BLOCKCHAIN

La révolution de l'économie de partage

SMILE

I.T IS OPEN

I - PREAMBULE

I.1 SMILE

Smile est le leader européen de l'intégration et de l'infogérance de solutions open source. Smile est membre de l'**APRIL**, l'association pour la promotion et la défense du logiciel libre, du **PLOSS** – le réseau des entreprises du Logiciel Libre en Ile-de-France et du **CNLL – le conseil national du logiciel libre**.

Smile compte plus de 1200 collaborateurs dans le monde ce qui en fait le premier intégrateur français et européen de solutions open source.

Depuis 2000 environ, **Smile mène une action active de veille technologique** qui lui permet de découvrir les produits les plus prometteurs de l'open source, de les qualifier et de les évaluer, de manière à proposer à ses clients les produits les plus aboutis, les plus robustes et les plus pérennes.

Cette démarche a donné lieu à **toute une gamme de *livres blancs*** couvrant différents domaines d'application. La gestion de contenus, les portails, la business intelligence, la virtualisation, la gestion électronique de documents, l'e-commerce, les Réseaux sociaux d'Entreprise, le Guide de l'open source, NoSQL, le Mobile, et plus récemment le Big data, l'ERP open source pour l'e-commerce, le DevOps et Linux pour l'embarqué. Chacun de **ces ouvrages présente une sélection des meilleures solutions open source** dans le domaine considéré, leurs qualités respectives, ainsi que des retours d'expérience opérationnels.

Au fur et à mesure que des solutions open source solides gagnent de nouveaux domaines, Smile sera présent pour proposer à ses clients d'en bénéficier sans risque. Smile apparaît dans le paysage informatique français comme **le prestataire intégrateur de choix** pour **accompagner** les plus grandes entreprises dans l'adoption des meilleures solutions open source.

Ces dernières années, Smile a également étendu la gamme des services proposés. Depuis 2005, un département consulting accompagne nos clients, tant dans les phases d'avant-projet, en recherche de solutions, qu'en accompagnement de projet. Depuis 2000, Smile dispose d'un studio graphique, devenu en 2007 Smile Digital – agence interactive, proposant outre la création graphique, une expertise e-marketing, éditorial, et interfaces riches. Smile dispose aussi d'une agence spécialisée dans la TMA (support et l'exploitation des applications) et d'un centre de formation complet, Smile Training.

Enfin, Smile est implanté à Paris, Lille, Lyon, Grenoble, Nantes, Bordeaux, Marseille, Toulouse et Montpellier. Et présent également en Suisse, au Benelux, en Ukraine, en Russie, au Maroc et en Côte d'Ivoire.

I.2 QUELQUES REFERENCES DE SMILE

SMILE est fier d'avoir contribué, au fil des années, aux plus grandes réalisations Web françaises et européennes. Vous trouverez ci-dessous quelques clients nous ayant adressé leur confiance.

I.2.a Sites Internet

EMI Music, Salon de l'Agriculture, Mazars, Areva, Société Générale, Gîtes de France, Patrice Pichet, Groupama, Eco-Emballage, CFnews, CEA, Prisma Pub, Véolia, NRJ, JCDecaux,
01 Informatique, Spie, PSA, Boiron, Larousse, Dassault Systèmes, Action Contre la Faim, BNP Paribas, Air Pays de Loire, Forum des Images, IFP, BHV, ZeMedical, Gallimard, Cheval Mag, Afssaps, Benetaux, Carrefour, AG2R La Mondiale, Groupe Bayard, Association de la Prévention Routière, Secours Catholique, Canson, Veolia, Bouygues Telecom, CNIL…

I.2.b Portails, Intranets et Systèmes d'Information

HEC, Bouygues Telecom, Prisma, Veolia, Arjowiggins, INA, Primagaz, Croix Rouge, Eurosport, Invivo, Faceo, Château de Versailles, Eurosport, Ipsos, VSC Technologies, Sanef, Explorimmo, Bureau Veritas, Région Centre, Dassault Systèmes, Fondation d'Auteuil, INRA, Gaz Electricité de Grenoble, Ville de Niort, Ministère de la Culture, PagesJaunes Annonces…

I.2.c E-Commerce

Krys, La Halle, Gibert Joseph, De Dietrich, Adenclassifieds, Macif, Furet du Nord, Gîtes de France, Camif Collectivité, GPdis, Projectif, ETS, Bain & Spa, Yves Rocher, Bouygues Immobilier, Nestlé, Stanhome, AVF Périmédical, CCI, Pompiers de France, Commissariat à l'Energie Atomique, Snowleader, Darjeeling…

I.2.d ERP et Décisionnel

Veolia, La Poste, Christian Louboutin, Eveha, Sun'R, Home Ciné Solutions, Pub Audit, Effia, France 24, Publicis, iCasque, Nomadvantage, Gets, Nouvelles Frontières, Anevia, Jus de Fruits de Mooréa, Espace Loggia, Bureau Veritas, Skyrock, Lafarge, Cadremploi, Meilleurmobile.com, Groupe Vinci, IEDOM (Banque de France), Carrefour, Jardiland, Trésorerie Générale du Maroc, Ville de Genève, ESCP, Sofia, Faiveley Transport, INRA, Deloitte, Yves Rocher, ETS, DGAC, Generalitat de Catalunya, Gilbert Joseph, Perouse Médical…

I.2.e Gestion documentaire

Primagaz, UCFF, Apave, Géoservices, Renault F1 Team, INRIA, CIDJ, SNCD, Ecureuil Gestion, CS informatique, Serimax, Véolia Propreté, NetasQ, Corep, Packetis, Alstom Power Services, Mazars…

I.2.f Infrastructure et Hébergement

Agence Nationale pour les Chèques Vacances, Pierre Audoin Consultants, Rexel, Motor Presse, OSEO, Sport24, Eco-Emballage, Institut Mutualiste Montsouris, ETS, Ionis, Osmoz, SIDEL, Atel Hotels, Cadremploi, SETRAG, Institut Français du Pétrole, Mutualité Française…

I.2.g **Systèmes industriels et embarqués**

Groupe THALES, NEXTER, Airbus, Sagemcom, Sagem Défense, Stago, ST Microeletronics, Intel, Dassault Aviation, RATP, Coyote...

Consultez nos références, en ligne, à l'adresse : *https://smile.eu/fr/nos-references*

I.3 **AUTEUR**

Merci à Christophe DOROTHE, Chef de projet R&D chez Smile, pour la rédaction de ce livre blanc.

II - SOMMAIRE

III - POURQUOI CE LIVRE BLANC

Comptant plus de **410 start-up positionnées sur la Blockchain** en juillet 2016, elle est devenue en 2017, après plus ou moins un an de « gestation » et d'expérimentations, une réalité technologique et économique dépassant ainsi, le simple phénomène de mode.

Considérée comme la « seconde révolution Internet », elle offre un changement de paradigme des systèmes de gouvernance actuels qui partent du principe que seul un ordre hiérarchique et centralisé permet de maintenir la cohésion d'une société humaine. Elle fait passer d'un système pyramidal à un ordre social qui émerge spontanément du comportement et des interactions des individus sans qu'aucune instance planificatrice ou créatrice n'ait structuré ou organisé un tel ordre. Elle forme une **société de réseau « d'ordre spontané ».**

Vraie opportunité économique de nouveaux services, elle s'est affranchie des cadres financier et bancaire dans lesquels elle était cantonnée. Depuis, la Blockchain s'est largement « répandue » et impacte de nombreux secteurs d'activités hétérogènes.

Portée par son succès et la dynamique de ses acteurs, de nombreux POC (Proof of Concept) ont vu le jour et on voit aujourd'hui apparaître les premiers cas d'usages concrets.

Complexe et sujette à bien d'interrogations, la compréhension et la maitrise de la technologie Blockchain nécessite la mobilisation de ressources importantes (temps, humain, financier) d'autant plus conséquentes, qu'elle fait intervenir une équipe pluridisciplinaire (DSI, marketing, relation client, etc.).

Malgré un investissement important, la Blockchain est, à n'en pas douter, le sujet *hype* du moment et un sujet sur lequel il faut se positionner.

Comme toute nouvelle technologie, la Blockchain dispose de son propre vocabulaire et s'accompagne de ses propres concepts. Loin d'être exhaustive, la première partie de ce livre blanc se concentre sur les principaux termes et concepts fondamentaux. Elle s'attache à les décrire et à les expliquer.

Afin d'illustrer comment la Blockchain peut être opérée et les avantages significatifs qu'elle apporte par rapport aux processus « classiques », la seconde partie présente deux cas d'usage de la Blockchain, le premier dans la gestion de l'énergie et le second dans la *supply chain*.

Dynamique et en pleine effervescence, son écosystème compte, aujourd'hui, un grand nombre de solutions et de protocoles différents car et il n'existe pas une, mais des blockchains !

Parmi ces solutions, deux en particulier se « distinguent » et connaissent un succès grandissant, il s'agit des blockchains open source **Ethereum** et **HyperLedger**, sur lesquelles d'ailleurs, on été bâtis plusieurs solutions blockchains annexes.

Ces deux blockchains ainsi que les solutions permettant de les opérer sont décrites dans la troisième partie de ce livre blanc.

La quatrième et dernière partie conclut ce livre blanc.

IV - LES CONCEPTS

IV.1 BLOCKCHAIN

IV.1.a Introduction

En octobre 2008, Satoshi Nakamoto publie son livre blanc «**Bitcoin : A Peer-to-Peer Electronic Cash System**» dans lequel il définissait un **système de monnaie électronique cryptographique pair-à-pair**, le **bitcoin**. En février 2009, il diffuse la première version du logiciel **Bitcoin** sur le site **P2P Foundation** et pour faire fonctionner le réseau, met à contribution son ordinateur. Il génère ainsi les premiers bitcoins. Se fondant sur un nouveau « protocole », une « nouvelle » technologie, la principale originalité du réseau **Bitcoin** tiens dans son « architecture » opérée au travers d'une chaîne de blocs. Satoshi Nakamoto venait de donner naissance à la **Blockchain**.

Sortie des cercles *geeks*, éclot au grand jour au début de l'année 2016, véritable *Buzzword*, la technologie Blockchain s'est imposée comme **le** grand sujet dans le monde numérique et sociétal. "Révolutionnaire", " Technologie d'une ampleur inédite", "Innovation de rupture sans précédent", dans les différents médias et la presse, les superlatifs ne cessent de s'accumuler. Devant cet emballement médiatique et la promesse (faite) de nouveaux modèles économiques, bon nombre d'investisseurs et de sociétés mobilisent dès à présent, des ressources financières importantes pour s'assurer, en se positionnant acteurs de cette « révolution », de ne pas manquer ces nouveaux et futurs marchés.

Véritable phénomène, elle repense tous les usages et impacte tous les secteurs d'activité. Elle a révolutionné la monnaie fiduciaire avec les bitcoins et elle va maintenant « **disrupter** » les banques, mais aussi les notaires, les avocats, les agents immobiliers, le monde de l'énergie, la santé, la culture, les administrations, ..., En résumé, ses usages semblent illimités. Présentée comme « The world Computer », elle permet en effet, à tous, de concevoir et distribuer des projets et des solutions qui repensent les organisations politiques, économiques et sociales.

IV.1.b Définition

Une Blockchain est une technologie « open source », de stockage et de transmission de données, décentralisée, désintermédiée, sécurisée, infalsifiable et fondée sur des échanges P2P.

Par extension, elle constitue une base de données publique (ou non dans le cadre des Blockchain privées) décentralisée fiable, inviolable et sans organe central de contrôle. Elle peut être assimilée à un grand livre de comptes anonymes.

Cette technologie se distingue, entre autres, par quatre caractéristiques majeures qui en font sa force :

Immutable :

Une fois les informations et les échanges inscrits dans une Blockchain, ils ne peuvent plus être modifiés et y sont stockés définitivement.

Transparente :

Il n'y a pas de « droit d'entrée ». Tout le monde peut consulter l'**ensemble** des échanges et transactions enregistrés sur une blockchain et ce depuis le premier bloc créé.

Pseudonyme :

Les utilisateurs sont identifiés par un « numéro de compte » rendant leur identification impossible.

Sécurisée :

Asymmetric key Encryption, Résolution de problèmes cryptographiques pour la validation des blocs. Son caractère distribué lui assure également une sécurité puisque tous les blocs sont répliqués sur tous les nœuds du réseau.

IV.1.c **Typologie**

Il existe trois « types » de blockchain. Chacune correspondant à un périmètre précis :

Publique :

Il s'agit de blockchains accessibles à n'importe qui dans le monde. Aucune permission n'est à demander pour effectuer des transactions ou pour participer au processus de consensus. Tous les acteurs sont en situation égalitaire dans leur participation au réseau. **Bitcoin** et **Ethereum** sont les deux principales blockchains publiques.

Privée :

Il s'agit de blockchains tournant sur un réseau privé, dans lesquelles tous les participants sont connus et pour lesquelles la gouvernance est assurée par une organisation. Personne ne peut y accéder et y participer sans y être autorisé.

Consortium (Hybride) :

Il s'agit de blockchains dans lesquelles le processus de consensus (validation des transactions/blocs) est contrôlé par un nombre connu et restreint de nœuds. Certains nœuds peuvent être rendus publics (accès autorisé en lecture seule) tandis que d'autres restent privés. Elles sont plus adaptées aux contextes régulés.

