ECE PARIS · LYON
ÉCOLE D'INGÉNIEURS

Rapport de Stage
Cycle Ingénieur • 2ème année
2019/2020

Elève ingénieur

Prénom : Alexander

Nom : Hoffmann

Majeure d'enseignement :

☒ SI ☐ OCRS
☐ SE ☐ FI
☐ EN ☐ SA

Entreprise d'accueil

Nom : Adaltas

Adresse : 6 Rue Jules Simon, 92100 Boulogne-Billancourt, France

Adresse du lieu de stage si différent :

Confidentiel : ☐ oui ☒ non

Rapport à remettre au tuteur de stage à l'issue de la correction : ☐ oui ☒ non

Signature du Tuteur de Stage (**obligatoire**) :

Description de la mission

As part of the internship, the student will participate in the architecture and deployment of the various components of the platform taking into account the requirements of security, fault tolerance and performance. To ensure the operation of the platform, he will set up streaming processing chains to collect, process, display and alert from events issued by the system.

Rapport de stage

Entreprise d'accueil :
Adaltas

MISE EN PLACE D'UNE SOLUTION D'AUTOMATISATION POUR LE DÉPLOIEMENT DE SYSÈMES BIG DATA

Auteur :
Alexander Hoffmann - alex@ahff.dev

Maître de stage :
David Worms - david@adaltas.com

14 septembre 2020

Remerciements

Les premiers pas dans le monde du travail se font rarement seul et sans aide ni soutien. Je souhaite remercier toutes les personnes ayant contribué à cette expérience professionnelle.

Je tiens tout d'abord à remercier toute l'équipe d'Adaltas pour son accueil, sa bienveillance et sa bonne humeur permanente. J'apprécie l'attention et la sollicitude qui m'ont été prodiguées par toute personne rencontrée.

Je voudrais ensuite exprimer ma sincère gratitude à David Worms, mon tuteur, pour la confiance qu'il a bien voulu m'accorder en acceptant de m'intégrer à Adaltas. Je le remercie pour sa grande disponibilité, sa patience, son soutien chaleureux et ses conseils avisés. Je tiens à lui exprimer ma profonde reconnaissance pour ses critiques constructives d'une rigueur absolue.

Mes remerciements s'adressent également à Prisca Borges pour son accueil, sa sympathie et ses conseils, ainsi qu'à Léo Schoukroun pour son encadrement et sa compréhension qu'il m'a accordé tout au long du stage.

En cette période inédite de crise sanitaire, j'ai eu la chance de pouvoir travailler avec une entreprise qui a su s'adapter aux difficultés posées par les mesures nécessaires à la protection de ses collaborateurs. Je suis reconnaissant d'avoir pu effectuer mon stage dans les meilleurs conditions possibles.

Résumé

Ce document décrit mon stage de deuxième année du cycle ingénieur effectué au sein de l'entreprise Adaltas. Cette dernière est une équipe de consultants experts en Open Source, Big Data et systèmes distribués. La société fournit à ses clients un savoir faire reconnu pour sa manière d'utiliser les technologies et pour convertir leurs cas d'usage en projets exploités en production, sur la façon de réduire les coûts et d'accélérer les livraisons de nouvelles fonctionnalités.

Au sein de cette structure, j'ai eu la chance d'évoluer en tant que développeur. J'ai travaillé sur Nikita [1], une librairie d'automatisation pour le déploiement de systèmes pour Node.js. Ce logiciel a été conçu dans le but d'aider les développeurs et les opérateurs à déployer des infrastructures et des logiciels Big Data de manière flexible et idempotente.

L'objectif de ce stage était de contribuer au développement d'un projet Open Source et ainsi d'en apprendre plus sur cet axe de valorisation. Adaltas est une société "open". Son engagement se construit sur les fondations d'un code source, d'une collaboration, de standards et d'une formation ouverts. Il s'agit d'une façon de penser et de travailler que je trouve intéressantes et dont j'apprécie les aspects. C'est également l'une des raisons pour lesquelles j'ai choisi d'effectuer mon stage chez Adaltas.

Mots clés : big data, déploiement, devops, infraops.

1. https://github.com/adaltas/node-nikita

Table des figures

Table des matières

Introduction

Mon intérêt pour la data science et plus généralement pour l'informatique et les nouvelles technologies m'a amené à chercher un stage dans une société qui en a fait son cœur de métier. Le début de ma recherche de stage a été le mois de juillet 2019. J'ai envoyé des candidatures à plus d'une centaine d'entreprises qui ont donné lieu à des entretiens chez Google, Facebook et Amazon entre autres. Les épreuves comportaient des questions algorithmiques appliquées à des problèmes ludiques qu'il s'agit de résoudre dans une durée impartie. Florent Diedler, enseignant en informatique à l'ECE, m'a d'ailleurs beaucoup aidé lors de la préparation à ces entretiens. C'est durant un cours de DevOps animé par Gregor Jouet, consultant chez Adaltas, que j'ai découvert cette entreprise, ses méthodologies et sa culture. Au mois de novembre, alors que nous travaillions sur un projet DevOps, Gregor Jouet a invité les élèves intéressés par les technologies data science et big data à lui envoyer leur CV. Quelques semaines plus tard, j'ai reçu un appel de David Worms, dirigeant de la société Adaltas, qui m'a proposé un entretien. C'est dans ce contexte qu'est parvenue ma demande de stage à Adaltas qui a décidé d'accepter ma candidature.

