Université Paris Diderot

-

IRIF

Rapport de Stage
du 01/06/2019 au 01/08/2019

éffectué à Paris

Sébastien Lecleire
Etudiant en MASTER 1 IMPAIRS

supervisé par
M. Yann Régis-Gianas

Contents

1 Remerciements

Avant tout développement sur cette expérience professionnelle, il apparaît opportun de commencer ce rapport de stage par des remerciements, à ceux qui m'ont beaucoup appris au cours de ce stage, et même à ceux qui ont eu la gentillesse de faire de ce stage un moment très profitable.
Tout d'abord, j'adresse mes remerciements à ma professeure et amie, Mme Ines Klimann, maître de conférences à l'Université Paris Diderot (Paris 7) qui m'a beaucoup aidé dans ma recherche de stage.
Je tiens à remercier vivement mon maitre de stage, M. Yann Régis-Gianas, maître de conférences à l'Université Paris Diderot (Paris 7) et responsable de la platforme LearnOCaml au sein de l'IRIF pour son accueil, son enthousiasme et ses conseils avisés. Grâce à sa confiance j'ai pu accomplir mes différentes missions en totale autonomie avec rigueur et efficacité. Il fut d'une aide précieuse dans les moments les plus délicats.
Je remercie également l'ensemble des employés de l'IRIF et de l'Université Paris Diderot (Paris 7) pour leur accueil, leur gentillesse et leur professionnalisme. Enfin, je tiens à remercier toutes les personnes qui m'ont conseillé et aidé lors de mon stage.

2 Introduction

Du 01/06/2019 au 01/08/2019, j'ai effectué un stage au sein de l'Institut de Recherche en Informatique Fondamentale (IRIF[1]). Au cours de ce stage, j'ai pu m'intéresser au développement des méthodes formelles ou mathématiques pour modéliser des systèmes existants, naturels ou artificiels, et pour résoudre des problèmes concrets. Plus largement, ce stage a été l'opportunité d'appréhender l'aspect pratique des metiers liés au développement d'une application de sa conception à sa mise en production et des méthodes pour rendre plus sûre et plus efficace la distribution de systèmes logiciels de grande dimension.

Au-delà d'enrichir mes connaissances informatiques, ce stage m'a permis de comprendre énormement de chose sur le travail en équipe, sur la vie en entreprise et m'a donné de nombreuses pistes sur mon orientation après mon Master 2. Mon stage au pôle Preuves, Programmes et Systèmes (PPS[2]) de l'IRIF1 a consisté essentiellement en l'amélioration de la platforme LearnOCaml et au développement d'applications permettant de faciliter et de simplifier son utilisation. Mon maître de stage étant maître de conférences à l'Université Paris Diderot (Paris 7) et membre permanent du pôle PPS2, j'ai pu apprendre dans d'excellentes conditions à utiliser de nombreux outils mis à disposition par Github et à réaliser un FrontEnd, un BackEnd, une API[3], à administrer des serveurs Cloud Openstack et des Nodes Kubernetes aussi bien en local qu'avec OVH.

En vue de rendre compte de manière fidèle des deux mois passés au sein de l'IRIF1, il apparaît logique de présenter à titre préalable le cadre du stage : l'IRIF1, et l'environnement du stage, à savoir le secteur PPS2. Ensuite, il sera précisé les différentes missions et tâches que j'ai pu effectuer au sein du pôle PPS2. Enfin les nombreux apports que j'ai pu en tirer.

[1] Institut de Recherche en Informatique Fondamentale
[2] Preuves, Programmes et Systèmes
[3] Interface de Programmation d'Application ou Interface de Programmation Applicative

3 L'IRIF

L'Institut de Recherche en Informatique Fondamentale (IRIF) est une unité mixte de recherche (UMR 8243) entre le CNRS et l'université Paris Diderot, qui héberge deux équipes-projets INRIA. Il est issu de la fusion des deux UMR LIAFA et PPS au 1er janvier 2016. L'IRIF est aussi membre de la Fondation Sciences Mathématiques de Paris (FSMP).

