Rapport de projet IMA-4

Secure And Verified Public Announcements through Blockchain

AIT MOUHEB Arezki

Sommaire

I. Introduction

Blockchain est une technologie de stockage et de transmission d'informations basée sur la cryptographie. Elle peut être vue comme une sorte de base de données distribuée, sécurisée et partagée. Autrement dit, c'est une structure de données qui contient des informations et qui n'est pas détenue par un serveur centralisé mais par toutes les machines connectées au réseau. Ces informations sont groupées en blocs d'une taille définie et chaque bloc est lié à celui qui le précède par un Hash Cryptographique créant ainsi une chaîne. Le véritable attrait de la Blockchain est de créer un registre de données qui est au même temps commun et extrêmement difficile à altérer.

Le potentiel de la blockchain

Le caractère décentralisé de la blockchain, couplé avec sa sécurité et sa transparence, promet des applications bien plus larges que le domaine monétaire.
On peut classer l'utilisation de la blockchain en trois catégories :

- Les applications pour le transfert d'actifs (utilisation monétaire, mais pas uniquement : titres, votes, actions, obligations…).
- Les applications de la blockchain en tant que registre : elle assure ainsi une meilleure traçabilité des produits et des actifs.
- Les smart contracts : il s'agit de programmes autonomes qui exécutent automatiquement les conditions et termes d'un contrat, sans nécessiter d'intervention humaine une fois démarrés.

II. Description générale du projet

L'Objectif annoncé pour le projet est de réaliser travail de recherche pour comprendre le fonctionnement de la technologie et l'analyse des différentes implémentations existantes. Le but est d'aboutir à un choix d'implémentation pouvant assurer le développement d' un système de diffusion de messages. Ce système devra être fédéré et doit permettre la vérification de l'accès et le contrôle des privilèges. Une fois le système en place développer de simples clients capables d'interagir avec lui (ligne de commande, interface Web ...).

III. Contexte

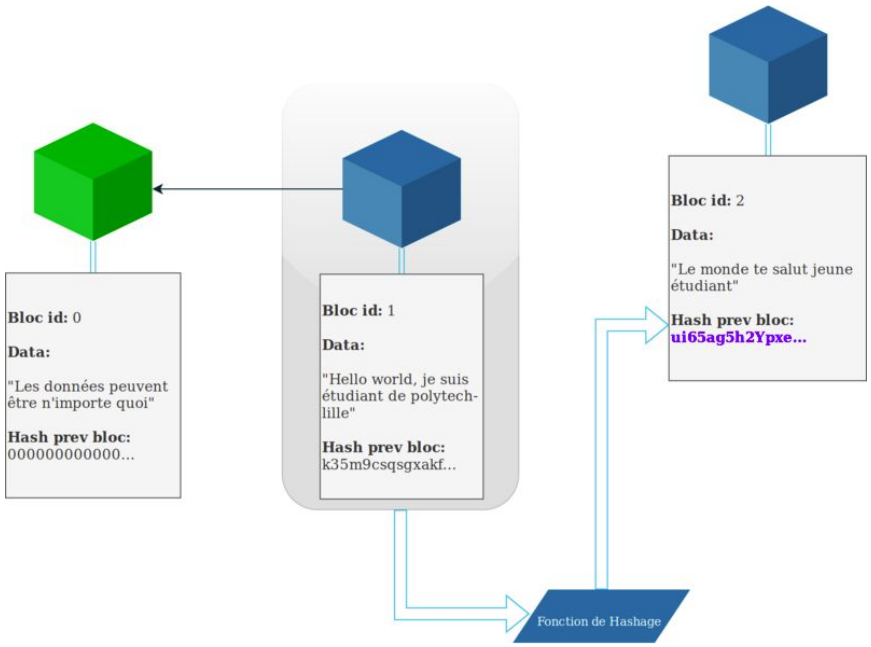

Figure1 :fonctionnement de la blockchain

En informatique, l'usage des fonctions de hachage cryptographiques telles que SHA256 est assez fréquent. Ce sont des fonctions qui, à une donnée de taille arbitraire, associent une image de taille fixe (digest ou hash). La propriété essentielle de ces fonctions est le fait qu'elles sont pratiquement impossibles à inverser. En d'autre termes, à partir d'un hash il s'avère très difficile de retrouver la donnée qui l'a produit. D'une autre part un petit changement dans la donnée d'entrée engendre un hash totalement différent.

Lorsque l'on insère un nouveau bloc dans une blockchain, on le relie au reste de la chaîne en y ajoutant un champ contenant le digest du bloc précédent. D'après ce qu'on a vu dans le paragraphe précédent, si l'on modifie un bloc (n), son digest ne sera plus le même donc le bloc suivant (n+1) n'aura plus un lien valide il faudra de ce fait le recalculer. En recalculant le lien dans le (n+1) nous changeons maintenant son contenu. Du coup, le bloc (n+2) n'aura plus un lien valide ...etc jusqu'au dernier bloc de la chaîne.

Ceci n'est pas suffisant pour garantir une sécurité car dans ce sens le calcul des Hash reste toujours faisable. C'est alors qu'intervient une autre composante importante qu'on appelle "protocole de consensus".

