

Rapport de stage en entreprise

Interface Web pour piloter l'outil de déploiement du jeu en ligne sur le Cloud

Vincent Munier : *vmunier@etudiant.univ-mlv.fr*
M2 informatique : parcours logiciel

31 juillet 2012

Maître de stage : Damian Arregui
Responsable de formation : Serge Midonnet



Résumé

le résumé ici, mettre le sujet du stage

Table des matières

| | | |
|----------|--|-----------|
| 1 | Présentation de l'entreprise | 3 |
| 1.1 | L'entreprise : Mimesis Republic | 3 |
| 1.2 | Le projet : Mamba Nation | 3 |
| 1.2.1 | Images de l'univers virtuel | 4 |
| 1.3 | Modèle économique | 5 |
| 1.4 | Partenaires | 5 |
| 1.5 | Les équipes | 6 |
| 1.5.1 | L'équipe Architecture | 6 |
| 1.5.2 | L'équipe Engine | 6 |
| 1.5.3 | L'équipe Infrastructure | 6 |
| 1.5.4 | L'équipe Studio | 6 |
| 1.5.5 | L'équipe Expérience | 6 |
| 2 | Les outils | 7 |
| 2.1 | Scala | 7 |
| 2.1.1 | Caractéristiques et points forts de Scala | 7 |
| 2.1.2 | Faiblesses du langage | 9 |
| 2.2 | Play! | 11 |
| 2.2.1 | Full stack | 11 |
| 2.2.2 | Choix libre de la technologie coté client | 11 |
| 2.2.3 | Architecture stateless | 11 |
| 2.2.4 | Rechargement à chaud et affichage des erreurs dans le navigateur | 12 |
| 2.3 | Simple Build Tool (SBT) | 12 |
| 2.4 | Amazon Web Services (AWS) | 12 |
| 2.5 | Chef | 13 |
| 2.6 | Jira | 13 |
| 3 | Méthodes de travail | 14 |
| 3.1 | Méthode agile Scrum | 14 |
| 3.1.1 | Sprints | 14 |
| 3.1.2 | Issues | 15 |
| 3.1.3 | Réunions | 16 |
| 3.1.4 | Glossaire | 18 |
| 4 | Le contexte : l'outil de déploiement du jeu en ligne | 19 |

| | | |
|----------|--|-----------|
| 4.1 | L'outil d'infra et AWS | 19 |
| 4.2 | L'outil d'infra et Chef | 19 |
| 4.3 | Usage | 19 |
| 5 | Mes réalisations | 22 |
| 5.1 | Présentation du framework Play! | 22 |
| 5.2 | Le déploiement | 22 |
| 5.2.1 | Amazon héberge l'application | 22 |
| 5.2.2 | Scripts Bash | 23 |
| 5.3 | Premier concept de webapp : appeler l'outil d'infra comme une commande shell | 23 |
| 5.3.1 | Communications asynchrones pour une application web temps réel | 24 |
| 5.4 | Composants de la webapp | 25 |
| 5.4.1 | Champs de saisie | 25 |
| 5.4.2 | Sélecteur de fichier | 26 |
| 5.5 | Restructuration du code JavaScript | 26 |
| 5.5.1 | RequireJS | 26 |
| 5.5.2 | CoffeeScript | 27 |
| 5.6 | Messages Done et Failed | 28 |
| 5.7 | Verroux sur les commandes à effet de bord | 28 |
| 5.8 | Transformer l'outil d'infra en bibliothèque | 29 |
| 5.8.1 | Des singletons transformés en simples classes | 29 |
| 5.8.2 | Libération mémoire du cache | 29 |
| 5.8.3 | Des types de retour significatifs | 30 |
| 5.9 | Fonctionnalités de la webapp | 30 |
| 5.9.1 | Complétion automatique des infras | 30 |
| 5.9.2 | Défilement automatique des logs de la commande en cours d'exécution | 31 |
| 5.9.3 | Contrôler les niveaux de log | 31 |
| 5.9.4 | La commande download-chef-log | 32 |
| 5.10 | Évolutions de l'outil d'infra | 33 |
| 5.10.1 | création d'infrastructure | 33 |
| 5.10.2 | optimisations en temps d'exécution | 33 |
| 5.10.3 | documentation dans le wiki | 33 |
| 5.11 | Plugin SBT | 33 |
| 6 | Conclusion | 34 |

1 Présentation de l'entreprise

1.1 L'entreprise : Mimesis Republic

Co-fondée par Nicolas Gaume et Sébastien Lombardo en 2007, Mimesis Republic est une société française basée à Paris et Bordeaux, spécialisée dans les technologies plurimedia (réseaux sociaux, web, jeux vidéo). Composée d'une équipe de 47 personnes et d'un management expérimenté, à l'origine de nombreux blockbusters dans le secteur de l'entertainment depuis plus de 15 ans, Mimesis Republic a placé le projet Mamba Nation au cœur de sa stratégie à destination des adolescents et des jeunes adultes. Mimesis Republic est financée par des investisseurs privés issus du monde de l'entreprise.

1.2 Le projet : Mamba Nation

Mamba Nation est un univers dédié aux adolescents et jeunes adultes qui ambitionne de relier la vie réelle et le monde virtuel avec les réseaux sociaux comme passerelle.

Cet univers virtuel est accessible au sein d'un navigateur internet à travers une applet Java. L'intégralité de son contenu est *streamé*, aucun client n'est à installer sur l'ordinateur de l'utilisateur.

Mamba Nation c'est le renouveau du réseau social qui se construit sur Facebook en y amenant l'Avatar, les jeux, les outils, les marques dont les ados et les jeunes adultes ont besoin. La véritable innovation pour les jeunes comme pour les marques, ce n'est pas de choisir entre le réel et le virtuel, c'est d'utiliser le virtuel pour positiver et enrichir sa vie réelle. C'est ce que nous appelons l'EXTRA LIFE.

Dans Mamba Nation, Vous pouvez vous faire des amis, vous amuser ensemble, et même obtenir de la reconnaissance à travers une multitude d'actions et d'interactions. En résumé « FRIENDS + FUN + FAME ».

Ici le jeu n'est pas une fin, c'est un moyen permettant à chacun de s'engager et révéler sa véritable personnalité, son véritable caractère. Chaque utilisateur peut donc créer son Avatar, voir ses différents avatars, chatter, inviter des amis, draguer, danser et chanter, etc. . .

Mamba Nation est génératrice de moments forts (virtuels) et partagés (réellement).

1.2.1 Images de l'univers virtuel



FIGURE 1.1 – Tchater dans la Nation



FIGURE 1.2 – Une partie de ping pong dans la Nation



FIGURE 1.3 – Regarder une vidéo Dailymotion en compagnie de ses potes

1.3 Modèle économique

La société mise sur un modèle Freemium, l'accès au service est gratuit, mais pour obtenir ou utiliser certains biens virtuels tel que des vêtements ou des animations d'avatar, les utilisateurs doivent acheter des "Blings" ou des "Vibes", les monnaies virtuelles du site. La société vise un taux de 3 à 10% d'utilisateurs payant pour financer son service et être rentable sous 2 ans.

La société mise également sur la publicité et les événements qu'elle organisera dans ses salons 3D grâce à ces nombreux partenaires. Elle parle notamment de concerts privées d'artiste avec leur avatar.

1.4 Partenaires

Mimesis Republic s'est entouré de partenaires importants pour développer son produit notamment Xavier Niel, fondateur et vice-président du groupe Iliad, via son fonds d'investissement Kima Ventures, Marc Simoncini, P-DG et fondateur du site de rencontres Meetic, via son fonds d'investissement Jaina Capital, Steve et Jean-Émile Rosenblum, fondateurs et dirigeants de Pixmania via leur fonds d'investissement Dotcorp Asset Management, François Pinault, via sa holding Artemis S.A, Laurent Schwarz co-fondateur d'Alten, Jean-François Cécillon ancien président d'EMI France, Pascal Nègre P-DG de Universal Music France.

La société a annoncé qu'elle avait signé des partenariats pour enrichir son univers virtuel avec :

- **Universal Music** pour promouvoir et lancer des artistes, réaliser des événements musicaux, etc.
- **Allociné** qui va animer des salons communautaires en 3D autour de séries TV et de films
- **Puma** qui va associer la Mamba Nation à ses campagnes publicitaires, et apporter des items

virtuels

- **Trace** qui va relouer l'habillage de ses chaînes de télévision (Trace TV, etc.) et de sa filiale de téléphonie mobile Trace Mobile
- **DailyMotion** qui va streamer de la vidéo au sein de l'expérience

