou bien restreint à un certain nombre de participants.

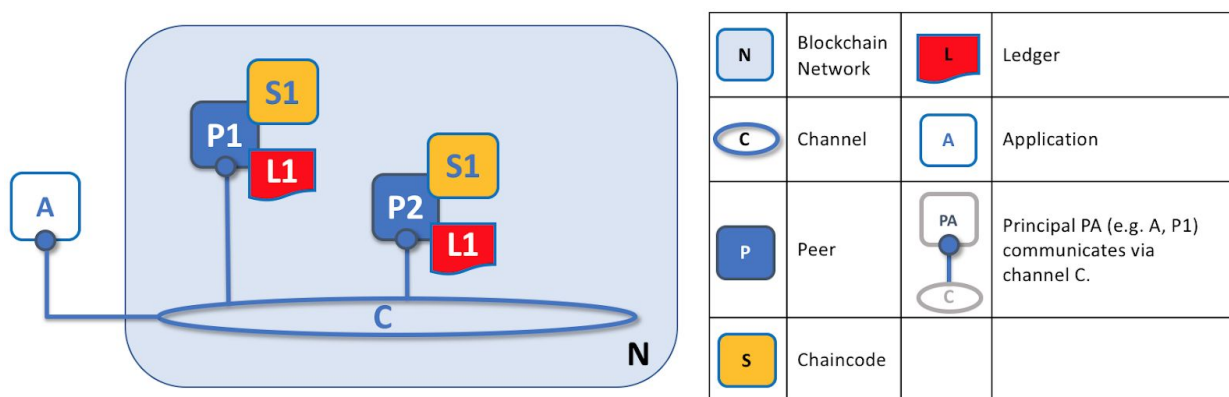


Figure 2 : Illustration d'un réseau avec un canal contenant deux pairs

Le cas de vente de voiture de l'UN/CEFACT

L'UN/CEFACT est un organisme des nations unies ayant pour but de faciliter les échanges commerciaux électroniques. L'objectif de ce TPE se base sur un scénario développé par cet organisme, la gestion d'une vente de voiture entre particuliers. L'idée serait d'avoir un service web qui s'adresse aux acheteurs et d'utiliser la technologie des smart contracts afin d'avoir une transaction sécurisée via une blockchain.

Le scénario serait le suivant :

Étape 1 : Bob écrit un smart contract pour déterminer les paramètres de la vente notamment le prix.

Étape 2 : la voiture et ses clés sont laissés dans un garage, fermé par un verrou électronique.

Étape 3 : Alice, qui veut acheter la voiture, signe le smart contrat et transfère l'argent nécessaire (prix défini dans le smart contract).

Étape 4 : chaque noeud du réseau regarde le smart contract sur son registre pour vérifier que Bob est le propriétaire de la voiture et qu'Alice a assez d'argent pour acheter la voiture.

Étape 5 : quand on atteint un consensus pour les deux conditions précédentes, Alice reçoit automatiquement le code du verrou électronique et peut aller chercher la voiture dans le garage. Alice devient le nouveau propriétaire de la voiture, le prix de la voiture est débité du compte d'Alice et est crédité sur le compte de Bob.

Travaux réalisés

Les outils utilisés

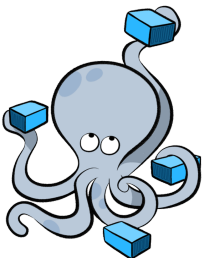
Avant de présenter les travaux réalisés, je vais vous présenter différents outils que j'ai pu utiliser :



J'ai tout d'abord installé la dernière version d'Ubuntu (<https://ubuntu.com/download/desktop>) sur ma machine. Le projet aurait pu être réalisé sur windows, mais toute la documentation d'hyperledger est basée sur ubuntu. C'était donc la solution la plus logique.



J'ai utilisé l'éditeur de code "visual studio code" (<https://code.visualstudio.com/>) qui est gratuit disponible sur windows, linux et mac os. Je l'ai choisi principalement pour sa simplicité d'utilisation et son ergonomie.