Les protocoles de consensus

Nous n'allons pas détailler ici tous les protocoles existant mais nous allons nous intéresser à deux exemples **La preuve de travail PoW bitcoin** et **Practical Byzantine Fault Tolerance**

1- Proof of work : (PoW)

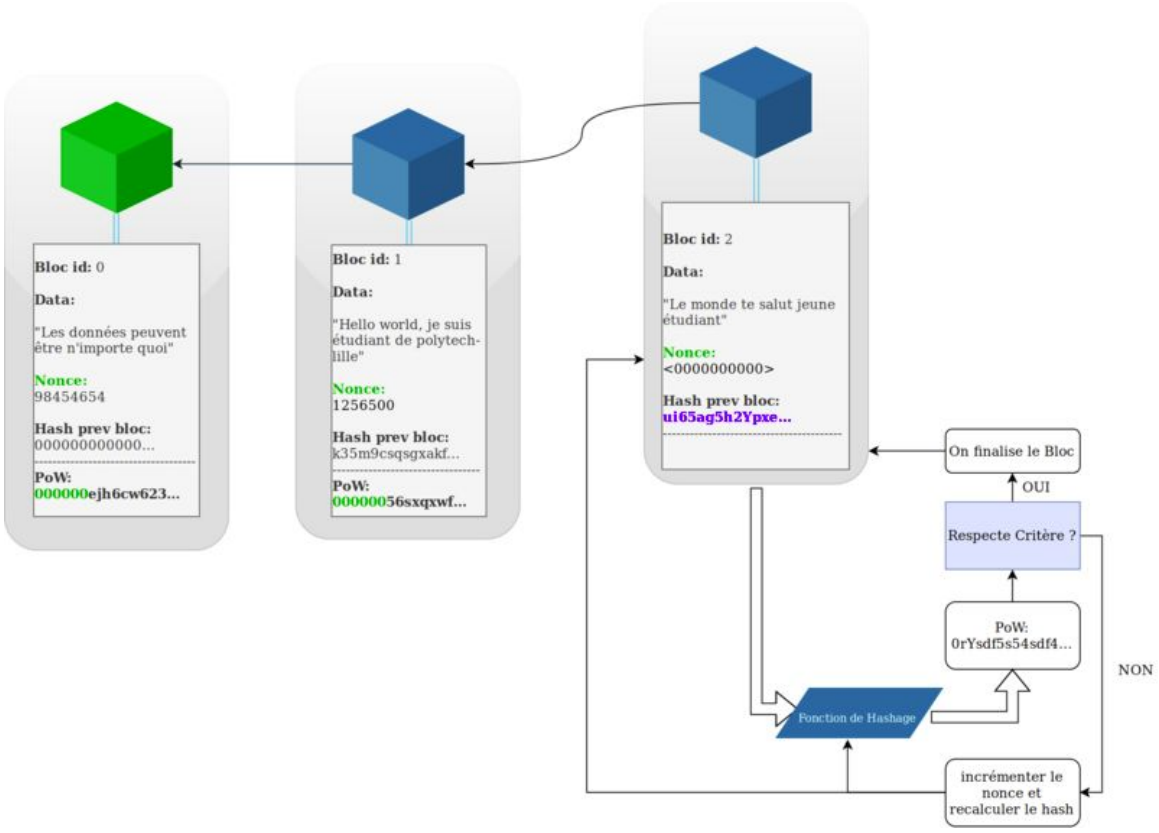

Figure2 :fonctionnement de la preuve de travail

C'est un protocole basé sur l'asymétrie du coût de calcul : le travail doit être difficilement réalisable, mais facilement vérifiable. Une implémentation de ce protocole peut être d'ajouter un champs dans le bloc "nonce" qui serait juste un chiffre dont le rôle est de faire varier le hash du bloc construit précédemment. D'une autre part on spécifie un critère pour le protocole par exemple :"Le hash après le calcul de la preuve de travail doit commencer par six '0' successifs." Comme les fonctions de hashage ne sont pas réversibles, la seule façon d ' y arriver est de calculer le Hash du bloc pour chaque valeur du "nonce" jusqu'à ce que le critère soit satisfait. Cette tâche est très demandeuse en terme de puissance de calcul. Un fois que le critère est satisfait le bloc est validé par le protocol qui calcule le hash du bloc et le compare à la preuve de travail. S'il correspond, le bloc est ajouté à la chaîne. Pour comprendre le niveau de sécurité de cette solution, essayons de voir ce qu'il faut faire pour réussir à la contourner. Supposons qu'un nœud malveillant décide de modifier un bloc (n)

pour quelque raison que ce soit. En faisant cela, il devra refaire la preuve de travail pour le bloc (n) puis recalculer le champ "Prev hash" dans le (n+1) puis sa preuve de travail et ainsi de suite pour le (n+2),(n+3)...etc ce qui s'avère être une tâche vorace en terme de puissance de calcul. Même s'il y arrive, il doit continuer à valider les nouveaux blocs (calculer la preuve de travail) pour maintenir la chaîne altérée qu'il vient de créer car le protocole est fait de telle sorte à considérer la chaîne la plus longue comme étant la chaîne de référence. Ceci donc nous amène à considérer les limites de cette solution:

- Si une entité malveillante possède 51% de la puissance de calcul du réseau elle pourra donc imposer sa version de la chaîne ce qui implique que le réseau doit être suffisamment grand pour que ça ne soit pas pratiquement possible qu'une entité puisse s'accaparer autant de puissance.
- Cet algorithme de consensus est vorace en énergie.

Conclusion: cet algorithme initial de consensus est certes très intéressant d'un point de vue sécurité sous réserve de certaines conditions mais ne correspond pas à tout type d'application.

2- Practical Byzantine Fault Tolerence : (PBFT)