3.1 Présentation

L'IRIF est reconnu pour ses contributions portant sur la conception et l'analyse d'algorithmes, l'étude des modèles de calculs et de représentation des données, les fondements des langages de programmation, le développement logiciel, la vérification et la certification.

Au CNRS, l'IRIF est principalement rattaché à l'Institut National des Sciences de l'Information et de leurs Interactions (INS2I). L'IRIF est membre de l'UFR d'informatique de l'université Paris Diderot, et accueille également en son sein plusieurs membres de l'UFR de mathématiques. Enfin, l'IRIF est associé à l'école doctorale des Sciences Mathématiques de Paris Centre (ED 386).

L'IRIF est structuré en neuf équipes thématiques regroupées en trois pôles de recherche :

* Pôle Algorithmes et structures discrètes
* Pôle Automates, structures et vérification
* Pôle Preuves, programmes et systèmes

L'IRIF compte actuellement une centaine de membres permanents, se répartissant environ en 48 enseignants-chercheurs, 27 chercheurs CNRS, 5 chercheurs INRIA, 8 membres émerites et 7 personnels administratifs ou techniques (en janvier 2019). L'effectif total de l'IRIF, incluant doctorants, postdoctorants, et visiteurs de longue durée s'élève à près de deux cents personnes.

Six membres de l'IRIF ont été lauréats de l'European Research Council (ERC), trois sont membres de l'Institut Universitaire de France (IUF) et deux sont membres de l'Academia Europæa.

3.2 PPS

Thèmes de recherche

Le programme scientifique du pôle Preuves, programmes et systèmes (PPS) de l'IRIF vise à renforcer les fondements théoriques des langages de programmation, des assistants de preuves et, plus généralement, des formalismes de calcul. Ces problématiques sont abordées en croisant trois points de vue complémentaires :

* une approche syntaxique, qui développe des langages théoriques issus de formalismes logiques
* une approche algébrique, qui étudie les structures mathématiques liées au calcul
* une approche pratique, qui modélise et analyse des systèmes de calcul réels

Le pôle est constitué de trois équipes thématiques, correspondant à chacun de ces trois points de vue. Elles développent leurs outils propres, et les mettent ensuite au service d'objectifs scientifiques communs :
Le pôle PPS héberge l'équipe-projet πr^2 commune à l'INRIA, au CNRS et à l'Université Paris-Diderot — Paris 7, ainsi qu'une partie des membres de l'IRILL (Initiative de Recherche et d'Innovation sur le Logiciel Libre), une structure commune à l'INRIA, à l'Université Paris-Diderot — Paris 7 et à l'Université Pierre-et-Marie-Curie — Paris 6.