J'ai également installé docker compose (<https://docs.docker.com/compose/>), c'est un outil qui permet de lancer une application docker qui utilise plusieurs conteneurs. Cet outil permettra d'émuler un réseau avec différents pairs.



Le sdk d'hyperledger fabric est disponible en plusieurs langages : Go, Java, Javascript, Python. Maîtrisant le mieux Javascript, j'ai décidé d'écrire mon code dans ce langage. Il a donc fallu installer node.js (<https://nodejs.org/en/download/>).



Enfin, j'ai bien évidemment utilisé le SDK Hyperledger Fabric (<https://github.com/hyperledger/fabric-sdk-node>) en version node.js comme dit plus haut.

Le réseau de test

Bien que le sujet initial de ce TPE était la programmation d'une blockchain avec Hyperledger Fabric. Ce sujet avait déjà été confié à des étudiants l'an dernier. On m'a demandé de repartir de ces travaux et d'implémenter principalement la partie des smart contract. Or, le réseau créé l'an dernier utilisait la plateforme Hyperledger Composer qui est aujourd'hui dépréciée et n'est plus tenue à jour.

J'ai donc décidé de reprendre le réseau de tests fourni par Hyperledger Fabric pour développer et installer un smart contract. Ce réseau convient parfaitement au scénario d'une vente de voiture entre deux personnes, car il est composé de deux pairs et d'un canal.

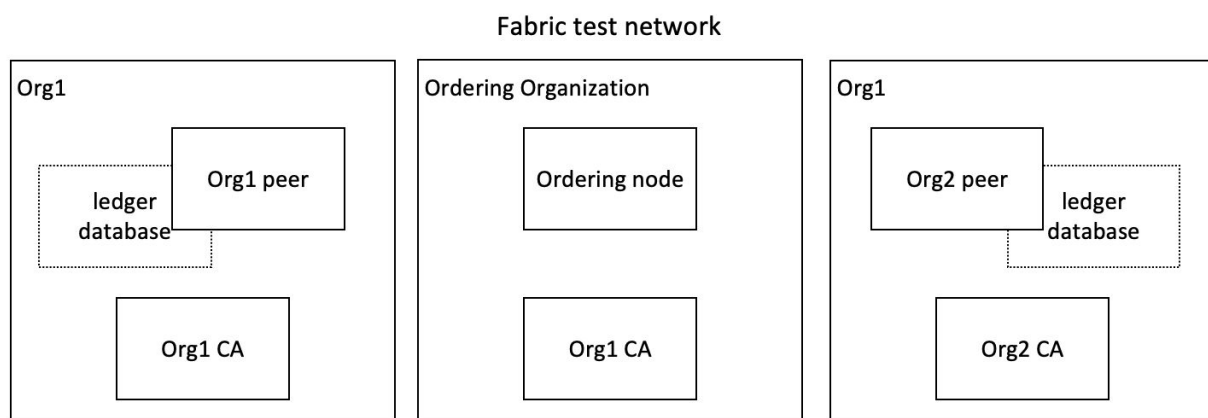


Figure 3 : Schéma des différentes organisations du Test Network d'hyperledger fabric

Chaque noeud qui interagit avec un réseau Fabric doit appartenir à une organisation qui doit elle-même être membre d'un réseau. Le groupe d'organisations d'un

même réseau est appelé consortium. Dans notre cas, nous avons deux organisations Org1 et Org2. Les pairs stockent le ledger et valident les transactions avant de les envoyer au ledger. Ils exécutent aussi les smart contrats qui gèrent les assets sur le ledger. Chaque pair doit appartenir à une organisation du consortium. Nous avons ici deux pairs qui appartiennent à deux organisations différentes.