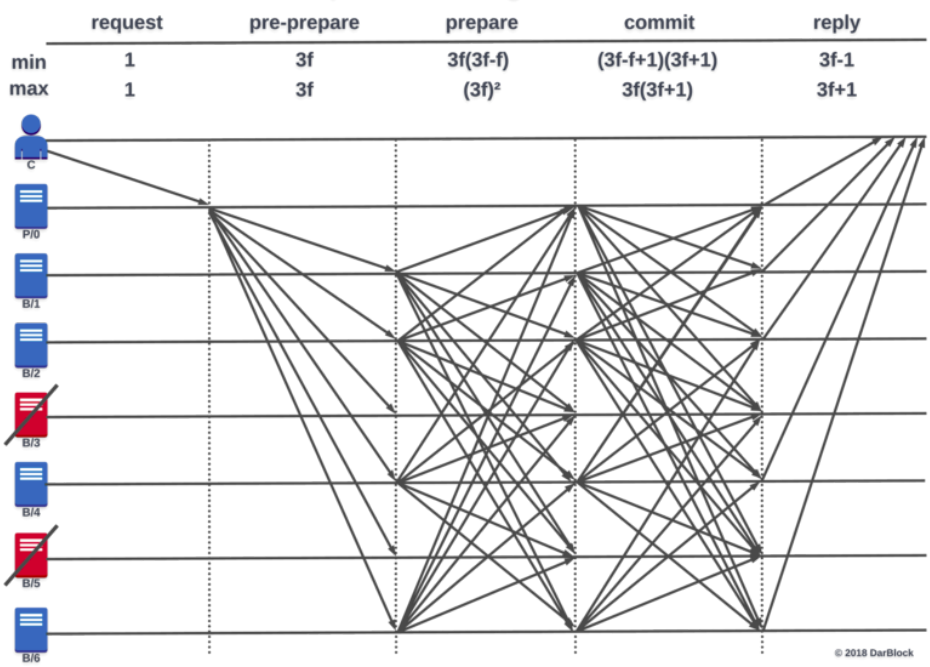

Figure3 :fonctionnement de Byzantine Fault Tolerence

"Byzantine Fault Tolerance" est l'habilité d'un système distribué basé sur le consensus à avoir assez d'éléments pour prendre une décision malgré la présence de nœuds malicieux ou défectueux en son sein. La problématique des généraux byzantins est à l'origine de cette réflexion mais le concept a été appliqué à beaucoup de systèmes critiques tels que les avions, les centrales nucléaires et les systèmes distribués modernes.
L'approche pratique (practical) prend en compte certaines suppositions:

- L'indépendance des nœuds en termes d'infection.
- La présence de messages altérés dans le système.

Dans ce système de validation, on tolère un nombre bien définit de nœuds malicieux **(m)** qui doit être inférieur au **tiers** du nombre de nœuds dans le réseau dans une fenêtre de vulnérabilité donnée. Dans ce cas, plus le réseau est grand, plus il devient sécurisé.
Dans ce système, les nœuds sont organisés de façon séquentielle avec un nœud Leader. Ils doivent communiquer entre eux de façon à décider du comportement du système en utilisant le principe de la majorité. A titre d'exemple, la validation, ou pas, d'un message.
La communication entre ces nœuds a deux buts:

- Prouver que le message est bien venu d'un nœud spécifique
- S'assurer que le message n'a pas été modifié durant la transmission.

Déroulement de l'algorithme:

- Un nœud envoie une requête au nœud Leader.
- Le Leader reçoit la requête puis la diffuse aux autres nœud réplica.
- Ces nœuds diffusent à leurs tours leurs réponses à la requête de façon à ce que chacun des nœuds sache ce que les autres ont répondu.
- Resultat:
    - Si **m+1** nœuds optent pour une **même** décision on est sûr qu'au moins **un nœud "honnête"** a répondu. Ainsi, la décision est Valide.
    - Si unanimité le système est sein autrement le/les intrus est/sont détecté(s).

Le nœud Leader, est constamment vérifié. On s'assure de ça transparence grâce à la signature électronique de la requête. De plus, il est systématiquement remplacé par un autre nœud de façon séquentielle en cas d'inactivité constatée.
Les limites:

- L'algorithme suppose que le traitement des nœuds est déterministe. Pour la même requête les nœuds seins doivent donner le même résultat.

Différence entre une application décentralisée et distribuée

Une application distribué est une application qui est lancée sur plusieurs serveurs quand le trafic de données est très important et la vitesse de calcul n'est pas assez haute sur une seule machine. Dans un tel système les données sont copiées sur plusieurs noeuds afin d'accroître leurs disponibilité. Une application peut donc être centralisée et distribuée en même temps mais si elle est décentralisée elle est obligatoirement distribuée.

La problématique de l'identité

Dans un environnement décentralisé, on est obligé d'avoir recours à des techniques plus avancées qu'un identifiant et un mot de passe afin de gérer les identités.

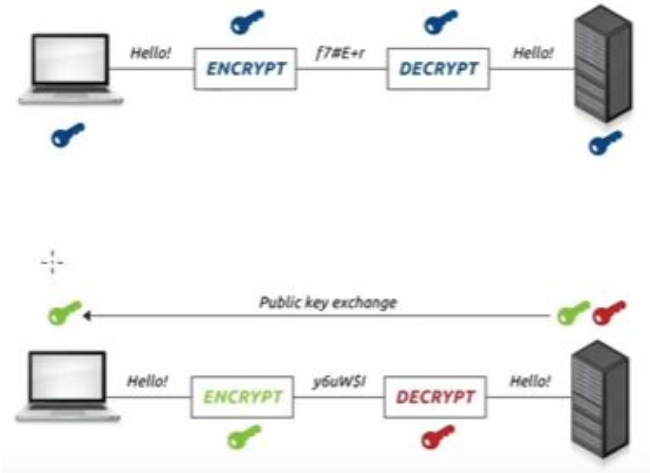

Figure4 : le chiffrage symétrique et asymétrique

Les systèmes les plus couramment utilisé repose sur les concepts de chiffrage symétrique (une clé) ou asymétrique (couple clé privée/clé publique) et les certificats électroniques.
Dans le cas symétrique, une même clé sert à chiffrer et à déchiffrer. Simple, mais à moi de trouver un moyen de se partager la clé pour la première fois sans qu'elle soit volée, cette solution reste vulnérable. Dans le cas asymétrique, on utilise deux clés mathématiquement liées de façon à ce que ce qui est chiffré avec l'une ne peut être déchiffré que par l'autre sans possibilité de dériver une clé de l'autre. C'est le concept de clé privée / clé publique. Avec ce système, je peux rendre disponible ma clé publique et je garde précieusement ma clé privée. Si quelqu'un chiffre un message avec ma clé publique il peut être sûr que je suis le seul à pouvoir le déchiffrer avec ma clé privée. Le problème de ce système est que si je reçois une clé publique comment pourrai-je savoir qu'elle appartient bien à la personne à qui je souhaite envoyer ? C'est là toute l'utilité des certificats électroniques. Ils introduisent le concept d'autorité de certification une tiers personne permettant

d’associer une clé publique à une personne, entité ou organisation.