1.5 Les équipes

TODO

1.5.1 L'équipe Architecture

1.5.2 L'équipe Engine

1.5.3 L'équipe Infrastructure

1.5.4 L'équipe Studio

1.5.5 L'équipe Expérience

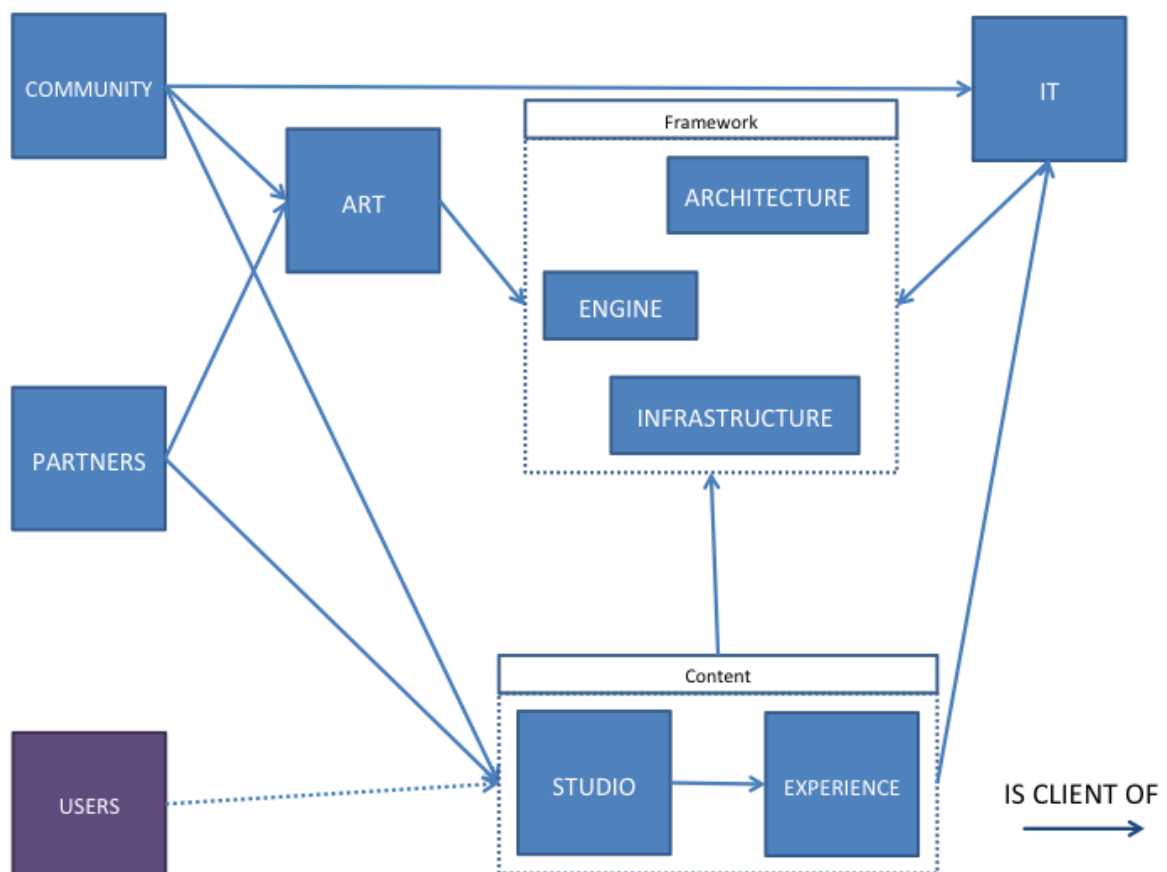


FIGURE 1.4 – Les relations client/fournisseur

2 Les outils

Les outils occupent une part importante du stage. L'équipe de Mimesis Republic est à la pointe de la technologie et utilise de nombreux outils pour accélérer le développement.

La plupart de ces outils sont open source, un certain nombre sont assez jeunes et rarement utilisés par les entreprises en général, qui ne prennent probablement pas le risque de s'engager avec des nouveaux outils.

Ceux-ci demandent effectivement un effort d'apprentissage et d'adaptation au début mais une fois maîtrisés, le gain de productivité est réel.

Je décris ci dessous chacun de ces outils, classés par temps d'utilisation dans le travail que j'ai effectué. Les outils que j'ai le plus utilisés sont présentés en premier et sont décrits de façon plus complète, avec leurs avantages et inconvénients.

2.1 Scala

Scala est le langage de programmation utilisé pour développer l'univers *Mamba Nation*. Le moteur 3D du jeu est codé en Scala, les mécaniques de gameplay sont codés en Scala, l'outil d'infrastructure coté serveur est aussi codé en Scala.

C'est un langage de programmation multi-paradigme conçu à l'École polytechnique fédérale de Lausanne (EPFL). Son nom vient de l'anglais Scalable language qui signifie "langage évolutif" .

2.1.1 Caractéristiques et points forts de Scala

- **Tourne sur la JVM :**

Il est prévu pour être compilé en bytecode Java (exécutable sur la JVM). Il existe aussi un portage sur la machine virtuel .Net qui reste néanmoins moins abouti que pour la JVM.

Si on souhaite utiliser Scala exclusivement avec la JVM, il est alors possible d'utiliser les bibliothèques écrites en Java de façon complètement transparente. Ainsi, Scala bénéficie de la maturité et de la diversité des bibliothèques qui ont fait la force de Java depuis une dizaine d'années.

- **Tout dans les bibliothèques :**

Le coeur du langage est minimal, la plupart des fonctionnalités majeures se trouvent être implémentées dans des bibliothèques.

C'est un point important pour l'évolutivité du langage.

Un exemple : les collections standards scala sont faciles à utiliser, concises et semblent faire

partie du langage même si elles sont en fait implémentées dans une bibliothèque. Quelques lignes suffisent pour partitionner une liste *people* en deux autres (*minors* et *adults*), en fonction de leur age :

```
class Person(val name:String, val age:Int)
val people = List(
  new Person("Hervé", 22),
  new Person("Julie", 17),
  new Person("Greg", 18))
```

```
val (minors, adults) = people partition (_.age < 18)
```

Ce morceau de code demanderait plusieurs boucles imbriquées en Java.

Ici, List est juste une classe et partition une méthode.

Ces fonctionnalités étant toutes implémentées dans des bibliothèques, le programmeur peut lui aussi écrire ses propres bibliothèques qui seront faciles d'utilisation, concises et extensibles.

- **Orienté objet pur :**

- Toute valeur est un objet
1.hashCode renvoie 1
- Toute opération est un appel de méthode :
le compilateur remplace $1 + 2$ par $(1) .+ (2)$

Les exceptions à ces règles de Java ont été supprimées :

Plus de primitives comme int, float (Int et Float à la place).

Plus de qualificateur “static” mais un mot clef “object” qui équivaut à créer automatiquement un singleton thread safe en scala.

Ces quelques changements contribuent à rendre le langage plus extensible que Java.

- **Fonctionnel :**

Toutes les fonctions sont des valeurs et puisque toute valeur est objet en Scala, toute fonction est aussi un objet. Scala offre une syntaxe légère pour les fonctions anonymes, supporte les fonctions de premier ordre, possède les case classes et le pattern matching.

Par contre la récursion terminale n'est pas complètement supportée à cause du manque de support de la JVM pour la récursion terminale. Le compilateur Scala optimise les cas simples d'appels terminaux en boucle while.

- **Statiquement typé :**

Scala est un langage fortement typé qui permet d'assurer une certaine sécurité dans les programmes, offrir de bonnes performances (aujourd'hui les langages statiquement typés ont tendance à être plus rapides que les langages dynamiquement typés) et facilite le refactoring de code car de nombreuses erreurs sont signalées par le compilateur lorsque un changement est effectué.

Les types ne doivent pas être nécessairement indiqués car ils peuvent être inférés automatiquement par le compilateur dans la majorité des cas.

L'indication des types reste obligatoire pour les paramètres de méthodes.

- **Autorise la Surcharge d'opérateurs :**

Tout opérateur est une fonction. Une surcharge d'opérateur n'est rien d'autre qu'une surcharge de méthode en Scala.

- **Possède un interpréteur en ligne de commande :**

Comme python et son ipython, Scala possède un environnement de programmation interactif en ligne de commande (REPL, read-eval-print-loop).

Le REPL facilite grandement l'apprentissage d'un nouveau langage puisqu'il donne de rapides retours sur le code entré.

- **Trait :**

Entre les interfaces Java et l'héritage multiple C++, les traits peuvent avoir des implémentations de méthode et des champs tout en rendant impossible l'héritage en diamant.

Ruby est un autre langage qui possède les traits.

- **Performant :**

Les performances de Scala sont comparables à celles de Java. Le paradigme fonctionnel de Scala incite à utiliser des données immuables et favorise l'activation de certaines optimisations de la JVM.

Papier publié par Google qui compare les performances de Java, C++, Go et Scala :

<https://days2011.scala-lang.org/sites/days2011/files/ws3-1-Hundt.pdf>

La principale caractéristique qui rend le langage évolutif reste que Scala fusionne les paradigmes fonctionnel et orienté objet à merveille, bien plus loin que n'importe quel autre langage.

Je pense en particulier aux programmeurs OCaml qui peuvent, à souhait, utiliser la programmation fonctionnel ou objet mais qui dans 90% des cas d'utilisations utilisent le côté fonctionnel du langage.

En Scala, la mixité entre les deux paradigmes est réelle avec une utilisation légèrement supérieure de l'orienté objet (estimation grossière : 60% objet, 40% fonctionnel).

Les paradigmes fonctionnel et orienté objet sont fondamentalement différents et chacun possède des caractéristiques qui facilitent la résolution d'un type de problème.

La complémentarité des paradigmes fonctionnel et orienté objet :

| fonctionnel | orienté objet |
|---|--|
| Facilite la production de code simple grâce aux fonctions de premier ordre, closures, et pattern matching | Facilite l'adaptation et l'extensibilité de projets complexes grâce aux classes, sous typage et héritage |

2.1.2 Faiblesses du langage

- Bien que le langage soit aujourd'hui mature et prêt à être utilisé en entreprise, peu ont fait le pas.

Toutefois, quelques entreprises bien connus utilisent aujourd'hui Scala. C'est le cas de Twitter, LinkedIn et Foursquare qui ont incorporés Scala au coeur de leurs projets.

Par exemple Twitter a transféré tout leur code coté serveur de Ruby en Scala, afin d'avoir les meilleurs performances possibles tout en gardant un langage extensible pouvant faire face à l'évolution croissante de leurs demandes.

Néanmoins, Scala reste pour l'instant un langage de niche avec une estimation de 0.237% d'utilisation pour le mois de juillet 2012 calculée par le site tiobe.com.

- Les plugins IDE Scala progressent lentement et restent faibles en fonctionnalités comparés à ce qu'offrent les IDE Java.
- Le compilateur du langage, scalac, est lent (approximativement 10 fois plus lent que javac pour un nombre de lignes de code équivalent)

Pour finir cette présentation du langage, voici deux citations de deux créateurs de langages :

James Gosling, créateur de Java :

"If I were to pick a language to use today other than Java, it would be Scala."