IV.1.d **Fonctionnement**

Les transactions effectuées entre les utilisateurs du réseau sont regroupées dans une structure de données appelée **bloc**. Les blocs sont ordonnés et hiérarchisés dans une seule et unique chaine. Chaque bloc pointe sur le dernier bloc précédent valide. Cette chaîne est « distribuée » et « répliquée » sur tous les nœuds du réseau.

Chaque nouvelle transaction et/ou Smart Contract, « en attente », sont groupés dans un nouveau bloc. Pour que ce bloc puisse être ajouté à la Blockchain, il faut qu'il soit validé.

Cette validation est réalisée au travers de certains nœuds spécifiques du réseau appelés « **miners** ». Le rôle de ces nœuds consiste à répondre à un *casse-tête crypto*

mathématique complexe. Chaque réponse trouvée est propre à un et à un seul bloc interdisant ainsi, sa réutilisation pour la validation d'un nouveau bloc.

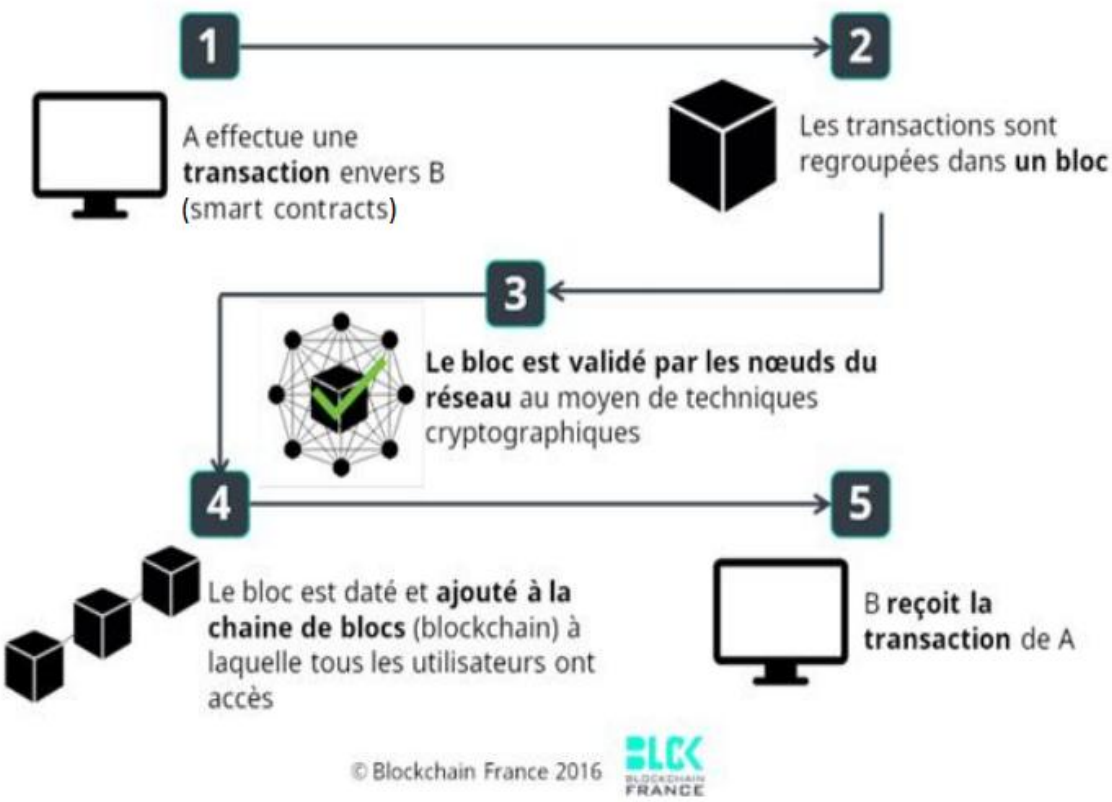
Le problème *crypto/mathématique*, propre à chaque bloc, est très difficile à résoudre. Il demande d'importantes ressources de calcul et donc de ressources financières (électricité, coût de maintenance matérielle, coût du personnel, etc.). Pour encourager les miners, essentiels au bon fonctionnement et à la viabilité du réseau, une rémunération leur est attribuée en récompense de leur travail.

La complexité du problème de validation est liée à une difficulté associée au bloc. Afin de maintenir constant, le temps de production d'un bloc, le niveau de difficulté est automatiquement ajusté par le réseau.

Extrêmement difficile à calculer, la validation de la solution est, à l'inverse, très facile. Il existe plusieurs solutions valides pour un bloc donné. Il suffit d'en trouver au moins une pour que le bloc soit validé au travers du processus de consensus.

Une fois que le bloc est validé par un « miner » et approuvé par les autres nœuds du réseau, il est horodaté et ajouté, en tête de la chaîne de blocs (dernier bloc valide créé) et tous les nœuds du réseau l'ajoutent dans leurs copies de la chaine.

La figure ci-dessous (issue du site Blockchain France) illustre le fonctionnement d'une Blockchain. Il est à noter que, par rapport à l'original, la notion de Smart Contract (étape 1, « utilisateur A») a été ajoutée.



Source : Blockchain France

IV.1.e **Concepts**

Transaction

Une transaction est un transfert de valeur (monnaie virtuelle, token, etc.) entre deux comptes, deux « portefeuilles ».

Chaque transaction est signée avec la clé privée du compte émetteur, fournissant ainsi la preuve « mathématique » qu'elle provient bien du propriétaire du compte émetteur. La signature empêche également toutes modifications de la transaction après son émission.

Une fois signée, la transaction est « placée/déployée » sur un nœud quelconque du réseau qui la diffuse à son tour, de proche en proche, sur tous les nœuds du réseau.

À présent, tout le monde sur le réseau peut utiliser la clé publique de l'émetteur pour vérifier et s'assurer que la demande de transaction provient bien du propriétaire légitime du compte.

Si la transaction est valide, elle est alors incluse, avec d'autres transactions en « attente », dans un bloc de la Blockchain, à son tour « exploité » par les mineurs.

Une fois le bloc validé, le destinataire peut voir, dans son portefeuille, le montant de la transaction.

Tout montant transféré est verrouillé sur l'adresse de réception et le montant à dépenser proviendra toujours des fonds précédemment reçus et actuellement présents dans le portefeuille.

Mining

Chaque transaction nouvellement créée donne lieu à une « écriture » dans une Blockchain. Cette écriture entre dans le réseau par un nœud du système, qui vérifie et contrôle que sa structure est correcte au regard des spécifications du protocole implémenté et qu'elle est légitime par rapport aux écritures déjà enregistrées.

Si la validation de l'écriture est satisfaite, elle est alors « mise en attente » dans une liste locale et diffusée via le réseau P2P à tous les nœuds du réseau. Dans le cas contraire, elle est rejetée.

N'importe quel utilisateur peut être un miner du réseau et tous travaillent simultanément. Ils sont libres de choisir quelles écritures ils incorporent dans leur bloc en construction. Ce bloc est ensuite complété par un en-tête, qui contient en particulier son hash et l'identifiant du bloc précédent.

Étant donné que le *mining* est (souvent) rémunéré, la validation d'un nouveau bloc conduit (mais pas forcément) à une « compétition » des *miners* entre eux. Il est alors possible, que la chaîne de blocs ait des *splits* (deux versions simultanées coexistant) qui surviennent. C'est le cas par exemple lorsque deux mineurs arrivent à deux solutions valides différentes pour le même bloc en même temps et à l'insu l'un de l'autre. Ce cas peut également se produire lors de la « mise à jour de l'ensemble des copies décentralisées » car la réplication des nouveaux blocs ne prend pas nécessairement le même temps sur l'ensemble des nœuds (latence réseau par exemple). Il alors possible qu'un des nœuds ne soit pas encore « synchronisé » alors qu'un nouveau bloc vient d'être validé et ajouté.

Le réseau est conçu pour résoudre ces *splits* et ce dans un court laps de temps, de sorte qu'une seule branche de la chaîne survit. C'est la chaine valide «la plus longue» qui est retenue. La «longueur» de la chaîne ne se réfère pas à son nombre de blocs mais à la difficulté la plus combinée. C'est une sécurité conçue pour empêcher les *forks*.

Comme les calculs nécessaires à la résolution d'un bloc sont volontairement énergivores et nécessitent de plus en plus de ressources à mesure que le réseau croit, la plupart des miners sont regroupés en coopératives. Ils forment alors une importante « puissance » de calcul et augmentent, par conséquent, leurs chances de voir leurs blocs acceptés et inclus dans la blockchain et être rémunérés.

Dès qu'un mineur reçoit un bloc d'un autre mineur, il arrête de construire le bloc en cours, qui n'a pratiquement plus aucune chance d'être accepté. Il l'élimine de sa liste d'attente locale toutes les transactions que contenait le bloc qu'il vient de recevoir, et commence à construire un nouveau bloc.

C'est de cette façon que de nouveaux blocs sont produits et diffusés en continu à travers le réseau.

Smart Contract

Un « contrat intelligent » est un programme informatique autonome dont le code contrôle et conditionne, sous certaines conditions, le transfert de devises ou d'actifs entre différentes parties.

Ils définissent non seulement les règles et les pénalités d'un accord de la même façon qu'un contrat traditionnel ou qu'un engagement transactionnel. Mais ils peuvent également, lorsque tous les engagements préalables à son exécution ont été respectés, faire appliquer automatiquement ces obligations sans qu'aucune des parties ne puissent s'y opposer.

Conçus et écrits à l'aide d'un langage de programmation haut niveau, les ***Smart Contract*** peuvent être très complexes. Leur potentiel dépasse, de fait, largement le simple transfert d'actifs. Ils permettent de simplifier et d'automatiser les processus « lourds » et répétitifs pour lesquels aujourd'hui les gens paient des avocats et des banques à des frais considérables.

Au vu de leur appellation, on a tendance à les assimiler à des contrats *classiques*. Cependant, ils n'ont pas en eux-mêmes d'autorité juridique. Un ***Smart Contract*** n'est qu'une application technique d'un contrat juridique.

DApp (Decentralized Application)

Une application décentralisée est une application connectée à et utilisant une Blockchain. Elle satisfait aux critères suivants :

- L'application doit être complètement ouverte (Open Source). Elle doit fonctionner de manière autonome et sans entité qui contrôle la majorité de ses *token*. L'application doit pouvoir adapter son protocole en réponse aux améliorations proposées au niveau de la Blockchain qu'elle utilise. Malgré tout, tous les changements doivent être décidés par consensus de ses utilisateurs.

- Les données d'application et les enregistrements d'opération doivent être stockés cryptographiquement dans une chaîne de blocs (publique ou non) décentralisée afin d'éviter tout point central d'échec.

- L'utilisation d'un jeton cryptographique (crypto-monnaie : bitcoin, ether ou un *token* natif de son système) est nécessaire pour accéder à l'application.

DAO (Decentralized Autonomous Organization)

Une DAO est une forme d'organisation, d'un fond d'investissement décentralisé et autonome *prenant place* sur une Blockchain. Ouverte à tous et ses règles étant publiques, elle ne s'appuie sur aucune juridiction et appartient aux personnes qui ont aidé à la créer et à la financer.

Une DAO est constituée de deux parties, d'une part des actionnaires, c'est-à-dire des détenteurs de *token* qui par leurs « investissements » ont aidé à la financer et d'autre part des prestataires qui souhaitent faire financer leurs projets.

Le processus d'une DAO se résume ainsi :

- Les actionnaires étudient et évaluent les projets qui leur sont soumis

- Ils débattent et décident tous ensemble par « vote » d'accepter ou de rejeter les projets qui leurs sont proposés

- Dans le cas où le projet est accepté, le prestataire doit se conformer à ses engagements et déploie/met en œuvre son service et/ou son produit

- Les avantages contractuels (monétaire ou non) négociés avec le prestataire lors de la création du contrat de financement sont fournis aux actionnaires de la DAO

- Ceux-ci peuvent décider, dans le cas où il s'agit d'argent gagné, de le réinvestir dans d'autres projets ou ils peuvent décider de se distribuer les récompenses et/ou partager les risques qui y sont relatifs.

L'intérêt de monter une DAO est de pouvoir lever une grosse somme « d'argent » afin de financer de gros projets « prometteurs » et ainsi, en cas de succès, récupérer plusieurs fois sa mise.

Consensus

Selon la définition de Wikipédia : « Un consensus caractérise l'existence parmi les membres d'un groupe d'un accord général (tacite ou manifeste), positif et unanime pouvant permettre de prendre une décision ou d'agir ensemble sans vote préalable ou délibération particulière. »

Il est parfaitement adapté pour traiter du « Problème des Généraux Byzantins ».

Dans le mode de la Blockchain, le consensus désigne le mécanisme qui consiste à garantir qu'une transaction n'est pas frauduleuse et qu'un bloc soit valide. Le seul moyen

de faire émerger une validation globale au sein du réseau, c'est d'obtenir un « vote général » entre toutes les parties prenantes. L'hypothèse faite, c'est que les utilisateurs malveillants seront toujours moins nombreux que les utilisateurs honnêtes.

Il est donc important pour la sécurité du réseau, qu'il existe des mécanismes et/ou des règles reconnus, partagés et validés par tous, qui permettent de valider une transaction puis un bloc avant qu'il soit ajouté dans la chaine de blocs.