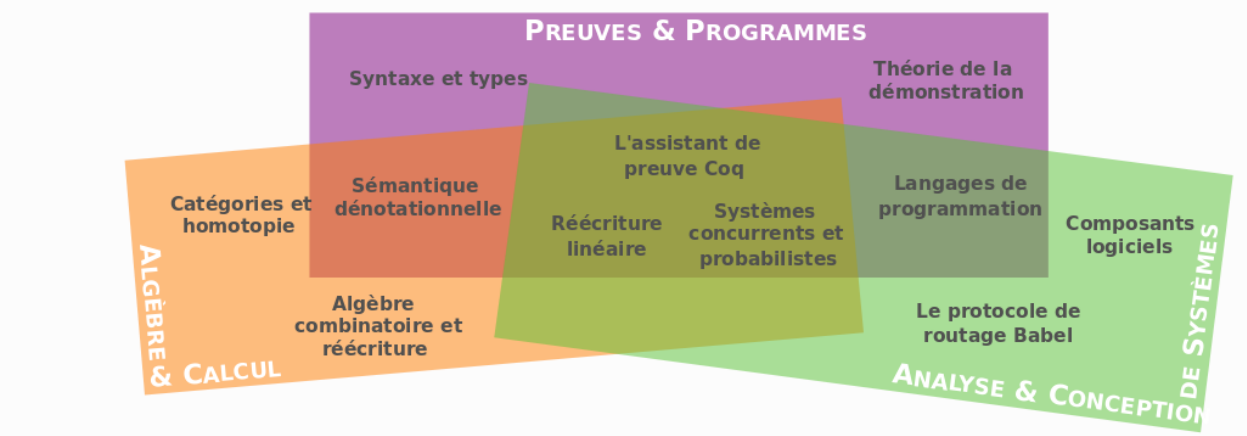


Figure 1: Diagramme des équipes PPS.

4 Mes missions

Au cours de ce stage, j'ai eu l'opportunité de découvrir un ensemble de métiers sous de nombreuses formes et de comprendre de manière globale les difficultés que les informaticiens pouvaient rencontrer. Pour une meilleure compréhension des tâches que j'ai pu effectuer, il apparaît approprié de traiter en premier lieu des outils qui étaient mis à ma disposition, puis de traiter de manière détaillée les tâches que j'ai pu effectuer.

4.1 Les outils

Au cours de ce stage, j'ai passé le plus clair de mon temps à réfléchir et à coder. A mesure que j'apprenais, mes recherches se sont approfondies. Ce n'est donc qu'à partir de la 2e semaine de mon stage que j'ai été véritablement opérationnel, du fait de ma meilleure maîtrise des ressources mises à ma disposition.

4.1.1 Le matériel informatique

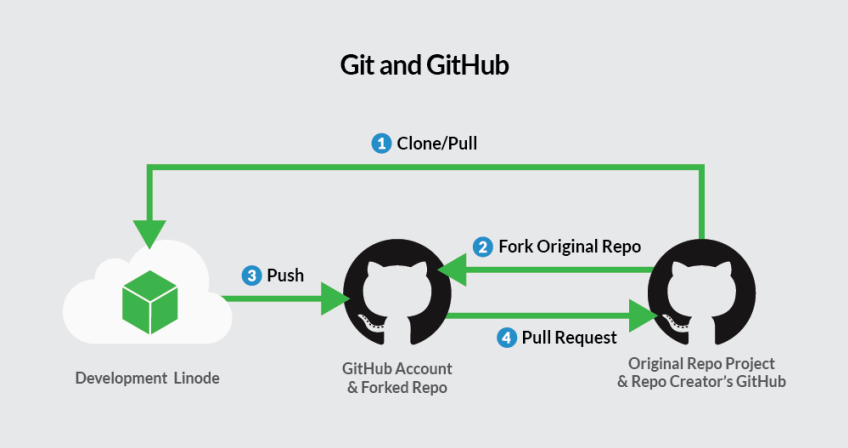
A été mis à ma disposition un Dell OptiPlex 7450 All-in-One doté d'un processeur Intel Core i5-7600, d'un SSD de 256GB, de 16GB de RAM et d'un écran 4K. Un vrai bijou de technologie suffisamment puissant pour faire tourner tous les serveurs et toutes les machines virtuelles dont j'avais besoin. Le tout relié à la fibre de l'université Paris Diderot (Paris 7), j'avais là de quoi travailler sans craindre le moindre problème de performance.

4.1.2 La distribution

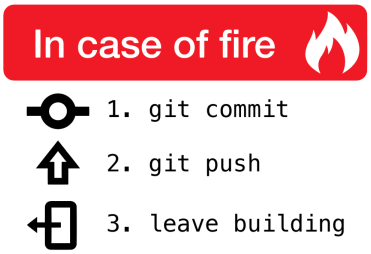
Rien de tel qu'une distribution Linux pour travailler ! Et Parrot OS, distribution basée sur Debian mettant l'accent sur la sécurité fut mon choix. Open source et basé sur un OS de la famille POSIX, Parrot OS est une distribution complète et très récente (lancée en 2013) alliant sécurité, modernité et efficacité.

