Institut Universitaire de Technologie,
Aix-Marseille Université

RAPPORT DE STAGE
Diplôme Universitaire de Technologie
Spécialité Réseaux et Télécommunications

Conception Cloud, Middleware d'intégration des leads en haute disponibilité

Damien RABELLINO

Viadom

Responsable entreprise : Guillaume Cyr

Responsable académique : Éric Würbel

2016

Table des matières

Remerciements

Je tiens premièrement à remercier Guillaume Cyr pour la préparation en amont qu'il a faite à mon égard, qui m'a permis une intégration rapide dans l'entreprise et une compréhension de ce qui m'étais demandé. Mais aussi pour m'avoir laissé ma chance.

Je tiens aussi à remercier toute l'équipe informatique, eux qui n'ont pas hésité à partager leur connaissance avec moi, toujours disponible en cas de besoin.

Plus particulièrement Yannick Escoffier qui m'a guidé à mon arrivée, et avec lequel j'ai pu échanger tout au long de la formation.

Je précise que tout ça n'aurait pas été possible sans l'aide de Théo Orengo, qui a œuvré pour me mettre en contact avec la société.

Plus globalement pour finir, je remercie tout simplement Viadom qui m'a permis d'effectuer ce stage dans leurs locaux.

Introduction

La technologie, c'est ce qui m'a poussé à m'engager dans un DUT Réseaux et Télécommunication. Cette formation est pour moi ce qu'il y a de plus complet dans le concept si vaste qu'est l'informatique.
Tous les grands domaines qui composent ce dernier s'y trouvent, permettant de découvrir un maximum de chose, et il est important au sortir d'un Bac+2, d'avoir une idée précise de ce que l'on vise dans l'avenir.

Aujourd'hui cette formation est pour moi synonyme de portes ouvertes, car elle permet de se diriger vers de nombreuses voies distinctes, mais surtout d'avoir le choix, et l'expérience suffisante pour pouvoir le décider.

Ce stage m'a permis d'étendre encore plus ma vision sur le futur, m'a donné de nouveaux atouts pour prendre mes décisions, au travers d'un projet structuré et innovant.

Travailler sur la technologie Cloud, qui représente aujourd'hui le futur pour nos données, a été un privilège.

En rappelant tout de même que ceci est un résumé.
Ce rapport contiendra un maximum d'éléments, et essaiera le plus justement possible de retracer ces dix semaines de travail dans le service web de Viadom.

Présentation de l'entreprise

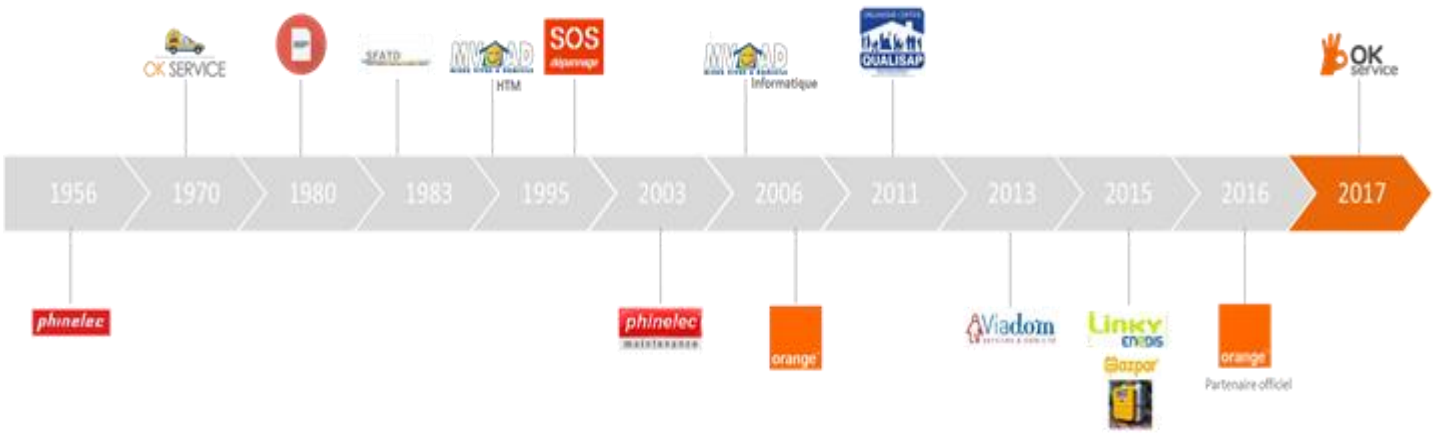
1. Historique

Viasphère est un des leaders français sur le marché des services aux particuliers et aux entreprises. Le groupe se positionne aujourd'hui comme un prestataire de services à domicile à travers six enseignes :

- Ok Service
- Viadom
- Merci+
- Menage.fr
- Phinelec Maintenance
- Phinelec

Viadom qui a été créée en 1990, à racheté le groupe OSE en avril 2013, il y a 4 ans maintenant, dans le but premier de compléter ses domaines d'activité, afin de continuer son expansion.

Historique du groupe OSE :



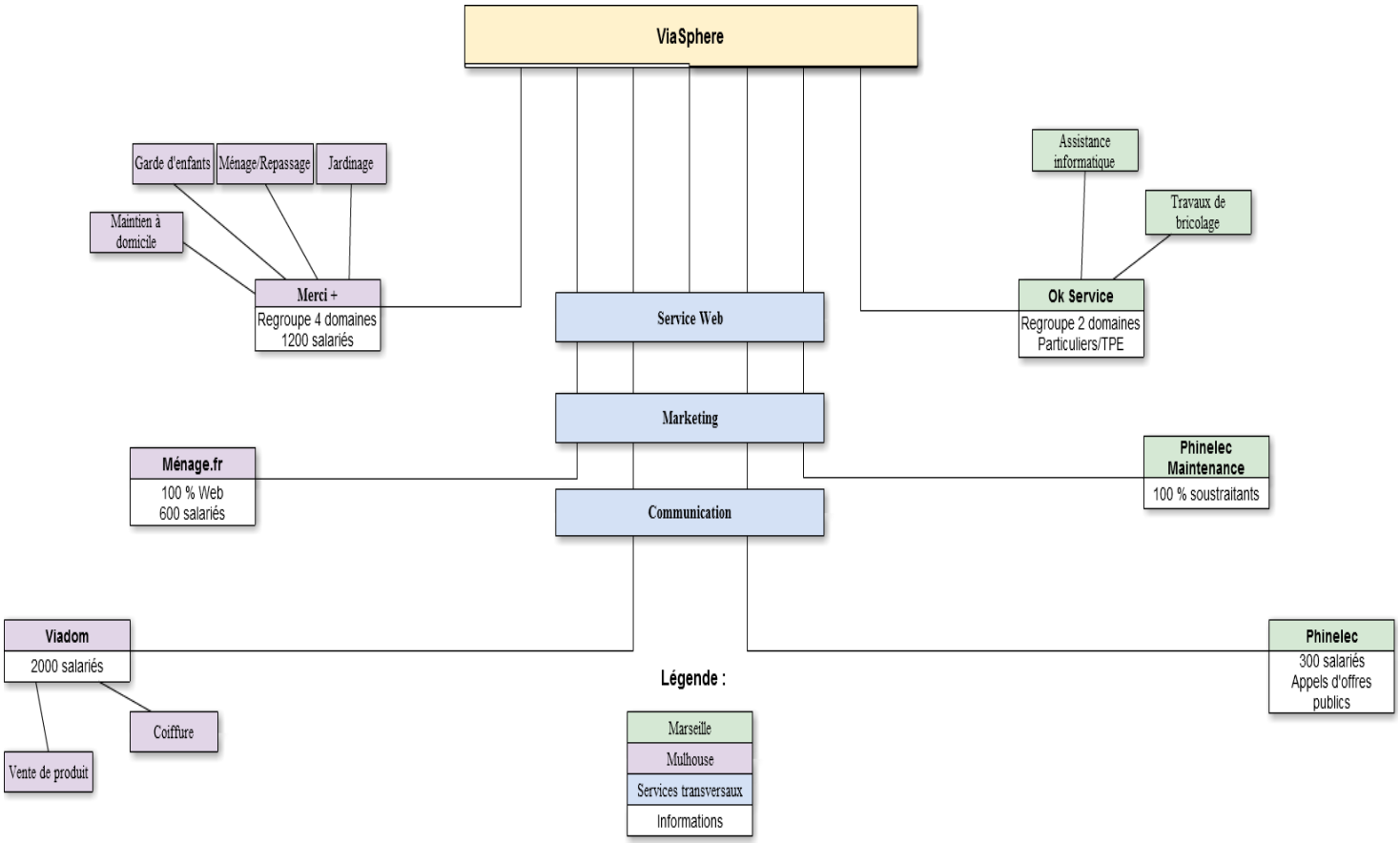
- ❑ **1956** : Création de **Phinelec** spécialisée dans l'entretien des systèmes de chauffage et chaudière
- ❑ **1970** : Création d'**OK Service**, réseau de dépannage urgent à domicile
- ❑ **1980** : Lancement du système d'**abonnement Atout Confort** destiné aux particuliers pour couvrir toute l'assistance technique du domicile
- ❑ **1983** : **SFATD**, Société Française d'Assistance Technique à Domicile.
- ❑ **1995** : Création de **Mieux Vivre A Domicile**, leader depuis 1996 de l'activité **« Hommes toutes mains »**
- ❑ **1995** : Rachat de la société **SOS Dépannage**
- ❑ **2003** : Création de **Phinelec Maintenance**, réseau de maintenance multi technique et multi-sites à destination des professionnels
- ❑ **2006** : MVAD lance l'abonnement d'**assistance informatique** et devient installateur **Orange**
- ❑ **2011** : Obtention de la première certification **Qualisap**, gage de qualité pour MVAD
- ❑ **2013** : SFATD intègre le groupe **VIADOM**
- ❑ **2015** : SFATD et Phinelec remportent une partie des appels d'offre **LINKY** et **GAZPAR**
- ❑ **2016** : MVAD devient partenaire officiel d'**Orange**
- ❑ **2017** : Relance d'**OK Service** qui devient la marque unique de l'activité Grand Public

2. Activités

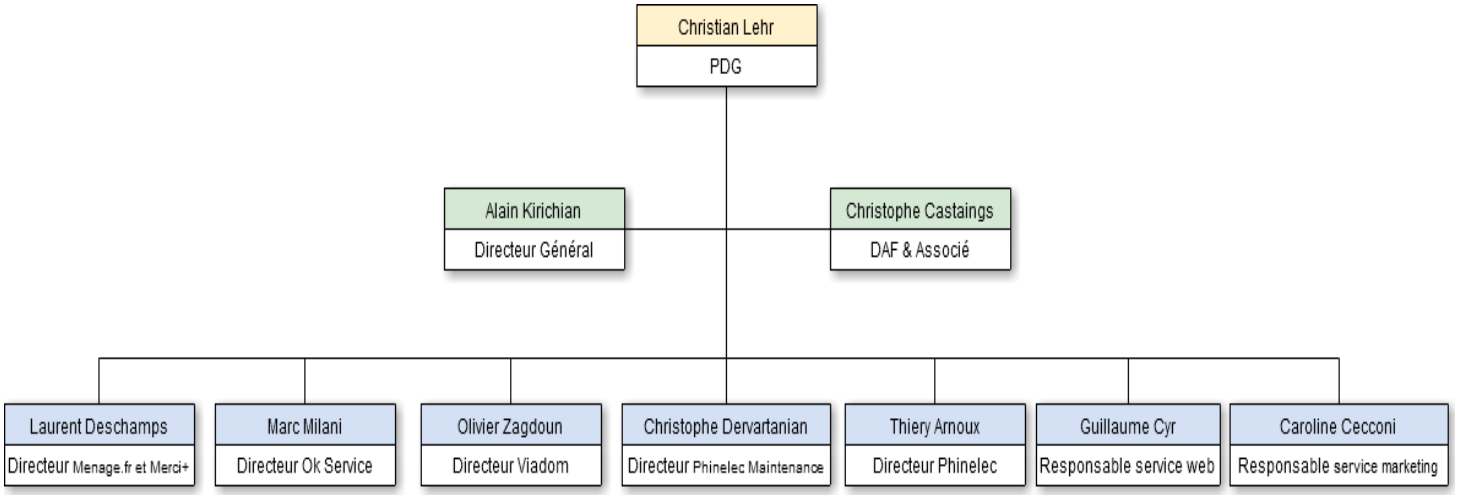
2.1 L'Organisation :

Comme vous l'aurez compris, les domaines d'activités du groupe sont nombreux et diversifiés.
Le groupe Viasphère est composé de plusieurs sociétés et d'agences locales.
Le groupe travaille avec plus de 4000 collaborateurs.

Ci-dessous un schémas général du groupe :



Direction et services transversaux :



2.2 Les services :

2.2.1 Mulhouse

- La coiffure et la vente de produits associés sont gérés par la marque Viadom, comprenant 2000 salariés.

- Menage.fr, qui se base essentiellement sur leur site web, est constitué de plus de 600 salariés.

- Et Merci +, qui gère le maintien à domicile, le jardinage, le ménage/repassage ainsi que la garde d'enfants, est composé 1200 salariés.

2.2.2 Marseille

- Ok Service
  Leurs activités consistent en l'intervention technique à domicile dans cinq domaines.
  L'assistance informatique, la domotique et le multimédia par des prestations de dépannage et d'installations.
  Ainsi que du bricolage à travers les métiers de l'électricité et de la plomberie.
  Les clients sont majoritairement des particuliers, voir TPE/PME.

  L'activité s'appuie sur dix agences locales qui proposent l'ensemble des métiers du groupe.
  Le but étant de concentrer au maximum l'activité de chaque agence pour mieux servir les clients, grâce a une proximité géographique.
  250 salariés dont 150 techniciens.

- Phinelec Maintenance, est spécialisée dans la maintenance curative et préventive des grands réseaux succursalistes. Par souci de couverture du territoire, elle a du se développer en s'appuyant sur un réseaux de plus de 3000 artisans. C'est la seule enseigne du groupe qui a construit son activité via la sous-traitance.

- Phinelec, s'occupe de la pose de compteurs intelligents et communicants Linky pour l'électricité et Gazpar pour le gaz. 300 salariés y travaillent.

2.2.3 Services Transversaux

Et enfin les services transversaux sont le service web, dont j'ai fait partie durant ce stage, le service marketing et le service communication.

Le service web accompagne les métiers dans la digitalisation de leurs activités. Il s'occupe principalement de la création des sites de chaque marque, leur mise à jour ainsi que leur sécurité. Mais aussi du développement d'applications mobiles, et la gestions de la passerelle de communication entre les différents services informatiques (interne ou externe).

L'équipe est composée de deux développeurs, un graphiste et un référant.
Ils ont d'ailleurs engagé un autre stagiaire dans une période quasiment similaire à la mienne pour pouvoir compléter leurs objectifs actuels.