James Strachan, créateur de Groovy :

"I can honestly say if someone had shown me the Programming in Scala book by Martin Odersky, Lex Spoon and Bill Venners back in 2003 I'd probably have never created Groovy."

Équivalences de code Java, Scala :

| Java | Scala |
|--|--|
| <pre>boolean hasUpperCase = false for (int i = 0; i < name.length(); i++) if (Character.isUpperCase(name.charAt(i))) { hasUpperCase = true; break; }</pre> | <pre>// Utilisation des collections et closures Scala val hasUpperCase = name.exists(_.isUpper)</pre> |
| <pre>public class Person { private String name; private int age; public Person(String name, int age) { this.name = name; this.age = age; } public String getName() { return name; } public int getAge() { return age; } public void setName(String name) { this.name = name; } public void setAge(int age) { this.age = age; } }</pre> | <pre>// Scala crée automatiquement les getters et // setters pour 'name' et 'age' class Person(var name: String, var age: Int)</pre> |

2.2 Play !

Play ! est un framework MVC inspiré de Rails qui permet de créer facilement des applications web avec Java et Scala.

2.2.1 Full stack

Play ! embarque un serveur web, il se suffit à lui-même. Cette notion est appelée “full-stack”. Elle signifie que Play ! est autonome dans son mode de développement en proposant tout ce qu’il faut pour faire une application web, de la couche présentation à l’accès aux données. En production, les applications Play ! s’exécutent grâce au serveur web intégré.

2.2.2 Choix libre de la technologie coté client

Le framework Play ! ne s’occupe que de la partie coté serveur. Côté client, nous avons le choix des technologies (HTML5 + CSS3 + jQuery par exemple).

Play ! offre un support natif pour deux technologies clientes : CoffeeScript et LESS.

- CoffeeScript est un petit langage qui compile vers du JavaScript. Il propose une syntaxe concise et permet d’éviter les nombreux pièges de JavaScript. Il y a une relation 1-1 entre du code JavaScript et du code CoffeeScript.
- LESS est un petit langage lui aussi mais qui compile vers du CSS. Grâce à LESS, il est possible de rajouter des variables, des opérations (comme additionner des couleurs ensemble) et faire de l’héritage dans nos fichiers de styles.

CoffeeScript a été utilisé pour le développement du projet mais le besoin d’utiliser LESS ne s’est pas fait sentir.

2.2.3 Architecture stateless

Play ! est stateless, le serveur n’a aucune vision du client. Il n’existe pas d’état (session) côté serveur, l’état est stocké côté client (dans des cookies et cache) et/ou en BDD.

C’est la requête qui “porte” toutes les informations nécessaires à l’exécution de l’action :

`http ://monsite/donne/moi/utilisateur/12`

Le serveur reçoit une requête, traite la requête puis renvoie une réponse. Après envoi de la réponse, le client est oublié.

Dans une architecture stateful, l’historique d’activité du client sur le site web va être gardé côté serveur (il est allé sur tel page, puis il a cliqué tel bouton) alors que dans une architecture stateless, cette information n’est pas gardée.

Une architecture stateless permet d’avoir de meilleures performances et une faible consommation mémoire. Play ! offre de bonne performance car les requêtes peuvent être traitées :

- dans le désordre
- par 2 (ou+) serveurs différents

et Play! consomme peu de mémoire car :

- il n'y a pas de session sauvegardée côté serveur
- il n'y a pas de réplication de session

2.2.4 Rechargement à chaud et affichage des erreurs dans le navigateur



Le cycle de développement est rapide, édition → test → correction. Avec Play! il y a un seul endroit où consulter le résultat de son application : le navigateur.

Avec des frameworks web Java classiques (JSF par exemple), une erreur peut se produire à plein d'endroits différents :

- Est-ce qu'il faut regarder dans son Eclipse pour trouver une erreur de compilation ?
- Est-ce qu'il faut regarder dans son navigateur pour trouver une erreur de rendu ?
- Est-ce qu'il faut regarder dans sa console pour trouver une erreur à l'exécution et chercher la ligne qui nous intéresse dans une stacktrace gigantesque ?

Le cycle de développement est plus simple avec Play!. On édite le code dans son éditeur de texte, on sauvegarde et enfin on rafraîchi la page dans son navigateur et c'est dans son navigateur qu'on vérifie si il n'y a pas d'erreur.

Bien sur, nous pouvons aussi utiliser notre IDE favori et avoir le surlignement des erreurs mais un simple éditeur de texte suffit pour développer avec Play! (TextMate, Vim, Emacs).

2.3 Simple Build Tool (SBT)

SBT est un outil de build écrit en Scala et configurable en Scala. C'est l'équivalent de Maven.

Les fichiers de configurations SBT sont moins verbeux que leurs fichiers POM équivalents. Ils sont écrits en Scala et peuvent donc profiter de toute la puissance du langage pour exprimer n'importe quel besoin. Par exemple, la commande *compile* peut être surchargée pour générer des fichiers sources avant que l'étape de compilation suive.

2.4 Amazon Web Services (AWS)

TODO

2.5 Chef

Administrer un seul serveur est une tâche assez simple. Administrer un parc de serveurs est déjà beaucoup plus difficile et la tâche devient extrêmement complexe quand les systèmes sont hétérogènes, c'est à dire lorsque nous avons des systèmes différents Unix, Linux et Windows (ce qui sera toujours le cas au bout de quelques années au moins).

Imaginons que nous voulions changer l'IP de nos serveurs DNS, il nous faudra alors se connecter sur chaque serveur pour modifier le fichier `/etc/resolv.conf`. Il en est de même pour vérifier les droits de certains fichiers critiques, pour s'assurer qu'un programme est lancé sur tels serveurs.

Chef répond à ces problèmes qui ont occupé et occupent toujours des bataillons d'administrateurs système. Avec sa DSL (Domain-Specific Language) ruby, Chef réalise une abstraction d'opérations de base comme la manipulation des IPs, de la configuration DNS ou NTP, etc.

Un de ses points forts est que la même fonction Chef permet de manipuler des choses décrites différemment selon le système : une IP ne s'affecte pas de la même façon sous Linux que Solaris ou Windows.

L'administrateur système décrit son infrastructure avec la DSL ruby et Chef se charge de maintenir dans le temps la cohérence de cette infrastructure.

2.6 Jira

C'est un logiciel de gestion de projet qui se présente sous forme de page web accessible depuis internet.

Le chef de projet crée des Issues qu'il met en ligne dans Jira. Une issue, identifiée par un numéro unique, est la description d'une fonctionnalité à ajouter ou d'un bug à corriger. Cette description de l'issue contient notamment :

- la date de création de l'issue
- la priorité de l'issue (Must, Should, Would, Could)
- le nombre d'heures estimé pour réaliser la tâche
- le nombre d'heures restant
- le nombre d'heures loguées
- l'historique des modifications apportées à cette issue.

Chaque membre de l'équipe peut laisser un commentaire sur l'issue pour proposer des idées de solution.

Avantages : Les objectifs voulus sont clairs, ils sont définis et notés dans une issue. De fait, lorsque l'auteur poste une issue, il a réfléchi à la demande avant de l'écrire.

Une pratique de quelques chefs de projet est de faire les demandes à l'oral. Une demande orale est bien moins utile qu'une demande écrite car le chef de projet oublie facilement des détails et se trouve être moins précis que s'il avait pris le temps de choisir les bonnes phrases dans une demande écrite.

Quant au développeur, avoir une demande écrite lui permet de l'analyser ultérieurement.

3 Méthodes de travail

Mimesis Republic utilise une méthode de travail agile, appelée Scrum.

3.1 Méthode agile Scrum

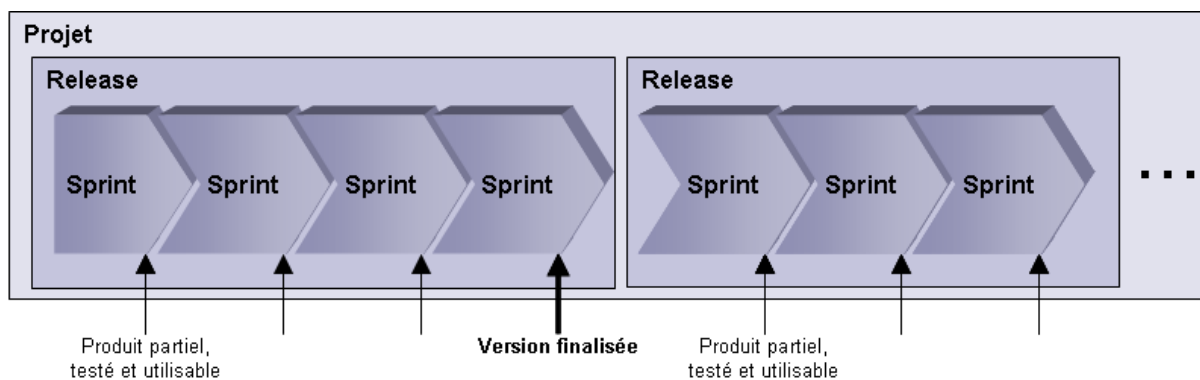
Scrum est une méthode agile dédiée à la gestion de projets. Parmi les méthodes de développement agiles existantes, on peut citer :

- Scrum
- Extreme Programming
- Adaptive Software Development (ASD)
- Dynamic System Development Method (DSDM)

Le terme Scrum est emprunté au rugby et signifie mêlée. Ce processus s'articule en effet autour d'une équipe soudée, qui cherche à atteindre un but, comme c'est le cas en rugby pour avancer avec le ballon pendant une mêlée.

3.1.1 Sprints

Scrum est un processus itératif : les itérations sont appelées des sprints et durent entre 2 et 4 semaines. Chez Mimesis Republic, les sprints durent habituellement 2 semaines.



3.1.2 Issues

Avant de commencer un sprint, on lui associe une liste de fonctionnalités qui devront être réalisées, tirées du backlog de produit (cf. Schéma Vue synthétique du processus Scrum).