4.1.3 Github

GitHub (exploité sous le nom de GitHub, Inc.) est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. Ce site est développé en Ruby on Rails et Erlang par Chris Wanstrath, PJ Hyett et Tom Preston-Werner. GitHub propose des comptes professionnels payants, ainsi que des comptes gratuits pour les projets de logiciels libres. Le site assure également un contrôle d'accès et des fonctionnalités destinées à la collaboration comme le suivi des bugs, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet.
Github fonctionne avec le logiciel de contrôle de version Git ce qui signifie qu'il gère les modifications d'un projet sans écraser n'importe quelle partie du projet.



(a) Fonctionnement en temps normal



(b) Fonctionnement en cas de crise majeure

4.1.4 Visual Studio Code

Visual Studio Code est un éditeur de code extensible, open source et gratuit, développé par Microsoft pour Windows, Linux et macOS. Doté de nombreux plugins, il est parfaitement approprié au développement dans plusieurs langages.

4.1.5 NodeJS

Node.js est une plateforme logicielle libre et événementielle en JavaScript orientée vers les applications réseau qui doivent pouvoir monter en charge. Elle utilise la machine virtuelle V8 et implémente sous licence MIT les spécifications CommonJS. Parmi les modules natifs de Node.js, on retrouve http qui permet le développement de serveur HTTP. Il est donc possible de se passer de serveurs web tels que Nginx ou Apache lors du déploiement de sites et d'applications web développés avec Node.js. Concrètement, Node.js est un environnement bas niveau permettant l'exécution de JavaScript côté serveur. Node.js est utilisé notamment comme plateforme de serveur Web, elle est utilisée par Groupon, Vivaldi, SAP, LinkedIn, Microsoft, Yahoo!, Walmart, Rakuten, Sage et PayPal.

4.1.6 npm

npm est le gestionnaire de paquets officiel de Node.js. npm fonctionne avec un terminal et gère les dépendances pour une application. Il permet également d'installer des applications Node.js disponibles sur le dépôt npm.

4.1.7 Yarn

Yarn est un gestionnaire de dépendances rapide, fiable et sécurisé. C'est un paquet (librairie javascript open source) pour Node.js, un environnement d'éxecution JavaScript permettant d'utiliser ce dernier pour créer des applications webs plus puissantes et maléables notamment grâce à son écosysytème de paquets qui permet de réutiliser du code fait par d'autres développeurs. Yarn est nous vient des équipes d'ingénieurs de Facebook avec la participation de Google, Exponent et Tilde. C'est une alternative très intéressante à npm qui se veut plus rapide et plus sécurisée ! L'utilisation de Yarn et de npm simultanément résulte d'une découverte tardive de Yarn et d'un besoin de npm pour executer mon Backend en mode développement.

4.1.8 Express

Express est une infrastructure d'applications Web Node.js minimaliste et flexible qui fournit un ensemble de fonctionnalités robuste pour les applications Web et mobiles. Grâce à une foule de méthodes utilitaires HTTP et de middleware, la création d'une API robuste est simple et rapide.

4.1.9 Bootstrap

Bootstrap est une collection d'outils utiles à la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.) de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option. C'est l'un des projets les plus populaires sur la plate-forme de gestion de développement GitHub. Mon choix s'est porté sur la librairie BootStrap pour gérer le style car elle m'est familière, populaire et bien documentée.

4.1.10 Angular

Angular (communément appelé "Angular 2+" ou "Angular v2 et plus") est un framework côté client open source basé sur TypeScript dirigée par l'équipe du projet Angular à Google et par une communauté de particuliers et de sociétés. Angular est une réécriture complète de AngularJS, cadriciel construit par la même équipe. Mon choix s'est porté sur Angular car je maîtrise bien ses différents aspects, sa logique me parle, il est bien documenté et on trouve énormément d'exemples bien expliqués sur Internet.

