Audit Technique

et Préconisations

Mai 2016

Table des matières

Principes

L'objectif du présent audit consiste à dresser un état des lieux de l'architecture existant, des éléments en production comme des projets en cours de développement afin de proposer un schéma directeur pour la mise en place de fondamentaux, de normes et d'outils permettant d'assurer l'industrialisation de l'offre Usitab.
Les recommandations seront notamment basées sur le manifeste « 12 Factor App »[1]

Au vu de l'analyse de l'existant et des objectifs définis les préconisations architecturales concernant la gestion du code et de son déploiement seront de plusieurs natures mais organisées en respectant quatre axes principaux détaillés ci-après :

- simplicité
- interopérabilité
- évolutivité
- scalabilité

Simplicité

Plus une architecture est complexe et plus les éléments qui la composent sont élémentaires.
Il est donc critique de s'appuyer sur un nombre limité de technologies dont la maitrise est facile à acquérir en interne ou sous-traiter. Ces technologies sont déjà pour l'essentiel maitrisées par Usitab.
On retrouvera également ce concept de simplicité avec les principes KISS[2] ou DRY[3] au niveau de la gestion du code.

1 http://12factor.net/fr/

2 https://en.wikipedia.org/wiki/KISS_principle

3 https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

Interopérabilité

Les applications qui composent une infrastructure sont communiquantes. Cette communication doit s'appuyer sur des standards ou des bonnes pratiques à chaque fois que c'est possible.

Evolutivité

Le code informatique est vivant, l'écosystème dans lequel il est exécuté également. On développe pour exploiter pendant plusieurs années voire plusieurs décennies. Il est nécessaire d'intégrer cette donnée dés la première ligne de code ou le premier choix architectural.

Scalabilité

C'est un challenge permanent de gérer la montée en charge des infrastructures déployées. Les récentes innovations techniques sont tournées vers la résolution de ce problème (les microservices notamment)

Problématiques IoT

L'une des promesses d'Usitab est d'accéder et de tirer parti de mesures générées par les chaines de production afin d'améliorer leur fiabilité et limiter leurs temps d'arrêt.

Dispositifs de relevé de mesures automatisées

Les dispositifs de relevé de mesure sont de deux natures principales :

- intégré à la chaine de production
- extérieur à la chaine de production

Modes d'obtention des mesures automatisées

Les mesures sont de différentes natures et toujours corrélées à un identifiant temporel :

- état (actif / inactif)
- quantité
- indice de masse ou de volume
- indice de température
- etc.

Mode invasif

On parle de mode invasif quand le relevé de la mesure implique de se connecter à un dispositif intégré à la chaine de production. Plusieurs modes sont possibles :

- connexion à un bornier (signaux électriques)
- connexion à un port RS232 (balances notamment)

C'est le mode le plus évident car il consiste à tirer parti de dispositifs natifs de la chaine de production, calibrés.

Il est cependant de nature à impacter la production à l'installation voire pendant le fonctionnement. Il est donc important qu'il soit réalisé par du personnel habilité, idéalement du client

Mode non invasif

Le mode non invasif consiste à implanter un dispositif extérieur à la chaine à proximité immédiate de celle-ci. L'installation et le calibrage sont alors deux opérations à réaliser directement. Il est important de s'assurer que les dispositifs ne puissent pas mettre en péril le fonctionnement de la chaine en cas de dysfonctionnement.

La variété des données pouvant être collectées ainsi que la diffiulté technique que cette collecte peut représenter font de ce mode une hypothèse immédiatement peu réaliste. Affichant en tout cas de nombreux verrous techniques.

Remontée des données

Les données collectées doivent pouvoir être sauvegardées afin d'exécuter les opérations souhaitées.

L'usine est dans ce contexte un environnement de communication plutôt hostile : bruyant, étendu, souvent constitué d'une structure métallique et abritant quantité d'autres structures de même nature. Sans parler des changements de température ou d'hygrométrie.

Afin d'assurer la remontée de données de manière robuste, les protocoles de communication sans fil doivent permettre de traverser les obstacles et une distance donnée. Plusieurs protocoles existent qui demandes à être testés (LoraWan, Sigfox).

Prototypage rapide

Le prototypage de dispositifs de mesure est nécessaire afin de vérifier la capacité à lever les verrous techniques.

Pouvoir assembler des modules de mesures et des modules de communication comme le propose la société espagnole **Libelium** via sa plateforme **Waspmote**[4] représente une possibilité intéressante et mobilisable rapidement.

Ces dispositifs peuvent de plus être connectés à des cartes Arduino ou Raspberry Pi.

Notre recommandation est de consacrer une attention particulière aux

4 http://www.libelium.com/products/waspmote/

quelques sociétés déjà actives dans l'univers de l'IoT.

Un contact pourrait notamment être initié avec Factory Systemes[5], société française qui pourrait apporter son conseil. Par ailleurs, la société française Sigfox[6] qui a déployé son propre réseau de communication dédié à l'IoT pourrait également être un partenaire, notamment via son programme dédié aux Startups[7].