Ces fonctionnalités sont sélectionnées pour être implémentées dans ce sprint.

Chaque fonctionnalité est découpée par l'équipe en tâches élémentaires de quelques heures et vont dans le backlog de sprint. Ces tâches élémentaires sont appelées des **issues** et peuvent être consultées à tout moment par les programmeurs.

L'image ci dessous présente la liste des issues qui m'ont été assignées. Chaque membre de l'équipe peut les consulter et y ajouter un commentaire, pour discuter de l'implémentation ou pour proposer des solutions possibles.
















| T | Key | Summary | Assignee | Created | P | Reporter | Status | Resolution | Updated | Due |
|---|----------|---|----------------|----------|---|----------------|-----------|------------|----------|---|
|  | MN-11831 | [INFRA] Cas particulier où le security group de l'application web n'est pas ajouté lors de la création d'infra | Vincent Munier | 02/07/12 | ↓ | Vincent Munier | Planned | Opened | 02/07/12 | |
|  | MN-11221 | [infra-admin-web] auto scroll down des logs | Vincent Munier | 08/06/12 | ↓ | Pierre Bittner | Closed | Fixed | 11/07/12 | |
|  | MN-11128 | MN-11015 / Faire le merge sur le head | Vincent Munier | 07/06/12 | ↓ | Pierre Bittner | Completed | Done | 08/06/12 | |
|  | MN-10744 | MN-10716 / créer une liste déroulante des infras disponibles | Vincent Munier | 25/05/12 | ↓ | Vincent Munier | Completed | Done | 31/05/12 | |
|  | MN-10743 | MN-10718 / impossible de lancer deux fois une même commande si celle-ci est déjà en cours d'exécution. | Vincent Munier | 25/05/12 | ↓ | Vincent Munier | Completed | Fixed | 25/05/12 | |
|  | MN-10741 | MN-10715 / Rajouter un champ optionnel clefs-valeurs pour la commande update-env. | Vincent Munier | 25/05/12 | ↓ | Vincent Munier | Completed | Fixed | 25/05/12 |  |
|  | MN-10719 | En tant que développeur, je souhaite avoir accès au log de chef dans l'outil d'infra en cas d'erreur lors du déploiement. | Vincent Munier | 25/05/12 | ↓ | Pierre Bittner | Submitted | Unresolved | 20/06/12 | |
|  | MN-10349 | MN-10189 / Enrichir les types de retour des méthodes de l'API | Vincent Munier | 10/05/12 | ↓ | Damián Arregui | Completed | Done | 25/05/12 | |
|  | MN-10346 | MN-10189 / Passer infra-admin en Scala 2.9 | Vincent Munier | 10/05/12 | ↓ | Damián Arregui | Completed | Done | 10/05/12 | |
|  | MN-9973 | MN-9770 / list (des infras) | Vincent Munier | 23/04/12 | ↓ | Damián Arregui | Completed | Done | 02/05/12 | |
|  | MN-9972 | MN-9770 / Valider une approche "end-to-end" GUI-WebApp-CLI | Vincent Munier | 23/04/12 | ↓ | Damián Arregui | Completed | Done | 02/05/12 | |
|  | MN-9969 | MN-9770 / create/destroy (sur les infras) | Vincent Munier | 23/04/12 | ↓ | Damián Arregui | Completed | Done | 25/07/12 | |
|  | MN-9968 | MN-9770 / update-env | Vincent Munier | 23/04/12 | ↓ | Damián Arregui | Completed | Fixed | 30/04/12 |  |

FIGURE 3.1 – Liste des issues qui m'ont été assignées

Une issue est la description très courte d'un besoin utilisateur.

Dans l'exemple ci-dessous, nous pouvons retrouver les différentes informations attachées à une issue.

Nous pouvons retrouver la date de création, le nombre d'heures estimé, le nombre d'heures loguées et bien d'autres informations.

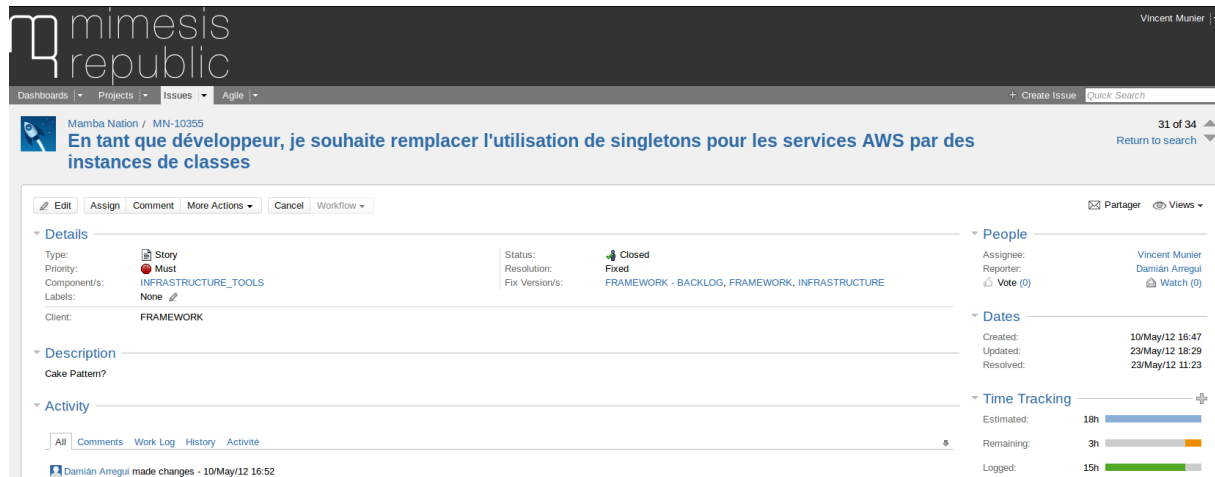


FIGURE 3.2 – Exemple d'issue qui m'a été attribué

Ce découpage de tâches de courtes durées est un des gros points forts de la méthodologie agile Scrum. En effet, il permet d'avoir un système testable qui peut être livré au client à chaque instant. Bien sur, nous devons prévenir le client que telle fonctionnalité n'est pas encore implémentée.

Mais le client peut ainsi nous faire des retours sur ce qu'il en pense, le plus régulièrement possible.

En cas de problèmes, il est bien plus facile de faire des interventions rapides que si le client donnait son avis tous les six mois.

Il peut également décider à tout instant de mettre son produit en production s'il le souhaite.

3.1.3 Réunions

Réunion chaque jour

Chaque journée de travail commence par une réunion de 15 minutes maximum appelée standup (Daily standup).

Le ScrumMaster donne la parole à chaque membre de l'équipe.

À tour de rôle, nous devons répondre à 3 questions :

- Qu'est-ce que j'ai fait hier ?
- Qu'est-ce que je compte faire aujourd'hui ?
- Quelles sont les difficultés que je rencontre ?

L'équipe se met ensuite au travail, elle travaille dans une même pièce.

Réunion entre deux sprints

À la fin du sprint, tout le monde se regroupe pour effectuer une réunion qui se déroule en deux étapes :

- La revue de sprint (sauf si c'est le premier sprint)

- La planification du prochain sprint

La durée de cette réunion est d'environ quatre heures.

Revue du sprint précédent

Le Scrum Master commence par faire une présentation de ce qui était attendu pour ce sprint. Puis il énonce les fonctionnalités qui ont été réellement implémentées, groupées par programmeur. Le Scrum Master donne ainsi la parole à chaque membre de l'équipe, pour qu'il donne un retour d'expérience, les tâches qu'il a accomplies avec succès, les imprévus qu'il a rencontrés.

Planification du sprint à venir

La réunion de planification consiste à définir d'abord un but pour le sprint, puis à choisir les fonctionnalités de produit qui seront réalisées dans ce sprint.

Dans un second temps, l'équipe décompose chaque fonctionnalité de produit en liste de tâches (items du backlog du sprint), puis estime chaque tâche en nombre d'heures.

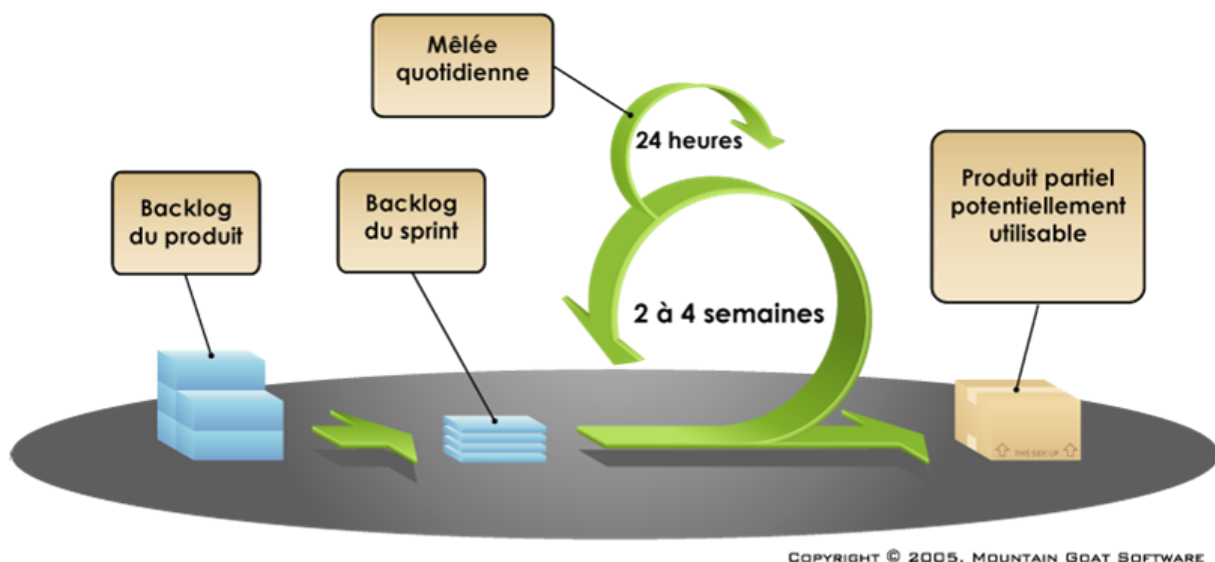


FIGURE 3.3 – Vue synthétique du processus Scrum

3.1.4 Glossaire

Backlog

Liste ordonnée de toutes les choses à faire. Il y a le backlog de produit qui énumère les exigences avec le point de vue du client et le backlog de sprint qui contient les tâches de l'équipe.