Ce document présente les travaux et missions effectués au sein de l'entreprise Adaltas durant mon stage se déroulant du 14 avril au 30 août 2020. Dans le cadre du stage, j'ai pu participer à l'architecture et au déploiement des différents composants d'un cluster en prenant en compte les impératifs de sécurité, la tolérance aux pannes et les performances. Pour assurer l'exploitation de la plateforme, j'ai mis en place des chaînes de traitement en streaming pour collecter, traiter, afficher et alerter à partir des évènements émis par le système.

La thématique de ce stage s'inscrit dans un contexte d'appréhension et d'approfondissement du monde de l'entreprise. Dans ce rapport, nous présenterons dans un premier temps, le contexte du stage, c'est-à-dire que nous décrirons l'entreprise d'accueil, nous étudierons son secteur d'activité et sa culture. Dans un second temps, nous traduirons les différents aspects de ma mission et les attentes de l'entreprise. Finalement, nous verrons les compétences et qualités acquises à la suite de ce travail ainsi que ma valeur ajoutée pour l'entreprise.

Chapitre 1

Présentation de la structure d'accueil : Adaltas

Fondée en 2004, Adaltas est une société d'expertise en High Tech construite à partir de deux idées :

- l'innovation comme facteur de différenciation décisif pour les entreprises ;
- la capacité à mobiliser les meilleurs talents comme condition de succès.

FIGURE 1.1 – Logo d'Adaltas représentant un oiseau. Adaltas signifie "vers le haut".

Adaltas aide ses clients à s'orienter dans le monde en perpétuelle évolution de l'Open Source, leur donnant les clés pour développer les meilleures solutions, qu'il s'agisse simplement d'écrire une application ou d'élaborer une plateforme de traitement stratégique à plus long terme. L'entreprise se définit comme un acteur du Big Data autour des technologies Hadoop et NoSQL.

Les équipes apportent une expertise sur l'analyse et le traitement des données, leur gouvernance, le développement et la gestion opérationnelle. Les consultants adhèrent à la culture DevOps et ils sont formés à la méthodologie SRE [1]. Ils accompagnent leur client dans la mise en place d'infrastructures et d'applications résilientes, conscients des rapides innovations apportés par la communauté Open Source et de la nécessaire stabilité des systèmes.

L'expertise d'Adaltas dans le domaine Big Data a commencé dès 2009 par l'accompagnement de la société EDF et la collecte des données Linky dit compteurs intelligents.

1. What is Site Reliability Engineering (SRE)?

En 2012, la société a entrepris l'exploitation d'une plateforme commune à l'ensemble du groupe EDF avec la mise à disposition des composants de l'éco-système Hadoop. Le nombre de composants s'est élargi avec le temps ainsi que les services et les cas d'usage qui ont accosté sur la plateforme sécurisé et multi-tenante.

Depuis 2014, l'équipe s'est élargie en accueillant des talents majoritairement formés à l'ECE, école dans laquelle Adaltas est à l'initiative du programme Big Data. Adaltas donne également des cours au Data Science Tech Institute[2] et à l'Université Paris-Sorbonne.

1.1 Objectifs de l'entreprise

L'acquisition d'un cluster à forte capacité répond à la volonté d'Adaltas de construire une offre de type PAAS pour disposer et mettre à disposition des plateformes de Big Data et d'orchestration de conteneurs. Les plateformes sont utilisées pour l'acquisition de nouvelles compétences, l'évaluation de nouvelles technologies, l'utilisation d'outils DevOps et la mise à disposition d'environnements de développement, de PoCs et d'exploitation. Elles hébergent des Data Lakes, des DataLabs, des traitements et des modèles de Data Science, des outils orientés DevOps ou encore des services applicatifs. L'objectif est de porter cette offre à maturité.

Dans le cadre de ses cours et formations, Adaltas s'intéresse au domaine Big Data et Data Science. Les cours effectuées au seins des différents établissements ont pour objectif de trouver des jeunes talents pour les faire monter en competence. Ainsi, la société cherche à recruter et former ses futurs consultants le plus tôt possible afin qu'ils soient opérationnels dès la fin du stage de dernière année d'école.

1.2 Une entreprise "open"

Adaltas est une société "open". Leur engagement se construit sur les fondations d'un code source ouvert, d'une collaboration ouverte, de standards ouverts et d'une formation ouverte.

Les entreprises et les gouvernements utilisent les technologies Open Source pour remplacer les solutions propriétaires. Initialement, ces acteurs ont été attirés par les réductions de coût et la promesse de s'affranchir de la dépendance d'un éditeur. L'Open Source est désormais central à la transformation digitale avec des avantages avérés dans la sécurité, la qualité, la personnalisation, la flexibilité, l'intéropérabilité, l'auditabilité et le soutien.

Adaltas maintient près de 50 dépôts Open Source sur GitHub et encourage chaque développeur et client à contribuer à ces projets.