5 http://www.factorysystemes.fr/

6 http://www.sigfox.com/

7 http://makers.sigfox.com/startups/

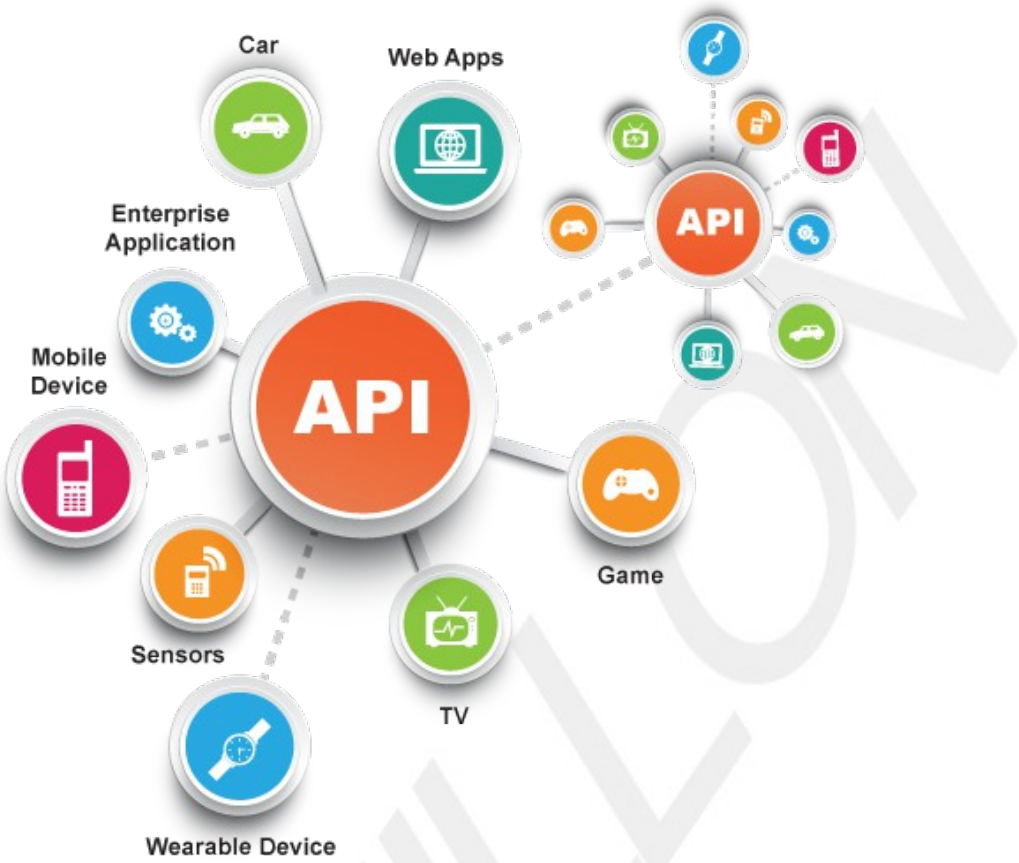
Architecture globale

Dans le cas présent, la société ne couvre encore qu'une partie de sont périmètre fonctionnel et technique et que la charge appliquée sur son infrastructure est encore limitée en terme de volumes de données entrantes et sortantes. Cette situation est vouée à évoluer à un rythme pas forcément maîtrisable.

API

Les API sont devenues incontournables ces dernières années, et à juste titre. Promettant l'accès à des services hétérogènes d'une manière simple et standardisée, quels que soient leur localisation ou leur langage de développement. Permettant de composer ainsi des applications en agrégeant services et données.

Il est ainsi possible de s'appuyer sur des services, plus nombreux chaque jour, accessibles sous forme d'API et généralement facturés à l'usage. On relève d'ailleurs que les géants du Cloud (Amazon, Google, Microsoft) proposent de plus en plus d'API extrèmement techniques (reconnaissance vocale, reconnaissance d'image, aggrégation de flux IoT, etc.)



En séparant le backend et le frontend, les API ont rendu possible la multiplicité des terminaux et induit une forte spécialisation des développeurs. Hier « Fullstack », c'est à dire capable de développer aussi bien du côté Backend que Frontend, le développeur doit se concentrer aujourd'hui sur l'un de ces environnements. Les API ont libéré le backend et permis l'émergence de nouvelles technologies (languages, frameworks, bases de données).
Dans le même temps, et avec le développement du mobile, le Frontend a acquis ses lettres de noblesse, avec les principes de l'UI (interface utilisateur) et de l'UX (expérience utilisateur) et l'apport de multiples bibliothèques (Javascript, CSS) et d'outils de développement.

Langage PHP

Le choix actuel de PHP pour la partie backend est cohérent. PHP est un langage web mûr dont la communauté de développeurs est l'une des plus vastes au monde.
PHP est un langage simple à apprendre et qui disposent de plusieurs frameworks open source de qualité (Zend Framework, Symfony, Laravel...) permettant d'ébaucher puis de

développer rapidement des applications web de toutes natures.

La branche 7.0 publiée fin 2015 apporte de plus des nombreuses améliorations en matière de performances comme de consommation mémoire grâe à un moteur interne entièrement réécrit.

En parallèle, Facebook, grand utillisateur de PHP et lassé ces dernières années de la lenteur d'évolution de PHP a créé sa version dérivée (quoique compatible) nommée HHVM (Hip Hop Virtual Machine)

C'est donc un choix valable pour tous les développements autour des API.

Nous recommandons de tester la compatibilité des logiciels existants avec PHP7 et de prévoir une intégration rapide de cette version.
Il est important de garder un œil sur les développements de HHVM mais notre recommandation n'est pas de migrer en l'état actuel des connaissances sur la suite que Facebook entend donner au projet.

MySQL/MariaDB

Très traditionnellement, PHP est utilisé en conjonction avec le SGBD MySQL pour tout ce qui concerne la persistance de données. Cette base de données Open Source, rachetée par Sun, lui-même racheté par Oracle est un produit solide et performant apportant les fonctionnalités attendues d'un SGDB critique. Les moteurs de stockage sont des briques additionnelles (« plugin ») ce qui permet de les mixer selon le besoin rencontré.

Suite aux différents rachats, une partie de l'équipe (dont le créateur original) ont quitté la société pour créer un projet dérivé (ce qu'on appelle un « fork », ce que permet la licence Open source) nommé MariaDB. Les deux projets sont compatibles pour l'instant mais MariaDB propose un nouveau moteur de stockage alors que MySQL reste sur l'historique InnoDB.

Nous recommandons de choisir l'un des deux projets et de faire en sorte de l'utiliser en production avec des versions proches ou

identiques.

En tout état de cause, éviter si possible de mixer dans l'infrastructure et MySQL et MariaDB pour faciliter la maintenance et le suivi des mises à jour.

Serveur Web

L'architecture autour de PHP et MySQL est souvent désigné sous l'acronyme LAMP (Linux Apache MySQL PHP) Outre que cet acronyme revêt parfois des significations différentes (P peut également représenter Python, un autre langage), Le A pour Apache (le serveur web) tend à devenir un peu moins vrai au fil du temps.

En effet, après plus de 20 ans, Apache demeure le serveur web le plus populaire, très puissant, il n'a jamais été la solution la plus performante pour PHP.
Ceci est lié à la façon dont PHP a été développé à l'origine et donc aux contraintes de communication entre Apache et PHP. Sans trop entrer dans les détails techniques, La capacité de monter en charge du couple Apache+PHP est historiquement faible, conduisant à multiplier les serveurs (via un cluster par exemple) ce qui induit d'autres problématiques.