Directeur de produit / Product owner

Le représentant des clients. Littéralement le propriétaire du produit.

Release

Une release correspond à la livraison d'une version. Par habitude, on parle de release pour considérer la période de temps qui va du début du travail sur cette version jusqu'à sa livraison et qui passe par une série de sprints successifs.

Scrum

Scrum c'est la méthode mais c'est aussi le nom usuel de la réunion quotidienne limitée à un quart d'heure.

Sprint

Bloc de temps aboutissant à créer un incrément du produit potentiellement livrable. C'est le terme utilisé dans Scrum pour itération. Aux débuts de Scrum, un sprint durait 30 jours. La pratique actuelle est de 2 à 4 semaines.

Scrum Master

Il dirige les réunions scrum. Il s'assure que le projet avance comme prévu.

4 Le contexte : l'outil de déploiement du jeu en ligne

Les ingénieurs de l'équipe *infrastructure* ont conçu un outil en ligne de commande pour déployer une infra. Chaque équipe de l'entreprise utilise cet outil pour gérer leur propre infrastructure. Par exemple, l'équipe *engine* utilise l'outil de déploiement pour gérer leur infrastructure *engine*.

Un développeur de l'équipe *engine* peut exécuter la commande *status engine* pour se renseigner sur l'état de son infra. L'outil en ligne de commande affiche :

```
engine_all_0 (i-51e60a18 / 54.247.21.95) : running
Instance reachability check passed
System reachability check passed
```

Tout va bien, l'instance `engine_all_0` est bien en train de s'exécuter.

4.1 L'outil d'infra et AWS

L'outil d'infra utilise l'API Java AWS pour :

- créer, supprimer une instance
- récupérer le status d'une instance
- démarrer, arrêter une instance
- créer, supprimer un groupe de sécurité
- autoriser des IP et ports sur un groupe de sécurité
- lister les groupes de sécurité

4.2 L'outil d'infra et Chef

Toutes les informations concernant une infra sont stockées dans Chef. Lors de la création d'une infra, l'outil en ligne de commande stocke entre autres le nom de l'infra, la région AWS, le nom de la base de données et le nom du groupe de sécurité dans un Databag Chef.

L'outil d'infra utilise l'API REST de Chef pour récupérer des informations ou effectuer des actions.

4.3 Usage

L'outil d'infra en ligne de commande reçoit en entrées un nom de commande et ses arguments. Si la commande n'existe pas, l'outil d'infra affiche l'aide d'usage. Voici l'affichage de l'aide, on peut y voir toutes les commandes disponibles :

```
Usage: infra <cmd> [<infra>] [<file>] [<key>=<value>]*
```

Infra admin tool config file:

Config parameters must be specified in the 'infra.conf' file located in the 'config' directory.
It should contain one <field>=<value> definition per line.

Infra creation, pass config file:

create <file>

Create an infrastructure as defined in the given config file.

migrate <file>

Migrate databases for a particular environment in an infrastructure as defined in the given config file.

Infra management, pass infra:

start <infra>

Start the given infrastructure.

stop <infra>

Stop the given infrastructure.

status <infra>

Show status of the given infrastructure.

destroy <infra>

Destroy the given infrastructure.

Service management, pass infra:

start-services <infra>

Start services on the given infrastructure.

stop-services <infra>

Stop services on the given infrastructure.

restart-services <infra>

Restart services on the given infrastructure.

status-services <infra>

Show status of the services on the given infrastructure.

Database management, pass infra:

deploy-db <infra>

Run the 'deployDb' command on the given infrastructure.

liquibase-install <infra>

Run the 'liquibase-install' command on the given infrastructure.

liquibase-sync <infra>

Run the 'liquibase-sync' command on the given infrastructure.

liquibase-update <infra>

Run the 'liquibase-update' command on the given infrastructure.

run-migration-tool <infra>

Run the 'run-migration-tool' command on the given infrastructure.

Text management, pass infra :

deploy-wti <infra>

Export the last texts from WTI on amazon S3 ;

to really use them you have to modify the file "launch.js.php" on the "a" environment.

Environment management, pass infra and optionally updated values:

`show <infra> [<key>]*`

Show the "a" environment on the given infrastructure.

`update-env <infra> <file>`

Update the "a" environment on the given infrastructure as defined in the given config file.

`update-env <infra> [<key=value>]*`

Update the "a" environment on the given infrastructure as defined in the key-value arguments.

Various utilities, rarely used:

`list`

List all infrastructures.

`start-all`

Start all infrastructures and all services.

`stop-all`

Stop all infrastructures.

`register-dns <infra>`

Register DNS records for the given infrastructure.

`switch-dns <infra>`

Switch LIVE dns to point to the given infrastructure.

`export <infra>`

Export the databag item describing the given infrastructure from the Chef server to a file.

`import <infra>`

Import the databag item describing the given infrastructure from a file to the Chef server.

`export-databags`

Export all databags from the Chef Server to the file system.

`import-databags`

Import all databags from the file system to the Chef Server.

5 Mes réalisations

5.1 Présentation du framework Play !

Play ! est un jeune framework qui permet de créer facilement des applications web avec Java et Scala. Mon stage a deux buts majeurs pour l'entreprise :

1. Rendre accessible à tous l'utilisation de l'outil d'infra en proposant une interface web ergonomique.
2. Explorer le potentiel du framework Play ! pour qu'il remplace éventuellement du code php existant dans différents logiciels de l'entreprise.

Durant mes deux premières semaines de stage, j'ai préparé des slides pour une présentation du framework. Cette présentation met en avant les caractéristiques de Play !, ses points forts et faiblesses, et comment il se place face aux alternatives existantes. L'équipe croit au potentiel du framework mais attend le résultat de l'interface web de l'outil d'infra pour décider de remplacer le code php existant.

La semaine suivante consiste à tester l'outil d'infra en ligne de commande et à réfléchir à l'interface utilisateur pour l'application web (url disponibles, dispositions des boutons et des champs de saisies).

Une nouvelle branche est ajoutée dans Mercurial pour accueillir l'application web. Le développement peut commencer !

5.2 Le déploiement

Le déploiement doit être fonctionnel avant de développer l'application.

En effet, il se peut que les contraintes d'infrastructure empêche le déploiement d'une application : l'espace disque dur qui peut être insuffisant pour accueillir l'application, l'impossibilité d'ouvrir des ports http. Nous devons donc vérifier que le déploiement fonctionne bien de bout en bout.

Pour cela, une application simpliste est créée. Elle affiche juste une page web avec du contenu statique.

Il faut maintenant héberger l'application puis automatiser le déploiement en programmant des scripts.

5.2.1 Amazon héberge l'application

Sur Amazon Web Services, une instance EC2 portant le nom 'infra-admin-web' est créée. C'est en fait le serveur qui hébergera l'application. Le système d'exploitation de cette instance est Ubuntu 12.04 . C'est une instance de type *t1.micro* qui est idéale pour le type d'application web attendue. En effet, les clients de l'application web seront les membres de la société. Du fait du nombre limité d'utilisateurs, environ 25 développeurs, une puissance CPU limitée suffit.

Une instance Micro (*t1.micro*) fournit une petite quantité de ressources CPU constantes et augmente dynamiquement sa capacité CPU sur de courtes durées lorsque des cycles supplémentaires sont disponibles. Donc elle convient bien aux applications à moindre trafic ou aux sites consommant un nombre de cycles significatif périodiquement.

Le schéma suivant illustre l'utilisation CPU typique d'une application web tournant sur une instance de type *t1.micro* :

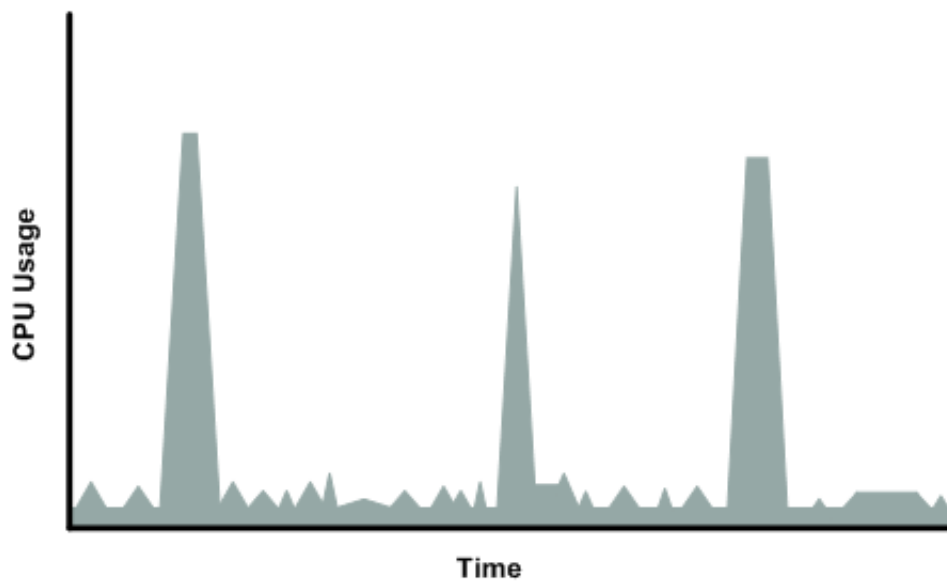


FIGURE 5.1 – utilisation CPU pour une instance de type micro

Les instances de type micro sont aussi les moins chers de tous les types d'instances proposés par Amazon. Leur prix est de seulement 2 centimes de dollars par heure.