Nous avons aussi une organisation appelée "Ordering Organization". Il y en a une dans tous les réseaux Fabric. Elles permettent aux pairs de se concentrer sur la validation de transactions et l'envoi des blocs de transactions au ledger. Pendant que cette organisation se charge de vérifier qu'il y a un consensus entre toutes les organisations puis de renvoyer aux pairs les blocs de transactions.

Le fonctionnement des smart contracts

Avant de voir le smart contract que j'ai développé, nous allons d'abord nous pencher sur la gestion des smart contracts par Hyperledger Fabric.

Un smart contract, appelé chaincode par Hyperledger Fabric, est un programme qui peut être écrit en Go, Node.js ou Java. Il est exécuté sur un container Docker indépendamment des autres pairs. Un chaincode gère l'état du ledger à travers des transactions envoyées par des applications.

Un chaincode permet de gérer un ensemble de règles que chaque membre du réseau a approuvé. Nous allons donc voir le cycle de vie d'un chaincode du point de vue d'un réseau blockchain.

Le cycle de vie d'un chaincode permet à plusieurs organisations de s'accorder sur le fonctionnement du chaincode. De nombreux cas sont possibles, mais nous allons nous concentrer sur le déploiement d'un chaincode sur un réseau.

Le déploiement d'un chaincode s'effectue en plusieurs étapes :

Etape 1 : emballer le chaincode. Le chaincode doit être compressé dans un fichier "tar.gz". Cela peut être fait en utilisant la commande "*peer lifecycle chaincode package*". Cette étape peut être réalisée par un ou plusieurs pairs du réseau.

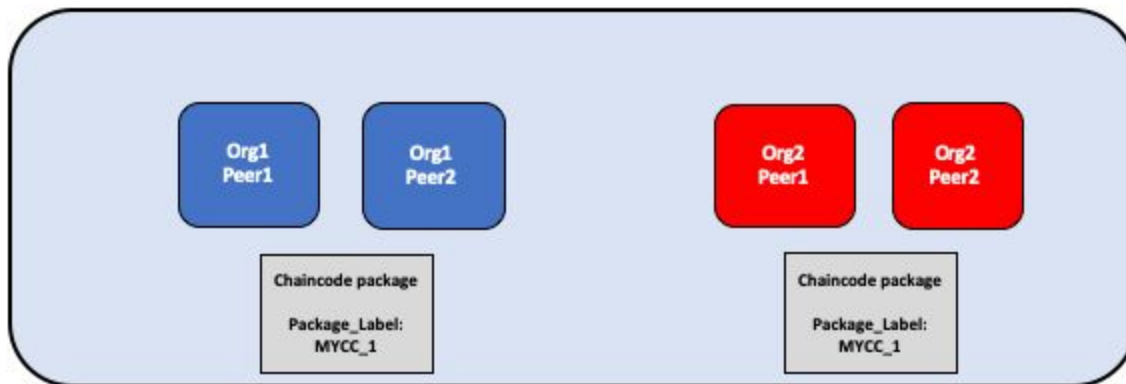


Figure 4 : Exemple avec deux organisations ayant chacun deux pairs et des chaincodes emballés séparément

Etape 2 : Installer le chaincode sur les pairs. Le chaincode doit être installé sur chaque pair qui doit exécuter des transactions. L'installation doit être faite par le pair "administrateur" de l'organisation. Cela s'effectue grâce à la commande "*peer lifecycle chaincode install*".

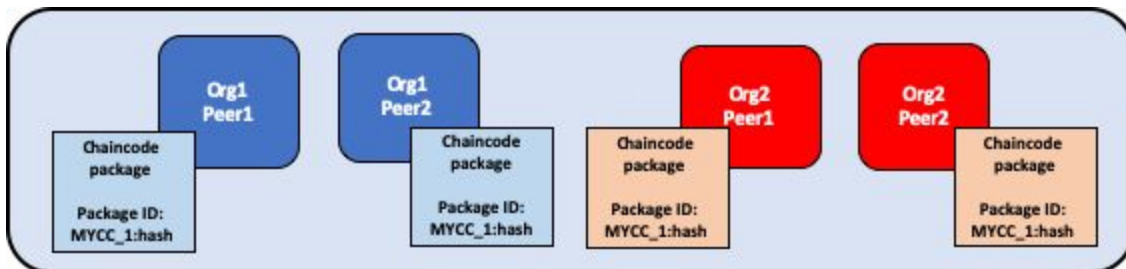


Figure 5 : Un pair administrateur de chaque organisation installe le chaincode pour tous les pairs de cette organisation