De plus, la consommation mémoire d'Apache est importante car liée aux modules qui sont activés dans sa configuration, qu'ils soient nécessaires à un instant T ou non.

Les dernières années ont permis l'avènement d'un autre modèle plus performant. Ce modèle tire partie du serveur web NGINX (prononcer « N – GINE - X ») notamment beaucoup plus sobre en matière de consommation mémoire et de PHP configuré comme un processus indépendant et non comme un module (PHP-FPM).

Les performances sont impressionnantes et le découplage du serveur web et de PHP permettent de dissocier le cycle de vie des deux éléments, notamment en matière de mises à jour.

Nous préconisons de tester puis déployer les applications existantes sur

Nginx+PHP-FPM afin de tirer partie du gain de performances sans modification des caractéristiques du serveur)

Linux

Les technologies enumérées sont disponibles sur une variété de plateformes (UNIX, Linux, Windows, etc.) mais sont surtout exploitées sur des serveurs faisant tourner l'OS Linux.

Linux est un terme générique qui décrit une architecture technique. Ce qui est installé sur les serveurs est nommé « distribution ». Il s'agit d'un ensemble de paquets cohérents apportant des fonctionnalités indispensables ou optionnelles. L'installation d'une machine sous Linux peut donc revêtir des réalités différentes selon qu'il s'agisse d'un serveur ou d'un poste bureautique, sans ou avec une interface graphique.

Deux distributions sont particulièrement populaires en matière de déploiement de serveurs : Debian et Ubuntu. La distribution Debian est historiquement plus orientée serveurs que stations de travail. Sa philosophie consiste à n'intégrer que des logiciels dont les versions sont particulièrement stables, ce qui assure sa stabilité mais ralentit l'intégration des nouvelles versions de logiciels.

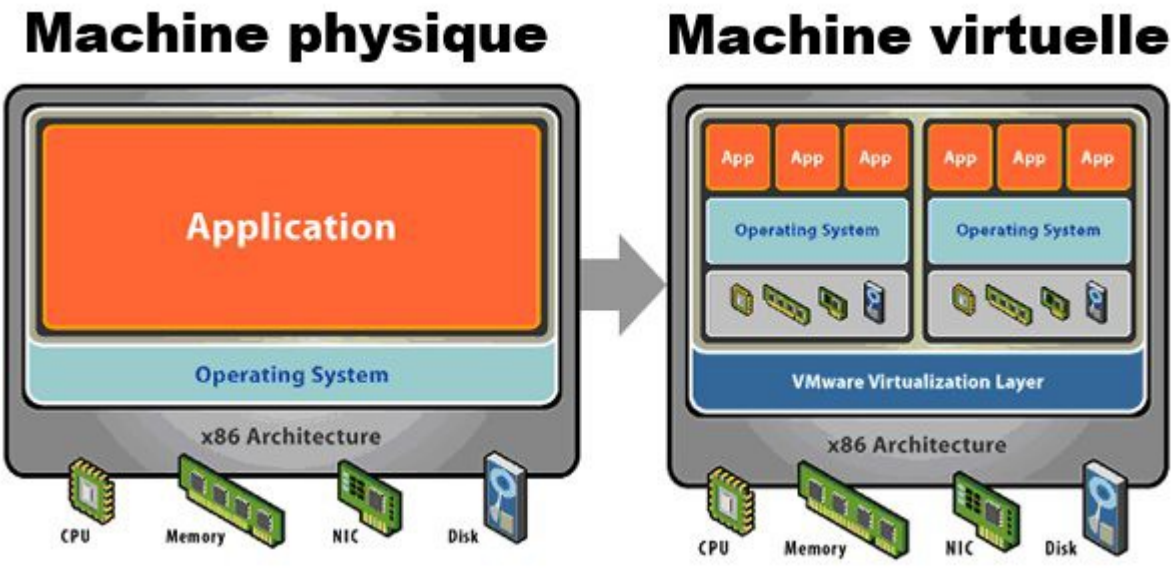
La distribution Ubuntu, plus orientée vers les stations de travail, gagne par ce biais, du terrain sur Debian en proposant un environnement qui peut être le même en développement comme en production et en proposant une adoption plus rapide des nouvelles versions de logiciels.

Nous considérons ces deux distributions également adaptées à la production sous réserve, naturellement qu'elles soient régulièrement mises à jour.
Elles sont par ailleurs disponibles en standard chez quasiment tous les hébergeurs Cloud en faisant ainsi des outils standard du déploiement d'applications web.

Des versions de Linux orientées spécifiquement vers les besoins de déploiement de containers dans des architectures microservices sont en train d'apparaitre sur le marché. CoreOS est l'un d'eux. Même si ces distributions ont tendance à se populariser rapidement, il ne nou semble pas nécessaire de s'y former imédiatement.

Virtualisation

Les technologies de virtualisation ont complètement disrupté les datacenters ces dix dernières années et ouvert la porte au Cloud.
Dans la virtualisation, on ne parle plus de serveur sinon pour décrire les hôtes physiques qui virtualisent les services qu'on appelle alors instances. Le nombre des serveurs est réduit drastiquement.
La maturité des solutions (commerciales comme open source) en fait aujourd'hui un incontournable et la baisse des prix rend possible la mise en œuvre et la gestion d'un datacenter complet à partir de trois serveurs physiques permettant de configurer et faire tourner plusieurs dizaines d'instances virtuelles parfaitement isolées et aux ressources contrôlées.