Technologies blockchain

Différentes implémentations de la technologie donnent lieu à deux classes de blockchain avec leurs particularités et leurs applications respectives

	Public	Private/Federated
Acces	Open R/W	Permissioned R/W
Speed	Slower	Faster
Security	Proof of work Proof of stake Other consensus mechanisms	Pre Approved participants
Identity	Anonymous	Known identities
Asset	Native asset	Any asset

Les spécificités du projet et l'étude initiale ont conduit au choix du framework Hyperledger pour la réalisation du prototype en vue du caractère fédéré requis et des outils de développement disponibles sur ce framework qui sont open-source et supportés par la Linux Foundation.

Modèle Hyperledger Composer

Il modélise les entités, la logique et les interactions au sein du domaine du réseau Cette modélisation se fait sous une structure bien définie comportant essentiellement:

- Un fichier de modélisation (.cto)
- Des fonctions de traitement des Transactions (.js)
- Un fichier de contrôle d’accès (.acl)
- Un fichier de requêtes (.qry)

Développer des applications avec ce Framework à introduit aussi :

- Les mécanismes de sauvegarde d'état
- Les identités
- Les Profils de connexion
- Business Network cards

Des composantes régissant l'interaction avec la logique applicative.

HyperLedger Fabric: Aspect Réseau (infrastructure)

Fabric est l'aspect infrastructurel sous Hyperledger qui permet de mettre en place techniquement la notion de registre distribué pour les applications. Il permet aussi de définir les membres du réseau à un niveau plus général en termes de consortiums, d'organisations et de nœuds tout en définissants les règles de fonctionnement et de transaction.

Les différentes composantes du réseau

Un réseau se compose d'une combinaison de ces éléments de base.

- **Les registres (Ledgers):** On en retrouve un par canal.Le canal est une façon d'isoler les transaction de ses membres de celle des autres. c'est à dire que les nœuds appartenant à un même canal peuvent avoir un registre qui leur est dédié pour enregistrer leurs transactions sans que les non-membres ne puissent les voir.
- **Le chaincode:** le programme (smart contract) qui permet de lire et d'écrire sur les registres.
- **Les nœuds (Peer):** Une entité du réseau qui se charge d'exécuter le chaincode et de maintenir des registres.
- **Ordonnanceurs:** permettent de vérifier les transactions, de les ordonner et d'appliquer les règles établies pour le réseau.
- **Autorités de certification:** Permettent la création dynamique de certificats électroniques aux différentes entités du réseau et leurs utilisateurs.

IV. Travail réalisé

Prototype 0:

La réalisation du prototype 0 repose sur l'utilisation des frameworks offert par Hyperledger composer. Le but étant de faire une approche simplifiée afin de développer une vue d'ensemble sur le développement d'applications avec Hyperledger. Cette partie a servi d'introduction par une approche de modélisation simplifiée du développement.

Pour cette partie le travail s'est reposé essentiellement sur une infrastructure (Une réseau) par défaut mais avec plus d'importance à la modélisation de la logique applicative.
Cette approche a conduit à un système de Broadcast simple et fonctionnel regroupant des entités de type User qui sont les acteurs du réseau qui peuvent via une transaction Broadcast créer une

instance de type Message qui sera signée par l'identité de son émetteur. Le prototype implémente aussi:

- Une vérification d'identité : à travers un concept de signature (mot de passe) un utilisateur ne peut signer un message que de son nom et seulement s'il est existant dans le système.
- Seul un Admin peut créer des User.
- Aucune entité ne peut créer de message à travers autre chose que la transaction de broadcast.
- Un User peut mettre à jour ces informations.
- Un utilisateur ne peut pas accéder au informations complètes d'un autre User ni d'un Admin.
- Toutes les entités peuvent lire les messages diffusés.
- On ne peut pas éditer un message diffusé.
- Des requêtes simples pour lister les messages (les plus récents en premier, pour un utilisateur spécifique ...).

Pour automatiser la manœuvre de déploiement et de test, quelques scripts shell reprennent les étapes importantes du processus:

Le script bash *mylauncher.sh* permet de:

- préparer et initialiser notre réseau.
- déployer le modèle.
- créer un utilisateur admin par défaut et se connecter avec ses identifiants.
- pinger le réseau
- générer un API-REST

Sur une machine virtuelle nous faisons le déploiement de notre réseau avec nos scripts. Nous créons donc un réseau avec:

- Un Ordonnanceur (Orderer)
- Une Autorité de certification (CA)
- Un Nœud (peer0)

pour ce faire nous lançon leurs centenaires respectifs avec cette configuration reprise dans le ***docker-compose.yml***. Comme chaque entité du réseau doit avoir un certificat et une clé privée,

nous utilisons les outils fournis pour les générer automatiquement (*fabric-tools/bin/cryptogen...*). La génération est automatisée grâce au modèle définit par ***crypto-config.yml*.** Suite à cela, il faut initialiser notre blockchain en créant manuellement le premier block *GenesisBlock* en utilisant ***configtxgen*.** Maintenant, il est nécessaire de créer un tunnel entre notre nœud et le validateur leur permettant de communiquer. Nous nous connectons avec l'identité Admin pour pouvoir installer notre modèle de réseau *.bna* puis nous le démarrons. Nous testons si le réseau a bien démarré et nous lançons la génération de l'API-REST sans les mesures de sécurité (sans login et en http clair pour le moment) sur le port *3000* de la VM. La seconde partie nous ne faisons que interagir avec le réseau en créant des utilisateurs .Des requêtes *HTTP* sont envoyées avec **curl** sur le l'API pour aussi réaliser des "broadcasts" de messages. Le système implémente vraisemblablement toutes les fonctionnalités attendues mais reste tout de même limité.