5.2.2 Scripts Bash

Maintenant que nous disposons d'une instance EC2 réservée pour l'application web, il faut automatiser les actions récurrentes qui devront être effectuées.

Quatre scripts Bash sont créés pour exécuter les tâches récurrentes : `deploy.sh`, `start.sh`, `stop.sh` et `restart.sh` pour respectivement déployer l'application web, la démarrer, l'arrêter et la redémarrer.

`deploy.sh` est un script qui se trouve sur la machine local. Ce script package l'application puis la copie sur l'instance EC2 via `scp`. Les scripts `start.sh`, `stop.sh` et `restart.sh` se trouvent sur l'instance EC2 et peuvent être appelés à distance via `ssh`.

5.3 Premier concept de webapp : appeler l'outil d'infra comme une commande shell

Les objectifs de la première version de la webapp sont les suivants :

1. La webapp exécutera l'outil d'infra en ligne de commande comme un programme externe.
2. La webapp doit permettre d'exécuter deux commandes en parallèle.

Le code coté serveur est écrit en Scala. Puisque toute classe Java peut être naturellement instanciée dans du code Scala, c'est la classe `Java ProcessBuilder` qui est utilisée pour exécuter l'outil d'infra comme programme

externe. Qu'en au code coté client, il doit afficher la sortie de la commande dans le navigateur.

Une contrainte apparaît : Certaines commandes de l'outil d'infra prennent beaucoup de temps à s'exécuter. L'output de la commande doit donc être affichée au fur et à mesure de son exécution. Pour ce faire :

Coté serveur, un `BufferedReader` récupère dynamiquement la sortie de la commande.

Coté client, du JavaScript affiche progressivement le résultat.

Le point restant est la communication entre le serveur et le navigateur du client pour afficher dynamiquement du contenu dans la page web.

5.3.1 Communications asynchrones pour une application web temps réel

Dans une application Web traditionnelle, lorsque l'utilisateur effectue une action, celle-ci est exécutée et le navigateur attend le résultat pour rendre la main à l'utilisateur. Lorsque cette action requiert un calcul coûteux au serveur, cela occasionne des délais et une attente pour le client.

Le mode asynchrone élimine cette attente. Les requêtes au serveur sont lancées sans que soit suspendue l'interaction avec le navigateur et la page est mise à jour lorsque les données requises sont disponibles.

La première solution bien connue pour disposer de ce comportement asynchrone est *Ajax*. *Ajax* est une technologie permettant de créer des applications qui simulent un comportement temps réel. Le navigateur envoie une requête HTTP à intervalle régulier et reçoit une réponse. Il existe aujourd'hui des alternatives à *Ajax* pour voir des données se mettre à jour dans sa page web sans appuyer sur le bouton refresh.

La webapp de l'outil d'infra utilise des *WebSocket*. C'est une autre technologie web, bien plus récente, qui permet de créer de vraies applications temps réel. Alors que le protocole HTTP opère par une succession de requêtes et réponses alternatives, *WebSocket* est bidirectionnel : une connexion statique s'établit entre le serveur et le client et les deux parties envoient des données à leur convenance. C'est un authentique canal de communication bidirectionnel (push depuis le serveur) transparent pour les firewalls, proxy, et routeurs. Il n'y a aucune latence réseau lors des transmissions, les performances réseaux qu'elle offre est l'un de ses gros points forts. Le framework Play! fournit une bibliothèque complète pour utiliser les *WebSockets*. La webapp fait donc usage des *WebSockets* pour récupérer dynamiquement le résultat de la commande et l'afficher dans le navigateur.

Pour chaque nouvelle page ouverte, une *WebSocket* est créée. Deux utilisateurs peuvent alors exécuter leur commande en parallèle sans blocage ni intercalage des logs des commandes.

Voici une capture d'écran de l'interface web disponible avec la première version de la webapp :



FIGURE 5.2 – Exécution de la commande “list” avec la première version de la webapp

5.4 Composants de la webapp

5.4.1 Champs de saisie

La webapp propose 30 commandes réparties dans 7 catégories différentes.

| Catégories | Commands | | | | |
|-------------|------------------|--------------------|------------------|-----------------|-----------------|
| Creation | create | migrate | | | |
| Infra | start | stop | status | destroy | |
| Services | start-services | stop-services | restart-services | status-services | |
| Database | deploy-db | liquibase-install | liquibase-sync | | |
| | liquibase-update | run-migration-tool | | | |
| Text | management | deploy-wti | | | |
| Environment | show | update-env-file | update-env | | |
| Utils | help | free | list | start-all | stop-all |
| | register-dns | switch-dns | export | import | export-databags |
| | import-databags | | | | |

Lorsque l'utilisateur clique sur l'une des commandes d'une catégorie, il voit apparaître en haut de l'application plusieurs champs de saisie qui décrivent les arguments que doit recevoir la commande. Il y a un champ de saisie pour un argument. La commande *restart-services* prend le nom d'une infra en arguments. Elle possède donc un champ “infra...”. La commande *update-env-file* prend le nom d'une infra et le chemin du fichier de description de mise à jour de l'infra en arguments. Elle possède donc un champ “infra...” et un sélecteur de fichier.



FIGURE 5.3 – Commande update-env-file

5.4.2 Sélecteur de fichier

Un sélecteur de fichier est ajouté pour les commandes *create*, *migrate* et *update-env-file* de l'outil d'infra.

Lorsque l'utilisateur clique sur le bouton 'Execute', le fichier sélectionné est uploadé vers le serveur et la commande elle-même est exécutée coté serveur avec le fichier qui vient d'être téléchargé. Du code Ajax est utilisé pour ne pas avoir besoin de recharger la page lors de l'envoi du fichier. Ainsi le client peut recevoir l'output de la commande dans la page web.

5.5 Restructuration du code JavaScript

La quantité de code JavaScript est devenue conséquente. Une restructuration du code s'impose pour améliorer sa lisibilité, simplifier sa maintenance et faciliter l'ajout de nouvelles fonctionnalités.

5.5.1 RequireJS

Dans un langage de programmation comme Java, on ne se soucie pas du chargement des dépendances. Il suffit de les définir (import java.util.Collection par exemple) et la JVM s'occupe de charger les modules de façon complètement transparente pour le développeur.

Au contraire de ces langages évolués, la gestion des dépendances n'est pas une tâche simple en JavaScript. JavaScript ne possède pas de système de modules intelligent, ce qui rend le découpage de code difficile.

Prenons un exemple. Les dépendances sont représentées comme des flèches du module du client à gauche jusqu'au module requis à droite :

module1 → module2 → module3, module4

En JavaScript, à chaque fois que module1 est utilisé, tous les autres modules doivent être importés dans le bon ordre de la façon suivante :

```
<script type="text/javascript" src="module4"></script>
<script type="text/javascript" src="module3"></script>
<script type="text/javascript" src="module2"></script>
<script type="text/javascript" src="module1"></script>
```

Heureusement, il existe quand même des moyens de définir de telles hiérarchies de dépendances en JavaScript. RequireJS est une bibliothèque qui rend possible la définition de dépendances entre modules de manière approprié pour le navigateur. RequireJS est une sorte de #include/import/require pour JavaScript. Voici un extrait de code de l'application utilisant RequireJS :

```
define(
// La liste des dépendances
// en python: import jquery, bootstrap-typeahead, bootstrap-button, bootstrap-dropdown
['jquery', 'bootstrap-typeahead', 'bootstrap-button', 'bootstrap-dropdown'],
function($) {
// RequireJS assure que les quatre dépendances seront chargées et accessibles à
// l'intérieur de la fonction.

// définition de la fonction commands
function commands(webSocketURL, listInfrasSocketURL) = {
...
}
```

```
// Utilisation du return pour exporter la fonction commands.  
// Tout ce qui n'est pas exporté reste privé au module.  
return commands;  
}
```

Listing 5.1 – Définition du module commands avec RequireJS

Bénéfique pour la qualité du code

Des APIs et namespace plus propres

La définition de modules avec RequireJS force le développeur à réfléchir à la façon dont le module va partager les variables. Forcer le développeur à décider ce qu'il veut exposer et quels sont les détails d'implémentation spécifiques qui doivent être cachés conduisent à une meilleure encapsulation du code.

En plus, lorsque le développeur importe un module avec RequireJS, les propriétés et méthodes exportées par le module sont accessibles à travers une variable "package". Ça permet à deux modules différents d'exporter un attribut avec le même nom sans que l'un d'eux ne cache la valeur de l'autre.

Bénéfique pour les performances

Charger de plus petites ressources avec moins de requêtes

Deux principaux facteurs affectent le chargement d'une page :

- La taille des ressources. Le temps de chargement augmente avec la taille des ressources.
- Le nombre de ressources. Plus il y a de ressources plus le nombre de requêtes augmente.

La meilleur approche pour gérer le premier cas est de "minifier" le code. Minifier consiste à supprimer tous les caractères du code qui sont seulement utiles pour rendre le code plus lisible (des espaces et retours à la ligne entre autres).

Pour diminuer le nombre de requêtes HTTP, la solution habituelle (sans rentrer dans la mise en cache) est de regrouper tous les fichiers dans un seul sans modifier le code. De cette manière, le nombre de requêtes requises pour récupérer les ressources du serveur est réduit à une seule.

RequireJS fourni un outil pour automatiquement minifier et regrouper tous les modules en un seul. Cet outil est capable de minifier chaque fichier CSS du projet et minifier et regrouper tous les fichiers JavaScript dont les dépendances ont été définies en tant que module RequireJS.

5.5.2 CoffeeScript

Pour éviter de retomber dans les nombreux pièges rencontrés au cours du développement de la partie cliente en JavaScript, j'ai décidé d'utiliser CoffeeScript à la place de JavaScript.