Etape 3 : Approuver la définition d'un chaincode pour son organisation. Le chaincode est gouverné par une définition de chaincode. Quand un membre d'un canal approuve la définition d'un chaincode, il l'approuve pour son organisation. Pour qu'un chaincode soit utilisable, plusieurs organisations doivent l'approuver (par défaut la majorité).

La définition d'un chaincode contient plusieurs paramètres qui doivent être les mêmes pour chaque organisation : le nom, la version, la séquence, la politique d'approbation. Ce dernier paramètre est très important puisqu'il permet de définir combien d'organisations doivent exécuter et valider une transaction.

Voici la commande permettant d'approuver un chaincode : *"peer lifecycle chaincode approveformyorg"*.

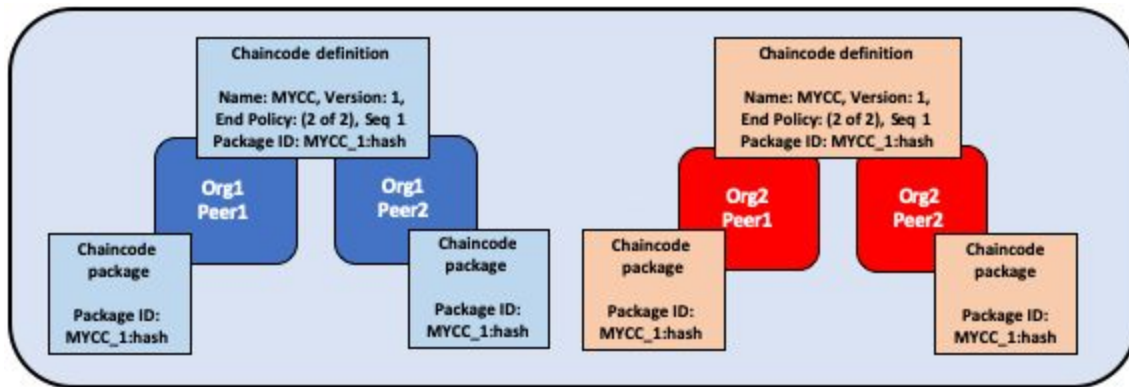


Figure 6 : Un pair administrateur a approuvé la définition du chaincode pour les deux organisation.

Etape 4 : Envoyer la définition du chaincode au canal. Quand un assez grand nombre de membres du canal a approuvé la définition du chaincode, n'importe quelle organisation peut envoyer le chaincode au canal via la commande : *"peer lifecycle chaincode commit"*. Quand le chaincode est envoyé au canal, un conteneur s'exécute sur chaque pair où le chaincode est installé, permettant aux membres du canal d'utiliser le chaincode.