Il existe de nombreuses méthodes d'obtention du consensus sur une blockchain. Les principales sont :

Proof Of Authority(PoA)
Une preuve d'autorité est le mécanisme de consensus d'une chaîne de blocs privée qui donne essentiellement à un utilisateur ou à un nombre spécifique d'utilisateurs, le droit de miner tous les blocs de la Blockchain.

Proof Of Stake(PoS)
Proof of Stake ou preuve d'enjeu en français est un mécanisme qui se base sur le montant de crypto-monnaie mis, à dessein, en dépôt par un utilisateur. Ce processus est appelé **minting** et on parle alors de *forgeurs*.

Le principe est le suivant :

Un certain nombre de possesseurs de crypto-monnaie mettent en dépôt une partie de leurs « avoirs » dans le cadre du mécanisme de preuve d'enjeu. Ils deviennent alors des « validateurs ».

Lorsqu'un nouveau bloc est proposé à l'ajout de la blockchain, un validateur est sélectionné, « aléatoirement » parmi tous les validateurs identifiés et se voit attribuer le droit de créer le prochain bloc et donc d'être rémunéré.

La sélection d'un validateur est pondérée en fonction du montant total de crypto-monnaie (ou token) qu'il a mis en dépôt. Ainsi par exemple, un validateur avec 10 000 « unités » aura dix fois plus de chance d'être sélectionné qu'un validateur avec 1 000 « unités ».

Si ce validateur ne crée pas le bloc dans un intervalle de temps donné, il est alors « abandonné » et un deuxième validateur est sélectionné, puis un troisième et ainsi de suite.

Proof Of work(PoW)

Proof of Work ou preuve de travail en français est un mécanisme de validation d'un bloc qui repose sur la résolution « d'un problème crypto-mathématique complexe ». Le processus de résolution est appelé **mining** et on parle de *miners*.

Comme expliqué, il existe une grande diversité de méthodes, chacune avec ses avantages et ses inconvénients, son niveau de maturité et sa qualité d'implémentation. Leur compréhension est « primordiale » car elles sont une composante essentielle au bon fonctionnement du réseau.

Citons en encore quelques une :

- « Pratical Bizantine Fault tolerance » (PBFT)
- « Proof of Hold » (PoH)
- « Proof of Use » (PoU)

- « Proof of Stake/Time » (PoST)
- « Proof of Minimum Aged Stake » (PoMAS)

V - CAS D'USAGES

V.1 INTRODUCTION

Les sociétés *FinTech* furent les premières à s'intéresser à la Blockchain et à l'utiliser pour remplacer et révolutionner les processus et solutions des secteurs bancaire, financier et assurantiel. Défaite de son étiquette *FinTech* son utilisation est envisagée et parfois mis en œuvre dans des domaines d'activités extrêmement variés : *supply chain*, IoT (pour la création de nouveaux *business model* s autour du *Sensing as a Service)* ou encore pour l'achat et la vente d'énergie.

Afin d'illustrer les possibilités et les avantages offerts par la technologie Blockchain, deux cas d'usage un peu particuliers (originaux) sont étudiés.

V.2 ÉCHANGE D'ENERGIE PAIR-A-PAIR

V.2.a SolarCoin

Lancée en janvier 2014, portée et pilotée par la fondation du même nom, **SolarCoin** est une blockchain publique utilisant une version simplifiée du protocole **BitCoin.**

S'inscrivant dans une démarche écologique de protection environnementale, son objectif est de mettre en place un dispositif incitant à la production d'électricité solaire au niveau mondial. Afin d'encourager et de motiver l'adhésion des producteurs autonomes d'énergie solaire, une récompense leur est attribuée par l'octroi de tokens spécifiques appelées solarcoins. Chaque solarcoin en circulation représente la production d'1 MWh (mégawatt-heure) d'électricité solaire et possède une valeur fiduciaire. Gratuite et publique, l'acquisition de solarcoins est ouverte à tout le monde et ne nécessite aucun frais d'adhésion.

À terme, le but de SolarCoin est de créer une place de marché permettant aux citoyens, entre eux, d'acheter et vendre de l'électricité produite par leurs installations photovoltaïques. Elle compte sur les réseaux « Smart Grid » pour développer et augmenter son utilisation.

L'inscription à SolarCoin suit un processus itératif très sécurisé pour s'assurer de l'authenticité et de la viabilité de ses utilisateurs.

Dans ce contexte, la blockchain apporte des avantages significatifs :

- Création d'une « marketpace » décentralisée et désintermédiée d'achat et vente d'électricité solaire autoproduite par des particuliers autonomes
- Historiser, tracer et fiabiliser (Proof of Stake Time) l'achat et la vente d'électricité solaire
- Réappropriation par les citoyens de leur production énergétique solaire
- La fixation du prix de l'énergie n'est plus fixée par une agence de tutelle mais par le marché lui-même.

V.3 SUPPLY CHAIN

V.3.a Everledger

Everledger, une société londonienne fondée en 2015, s'appuie sur la blockchain **HyperLedger Fabric** pour sa solution de traçabilité, de suivi et d'authentification de produits (alimentaires, etc.) et de marchandises.

Elle a développé un réseau pour la certification et le suivi de diamants, capable de prouver leurs provenances et d'identifier leurs propriétaires.

40 attributs par diamant (taille, pureté, lieu d'extraction…) sont utilisés pour la génération d'un numéro de série gravé microscopiquement sur le diamant et ajouté à la blockchain. Toutes les informations diamantaires sont répliquées sur les différents nœuds de la blockchain, protégeant ainsi les données contre toute attaque et toutes falsifications.

En réalité, deux blockchains sont utilisées pour sa mise en œuvre :

- Pour assurer que les informations, à caractère « privé », échangées et partagées entre les négociants en diamants et les acquéreurs ne soient pas diffusées à tout le monde, elles sont stockées et gérées dans une blockchain privée (**HyperLedger Fabric**).
- La blockchain Bitcoin est utilisée quant à elle pour le suivi (historisation, horodatage, immutabilité) des informations privées.

Là encore, l'utilisation de la blockchain apporte des avantages significatifs :

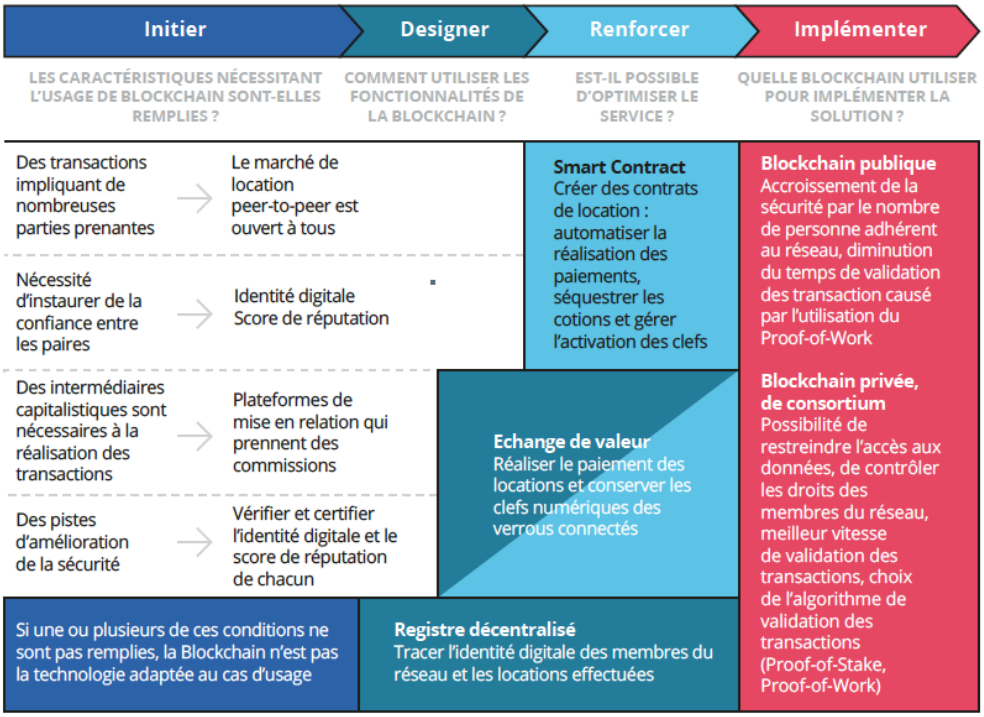
- Traçabilité infalsifiable et immutable des intervenants, du circuit et des données des diamants permettant de lutter, efficacement, contre la fraude et le vol.
- Automatisation et autonomie des paiements grâce à la mise en place de *Smart Contract*.
- Sociale : Contrôle des bonnes conditions de travail des sous-traitants en relevant les incohérences entre le nombre de travailleurs déclarés et l'historique des volumes de production
- Fluidifier les échanges et les traitements internationaux et douaniers

Dans un autre registre, en Chine, **Walmart** teste la technologie blockchain **HyperLedger Fabric** pour améliorer et assurer la sécurité alimentaire de consommation de viande de porc. Chaque morceau ainsi que le lieu et la façon dont il a été transformé, sa température de stockage et sa date limite de conservation sont enregistrés dans la blockchain offrant ainsi une « solution » de traçabilité et de certification de la « qualité » de la viande.

V.4 CONCLUSION

Représentant un investissement non négligeable, la Blockchain n'est pas une solution miracle et n'est pas nécessairement applicable (« rentable ») à tous les types de cas d'usages.

Afin de valider la pertinence de sa mise en place, le cabinet Deloitte a développé une matrice de décision composée de plusieurs conditions et de plusieurs questionnements auxquels il faut répondre :



VI - PANORAMA DES BLOCKCHAINS OPEN SOURCE

VI.1 ETHEREUM

VI.1.a Présentation

Ethereum (portée par la fondation du même nom) est une chaine de blocs créée par un jeune programmeur russo-canadien, **Vitalik BUTERIN**.

En janvier 2014, il publie le White Paper « ***A Next-Generation Smart Contract and Decentralized Application Platform*** », annonçant officiellement le projet **Ethereum**.

En avril de la même année Dr. Gavin WOOD (co-créateur d'Ethereum et co-fondateur d'ETH DEV) publie le document de référence : Les Spécifications formelles du protocole **Ethereum**.

À partir de là, différentes implémentations voient le jour et un an plus tard, en juillet 2015, la première version d'**Ethereum**, **Frontier**, est lancée et mis à disposition du « public ».

L'originalité principale de la blockchain **Ethereum** est qu'elle est programmable, c'est-à-dire qu'elle est capable « d'héberger » et « d'instancier » des programmes autonomes de code informatique, les fameux *Smart Contract*. Plutôt que de donner aux utilisateurs un ensemble d'opérations prédéfinies, comme le sont par exemple les transactions ***bitcoin***, **Ethereum** permet à tous les utilisateurs de créer leurs propres « applications », quels qu'en soit la complexité et des DApp.

Le *jeton d'échange* d'**Ethereum** s'appelle l'**Ether.** C'est la monnaie virtuelle utilisée pour le paiement des transactions et des opérations associées. Elle sert également à récompenser les *mineurs*.

L'objectif d'**Ethereum** est de « décentraliser le WEB » et dans le but de gagner en popularité et toucher le plus grand nombre, différentes étapes sont incluses dans sa feuille de route:

Frontier

- Frontier, lancée en juillet 2015, a été la version initiale du réseau **Ethereum**. Version *Bare Bone* c'était une plateforme « d'expérimentation » dédiée aux **développeurs** pour la création et les tests d'applications décentralisées, le minage et l'échange de transactions.

Homestead

- Version actuelle, c'est un *patch work* & *bug fixes*. Elle s'était fixée l'objectif de créer une première implémentation officielle du protocole en GO. Cela a **geth**. Elle fut également marquée par l'introduction de frais par block minés fixés à 5 **ether**.

Metropolis

- La version **Metropolis** (date de sortie inconnue) sera la version ouverte « aux masses ». Son objectif est de permettre aux personnes non techniques d'utiliser la blockchain **Ethereum**.

Serenity

- **Serenity** est la dernière version **Ethereum** prévue. Sa principale caractéristique est la mise en place de la *POS* (*Proof Of Stake*) en remplacement de la *POW* (***P**roof **O**f **W**ork*) pour le mécanisme de consensus.

VI.1.b Protocole

Comptes

Tout utilisateur de la blockchain **Ethereum** est défini et identifié par un compte. Deux types de comptes existent :

- Les comptes utilisateurs «classiques »
- Les comptes « ***Smart Contract*** »

Les comptes, stockés dans un *keystore*, sont identifiés et référencés au travers d'un « champ » *address*. Cette adresse est générée à partir de la clé publique du couple clé privée/clé publique créé lors de la création du compte. Ces adresses sont nécessaires et « obligatoires » pour la mise en œuvre des transactions car elles permettent d'identifier l'émetteur et le destinataire de la transaction. Chaque compte a un champ *Balance* qui représente la somme d'***ether*** qu'il possède et l'association d'une *adresse* et d'une *balance* constitue un *wallet* (portefeuille).

La valeur de la *balance* détenue par un compte est importante car elle permet d'acheter du ***gas***, utilisé pour « payer l'exécution des ***Smart Contract*** ».

« Gas » est le nom d'une unité spéciale utilisée dans **Ethereum**. Elle mesure combien une action ou un ensemble d'opérations consomme comme « ressources de travail » pour son exécution. Toutes les opérations effectuées par une transaction ou un contrat coûtent un certain nombre de gaz et plus elles nécessitent des ressources informatiques plus elles coûtent de *gas*.

