ECOLE PRATIQUE DES HAUTES ETUDES COMMERCIALES

EPHEC

INSTITUT D'ENSEIGNEMENT SUPERIEUR DE TYPE COURT

Av. du Ciseau 15 – 1348 Louvain-la-Neuve

Bachelier en Informatique et Systèmes orientation Technologie de l'Informatique

Rapport de stage en entreprise

Effectué à

Rue Royale 47 – 1000 Bruxelles

Par

Julien Castiaux – 3TL1

ANNÉE SCOLAIRE 2017 – 2018

Table des matières

Remerciements

J'aimerais tout d'abord remercier la société *StepStone* qui a bien voulu me prendre comme stagiaire et qui a rendu possible cette aventure.

J'aimerais ensuite remercier mes collègues des bureaux de Bruxelles, Munich et Berlin. Le quotidien au sein de l'entreprise était agréable, tout le monde était souriant, tout le monde était de bonne humeur.

Un grand merci à mes collègues de *Search and Match* et plus particulièrement aux équipes *Services* et *DevOps* avec qui j'ai travaillé au quotidien. Vous avez toujours été disponibles pour répondre à mes questions et à me guider dans mes projets. Comme je l'ai souligné dans chaque backlog, l'ambiance au sein de l'équipe a vraiment été géniale, merci à tous !

Merci aussi à l'EPHEC pour la formation que nous avons reçue et qui m'a permis d'être opérationnel tout au long du stage.

En dernier mais pas des moindres, merci à ma famille qui m'a soutenu et qui continue de me soutenir au quotidien.

Introduction

J'ai réalisé mon stage à StepStone dans les bureaux de Bruxelles. J'ai intégré le service *Search and Match* et plus précisément l'équipe *DevOps*.

Objectifs du stage

L'objectif principal de mon stage était d'intégrer *Recommender* (un logiciel développé par *Total Jobs* et racheté par *StepStone*) à notre logique de déploiement continue. Je devais comprendre la logique de déploiement mise en place par *Total Jobs* et l'adapter à notre logique propre.

Mon maitre de stage a estimé que j'aurais besoin de toute la durée de mon stage pour effectuer cette migration. Au final la migration s'est effectuée plus rapidement et j'ai pu travailler sur d'autres projets.

Attentes personnelles

Je cherchais particulièrement à rejoindre une boite avec une méthodologie agile, une culture DevOps et une base de code en Python. Vu qu'aucun stage de la base de données EPHEC était à la hauteur de mes attentes, j'ai entrepris des démarches pour trouver un lieu de stage dans ces domaines.

Le service *Step and Match* de StepStone répondait exactement à mes attentes :

- L'équipe travaille avec un judicieux mélange de Scrum et Kanban.
- Mon équipe *DevOps* (ops) et l'équipe *Services* (devs) travaillent main dans la main au quotidien : même open-space, même standup-meeting, même kanban, même backlog review. De l'extérieur nous sommes visibles comme une seule et même équipe.
- Mon équipe travaille en bash et en Python sur Debian.

Les méthodologies agiles m'intéressaient beaucoup et j'avais hâte de les essayer sur le terrain au sein d'une équipe conséquente pour en mesurer l'efficacité.

Les pratiques DevOps sont importantes pour moi. Ayant toujours eu la double casquette de dev et d'ops et faisant du déploiement continue pour mon TFE, rejoindre une équipe DevOps me permettait de continuer avec cette double casquette en plus d'améliorer mes propres pratiques.

Le langage Python est un langage que j'aspire à maitriser. En rejoignant une équipe qui utilisait ce langage, je trouvais des personnes ressources avec qui partager et qui pouvaient m'enseigner les bonnes pratiques.

Présentation de l'entreprise et du département

StepStone est un *job board*, sa mission est de faire correspondre des chercheurs d'emploi avec des employeurs. Son business modèle est de faire louer la plateforme aux employeurs, ils peuvent ainsi décrire le job qu'ils proposent et StepStone essayera de faire correspondre des demandeurs d'emploi avec ces offres.