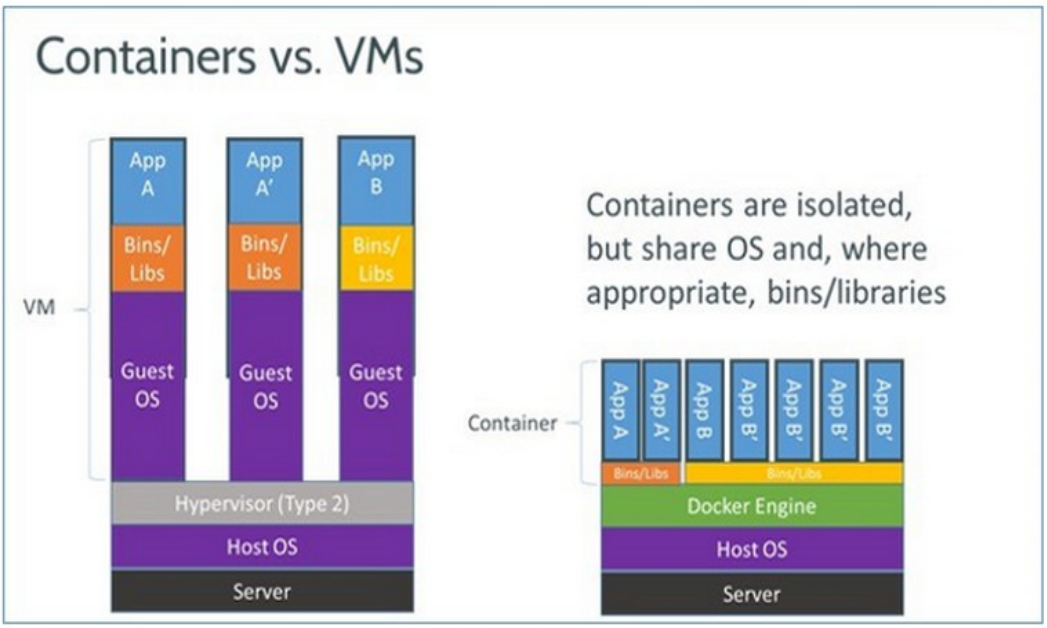
Le point fort de la virtualisation est l'utilisation optimale des ressources des serveurs physiques (les hôtes). Là où un serveur physique gaspillait souvent ses capacités de calcul, sa mémoire ou son espace de stockage, la virtualisation permet la mutualisation des ressources et la répartition des instances dans le respect de leur disponibilité.

Notre recommandation est de tirer partie des apports de la virtualisation, notamment en matière de création de modèles d'instances et de sauvegarde à chaud permettant de garantir l'intégrité des de l'infrastructure et des données.
Ces fondamentaux facilitent par ailleurs la mise en place d'un Plan de Reprise d'Activité (PRA) crédible et mobilisable en cas d'incident majeur.

Containerisation

La containerisation est la dernière étape de la disruption du datacenter. Il s'agit d'isoler chaque processus dans un espace système donné et de définir finement les échanges (on parle d'entrées/sorties ou d' « IO ») qu'il peut établir avec d'autres processus.
Dans ce paradygme on ne parle plus d'instance (celle-ci a été éclatée sur la base des processus qu'elle recoupe) mais de container.
Le container n'a pas d'OS, il partage les ressources dont il a besoin avec d'autres containers. Par défaut le container est d'ailleurs sans état (« stateless »), c'est à dire qu'à l'extinction, il perd toute mémoire et qu'il redémarre vierge.
Naturellement, il est possible de lui apporter de la persistence en définissant des volumes dédiés à la conservation des données entre chaque extinction/redémarrage)

L'apport des containers rebat totalement les cartes du développement web en facilitant les approches SOA (« Service Oriented Architecture ») théorisées il y a plus de dix ans mais dont la mise en œuvre s'est heurtée à des verrous techniques, en voie aujourd'hui d'être levés.
On ne parle d'ailleurs plus tellement de SOA mais de Microservices, sujet que nous allons évoquer bientôt et qui fera l'objet d'une préconisation à part entière.



L'intégration des technologies de containerisation permet également d'étendre le champ de l'intégration continue vers le déploiement continu.

L'univers des containers connait depuis quelques mois une accelération de l'offre d'orchestrateurs. En effet, la gestion de dizaines voire de centaines de containers devient rapidement complexe. Pour adresser cette problématique, il est indispensable de disposer d'outils permettant de lancer et de surveiller des containers, notamment organisés en grappes.
Docker propose plusieurs solutions distinctes (Docker Cloud[8], Universal control Plane[9]), Google propose Kubernetes[10], accessible dans sa plateforme Cloud.