En exigeant qu'une transaction (ou un contrat) paye des frais appropriés pour chaque opération qu'elle effectue, « **Ethereum** » veille à ce que le réseau ne soit pas « encombré » par des travaux intensifs inutiles (utiles à personne) et maintient ainsi une certaine « qualité » du réseau.

Il s'agit d'une stratégie différente de celle des frais de transaction ***Bitcoin***, qui repose uniquement sur la taille en kilo-octets d'une transaction. Étant donné qu'**Ethereum** permet d'exécuter du code informatique arbitrairement complexe (***Smart Contract***), un petit programme (quelques lignes de code) peut effectivement entraîner beaucoup de travaux de calcul. Il est donc important de mesurer le travail effectué directement au lieu de simplement choisir un tarif en fonction de la taille d'une transaction ou d'un contrat.

Bien que le *gas* soit une unité dans laquelle les choses peuvent être mesurées, il n'y a pas de *token* (jeton) réel pour le *gas*. C'est-à-dire que vous ne pouvez pas posséder

1000 gas. Le *gas* n'existe que dans la machine virtuelle **Ethereum** (EVM) en tant que compte de la quantité de travail effectué.

En ce qui concerne le paiement effectif du gas, les frais de transaction sont facturés en un certain nombre d'éther, calculé en fonction du nombre de *gas* consommé et du prix en ***Ether*** d'une unité de *gas*. La raison pour laquelle les opérations « ne comportent » pas directement un coût mesuré en ***Ether*** c'est que, comme les bitcoins, ***l'Ether*** a un prix de marché (son court) qui peut changer rapidement. Comme le coût de calcul de travail n'est pas défini (monte, baisse) en fonction des variations du prix de l'***Ether***, le *gas* sert à séparer le prix du *travail*, de sa valorisation.

Nœud

Comme toute blockchain, le « réseau » **Ethereum** est composé de plusieurs nœuds. Un nœud est une machine participant au réseau P2P et maintient la viabilité de la blockchain **Ethereum**.

Il existe trois grands types de nœud dans **Ethereum** :

Full node

- Nœuds contenant une copie intégrale de la blockchain **Ethereum** depuis sa création. Plusieurs niveaux de détail sont possibles suivant le niveau de complétude souhaité, pour l'historique des différents *états* des transactions et des *Smart Contract*.

Light node

- Nœuds contenant « uniquement » des comptes, capables de créer des transactions et des *Smart Contract*.

Miner node

- Nœud *full* capable de construire des blocs et de les valider.

L'association d'un nœud avec un autre nœud forme un ***peer***.

L'initialisation d'un nœud **Ethereum** s'effectue via un fichier spécifique nommé ***genesis***. Utilisant la grammaire ***JSON***, c'est un fichier qui permet de définir les paramètres et les caractéristiques du **premier bloc** de la Blockchain. C'est dans ce fichier qu'est spécifiée, par exemple la difficulté de validation d'un bloc. Il permet également, dans le cas d'une Blockchain privée/test de pré-alloué une somme d'**ether** pour un compte donné.

Deux nœuds font partis de la même blockchain **Ethereum** si, et seulement si, ils ont le même bloc ***genesis***.

Transaction

Une transaction dans **Ethereum** se réfère à un paquet de données signées qui stocke un « message » à envoyer d'un compte à un autre compte.

Les transactions contiennent :

- Le destinataire du message,
- Une signature identifiant l'expéditeur et prouvant son intention,
- Le montant de ***wei*** à transférer de l'expéditeur au destinataire (le ***wei*** est la plus petite unité, 1 ether équivaut à 10^18 wei),
- Un champ de données personnalisable facultatif, qui peut contenir par exemple des données envoyées à un contrat,

- Une valeur STARTGAS, représentant le nombre maximal d'étapes de calcul que l'exécution de la transaction est autorisée à utiliser (évite les *infinite loops*),
- Une valeur GASPRICE, représentant les frais que l'expéditeur est prêt à payer pour le **gas**.

Un message est une transaction émise par un *Smart Contract* lors de l'appel d'une fonction/méthode d'un autre *Smart Contract*.

Les messages contiennent :

- L'expéditeur du message (implicite).
- Le destinataire du message
- Le montant de ***wei*** à transférer à l'adresse du contrat,
- Un champ de données optionnel. Il s'agit des données d'entrée réelles du contrat
- Une valeur STARTGAS, qui limite la quantité maximale de gaz que peut entraîner l'exécution du code déclenchée par le message.

Mining

Ethereum utilise un modèle de sécurité incitatif. Ce modèle appelé **Consensus** est basé sur la sélection, l'élection et l'inscription, « communes », d'un bloc validé à la tête de la blockchain. C'est le bloc avec la plus grande difficulté totale qui est mis en compétition.

Le *minage* est basé sur le calcul d'une « preuve de travail » ou POW (Proof Of Work) dont la complexité de résolution est liée à une *difficulté*.

Une caractéristique de ce système est l'asymétrie du calcul : le travail doit être difficile mais réalisable pour le demandeur, et facile à vérifier pour un tiers.

L'algorithme de preuve de travail utilisé s'appelle **Ethash** et implique de trouver un *nonce* (nombre arbitraire) dont le résultat algorithmique (calcul *cryptographique*) est inférieur à un certain seuil calculé en fonction de la difficulté. La particularité des algorithmes POW c'est qu'il n'y a pas de meilleure stratégie pour trouver ce nombre que d'énumérer toutes les possibilités (*brut force*) alors que la vérification d'un *nonce* est banale et peu coûteuse. Si la génération de nouveaux blocs à une distribution uniforme, nous pouvons garantir que, en moyenne, le temps nécessaire pour trouver un *nonce* dépend du seuil de difficulté. Cela permet de contrôler la fréquence d'ajout des blocs simplement en manipulant une *difficulté*.

La difficulté s'ajuste dynamiquement de manière à ce que, en moyenne, un bloc soit produit par l'ensemble du réseau toutes les 12 secondes.

L'exécution du minage entraine la génération et l'utilisation d'un fichier DAG (graphique acyclique dirigé) pour la résolution de la preuve de travail. Un nouveau fichier est généré à chaque nouvelle « *époque* », c'est-à-dire tous les 30000 blocs (environ toutes les 100 heures).

Smart Contract

Les *Smart Contract* sont des programmes de code informatique permettant de matérialiser n'importe quel type de processus, n'importe quelles règles métier.

Trois langages de programmation sont disponibles pour écrire des ***Smart Contracts*** **Ethereum** :

Solidity

- Similaire à JavaScript, ***Solidity*** est le langage de référence pour la création des ***Smart Contract*** et c'est aujourd'hui le langage le plus utilisé et le plus populaire.

Serpent

- ***Serpent*** est un langage similaire à ***Pyhton*** et a été conçu pour être simple et facile d'utilisation.

LLL

- **Lisp Like Language** est un langage de bas niveau similaire à l'***Assembleur***, qui permet de coder directement en *bytecode* EVM.

Une fois compilé, un ***Smart Contract*** est déployé dans **Ethereum** « via » une transaction qui doit être incluse dans un bloc validé pour qu'il soit fonctionnel. Une fois déployé il expose, au travers d'une ABI (**A**pplication **B**inary **I**nterface), les méthodes et paramètres qui permettent d'interagir avec lui.

Le code compilé des ***Smart Contract*** est exécuté au sein même de la blockchain Ethereum par EVM (**E**thereum **V**irtual **M**achine). C'est un environnement d'exécution complètement isolé, « interdisant » au code des ***Smart Contract*** l'accès à des ressources systèmes ou techniques : réseau, système de fichiers, processus, etc.

L'utilisation de données et/ou de services externes (c'est-à-dire accessibles en dehors **d'Ethereum**) comme données en entré d'un *Smart Contract* (remontée d'une station météo, état d'un verrou électronique par exemple) n'est possible qu'au travers de services spécifiques appelés ***Oracles***.

VI.1.c Solutions

Ethereum dispose d'une forte dynamique de développement/d'évolutions et d'une communauté d'utilisateurs et d'acteurs importante. Très riche, son écosystème compte aujourd'hui de nombreuses solutions et nous avons fait le choix de s'attarder sur trois d'entre elles.

Librairies Java, à l'exception de la première solution, elles abordent trois aspects, trois approches différentes de la programmation **Ethereum** : une CLI, un SDK et une librairie cliente. Les deux premières solutions correspondent à des implémentations de référence du protocole **Ethereum**, respectivement en Go et en Java (d'autres existent en Python et C++) et la troisième et dernière solution est une implémentation complète, en Java et Android, de l'API client JSON-RPC. Elle a été conçue comme un facilitateur d'applications interagissant avec un (ou plusieurs) nœuds **Ethereum**.

Geth

geth est l'implémentation *full* officielle d'**Ethereum.** Écrit en langage ***Go***, c'est une solution qui se présente sous la forme d'un utilitaire en ligne de commande disponible sous Linux, Mac Os et Windows.

Deux modèles sont définis au niveau du mode de licence, le protocole Ethereum (*The Ethereum Core Protocol*) est sous licence *GNU Lesser General Public License* et tous les

outils *Frontend* (ensemble des utilitaires de la « suite » **geth**) sont sous licence *GNU General Public License*.

Il est possible d'interagir avec **geth** de trois façons différentes :

- Via une console Javascript interactive qui permet d'administrer et gérer un nœud **Ethereum**.
- Via un serveur JSON-RPC exposant, au travers de l'instanciation de **geth**, l'API Ethereum JSON-RPC
- Via une CLI (Command Line Interface).

L'installation de **geth** est simple et ne nécessite aucun paramétrage ni aucune configuration. Prêt à l'emploi, une fois l'installation effectuée, il suffit « juste » d'ouvrir un terminal en ligne de commande pour créer et gérer un nœud *full* **Ethereum**.

geth est capable de gérer les trois types de nœuds suivants :

full node
C'est un nœud qui contient, en local, une copie complète de la blockchain et qui permet d'utiliser toutes les fonctionnalités d'**Ethereum**. Les miners sont des nœuds complets.

archive node
C'est un nœud qui maintient tous les historiques de la blockchain Ethereum (ses états, ses transactions, etc.)

light node
C'est un nœud qui ne contient pas de copie de la blockchain et qui permet de vérifier l'exécution d'une transaction et de maintenir la cohérence de l'état

Dans **geth**, la création d'un compte nécessite la saisie d'un mot de passe. Deux modes sont proposés pour renseigner ce mot de passe : un mode interactif dans lequel le mot de passe est saisi manuellement et un mode non-interactif dans lequel le mot de passe est passé au travers d'un fichier. Pour des raisons évidentes de sécurité, le mode non-interactif est à réserver dans le cadre de Blockchain de tests ou dans des environnements sécurisés connus.

Tout compte nouvellement créé est, par défaut, *locké* et ne peut être « activé » qu'en utilisant le mot de passe utilisé lors de sa création. Cette sécurité permet de s'assurer que seul le détenteur du compte peut l'utiliser.

Le mot de passe est à retenir car c'est la seule donnée d'un compte qui ne peut être récupérée. Il n'y a pas, en effet, d'option « Oubli de mot de passe ».

Il est possible, par une simple copie des fichiers de comptes, de transférer (copier) les comptes d'un nœud à un autre nœud et/ou d'une implémentation à une autre (C++, Go, Python). De fait, Il est fortement recommandé de sauvegardés régulièrement ces fichiers.

geth permet de gérer le « cycle de vie » d'une transaction (exception faite de la suppression/destruction qui n'est pas possible sur une Blockchain) et propose, pour ce faire, plusieurs fonctionnalités :

- La création et l'envoi d'une transaction
- La récupération du nombre de transactions envoyées par un compte
- La récupération du nombre de transactions contenues dans un bloc

- La récupération des informations d'une transaction (soit par son *hash*, soit par son index de position dans un bloc donné)
- La récupération de l'état « retour » d'une transaction

Deux types de transactions sont supportés par **geth** et dépendent du type de compte à l'origine de la transaction :

- Transactions « simples » (exemple : transfert d'un montant d'***ether***)
- Transactions Smart Contract

Mining

Contrairement à d'autres opérations comme la synchronisation des nœuds entre eux, le *mining* n'est pas une opération automatique et doit être lancée manuellement.

Comme les *miner* sont rémunérés pour leur travail, il est nécessaire, pour que le processus de minage puisse s'exécuter, de définir le compte utilisateur *miner* afin de pouvoir lui reverser, une fois le minage terminé, la récompense attendue . Dans le langage **geth**, cela revient à définir ce que l'on appelle un ***etherbase*** (ou ***coinbase***) qui n'est qu'autre que l'adresse d'un compte.
Afin de réduire le temps de calcul du processus de minage, il est possible de définir le nombre de threads (lancés en parallèle) participant à la résolution de la preuve de travail (POW) du futur bloc à valider.

Le lancement du *mining* déclenche la création et la génération du fichier DAG pour l'époque en cours (tous les 30000 blocs, une nouvelle époque est créée) et ce n'est que lorsque que cette étape est terminée que le processus de *mining* démarre réellement.

Il est à noter que dans la version testée de **geth** (version 1.6), seul le minage CPU est possible. Il est prévu, dans les futures versions, de pouvoir utiliser le GPU pour miner.

Smart Contract

Tout comme les programmes écrits dans un langage de programmation « haut-niveau », les ***Smart Contract*** doivent être compilés pour pouvoir être déployés dans **Ethereum**.