Au sein de StepStone, j'ai travaillé dans le groupe *Search and Match*. Mon groupe développe les outils utilisés par le reste de StepStone. Au sein de ce groupe, je suis dans l'équipe *Search* qui regroupe les développeurs (les autres équipes sont constituées de linguistes et de chercheurs). Mon équipe est encore divisée en deux : *Services* et *Evolution* chacune travaillant sur des projets propres.

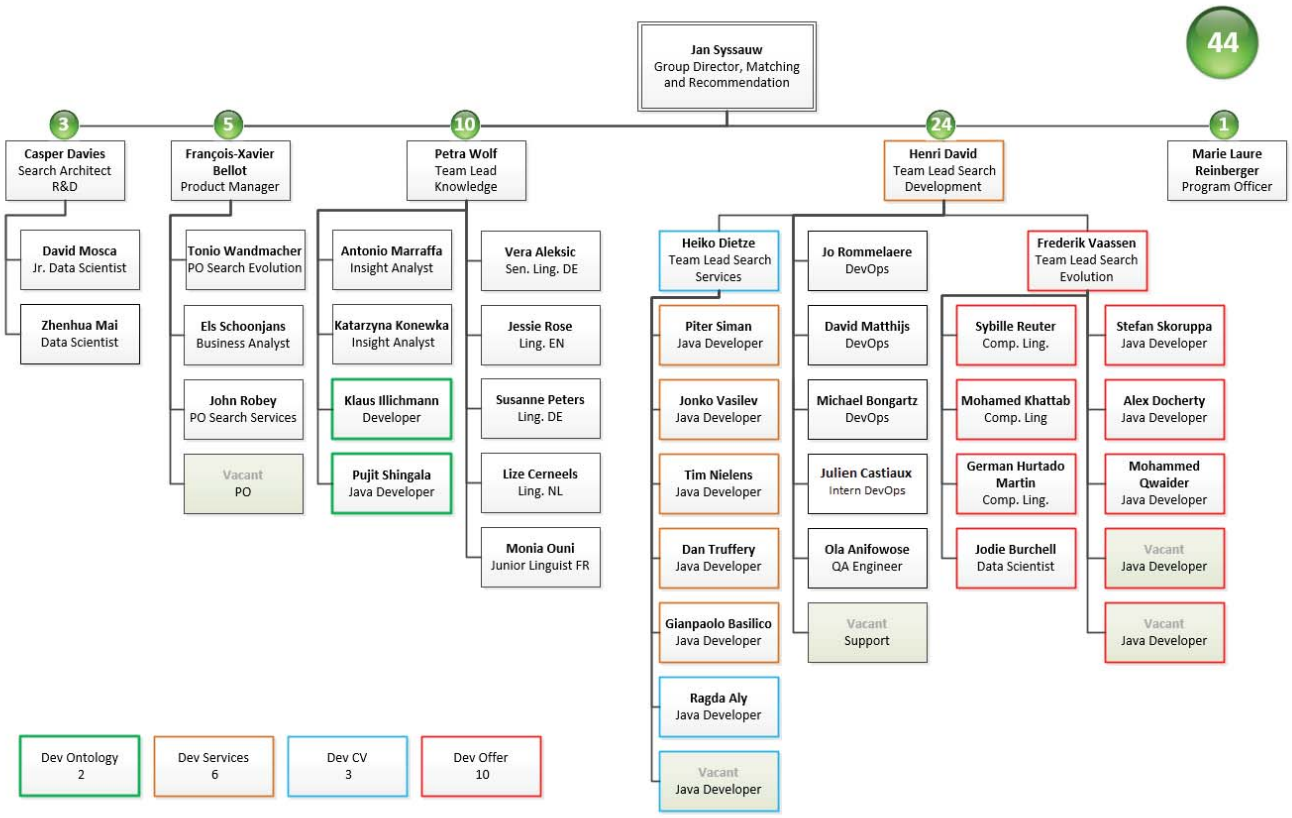

Illustration 1: Structure du service Search & Match

Si toute l'équipe *Search* était en réalité de culture DevOps, mon rôle (mal intitulé) *DevOps* était d'être à la fois:

Expert AWS :

gestion du parc, création des différents environnements et gestion de la sécurité

Administrateur système :

installation de services, monitoring proactif et support réactif

Développeur :

standardisation et création d'outils de déploiement continu

Exposé de la nature du stage et modalités de réalisation

Mon stage a été inscrit dans la volonté de StepStone de migrer tous leurs services vers AWS. Mon équipe avait déjà mis en place tout un pipeline de déploiement continu et était dans la phase de migration de chacun des projets de StepStone à ce pipeline. Comme mes collègues, j'ai donc travaillé à la migration des services et à l'amélioration de ce pipeline.