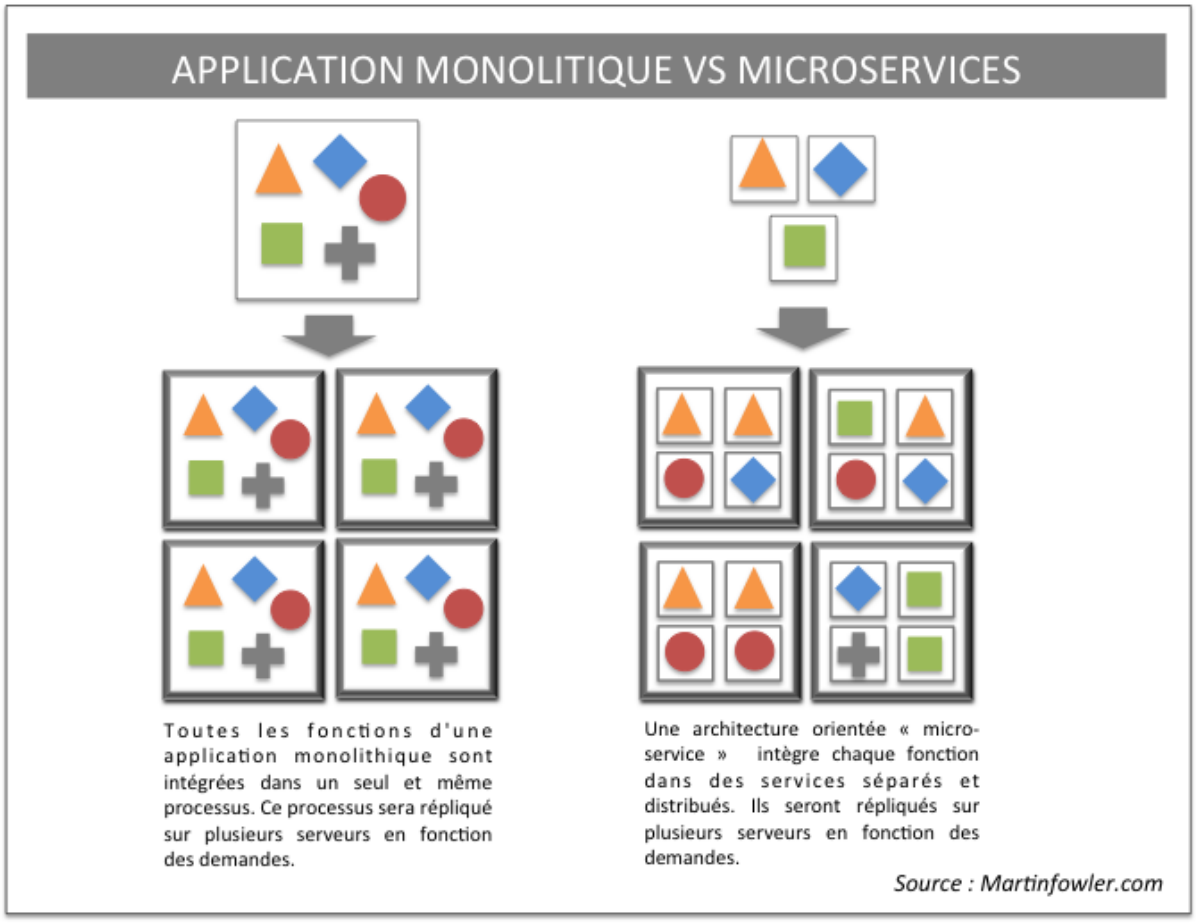
Microservices

L'approche microservices représente une évolution fondamentale du développement des logiciels.

8 https://www.docker.com/products/docker-cloud

9 https://www.docker.com/products/docker-universal-control-plane

10 http://kubernetes.io/



Elle n'est au final que la conclusion évidente d'un ensemble de problématiques liées au web dont la solution est rendue possible par des avancées techniques et l'adoption de méthodologies.

Les microservices proposent d'implémenter le principe de single responsability au delà des algorithmes, au niveau des serveurs eux-mêmes, plus précisément des instances et des containers.
Les avantages apportés par cette implémentation sont multiples : Alors que le développement logiciel se complexifie tandis que son périmètre s'accroie, le découplage des éléments logiques permet de les faire évoluer de manière indépendante, sans interruption des processus.

Qu'il s'agisse de changer la version installée du logiciel d'un composant ou d'ajuster en temps réel ses capacités de traitement, les microservices sont la solution.

Naturellement, à nouveau paradigme, nouvelle approche. Les microservices ne viennent pas seuls, ils émergent d'un ensemble de bonnes pratiques qui s'imposent depuis plus de dix ans :

- les tests unitaires et leur automatisation
- les services web, historiquement SOAP, aujourd'hui RESTful,
- l'intégration continue
- les apports de la virtualisation et plus encore de la containerisation.

On le voit bien, des microservices est grandement profitable en terme d'optimisation du cycle de vie du logiciel, aussi bien pour son développement que son déploiement. Pour autant, en tirer partie réclame une stratégie dans le temps et le respect d'une méthodologie importante.

Dans le cas d'Usitab, nous relevons un paradoxe dans le sens que la société est au bon moment de son histoire et de ses développement pour mettre en œuvre et déployer une stratégie visant à adopter largement les microservices mais que dans le même temps, elle ne dispose pas à l'heure actuelle des ressources, notamment techniques, lui permettant de mener cette opération dans les meilleures conditions.

Si l'on considère que les outils existants constituent la version 1 de l'infrastructure de l'entreprise, la mise en œuvre des microservices représenterait la version 2, c'est à dire une nouvelle version majeure qui ne serait pas forcément compatible avec sa parente.

Notre préconisation est de réfléchir dés maintenant à cette nouvelle architecture et à intégrer cette problématique dans les développements prévus.
Un accompagnement est recommandé qui peut aller crescendo au fur et à mesure de la concrétisation de l'adoption de cette approche.

Bonnes pratiques

Dépôt de code

Le code est le cœur du réacteur du système d'information. A ce titre, sa gestion doit être particulièrement encadrée afin de sécuriser son historique et ses évolutions tout en facilitant ses utilisations.
L'utilisation des dépôt de code (Version Control System ou VCS) est un préalable à toute organisation de ce type. Les plus populaires sont Subversion et Git, plus récent et plus moderne (quoique parfois plus complexe) que son ainé.
Un dépôt de code permet de stocker le code dans tous ses états tout au long de son cycle de vie. Il permet ainsi de retrouver la trace de tout ajout ou modification.
Au delà de cette fonctionnalité élémentaire mais indispensable, il ouvre la collaboration entre développeurs en permettant le partage du code stocké et la gestion des mises à jour incrémentales incluant la détection de conflits.
Enfin, il permet d'ouvrir l'accés au code à tout outil permettant de s'assurer de sa qualité et de sa conformité à un ensemble de contraintes.

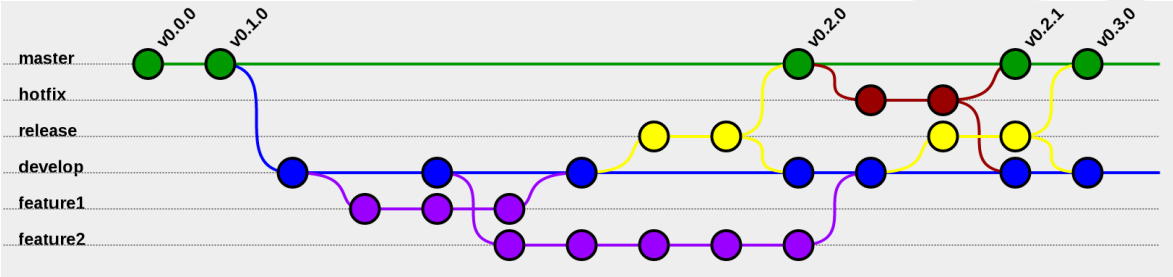
Afin de garantir l'accès à toutes ces fonctionnalités, nous recommandons de migrer le code existant et de développer le code à venir chez l'un des acteurs majeurs du marché (Github ou Bitbucket) en utilisant le protocole Git.
Ces sites offrent des formules peu onéreuses d'hébergement ainsi qu'une offre de services annexes en perpétuelle évolution.