Limites

Le fait d'avoir utilisé le framework a certes apporté beaucoup de simplicité mais il a aussi masqué beaucoup de détails sur l'implémentation, le développement et l'architecture. De plus pas toutes les attentes ont été réalisées. Malgré cela, il a eu le mérite de donner une vue d'ensemble sur le processus.

Prototype 1:

Choix de la topologie du réseau

Lors de la réalisation du prototype 0, une phase importante n'a pas été explicité relevant de la spécification du réseau lui-même car un modèle par défaut a été utilisé. Le but ayant été d'aboutir à un prototype qui permet d'appréhender les différentes étapes sans se perdre au premier passage dans les détails qui risqueraient d'obscurcir la vue d'ensemble. En revanche cette étape est expliquée ici en détails.

Après entretien avec mes tuteurs il m'a été conseillé de réaliser un réseau pour un cas de figure particulier avec un scénario d'usage orienté "métiers". Le scénario est le suivant: Imaginons que les différents ministères d'un état souhaitent mettre en place un réseau en utilisant la technologie dont il est question ici. Ces ministères coopèrent ensemble et s'échangent des messages en obéissant à des protocoles donnés pour informer mutuellement ou enclencher des procédures.

Pour faire court, réaliser une topologie semblable au prototype 0 (UN ordonnanceur + Un noeud) est insensée car on perdrait l'avantage d'un système distribué en centralisant mais aussi les performances seront catastrophiques sans parler de l'aspect sécurité qui n'est géré par nous qu'au niveau applicatif.

Ceci dit, il est maintenant nécessaire d'expliciter comment une transaction se déroule sur un réseau Hyperledger-Fabric pour comprendre comment dimensionner le notre.

A. Flux de transaction

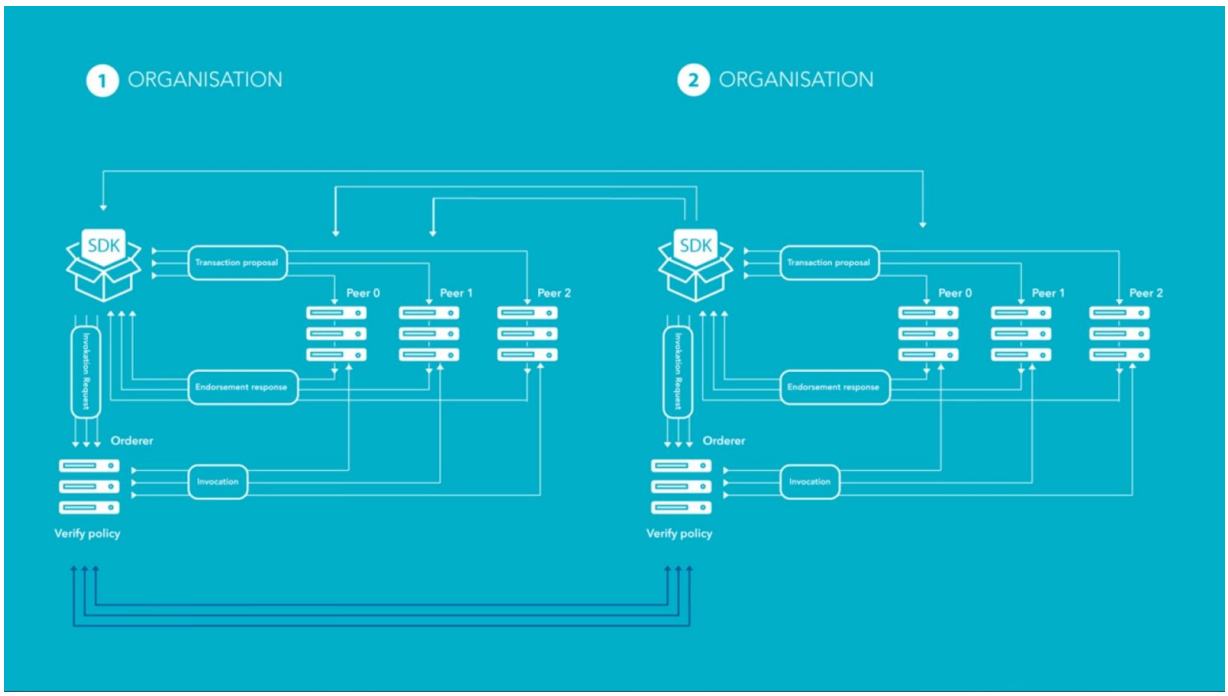

Figure5 : flux de transaction

Supposons qu'un réseau est en place. Un client va initier une transaction (création, modification, suppression ... d'une entrée dans le registre distribué). Il fait donc appel à l'SDK qui lance une **proposition de transaction**. Cette requête est simulée (calculée par l'exécution du chaincode avec les arguments soumis) par tous les nœuds qui renvoient la requête, les arguments, les résultats et les signatures des nœuds avec la version de l'entrée. Ce numéro de version permet de garantir l'intégrité des données (nous le verrons plus bas). Donc, les nœuds renvoient leurs **approbations** au client. Après avoir vérifié les réponses et les signatures, le client renvoie une **requête d'invocation** à l'ordonnanceur du réseau celui-ci vérifie encore une fois les signatures cryptographiques, les réponses des nœuds et le respect des règles établie et ajoute la transaction à

sa file. Cette file est paramétrable en temps et en nombre de transactions. Une fois une des conditions validée, un ordre **d'invocation** est émis vers les nœuds pour mettre à jour l'entrée. C'est à ce niveau que la version de l'entrée est utile car elle permet de vérifier si le calcule a été fait avec la version la plus récente de la donnée autrement la transaction est enregistrée mais la mise à jour n'est pas faite (le calcul doit être refait).

B. Topologie

Nous avons mentionné plus haut l'existence de certaines entités pour lesquelles il est maintenant temps pour plus d'explications.