En standard, **geth** ne dispose pas de compilateur intégré, il est donc nécessaire d'en installer un pour pouvoir déployer un ***Smart Contract*** depuis **geth**.

Aujourd'hui, le compilateur le plus utilisé se nomme **solc** et permet de compiler un ***Smart Contract***, écrit en langage ***Solidity***, en *bytecode* EVM qui est, pour rappel, le moteur d'exécution d'**Ethereum**.

La compilation d'un ***Smart Contract*** se « décompose en deux étapes » :

- Formatage de la mise en forme du code pour qu'il tienne sur une seule ligne
- Lancement de la compilation

Ethereumj

Ethereumj est une implémentation *full*, en pure Java, du protocole **Ethereum,** c'est-à-dire que tous les composants technologiques tels que le mining ou la POW sont écrits et implémentés en java. La version Homestead d'Ethereum est supportée.

Sous licence LGPL-V3, **Ethereumj** peut être utilisé sous plusieurs formes :

- En tant qu'application autonome : un nœud effectif *full* **Ethereum**
- Une librairie pouvant être intégrée dans n'importe quel projet Java/ Scala pour fournir un support complet de la blockchain **Ethereum**.
- Un SDK. Il est possible d'utiliser **Ethereumj** pour créer différents types d'applications dédiées à la blockchain **Ethereum** : outils d'administration, explorateur de blocs, DApp, etc.
- L'introspection et l'analyse du code, permettent d'une part de mettre en œuvre des fonctionnalités avancées telles que la génération *just in time* de la configuration complète d'un nœud *full* (paramètres de configuration système et fichier genesis). Et d'autre part une gestion très fine des arcanes d'**Ethereum**.

Initialement développé par Roman Mandeleil, il est désormais sous parrainage d'Ether Camp.

Ethereumj dispose d'un système de configuration du protocole **Ethereum** très poussé. Celui-ci se présente sous la forme d'un fichier clair et structuré. Chaque option est référencée par un *path*, possède un type (entier, booléen,chaine de caractère, etc.) et une valeur associée. A titre d'exemple voici le *path* qui permet de définir le port d'écoute d'un peer : *peer.listen.port*.

Les clés privées et publiques d'un compte sont créées au travers de l'implémentation de l'algorithme ECDSA (*Elliptic Curve Digital Signature Algorithm*). Un compte contient une adresse, une balance et la liste de toutes ses transactions en attente.

Une transaction est un simple objet java composé de tous les paramètres attendus :

- Le hash de la transaction
- Le nonce
- La valeur à transmettre
- L'adresse du compte émetteur et du compte destinataire
- La valorisation, en **ether**, d'un *gas*
- La quantité maximale de *gas* pouvant être utilisé pour l'exécution de la transaction
- Un champ optionnel permettant de stocker des données
- La signature du compte utilisé pour la création de la transaction

Les objets et données d'une transaction sont encodés en RLP (Recursive Length Prefix) qui est le format utilisé dans **Ethereum** pour sérialiser les objets.

Mining

Au niveau du mining, Ethereumj prend en charge l'exploitation du CPU et supporte deux modes :

Un mode *light*

- Petit fichier DAG, pour une génération en quelques minutes. Par contre, les calculs seront plus « lents ».

Un mode *full*

- Gros fichier de quelques gigas, dont la génération peut prendre plusieurs heures. À l'inverse, les calculs sont très rapides.

Smart Contract

L'utilisation et la mise en œuvre d'un *Smart Contract* peut se réaliser de deux façons :

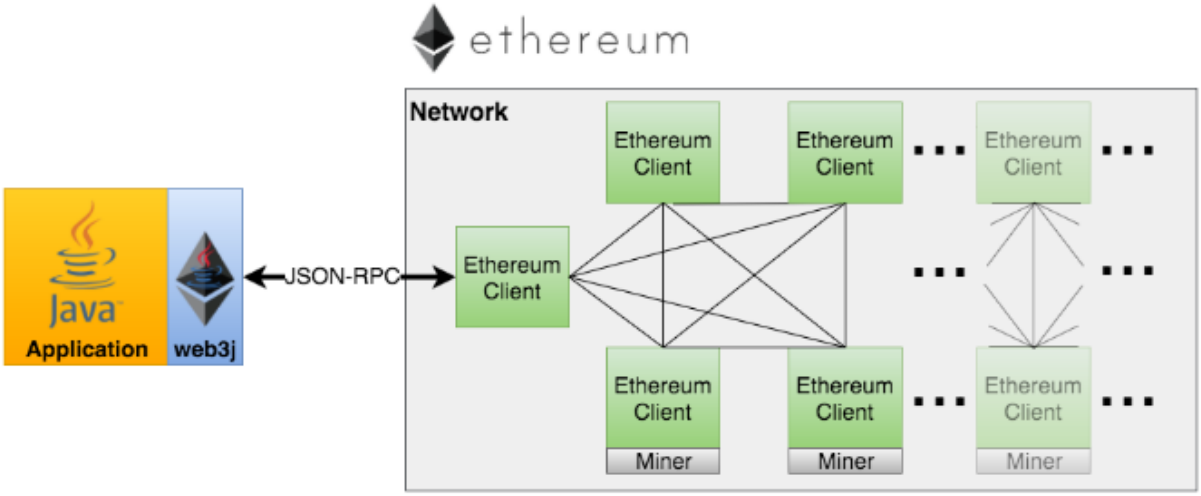
- Soit en mode externe, dans lequel le *Smart Contract* est compilé via un outil externe comme le compilateur **solc** par exemple. Une fois compilé, le code du *Smart Contract* peut être chargé.
- Soit en mode interne. **Ethreumj** dispose en effet d'un package de classes dédié à la compilation et à la gestion d'un *Smart Contract*. C'est le langage **Solidty** qui est supporté et la création d'un *Smart Contract* se résume à la création d'une chaine de caractère contenant le code complet du « *contrat* ».

C'est un SDK très puissant mais qui demande une phase d'acquisition relativement importante et une très bonne maitrise du protocole **Ethereum**, pour en exploiter toute la quintessence.

web3j

web3j est une librairie légère écrite en java et compatible Android, qui permet d'écrire des applications qui manipulent et interagissent avec un nœud (ou des nœuds) **Ethereum** et ce sans avoir à écrire la glue nécessaire à l'interfaçage avec le protocole **Ethereum**.

Développé par Conor Svensson, sous licence *Apache License 2.0*, **web3j** est une implémentation complète de l'API *client* Ethereum's JSON-RPC et peut être utilisée via http ou IPC.



Architecture de web3j
Source : *https://github.com/web3j/web3j*

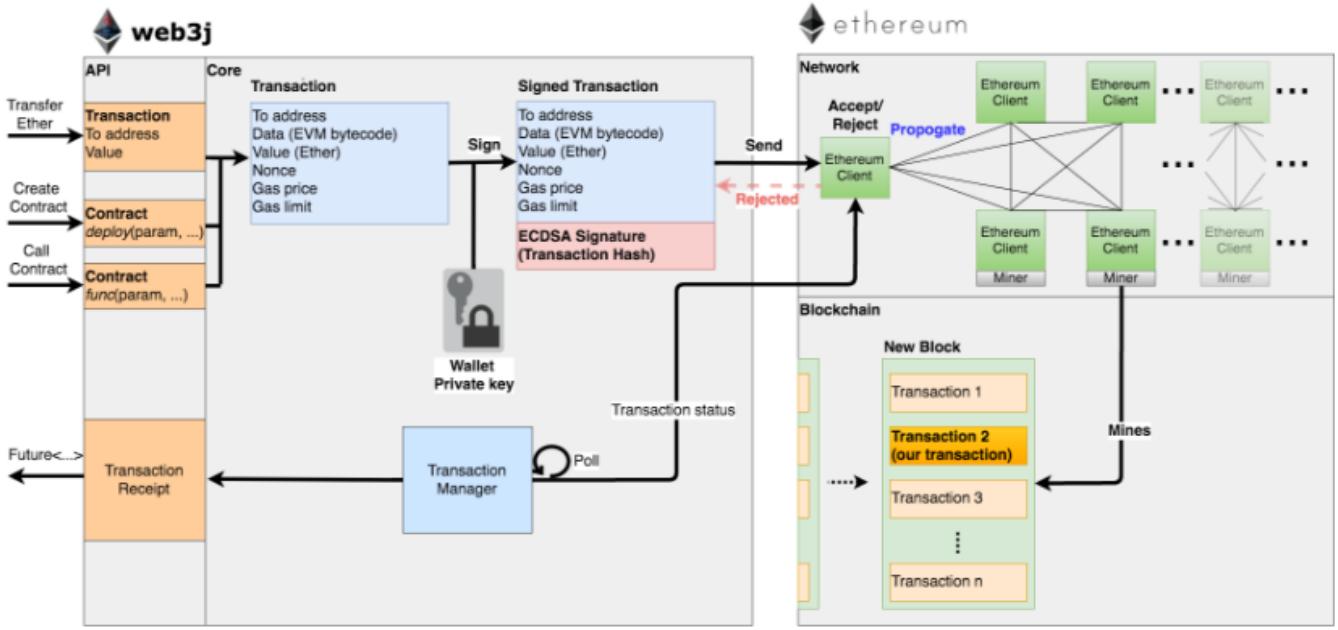
Les principales fonctionnalités de web3j sont :

- Support d'**Ethereum Wallet**, une application Open Source en ligne de gestion de portefeuille **Ethereum**
- Support de la programmation *reactive* et *fonctionnelle*
- Génération automatique de *wrappers* pour la création, le déploiement et l'utilisation d'un *Smart Contract* depuis un code java.
- Support des API clientes **geth** et **Parity**
- Support d'**INFURA**. **INFURA** est une infrastructure de nœuds **Ethereum** et **IPFS** (InterPlanetary File System) sécurisée, stable, robuste, équilibrée et tolérante aux pannes.
- Gestion de filtres, pour le *monitoring* et le *tracking* des évènements.

La création et la gestion de comptes et de portefeuilles sont supportées. Les clés privées et publiques d'un compte sont générées via **SECP-256k1 curve**. Il est possible, avec **web3j**, d'activer un compte *locké* et de l'utiliser pour signer des transactions.

web3j supporte trois types de transaction :

- Transfert d'**ether** d'un compte à un autre
- Création et instanciation d'un *Smart Contract*
- Exécution d'un *Smart Contract*



Source : *https://docs.web3j.io/transactions.html*

Il est possible de créer et de déployer un *Smart Contract* avec **web3j**. N'embarquant pas de compilateur ***Solidity*** ou autres, il faut donc utiliser un compilateur externe pour compiler le code d'un *Smart Contract*.

Une fois compilé, l'import d'un *Smart Contract* dans **web3j** passe par le chargement de son code compilé et son exécution par le lancement des *opcodes* EVM générées. Ce code compilé se présente sous la forme d'une chaine de caractère composée d'une suite d'*opcodes*.

web3j propose également une API asynchrone de gestion des évènements **Ethereum**. Elle permet de créer des filtres pour la notification d'événements sélectionnés.

Trois types de filtres sont gérés :

- Block filters
- Pending transaction filters
- Topic filters

Les filtres *Block Filters* et *Pending transaction filters* permettent d'être notifié lors de la création d'une nouvelle transaction ou d'un nouveau bloc. Les *Topic Filters* permettent de définir et de créer des filtres personnalisés.

VI.2 HYPERLEDGER

VI.2.a Présentation

Annoncé en décembre 2015 et chapeauté par la Fondation Linux, **Hyperledger** est une initiative (un effort) collaborative open source qui vise, au travers de la promotion de la

Blockchain et de la collaboration inter-industrielle de ses membres, à établir un standard technologique de *registres décentralisés*, adapté aux besoins de tous les secteurs d'activité.

Fort de ses promesses et de sa popularité grandissante, **Hyperledger** compte plus de 130 membres actifs tous secteurs confondus (finance, banque, IOT, logistique, manufacture) et Brian Behlendorf, fondateur d'ASF (Apache Software Foundation), en est le directeur exécutif.

Projet très actif, l'écosystème **Hyperledger** se compose, aujourd'hui, de plusieurs solutions, de plusieurs « framework orienté blockchain » :

- Hyperledger Burrow :

Hyperledger Burrow est un client blockchain incluant un « interpréteur » de contrats intelligents, en partie construit à partir des spécifications EVM (**E**threum **V**irtual **M**achine : environnement d'exécution des *Smart Contract*)

- Hyperledger Fabric

Hyperledger Fabric est une brique logicielle d'implémentation du protocole DLT (Distributed Legder Technology), servant de ***base*** pour la mise en œuvre de *plateforme* blockchain et le développement d'applications (Smart Contracts).

- Hyperledger Iroha

Hyperledger Iroha basée sur **Hyperledger Fabric** , est une blockchain orientée applications mobiles.

- Hyperledger Sawtooth

Réécrit depuis zéro, **Hyperledger Sawtooth** est une blockchain modulaire conçue pour être polyvalente et évolutive. Sa principale caractéristique est la mise en œuvre de la preuve de temps écoulé (PoET : Proof of Elapsed Time) pour l'algorithme de **Consensus**.

La « brique logicielle » qui nous intéresse est **Hyperledger Fabric** puisque c'est elle qui permet de mettre en place une blockchain *Hyperledger*. Contrairement aux Blockchains publiques traditionnelles, **Hyperledger Fabric** est une *permissioned blockchain*, c'est-à-dire qu'elle fonctionne sur un réseau privé et que pour y participer, un utilisateur doit forcément s'être inscrit auprès d'un *service d'adhésion (le service Membership)*.