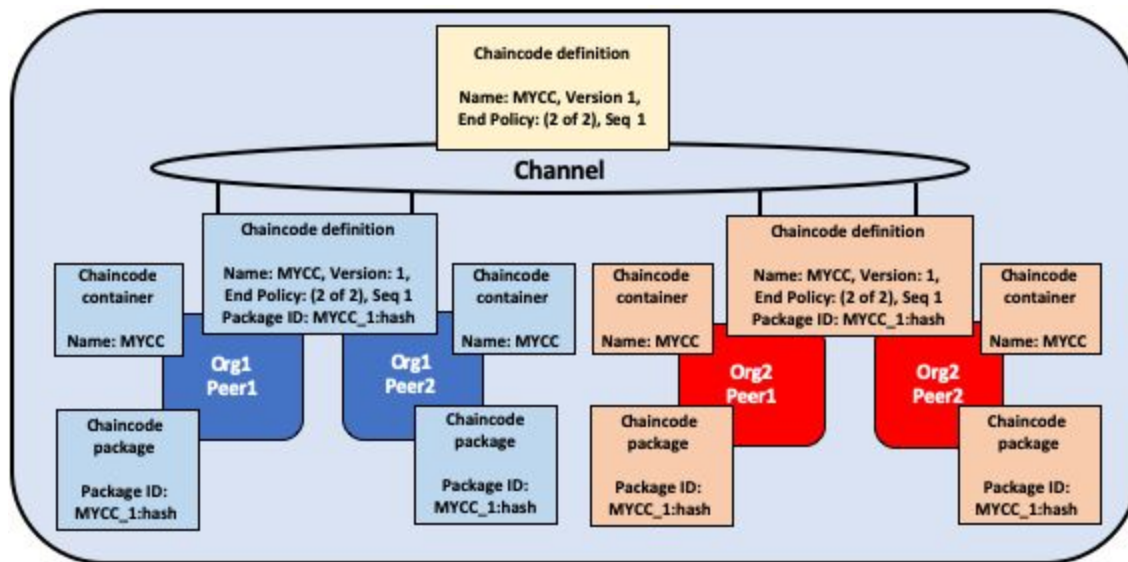


Figure 7 : La définition du chaincode est installée sur le canal, le chaincode est exécuté sur chaque pair qui peut maintenant l'utiliser pour effectuer des transactions sur le ledger.

Un smart contract pour gérer une vente de voiture

Nous allons maintenant voir de quoi est composé le code du smart contract qui permet de gérer une vente de voiture.

On a tout d'abord une classe **car.js** qui permet de gérer un objet qui définit une voiture.

On trouve ensuite un fichier nommé **carcontract.js**. Il s'agit du fichier qui contient le chaincode. Passons en revue les différents éléments de ce fichier.

```
const {Contract, Context} = require('fabric-contract-api')
```

La première ligne permet d'importer deux classes clés d'Hyperledger Fabric : Contrat et Context.

```
class CarContract extends Contract{
```

On définit ensuite la classe du smart contract. Elle est basée sur la classe **Contract** et contient les principales méthodes qui implémentent les transactions nécessaires.

```
async sale(ctx, seller, carNumber, carBrand, carModel, price){
    // Create an instance of the car
    let car = Car.createInstance(seller, carNumber, carBrand, carModel,
price)

    // Set the car into the forsale state
    car.setForSale();

    car.setOwner(seller);

    // Add the car to the list of cars into the ledger state
    await ctx.carList.addCar(car);

    return car;
}
```

Cette méthode définit la transaction "sale", c'est-à-dire une mise en vente du véhicule, elle modifie l'état du véhicule et l'ajoute via la méthode "addCar" au ledger.

```
async buy(ctx, seller, carNumber, currentOwner, newOwner, price){
```

La méthode "buy" permet de simuler un achat de véhicule. On récupère le véhicule sur le ledger et on vérifie différents paramètres avant de modifier l'état du véhicule pour dire qu'il est vendu.

Nous avons donc vu de quoi est composé un smart contract et comment il s'utilise sur un réseau hyperledger fabric. Nous allons bientôt voir les résultats que j'ai pu avoir sur ma machine en reprenant les 4 étapes que j'ai expliquées et en les appliquant à ce smart contrat.

Une application pour utiliser le smart contract

Avant de voir l'exécution de ce chaincode il faut également parler de l'application qui va l'exécuter, une application est un bout de code qui va gérer l'envoi des transactions au ledger. Voyons les différents éléments qui la composent.

A noter qu'il y a ici deux applications différentes, une qui va agir en tant qu'Alice et mettre en vente une voiture, et une qui va agir en tant que Bob et créer la transaction d'achat de voiture. Nous allons aborder seulement celle d'Alice le fonctionnement étant similaire. Seule, la transaction change.