3. Organisation

Je vais présenter dans cette partie uniquement l'organisation du pôle de Marseille, qui a été mon lieu de travail durant ce stage.

Il y a sur Marseille la présence d'un centre appel dédié à Ok Service, Phinelec et Linky/Gazpar.
Il permet de répondre à toutes les demandes des clients, que ça soit pour la demande d'intervention ou bien une demande d'informations. Toutes les agences, ainsi que les techniciens qui y sont rattachés, sont gérées par celle de Marseille.

Mon poste se trouve dans un "open space" dédié au service informatique de l'entreprise, sous la tutelle de Monsieur Cyr.

Présentation du sujet de stage

1. Enoncé du projet

Le projet principal qui m'a été confié ces dix semaines chez Viadom est basé sur une notion vaste qu'est l'intégration continue. Le but étant donc de faire avancer leurs recherches dans la quête de cette performance, et de mettre en place certains des éléments.
Acquérir et comprendre pour enfin transmettre, sous plusieurs formes, écrites et orales, avec divers supports, et quelques exemples fonctionnels.

2. L'Intégration Continue

L'intégration continue, dont la définition va être étoffée tout au long de ce rapport, en détaillant certains éléments qui y sont rattachés. Ce concept, apparu il y a plusieurs années, est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification du code source que le résultat des modifications ne produit pas de régression dans l'application développée.
Pour schématiser la chose, trois étapes essentielles constituent la voie vers l'intégration continue. La première consiste à tenir un code propre, c'est-à-dire le mieux structuré possible, notamment à l'aide du versioning obtenue avec git. Après cela, il faut avoir des tests automatisés, créé par les développeurs mêmes, en fonction du code, de ses besoins et de son but final.
Enfin, il faudra ajouter au processus un serveur qui va gérer l'intégration continue, il existe plusieurs services, nous en aborderont un, Jenkins, peut être le plus connu de tous, il est fort de ses nombreuses années de mise à jour, et possède une grande communauté.
Coordonner son développement permet un gain d'efficacité, et donc de temps, épargnant une étape d'intégration dite « à la main » assez longue, car elle révèle souvent des bugs, qui peuvent être évités et corrigés en amont grâce aux fameux tests.

3. Problématique

Ces 10 semaines m'auront vraiment beaucoup apporté, je ne dirais pas connaitre aujourd'hui Amazon Web Service comme ma poche, mais j'ai acquis une réelle expérience et connaissance de la chose. J'ai donc dû apprendre au cœur d'AWS grâce à leur documentation (française comme anglaise) et à la pratique ; mais pas que. J'ai dû aller me renseigner sur des bases et des concepts totalement nouveaux pour moi, pour pouvoir comprendre certains aspects, et mieux maitriser mon sujet.

Comment permettre une mise en production d'applications, de logiciels et de sites web le plus simplement possible, automatiquement et avec contrôle. Que choisir comme solution dans la masse qui nous est proposée aujourd'hui. J'ai cherché la meilleure réponse à cette question.

Présentation du travail réalisé

Pour mener a bien la mission qui m'a été confiée, mes deux premières semaines ont été une totale autoformation. Ayant reçu des instructions qui m'ont aiguillé au mieux, j'ai dû assimiler par de nombreux cours, des recherches, de la lecture de documentations techniques, ainsi que de nombreux cours et vidéos explicatives en ligne d'un certain Grafikart, qui m'ont été conseillés par l'équipe.

J'ai aussi eu la chance un peu plus tard d'avoir accès en version papier au livre AWS Certified Solutions Architect - Official study guide, livre intégralement en anglais.

Ce guide du savoir destiné principalement a préparer, pour ceux qui le souhaitent, une certification qui porte le même nom : AWS Certified Solution Architect. Son obtention prouve un niveau de maitrise de la plateforme et ces concepts avancés.

Ce guide qui contient de nombreux détails, des explications, définitions et astuces, qui m'ont permis, combinés aux documentations fournies gratuitement par Amazon, de comprendre et d'avoir suffisamment d'éléments pour pouvoir prendre en main cet "outil".

J'ai eu très vite a commencer en parallèle à tenir un glossaire, pour toutes nouvelles notions rattachées à de plus grands concepts ou bien des acronymes importants. Celui-ci m'a paru indispensable, il m'a permis d'apprendre plus vite, et surtout avec efficacité.

Partie 1 : Prise de connaissances, acquisition des outils 

1- Haute disponibilité

L'indisponibilité d'un service est un réel problème. Certains services ont besoin d'être accessible 24h/24h. Il suffit de prendre exemple sur un site web, l'idéal est qu'il soit disponible a n'importe quel moment.
Plus problématique, une chaine de production industrielle ne peut plus se passer des services informatiques qui y sont liés.
Il existe plusieurs moyens ou techniques pour parvenir à la haute disponibilité, ou qui en font simplement partie.

La sécurisation des données par exemple avec SRDF ou un système de snapshots.
La modularité du système mis en place, pour pouvoir le reconfigurer facilement et avoir un plan de secours en cas de panne.

Il faut chercher à réduire le nombre de pannes et leur durée :
La mise en place de processus adaptés permettant de réduire les erreurs, et d'accélérer la reprise en cas d'erreur, ITIL contient de nombreux processus de ce type.

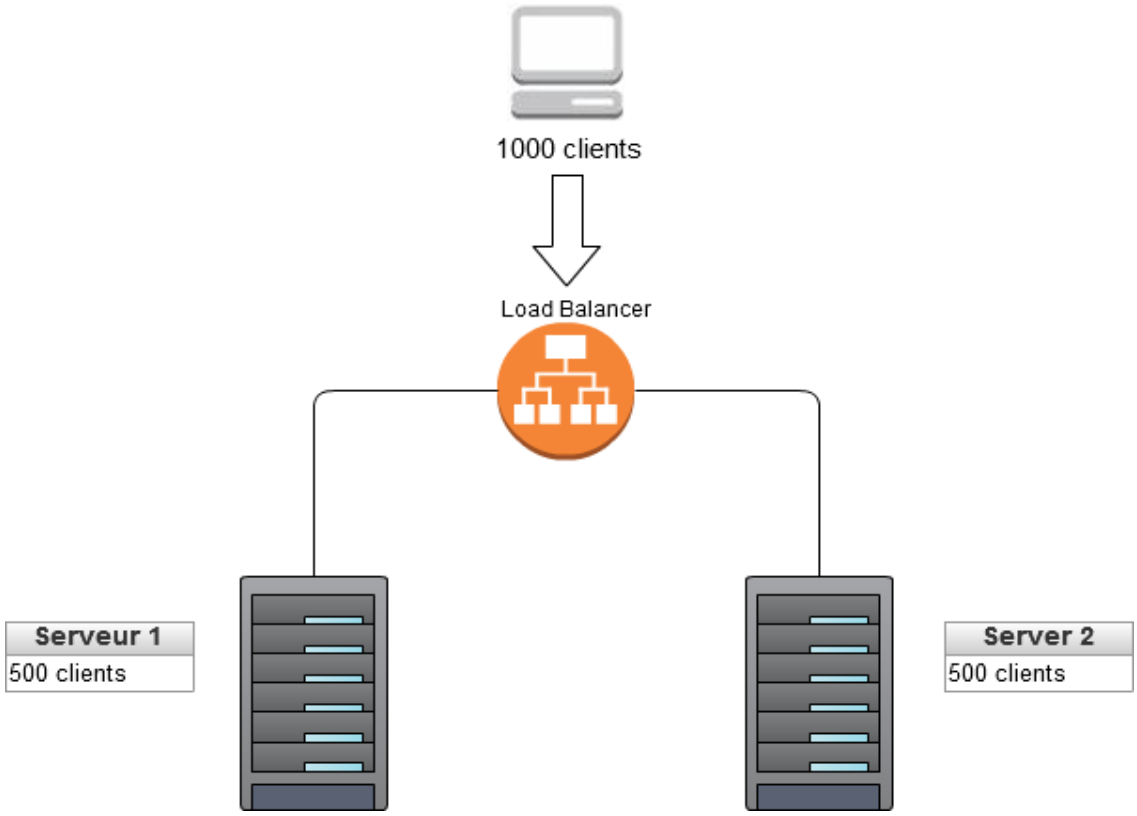
Mais le principal besoin c'est la mise en place d'une infrastructure matérielle spécialisée en se basant sur la redondance matérielle, en utilisant un cluster de haute-disponibilité : C'est une grappe d'ordinateurs dont le but est d'assurer un service en évitant aux maximum les indisponibilités.

Ce regroupement permet une gestion globale et dépasse les limitations d'un seul pc pour pouvoir :

- Augmenter la disponibilité
- Faciliter la montée en charge
- Faciliter la gestion ressources
- Permettre une répartition de la charge

1.1 Répartition des charges

La répartition des charges, ce concept fait référence à la distribution partagée pour pouvoir réduire les "efforts".
Prenons un exemple simple, 1000 clients se connectent sur le site exemple.fr :



Le but d'un load balancer, ou un répartiteur de charge, est de, selon sa politique, diriger les clients vers telle ou telle machine, dans cet exemple se trouvent seulement deux serveurs, il pourrait y en avoir bien plus.

Ces machines fournissent donc le même service, ici contiennent toutes deux le site en question, donc le client A qui se trouve que le serveur 1 on lui servira la même chose qu'au client b du serveur 2, mais attention, pas forcément de la même façon, car dans ce cas, les serveur pourrait ne pas avoir les mêmes capacités.
Plusieurs politiques de balancement sont disponibles :

- Dans notre cas, il s'agit du plus basique, le "*Round-Rodin*", qui consiste à renvoyer un client vers un serveur puis le suivant vers un autre et ainsi de suite. Le problème dans ce cas est tout aussi simple, un client une fois distribué le balancer ne s'en occupe plus. Cependant il se pourrait que depuis le début des 1000 connections il y en ait 50 qui soient déconnectés sur le serveur 1 contre 200 sur le second. Il y aurait une forte différence entre la charge du serveur numéro 1 et celle du serveur 2.

- Pour palier à ce problème voici une seconde politique, qui se base sur le nombre de connections en cours ou de requêtes effectuées, pour pouvoir assigner les opérations suivantes au serveur qui en a exécuté le moins.

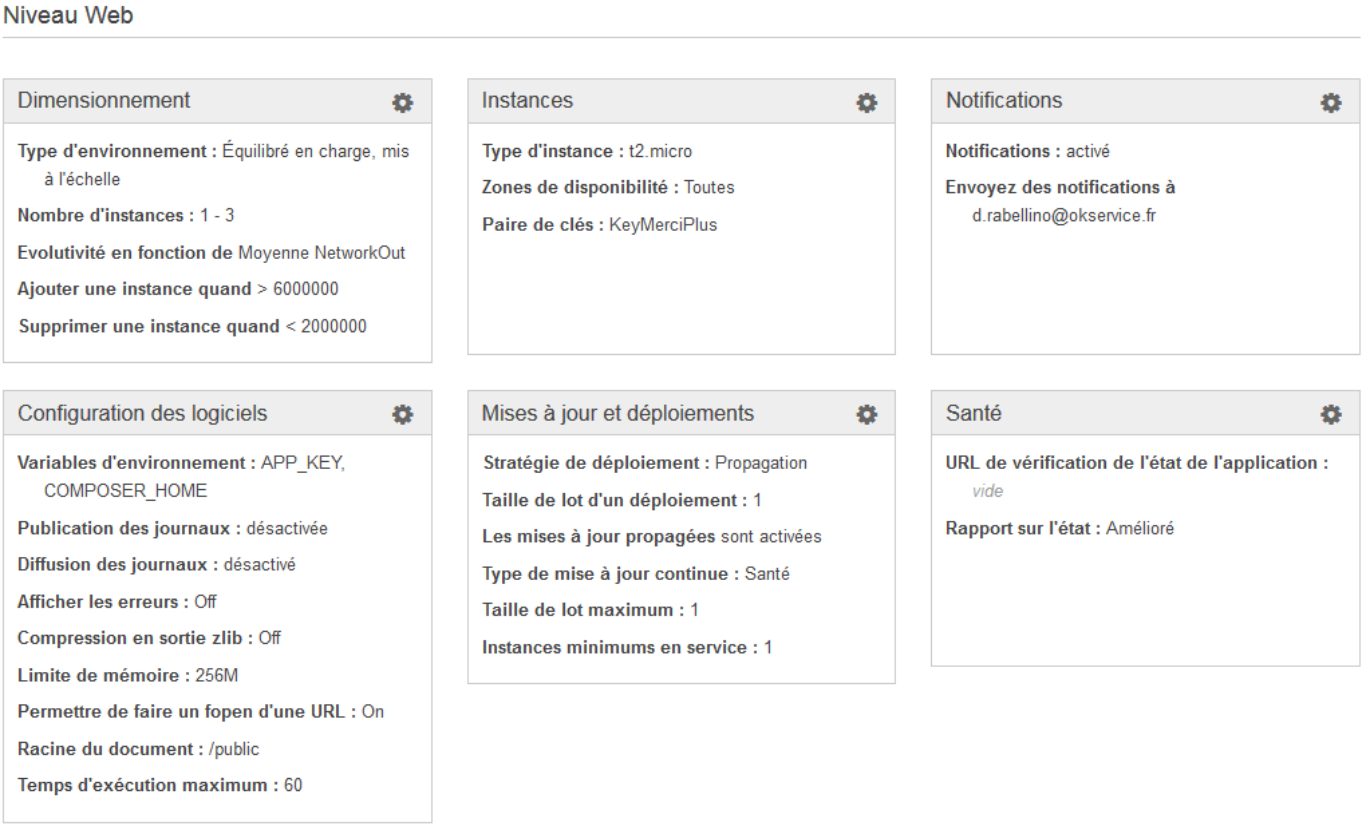
- Pour finir, j'ai évoqué le fait que les serveurs pourraient avoir des capacités différentes, pour gérer cela au mieux, il est possible de pondérer nos serveurs. C'est à dire que chaque serveur va se voir assigner un poids. les poids les plus forts vont recevoir d'avantage de charge que les serveurs de poids faibles.

2.2 Elastic Beanstalk

Elastic Beanstalk est un service qui gère le load balacing au travers d'un environnement souple est configurable, avec de nombreux paramètres modifiables, dont, par exemple, la politique de déploiement sur les serveurs qui composent le dit environnement.
EB fait partie des services que propose Amazon.
Un environnement est lié a un load balancer, et il représente une seule ou un groupe d'instances qui obéissent à des règles définies.

Ces options peuvent être très efficaces si elles sont bien ajustées au besoin. L'inverse, lui, pourrait faire faillir le service, et tout au moins serait dangereux.

Voici une partie des axes configurables :



Il y a en tout neuf grands groupes de configuration, les principaux sont :

- Le Dimensionnement, où l'on va pourvoir définir le nombre de machines minimum et maximum que pourra contenir le load balancer, ainsi que l'élément qui va déclencher le redimensionnement, c'est a dire l'ajout ou le retrait d'une instance. Régler au mieux le dimensionnement, le plus précisément possible, permet des ajouts fluides d'instances pour pouvoir assurer le service dans les meilleures conditions, malgré par exemple un fort pic de connections.