2. https://www.datasciencetech.institute/

1.3 La culture d'Adaltas

Adaltas préserve un esprit familial qui privilégie toujours une vision à long terme. L'entreprise a pour vocation d'assurer le développement de chacun de ses consultants dans le respect de leur identité et de leur autonomie en mettant à disposition toutes les ressources nécessaires. Chaque service ou fonctionnalité proposé est le fruit d'une collaboration où chacun contribue aux idées des autres. L'objectif principal est de créer les meilleures expériences possibles pour les clients.

Le respect de ces valeurs est l'une des clefs de la performance d'Adaltas, de son ancrage dans l'air du temps et dans la société qui l'entoure.

1.4 Par rapport à l'épidémie de Covid-19

La réponse d'Adaltas face à l'épidémie de Covid-19 qui a touché la France au début de l'année 2020 était remarquable. La mise en place du télé-travail durant la période de confinement n'a pas affecté le déroulement de mon stage. Au contraire, j'ai apprécié la flexibilité apportée par le télé-travail et j'ai observé une augmentation de ma productivité durant cette période. Suite au déconfinement, j'ai demandé à continuer à travailler à distance. Au vu de mes prestations, David Worms a accepté ma requête.

Chapitre 2

Présentation de la mission

La mission principale de mon stage se concentrait autour du développement du logiciel d'automatisation de déploiement Nikita. Plus précisément, il s'agissait de migrer plusieurs packages, modules et actions ainsi que de *refactor* une partie du code vers une nouvelle version du logiciel.

2.1 Cahier des charges

Sur la base des différents objectifs présentés précédemment, un cahier des charges a été établi courant avril 2020 par David Worms. Les activités précisées sont les suivantes :

1. Refactoring du code source du package `engine` afin d'en améliorer la lisibilité et, par voie de conséquence, la maintenance, et à le rendre plus générique.
2. Migration des actions du module `file`, initialement contenue dans le package `core`, vers son propre package.
3. Amélioration de la documentation dans les packages `engine` et `file` dans le but de la rendre plus complète et compréhensible.
4. Conception de tests unitaires permettant de vérifier le bon fonctionnement des packages `engine` et `file`.
5. Délivrance d'un rapport de stage à l'ECE Paris et à Adaltas.

2.2 Planning

Le découpage temporel des missions, proposé au début du stage est décrit sur la figure 2.1. Ce planning a été formulé en fonction de ma progression prévisionnelle de l'apprentissage des technologies nécessaires au développement de Nikita. Lesdites technologies seront étudiées en détail ci-après.

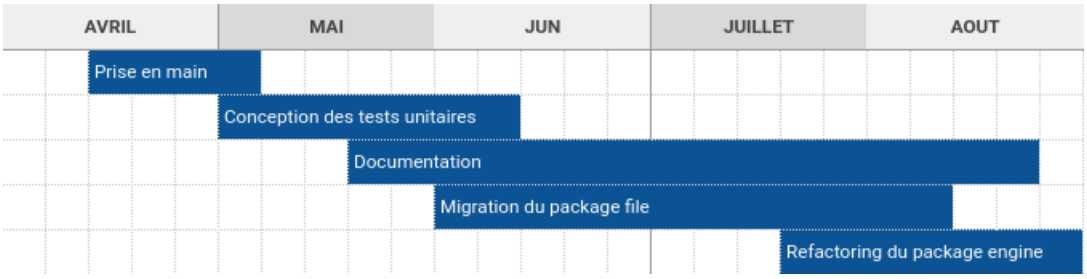

FIGURE 2.1 – Planning sous forme de diagramme de Gantt

2.3 Environnement de travail

La plus grande partie de mon stage s'est déroulée à distance. Quand j'étais sur les lieux de l'entreprise, j'ai eu l'opportunité de rencontrer et d'intéragir de façon privilégiée avec les différentes collaborateurs. Les bureaux à Meudon étant en cours de construction, les locaux se situent actuellement à Boulogne-Billancourt, près du Pont de Sèvres, dans un bâtiment qui loue des espaces de travail. Habituellement, David, Prisca et les stagiaires sont les seuls présents dans les locaux étant donné que les consultants sont en mission chez les clients.

Adaltas a suivi toutes les recommandations des collectivités locales et de l'Organisation mondiale de la Santé (OMS) pour mettre en place des mesures d'hygiène et de distanciation physique dans les locaux, afin d'offrir à ses collaborateurs un environnement de travail plus sûr. Ainsi, nous avons dû nous organiser à l'avance pour établir un calendrier de présence au bureau.

Les employés d'Adaltas ont des échanges quotidiens via le chat interne de l'entreprise (Keybase[1]). Ainsi, j'ai pu solliciter l'expertise de chacun lorsque j'ai rencontré des difficultés.

Nous avions un meeting tous les deux jours pour faire un point sur l'avancée de nos missions. Cette réunion dure entre 15 et 20 minutes. Chaque membre de l'équipe prend la parole à tour de rôle et décrit au reste de l'équipe ce qu'il a fait la veille, les objectifs qui ont été atteints, ce qu'il prévoit de faire aujourd'hui avec les nouveaux objectifs, et les éventuels problèmes qu'il rencontre. De cette façon, il est facile de savoir qui peut lui venir en aide et comment, afin de résoudre ses problèmes et de lui permettre d'avancer de nouveau.