On retrouve tout d'abord un fichier JavaScript nommé **addToWallet** commun aux deux applications. Il permet de créer une identité nommée Alice dans un cas ou Bob dans l'autre, appartenant respectivement à l'organisation 1 et 2. Un pair doit obligatoirement avoir une identité dans le wallet afin d'accéder à ce qu'on appelle la gateway. La gateway fait le lien entre une application client et un pair du réseau.

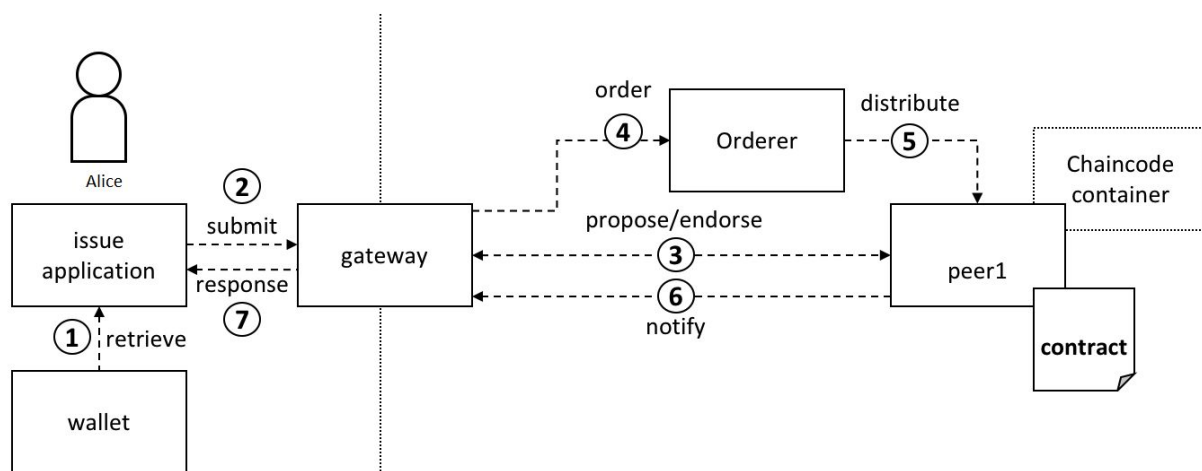


Figure 8 : Illustration d'une transaction faite par une application

On voit sur ce schéma le fonctionnement de la gateway, elle permet à une application de se concentrer seulement sur la génération d'une transaction et son envoi aux pairs.

On trouve ensuite pour Alice un fichier **sell.js**. Passons en revue le code.

```
const fs = require('fs');
const yaml = require('js-yaml');
const {Wallets, Gateway} = require('fabric-network');
const Car = require('../contract/lib/car.js');
```

On commence par importer les différents composants importants.

```
try{
    const userName = 'Alice';

    let connectionProfile =
yaml.safeLoad(fs.readFileSync('../gateway/connection-org1.yaml', 'utf8'));

    let connectionOptions = {
        identity: userName,
        wallet: wallet,
        discovery: {enabled: true, asLocalhost: true}
    };

    //Connect to the gateway using the specified parameters
    console.log('Connect to the fabric gateway');
    await gateway.connect(connectionProfile, connectionOptions);
```

On s'identifie ensuite à la gateway en tant qu'Alice grâce à l'identité qu'on a ajouté au wallet précédemment.

```
//Getting the network
    console.log('Using network channel : mychannel');
    const network = await gateway.getNetwork('mychannel');

    //Getting the smart contract
    console.log('Use org.carmarket.car smart contract');
    const contract = await network.getContract('carcontract');
```

On se connecte au network et on récupère le smart contract qu'on veut utiliser.

```
//Submit transaction for car sale
    console.log('Submit car sale transaction');
    const sellResponse = await contract.submitTransaction('sale',
'Alice', '00001', 'Renault', 'Clio 1', '500');
```

La transaction est alors envoyée grâce à la méthode **submitTransaction**. Ici, c'est la transaction "sale" du chaincode qui est utilisée.