- Le type d'instances que va contenir l'environnement. Il n'est, par exemple, pas possible dans un environnement Elastic beanstalk de contenir des instances de types ou d'images systèmes différentes.

- Mises à jour et déploiement permet de configurer la stratégie de déploiement, lors d'une mise a jour du site il faut donc redéployer le contenu sur les serveurs, tout en gardant le service disponible le temps de la mise à jour.

De base la politique est dite "All at once", on va déployer la nouvelle version sur chaque machine, et mettre le service à l'arrêt.
On pourrait mieux faire, par exemple configurer l'environnement en "Rolling", on va effectuer des déploiement par lots, certaines instances sont mises à jour, pendant que les autres attendent en assurant le services avec l'ancienne version, avant de changer à leur tour.
Plus rapide, il existe un Rolling avec une variante, lorsque une mise à jour doit être faite, on commence par lancer un nouveau lot d'instances avec la nouvelle version, qui va venir une fois prête remplacer les anciennes machines d'un lot et va mettre les autres à jour. Méthode couteuse mais surement la plus efficace.
La dernière laisse une addition plus salée encore, elle consiste a déployer un nombre égal de machines à celles déjà en place, et une fois prêtes à l'emploi, on supprime toutes les autres.
Même si cette dernière est très efficace, ce n'est pas la plus rentable, en tout cas pas à petite ou moyenne échelle.

- Equilibreur de charge, dans cette partie l'on va pouvoir choisir d'autoriser HTTP ou HTTPS, mais aussi la durée de l'intervalle de vérification de l'état des instances ainsi que les seuils de vérifications des états pour juger une instance sur sa bonne ou mauvaise santé.

- le VPC lui permet de choisir s'il faut associer ou non une adresse ip publique aux instances de l'environnement ou pas, et de choisir dans quel domaine réseau, pour les adresses privées, voudrons nous que les instances soient crées.

2- Outils Majeurs

2.1 Git

Parmi tous ces nouveaux concepts, il y en a forcément des essentiels, git ou la notion de commit en font partie.
Grafikart, qui possède son propre site web, ainsi qu'une chaine Youtube où il rend accessible ses cours, est d'abord un simple passionné du web qui veut partager ses connaissances. Aujourd'hui il a prit de l'ampleur et il est connu dans ce domaine. Il propose du contenu complet de qualité professionnelle.
A l'aide de son cours structuré qui allie théorie et prise en main par le biais d'exemple et de travaux pratiques à faire en parallèle, c'est avec aisance que j'ai pu me faire la main sur Git.
Le "versioning", juste un petit rappel de ce que ça représente :

Aujourd'hui tout évolue, et tout a un historique dès sa création. Prenons l'exemple des smartphones d'aujourd'hui, les plus connus ont des versions qui sont facilement reconnaissables de par leur nom, le téléphone de troisième génération est appelé téléphone 3. mais avant lui il a eu sa version 1 et 2, voir peut être même plus.

Pareil pour le logiciel que vous utilisez depuis une quinzaine d'années, et qui s'est vu connaitre des centaines de versions différentes en passant de la v1.0.0 à la v32.64.2.
Même nous, nous avons des versions, mais la on va partir dans un débat philosophique.
Pourquoi donc donner une version à des logiciels, des sites, des jeux, voir même a des objets.
Ce versionnement représente un historique des modifications qui ont été apportées permettant un suivi structuré de la progression du concerné.

Le principe de base consiste à connaitre l'état à la version 1 et à la version 2, pour pouvoir les comparer premièrement.
Il se pourrait aussi que la version 2 ne sois plus souhaitable et qu'il vaudrait mieux, pour une quelconque raison, revenir à la version précédente.
Git va nous permettre de gérer ces historiques et les contrôler, pour pouvoir les manipuler à souhait.
Cet outil est en réalité le logiciel de gestion de versions le plus populaire.
Une fois installé, git donne accès a de nombreuses commandes, que j'ai appris à utiliser.
Mais git c'est toute une structure et des règles à connaitre, en premier lieu il faut l'initialiser dans un projet pour pouvoir utiliser ses capacités.
Un dossier caché portant le nom de git sera alors ajouté à votre projet. Ce dossier contient justement l'historique, ainsi que bien d'autres choses utiles comme le fichier gitignore, qui va permettre de masquer certains fichiers voire dossiers du projet par exemple.
Juste une chose importante ici c'est la structure à utiliser avec git pour en tirer un maximum d'efficacité.

Le schémas ci-dessus représente cinq branches, avec leur historique respectif.
La force de git est là, dans les branches. Git flow c'est la stratégie à adopter pour réussir avec git.

Travailler sur un projet à plusieurs, quel qu'il soit, sans structure devient rapidement un problème. Chez les développeur c'est vraiment ce qu'il y a de pire, ses phases d'intégration ou il faut coordonner le code de chacun pour ne pas faire tout planter en dépendant de la partie des autres. Difficile de maîtriser.
La solution se trouve dans ces branches. Avoir simplement une branche, une échelle de version, sur laquelle tout le monde va donc travailler et modifier des choses à tout bout de champ, finir par se retrouver avec un nombre incalculable de versions, et être un peu perdu dans tout ça, c'est une perte de temps considérable.

Alors un peu comme le principe du brouillon, la première chose à faire c'est d'avoir une seconde branche, sur laquelle on va pouvoir travailler et faire beaucoup de modifications sans craintes, comme ça on peut distinguer la version propre, livrée et publique et donc en cour d'usage, de la version en construction, en phase de développement.
On a donc deux branches principales, Master, qui est destinée à la version "stable", et Develop, qui contient la prochaine version, pour l'instant en développement.

Mais ça ne va pas suffire.
On voudrait avoir une phase de test concrète avant de déployer une nouvelle version, pour permettre au besoin de faire les derniers correctifs.
La liaison entre nos deux branches majeures va se faire par le biais d'une troisième appelée Release.

Dans l'idéal, la structure retenue aujourd'hui possède deux branches supplémentaires :

Hotfix, pour permettre de corriger des bugs de dernière minute sur Master, et enfin Features, qui est la sous branche de Develop, consacrée au développement des nouvelles fonctionnalités

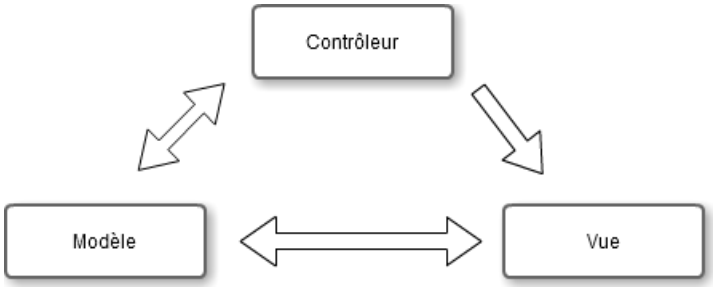
2.2 Le framework Laravel

Avant de passer à la suite, parlons du framework utilisé par le service web pour leur site récent, Laravel.
La force de Laravel c'est sa structure de projet super accessible et puissante à la fois.
Un projet de ce type est doté d'une multitude de fichiers préconstruits et de fonctions basiques déjà prêtes.

En plus Artisan, l'interface de ligne de commande intégrer à Laravel permet de créer des fichiers selon des modèles préconstruits.

Tout est fait pour simplifier des taches fastidieuses de structuration de projet web, pour laisser place au réel développement des sites.
A noter que Laravel utilise Eloquent comme ORM.

Ce framework fonctionne sous un modèle MVC pour Modèle Vue et contrôleur.
On va parler de trois couches.



- Un contrôleur réalise des actions basiques comme lire des données (dans un modèle) et les afficher (dans une vue), il créer le lien entre les deux autres couches.

- Le modèle : Chaque fois que l'on va vouloir créer, modifier, lire ou supprimer une données, une fonction spéciale sera appelée suivant le résultat voulu. Essentiellement des appels de fonctions, même si l'on peut retrouver des requêtes dans le modèle.

- la Vue : Elle génère le code HTML. Le contrôleur n'affiche jamais directement les données, une page sera appelée pour l'affichage. Permet de séparer HTML (la vue) du reste.

3- Amazon Web Services (AWS)

Tout naturellement tourné vers Amazon web services, leader dans son domaine. Des recherches avait été effectuées avant moi, qui ont permis de conclure que AWS était la meilleure solution.
Je l'ai confirmé en rapportant qu'Amazon est en avance sur les autres, et propose une multitude de services, comme j'avais pu m'en faire déjà une petite idée dans le cours, très court malheureusement, qui avait été donné à l'IUT.

Après toutes les notions déjà citées jusque là, d'autres sont venues se rajouter au fur et à mesure à mes besoins de connaissance.

On va essayer de faire le tour des principaux services, sachant que l'on a déjà évoqué EB.

Commençons par Docker, ou conteneur en français.
Un conteneur n'embraque pas de systèmes d'exploitation comme le fait une machine virtuelle classique. Un conteneur est donc bien plus léger.
Son faible poids permet un déplacement rapide d'une machine à une autre.
Il peut exécuter une seule tache/application.
J'en parle, car ce concept se rapproche de ce que sont les instances d'Amazon, surtout au vu de leurs utilisations.

CodeCommit, qui est tout simplement lié à Git. commit est une commande primordiale de Git, car c'est cette commande qui permet l'envoi des nouvelles données mises à jour.
CodeCommit est rattaché a une application/projet dans AWS. Lorsque l'on y accède on y trouve les fichiers déjà commit ainsi que quelques informations relatives à Git, mais surtout l'URL accessible par HTTPS. A la création d'une application, il est généré un identifiant et un mot de passe à utiliser lors d'un envoi vers CodeCommit.

L'intégration continue se compose d'un outil qui surveille les modifications de code dans votre gestionnaire de configuration. Dès qu'un changement est détecté, l'outil en question va automatique détecter le changement, et va compiler et tester le code.

Si la moindre erreur survient, l'outil va immédiatement avertir à l'aide de messages d'erreurs, voire de notifications, afin qu'ils puissent tout de suite corriger le problème.
C'est d'un serveur d'intégration continue dont-il s'agit ici. Jenkins, c'est lui qui va effectuer des tests pour contrôler la qualité du code. On en reparlera un peu plus tard.

Elastic Beanstalk intègre directement certains services, les Elastic Load Balancer en font partie, mais il contient aussi le service CodeDeploy, qui est en fait le service qui gère les mise à jour des instances suite a une nouvelle version de l'application.

CodePipeline, pièce maitresse, qui lie le commit, Jenkins avec les tests unitaires et le déploiement dans cet ordre là. C'est un service d'intégration et de diffusion continues. Il représente la modélisation et l'automatisation des logiciels.
Un pipeline comporte une série d'étapes, chaque étape effectuant leurs propres actions.
L'exécution des ces étapes peut être réfléchie afin d'exécuter, si possible, des taches en parallèle et augmenter la vitesse de travail.

J'ai vu des idées toutes neuves comme le Bleu Ocean de Jenkins, une plateforme permettant de visualiser ses pipelines et ses tests, de les réorganiser à souhait.
IAM est un service de contrôle qu'il ne faut pas oublier.

Amazon est souvent à jour, malheureusement certaines documentations ne suivent pas la cadence, alors on peut trouver certaines incohérences avec les versions actuelles de la console notamment.

Etant basé en Irlande sur Amazon, nous n'avons pas accès par exemple aux surveillances de facturations, qui permet de définir un seuil de dépense par mois, ou une alerte va être envoyée lorsque le dit seuil sera atteint.

Amériques | **Europe/Moyen-Orient/Afrique** | Asie-Pacifique

Services proposés :	Irlande	Francfort	Londres
AWS Certificate Manager	✔	✔	✔
AWS CloudFormation	✔	✔	✔
AWS CloudHSM	✔	✔	
AWS CloudTrail	✔	✔	✔
AWS CodeBuild	✔	✔	
AWS CodeCommit	✔		
AWS CodeDeploy	✔	✔	✔
AWS CodePipeline	✔	✔	
AWS CodeStar	✔		
AWS Config	✔	✔	✔
AWS Database Migration Service	✔	✔	✔
AWS Data Pipeline	✔		
AWS Device Farm			
AWS Direct Connect	✔	✔	✔

Nous sommes Basé en Irlande car elle possède un avantage majeur:

En tout 90 services sont disponibles dans le monde, mais chaque région n'a pas accès à tout les services, du moins pas encore.
Le 19/05/2017 la région Irlande possède 21 services de plus que Francfort, et 26 de plus que Londres, dont certains, comme vous pouvez le voir, ci-dessus qui sont essentiel à notre architecture.

Il faut noté qu'AWS arrive bientôt en France, il va y avoir une région Paris, dernière région qui va faire son entrée dans celles annoncées en 2017.
Il y en avait cinq dont la Chine (disponible seulement en chine), l'Ohio, Londres, Canada, et donc Paris pour la France.

Toutes celles cités sont déjà disponibles sauf Paris.
Il faudra donc être attentif, voir les services qui vont être disponible ou non et la différence de fluidité (Paris plus proche que l'Irlande).

Petite parenthèse, j’ai pu assister à une réunion de projet de développement en entreprise, le but étant la mise en place d’un nouveau site web ainsi qu’une adaptation mobile. Intéressant de voir comment communiquent deux équipes de développeurs distantes, comment définir les taches, l’organisation et la répartition d’un tel travail.

A la fin des deux premières semaines, j’avais en quelque sorte droit à ma réunion personnelle, pour suivre l’avancée de mon apprentissage et de ma compréhension des éléments. Ces entretiens avec mon tuteur m'ont permis de poser mes questions, important pour cerner ce qui m’était demandé, et pour poursuivre une nouvelle semaine dans le sens de la réussite.
Ces réunions ont bien évidement continué tout au long du stage, mais il n’y avait plus de formalité, c’était plus naturel et tout simplement j’étais intégré à l’équipe.

Je me suis attaqué à l’écriture d’une doc (que je vais mettre à jour par la suite tout au long du stage), contenant des explications d'usage technique des services Amazon, ainsi que deux éléments importants du projet, l'évaluation des coûts et l'architecture (j’ai donc esquissé un croquis d’architecture ainsi que fait des simulations et tests d’achats pour commencer à avoir une réelle idée du prix, discuté de l’architecture des besoins et des choix possibles comparé aux contraintes techniques et en ayant toujours pour idée d'optimiser au maximum pour atteindre le fonctionnel à moindre coût, en se positionnant sur ce qui va être réellement utilisé comme ressources dans la pratique et ce qui est susceptible de moins l'être).

Suite à un deuxième échange nous avons convenu de certaines modifications ainsi que la suite des choses, la mise en place des instances et des services pour mieux les comprendre, et apprendre notamment comme mettre en place et structurer sur AWS.

Il faut réfléchir d'abord à une architecture.