Quelques infos pour la suite...

Au sein de StepStone, 5 environnements existent (ou devraient exister) pour chaque service:

1. alpha (développement)
2. alphasearch (développement avec connexion à un réplica des bases de données live)
3. bêta (quality-assurance)
4. gamma (pre-live)
5. live

Vu que stepstone est un job-board principalement européen, les services sont déployés dans AWS soit dans le datacenter EU-WEST-1 (Ireland) soit EU-CENTRAL-1 (Frankfurt).

La stack haute-disponibilité de AWS se présente comme suit:

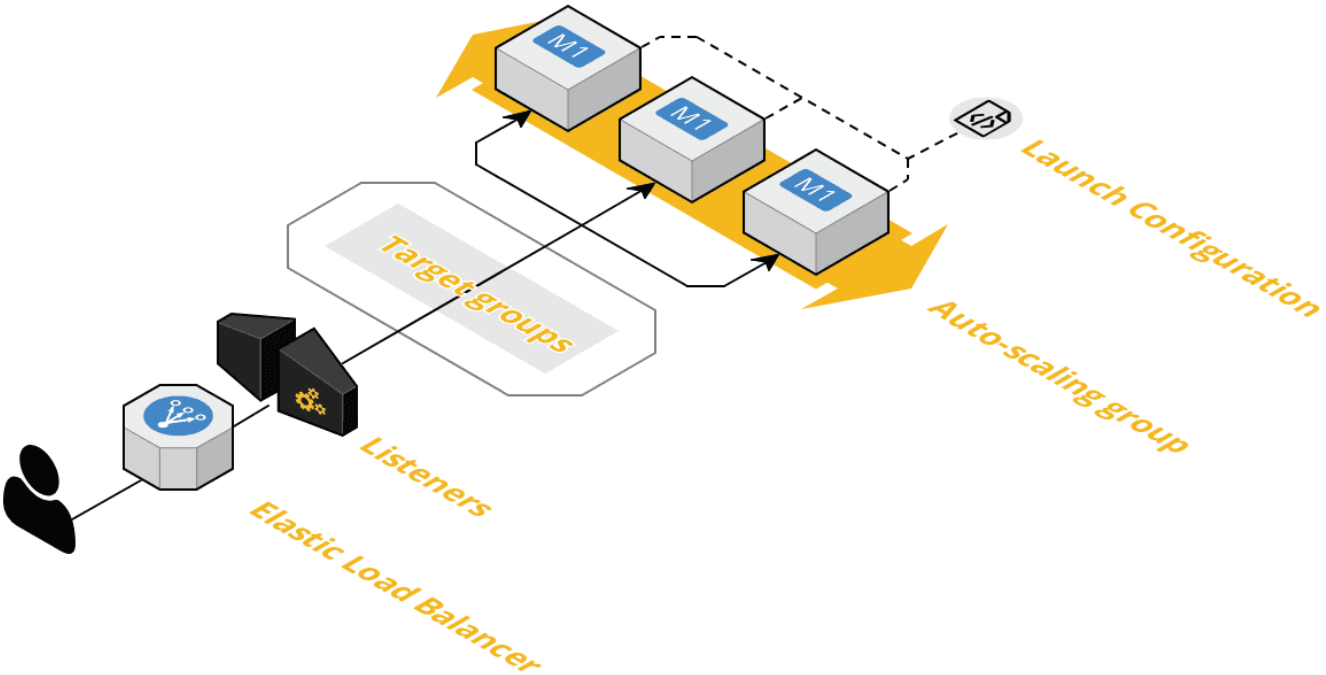

Illustration 2: Architecture hautement-disponible sur AWS

- Un elastic load balancer (ELB) en amont au projet reçoit tout le trafic vers ce projet
- Des listeners au niveau de l'ELB écoutent sur certains ports et redirigent vers des target-groups