Les avantages sont nombreux :

- le dépôt de code est directement géré par Usitab, indépendamment de ses fournisseurs existants ou à venir,
- Usitab peut ainsi définir les droits d'accès de chaque développeur à chacun des projets hébergés
- Ces outils proposent des services de création de Wiki intégré ou de gestion des tickets
- Ces outils proposent un Git Flow permettant de gérer les versions du code
- Ces outils sont extrêmement populaires auprès des développeurs

Git Flow

Un Workflow Git permet de définir les règles de développement et de livraison du code. Il met en œuvre différentes méthodologies permettant de s'assurer de la cohérence des phases de développement et de faciliter l'intégration du code ajouté ou modifié.



Il est particulièrement utilisé dans les équipes tirant partie des méthodologies de développement dites « Agile » et notamment Scrum.
Le travail par branche permet d'isoler temporairement le nouveau code tant qu'il n'est pas stable et ne fait pas l'objet de suffisamment de tests. Cela permet de dissocier également les fonctionnalités des versions et de ne publier que ce qui est prêt à une date donnée (principe du « Sprint » en Scrum ou un temps de développement défini donne lieu à une version qui ne comporte que les fonctionnalités abouties)

Tests

Tester une application permet de s'assurer que son fonctionnement est conforme aux spécifications. Ainsi, les tests doivent être réalisés chaque fois que le code change.
Trois niveaux de tests existent :

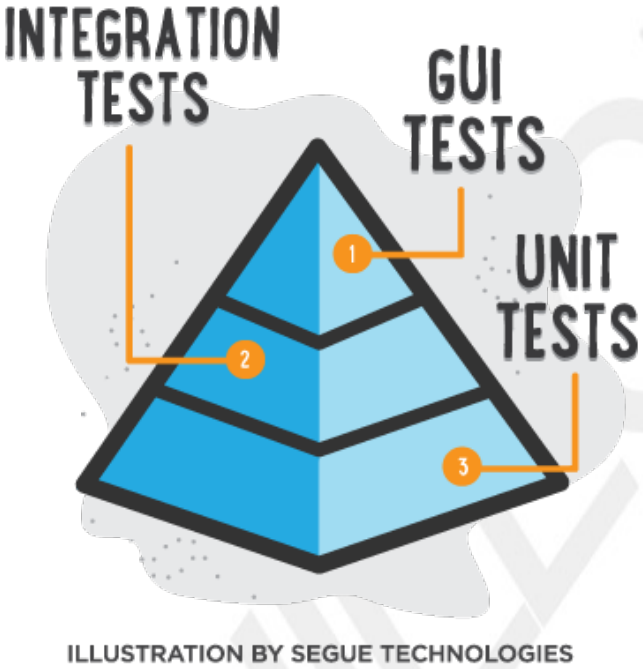
- tests unitaires (Unit Tests)
- tests d'intégration (Integration Tests)
- tests d'acceptation (Acceptance Tests ou GUI Tests)

Tests unitaires

Dans le paradigme de la programmation objet, on représente les éléments manipulés dans une application sous forme d'objets. Par exemple, dans une application de gestion commerciale, une facture sera un objet, les lignes de cette factures des objets également,

les produits ou services auxquelles les lignes font référence de objets, de même que le client, son adresse de facturation ou ses conditions de compte.

Ces objets sont décrits dans des classes. Ces classes elles-mêmes composées de propriétés (attributs) et de méthodes (fragments de logique pouvant être combinés)



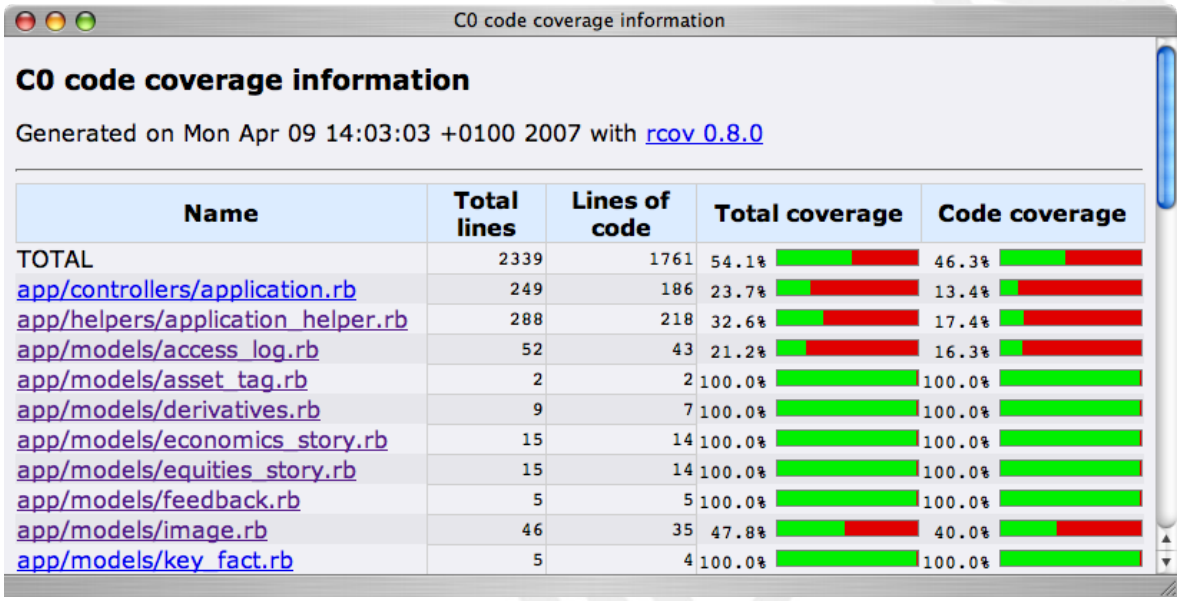
chaque élément doit pouvoir être testé indépendamment de manière très simple. Chaque méthode dispose d'une interface d'entrée lui permettant d'être appelée, notamment en lui passant des paramètres (dont la liste peut être définie en type et en nombre) et d'une interface de sortie permettant de récupérer le résultat de son traitement.

Dans le cycle de vie de la méthode, en l'absence de modification de la logique qu'elle implémente, les mêmes paramètres donnés en entrée doivent entrainer le même traitement et donc la même réponse en sortie. Le test unitaire consiste à vérifier cette hypothèse de manière certaine en utilisant les différentes combinaisons de paramètres.