2.4 Environnement de développement

Pour remplir ma mission, Adaltas a mis à ma disposition un ordinateur à la pointe de la technologie. Il s'agit d'une machine de la marque Dell comportant les caratéristiques suivantes :

- Processeur Intel Core i7-9750H de 9e génération ;

1. https://keybase.io/

- Disque SSD hautes performances M.2 PCIE 40 de 1 To ;
- 32 Go de mémoire, 2 x 16 Go, DDR4 à 2 666 MHz.

Ces spécifications techniques sont nécessaires car les stagiaires sont amenés à créer des clusters Big Data avec plusieurs noeuds nécessitant une plus grande puissance de calcul. Au début de mon stage, j'ai dû installer Arch Linux sur cet ordinateur. L'installation habituellement assez périlleuse car il faut mettre en place un grand nombre de services manuellement (notamment les services réseaux et l'interface graphique). Pour faire face à cela, Adaltas a développé une solution nommée Nikita Arch [2], un logiciel de déploiement pour le système d'exploitation Arch Linux. Arch Linux est plus simple que Debian ou Ubuntu car pacman ne touche pas à la configuration des paquets (ce que fait dpkg). Arch Linux est plus tolérant que Debian à propos des paquets "non-libres" tels que définis par GNU. Il est optimisé pour x86_64, et donc est plus rapide que Debian (i386). Les paquets d'Arch Linux sont plus récents que les paquets Debian. En revanche Debian est largement plus stable, c'est pour ça que l'on utilise souvent pour les serveurs.

2. https://github.com/adaltas/node-nikita-arch

Chapitre 3

Déroulement de la mission

Cette partie décrit les activités effectuées dans le cadre du stage. La plupart d'entre elles ont été réalisées simultanément. La description "chronologique" qui suit est donc très relative.

3.1 Présentation des outils utilisés

3.1.1 vim

Qui dit développement d'application dit environnement de développement « intégré » (abrégé EDI en français ou IDE en anglais, pour *integrated development environment*). Un bon IDE permet d'augmenter la productivité des développeurs. Il comporte un éditeur de texte destiné à la programmation, des fonctions qui permettent, par pression sur un bouton, de démarrer le compilateur ou l'éditeur de liens ainsi qu'un débogueur en ligne, qui permet d'exécuter ligne par ligne le programme en cours de construction.

Vim est un éditeur de texte directement inspiré de vi (un éditeur très répandu sur les systèmes d'exploitation de type Unix). Son nom signifie d'ailleurs Vi IMproved, que l'on peut traduire par « VI aMélioré ». A priori, Vim n'est pas un IDE mais un simple éditeur de texte. Cette affirmation serait vraie si Vim n'était pas autant personnalisable. En effet, l'ajout d'extensions, ou la modification de son fichier de configuration en fait un environnement de développement optimal. L'avantage est qu'il n'est pas nécessaire de maîtriser plusieurs IDE, Vim suffit.

Etant déjà que j'avais déjà quelques notions de Vim avant mon stage, j'ai été chargé de rédiger un tutoriel sur le site du cloud d'Adaltas. Cette introduction est destinée aux personnes n'ayant jamais utilisé Vim. Voici le lien vers mon tutoriel.

3.1.2 git

Git est un outil de contrôle de version similaire à CVS, Subversion et Mercurial. Cette famille d'outils s'appelle Système de Contrôle de Version (SCV) ou Gestion du Contrôle des Sources (GCS). Un contrôle de version permet de garder une trace des

modifications apportées à un ou plusieurs fichiers au fil du temps afin de pouvoir accéder ultérieurement à une version spécifique. Voici quelques exemples : un développeur veut garder une trace de l'évolution de son code ; un ingénieur DevOps veut déclencher des tests sur les changements publiés et déployer de nouvelles versions à partir de points d'accès bien définis de l'historique du logiciel ; un développeur web a besoin de stocker chaque version d'une image ou d'une mise en page ; un ingénieur infrastructure veut stocker et garder une trace de ses procédures de déploiement et des changements de configuration ; un Data Scientist veut enregistrer toutes ses expériences et les évolutions des fonctionnalités. Chez Adaltas, la procédure d'installation des systèmes Arch Linux utilisée sur la majorité de nos ordinateurs portables est stockée et partagée sur un dépôt public.

3.1.3 NodeJS

Node.js est une plateforme logicielle libre en JavaScript orientée vers les applications réseau événementielles hautement concurrentes qui doivent pouvoir monter en charge. Autrement dit, il s'agit d'un programme (écrit essentiellement en C/C++) qui est capable de lire du code JavaScript, de le compiler en JIT et d'exécuter les instructions correspondantes. Un tel programme s'appelle techniquement un environnement d'exécution.

Contrairement à une idée répandue, Node.js n'est pas que du JavaScript côté serveur. Il est possible d'écrire des applications Node.js "clientes", comme un client de messagerie. Par ailleurs, la plupart des outils en ligne de commande de l'écosystème JavaScript sont développés pour Node.js.

3.2 Présentation de Nikita

Nikita est une librairie d'automatisation pour le déploiement de systèmes pour Node.js. Le logiciel est développé en JavaScript. Le développement a commencé lorsqu'Adaltas a déployé un cluster Hadoop pour l'un de ses clients en 2011. Les actions Nikita sont variées : créer un dossier, générer un fichier, initier un service systemd, ajouter une règle de pare-feu iptable, etc.