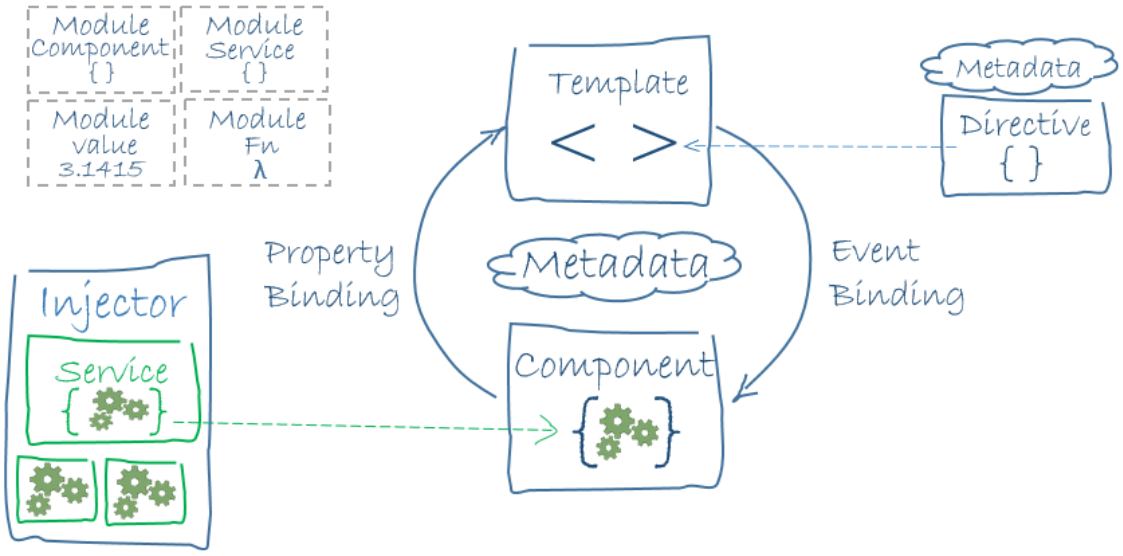
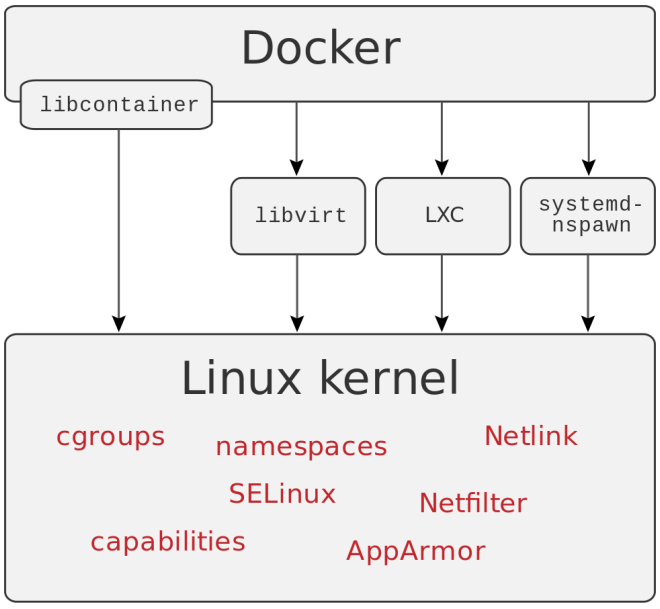


Figure 12: L'Architecture de l'application. Les principaux blocs de construction sont des modules, des composants, des modèles, des métadonnées, la liaison de données, des directives, des services et de l'injection de dépendance.