Partie 2 : Mise en place

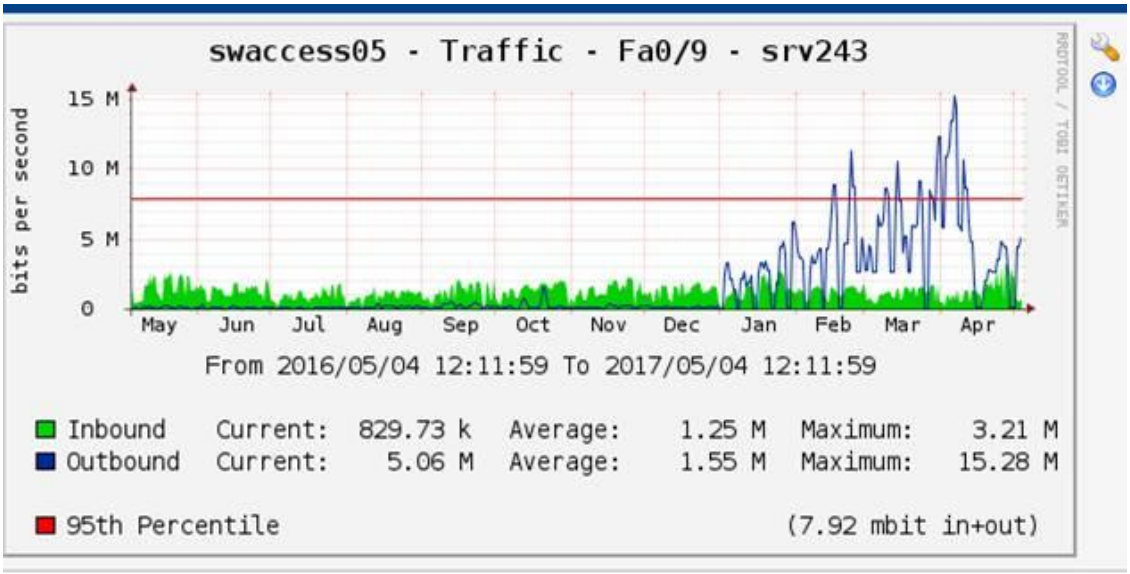
1- Architecture et coût

Je me suis aussi inscrit sur Cacoo, outil que je ne connaissais pas non plus, et que j'ai trouvé en cherchant comment faire des schémas pro avec logiciel. Cacoo m'a plu, simple et accessible partout puisqu'il ne s'agit pas d'un logiciel mais plutôt d'une plateforme en ligne gratuite. Petit atout, il possède des éléments tirés d'AWS, comme des icones qui font référence aux services, oui tout simplement les même éléments qu'utilise Amazon pour ses schémas.
Parfait pour moi et mes besoins.

J'ai donc avec cet outils, une fois compris son fonctionnement, fait une nouvelle architecture plus propre et à jour, en la complétant avec de nombreux détails, notamment avec des adresses publiques, le choix des différents types de stockages, le découpage clair de chacune des zones avec les connections possibles entres ces zones, ainsi que les services valides et disponibles.

Suite à un échange et une phase de réflexion, j'ai effectué de nouvelles simulations d'achats, en rapport à la nouvelle architecture, et à quelques ajustements. j'ai pu fixer un premier prix avec plus de précision, notamment avec un prix pour chaque zone.
Cependant cela n'est pas encore représentatif de ce que sera la réalité. Je n'ai pas pris réellement en compte le coût de "data transfert", je manquais d'information concernant la boite à ce sujet.
Donc après ma première version de l'évaluation des coûts j'ai logiquement était sollicité pour de nouveaux détails, et l'on m'a transmis les informations nécessaires, des valeurs de l'entreprise sous formes de graphiques, ou il a fallu passer du temps, le tuteur également, sur savoir comment les interpréter le plus justement possible.

Voilà les types de graphiques qui m'ont été fournis :



Le 95th percentile correspond à la valeur maximum prise sur 95% des valeurs les plus faibles. C'est-à-dire que la valeur max qui devrait se trouver au 100ème % normalement n'est pas prise en compte. Ici les 5% des valeurs hautes ne sont pas comptabilisées.

Une fois leur consommation de data étudiée et évaluée j'ai pu utiliser ces connaissances pour faire des simulations toujours plus précises.

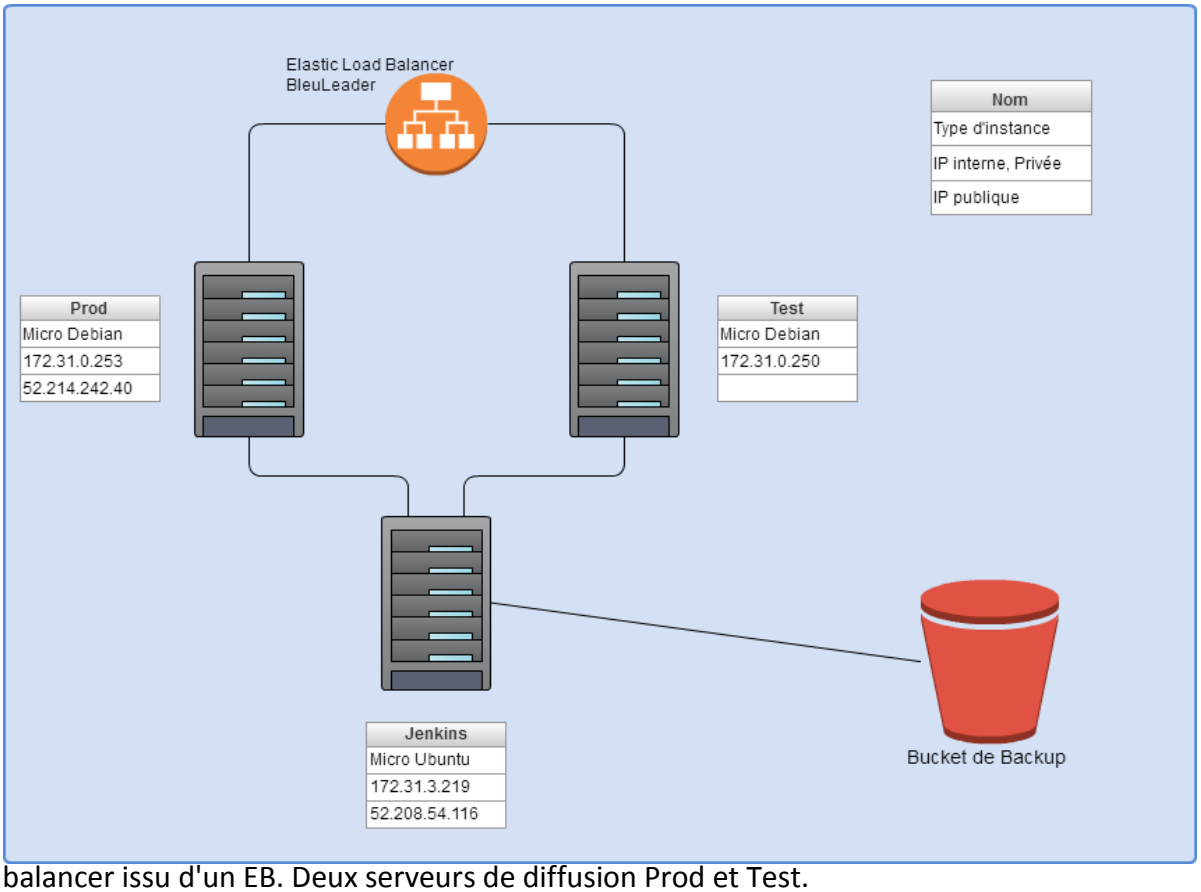
L'architecture a évolué tout au long du stage étant donné qu'il y a eu des découvertes, des utilisations de méthodes et d'autres services qui ont poussé à des changements.
Voici un exemple de choix d'instances dans le simulateur :

Compute: Amazon EC2 Instances:

	Description	Instances	Usage	Type	Billing Option	Monthly Cost
⛔		3	100 % Utilized/Mo	Linux on t2.micro	3 Yr All Upfront Res	$ 0.00
⛔		1	100 % Utilized/Mo	Windows on t2.small	3 Yr All Upfront Res	$ 0.00

Ici je veux faire remarquer que j'ai réservé ces instances pour trois ans. Lors de l'évaluation des coûts, je suis parti de ce principe là. Cette réservation permet d'obtenir le prix le plus intéressant. L'inconvénient c'est qu'il faut bien prévoir son coup, une machine qui ne sert à rien pendant trois ans c'est de l'argent perdu.

Je ne vais pas montrer l'architecture complète, mais seulement une zone que j'ai mise en place pour y effectuer des tests :



Un load balancer issu d'un EB. Deux serveurs de diffusion Prod et Test.
Un serveur Jenkins qui opère dans son étape du pipeline (voir pipeline simple en annexes).
Un serveur de stockage (Backup). Et des IP pour les rendre accessibles.

2- Instances, configuration

J'ai mis en place une première instance simple (les instances sont appelées EC2) le 24/04 avec un linux de 2 VCPU et 4Go de RAM et en y implémentant le service CodeStar qui est une plateforme de vision d'information un peu identique à BitBucket sur la forme. Elle permet notamment de créer rapidement un projet où l'on a à choisir une trame, par exemple donc mon cas j'ai choisi donc du développement web en PHP avec Laravel et CodeDeploy (et pas EB), sachant que CodeCommit n'est pas un choix, il est lui compris d'office. CodeStar permet d'intégrer facilement notre projet aux autres services d'AWS. Il permet même une gestion d'utilisateur et de droits au sein du projet.

Projet créé, j'ai pu tester, et mettre un place une simple page web, fait des changements dessus et à l'aide des connaissances acquises précédemment (Laravel et git) j'ai livré mes changements au serveur et en temps réel j'ai pu observer sur la plateforme de CodeStar les étapes effectuées avant de déployer la nouvelle version donnée. J'ai en parallèle rédigé une doc sur l'installation l'utilisation et le fonctionnement de CodeStar (voir CodeStar en annexes).

Surveillance d'instances avec CloudWatch :
Ce service permet grâce à 14 métriques (en versions gratuites), d'obtenir des informations et de contrôler ces instances, et ses environnements. (voir CloudWatch en annexes)

CloudWatch propose aussi un système de journalisation (de log), qui permet de dater et d'archiver tout évènement, d'un changement d'état, un simple get sur un des serveur ou bien des résultat de Jenkins.

3- Jenkins

Je me préparais aussi à côté en apprenant Jenkins et ses concepts, en cherchant les détails intéressants qui seraient liés à mon projet. Grace au livre Le guide complet Jenkins, le site dédié à Jenkins et quelques recherches à son sujet, j'ai pu en apprendre suffisamment pour pouvoir commencer.
Tellement cet élément est important j'aurais pu y consacrer une partie entière du rapport, mais je n'aurais pas pu parler de tout. Une chose est sure il faut garder à l'esprit que c'est le cœur de l'intégration continue. Une fois mis en place avec la configuration adéquate, Jenkins détectera les erreurs d'intégration, et si cela est prévu, les corrigera automatiquement.

Il faut que je mette en place des tests unitaires avec Jenkins, encore un nouveau concept qui m'était totalement inconnu avant ce stage. Ces tests constituent une étape importante de l'intégration continue.
Encore un bon nombre de documentation à lire et à assimiler, dont le fameux livre AWS Certified Solutions Architect – Official study guide.

Les tests unitaires consistent à mettre à l'épreuve individuellement les composants d'une application. On pourra ainsi valider la qualité du code et ses performances.

PHPUnit est à utiliser dans le serveur pour faire appel aux tests écrits en amont.
Il y a trois étapes à suivre pour la construction d'un test :

1 - Initialisation des données
2 - Agir sur ces données
3 - Vérifier que le résultat soit conforme aux attentes

Il est important de bien organiser les tests pour pouvoir s'y retrouver.
Des helpers pour intégrer les tests dans une application réalisée avec Laravel, sont fournis par le framework.

J'ai installé la dernière version de Jenkins, et commencé à le prendre en main et en faisant des tests. Par la suite j'ai lancé une instance destinées à être le serveur Jenkins, en tout il y a eu des dizaines de machines qui ont été crées dans ce but.

Des problèmes de connections par powershell et ssh en ligne de commande sont survenus.
J'ai finalement décidé de me connecter à l'aide de PuTTY, sauf qu'une fois l'installation de Jenkins faite sur la VM, impossible d'accéder au serveur en utilisant le navigateur et le port 8080.
Essaie de changement de port sur le 8443, en modifiant le fichier /etc/sysconfig/jenkins.
Le firewall m'empêchait tout simplement de passer.

Je me suis attaché avant à faire des modifications sur les instances en cours d'exécution, pour changer la condition de commit et essayer de le faire passer par BitBucket.
Le passage par Bitbucket pour le Deploy a soulevé bon nombre de problèmes, CodeStar n'étant pas adapté a passer par ce dernier, la solution de supprimer le projet lié à CodeStar pour reprendre proprement avec Bitbucket m'est venue, étant la plus simple.
Sauf qu'il aurait était fort dommage d'abandonner CodeStar, très intuitif et complet, cette interface facilite la vie.
Et puis placer Bitbucket comme le moyen de Deploy sur un projet CodeStar, que je croyais simple dans le fond, et qu'il suffisait de dire au pipeline du projet que l'étape de déploiement se fait en passant par BitBucket, en théorie, n'est pas exact.
Avec Monsieur Cyr nous avons finalement convenu à ce sujet que codeStar était utile, et qu'il valait peut être mieux laisser BitBicket de côté. Chose dite, chose faite.

Revenons à Jenkins, une fois le pare feu passé j'essaie une première installation de Jenkins, en vain. Après avoir suivi autant de tutoriels d'installation différents que de machines utilisées, plusieurs types d'instances (niveau capacité), plusieurs systèmes d'exploitations, et différentes versions et images de ces systèmes, c'était devenu compliqué.

La joie fût grande, quand est venu le moment de la réussite, le problème était vaincu, lui qui m'a coincé plusieurs jours.

En effet, le problème venait bien des dernières versions des systèmes disponibles pour les instances, ou de la documentation, et les instructions à suivre n'étaient plus au gout du jour, menant à certaines erreurs, et l'inaccessibilité du service.
J'ai donc mis en service une nouvelle instance, en changeant l'image système par celle-ci : bitnami-jenkins-2.5-0-linux-ubuntu-14.04.3-x86_64-hvm-ebs (ami-0067f373)
C'est une image issue de la communauté, ce n'est pas une officielle d'Amazon, ce qui explique que je ne l'ai pas choisie avant un certain temps.

Une image qui se veut donc faite pour Jenkins. Une fois l'instance lancée et une connexion ssh réussie, j'effectue les paramétrages basiques, et installe Jenkins. Aucun problème rencontré sur cette version d'Ubuntu, je lance le service et après une longue attente, il fonctionne.
Je m'empresse de mettre à télécharger tous les plugins essentiels, et configurer le premier utilisateur. C'est bon je suis lancer pour la partie test unitaire.
J'ai effectué une petite redirection du port 8080 vers le port 80, histoire que l'on ait pas besoin de stipuler le port pour accéder à Jenkins par le web:

```
#Ce qui vient de l'extérieur
iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8080
#Ce qui vient de la machine
iptables -t nat -I OUTPUT -p tcp -d 127.0.0.1 --dport 80 -j REDIRECT --to-ports 8080
```

Configurer Jenkins de prime abord et l'associer à Pipeline paressait facile. Le pipeline ne m'a pas indiqué une quelconque erreur quand je lui ai intégré Jenkins, tout semblait être sur la bonne voie.