Nikita est organisé comme un seul et unique dépôt multi-packages (monorepo) hébergé sur GitHub [1]. La figure 3.1 représente la structure du monorepo. Il comprend le moteur de base, les actions des utilisateurs et les fonctions utilitaires. Tous ces éléments sont associés à leurs tests unitaires. Lerna [2] est utilisé pour diviser le projet en packages indépendants. Cela permet d'optimiser les besoins en temps et en espace, permettant un refactoring massif, une mise à jour et un enrichissement des fonctionnalités plus efficace. La figure 3.2 représente un exemple d'une action Nikita.

1. https://github.com/adaltas/node-nikita
2. https://github.com/lerna/lerna

```
CHANGELOG.md
lerna.json
LICENSE
package.json
packages/
  core/
  db/
  docker/
  engine/
  file/
  filetypes/
  ipa/
  java/
  krb5/
  ldap/
  lxd/
  nikita/
  service/
  tools/
README.md
yarn.lock
```

FIGURE 3.1 – Structure de fichiers Nikita

```
nikita = require('nikita')
nikita.call({
  who: 'leon'
}, function({options}){
    console.info(options.who)
  }
)
```

FIGURE 3.2 – Exemple de code Nikita

3.2.1 Les packages

Comme évoqué précédemment, Nikita est divisé en plusieurs packages. Chaque package a un domaine d'application spécifique. Par exemple, le package `core` comporte les actions génériques, il permet de travailler sur des objets JavaScript, modifier des fichiers, etc. Le package `docker` contient des actions liées à Docker. Le package `file`, que je suis chargé de migrer, comporte des actions permettant de travailler sur le système

de fichier. Par exemple, nous pouvons y trouver des fonctions pour créer un fichier, supprimer un répertoire, télécharger un document, etc.

3.2.2 Les actions

Une action est un élément fondamental dans Nikita. Il s'agit essentiellement d'une fontion que nous appelons `handler` et des metadonnées associées appelées `options` ou `config`. Les actions génériques et bas niveau se situent dans le package `engine`. Par exemple, dans `engine/src/actions/fs/base/mkdir.coffee.md`, nous trouvons l'action `mkdir` qui permet de créer un répertoire.

3.3 Prise en main de Nikikta

Ma première activité consistait à comprendre la logique de Nikita. Pour pouvoir m'approprier et faire évoluer ce logiciel, il me fallait tout d'abord comprendre sa manière de fonctionner. Pour ce faire, j'ai dans un premier temps suivi les instructions du tutoriel [3] disponible sur le site internet. L'utilisation de Nikita nécessite l'installation de Node.js et de npm ou yarn. Ayant déjà utilisé ces technologies dans plusieurs cours dispensés à l'ECE, j'étais familié avec l'environnement de développement de Nikita fort de quoi j'ai pu avancer à un rythme plus soutenu durant ce tutoriel. Le fonctionnement de Nikita est fondé autour des fichiers. Toutes les actions de la librairie touchent de près ou de loin à un fichier. Ainsi, il est possible de suivre le déroulement d'une session Nikita en observant les modifications effectuées sur les différents fichiers.

Nikita est executé par le moteur Node.js. Cela signifie que le développement du logiciel nécessite des connaissances en JavaScript. Fort heureusement, les enseignements de l'ECE couvrent également les bases de ce langage de programmation. De surcroît, j'ai utilisé à titre personnel le langage JavaScript pour plusieurs projets. J'avais donc des fondations solides pour me lancer dans le code source. C'est en tout cas ce que je pensais. Il s'avère que, même s'il est possible d'utiliser Nikita en travaillant en JavaScript, le code source est écrit en CoffeeScript, un langage de programmation, qui se compile en JavaScript. En d'autres termes, le code écrit en CoffeeScript est transformé en JavaScript. Cela étant dit, les syntaxes entres ces deux langages sont très similaires comme le montre la figure 3.3.

3. https://nikita.js.org/about/tutorial/

```
a = 2
square = (x) -> x * x
b = square a
console.log b // 4
```

```
var a, b, square;
a = 2;
square = function(x) {
  return x * x;
};
b = square(a);
console.log(b); // 4
```

FIGURE 3.3 – Comparaison de code JavaScript et CoffeeScript

CoffeeScript a une syntaxe très claire et se prête parfaitement à l'aspect déclaratif de Nikita. Somme toute, le code source ressemble à du YAML, ce qui le rend beaucoup plus lisible tout en préservant les avantages d'un langage procédural tel que le JavaScript. Un autre avantage de CoffeeScript est qu'il permet d'intégrer la documentation liée à une action directement dans le code source.