4.1.11 Docker

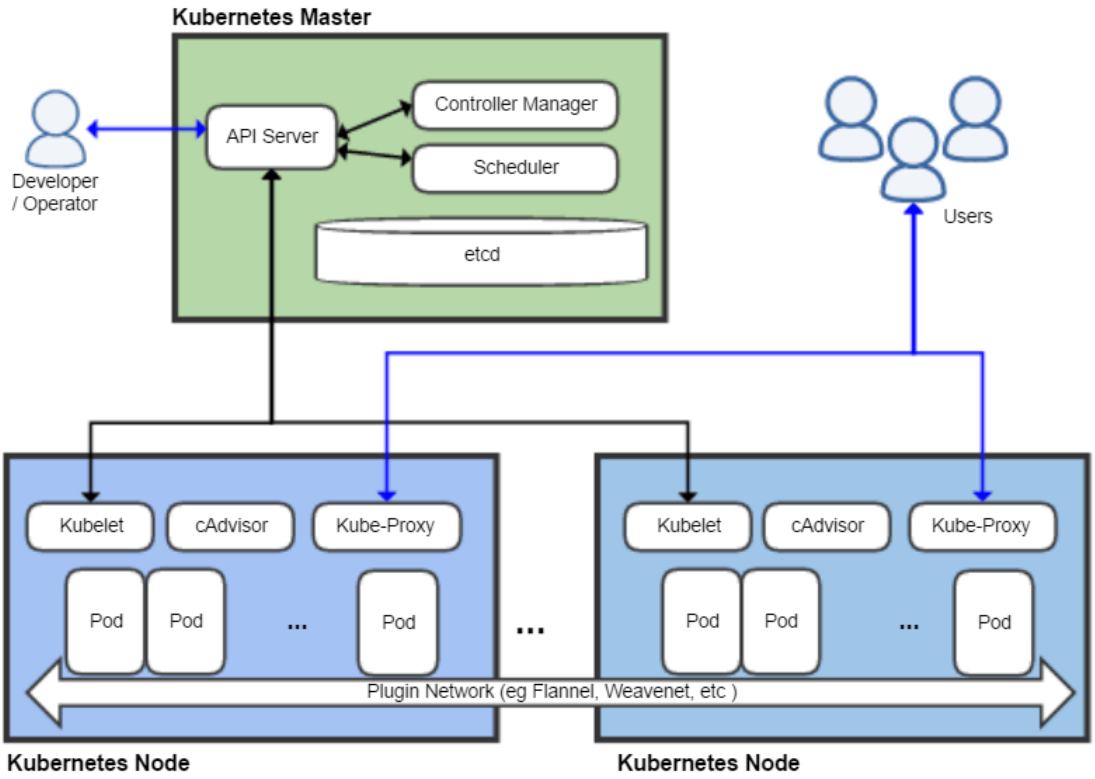
Docker est un logiciel libre permettant facilement de lancer des applications dans des conteneurs logiciels. Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur. Il ne s'agit pas de virtualisation, mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement. Cette approche permet d'accroître la flexibilité et la portabilité d'exécution d'une application, laquelle va pouvoir tourner de façon fiable et prédictible sur une grande variété de machines hôtes, que ce soit sur la machine locale, un cloud privé ou public, une machine nue, etc..
Techniquement, Docker 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. Pour arriver à ses fins, Docker utilise entre autres LXC, cgroups et le noyau Linux lui-même. Contrairement aux machines virtuelles traditionnelles, un conteneur Docker n'inclut pas de système d'exploitation, mais s'appuie au contraire sur les fonctionnalités du système d'exploitation fournies par la machine hôte.



(a) Schéma des interfaces de Docker

4.1.12 Kubernetes

Kubernetes (communément appelé "K8s") est un système open source qui vise à fournir une plate-forme permettant d'automatiser le déploiement, la montée en charge et la mise en œuvre de conteneurs d'application sur des clusters de serveurs. Il fonctionne avec toute une série de technologies de conteneurisation, et est souvent utilisé avec Docker. Il a été conçu à l'origine par Google, puis offert à la Cloud Native Computing Foundation.