Ce service a une double fonction :

- Identification que l'utilisateur est bien connu et authentification forte que l'utilisateur est bien celui qu'il prétend
- Définition de droits pour un utilisateur, afin de limiter et contrôler ses interactions.

Une des particularités d'**Hyperledger Fabric**, c'est sa modularité, rendue possible grâce à ses composants *plug-and-play*.

À titre d'exemple :

- Toutes les données inscrites dans la blockchain peuvent être stockées sous différents formats
- Le mécanisme de **Consensus** peut-être changé
- Plusieurs fournisseurs de *service d'adhésion* sont supportés

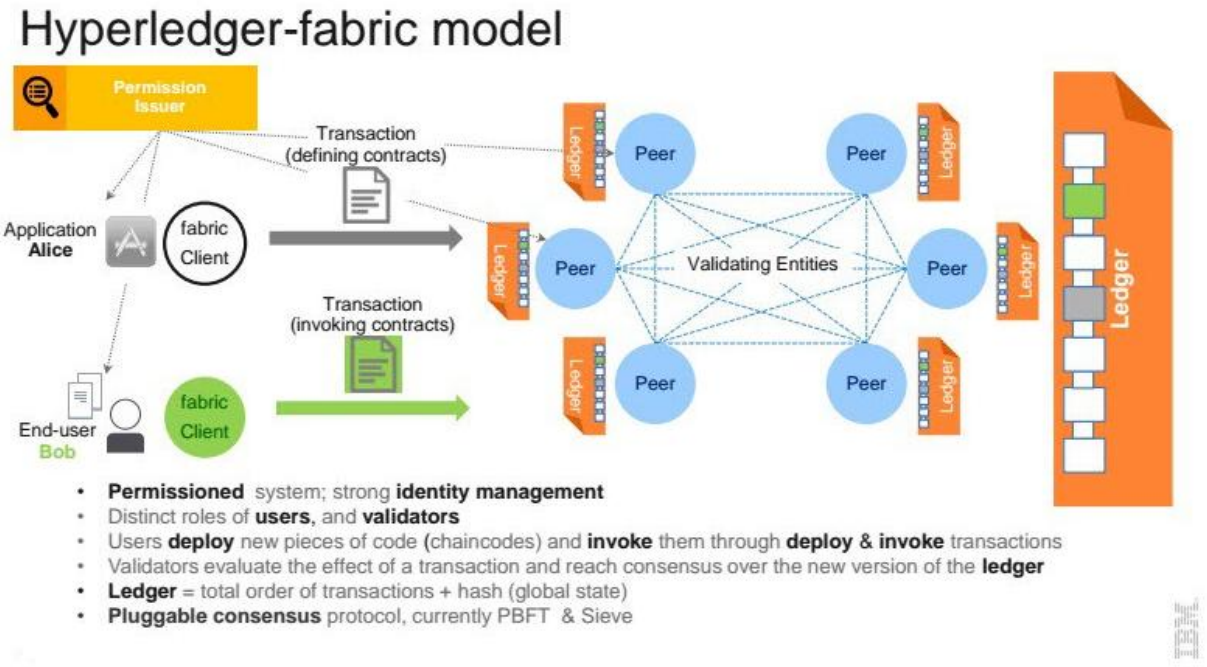
Hyperledger Fabric offre, également, la possibilité de créer des *channels* qui permettent à un groupe de participants de créer des transactions privées, c'est-à-dire non visibles des autres participants. C'est une option particulièrement importante pour les réseaux où certains participants pourraient être concurrents et ne veulent pas que les transactions

qu'ils font soient connues de tout le monde, comme par exemple un prix spécial offert à certains participants et pas à d'autres.

Il n'y a pas de crypto-monnaie sur **Hyperledger Fabric** et il n'est pas prévu d'en créer une. Du coup, à travers la possibilité de définir son propre type d'actif (*token*), il est possible « d'échanger presque n'importe quoi » sur le réseau : aliments, voitures de collection, contrats à terme sur devises, etc.

Tous les composants (les différents types de nœuds et le service d'autorité) se présentent sous la forme de conteneurs Docker.

Enfin, **Hyperledger Fabric** est distribué sous licence « Apache License Version 2.0 Software License »



Source : *https://www.linkedin.com/pulse/ibm-blockchain-in-depth-understanding-hyperledger-open-vinit-bansal*

VI.2.b **Protocole**

Compte

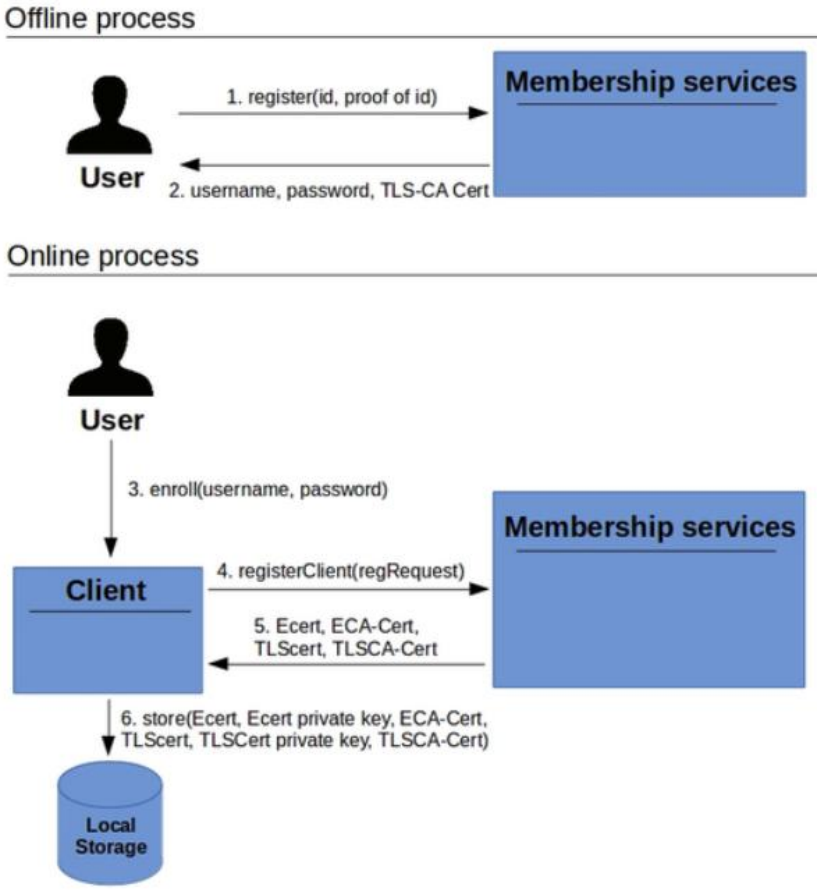
Hyperledger Fabric étant un réseau privé, il faut, pour qu'un utilisateur puisse s'y connecter et l'utiliser, qu'il s'inscrive en tant que participant.

Cette inscription est effectuée via le service PKI (Public Key Infrastructure) d'authentification et d'autorisations de **Fabric** : *Membership service*. Cette inscription n'est à faire qu'une seule fois et pour que l'inscription soit validée, il faut que l'utilisateur fournisse une « preuve d'identité » forte (*Proof of ID*).

Un compte est identifié par le couple : nom utilisateur/mot de passe et par un certificat lorsqu'il est instancié.

Il est possible d'associer des rôles à un utilisateur afin de lui définir des droits spécifiques. Ces droits définissent les éléments qu'il peut manipuler et les actions/opérations qu'il est autorisé à faire.

La création et l'instanciation d'un compte suivent la cinématique suivante :



Source : *https://hyperledger-fabric.readthedocs.io/en/latest*

Nœud

Il existe trois types de nœuds dans le réseau **Hyperledger Fabric** :

Client

- C'est un nœud qui créé des transactions et les « diffusent » aux nœuds dont le rôle est de les valider (*endorsing peer*).

Peer

- C'est un nœud dont le rôle est l'enregistrement effectif d'une transaction et qui maintient l'état global du réseau via une copie locale complète de la blockchain. Ces nœuds peuvent également jouer le rôle de validateur d'une transaction, on les appelle alors ***endorser nodes***.

Orderer

- C'est un nœud participant au mécanisme de ***Consensus***.

Transaction

Dans **Hyperledger Fabric**, une transaction est toujours liée à une *chaincode* (*Smart Contract*).

Il existe trois *types* de transactions :

- Deploy transaction :

C'est une transaction qui permet de déployer un **chaincode** sur un nœud du réseau, qui à son tour, de proche en proche, le diffuse sur tous les nœuds du réseau. Une fois le déploiement effectif, le Smart Contract est inscrit dans **Hyperledger Fabric**

- Invoke transaction :

C'est une transaction qui permet d'exécuter un **chaincode** (une de ses fonctions), préalablement inscrit par une transaction *deploy*.

- Query transaction :

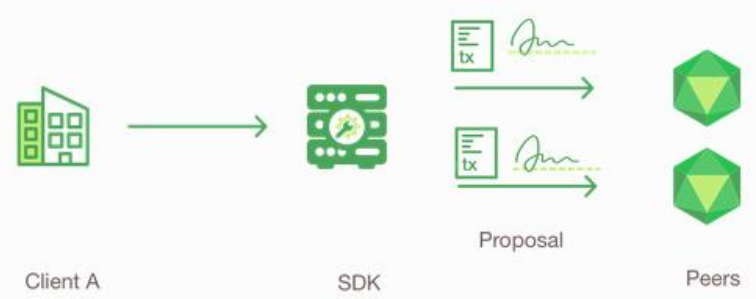
Permet de récupérer l'état et les données d'un **chaincode**

Minage

Il n'y a pas, à proprement parlé, de processus de minage dans **Hyperledger Fabric**. Aucun algorithme de calcul de preuves (PoW, PoS, PoA, PoET) n'est utilisé pour la validation d'un bloc.

La validation d'un bloc et de toutes les transactions qu'il contient suit, la mécanique transactionnelle suivante, composée de six étapes :

- Initialisation d'une transaction



Source : *https://hyperledger-fabric.readthedocs.io/en/latest*

Pour chaque transaction émise sur le réseau, une *proposition d'appel* est construite. Vue par extension comme une **proposition de transaction**. Cette *proposition d'appel* s'apparente à un *stub*, à un contrat d'interface décrivant à la fois, la procédure d'invocation (appel) de la fonction du chaincode (*Smart Contract)* que l'on souhaite appeler et les paramètres attendus en entrée ainsi que les données produites en sortie. Chaque *proposition d'appel* possède une signature unique, générée à partir des données cryptographiques de l'utilisateur (certificats) à l'origine de la transaction.

- Contrôle et validation des *stubs d'appel*

Source : *https://hyperledger-fabric.readthedocs.io/en/latest*

Cette étape, réalisée par les nœuds *Peer* « validateurs » (*endorser*), consiste à vérifier que la *proposition d'appel* (*transaction*) est bien formée (format : *protocol buffer over gRPC*), qu'elle n'a pas déjà été envoyée, que sa signature est valide et que l'émetteur de la transaction dispose des droits suffisants.

Une fois ces *propositions d'appel* vérifiées, elles sont utilisées pour exécuter le chaincode concerné et les données produites en retour forment, à leur tour, une *proposition de réponse*. Cette *proposition de réponse* est signée par un nœud *endorsing peer*.

A ce stade, aucune modification n'a été apportée à la blockchain, puisqu'il s'agit encore que de *propositons*.

- Vérification des propositions de réponse

Source : *https://hyperledger-fabric.readthedocs.io/en/latest*

Cette étape permet de contrôler que pour un chaincode donnée, « ses » différentes *propositions de réponse*, « générées » par les différents nœuds « *endorsing* », sont identiques entre elles et que la politique d'approbation spécifiée a bien été remplie, et satisfaisante.

- Construction des blocs

Source : *https://hyperledger-fabric.readthedocs.io/en/latest*

Une fois l'étape précédente finalisée, les « couples » valides *proposition d'appel/ proposition de réponse* sont combinés au travers d'une transaction particulière diffusés dans le réseau et traités par le service *Ordering*.

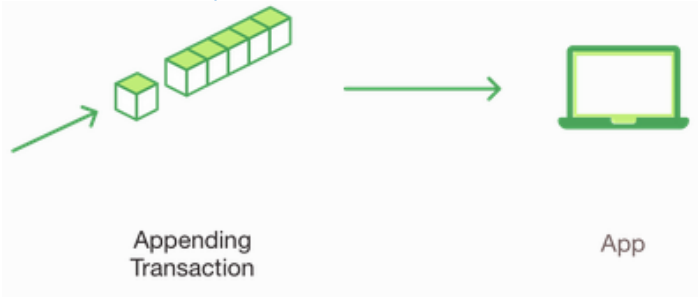
Le rôle de ce service est d'ordonner, dans l'ordre chronologique et par canaux (*channel*), les différentes transactions reçues, et de les assembler (regrouper) ensuite dans un bloc, avec un bloc par canal.

- Validation

Source : *https://hyperledger-fabric.readthedocs.io/en/latest*

Les blocs sont diffusés à tous les nœuds *Peer*, par canal.