Nous avons donc expliqué comment une application peut utiliser un smart contract pour gérer des transactions sur la blockchain.

Les résultats

Nous pouvons maintenant voir ce que ça donne dans la pratique.

On commence par lancer le réseau de test avec un fichier nommé **network-starter.sh**, il permet de lancer le réseau de test fourni par Hyperledger Fabric.

On ouvre ensuite deux terminaux, un pour Alice et l'autre pour Bob, grâce à un fichier permettant de gérer les variables d'environnement. On va pouvoir agir en tant que pair 1 dans un des terminaux et en tant que pair 2 dans l'autre.

On commence par emballer et installer le chaincode (étape 1 et 2) en tant qu'Alice.

```
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Alice$ peer lifecycle chaincode package car.tar.gz --lang node --path ./contract --label car_0
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Alice$ peer lifecycle chaincode install car.tar.gz
2020-06-11 13:13:08.937 CEST [cli.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed remotely: response:<status:200 payload:"\nFcar_0:7778fe966e1cfe11982c5f9d2d1a8eb1a225e4f547df8bac40ca50569fbd076d\022\005car_0" >
2020-06-11 13:13:08.937 CEST [cli.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode code package identifier: car_0:7778fe966e1cfe11982c5f9d2d1a8eb1a225e4f547df8bac40ca50569fbd076d
```

On va ensuite approuver (étape 3) la définition du chaincode avec Alice également :

```
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Alice$ peer lifecycle chaincode approveformyorg --orderer localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channel ID mychannel --name carcontract -v 0 --package-id $PACKAGE_ID --sequence 1 --tls --cafile $ORDERER_CA
2020-06-11 13:16:31.769 CEST [chaincodeCmd] ClientWait -> INFO 001 txid [59a51a1a5644bc2fb33e7e213d74a1df2ea3906ade69797176425e8369cf6ce0] committed with status (VALID) at
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Alice$
```

On effectue ces mêmes étapes avec Bob :

```
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Bob$ peer lifecycle chaincode
package car.tar.gz --lang node --path ./contract --label car_0
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Bob$ peer lifecycle chaincode
install car.tar.gz
2020-06-11 13:17:11.192 CEST [cli.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed re
motely: response:<status:200 payload:"\nFcar_0:7778fe966e1cfe11982c5f9d2d1a8eb1a225e4f547df8bac40ca50
569fbd076d\022\005car_0" >
2020-06-11 13:17:11.192 CEST [cli.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode co
de package identifier: car_0:7778fe966e1cfe11982c5f9d2d1a8eb1a225e4f547df8bac40ca50569fbd076d
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Bob$ export PACKAGE_ID=car_0:7
778fe966e1cfe11982c5f9d2d1a8eb1a225e4f547df8bac40ca50569fbd076d
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Bob$ peer lifecycle chaincode
approveformyorg --orderer localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channelID
mychannel --name carcontract -v 0 --package-id $PACKAGE_ID --sequence 1 --tls --cafile $ORDERER_CA
2020-06-11 13:19:14.824 CEST [chaincodeCmd] ClientWait -> INFO 001 txid [4f05885c3e659d01e40cd9d716b0
df74e970648a3a2e8fa081e266ccfc009a61] committed with status (VALID) at
```