CoffeeScript est un langage influencé par ruby qui fourni, entre autres choses, les compréhensions de listes, une meilleure gestion des variables sans polluer le namespace et plein d'autres outils qui rendent le développement JavaScript plus simple. CoffeeScript compile vers du JavaScript lisible et bien formaté, ce qui facilite le débogage. Tout le code JavaScript écrit jusqu'ici est traduit en CoffeeScript. Le code est plus concis (gain de ~15% pour le nombre de lignes de code) et le développement plus fluide.

5.6 Messages Done et Failed

Lorsqu’une commande se termine, l’utilisateur reçoit un message “Done” si la commande a réussi ou un message “Failed!” si la commande a échoué.

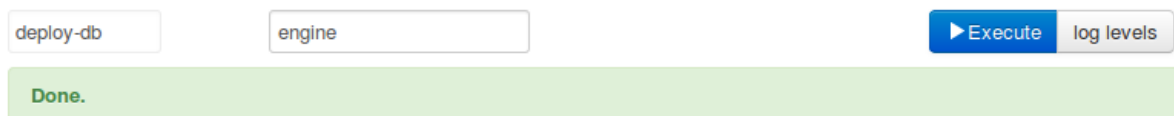


FIGURE 5.4 – Déploiement réussi de la base de données pour l’infra engine

Un message “Failed!” est accompagné de l’exception Scala émise par l’application.

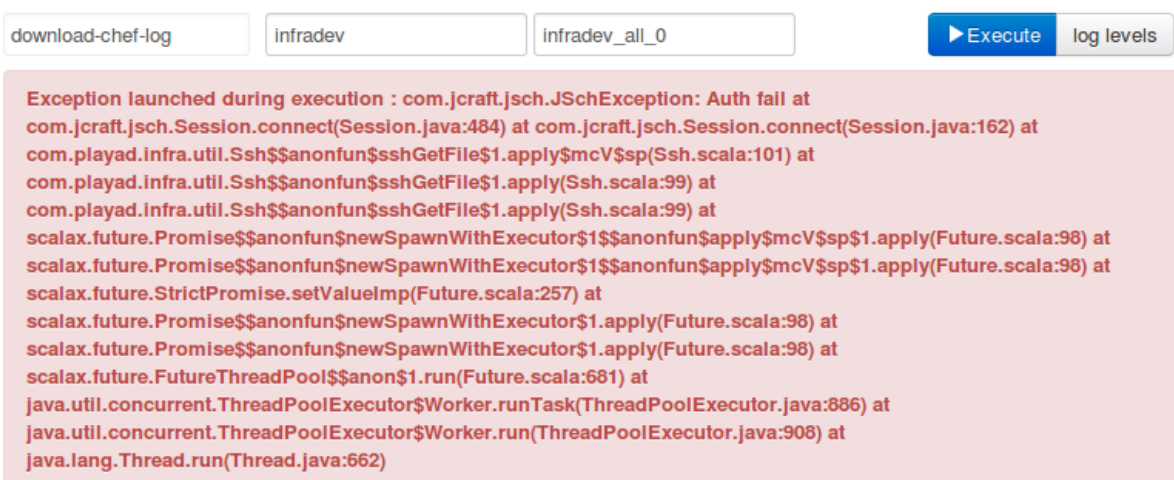


FIGURE 5.5 – Échec de la récupération des logs Chef sur le nœud infradev_all_0.

L’application ne peut apparemment pas se connecter sur l’instance EC2 de ce nœud car l’authentification a échoué

5.7 Verroux sur les commandes à effet de bord

La demande : interdire l’exécution de deux commandes identiques en même temps si celles-ci ont des effets de bord.

L’application web coté serveur enregistre les commandes en cours d’exécution. À chaque nouvelle exécution d’une commande, l’application verrouille toute autre exécution de la même commande. Lorsque la commande est terminée, qu’elle ait réussie ou qu’elle ait échouée, l’application enlève le verrou.

Si un utilisateur exécute une commande qui est déjà en cours d’exécution par un autre utilisateur, l’application renvoie alors un message d’alerte à l’utilisateur lui indiquant que sa commande est refusée.

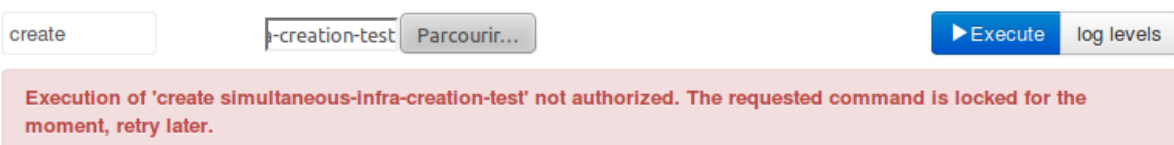


FIGURE 5.6 – La commande ‘create simultaneous-infra-creation-test’ est déjà en cours d’exécution.

Certaines commandes n'ont pas de verrou car elles n'ont pas d'effet de bord. Par exemple les commandes `status`, `status-services` et `show` n'ont pas besoin de verrou car elles ne font que récupérer des informations concernant une infra.

5.8 Transformer l'outil d'infra en bibliothèque

La demande : pouvoir appeler les méthodes de l'outil d'infra via l'application web. Mais, très important, l'outil doit toujours fonctionner en ligne de commande.

L'objectif : transformer l'outil d'infra en bibliothèque. Le jar de l'outil d'infra pourra ensuite être intégré à l'application web.

La transformation de l'outil d'infra en API impose plusieurs changements.

5.8.1 Des singletons transformés en simples classes

Toutes les classes étaient des services. Elles étaient toutes des singletons. Les services proposent des méthodes utilitaires et n'ont pas d'état. Il était donc logique d'avoir des singletons car nous n'avions pas besoin de plus d'une instance par service.

De même, il existait un singleton `Logger` qui était utilisé par tous les autres singletons afin de logger leurs différentes actions.

Dans l'outil d'infra en ligne de commande, un seul logger suffisait, un simple logger qui enregistre tout dans un fichier. Mais pour que l'outil se transforme en API, l'utilisateur doit avoir le contrôle sur le logger. L'utilisateur doit pouvoir choisir son logger, il doit pouvoir en utiliser plusieurs si il souhaite.

C'était en effet le cas de l'application web qui allait être le premier programme à utiliser l'API. L'application web a besoin de plusieurs loggers. Elle a en fait besoin d'un logger par internaute. Une nouvelle instance du `Logger` est créée pour chaque nouvel internaute afin que les résultats des commandes lancées par un internaute soient isolés dans son propre fichier. Les logs des internautes ne doivent pas être mélangés.

Un refactoring important a permis la transformation de l'outil d'infra en API. Tous les singletons sont transformés en simples classes. Puisqu'il n'existe plus de singleton, les constructeurs de ces classes prennent alors en paramètre les instances d'autres classes dont elles dépendent.

Par chance, l'outil d'infra a été développé avec un langage statiquement typé, Scala en l'occurrence. Lorsqu'un singleton se transforme en simple classe, le code extérieur qui utilisait ce singleton doit changer sa façon de l'appeler.

Grace au typage statique, Eclipse affichait les erreurs de compilation dans les différents fichiers faisant usage de la classe modifiée. Il suffisait alors de suivre ces erreurs et de les corriger.

5.8.2 Libération mémoire du cache

L'outil faisait aussi utilisation d'un cache pour l'optimisation de méthodes appelées plusieurs fois durant l'exécution d'une même commande.

Puisque ce programme était conçu pour exécuter une seule commande, la consommation mémoire de ce cache ne posait pas problème. À la fin de l'exécution de la commande, le programme se termine et la JVM se charge de libérer la mémoire allouée.

Dans le cadre d'une API, le problème est différent. Le client qui utilise l'API de l'outil d'infra ne souhaite peut-être pas exécuter une seule commande. Si le client exécute plusieurs commandes successives, le cache

grossit pour chaque nouvelle commande exécutée et consomme de plus en plus de mémoire.

C'est le cas de l'application web dont la durée d'exécution est *supposée* infinie. Une fois mise en production, l'application web doit être continuellement disponible pour les utilisateurs. Seul le déploiement d'une nouvelle version de l'application web doit engendrer son redémarrage.

Pour corriger ce problème, les données relatives à une commande sont supprimées du cache lorsque l'exécution de cette commande se termine.

5.8.3 Des types de retour significatifs

Plusieurs méthodes ne renvoyaient rien. Certaines d'entre elles ont été modifiées pour qu'elles aient un type de retour plus riche.

Par exemple, la commande 'list' affiche sur la sortie standard la liste de toutes les infrastructures existantes. Son type de retour était Unit (void en Java) et devient Seq[Infra] (en Java, le type le plus proche serait List<Infra>). Le client de l'API peut alors se servir de la liste des infrastructures retournée.

5.9 Fonctionnalités de la webapp

5.9.1 Complétion automatique des infras

L'application web fournit une complétion automatique des noms d'infrastructures pour les champs qui requièrent une infra.

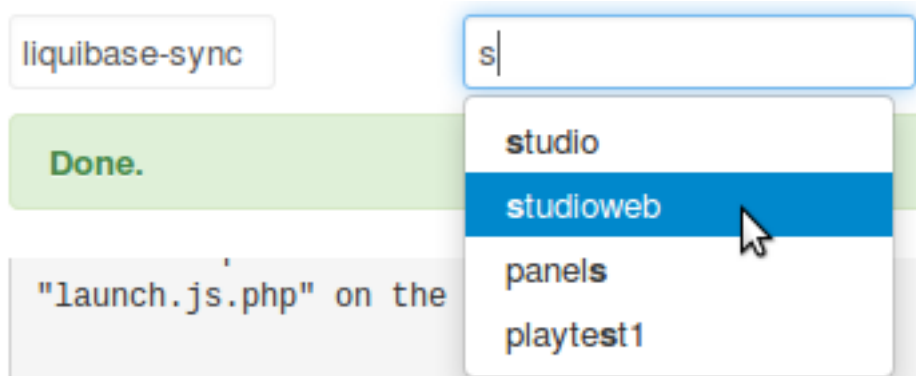


FIGURE 5.7 – Complétion automatique des noms d'infrastructures disponibles

La liste déroulante assiste l'utilisateur lorsqu'il doit indiquer l'infrastructure sur laquelle s'appliquera sa commande. Cette fonctionnalité améliore l'expérience utilisateur, l'utilisateur gagne du temps et évite les erreurs de frappe.