J'ai alors mis en place des tests, de simples tests que j'ai rajouté dans mon projet Laravel.

Jusqu'au moment du premier test sur Jenkins, il s'agissait d'un rake pour être précis, une sorte de make (il faut dire que c'est l'exemple pris par amazon eux même tout simplement).

J'ai choisi ça sans faire réellement attention à ce que je fessais, mais il y a deux raisons, je cherchais à mettre en place un test de PSR. Je suis donc tombé simplement sur rake, et cherchant juste un test pour me permettre de savoir si le serveur fonctionnait je n'ai pas cherché plus loin.
Mais, et c'est la deuxième raison ; il y avait des bibliothèques et des modules à rajouter, et je trouvais intéressant l'idée de tester le serveur sur ce point-là, et de mon côté me faire découvrir encore un peu plus, sans compter le travail à effectuer qui m'a bien permis de reprendre la main sur les configurations, ou plus l'administration de serveur linux.
Le truc inutile c'est que rake vient du langage de programmation Ruby, alors que par la suite je vais effectuer des tests pour php.

Après avoir configuré ce qu'il faut pour utiliser rake, y a plus qu'à envoyer un commit, et voir ce qu'il se passe dans le pipeline. Le commit se déroule bien, on passe à la partie test de Jenkins et la rien ne se passe, et j'aurais pu attendre une heure que rien n'aurait bougé, au-delà de ce temps une erreur apparait de toute façon pour time out.

Je vais voir ce qu'il s'est passé sur le serveur, que j'avais configuré de façon à ce qu'il effectue un test chaque minute. Une erreur, surement dans ma traduction d'une documentation anglaise mais aussi de l'adaptation d'un nouvel agencement des paramètres de Jenkins (voir Annexes).

Dans tous les cas, j'en étais conscient et me suis dit que ça me permettait de voir comment le serveur réagissait quand il avait rien a traiter.

Revenons au processus, je regarde donc ce qu'il se passe dans Jenkins, sachant que le serveur a dû recevoir le commit. Des messages d'erreurs apparaissent, m'indiquant non pas qu'il ne récupère pas les fichiers du commit, mais qu'il ne trouve pas de rakefile. Tout ce que j'en comprends c'est qu'il manque donc ce fameux fichier. Je me renseigne un peu plus et comprends le fonctionnement de rake et ses besoins, je rajoute ce qu'il faut sans me douter du fond du problème et recommence. Même message d'erreur qui n'a pas bougé d'un poil. A force d'étudier ce qu'il se passe j'en conclue finalement que le commit n'est pas transmis au serveur, le pipeline qui charge dans le vide, ce n'était pas super représentatif. Après vérification, dans ma configuration tout est ok le problème vient d'ailleurs.

J'avais lu que au lieu de configurer directement sur Jenkins l'adresse de proxy, les authentifiants et autres options menant à la récupération du commit, on pouvait configurer ça dans le module de pipeline du serveur, à l'aide AWS CLI, qui est l'interface de ligne de commande d'Amazon.

```
aws configure
```

```
AWS Access Key ID [None]: Identifiant
AWS Secret Access Key [None]: Mot de passe
Default region name [None]: us-west-1 qui correspond à la région Irlande
Default output format [None]: json c'est le format par défault
```

Une fois modifications faites, le message d'erreur est nouveau. La console m'indique enfin que les fichiers sont bel et bien récupérés, mais qu'il ne peut pas accéder au fichier haml.

J'ai vérifié directement sur la machine Jenkins que les fichiers sont bien là, et c'est le cas, même le fameux fichier haml qui allait avec rakefile et tous les autres dont mon test avait besoin.
Il manquait encore des bibliothèques pour que le serveur puisse interpréter un fichier haml.
Le compte y est. Après 200 tests lancés, avec le minuteur, à la main ou au final un vrai déclencheur, le premier test fonctionne.

Le déploiement, qui constitue la dernière étape, n'ayant subit aucun changement, devrait fonctionner à ce moment là.
Justement non, les artéfacts d'entrées et de sorties correspondent à ce qui va être envoyé et reçu à chaque étapes du pipeline.

Ayant rajouté l'étape de Test entre le commit et le Deploy, il faut changer la sortie du serveur de test pour la faire correspondre à l'entrée du déploiement.
Et voilà, le pipeline est de nouveau fonctionnel avec l'étape de build qui pourrait être du test, en plus.

Au final je vais mettre en place de vrais tests avec phpunit et en rapport avec un projet Laravel.
Toutes ces expériences m'auront permis de trouver le profil d'instance adéquat pour un serveur Jenkins, et ce dont il a besoin pour son bon fonctionnement.

J'ai dressé une liste des bibliothèques et logiciels à installer :

- Php7.0, avec -cli, -common, -json, -readline, et curl
- Composer
- Pip
- openjdk-7-jre
- Jenkins, avec sa bibliothèque java et json
- Awscli
- Bison
- Make
- python-software-propertie
- Gcc
- Node-js

4- Présentation

On est dans la sixième semaine, une pression s'installe, fin de semaine présentation devant tout le service informatique de mes recherches, l'architecture, les choix à faire par rapport au besoin de l'entreprise, les services à utiliser et j'en passe.

Pour me préparer j'effectue quelques changements pour la propreté de l'architecture que j'ai mise en place, permettant par la même occasion de revoir certains points.

Ecriture d'une documentation complète en rapport à la présentation. J'ai préparé un support de 50 diapositives, pour une présentation qui a duré une heure, avec des questions bien sûr à son issue.

Partie 3 : Implémentation finale

Je suis revenu d'abord sur Elastic Beanstalk et ses principes, et j'ai fait la comparaison avec le simple Load Balancer pour pouvoir me décider avant de passer à la suite
Avant de passer aux cas concrets avec l'adaptation de deux des sites en phase d'intégration continue, il fallait effectuer quelques tests de montée en charge pour finaliser mes recherches.

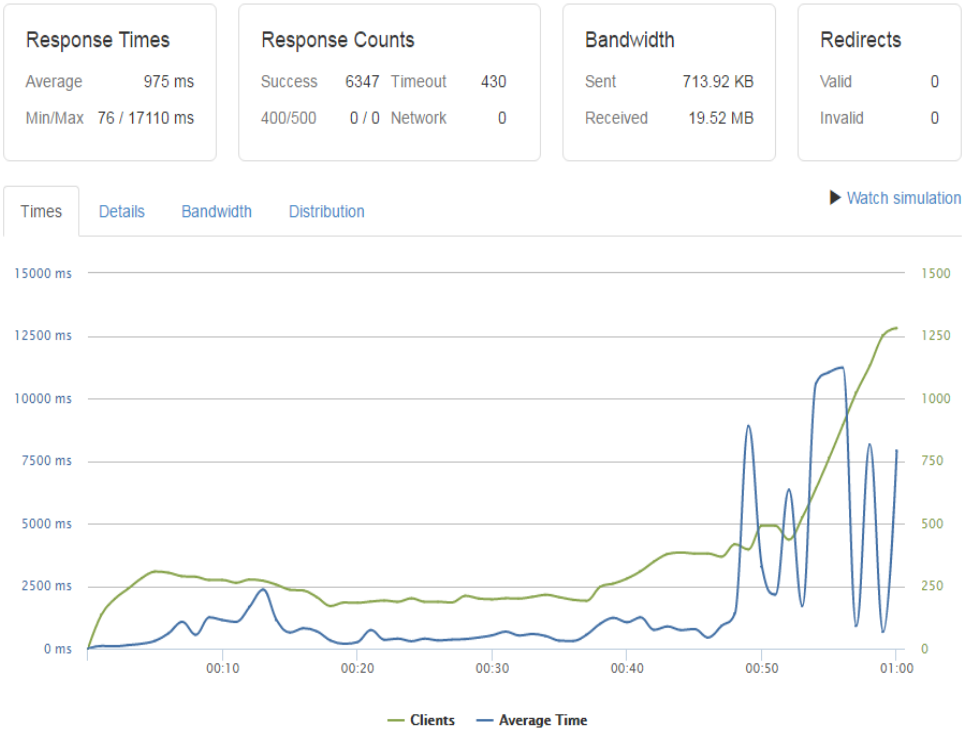
1- Test de montée en charge

Tous les tests sont effectués sur une minute (contrainte de loader.io)
Ce sont des envois de requêtes Get (ce qui consiste a récupérer les information d'une page) symbolisant des connections de clients. Si tout se passe bien, et que le serveur ne se retrouve pas submergé, l'intégralité sera reçue, sans aucun échec (timeout).

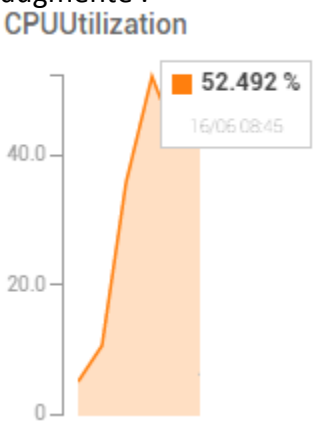
J'ai effectué de nombreux tests, voici les conclusions que j'ai pu en tirer :

Pour une machine micro :

Après avoir effectué des tests en partant de 2000 Get, pour monter de 1000 en 1000, j'ai pu remarquer logiquement que le temps de réponse augmente au fur et à mesure.
J'atteins ici 8000 Get demandé, j'en obtiendrais seulement 6347 avec en 430 timeout.



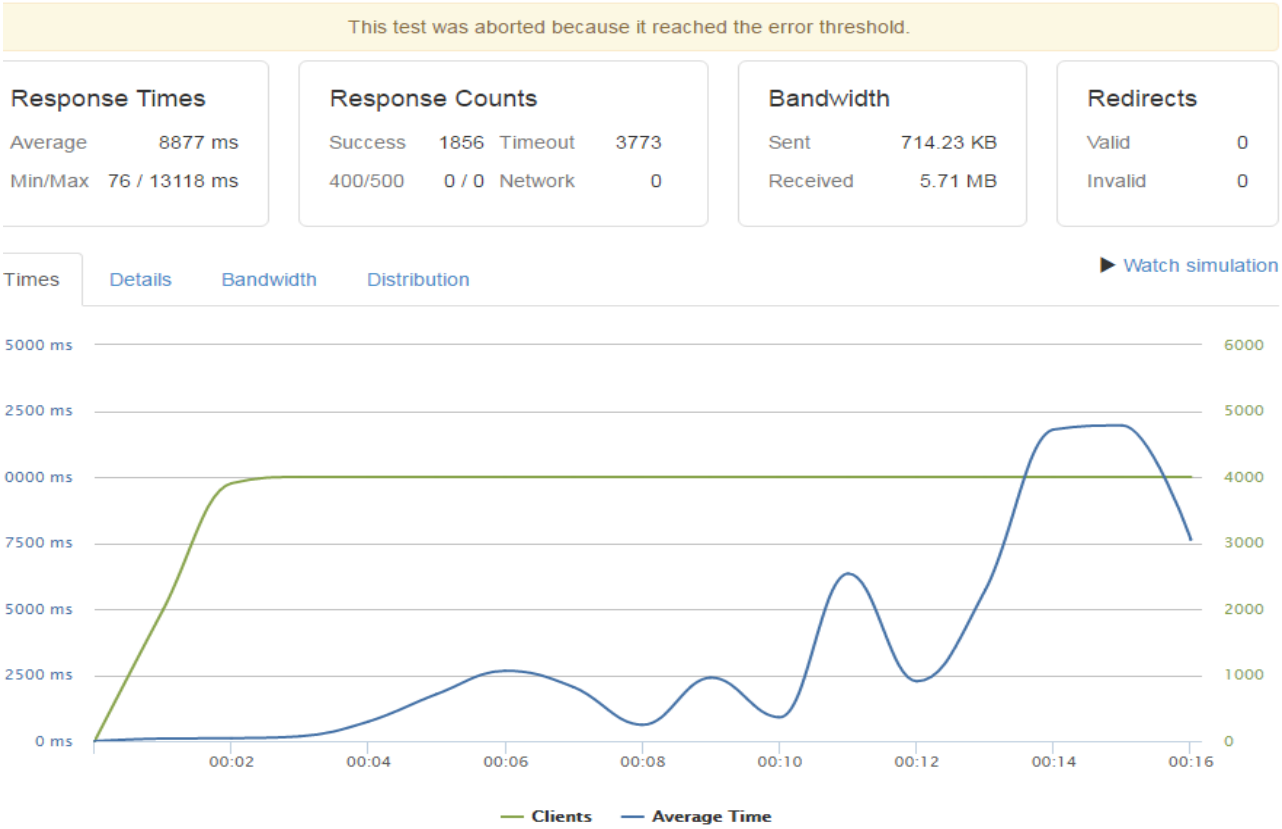
Deux remarques, la première est que sur la fin, le nombre de client connecté sur le site est très élevé (près de 1300 clients) ce qui va allonger fortement le temps de réponse.
La seconde étant que pour ce test le serveur a eu beaucoup de mal à assurer, son CPU à considérablement augmenté :



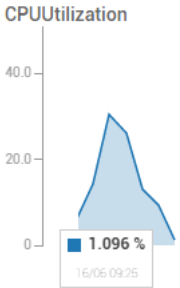
De telle sorte que, le test suivant n'a pas abouti, et le serveur ne répondait pas.

Le problème de ces tests, c'est que tout dépend de l'état du serveur au moment où l'on va l'utiliser, il y a plein d'éléments dans son état qui sont à prendre en compte. On pourrait refaire les mêmes tests des centaines de fois, qu'il serait presque impossible de retomber deux fois sur les mêmes résultats. Par exemple, si le serveur a été sollicité ou contient déjà plusieurs clients connectés, un test à 4000 Get pourrait ne pas se dérouler à la perfection, et contiendrait des timeout.
Le contenu de la page chargée peut faire différer lui aussi les résultats.
Il faut donc mesurer l'utilisation et l'interprétation de ces résultats, en considérant chacun des paramètres.

Pour une machine small :
On va partir de 5000 Get. Tout se passe bien.
Je fini par lancer un 12000 Get :



67% d'erreur (de timeout), c'est la limite à ne pas atteindre, le test est donc coupé.
Il n'y a pas que la puissance qui joue, peu de connections ont abouti et le serveur n'a pas forcé :



1% d'utilisation à la fin du test.