- Inscription du bloc



Source : *https://hyperledger-fabric.readthedocs.io/en/latest*

Lorsqu'un bloc est validé, il est automatiquement inscrit dans la blockchain

Smart Contract

Appelés **chaincode** dans l'écosystème **Hyperledger**, les **chaincode** sont, à l'instar des ***Smart Contract*** dans **Ethereum**, des programmes de code informatique fonctionnant sur la blockchain **Hyperledger Fabric**.

Actuellement deux langages de programmation sont supportés pour écrire un **chaincode** : les langages Go et Java.

Un chaincode est identifié par un nom et un numéro de version et les données qu'il manipule et inscrites dans la blockchain ne sont accessibles que par lui, aucun autre chaincode ne peut y accéder directement. Il est néanmoins possible, via une permission appropriée, « d'autoriser » un chaincode à appeler un autre chaincode pour récupérer, par son intermédiaire, tout ou une partie de ses données.

Toutes les données manipulées et inscrites dans Hyperledger sont de type <clé/valeur>.

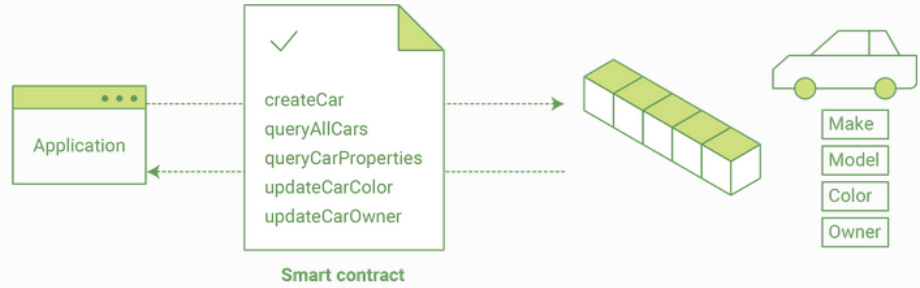
L'écriture d'un **chaincode** impose l'implémentation d'une *interface* composée des trois méthodes suivantes:

- Init

Cette méthode est appelée lors du premier déploiement d'un chaincode et comme son nom l'indique, elle est utilisée pour initialiser les données nécessaires et/ou requises manipulées et/ou utilisées par le chaincode.

- Invoke

Cette méthode est appelée lors de l'appel d'une fonction définie dans le chaincode :



Source : *https://hyperledger-fabric.readthedocs.io/en/latest*

- Query

Cette méthode est appelée à chaque récupération d'une clé (clé/valeur) d'un chaincode

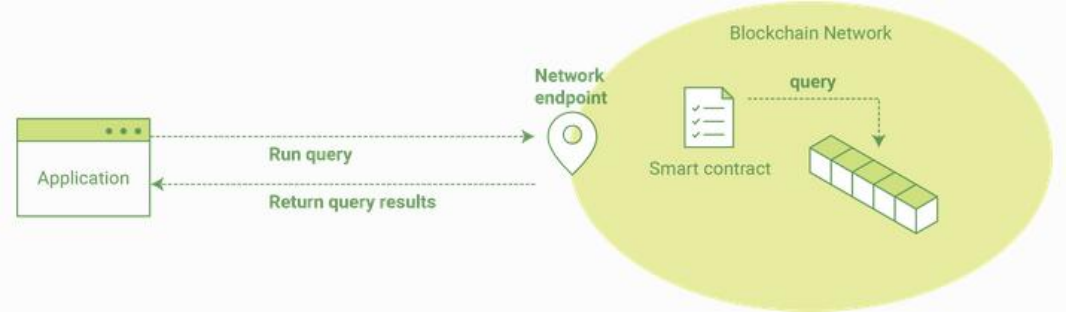
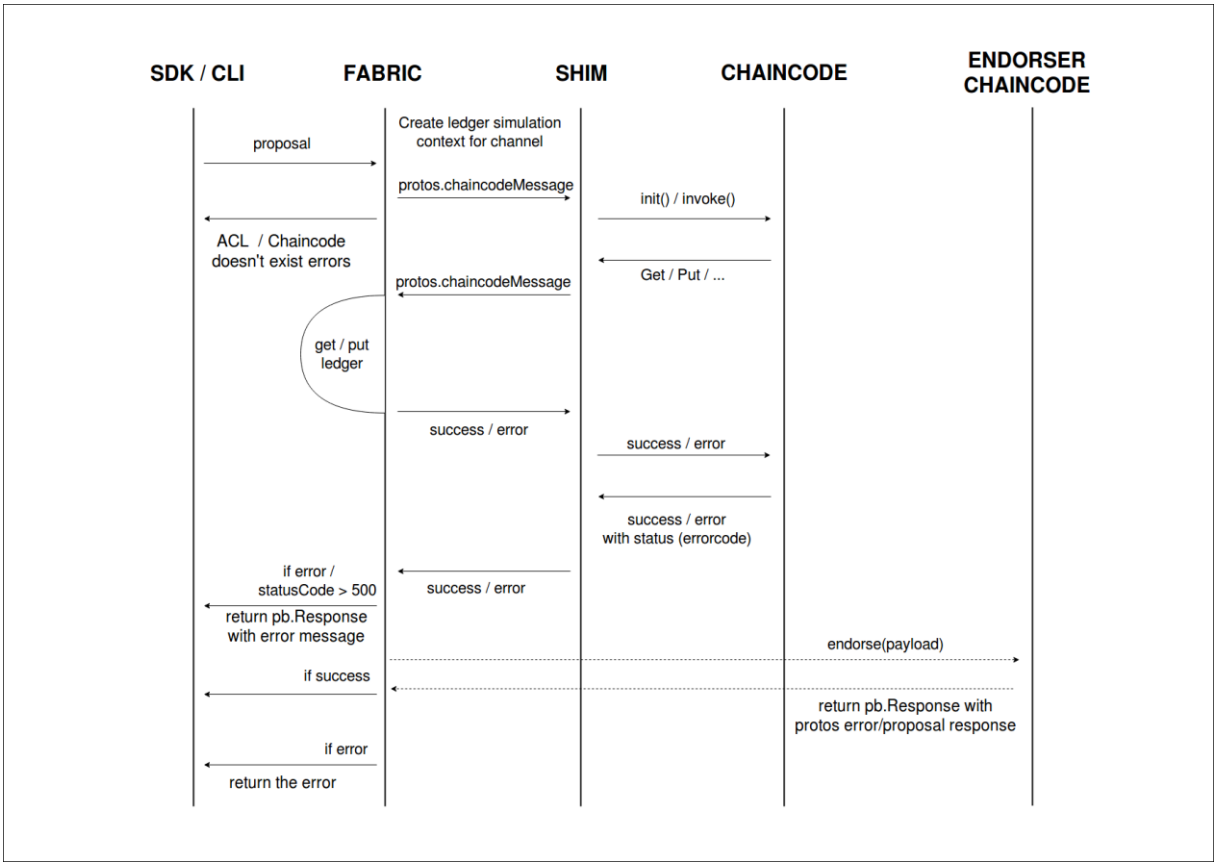


Diagramme d'exécution

Source : *https://hyperledger-fabric.readthedocs.io/en/latest*



VI.2.c **Solutions**

Le choix des solutions retenues et présentées s'est fait sur la sélection des « seuls » outils disponibles.

shim

shim est un package de classes (disponible en Go et en Java) développé par la communauté Hyperledger et fait partie intégrante du projet **Hyperledger Fabric**.

shim se présente sous la forme d'un mini SDK, dédié pur chaincode. Il fournit un ensemble d'APIs qui permettent de créer et de développer un chaincode capable d'interagir avec les nœuds du réseau, d'accéder aux variables « d'état » (couple « clé, valeur »), aux contextes des transactions et aux certificats d'appels. L'API permet également à des chaincode de communiquer entre eux.

Ce SDK est « découpé » en trois catégories :

- Logging

Création et gestion des logs au niveau d'un chaincode

- Stub

Classes utilitaires permettant d'interagir avec les éléments de la blockchain

- Mocking

Création et mise en place de tests

La création d'un chaincode s'effectue en deux étapes :

- Création de l'interface de communication

Cette première étape consiste à implémenter les méthodes d'invocation du chaincode. Pour ce faire, trois méthodes, définies dans l'interface Chaincode, sont à implémenter dans notre code chaincode :

init

Cette méthode est appelée lorsque le chaincode est déployé la première fois et permet de définir son « contexte et le paramétrage associé » (initialisation de variables par exemple).

invoke

Cette méthode est appelée à chaque fois, lorsque l'on souhaite appeler des fonctions définies dans un chaincode. L'appel d'une fonction donne lieu à l'instanciation d'une transaction. L'état du chaincode est mis à jour pour refléter les modifications apportées par les opérations effectuées.

query

Cette méthode est appelée à chaque fois qu'est interrogé l'état d'un chaincode. Elle est utilisée pour lire (récupérer) la valeur d'un paire clé/valeur positionnée par le chaincode.

Cette méthode n'est plus présente dans la version 1.0.0-rc1 et c'est au travers de la méthode invoke qu'elle est exécutée.

- Implémentation du *Stub*

Cette étape consiste à implémenter les méthodes d'interaction du chaincode. Elles sont « formalisées » (liste prédéterminée et fermée) et définies dans l'interface **ChaincodeStubInterface**.

Les principales fonctions sont :

GetArgs : récupère les paramètres passés lors de l'invocation d'un chaincode (**init** et **invoke**).

GetStringArgs : même chose que GetArgs sauf que les paramètres sont récupérés sous la forme de chaines de caractères.

GetFunctionAndParameters : récupère la fonction du chaincode (et ses arguments) dont le nom est passé en paramètre.

GetArgsSlice : même chose que GetArgs, sauf que les paramètres sont récupérés par *tranche*.

GetTxID : retourne l'identifiant de la transaction.

InvokeChaincode : permet d'appeler un autre chaincode.

GetState : permet de récupérer une donnée. Les données, stockées sous la forme de couples clé/valeur, c'est la valeur associée à la clé qui est retournée.

PutState : permet de positionner la valeur associée à une clé.

DelState : permet de détruire une donnée (couple clé/valeur) identifié par sa clé.

CreateCompositeKey : crée une clé composite formée par la concaténation de termes (string) passés en paramètres.

SplitCompositeKey : permet de décomposer la « nomenclature » d'une clé composite.

GetHistoryForKey : renvoie un historique des valeurs associées à une clé, à travers le temps.

GetCreator : retourne l'identité de l'utilisateur qui soumet la transaction.

GetTxTimestamp : retourne la date et l'heure de la transaction

SetEvent : permet d'associer un évènement à la transaction. Si la transaction est validée et « confirmée », l'évènement est publié dans le *monitor* observateur des évènements.

Une fois le chaincode créé, il est déployé à l'aide de l'API REST de **Hyperledger Fabric** via une requête POST sur le endpoint : host:port/chaincode :

```
{
  "jsonrpc": "2.0",
  "method": "deploy",
  "params": {
    "type": 1,
    "chaincodeID":{
        "path":"github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02"
    },
    "ctorMsg": {
            "function":"init",
                    "args":["a", "1000", "b", "2000"]
    },
    "secureContext": "alice"
  },
  "id": 1
}
```

chaincodeID : définit le chemin jusqu'au répertoire contenant le code du chaincode

ctorMsg : définit les paramètres passés. Ici, au déploiement, la méthode **init** est appelée avec les paramètres listés ("a", "1000", "b", "2000")

secureContext : contient l'ID d'enregistrement d'un utilisateur définit et inscrit, au préalable, dans le service *MemberShip*. La présence de cet élément active la sécurité.

La réponse à une demande de déploiement chaincode contiendra un élément *status* confirmant la réussite de la demande. La réponse à une demande de déploiement réussie contiendra également le hash du chaincode. C'est ce hash qui doit être utilisé dans les requêtes ultérieures envoyées à ce chaincode.

A titre d'exemple, voici le message retourné en cas de succès :

```
{
    "jsonrpc": "2.0",
    "result": {
        "status": "OK",
        "message":
"395b5e34d8d4a7cd69fb6aa00099b8fabed83504ac1c5d61a425aca5b3ad3bf96643ea4fdaac132c417c37b
00f88fa800de7ece387d008a76d358652b0d803fc"
    },
    "id": 1
}
```

L'API REST **Hyperledger Fabric** définit les *endpoints* suivants :

Block
GET /chain/blocks/{Block}
Blockchain
GET /chain
Chaincode
POST /chaincode
Network
GET /network/peers
Registrar
POST /registrar
DELETE /registrar/{enrollmentID}
GET /registrar/{enrollmentID}
GET /registrar/{enrollmentID}/ecert
GET /registrar/{enrollmentID}/tcert
Transactions
GET /transactions/{UUID}

Package « Chaincode » de référence, **shim** est un SDK relativement simple d'apprentissage et d'utilisation.

Hyperledger Fabric Client (HFC)

Ecrit en Typescript, **Hyperledger Fabric Client** (HFC) est un SDK pour **Node.js** permettant d'interagir avec une chaine de blocs de type **Hyperledger Fabric**.

Il fournit un ensemble d'API permettant :

- D'enregistrer et d'inscrire des utilisateurs au réseau de la blockchain (service Membership). Un administrateur d'applications Web disposant des droits suffisants (registrar) peut enregistrer et inscrire de manière dynamique des utilisateurs qui se sont authentifiés auprès de l'application Web.
- De soumettre des transactions au réseau de la blockchain (deploy, invoke et query). Toutes les transactions sont anonymes, confidentielles.
- De déployer et d'interagir avec des chaincode
- De stocker des clés privées et des certificats, sensibles, dans un *magasin* custom de votre choix. Par exemple, une base de données à l'extérieur de la chaîne de blocs.