- **Une Organisation:** C'est le plus petit réseau possible on peut faire une analogie avec un réseau LAN (seulement une analogie). C'est à dire il permet l'interconnexion de nœuds avec leurs différents services. Dans le paradigme Hyperledger elle peut être assimilée à une organisation physique ou une entreprise avec cette notion de réseau local.
- **Un consortium:** C'est la résultant de l'interconnexion de plusieurs organisations.

Il est aussi utile de mentionner que l'on peut avoir plusieurs Ordonnanceurs pour une seule Organisation et même un ordonnanceur partagé par plusieurs. Par défaut il est nécessaire de créer au moins un canal au niveau de l'organisation et d'y inscrire tous les nœuds pour qu'ils puissent se parler.
Maintenant que nous savons cela nous pouvons dire que pour simuler notre réseaux de ministères il est approprié que chaque ministère soit représenté par une organisation qui, à sont tour, sera composée de plusieurs nœuds. En raison de la puissance de calcul à disposition nous nous contenterons de simuler **un réseau de 2 ministères avec 2 nœuds dans chacun.** Pour avoir un réseau décentralisé, il faudrait mettre en place plusieurs ordonnanceurs aussi. Mais en première approche nous nous contenterons d'un seul. Aussi nous créerons d'emblée les certificats et les clés privées de quelques utilisateurs. L'interconnexion des deux organisations constitue donc **un Consortium**.

C. Génération des certificats cryptographiques et des clés privées

Toute entité du réseau doit avoir son propre contenu cryptographique (Clé privée et certificat électronique) afin de pouvoir s'identifier auprès des autres. Le standard utilisé ici est le X.509 qui est largement utilisé dans beaucoup de protocoles internet (ex:HTTPS). Grâce au programme

cryptogen dans la suite d'outils téléchargés il est possible de générer les artefacts cryptographiques en fournissant la topologie du réseau que l'on souhaite créer sous un format (.yaml).
Dans le fichier **p22CryptoConfig.yaml**, nous définissons les spécificité de notre infrastructure. Plus de détails sont mentionnés sur le Wiki.

D. Création du bloc de genèse, configuration des canaux et des nœuds d'encrage

Il est possible d'accomplir les phases de configuration citées dans le titre de cette section en utilisant un fichier de configuration (.yaml) et le programme **configtxgen**. Les détails des configurations sont mentionnés sur le wiki.
Maintenant que notre fichier de configuration est prêt nous procédons à la génération du bloc de genèse. Le bloc de genèse (Genesis Block) permet l'initialisation de la blockchain mais aussi la sauvegarde de toute la configuration du réseau de façon immuable dans le registre distribué lui-même. De cette manière celle-ci n'est modifiable qu'en passant par une procédure spéciale nécessitant une authentification Administrateur ou l'arrêt total. En parallèle, la création du tunnel par défaut a été faite. Nous procédons alors au

- Canal inter-organisation
- Nœud d'encrage sur le ministère 1
- Nœud d'encrage sur le ministère 2