- Les target-groups (TG) sont responsables de diriger les requêtes sur des instances viables. Pour distinguer les instances viables des autres, il effectue des health-check régulier sur chaque instance pour vérifier si l'application fonctionne (un health-check est par exemple une requête HTTP sur une url du service qui doit répondre avec un code http 200)
- Des auto-scaling groups (ASG) qui sont responsables de maintenir un nombre d'instances viables (lorsqu'une machine devient *unhealthy*, l'ASG en démarre une nouvelle pour la remplacer). Il peut également être configuré pour faire évoluer le nombre d'instances désirées en fonction du trafic et de la charge (auto-scaling in/out)
- Un ASG va démarrer des machines selon une *Launch Configuration*, il s'agit de la configuration intégrale de la machine : image à utiliser, type d'instance, configuration du firewall, configuration de l'espace disque et ainsi de suite.

Vault

Vault est un outil de centralisation des secrets, il se présente comme un serveur central qui possède tous les secrets et des agents déployés sur chaque machine voulant communiquer avec le serveur. Vault nous permet de générer des OTP (one-time-password) qui permettent d'accéder aux serveurs dans AWS. Lorsqu'un développeur veut accéder en SSH à une machine, il doit d'abord s'authentifier auprès du serveur vault au moyen de ses identifiants LDAP. Une fois authentifié, le serveur va créer un mot de passe unique qu'il configure sur le compte de l'utilisateur sur la machine cible et communique ce mot de passe à l'utilisateur. L'utilisateur peut alors se connecter à la machine. Une fois le mot de passe utilisé, celui-ci est invalidé pour empêcher tout rejeu du mot de passe.

Mon rôle concernant vault a été double :

- Premièrement l'intégrer à notre pipeline de déploiement continu afin de permettre la mise à jour du logiciel à chaud.
    - Transformation du binaire vault serveur en service système via systemd.
    - Configuration du packaging Debian.
    - Création d'un *build-plan* et d'un *deploiement-plan* au niveau de bamboo.
- Deuxièmement faire en sorte que le service soit hautement disponible
    - Déploiement d'une stack *high-availability* au niveau d'AWS (Elastic Load Balancer, Target Group avec health-checks, Auto-Scalling Group en mode self-recovery)
    - Redondance du service pour avoir une instance master et une instance slave.

Mon équipe m'avait gardé ce petit projet pour que je puisse comprendre tous les rouages mis en place et me faire les dents sur AWS et Bamboo.

IJM

Instant Job Match est un service backend de StepStone, ce service a été récupéré lors du rachat de *Total Job*, un job-board anglais. Ce service sert à avertir les chercheurs d'emploi lorsqu'une annonce pouvant les intéresser arrive dans la base de données de StepStone.

Mon rôle concernant IJM a été de migrer sa logique de déploiement continu vers notre logique afin de l'intégrer à nos services :

1. Étude du système mis en place

    - Découverte d'AWS *Opswork*, un service d'AWS permettant le déploiement continu au moyen de scripts Chef ou Puppet. Ce service fonctionne selon une succession de couches superposées, chaque couche ajoute son lot d'installations de services, de configurations et d'intégrations à AWS. Au sein d'AWS OpsWorks, 23 environnements existent pour IJM seulement.

    - Découverte de chef, un outil de *provisionning* automatique basé sur des scripts Ruby. Le repository où sont stockés ces fichiers contient pas moins de 22 sous-projets dédiés au déploiement de services dans OpsWork. La complexité de la structure, les liens entre des scripts entre plusieurs projets (pas toujours liés à IJM) ont fait que l'étude de cette partie a été très laborieuse.