L'écriture de tests, quoique simple, doit être réalisée en amont du processus de développement. Ajouter des tests à une application déjà écrite est un processus long et complexe, le plus souvent voué à l'échec.

Chaque méthode peut se voir associer un ou plusieurs tests, suivant la complexité des traitements qu'elle réalise.

L'exécution des tests permet de tracer le passage du moteur d'interprétation du langage dans les lignes de code. On obtient ainsi une analyse des méthodes couvertes mais au delà ce que l'ont appelle « code coverage » c'est à dire le pourcentage de lignes de code qui sont parcourues dans le cadre de cet exercice. Cela permet de deviner les tests manquants comme de détecter les portions de « code mort ».


C0 code coverage information

C0 code coverage information

Generated on Mon Apr 09 14:03:03 +0100 2007 with rcov 0.8.0

Name	Total lines	Lines of code	Total coverage	Code coverage
TOTAL	2339	1761	54.1%	46.3%
app/controllers/application.rb	249	186	23.7%	13.4%
app/helpers/application_helper.rb	288	218	32.6%	17.4%
app/models/access_log.rb	52	43	21.2%	16.3%
app/models/asset_tag.rb	2	2	100.0%	100.0%
app/models/derivatives.rb	9	7	100.0%	100.0%
app/models/economics_story.rb	15	14	100.0%	100.0%
app/models/equities_story.rb	15	14	100.0%	100.0%
app/models/feedback.rb	5	5	100.0%	100.0%
app/models/image.rb	46	35	47.8%	40.0%
app/models/key_fact.rb	5	4	100.0%	100.0%

L'utilisation des tests unitaires permet de s'assurer à chaque instant que les évolutions du code n'introduisent pas de régressions, c'est à dire de détecter sans délai des comportements inattendus des méthodes et des classes induisant un fonctionnement incorrect de l'application.
L'exécution de ces tests peut par ailleurs être automatisée.

Tests d'intégration

De la même manière qu'on teste le code au niveau des méthodes, on peut tester à des niveaux plus fonctionnels.

Ainsi dans le cadre du développement d'une API, on testera les différents appels possibles aux différents endpoints.

A ce niveau, on teste la qualité de la composition. En effet, les tests unitaires ont pu démontrer que chaque méthode avait un comportement conforme, la composition du code qui consiste à connecter ces méthodes entre-elles peut comporter des erreurs.

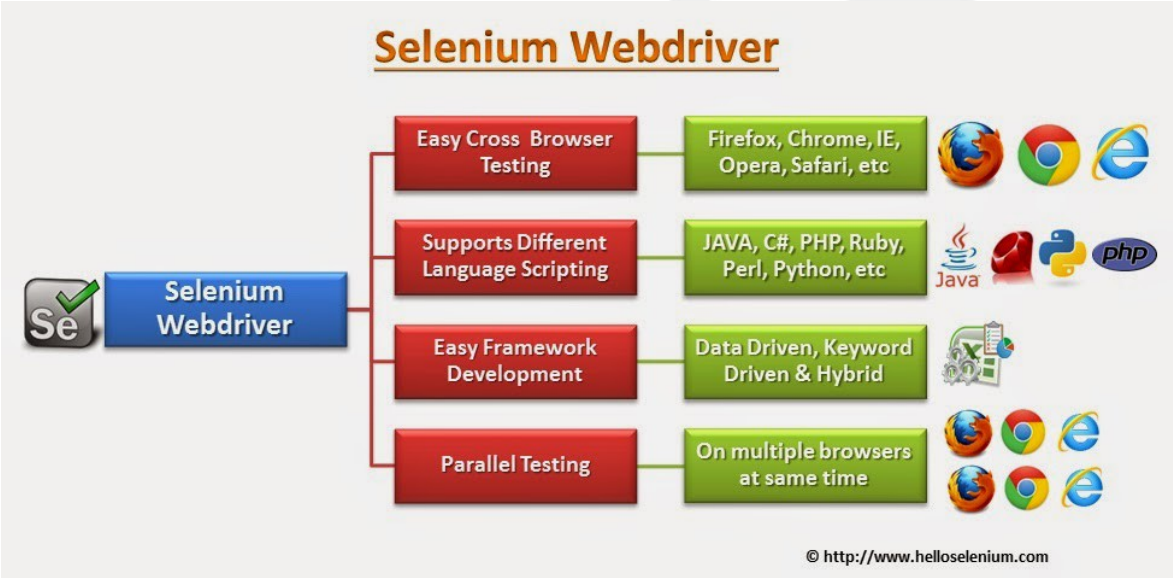
L'exécution de ces tests peut également être automatisée.

Tests d'acceptation

En l'absence de tests écrits, ce sont généralement les seuls tests qui sont réalisés et qui se résument à tester les différentes fonctionnalités de l'application accessibles aux utilisateurs via son interface.

Bien que cela soit un tout petit peu plus compliqué que pour les autres types de tests, les tests d'acceptation peuvent être codés via des plateformes comme Selenium par exemple.

Mieux, il est possible de les automatiser afin de les exécuter sur différentes systèmes d'exploitation et navigateurs de manière concurrente.



Ainsi, on peut s'assurer en lançant l'ensemble des tests que toute modification du code n'a pas d'impact négatif sur l'ensemble.

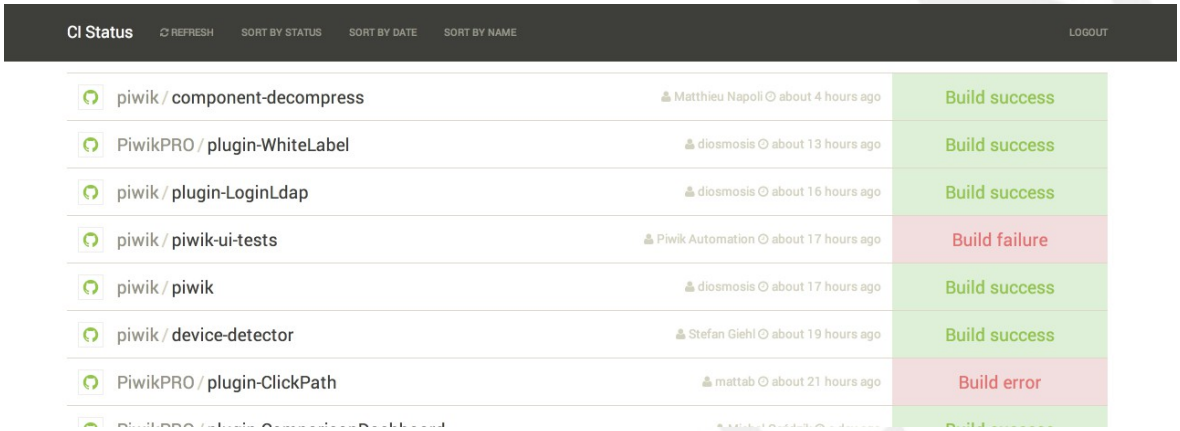
Intégration continue

On l'a évoqué, tous ces jeux de tests peuvent être exécutés de manière automatisée. Cela devient même une obligation quand les ajouts de fonctionnalités et les corrections de bogues se multiplient.