C'est le plugin JavaScript Typeahead de Twitter Bootstrap qui est utilisé pour avoir ce design sympathique. L'internaute tape quelques lettres de l'item qu'il recherche et le plugin affiche les items qui contiennent cette séquence de lettres en facteur.

Une fois le plugin inclus dans le projet, il faut renseigner l'attribut *data-source* de la balise input avec les différents choix possibles de la liste.

```
<input type="text" data-provide="typeahead" data-items="4" data-source='[]'>
```

Listing 5.2 – utilisation de bootstrap-typeahead

Comme présenté dans le code html ci-dessus, la liste est vide au début. Elle contiendra la liste des infras disponibles qui sera générée coté serveur.

Lorsque l'utilisateur se connecte à l'application, la page web lui est envoyée instantanément. Il peut exécuter la commande qu'il souhaite mais ne dispose pas encore de la complétion automatique des infrastructures.

Une WebSocket est utilisée pour remplir dynamiquement cette liste. En asynchrone, la commande *list* est exécutée coté serveur. Cette commande récupère la liste de toutes les infrastructures disponibles en requêtant Chef qui dispose de cette information. La commande *list* met environ 5 secondes à s'exécuter sur l'instance EC2 qui héberge l'application web. Une fois la commande *list* terminée, le serveur envoie cette liste des infrastructures au client via la WebSocket. Une fonction JavaScript renseigne alors l'attribut data-source avec la liste des infrastructures reçue.

Le client ne ressent aucune latence de l'application car les briques essentielles de l'interface web pour exécuter une commande sont instantanément disponibles. Les fonctionnalités complémentaires comme l'auto-complétion viennent se greffer dynamiquement à l'interface et n'altèrent pas la navigation de l'utilisateur.

5.9.2 Défilement automatique des logs de la commande en cours d'exécution



FIGURE 5.8 – Défilement automatique des logs - exécution de la commande status-services panel
L'image de droite est prise 4 secondes après celle de gauche

Lorsqu'une nouvelle ligne est ajoutée à la suite de l'output du résultat de la commande, la barre de défilement se met automatiquement en bas.

Si l'utilisateur déplace la barre de défilement, le défilement automatique se désactive. Si l'utilisateur replace manuellement la barre de défilement tout en bas, le défilement automatique est de nouveau actif.

5.9.3 Contrôler les niveaux de log

Avant d'exécuter une commande, les niveaux de logs peuvent être activés ou désactivés. Les informations affichées au cours de l'exécution de la commande sont alors filtrées pour n'afficher que celles correspondant aux niveaux de logs. Il y a cinq niveaux de logs utilisés dans l'application : Trace, Debug, Info, Warn et Error. Par défaut, tous les niveaux de logs sont activés.

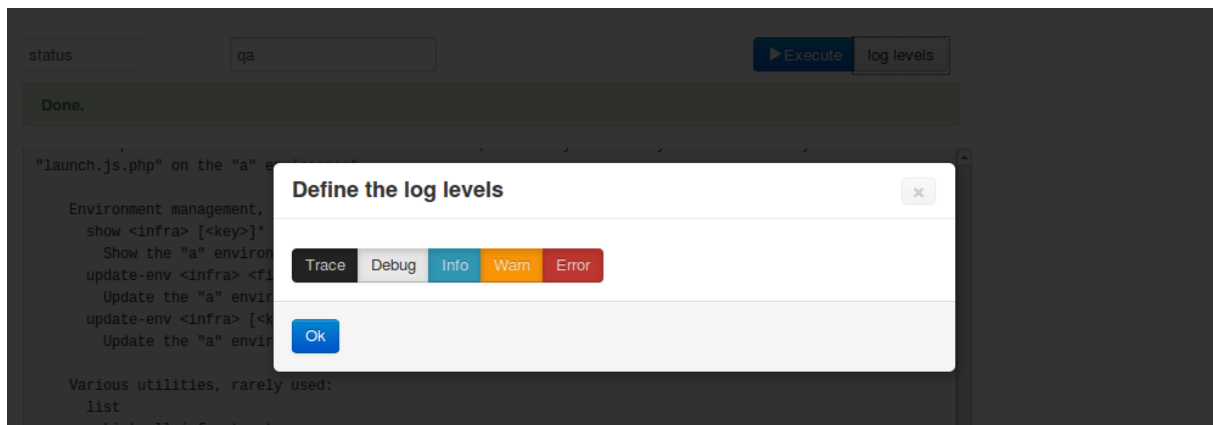


FIGURE 5.9 – Tous les niveaux de logs sont activés

La commande *status* fait un appel au `logger.debug` durant son exécution. Un utilisateur laissera le niveau Info actif et désactivera Debug si il souhaite uniquement recevoir le résultat du *status* et ne veut pas être encombré des tous les logs de Debug. Du coup, lors de l'exécution de la commande *status qaz*, la ligne **EXEC: Perform get on data/infrastructures/qaz** ne sera pas affichée car le niveau Debug a été désactivé par l'utilisateur.

5.9.4 La commande *download-chef-log*

La création et la mise à jour d'une infra sont des tâches délicates qui peuvent souvent échouer. La cause de cet échec peut varier. Ça peut venir par exemple de la création de la base de donnée RDS qui ne s'est pas correctement terminée comme ça peut aussi venir de la limite du nombre de groupes de sécurité fixée par Amazon qui est atteinte.

Mais la cause la plus fréquente est l'échec de l'exécution du chef-client. À chaque fois que ce programme plante, un administrateur système récupère le fichier de logs Chef pour connaître la cause. Il recherche sur amazon l'url de l'instance EC2 qui héberge l'infrastructure car c'est cette instance qui exécute chef-client et qui possède les logs Chef. Une fois cette url trouvée, il peut se connecter à l'instance via ssh et récupérer le fichier de log.

Pour éviter cette tâche fastidieuse aux administrateurs systèmes, la commande *download-chef-log* a fait son apparition dans l'application web.

Depuis l'interface web, il suffit de renseigner deux champs pour récupérer le fichier de log. Un premier champ contient le nom de l'infrastructure et un deuxième contient le nom du nœud sur lequel la commande a échoué.

Contrairement à l'url de l'instance qui n'était pas connu par l'administrateur système et qu'il devait rechercher sur Amazon, il connaît déjà le nom d'infra et le nom du nœud.

De plus, pour faciliter son utilisation, les deux champs de cette nouvelle commande *download-chef-log* possèdent la complétion automatique.

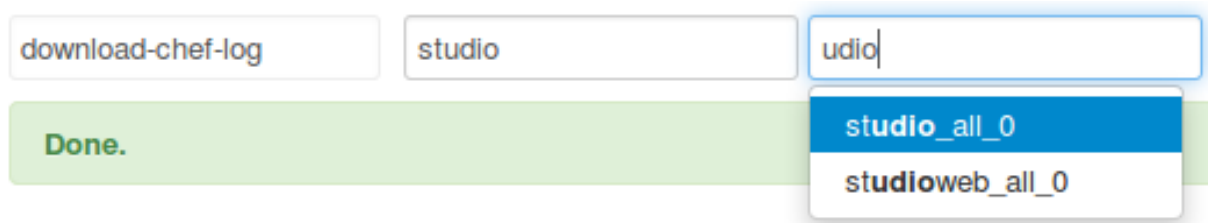


FIGURE 5.10 – Complétion automatique disponible aussi pour la liste des nœuds

À partir du nom de l'infra et du nœud, le serveur web requête Chef pour connaître l'url de l'instance EC2 correspondante. En utilisant la commande *scp*, le serveur copie le fichier de logs Chef distant dans un dossier local publique. L'application génère ensuite une URL qui pointe sur le fichier local téléchargé et envoie cette URL au client. L'URL est alors affichée sur la page web de l'utilisateur. Il lui suffit de cliquer dessus pour télécharger le fichier de log qu'il recherchait.



FIGURE 5.11 – Lien de téléchargement du fichier de log généré par la commande *download-chef-log*

5.10 Évolutions de l'outil d'infra

TODO

5.10.1 création d'infrastructure

5.10.2 optimisations en temps d'exécution

5.10.3 documentation dans le wiki

5.11 Plugin SBT

TODO

6 Conclusion

Après les deux premiers mois de stage, l'application web est rendu disponible. Les premiers retours sont positifs. De nombreuses idées d'améliorations font aussi surface de la part des nouveaux utilisateurs. Ces améliorations ont été implémentées au cours du stage.

L'outil web est maintenant utilisé quotidiennement par les différentes équipes de la société. C'est aujourd'hui l'outil le plus utilisé pour gérer son infrastructure.

En dehors de l'outil web, j'ai aussi contribué activement à l'évolution de l'outil d'infra en ligne de commande.

Ce stage fut aussi très enrichissant au niveau technique. J'ai appris à utiliser de nombreux outils évolués (SBT, AWS, Mercurial) et j'ai également aiguisé mes compétences par la pratique pour ceux que je connaissais déjà (Scala).

Lors de mon stage précédent chez Normation, je développais déjà en Scala et je souhaitais continuer à l'utiliser. C'est ce langage Scala qui m'a attiré et c'est pour pouvoir améliorer mes connaissances sur cette technologie que j'ai voulu travailler chez Mimesis Republic.

À l'avenir, je souhaite travailler pour une entreprise semblable à Mimesis Republic. C'est à dire travailler sur un projet ambitieux, aux cotés de programmeurs expérimentés et dans une équipe de taille moyenne tout en restant dans le cadre dynamique qui est celui des start-up.