      Nombre de fichiers dans le repo opswork:

      ```
      PS C:\Users\castij01\projets\aws-opsworks> Get-ChildItem -Recurse | Measure-
      Object
      Count    : 696
      ```

    - Étude de l'environnement sur les machines de développement, scripts de gestion, intégration à l'OS, comportement vis-à-vis de l'application.

2. Création d'un second environnement

    - Création d'un second *build-plan* et d'un second *deployement-plan* à IJM dans bamboo pour continuer à construire et déployer l'application en utilisant l'ancienne logique et commencer en même temps la migration.
    - Création des environnements dans AWS (toujours avec la solution de haute-disponibilité), parmi les 23 environnements utilisés dans Opsword, seuls 17 ont dû être créés avec ma logique.
    - Adaptation du fichier `pom.xml` pour prendre en compte ce 2e environnement.

1. Migration

    - Nettoyage de tous les scripts : suppression des scripts inutiles, révision des scripts obsolètes et adaptation à notre logique de déploiement.
    - Migration de l'intégration système de centos/initv à debian/systemd.
    - Duplication et révision des *security groups* (règles de pare-feu) dans AWS.

  Au final toute la quasi-totalité de la couche système a été refaite, chaque script a été retravaillé et dépoussiéré. L'intégration à AWS a également été réévaluée et adaptée.

3. Tests

   Malheureusement au vu de nombre important de changements en profondeur, la charge au niveau du débogage a été considérable et a subi un problème inhérent à notre pipeline. Afin d'avoir des déploiements rapides, chaque service est intégralement intégré sur une image Debian et packagé dans une AMI AWS via BootstrapVZ, la création de l'image et sa copie vers AWS pouvaient prendre jusqu'à 20 minutes pour IJM. Ces 20 minutes étaient généralement suivies par 20 autres minutes nécessaires au déploiement.

   Un autre problème a été le manque de disponibilité de mes collègues développeurs, seuls capables d'effectuer toute une batterie de tests manuels comparatifs entre les serveurs déployés via Opswork et ceux via Bamboo.

Au final, après environ deux mois de migration, IJM fonctionne sur les environnements Dev et AlphaSearch, les autres environnements n'ont pas encore été testés.