Ma première mission pour prendre en main Nikita était la migration d'une action de Nikita Arch [4] vers Nikita dans le package `core`. Les actions contenues dans le dépôt Nikita Arch sont indépendantes et, de façon générale, plus simple que celles sur Nikita. J'ai ainsi pu me familiariser avec la logique et syntaxe du logiciel. Dans ce cas précis, je n'ai pas eu à écrire de nouveau code. Je devais prendre le code existant, le copier vers le package `core` et modifier certaines lignes relatives à l'environnement d'exécution. J'étais également chargé d'écrire des tests unitaires. Il s'agit d'une procédure permettant de vérifier le bon fonctionnement d'une partie précise du logiciel (appelée « unité » ou « module »). Reprenons l'exemple de la figure 3.3. Ce code décrit une fonction `square` dont le but est d'élever au carré l'entier qui est passé par paramètre. Le test unitaire le plus basique serait de vérifier que si l'on envoie le chiffre 3 dans cette fonction, cette dernière retourne le chiffre 9. L'objectif final est de "blinder" le code, c'est-à-dire vérifier qu'il se comporte comme le développeur l'a prévu. L'éciture de tests unitaires permet de trouver rapidement les erreurs au sein du code, de sécuriser la maintenance et finalement de documenter le code. Ils peuvent servir de complément à l'API, il est très utile de lire les tests pour comprendre comment s'utilise une méthode. De plus, il est possible que la documentation ne soit plus à jour, mais les tests eux correspondent à la réalité de l'application. C'est d'ailleurs ce que m'avait conseillé mon maître de stage durant ma montée en compétences sur le logiciel Nikita.

3.4 Migration du package `file`

Comme défini précédemment, la migration du package `file` était ma mission principale au cours de ce stage. Par "migration", nous entendons le passage d'une partie du code d'une certaine version à une autre version. En l'occurence, `file` était une action

4. https://github.com/adaltas/node-nikita-arch

contenue dans le package `core`. Ce dernier a disparu dans la nouvelle version de Nikita, il fallait donc déplacer toutes les actions le composant. Une grande partie du code avait déjà été migré avant mon arrivée chez Adaltas vers un nouveau package nommé `engine`. Ce dernier se différencie de l'ancienne version du logiciel de par la modification de l'implémentation du moteur de Nikita.

3.4.1 Architecture du package

La structure des fichiers du package `file` est décrite dans la figure 3.4.

```
env/
node_modules/
package.json
src/
  cache.coffee.md
  cson.coffee.md
  download.coffee.md
  index.coffee.md
  ini.coffee.md
  json.coffee.md
  properties/
    index.coffee.md
    read.coffee.md
  register.coffee
  render.coffee.md
  touch.coffee.md
  upload.coffee.md
  utils/
    curl.coffee
    diff.coffee.md
    index.coffee
  yaml.coffee.md
test/
test.coffee
```

FIGURE 3.4 – Structure de fichiers du package file

Le dossier `env` comporte des environnements de développement et de tests indépendants du système du développeur. Il s'agit d'un conteneur Docker qui étend le format de conteneur Linux standard, LXC, avec une API de haut niveau fournissant une solution pratique de virtualisation qui exécute les processus de façon isolée. Les tests unitaires évoqués précédemment sont exécutés au sein de ces conteneurs, cela afin d'être sûr que Nikita fonctionne sur tous les systèmes d'exploitations.

Les modules npm et leurs dépendances sont stockés dans un répertoire `node_modules`. Par exemple, l'action `index` nécessite une dépendance appelée `fs`. Le module `fs` permet d'interagir avec le système de fichiers d'une manière modelée sur les fonctions POSIX standard. Ce module est téléchargé puis décompressé dans le dossier `node_modules` afin que l'application puisse y accéder rapidement.

Tous les paquets npm contiennent un fichier, généralement à la racine du projet, appelé `package.json`. Ce fichier contient diverses métadonnées pertinentes pour le projet. Il est utilisé pour donner des informations à npm qui lui permettent d'identifier le projet ainsi que de gérer les dépendances du projet. Il peut également contenir d'autres métadonnées telles qu'une description du projet, la version du projet dans une distribution particulière, des informations sur la licence, voire des données de configuration - tout cela peut être vital à la fois pour npm et pour les utilisateurs finaux du paquet. Le fichier `package.json` est normalement situé dans le répertoire racine d'un projet Node.js.

Le répertoire `src` contient le code source des différentes actions contenues dans `file`. L'action `index` correspond à l'action principale du package. Elle permet de créer, modifier et supprimer un fichier. Les autres actions du package sont éponymes. Ci-dessous une liste des fonctionnalités :

`cache` Télécharge un fichier et le place dans un dossier local ou distant pour une utilisation ultérieure.

`cson` Ecrit le contenu d'un objet `cson` dans un fichier. L'objet en question peut être sujet à modifications.

`download` Télécharge un fichier en utilisant différents protocoles.

`index` Ecrit, modifie ou supprime un fichier ou une partie d'un fichier.

`ini` Écrit un objet sous forme de fichier `.ini`.

`json` Ecrit le contenu d'un objet `json` dans un fichier.

`properties.index` Ecrit un fichier au format des propriétés Java.

`properties.read` Lit un fichier au format des propriétés Java.

`render` Génère un fichier selon un modèle.

`touch` Crée un fichier vide s'il n'existe pas.

`upload` Upload un fichier vers un emplacement distant.

`yaml` Ecrit un objet sérialisé au format YAML.

Le dossier `utils` contient des fonctions utilitaires permettant par exemple de simplifier des objets JSON en supprimant les propriétés dont les valeurs sont `null` ou `undefined` ou encore d'afficher les différences entre deux textes.

Le répertoire `test` contient les tests unitaires associés aux actions du package `file`.

Enfin, le fichier `test.coffee` contient les intructions et configurations pour l'exécution des tests unitaires.