Deux méthodes sont disponibles pour inclure et utiliser le SDK HC dans une application **Node.js** :

- Méthode Offline

Cette méthode passe par la copie de tous les fichiers du SDK dans le répertoire **/lib** de l'application **Node.js**, puis par son « inclusion » (~ import) dans l'application (var hfc = require("./lib/hfc");).

- Méthode « npm »

Cette méthode passe par l'installation du SDK HC via le gestionnaire de paquets officiel de **Node.js**, npm. Une fois l'installation effectuée, il doit être référencé dans l'application pour être utilisé (var hfc = require('hfc');).

Les objets HFC (classes et interfaces) sont hiérarchisés et classés comme suit :

- Chain, est la classe principale. Elle est la représentation client d'un réseau blockchain. HFC vous permet d'interagir avec plusieurs réseaux et de partager un objet KeyValStore (magasin clé/valeur) et MemberServices unique avec plusieurs objets *Chain* si nécessaire. Pour chaque réseau de blockchain, vous devez ajouter un ou plusieurs objets *Peer*, lesquels représentent les nœuds finaux auxquels se connecte HFC pour soumettre des transactions.

- L'interface KeyValStore est utilisée par HFC pour stocker et extraire toutes les données persistantes. Ces données incluent des clés privées, qui doivent être conservées en lieu sûr. L'implémentation par défaut, située dans la classe FileKeyValStore, est une version basée sur des fichiers.

- L'interface MemberServices est implémentée par la classe MemberServicesImpl. Elle assure la sécurité et fournit des fonctions liées aux identités, comme le caractère privé des données, l'impossibilité de relier des transactions (chaincode) et la confidentialité. Cette implémentation émet des certificats d'enregistrement (eCerts) (certificats d'enregistrement de membre) et des certificats de transaction (tCerts) (certificats de transaction pour chaque membre).

- La classe Member représente les utilisateurs finaux qui effectuent des transactions sur le réseau, et d'autres types de membres, comme les (noeuds) *Peer*. La classe Member, qui interagit avec l'objet MemberServices, est utilisée pour enregistrer et inscrire les membres et les utilisateurs (*Membership*). Elle permet aussi de déployer, interroger et appeler un chaincode directement depuis la classe Member. en effectuant une transaction avec les objets Peer ; cette implémentation délègue simplement le travail à un objet TransactionContext temporaire.

- La classe TransactionContext implémente la logique de déploiement, d'appel et d'interrogation. Chaque instance TransactionContext reçoit un certificat de transaction (tCert) de MemberServices. Ce certificat permet de soumettre et d'émettre des transactions. Pour émettre plusieurs transactions avec le même certificat (tCert), il faut récupérer un objet TransactionContext directement depuis l'objet Member. Ce TransactionContext fournit alors, tout l'outillage pour émettre des transactions *deploy*, *invoke* et *query*. Toutefois, l'utilisation d'un seul certificat tCert pour plusieurs transactions a pour conséquence de les relier. Il est alors possible d'identifier une transaction dès lors qu'elles impliquent le même utilisateur anonyme. Pour éviter la liaison de transaction, il faut appeler deploy, invoke et query depuis un objet User ou Member.

VII - CONCLUSION

Phénoménal *buzzword*, la blockchain est sur toutes les lèvres. C'est le SUJET *hype* du moment dont tout le monde parle. En témoigne son incroyable emballement médiatique.

Modèle sécurisé de consensus décentralisé, la Blockchain ouvre le champ des possibles. Pas un secteur d'activité ne lui échappe, ses cas d'usages semblent « illimités ».

Révolution économique, sociale et sociétale, elle a eu un impact mondial très fort. Déjà, en fin d'année 2016, la devise était : « **Partout et pout tout !** ».

Considéré comme la seconde révolution internet, elle promet de s'affranchir des entités de confiance privée, comme les banques, les assureurs, les notaires, etc. Elle porte la vision de redonner du pouvoir aux citoyens.

Pour autant, l'écart entre ses promesses attendues et ses mises en œuvre effectives a fait douter de sa réelle viabilité et de son employabilité.

Il a fallu du temps (un an environ depuis l'annonce) avant que n'émerge des vrais cas d'applications concrets et que la Blockchain devienne une « réalité ».

Aujourd'hui, en 2017, la mise en pratique et les avantages apportés par la Blockchain ne sont plus à démontrer. Son marché compte plus d'une centaine de solutions. Il y en a vraiment pour tous les « goûts ! ». Ses possibilités sont tellement nombreuses, qu'elle crée même de nouveaux services, de nouveaux marchés.

Porté par sa communauté très active et ses acteurs moteurs, l'écosystème Blockchain a atteint cette année, une certaine maturité.

Deux grandes catégories de Blockchain se distinguent. D'un côté les blockchain publiques, ouvertes à tout le monde (Bitcoin, Ethereum , Tezos, etc.), et de l'autre côté les blockchain entreprises (Hyperledger, Ripple, MultiChain,etc.). Privées, elles ne sont accessibles que pour des utilisateurs autorisés et connus. Son design et son activité sont sous la tutelle d'un tiers de confiance.

Parmi ces catégories, deux blockchains tirent leur épingle du jeu et sortent vraiment du lot :

Ethereum
Après un début difficile (DAO Attack), **Ethereum** est aujourd'hui la référence en matière de Blockchain publique.

Première à implémenter le concept de ***Smart Contracts***, elle a fortement contribué à l'ouverture des fonctionnalités de la Blockchain.

La richesse de sa documentation et la qualité des implémentations ont contribué à son développement et continuent d'y contribuer. Assez simple à comprendre, la mise en œuvre d'expérimentations de petites blockchain locales se fait très vite et très bien. Très riche dans ses possibilités, sa maitrise demande, en revanche, un « long » investissement en temps (un peu moins d'un mois).

Ethereumj (l'implémentation d'**Ethereum** en Java) est le SDK à maitriser pour les javaistes. Implémentation de référence du protocole **Ethereum**, il en expose tous les aspects et toutes les composantes. Il permet, pour les amoureux de la connaissance et du détail, de descendre *très bas niveau* et d'agir finement au niveau du protocole et des composants techniques.

Dotée d'une feuille de route claire et précise, sa vision à moyen terme (courant 2017) est d'être accessible et utilisable par le grand public (personnes non techniques).

Ajouté à cela, ces constantes améliorations font d'**Ethereum** une solution pérenne et sur laquelle il faudra compter. Elle est construite pour durer.

Hyperledger Fabric

Blockchain privée, **Hyperledger Fabric** est une référence dans le monde des Blockchains entreprises.

Son *token* programmable permet « d'échanger » n'importe quels types d'actifs. La versatilité de ses cas d'usages confère à **Hyperledger Fabric** des usages multiples et très intéressants. Grace à ces composants *Plug&Play*, on peut pratiquement bâtir la blockchain que l'on souhaite précisément.

Comme **Ethereum**, elle dispose d'une documentation riche et de qualité. Plus complexe à appréhender par contre, son apprentissage demande plus de travail.

Bâtie sur des conteneurs Docker, l'installation et la mise en œuvre d'une blockchain **Hyperledger Fabric** est simple et efficace. Là encore, la mise en œuvre d'expérimentations de petites blockchain privées locales se fait très vite et très bien.

Si l'on souhaite se concentrer uniquement sur le développement et la mise en œuvre de chaincodes, **Shim** est la solution parfaite.

Construit sur une architecture logicielle basée sur des interfaces à implémenter, le développement d'un chaincode suit une logique précise, structurée et normée. Loin d'être pénalisant, ce design forme un cadre didactique plus simple à maîtriser.

Dans **Hyperledger Fabric**, les chaincodes sont écrits soit en langage Go, soit en langage Java. Cette caractéristique a deux impacts positifs. Le premier, c'est qu'elle simplifie énormément le développement et l'écriture du code du chaincode. Le deuxième, c'est que ces langages, par définition, permettent d'écrire « tout ce que l'on veut ».

Si par contre on cherche à construire une application d'administration et d'orchestration du réseau, il faut se tourner vers des SDK complets. **Hyperledger Fabric Client (HFC)** ou **Fabric SDK Java** en sont des exemples. Comme dans le cas d'**Ethereum**, avec qui on peut descendre très bas niveau et tout contrôler.

Ethereum et **Hyperledger Fabric** ne sont pas les seules blockchain intéressantes. Pour en citer quelques-unes, Tezos, Corda, Ripple, « Lightning Network » sont d'autres Blockchains à surveiller.

Impossible à normer et à fédérer, l'un des problèmes mais aussi l'un de ses atout, , est qu'il n'existe pas une Blockchain mais plusieurs. À chaque cas d'usage « propice », correspond une Blockchain spécifique. Chacune avec ses avantages et ses défauts. Devant cette grande diversité des technologies et des solutions, il est facile de se perdre et faire des choix s'impose. **Ethereum**, **Hyperledger Fabric** et **Tezos** sont, sans conteste, les blockchains à connaitre, à suivre et à surveiller.

L'une des constations frappante concernant l'écosystème open source de la Blockchain, c'est que malgré sa richesse, il n'existe pas encore, à ma connaissance, de solutions *full UI, Click & Play*.

La mise en œuvre d'une Blockchain nécessite et impose toujours une phase de développement. Plus ou moins longue et plus ou moins complexe, c'est une composante importante à prendre en compte dans un projet Blockchain. La

connaissance des différentes Blockchain est le premier pré requis. La sélection du type de Blockchain, adéquat aux besoins, est primordiale. Un mauvais choix et c'est tout le projet qui est en péril. La « parfaite » maitrise technique des technologies et des protocoles sont les deux autres pré requis « obligatoires » à la réussite du projet.

Face à cette problématique, un nouveau type d'acteur à fait son apparition dans le monde informatique, les FAB.

Tout le monde connait le rôle des FAI (Fournisseur d'Accès à Internet), maintenant avec la Blockchain il existe aussi des FAB (Fournisseur d'Accès à la Blockchain). Des sociétés comme **Chain Orchestra** ou **Stratumn** se proposent de développer, d'héberger et d'administrer des Blockchains conformes aux besoins client attendus. Nouveau marché, la Blockchain as a Service (BaaS) attire de grands acteurs du monde informatique. **Microsoft** et **IBM** sont les deux premiers à proposer une offre commerciale BaaS. La solution de **Microsoft** s'appuie sur **Ethereum** et **Azure**. La solution d'IBM, quant à elle, s'appuie sur **HyperLedger** et **Bluemix**.

Il est difficile d'évaluer les gains financiers générés par la Blockchain dans les années à venir. La tendance veut qu'ils soient conséquents. Il est acquis que son modèle économique soit une « réalité viable ».

Malgré tous ses atouts et tous ses avantages, la Blockchain n'est pas la « solution idéale ». On lui prête des vertus qu'elle n'a pas. Certains cas d'usage sont impossibles à transposer en Blockchain. C'est le cas par exemple pour les salles de marché ou le réseau VISA. Nerf de la guerre, la nécessité d'un grand nombre de transactions par seconde (20 000 transactions par seconde pour VISA) est vital à leur fonctionnent et à leur développement.

A l'heure actuelle, aucune Blockchain n'est capable de traiter autant de transactions par seconde. Dans le réseau **Bitcoin**, c'est toutes les 10 minutes qu'un bloc est ajouté. Dans **Ethereum**, c'est toutes les 17 secondes. On est bien loin du compte.

« Récemment », un nouveau projet, basé sur Bitcoin, s'est créé pour adresser et répondre à cette problématique. Il s'agit du projet **Lightning Network**.

La performance (volume de transactions) n'est pas le seul frein à la mise en place d'un système Blockchain.

D'autres freins existent :

Scalabilité

Une des caractéristiques de la Blockchain est qu'elle enregistre continuellement de nouvelles informations sans jamais rien supprimer. On a donc une problématique de stockage exponentiel qui n'est a priori pas viable à long terme.

Gouvernance

Sans organe de contrôle, sa gouvernance « pose » plusieurs problèmes :

- Qui décide ? Les développeurs ? Les mineurs ?
- Comment se prennent les décisions ? Par vote ? Par adoption ?
- Quels recours en cas de litige ?

Juridique

Développant de nouveaux modèles, la Blockchain souffre de deux « vides » juridiques concernant les questions sur la responsabilité et la régulation.

Pour conclure :

La Blockchain est un sujet passionnant mais qui demande du temps et un investissement important afin d'en tirer toute la quintessence. Un temps d'autant plus important que le marché est encore en pleine évolution. Non triviale, la réalisation d'un projet Blockchain peut très vite devenir complexe, voire insoluble. Il est donc impératif de s'assurer que l'utilisation de la Blockchain se justifie dans le cas d'usage sélectionné.

De fait, c'est dès à présent qu'il faut investir dans la Blockchain. Le choix de la ou des Blockchain(s) à étudier et à maitriser est primordial. L'écosystème Blockchain compte trop de solutions différentes pour qu'elles soient toutes apprises.

Pour autant il ne se passe pas une journée sans que l'on parle des Blockchains. Et qui ne comptera pas de Blockchain à son actif en 2017, pourrait bien être dépassé.

N'hésitez pas à nous transmettre vos avis et évaluations sur ce livre blanc !

Envoyez-nous un message par ici : ***https://www.smile.eu/fr/contact***

Édition 2017