Je conclus sur le fait que la capacité du serveur va effectivement jouer et augmenter le nombre de connections et de demandes possibles, mais pas de façon proportionnelle.
Une instance micro suffira largement aux besoins de l'entreprise.
Il faut être vigilant et placer un seuil à ne pas atteindre assez bas par rapport au résultat ici présent. Un serveur micro va s'emballer beaucoup plus facilement qu'un autre, et même s'il pourrait être remplacé, cette forme d'erreur est difficile à détecter, et l'environnement aura simplement envie d'attendre que la machine revienne à son état stable avant de pouvoir assurer à nouveau ses fonctions.

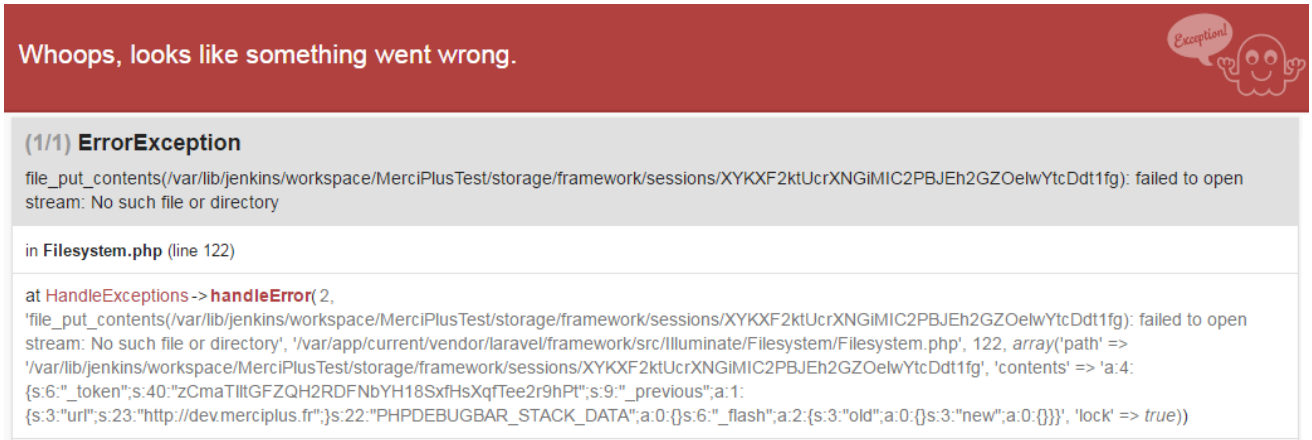
2- Adaptation

Les machines types pour Elastic Beanstalk comportent des images systèmes simples, elles sont optimisées pour peser le moins possible avec le strict minimum veillant à leur bon fonctionnement. Plusieurs commandes ne sont donc pas disponibles, comme dpkg ou encore apt-get qui permettent l'installation des paquets. Difficile de personnaliser à la main ces instances, si l'on ne sait pas qu'elles acceptent la commande yum.
La question est, pourquoi devrais-je avoir à modifier ces instances-là, alors qu'elles accueillent juste le projet final, après le commit et la vérification et les modifications de Jenkins, tout simplement après avoir étés complétées dans le pipeline.

En fait lors de l'implantation d'un site de l'entreprise, il me fallait accéder à leur base de données contenant des éléments du site web. Mais avant de faire ça, il était bon de savoir que ce site pourrait être hébergé dans de bonnes conditions. L'idée était donc de tester, en local sur le serveur qui hébergerait le site, que tout fonctionne. Il fallait donc créer une base de données sur le dit serveur pour simuler une réelle connexion et corriger les problèmes, s'il y en a.

Voilà pourquoi la possibilité d'installation des paquets est primordiale. Une fois le serveur, ici MySQL, créé sur la machine dans l'environnement EB voulu, une modification effectuée dans un fichier de configuration pour pointer sur une connexion locale, la partie base de données fonctionnait.

Des problèmes il y en a toujours, je vais m'arrêter sur un de ceux que j'ai rencontré sur cette phase finale de mon projet en entreprise. Il n'y a pas eu que des problèmes de base de données lors de l'implémentation d'un réel site web. Des problèmes de configuration qui ne correspondaient plus, des problèmes de droits d'accès ou d'écriture dans le projet pour Jenkins, qui a fini pour ne faire perdre mon serveur, et j'en passe.
Le dernier corrigé n'a pas était le plus simple, tant l'erreur donné n'était pas explicite.



On m'indique donc une erreur qui proviendrait des sessions. Une réflexion importante ici, les erreurs qui me sont données sont des fins de processus, et donc lors de ceux-ci, de nombreuses tâches sont effectuées, mais seule la finalité nous est communiquée.
Le problème peut donc venir d'ailleurs, comme un élément avec des informations à récupérer pour le bon fonctionnement d'une partie du processus qui ne peut pas être récupéré.

Pour confirmer mes propos, ici le problème, et donc la solution, viennent de cache.php un fichier se trouvant ailleurs dans le projet, dans un dossier racine nommé config. Pour régler ça j'ai changé mon script shell que j'exécute sur le serveur à chaque test pour pouvoir ignorer le cache.

En parlant des scripts, je finirais avec deux différents :

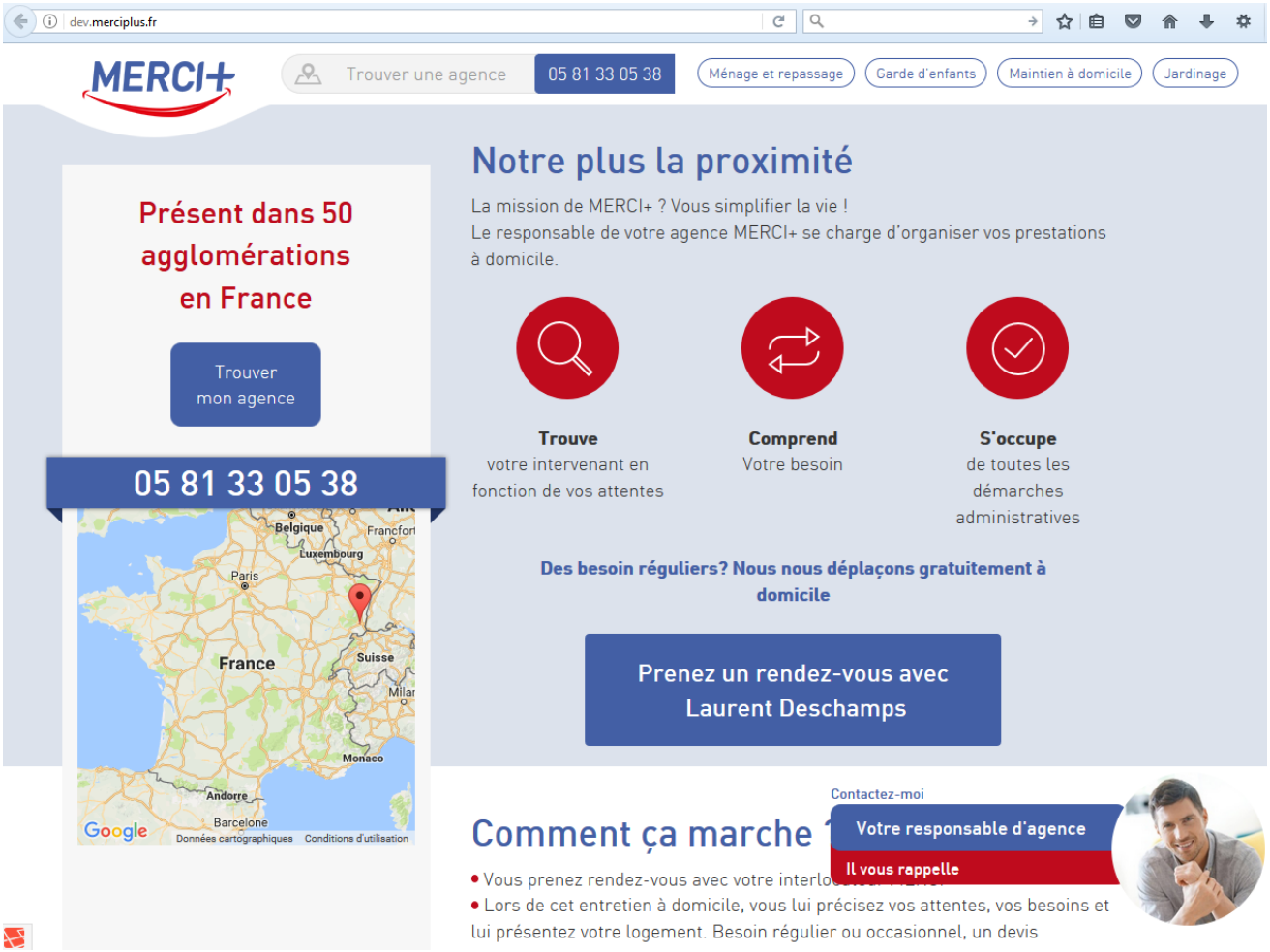
Script pour reconstruction seule du projet :

```
#Je recupère le dossier vendor, qui contient #des ressources pour le projet
/bin/composer.phar update;
#Les modif sont faites d'abord sur l'exemple, je le copie dans le normal pour qu'il devienne effectif
cp .env.example .env
#Je génere une clé pour qu'elle soit conforme au format actuel.
php artisan key:generate
```

Script pour reconstruction et lancement des tests :

```
/bin/composer.phar update;
cp .env.example .env
php artisan key:generate
#Je donne tout les droits sur le dossier storage et config, pour
pouvoir les modifier sans conflit.
chmod -R 777 storage
chmod -R 777 config
#J'efface le cache pour ne pas gêner les tests
rm -r config/cache.php
#Je lance les tests
phpunit
```

Au final le site est fonctionnel, et cette victoire vient terminer mon rapport.



Bilan

Un premier pas dans la vie professionnelle réussi, qui m'a permis d'en comprendre beaucoup sur le fonctionnement d'un grand groupe. La communication entre les services, la répartition des taches intra-service, la politique adoptée et de façon générale un savoir être adapté, visant à être un élément qui apporte à toute l'équipe.

Je suis heureux d'avoir intégré ce service, et plus particulièrement d'avoir travaillé sur ce projet, qui pour moi a été une source de progression conséquente, en acquérant de nouvelles connaissances et compétences.

Ce fut extrêmement plaisant, j'ai énormément appris, la tâche n'a pas étaie simple en partant de zéro sur de nombreux domaines.
Cependant la structure du projet qui m'a été donné, la façon dont il a été suivi, et le cap fixé étant clair, l'avancement à toujours suivi sont cours sans même que je m'en rende compte parfois, plongé dans toutes ces nouvelles notions.

Une expérience enrichissante, dans un cadre idéal pour prendre connaissance du quotidien dans ce type d'entreprise. J'aurais croisé durant ces 10 semaines de nombreux corps de métier, et eu la chance de pouvoir échanger avec beaucoup de personnes.

Un aspect que j'ai trouvé intéressant, à été de savoir comment est organisée la communication entre les équipes, primordiale pour la coordination de l'entreprise, ainsi que la communication au sein même d'une équipe.

Au final ces deux ans auront été pour moi remarquablement efficaces, j'ai pu évoluer et en apprendre vraiment beaucoup. Mes connaissances qui ont été étendues en majeure partie par les enseignements du DUT, se sont vues complétés par cette dernière expérience qui marque la fin de cette formation.

Glossaire

AWS : AMAZON WEB SERVICE
La plateforme de service Cloud proposé par Amazon.

MVC : Modèle Vue Controller (PHP)
Motif d'interface logicielle destiné aux interfaces graphiques.

ORM : OBJECT RELATIONNAL MAPPING
Correspondance entre monde objet et relationnel.

ERP : ENTREPRISE RESSOURCE PLANNING

Deux dimensions qui caractérisent ERP :

- **DI :** Degré d'intégration, capacité à fournir à l'ensemble des acteurs de l'entreprise les informations nécessaire à leurs fonctions.
- **CO :** Couverture opérationnel, capacité à fédérer l'ensemble des processus de l'entreprise, pour optimiser la productivité.

Prologiciel de gestion intégré, gestion globale, cohérente, simple.

PSR : PROPOSE A STANDARDS RECOMMENDATION

Recommandations PHP ayant pour objectif d'améliorer l'interopérabilité entre les frameworks.

Il existe aujourd'hui plusieurs PSR, les 4 principalement utilisés sont :

- PSR-0 : qui traite du chargement des classes PHP et de l'autoloading.
- PSR-1 : qui fixe les conventions minimales de codage.
- PSR-2 : qui défini le style et l'organisation du code.
- PSR-3 : qui s'occupe de l'interface des loggers.

ITIL : INFORMATION TECHNOLOGY INFRASTRUCTURE LIBRARY

Un ensemble recensant les bonnes pratiques du management du système d'informations.

DAF : Directeur Administratif et Financier

SRDF : SYMMETRIX REMOTE DATA FACILITY

Permet la réplication de données informatiques entre deux baies de disques en utilisant le mécanisme miroir Maitres/Esclaves. Les disques maitres (nommé R1) sont en accès RW (Read-Write) et les disques esclaves (nommé R2) sont en WD (Write Disable).
Le mode synchrone : (5 étapes)

- Demande écriture baie locale (1)
- Validation dans le cache de la baie locale (2)

- Envoyé par le lien SRDF sur baie distante (3)
- Validation dans le cache de la baie distante (4)
- Baie locale reçoit l'acquittement de baie distante et valide l'écriture (5)

SAN : STORAGE AREA NETWORK

Réseau spécialisé permettant de mutualiser des ressources de stockage. Peut assurer la redondance de stockage. En doublant au minimum chacun des éléments.
Le SAN peut fonctionner dans un environnement complètement hétérogène (Unix, Windows…)

Infogérance :

Un service défini comme le résultat d'une intégration d'un ensemble de services élémentaires, visant à confier à un prestataire informatique tout ou partie du système d'information d'un client.

Monté en charge :

Un rush, sollicitation brutale d'un service, un pic de connexions par exemple.

CIDR : CLASSLESS INTER DOMAIN ROUTING

Il s'agit du système d'adressage sans classe.

Requête serveur :

Une connexion à un serveur a été précédée d'un Get du client, qui est une requête. Il en existe plusieurs, qui opèrent lors d'un échange client-serveur.

IC : INTEGRATION CONTINUE

POSIX : PORTABLE OPERATING SYSTEM INTERFACE UNIX

X fais référence à l'héritage UNIX. POSIX est un standard de système.

IAM : IDENTITY and ACCESS MANAGEMENT

Permet de contrôler l'accès aux services et ressources et la manière dont ils peuvent les utiliser, en lecture ou écriture par exemple (config d'autorisations par user).
Accessible sur AWS Management Console.

Middleware :

C'est un logiciel qui permet la communication entre différentes applications ou réseaux afin de permettre un échange d'information.

Bibliographie

Nicolas Hachet. (10 janvier 2013). Qu'est-ce que les recommandations PSR en PHP ?