3.4.2 Workflow de la migration

La migration du package s'est faite par action. J'ai créé une nouvelle branche sur Git pour chaque module. Après avoir *refactor* le code source et les tests unitaires d'une action, j'ai push la branche sur GitHub et attendu l'examen de mon travail. L'avantage de cette méthodologie est que l'examinateur du code peut revenir sur chaque ligne de code et ajouter un commentaire. Après un certain nombre d'aller-retours, la branche est fusionnée avec la branche principale puis supprimée. Ce workflow est décrit dans la figure 3.5.

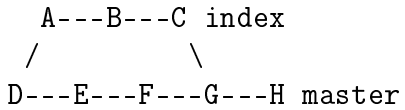

FIGURE 3.5 – Représentation des branches sur GitHub

Dans l'exemple décrit par la figure 3.5, nous avons créé une branche `index` à partir de la branche `master` sur laquelle nous développons l'action du même nom. Observons que des commits sont ajoutés à la branche `master` alors que nous travaillons sur une branche parallèle. Il est important de *rebase* la branche `index` avant de pouvoir la fusionner avec la branche `master` pour conserver les modifications apportées à cette dernière et les appliquer à notre nouvelle branche.

3.4.3 Migration d'une action du package

Une action est composée de plusieurs éléments :
- **hook** : vérifie la configuration passée à l'action et ajoute les valeurs par défaut.
- **schema** : décrit la configuration de l'action.
- **handler** : implémente la logique de l'action.
- **exports** : exporte les composants de l'action.
- **dependencies** : décrit les dépendances de l'action.

Prenons l'exemple d'une action simple telle que `touch`. Pour rappel, ce module permet de créer un fichier s'il n'existe pas.

Tout d'abord, traitons du **schema** de `touch`. Il est décrit dans la figure 3.6. Nous y retrouvons les différentes propriétés de l'action, le type associé ainsi qu'une description. Prenons l'exemple de la propriété `target`. Cette dernière doit être du type `string`, c'est-à-dire une chaîne de caractères, ou bien une fonction qui retourne une chaîne de caractères. `target` correspond au chemin local ou distant vers lequel le fichier sera écrit s'il n'existe pas à cet endroit. Les autres propriétés existent déjà dans d'autres actions Nikita. Afin d'éviter toute redondance, nous définissons un lien vers la définition de ces attributs.

```
schema =
  type: 'object'
  properties:
    'gid':
      $ref: 'module://@nikitajs/engine/src/actions/fs/base/chown#/properties/gid'
    'mode':
      $ref: 'module://@nikitajs/engine/src/actions/fs/base/chmod#/properties/mode'
    'target':
      oneOf: [{type: 'string'}, {typeof: 'function'}]
      description: """
      File path where to write file or a function that returns a valid file path.
      """
    'uid':
      $ref: 'module://@nikitajs/engine/src/actions/fs/base/chown#/properties/uid'
  required: ['target']
```

FIGURE 3.6 – Schema de l'action `touch`

Ensuite, étudions le **hook**. Pour cette action, il est extrêmement simple. Il consiste uniquement en la modification de la propriété `target` vue précédemment si et seulement si elle est déjà définie dans les metadonnées de l'action.

```
on_action = ({config, metadata}) ->
  config.target = metadata.argument if metadata.argument?
```

FIGURE 3.7 – Hook de l'action `touch`

A présent, passons à l'implémentation du **handler** décrit dans la figure 3.9. L'action prend en paramètre la configuration telle que définie dans le **schema** ainsi qu'une fonction `log` permettant d'afficher des messages lorsque le mode debug est actif. La logique de cette action est décrite dans la figure 3.8. Nous vérifions si le fichier passé dans la configuration existe. Si c'est le cas, alors nous modifions ce fichier grâce à l'action `index` en supprimant son contenu. Sinon, nous créons un fichier grâce à la commande `touch`[5] intégrée dans Linux.

5. touch(1) — Linux manual page

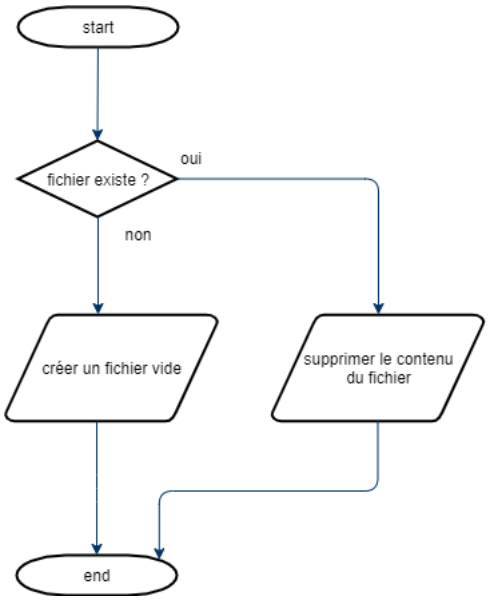

FIGURE 3.8 – Logigramme représentant la logique de l'action `touch`.

```
handler = ({config, log}) ->
  log message: "Entering file.touch", level: 'DEBUG', module: 'nikita/
   lib/file/touch'
  {status} = await @call ->
    log message: "Check if target exists \"#{config.target}\"", level:
    'DEBUG', module: 'nikita/lib/file/touch'
    exists = await @fs.base.exists target: config.target
    log message: "Destination does not exists", level: 'INFO', module:
    'nikita/lib/file/touch' if not exists
    !exists
  if status
    @file
      content: ''
      target: config.target
      mode: config.mode
      uid: config.uid
      gid: config.gid
  else
    @execute
      cmd: "touch #{config.target}"
      shy: true
  {}
```