E. Configuration des conteneurs

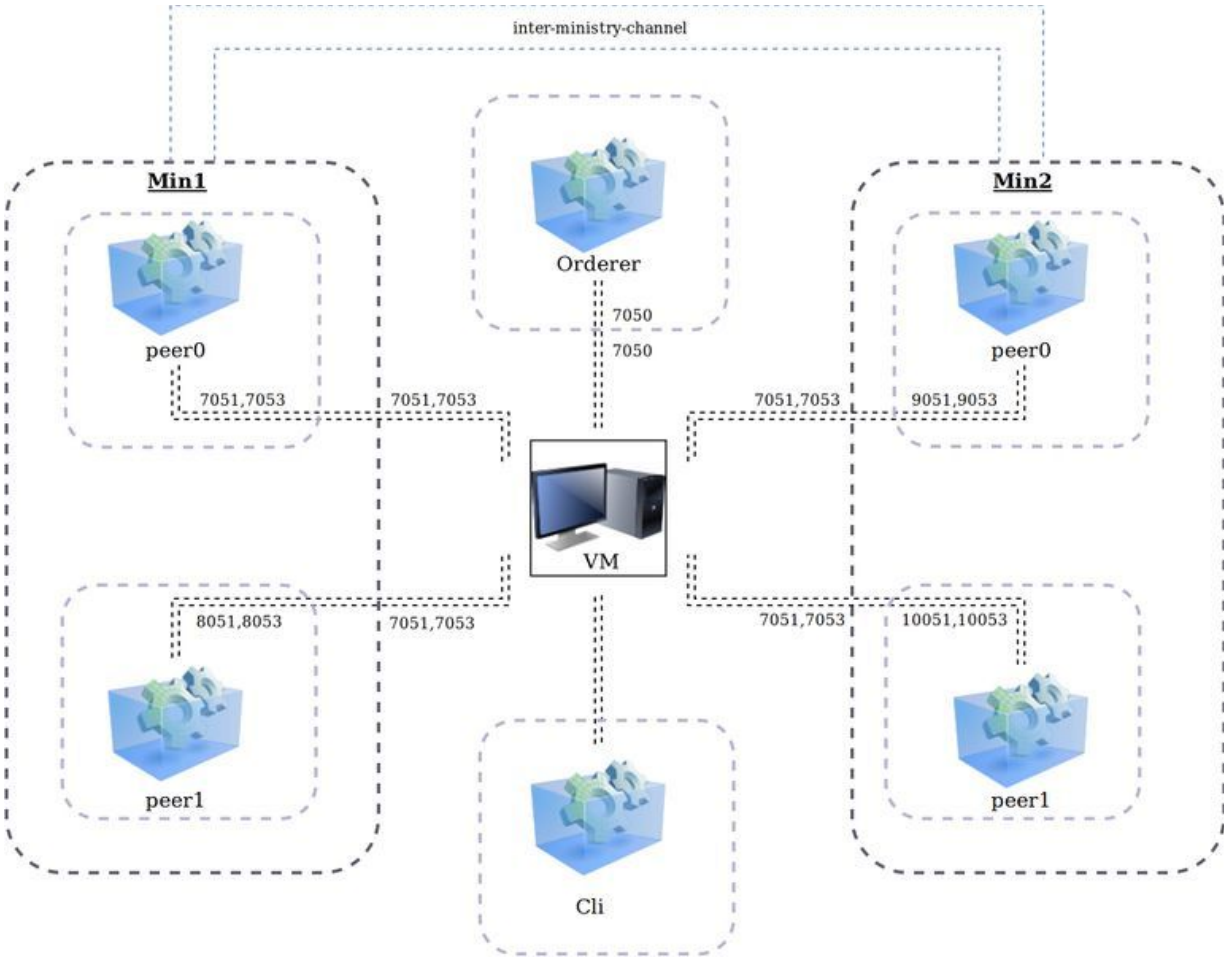

Figure6 : Modélisation du réseau

Vu que nous travaillons avec docker et en vue du nombre de conteneurs que nous devons lancer et configurer il est nécessaire d'automatiser cette tâche. Plusieurs systèmes de gestion de conteneurs peuvent faire le travail.De ce fait l'usage de docker n'est pas indispensable mais en vue de la présence d'une meilleure documentation pour celui-ci, il est préférable de l'utiliser. Nous utiliserons docker-compose pour créer les scripts de déploiement. Vue que nous voulons implémenter un réseau avec un ordonnanceur et 4 nœuds il est donc intuitif de préparer un conteneur pour chacun d'eux. Pour mieux gérer ce réseau on va aussi ajouter un conteneur qui va nous servir de ligne de commande pour effectuer les interactions avec le réseau en le configurant avec les variables d'environnement et le contenu nécessaire. De ce fait nous gardons un environnement propre. Vu que tous les nœuds auront plus ou moins la même configuration à quelques détails près il sera donc judicieux de faire une configuration de base (*peer-base.yaml*) et la faire hériter par les différents nœuds.

Automatisation du déploiement du réseau

Les grandes lignes du travail accompli par le script de déploiement est donc de:

1. Permettre de régénérer si besoin les artefacts cryptographiques.
2. Permettre de régénérer aussi le bloc de genèse, les configs de nœuds d'encrage ...etc
3. Permettre de démarrer le réseau à partir de la configuration.
4. Permettre d'ajouter les options.
5. Se déplacer vers le conteneurs "cli" avec un "docker exec" afin d'exécuter les scripts de déploiement et d'installation de chaincode.
6. Arrêter proprement le réseau.
7. Nettoyer l'environnement.

Automatisation de l'installation et de l'instanciation du chaincode

Souvenez-vous qu'en configurant les conteneurs nous avons monté un dossier "scripts" qui doit contenir les scripts dont il est question ici. Nous préparons un script qui permet d'exécuter certaines tâches pour finaliser le lancement du réseau et l'installation automatique du chaincode sur les nœuds. Après nous testons le tout avec un exemple. Le tout fonctionne parfaitement.

Développement de la logique applicative

Nous parlerons ici des fonctionnalités qu'implémentera le chaincode que nous allons développer pour les noeuds. Dans le contexte de notre application de broadcast de messages.

A. Choix du langage

Comme annoncé précédemment le développement peut se faire avec plusieurs langages, mais nous préférons le faire en Java pour la familiarité avec ce dernier et son rapport avec les cours vu durant cette année. Nous utiliserons pour la gestion des dépendances l'outil Gradle car il est nécessaire pour que les noeud puissent faire le build au moment de l'installation du chaincode.

C. Stockage et structure de données

Pour plus de simplicité et afin d'aboutir plus rapidement à un prototype fonctionnel nous optons pour un stockage utilisant **stateDB** pour le maintien du registre distribué. Cette base de données se présente sous forme de couples <clé : valeur>. Le seul compromis que nous faisons contrairement à l'usage d'un autre système de stockage plus évolué tel que **coucheDB** est le fait de ne pas bénéficier

des fonctionnalités de requêtes riches. Ceci n'est pas vraiment un problème car nous avons une solution, qui, dans le cadre de l'application que nous souhaitons réaliser, nous permettra de nous en affranchir:
Sur stateDB les possibilités offertes par l'API sont le stockage:

- <clé(String) : valeur (byte[])> : en utilisant putState().
- <clé(String) : valeur (String)> : en utilisant putStringState().

La solution que nous proposons est d'utiliser la seconde combiné à la library **org.json** nous permettant d'utiliser des structures de données avancées (Objets JSON) et une particularité essentielle de sérialisation et de désérialisation en String pour faciliter le stockage. Vous avez sans doute remarqué qu'elle a été ajoutée à notre fichier de gestion des dépendances.
En ce qui est de la structure de données nous aurons un **Id** pour clé représentant l'Identifiant d'un utilisateur du système. La valeur, comme annoncé précédemment, sera un structure **User** représentée par ce squelette Json.

```
{
 "Identity":{
            "NAME" : .....,
            "ROLE" : .....,
            "PWD"  : .....,
            "AFFIL": .....,
            "CERT" : .....
          },
 "Messages":[
           {.......},
           {.......},
            .......
          ]
}
```

La clé **Messages** représentera un tableau qui contiendra des Objets de type **Message** représenté ci-dessous:

```
{
    "ID"     : ......,
    "TIME"   : ......,
    "TITLE"  : ......,
    "CONTENT" : ......
}
```

Tests de fonctionnement

A partir du conteneur **cli** nous pouvons, grâce au CLI API de Fabric nous pouvons lancer des requêtes d'invocation depuis le terminal et ainsi tester le bon fonctionnement avec quelques scripts shell.