On peut maintenant envoyer la définition du chaincode avec Alice seulement car toutes les organisations l'ont approuvé :

```
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Alice$ peer lifecycle chaincod
e commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --peerAddresses localhost
:7051 --tlsRootCertFiles ${PEER0_ORG1_CA} --peerAddresses localhost:9051 --tlsRootCertFiles ${PEER0_O
RG2_CA} --channelID mychannel --name carcontract -v 0 --sequence 1 --tls --cafile $ORDERER_CA --waitF
orEvent
2020-06-11 13:23:58.982 CEST [chaincodeCmd] ClientWait -> INFO 001 txid [b4959e7933b4ad32d825080f38f6
6a8dcf4b840068b87ca4961280e5e423818b] committed with status (VALID) at localhost:9051
2020-06-11 13:23:59.176 CEST [chaincodeCmd] ClientWait -> INFO 002 txid [b4959e7933b4ad32d825080f38f6
6a8dcf4b840068b87ca4961280e5e423818b] committed with status (VALID) at localhost:7051
```

On peut maintenant utiliser l'application créée pour Alice afin d'ajouter la voiture en vente sur le ledger :

```
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Alice/application$ node addToWallet.js
done
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Alice/application$ node sell.js
Connect to the fabric gateway
Using network channel : mychannel
Use org.carmarket.car smart contract
Submit car sale transaction
Process sell transaction response {"class":"org.carmarket.car","key":"\\"Alice\\":\\"00001\\"","currentState":1,"
seller":"Alice","carNumber":"00001","carBrand":"Renault","carModel":"Clio 1","price":"500","owner":"Alice"}
Alice's car : 00001 successfully put on market for 500$
Transaction complete
Disconnecting from Fabric Gateway
Sell program complete.
```

Enfin, on peut utiliser l'application créée pour Bob afin de modifier l'état de la voiture sur la blockchain pour simuler une vente :

```
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Bob/application$ node addToWallet.js
done
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Bob/application$ node buy.js
Connect to the fabric gateway.
Use network channel : mychannel
Use org.carmarket.car smart contract
Submit car buy transaction
Process buy transaction response
Alice car's : 00001 successfully bought by Bob for 500$
Transaction complete.
Disconnect from Fabric gateway.
Buy program complete.
```

En utilisant le fichier **info.js** présent dans les deux applications, on constate que le véhicule a bien changé de propriétaire et passé de l'état 1 (en vente) à l'état 2 (vendu) :

```
amine@amine-H81M-S1:~/Documents/TPEBlockchain/VenteVoiture/organization/Bob/application$ node info.js
Connect to the fabric gateway
Using network channel : mychannel
Use org.carmarket.car smart contract
Submit car info transaction
Process sell transaction response {"class":"org.carmarket.car","key":"\\"Alice\\"":"00001\\","currentState":2,"carBrand":"Renault","carModel":"Clio 1","carNumber":"00001","owner":"Bob","price":"500","seller":"Alice"}
Car info displayed
Transaction complete
Disconnecting from Fabric Gateway
Sell program complete.
```

Nous avons donc vu comment une application peut utiliser un smart contract basique afin de gérer via une blockchain une vente de véhicule. Tous les détails sont approuvés par les deux parties et les fraudes sont donc limitées.

Conclusion

En conclusion, ce projet a permis d'avoir une ébauche de l'utilisation d'Hyper ledger Fabric, en se focalisant principalement sur le fonctionnement des chaînes et de son utilisation par des applications. On peut imaginer à l'avenir un chaincode encore plus avancé permettant une gestion d'une somme d'argent ou autre. On peut également se concentrer sur la création d'un plus grand réseau blockchain avec plus de deux pairs, chose qui m'a posé beaucoup de difficultés durant ce projet. D'un point de vue plus personnel, j'ai énormément appris sur la blockchain et ses applications. C'est un domaine qui m'était quasiment inconnu et qui me paraissait très abstrait avant ce projet.

Lien vers le répertoire du git contenant le code source du projet :

<https://www-apps.univ-lehavre.fr/forge/ba160129/tpeblockchain>

Bibliographie

- [1] https://hyperledger-fabric.readthedocs.io/en/latest/getting_started.html
- [2] <https://blockchainfrance.net/decouvrir-la-blockchain/c-est-quoi-la-blockchain/>
- [3] <https://fr.wikipedia.org/wiki/Blockchain>
- [4] <https://www.hyperledger.org/use/fabric>