FIGURE 3.9 – Handler de l'action `touch`

Enfin, l'action est composée d'une fonction qui exporte tous les éléments présentés

précédemment. Cette dernière permet au moteur de Nikita d'exécuter les méthodes dans le bon ordre.

```
module.exports =
  handler: handler
  hooks:
    on_action: on_action
  schema: schema
```

FIGURE 3.10 – Exports de l'action `touch`

Les éléments que nous venons d'étudier composent toutes les actions du package `file`. La logique de migration est toujours la même. La différence réside principalement dans l'implémentation de la logique du **handler**.

3.4.4 Migration des tests associés à une action

La migration des tests était simple car il ne fallait que réécrire les tests existant de façon légérement différente. Un exemple lié à `touch` est donné dans la figure 3.11.

```
nikita
  ssh: ssh
.file.touch "#{tmpdir}/file", (
   err, {status}) ->
  status.should.be.true()
   unless err
.file.assert
  target: "#{tmpdir}/file"
  content: ''
.promise()
```

```
nikita
  ssh: ssh
  tmpdir: true
, ({metadata: {tmpdir}}) ->
  @file.touch "#{tmpdir}/file"
  .should.be.resolvedWith
   status: true
  @fs.assert
    target: "#{tmpdir}/file"
    content: ''
```

FIGURE 3.11 – Comparaison d'un test de `touch` avant/après

Bilan et perspectives

Fort des éléments énoncés précédemment, ce stage technique effectué au sein de l'entreprise Adaltas constitue une expérience des plus enrichissantes étant donné la complexité technique des missions auxquelles j'ai pu prétendre. Outre l'aventure humaine que j'eus la chance et le privilège de vivre, ce stage m'apprit le sens de la rigueur, du professionnalisme, ainsi que l'importance du temps et de son agencement. Grâce à cette expérience, j'ai acquis des nouvelles compétences. De plus, les différentes missions effectuées m'ont permis d'accroître ma volonté de savoir et de connaissance, notamment dans le domaine du Big Data.

La réalisation des travaux décrits dans ce rapport m'a permis d'acquérir de nouvelles compétences en matière de développement et maintenance de code. En effet, le code tel qu'il existait précédemment était fonctionnel. Néanmoins, Adaltas a pris la décision de *refactor* le code. L'explication est la suivante : au cours de la vie d'un logiciel, des fonctionnalités sont ajoutées et des bugs sont corrigés. Ces modifications successives, n'améliorant pas en général la lisibilité du logiciel, ne facilitent pas, de ce fait, sa maintenance ultérieure. Le code source d'un programme a tout intérêt à rester, malgré ses modifications, le plus clair possible.

En terme de compétences techniques nouvellement acquises, nous pouvons énoncer :
- maîtrise du langage de programmation CoffeeScript et son environnement de développement ;
- utilisation de l'outil de versioning Git ;
- workflow développement logiciel ;
- technologie Node.js.

Ma contribution à ce projet à été chaleureusement accueilli par Adaltas. En effet, étant donné que Nikita est un logiciel Open Source, il génère peu voire pas de revenus. Mettre un consultant sur une telle mission n'est donc pas rentable pour l'entreprise. C'est pourquoi mon intervention a été la bienvenue. Aussi, je peux prétendre être une des rares personnes à bien maîtriser Nikita et son environnement de développement.

ECE PARIS • LYON
ÉCOLE D'INGÉNIEURS

Fiche d'évaluation "Entreprise" à compléter par le tuteur de stage

à insérer dans le rapport de stage

Fiche d'évaluation du stage
Cycle Ingénieur • 2ème année
2019/2020

Etudiant : Alexander Hoffmann — Majeure : SI

Entreprise : Adaltas

Tuteur de stage : David Worms

☎ du tuteur : +33 6 76 88 72 13 — Email du tuteur : david@adaltas.com

	Possibilité d'accorder des ½ points	**Note**
Technique		
• Difficulté de l'étude	4	/4
• Résultats obtenus / respect du CDC	6	/6
• Qualité du travail (méthodologie, réalisation, dossier …)	9	/10
Total Note technique		**/20**
Qualités humaines		
• Autonomie et sens des responsabilités	6	/7
• Esprit d'initiative et créativité	5	/5
• Intégration / sociabilité au sein de l'équipe	5	/5
• Comportement / Attitude	3	/3
Total note Qualités Humaines		**/20**
TOTAL	38	**/40**
Soit	19	/20

Observations :

A Boulogne Billancourt, le 20 aout 2020

Signature du Tuteur de stage
(obligatoire)

Cachet de l'entreprise d'accueil
(obligatoire)

ADALTAS
SAS au capital de 8000 €
6 rue Jules Simon
92100 Boulogne-Billancourt - France
RCS Nanterre B 452 561 913
Siret: 452 561 913 00032
TVA Intracommunautaire: FR31452561913