- Le site Amazon Web Services :

https://aws.amazon.com/fr/

- Le simulateur de cout d'AWS :

https://calculator.s3.amazonaws.com/index.html

- Le site de graphique, de nombreux cours m'ont aidé durant le stage :

https://www.grafikart.fr/

- Information sur PSR :

http://blog.nicolashachet.com/technologies/php/quest-ce-que-les-recommandations-psr/

- Loader.io, pour les tests de montée en charge :

https://loader.io/tests/9ad1384bdba73e1d5da1e35a586bce31/summaries/fd79ee9fdc55d69fb08d0c83ace01dd2

Institut Universitaire de Technologie,
Aix-Marseille Université

ANNEXES
Diplôme Universitaire de Technologie
Spécialité Réseaux et Télécommunications

Conception Cloud, Middleware d'intégration des leads en haute disponibilité

Damien RABELLINO

Viadom

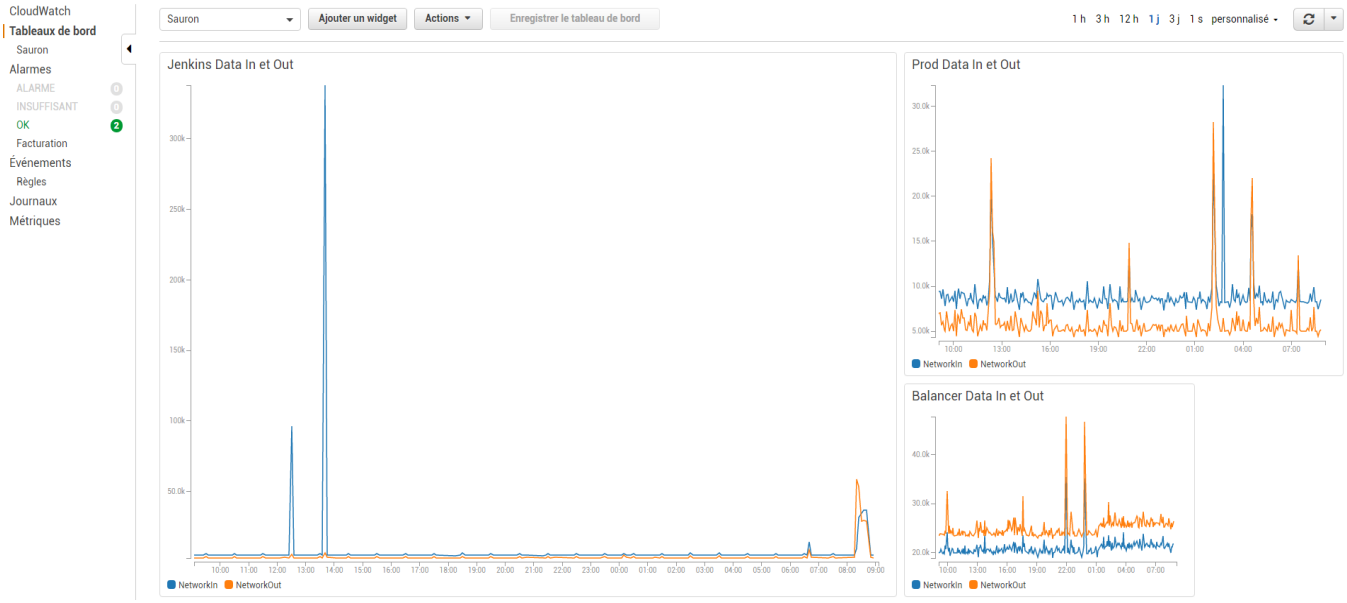
Responsable entreprise : Guillaume Cyr

Responsable académique : Éric Würbel

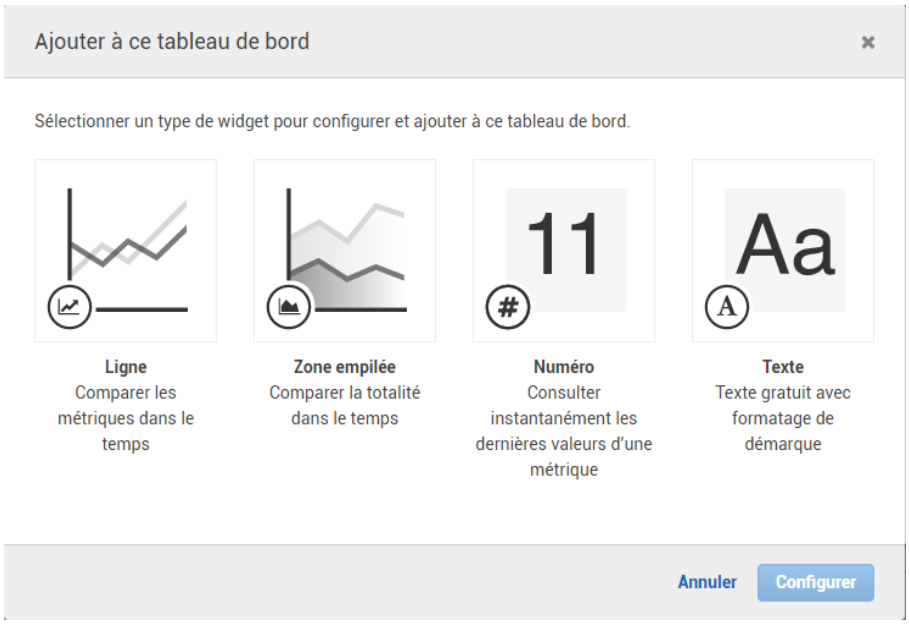
2016

1. CloudWatch :

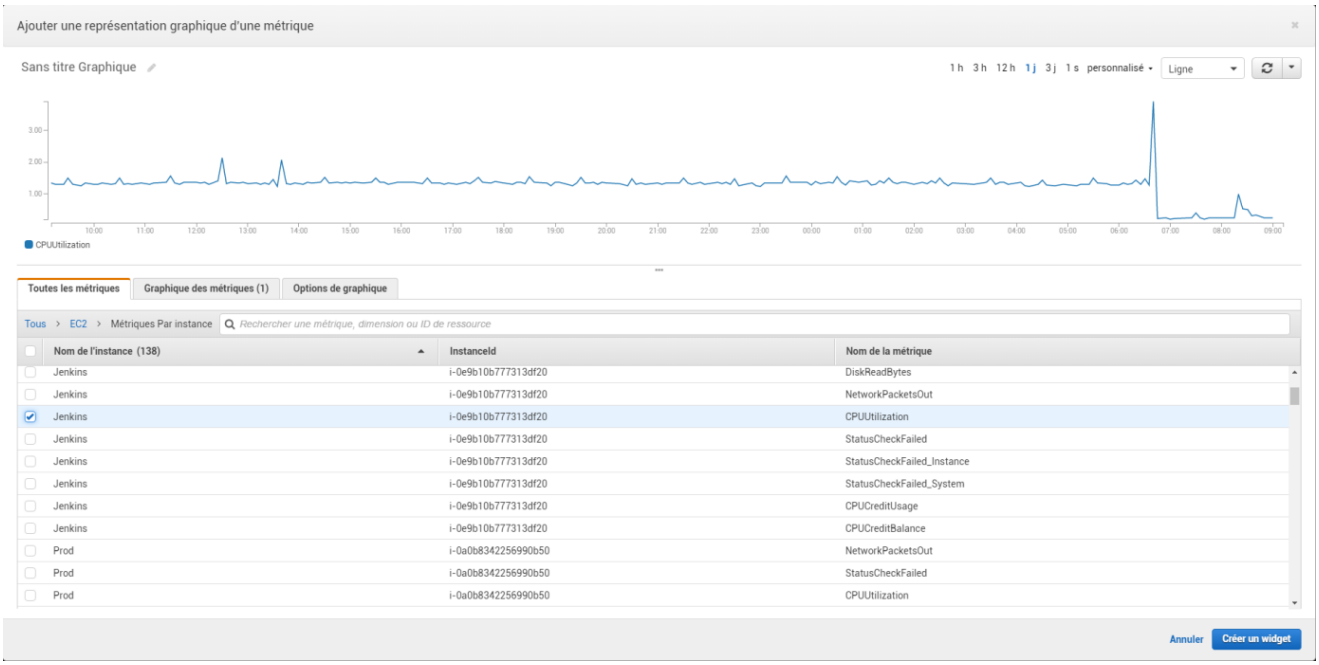
Vue d'un tableau de bord :



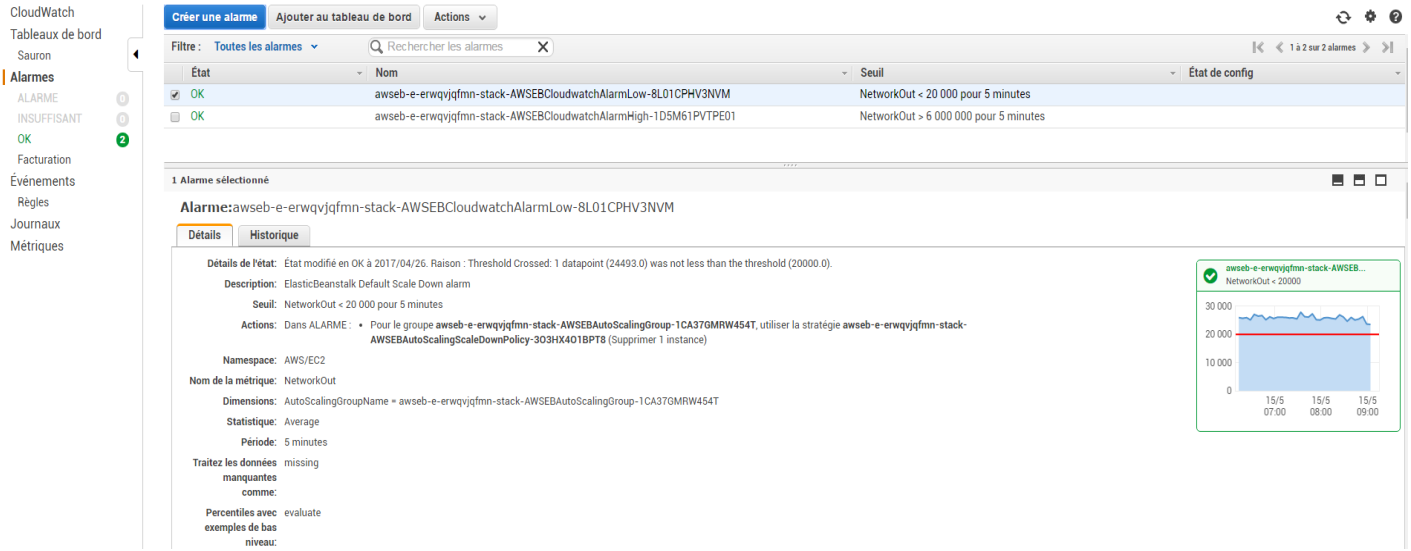
Les types de graphiques disponible :



Interface de création d'un graphique :

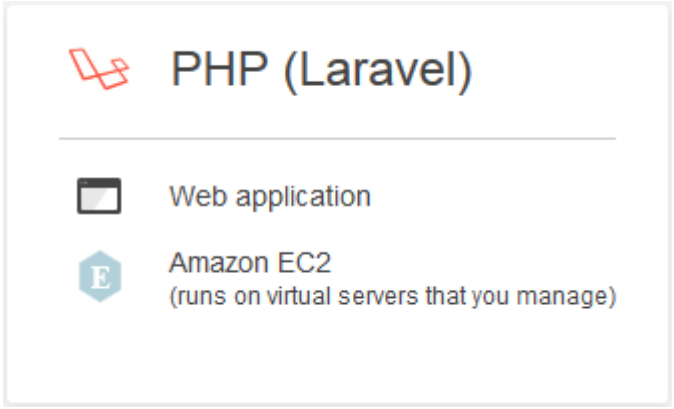


Alarme :

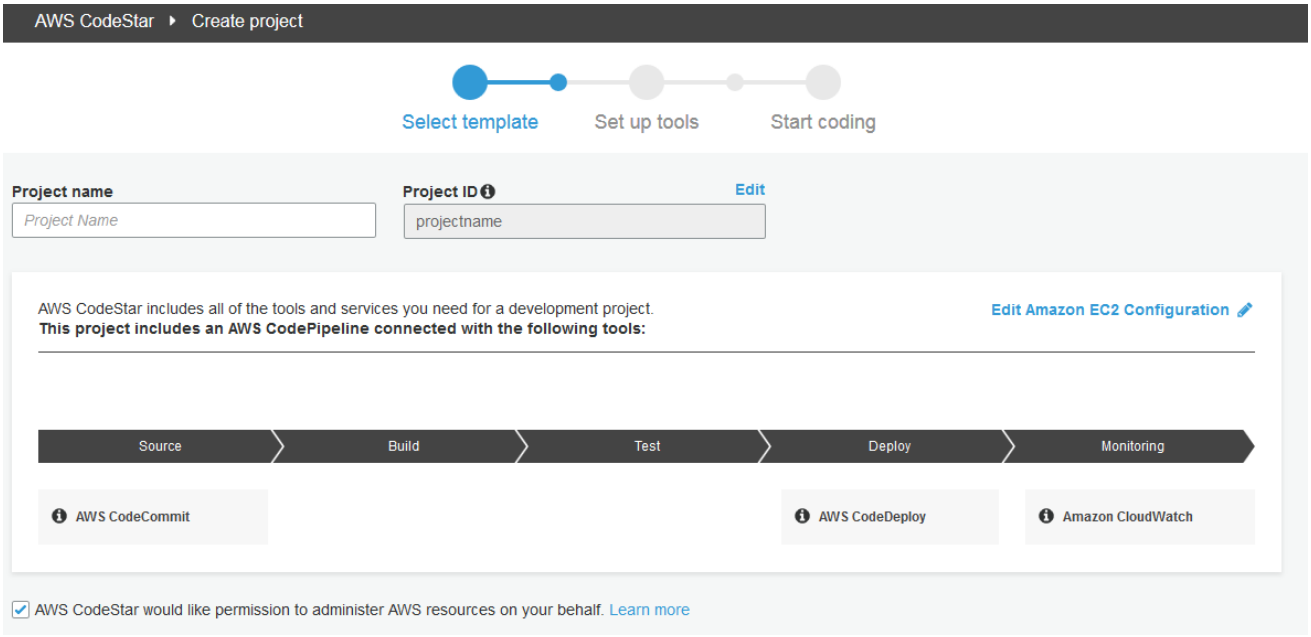


2. CodeStar :

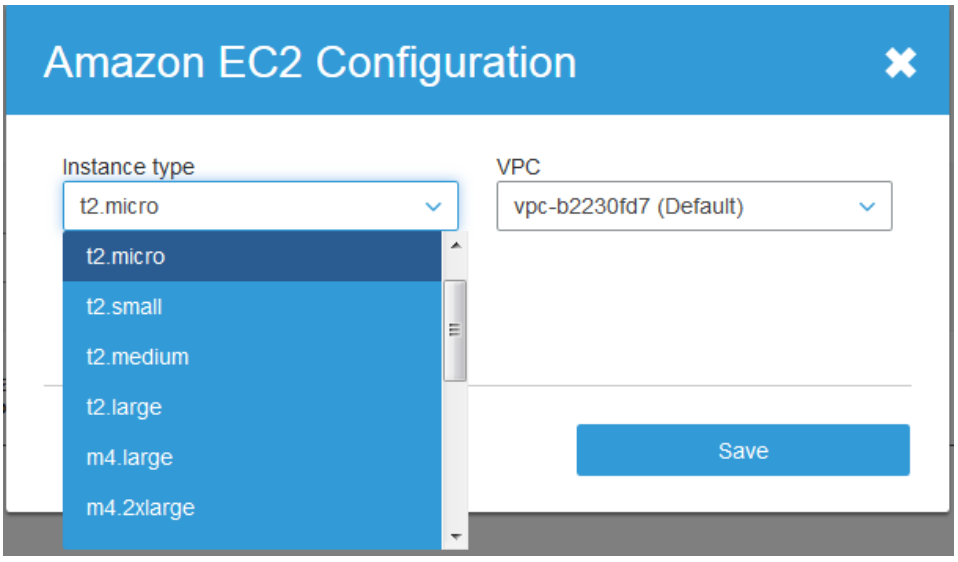
Choix du template :