(a) Plan de contrôle Kubernetes

Le maître Kubernetes est l'unité de contrôle principale qui gère la charge de travail et dirige les communications dans le système. Le plan de contrôle de Kubernetes consiste en plusieurs composants, chacun ayant son propre processus, qui peuvent s'exécuter sur un seul node maître ou sur plusieurs maîtres permettant de créer des clusters haute disponibilité. Les différents composants du plan de contrôle de Kubernetes sont décrits à la page suivante.

Les différents composants du plan de contrôle de Kubernetes :

etcd

etcd est une unité de stockage distribuée persistante et légère de données clé-valeur développée par CoreOS, qui permet de stocker de manière fiable les données de configuration du cluster, représentant l'état du cluster à n'importe quel instant. D'autres composants scrutent les changements dans ce stockage pour aller eux-mêmes vers l'état désiré.

serveur d'API

Le serveur d'API est un élément clé et sert l'API Kubernetes grâce à JSON via HTTP. Il fournit l'interface interne et externe de Kubernetes. Le serveur d'API gère et valide des requêtes REST et met à jour l'état des objets de l'API dans etcd, permettant ainsi aux clients de configurer la charge de travail et les containers sur les nœuds de travail.

L'ordonnanceur

L'ordonnanceur est un composant additionnel permettant de sélectionner quel noeud devrait faire tourner un pod non ordonnancé[4] en se basant sur la disponibilité des ressources. L'ordonnanceur gère l'utilisation des ressources sur chaque noeud afin de s'assurer que la charge de travail n'est pas en excès par rapport aux ressources disponibles. Pour accomplir cet objectif, l'ordonnanceur doit connaître les ressources disponibles et celles actuellement affectées sur les serveurs.

Controller manager

Le gestionnaire de contrôle (controller manager) est le processus dans lequel s'exécutent les contrôleurs principaux de Kubernetes tels que DaemonSet Controller et le Replication Controller. Les contrôleurs communiquent avec le serveur d'API pour créer, mettre à jour et effacer les ressources qu'ils gèrent (pods, service endpoints, etc.).

Node Kubernetes

Le Node aussi appelé Worker ou Minion est une machine unique (ou une machine virtuelle) où des conteneurs (charges de travail) sont déployés. Chaque node du cluster doit exécuter le programme de conteneurisation (par exemple Docker), ainsi que les composants mentionnés ci-dessous, pour communiquer avec le maître afin de configurer la partie réseau de ces conteneurs.

Kubelet

Kubelet est responsable de l'état d'exécution de chaque nœud (c'est-à-dire, d'assurer que tous les conteneurs sur un nœud sont en bonne santé). Il prend en charge le démarrage, l'arrêt, et la maintenance des conteneurs d'applications (organisés en pods) dirigé par le plan de contrôle.

Kubelet surveille l'état d'un pod et s'il n'est pas dans l'état voulu, le pod sera redéployé sur le même node. Le statut du node est relayé à intervalle de quelques secondes via messages d'état vers le maître. Dès que le maître détecte un défaut sur un node, le Replication Controller voit ce changement d'état et lance les pods sur d'autres hôtes en bonne santé.