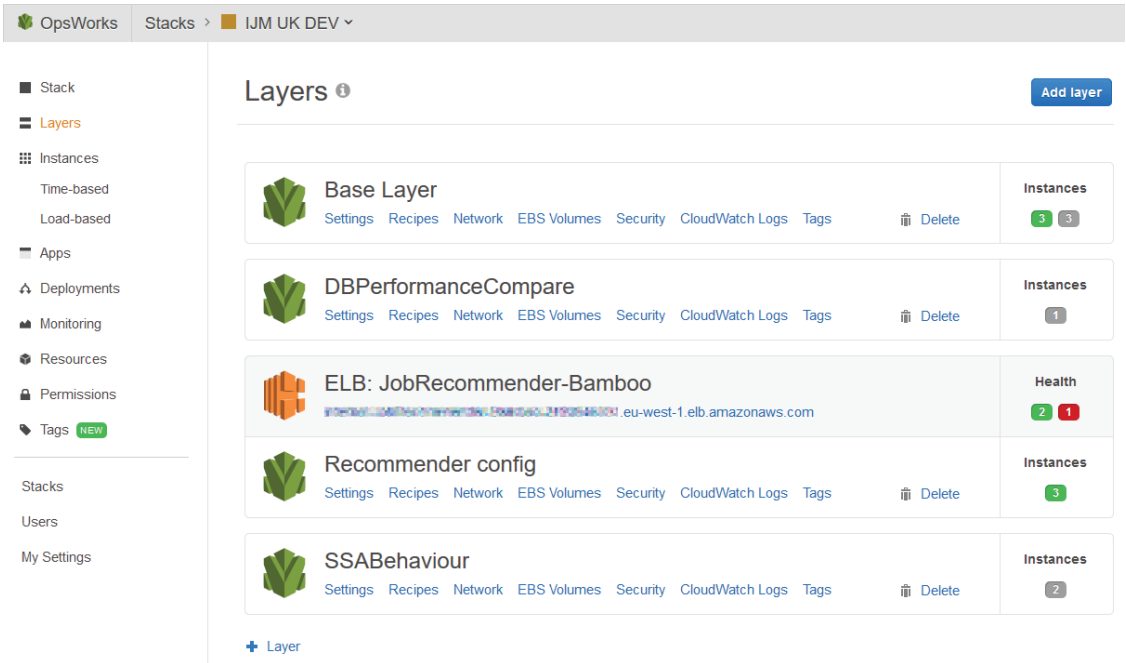

Illustration 3: IJM danss OpsWork : Différentes couches d'un environnement

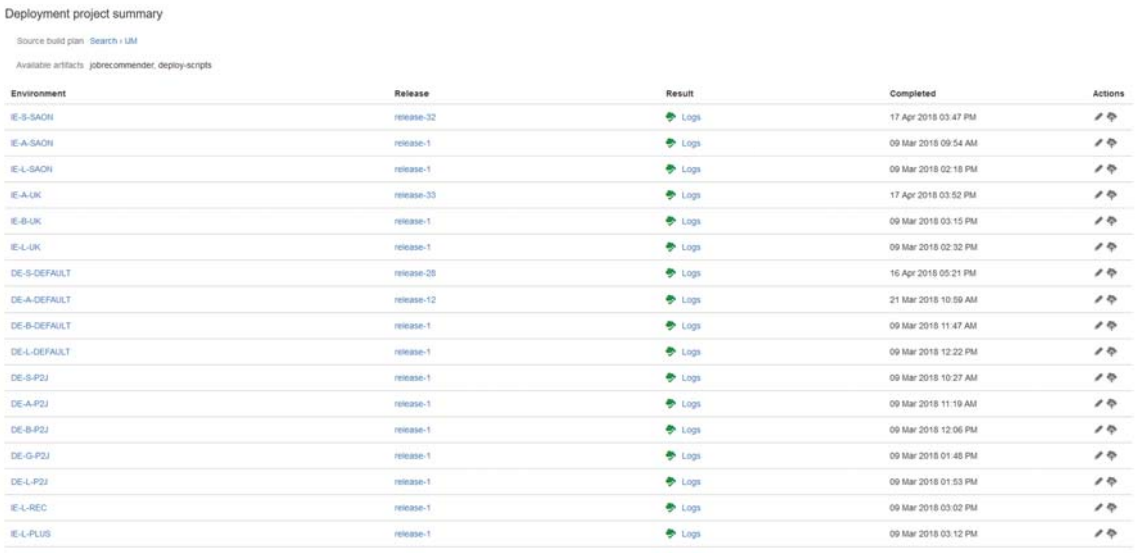

Illustration 4: IJM dans Bamboo : Tous les environnements migrés

Blue-green

Voir https://www.grossum.com/blog/continuous-delivery-blue-green-deployment-or-how-you-can-test-on-production-and-not-kill-your-project

Afin de rendre plus robuste notre stratégie de déploiement, il m'a été demandé d'étudier des solutions pour mettre en place du *blue-green deployement*. Comme décrit sur l'URL plus haut et dans le cours de M. Thiry, du *blue-green deployement* consiste à faire coexister deux infrastructures en production :

- une active qui est utilisée par les utilisateurs
- une en sommeil sur laquelle les développeurs peuvent travailler

Pour déployer une mise à jour dans ce genre d'environnement, il suffit d'appliquer la mise à jour dans l'environnement en sommeil, aucune stratégie n'est nécessaire : éteindre les anciennes machines et déployer des nouvelles. Le down-time sur cet environnement n'impacte pas la production puisqu'il n'est pas utilisé.

Une fois la mise à jour déployée, on peut lancer une batterie de tests d'intégration pour vérifier que tout est ok. Une fois le feu vert donné, on change d'environnement au niveau d'un load-balancer : l'environnement en sommeil devient l'environnement de production et l'environnement de production devient l'environnement en sommeil.

Pour mettre en place du blue-green deployement dans AWS, je me suis basé sur notre stack hautement-disponible où j'ai simplement dupliqué les *targets-groups* et les *auto-scaling-groups*. Pour opérer la transition il suffit de faire pointer les *listeners* de l'*elestic load balancer* sur le target group du blue ou du green.