- **addUser.sh**: ajouter un utilisateur client cette requête ne marche que si l'invocateur est administrateur

resultat:

...
...
2019-05-05 21:25:37.128 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 04f Chaincode invoke successful. result: status:200 message:"User added Successfully:\n"
payload:"{\"Messages\":[],\"Identity\":{\"ROLE\":\"Client\",\"AFFIL\":\"Min1\",\"CERT\":\"null\",\"PWD\":\"passwdofuser1\",\"NAME\":\"user1\"}}"

- **broadcastMsg.sh**: poster un message

resultat:

...
...
19-05-05 21:27:43.467 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 04f Chaincode invoke successful. result: status:200 message:"Message broadcasted Successfully:\n"
payload:"{\"Messages\":[{\"TIME\":\"2019-05-09T21:27:43.407585253Z\",\"TITLE\":\"Viva la vida\",\"CONTENT\":\"Lorem epsum in dolore\",\"ID\":\"70b4817e6c43c8711711c89ed0981d7e35b3730e93e6c49ab4a177200eaa2c55\"}],\"Identity\":{\"ROLE\":\"Client\",\"AFFIL\":\"Min1\",\"CERT\":\"null\",\"PWD\":\"passwdofuser1\",\"NAME\":\"user1\"}}"

- **broadcastMsgUserNotExist.sh**: poster un message avec un utilisateur non-existant

resultat:

...
...
2019-05-05 21:29:45.070 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> DEBU 04c ESCC invoke result: response:<status:500 message:"transaction returned with failure: \nError: USER WITH ID [9999] DOES NOT EXIST\n" > Error: endorsement failure during invoke. response: status:500 message:"transaction returned with failure: \nError: USER WITH ID [9999] DOES NOT EXIST\n"

- **broadcastWrongPass.sh**: poster un message avec un mauvais mot de passe pour l' Id utilisateur fourni.

resultat:

...
...
2019-05-05 21:32:07.955 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> DEBU 04c ESCC invoke result: response:<status:500 message:"transaction returned with failure: \nError: WRONG PASSWORD. NO RIGHTS TO PUBLISH" > Error: endorsement failure during invoke. response: status:500 message:"transaction returned with failure: \nError: WRONG PASSWORD. NO RIGHTS TO PUBLISH"

...
Les résultats des tests semblent concluants.

Développement de l'application client

Sur Fabric, l'application client est ce qui relie notre chaincode déployé sur le réseau au monde extérieur. Cette interaction passe à travers le SDK qui est disponible dans plusieurs langages (Java, NodeJs). Nous utiliserons cette fois le SDK Node pour deux principales raisons:

- Documentation plus importante qu'en Java.
- Facilité de créer un API rest moyennant des packages javaScript.

Les packages NPM du SDK sont:

- fabric-client: fonctions nécessaires à l'instantiation, la soumission des requêtes et des transactions vers le réseau.
- fabric-ca-client: pour interagir avec les autorités de certification de Hyperledger Fabric. Gestion des enregistrement des clients et des rôles.

a. Profils de connexion

Un profil de connexion est un moyen de conserver les informations nécessaires sur l'infrastructure de notre réseau pour que nos client puissent s'y connecter. Il contient entre autres les **urls** vers les différentes entités ainsi que les certificats cryptographiques nécessaires.

b. Authorités de certification

Les autorités de certification servent à fournir, révoquer ou vérifier des certificats. Pour avoir accès à ce service, il existe pour Fabric un conteneur spécial remplissant cette fonction **fabric-ca**.Etant donné que nous avons deux organisations (Ministères) il va donc falloir en créer deux.

c. Gestion des utilisateurs

adminEnrollment.js

En utilisant le SDK sur NodeJs, nous enregistrons l'utilisateur administrateur auprès de l'autorité de certification. Une fois, fois que nous avons un administrateur nous pouvons enregistrer des clients. Le résultat de cette opération et la génération d'un couple Clé privée / clé publique et d'un certificat pour l'administrateur.

registerUser.js

Nous montrons ici comment enregister un utilisateur "user1" auprès de l'autorité de certification. Le résultat de cette opération et la génération d'un couple Clé privée / clé publique et d'un certificat pour l'utilisateur "user1".

chaincodeInvoke.js {problème}

Cette partie est supposée nous permettre d'invoquer notre chaincode et ainsi d'exécuter des transactions. Malheureusement, nous n'arrivons pas à déterminer l'origine du problème empêchant la communication entre le script et le réseau malgré le fait qu'ils soient tous les deux en localhost.

Propositions d'amélioration

Durant cette première approche introductive, beaucoup de décisions ont été prises lors de la réalisation de façon à simplifier le travail. Ce qui offre des possibilités d'amélioration importantes:

- Déverminer les fonctionnalités non-abouties.
- Implémenter l'interface web.
- Revoir l'architecture du réseau: ajouter notamment des ordonnanceurs pour décentraliser le travail (Kafka).
- Etoffer les fonctionnalités implémentées par le chaincode par des fonctions de recherches avancées.
- Utiliser la base de données CouchDB pour avoir accès à des fonctionnalités avancées.
- Explorer plus en profondeur la gestion de l'accès.
- Implémenter sur un vrai réseau et non en localhost.

V. Difficultés rencontrées

- Domaine sans connaissances préalables.
- Documentation très vagues sur certains points.
- Technologie récentes et donc peu de support pour le déverminage en ligne.
- Majorité des exemples et certaines librairies ne sont disponibles qu'en langage Go.
- Surcharge de travail entre la recherche , l'apprentissage et l'implémentation sans oublier le wiki et le rapport pour une personne.

VI. Conclusions

Les champs d'exploitation de la technologie blockchain sont immenses : banques, assurance, santé et industrie pharmaceutique, supply chain de nombreux secteurs (agroalimentaire, luxe, commerce international, distribution, vins, aéronautique, automobile…), industrie musicale, énergie, immobilier, vote…

Surtout, la blockchain ouvre la voie d'un nouveau web, le web décentralisé, et d'une nouvelle économie numérique. Bien évidemment, ces promesses ne sont pas exemptes de défis, qu'ils soient économiques, juridiques, de gouvernance, ou encore écologiques.

Malgré que le projet n'a pas abouti à 100% à cause d'un soucis sur la phase finale de test. Il aura été d'un grand intérêt car il a constitué une opportunité de travailler sur une technologie innovante et d'aborder beaucoup de sujets intéressants et d'actualité.