Etape de création du projet :

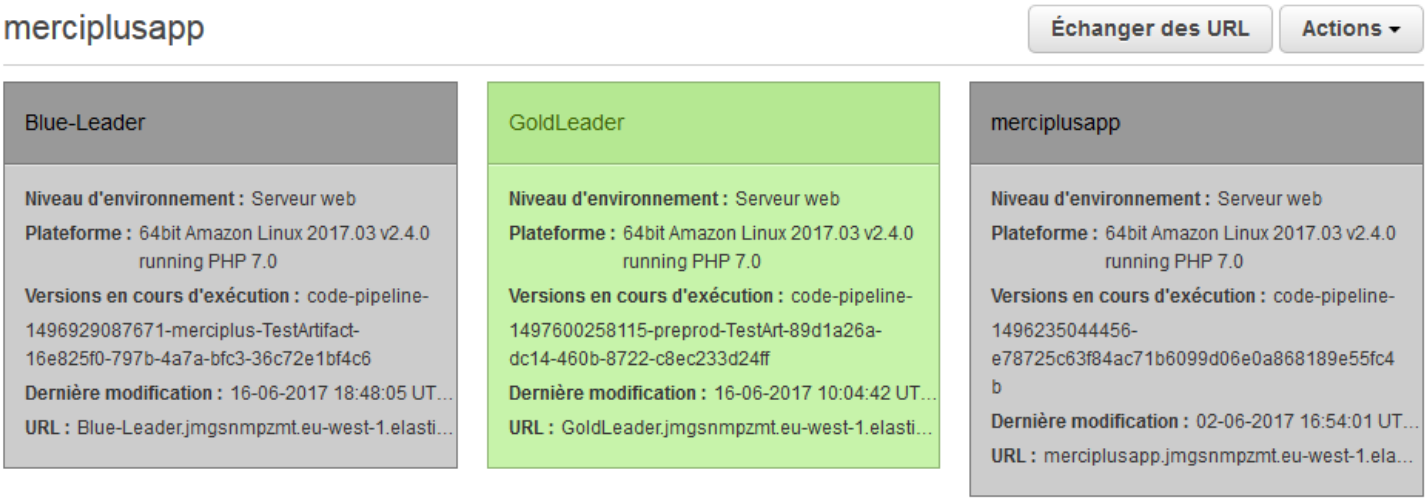


Choix de la configuration des EC2 :



3. Elastic Beanstalk :

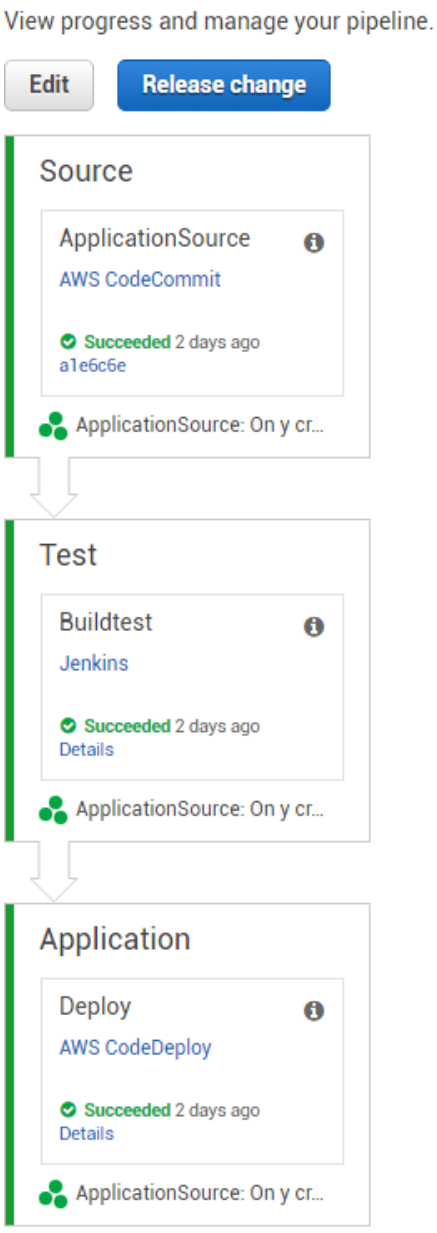
Une vue sur les environnements :



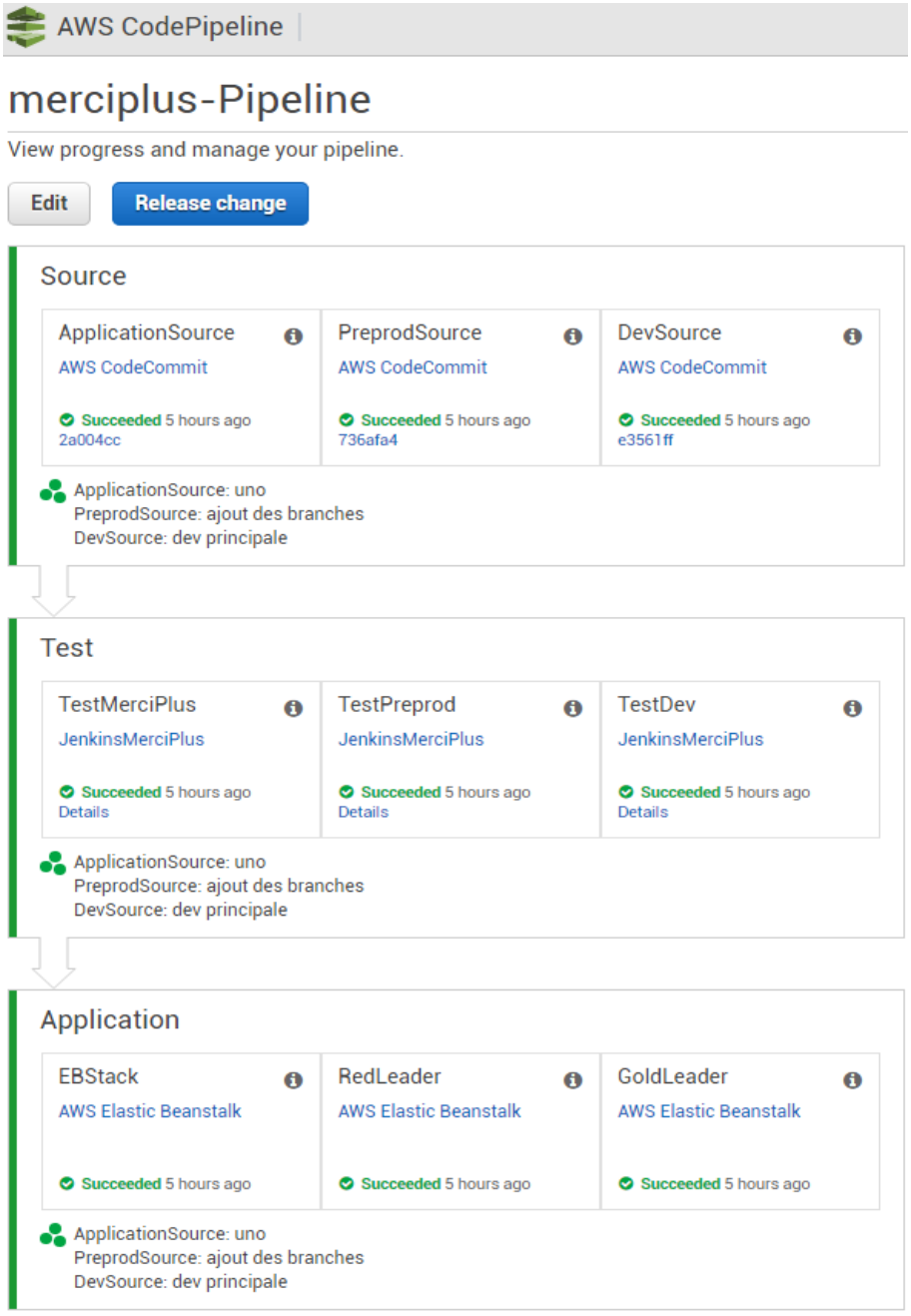
Les environnements sont regroupés avec leurs applications (projets) respectifs. Le code couleur indique l'état de santé de l'environnement, par exemple vert pour Ok, c'est a dire fonctionnel, rouge pour grave, indiquant un problème dans l'environnement, et gris qui signifie que cet environnement ne fait rien, il n'a pas d'état car il ne contient pas d'instances.

4. Pipeline :

Pipeline simple :



Pipeline final avec la prod, la preprod et la dev:



5. Evaluation des coûts :

Un exemple de calcul obtenu par simulateur :

⊖	Amazon EC2 Service (Europe)		$ 2497.72
	Compute:	$ 2458.80	
	Elastic IPs:	$ 18.30	
	Elastic LBs:	$ 20.50	
	Data Processed by Elastic LBs:	$ 0.12	
⊕	Amazon S3 Service (Europe)		$ 0.14
⊕	Amazon DynamoDB Service (Europe)		$ 0.00
⊕	Amazon SNS Service (Europe)		$ 0.00
⊕	Amazon SQS Service (Europe)		$ 0.00
⊕	AWS Data Transfer In		$ 0.00
⊕	AWS Data Transfer Out		$ 1.35
⊕	AWS Support (Basic)		$ 0.00
Free Tier Discount:			$ -36.66
Total Monthly Payment:			$ 2462.55

Voici un exemple de paramétrage :

Elastic IP:

Number of Additional Elastic IPs:	3	
Elastic IP Non-attached Time:	0	Hours/Month
Number of Elastic IP Remaps:	0	Per Month

Data Transfer:

Inter-Region Data Transfer Out:	10	GB/Month
Data Transfer Out:	10	GB/Month
Data Transfer In:	100	GB/Month
VPC Peering Data Transfer:	0	GB/Month
Intra-Region Data Transfer:	0	GB/Month
Public IP/Elastic IP Data Transfer:	0	GB/Month

Elastic Load Balancing:

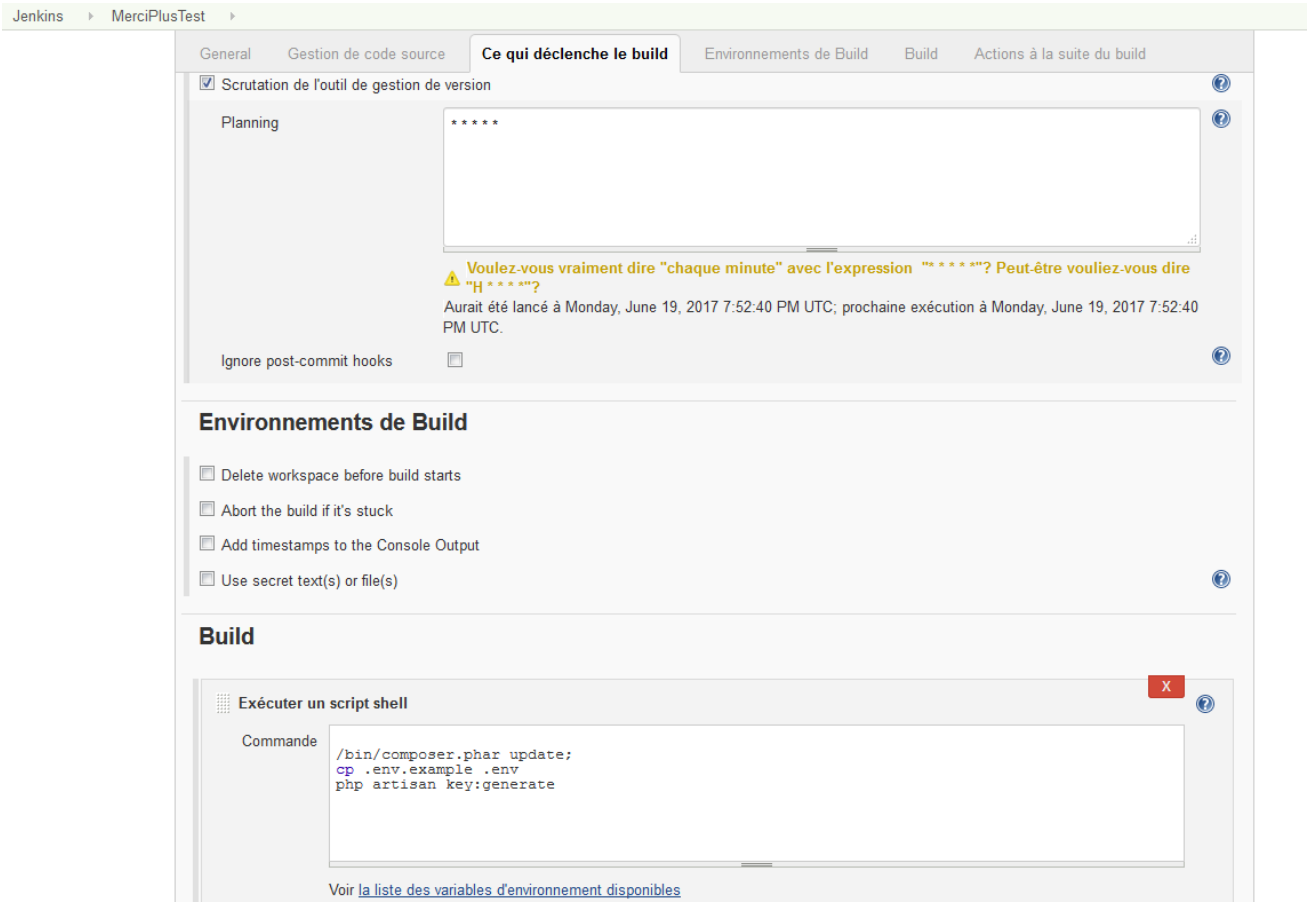
Number of Elastic LBs:	3	
Total Data Processed by all ELBs:	100	GB/Month

Standard Storage:

Storage:	100	GB
PUT/COPY/POST/LIST Requests:	100000	Requests
GET and Other Requests:	100000	Requests

6. Jenkins :

Une partie de l'interface de configuration :



Page principale :