Pour se faire, des outils permettent d'exécuter automatiquement tout ou partie des tests

dés qu'une portion de code est envoyée au dépôt. Si le code « casse » l'application, il peut même être rejeté automatiquement.

Les sites comme Github ou Bitbucket bénéficient d'un écosystème d'outils d'intégration continue SaaS comme Travis ou Codeship. Il est également possible de bénéficier d'outils « on premise » comme Jenkins.



Ces outils détectent automatiquement les changements de code et lancent l'exécution des tests dans un ou plusieurs environnements dédiés.

Ils peuvent également, entre autres choses, exécuter des diagnostics sur la qualité du code ou générer de la documentation.

Déploiement continu

La suite logique de l'intégration continue consiste à déployer automatiquement le code validé en production.

Traditionnellement, des équipes distinctes sont responsables du développement, des tests puis de la mise en production.

Cette nouvelle approche des choses tend à faire du développeur un « DevOps », c'est à dire à le rendre autonome et responsable de son code de l'écriture à la production.

Cette simplification des procédures ne s'improvise pas. Pour autant, une fois instaurée, elle conduit à des mises en production plus fréquentes et plus unitaires (notamment dans une approche microservices)

Notre recommandation est de définir sans tarder une stratégie d'intégration continue qui pourra évoluer vers le déploiement continu. Il semble nécessaire de se faire accompagner dans la définition de cette stratégie et possiblement les premières étapes de son déploiement.

Autres Préconisations

Estimation et suivi de la dette technique

Le danger principal qui menace les entreprises du logiciel s'appelle la dette technique. Par ce terme, on décrit l'ensemble des verrous techniques acquis empéchant les évolutions d'un code. De manière non exhaustive, on citera :

- le non respect des bonnes pratiques de programmation object (principes SOLID)
- le non respect des bonnes pratiques de conception de base de données
- l'absence de documentation :
    - commentaires dans le code
    - diagrammes de séquence
    - documentation d'API
- l'absence de tests (tests unitaires, d'intégration et d'acceptation)
- dépendances à d'autres projets peu maintenus voire abandonnés (CMS, Frameworks)

Cette dette technique obère les possibilités d'imaginer des nouvelles fonctionnalités et complexifie également le recrutement et l'intégration de développeurs de même que le coût d'intervention de prestataires extérieurs.

Organisation d'une veille permanente

L'univers du développement web et celui de l'IoT bouillonnent dans le pays. Startups, accelérateurs, grands acteurs publics et privés, tout le monde ou presque est sur le pied

de guerre pour prendre des positions dans une industrie qui représente déjà 5 % du PIB du pays. C'est le bon moment pour nouer des contacts et des alliances aussi bien commerciales que techniques.

La France, avec son organisation très jacobine concentre beaucoup des lieux de rencontre et d'échange en région parisienne. Salons, conférences, meetups, chaque semaine des dizaines d'événements permettent de découvrir des solutions innovantes ou de précieux retours d'expérience de ceux qui ont plongé les premiers.

Il est naturellement impossible et de toute manière vraisemblablement improductif de suivre tous les sujets mais une industrie est en train de se constituer qui va refondre profondément le fonctionnement du monde et, à notre échelle, celui du pays et de ses voisins.

Notre recommandation est de définir (annuellement par exemple) les priorités de veille et de participer à au moins un événement en personne par mois. Les webinars et les conférences filmées étant de plus en plus nombreux, il est également possible d'assister à distance et en différé à des événements trop lointains ou incompatibles avec un agenda.

Développement de partenariats

La suite logique de ce processus consiste, après identification d'acteurs non concurrents de développer des partenariats. Ces partenariats peuvent être de différentes natures :

- technique : adhésion aux programmes startup (Sigfox en propose un par exemple)
- institutionnel : adhésion aux associations professionnelles (IoT, éditeurs de logiciels, etc)
- commercial : …

L'objectif immédiat est de profiter de la visibilité des partenaires pour se faire mieux identifier et connaître en revendiquant l'association avec eux. L'établissement de relations permet également de simplifier la veille en profitant d'informations arrivant directement.

RH

Recrutement d'un développeur backend sénior

On l'a vu, le backend contient les données et les algorithmes de traitement de la société, c'est un point critique qui dont faire l'objet d'une attention particulière.

Un profil de développeur sénior pouvant évoluer ensuite sur un poste de Lead Developer voire d'architecte est indispensable en interne.

Il sera chargé de concevoir, développer et maintenir les entrepots de données, les algorithmes et les différentes API. Il aura également la responsibilité de constituer et maintenir la documentatition, notamment technique.

Cette charge de travail est importante. Il est recommandé de lui adjoindre rapidement un ou plusieurs développeurs, éventuellement junior, qui peuvent être recrutés en CDI ou via contrat de prestation avec une ou plusieurs SSII. L'important étant qu'ils évoluent dans un cadre aux pratiques et outils bien définis.

On veillera à sélectionner des profils intéressés par l'approche DevOps.

Recrutement d'un profil orienté frontend

Le frontend est également un sujet critique mais la visibilité sur es développements à venir semble insuffisante à ce stade. La recommandation est de temporiser le recrutement d'un profil plus orienté frontend. Il est ainsi difficile d'affirmer qu'il s'agirait d'un développeur plutôt qu'un chef de projet, notamment si la sous-traitance des développements se révèle un bon choix technique et financier.