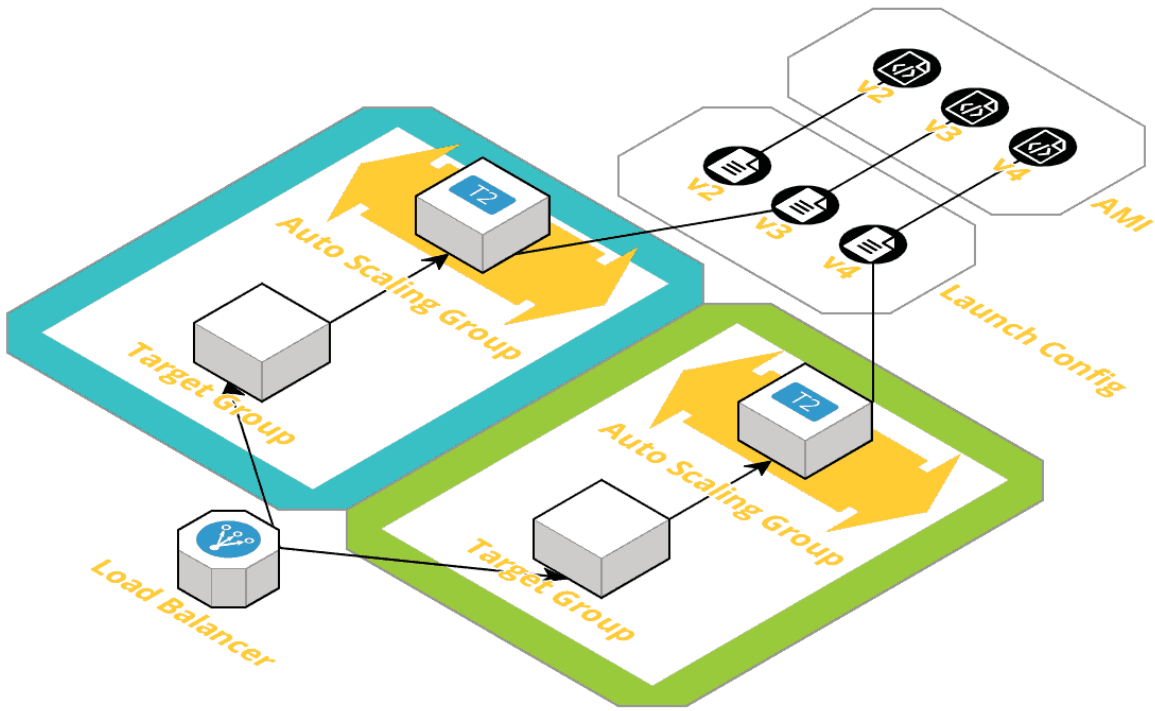

Illustration 5: Structure blue-green au sain d'AWS

Plusieurs problèmes se sont présentés vis-à-vis de cette structure :

- Le premier problème est que certains projets utilisent plusieurs target groups (par exemple nginx en utilise deux: un pour le trafic HTTP et un pour le trafic HTTPS). J'ai dû prendre en compte ces environnements dans mes projets.

- Le second problème s'est présenté au niveau des health-checks configurés sur les target groups, ceux-ci ne s'effectuent qu'à la condition d'être liés à un listener en amont, la solution a été de mettre en place autant de listener qu'il n'y a de target groups dans l'environnement en sommeil écoutant sur un port fictif. Le port fictif étant filtré par le pare-feu, aucun trafic ne peut venir le l'extérieur.

Mon script a les capacités suivantes :

- Mettre à jour une infrastructure existante vers du blue-green sans impact sur l'environnement courant (`upgrade`)
- Déployer une nouvelle version sur l'environnement en sommeil (`deploy`)
- Inverser l'environnement actif et celui en sommeil avec vérification de tous les health-checks et arrêt automatique en cas d'erreur (`switch`)
- Arrêter l'environnement en sommeil (après un switch), l'arrêt n'est pas automatique pour permettre de `switch`-back manuellement en cas de problème (`scaledown`)

J'ai également codé un mécanisme de rollback automatique pour prévenir des erreurs : pour chaque modification de l'infrastructure, la modification inverse est sauvegardée, en cas d'erreur non gérée l'ensemble des modifications inverses sont jouées pour rétablir le système dans son état d'origine. J'ai mis en place ce mécanisme en réaction à une erreur professionnelle : la suppression manuelle d'un ELB de production à la place de l'ELB que j'utilisais pour mes tests (plus de détails dans les faits saillants).

Aujourd'hui, *blue-green deployement* est utilisé pour les projets :

- **nginx** front-end load-balancer (10 environnements, **1.700** requêtes par secondes, **360Gbps**, **40%** de tout le traffic search)
- **IJM** instant-job-match (17 environnements)
- **AWS-Access** service interne (2 environnements)

Mon script est utilisé comme script de déploiement par défaut pour chaque nouvelle migration.

Rolling-update, Playground et Inventory script

Arrivant vers la fin de mon stage, il m'a été donné des projets plus légers.

- J'ai mis à jour le script de rolling-update pour utiliser le SDK d'AWS (SDK qui n'était pas encore complet lorsque mes collègues avaient développé le script de rolling update original).
- J'ai créé un environnement de test pour pouvoir tester l'ensemble de nos scripts de déploiement ainsi que le comportement des serveurs nginx en cas d'erreur logicielle ou de problèmes réseaux.
- J'ai commencé à travailler sur une solution pour faire l'inventaire de nos serveurs de manière plus intelligente (actuellement un script fait l'inventaire de tous nos serveurs une fois par jour).

Fait saillant

Bien que je faisais très attention à tester mes scripts avant de les déployer, il m'est arrivé plusieurs fois de commettre des erreurs. Genre d'erreur qu'on ne peut pas faire lorsqu'on travaille en temps qu'ops.

Pour l'écriture du script de création d'infrastructure blue-green, j'avais besoin de travailler à la fois sur le compte dev où j'essayais de faire fonctionner mon script et le compte live où je récupérais les informations sur l'infrastructure en place. Plusieurs fois mon script plantait en plein milieu et j'avais besoin de corriger des erreurs pour recommencer, or la création d'un load-balancer était la première étape de mon script : régulièrement j'ai dû supprimer le load-balancer créé par mon script à la main. Une fois par mégarde, j'ai supprimé le load-balancer du service sur lequel je travaillais... sauf que j'ai supprimé celui du compte live.

Le service s'est retrouvé indisponible pendant plusieurs longues minutes, étant concentré sur mon script je ne me suis pas rendu compte que des collègues se plaignaient de l'indisponibilité de ce service et c'est seulement quand un collègue DevOps a demandé à haute voix "C'est normal qu'il n'y ait plus de load-balancer ?" que je me suis rendu compte du problème en cours.

J'ai immédiatement mis mon travail de côté pour corriger ce problème, ayant l'exact schéma de ce load-balancer en tête j'ai pu le recréer en quelques secondes. J'ai ensuite développé une solution de rollback dans mon script pour prévenir de ce genre d'erreur humaine : en cas d'erreur, le script annule méthodologiquement chaque changement effectué. Cette fonctionnalité a été tellement prisée par mes collègues qu'ils l'ont également implémentée au script de rolling-update à l'époque utilisé pour les déploiements.

Cette fonctionnalité nous a permis de rendre nos scripts plus fiables de manière générale et beaucoup moins d'incidents de ce type se sont produits par la suite.

Conclusion

En conclusion le stage à StepStone m'a vraiment responsabilisé. J'ai appris à collaborer dans des équipes pluridisciplinaires, j'ai appris à communiquer pro-activement avec mes collègues pour régler les problèmes du quotidien et j'ai plusieurs fois endossé la responsabilité de donner le feu vert pour une mise en production.

J'ai également beaucoup appris sur la méthodologie agile et les pratiques DevOps.

Je recommande ce stage à quiconque veut mettre à l'épreuve ses compétences en administration système Linux et en scripting bash/Python.