Kube-proxy

[4] pod n'appartenant pas encore à un noeud

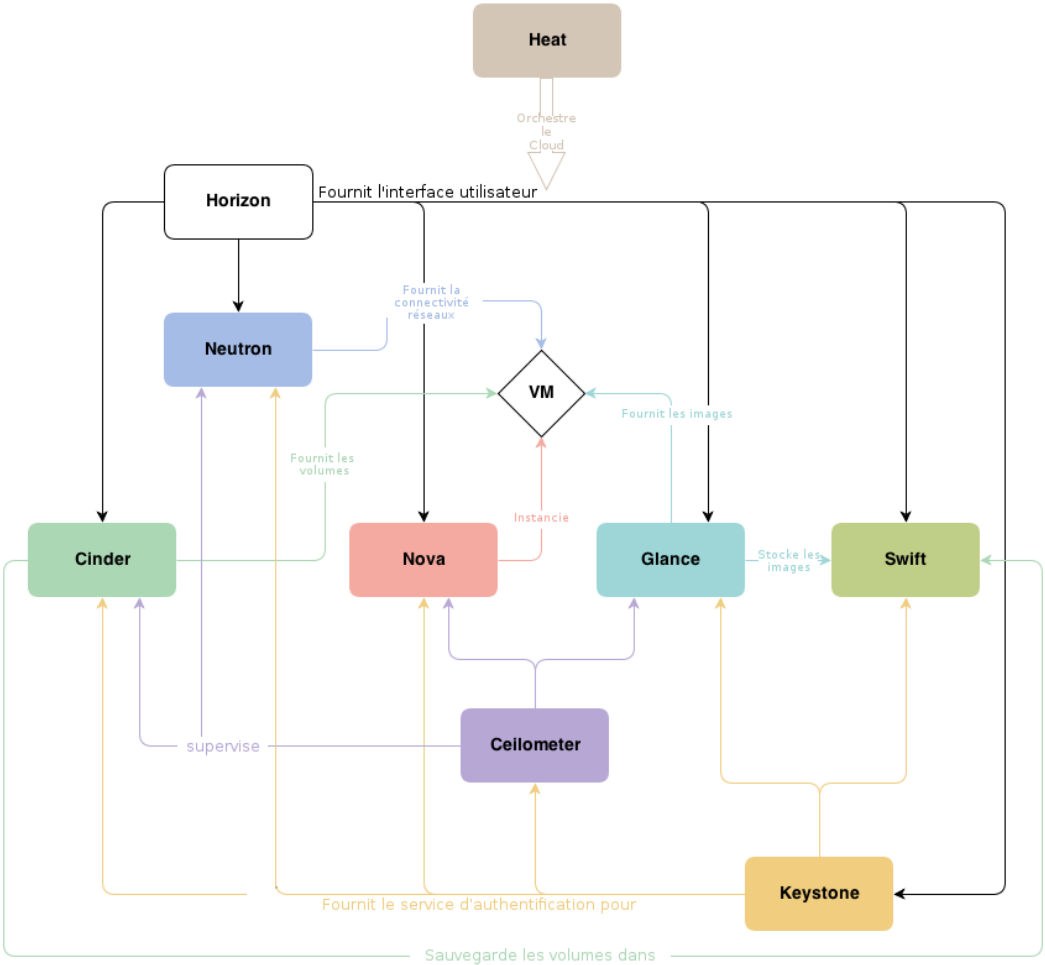
Le kube-proxy est l'implémentation d'un proxy réseau et d'un répartiteur de charge, il gère le service d'abstraction ainsi que d'autres opérations réseaux. Il est responsable d'effectuer le routage du trafic vers le conteneur approprié en se basant sur l'adresse IP et le numéro de port de la requête entrante.

cAdvisor

cAdvisor est un agent qui surveille et récupère les données de consommation des ressources et des performances comme le processeur, la mémoire, ainsi que l'utilisation disque et réseau des conteneurs de chaque node.

4.1.13 Openstack

OpenStack est un ensemble de logiciels open source permettant de déployer des infrastructures de cloud computing (infrastructure en tant que service). La technologie possède une architecture modulaire composée de plusieurs projets corrélés (Nova, Swift, Glance...) qui permettent de contrôler les différentes ressources des machines virtuelles telles que la puissance de calcul, le stockage ou encore le réseau inhérents au centre de données sollicité.
Le projet est porté par la Fondation OpenStack, une organisation non-commerciale qui a pour but de promouvoir le projet OpenStack ainsi que de protéger et d'aider les développeurs et toute la communauté OpenStack.
De nombreuses entreprises ont rejoint la fondation OpenStack. Parmi celles-ci on retrouve : Canonical, Red Hat, SUSE, eNovance, AT&T, Cisco, Dell, IBM, Yahoo!, Oracle, Orange, Cloudwatt, EMC, VMware, Intel, OVH, NetApp.



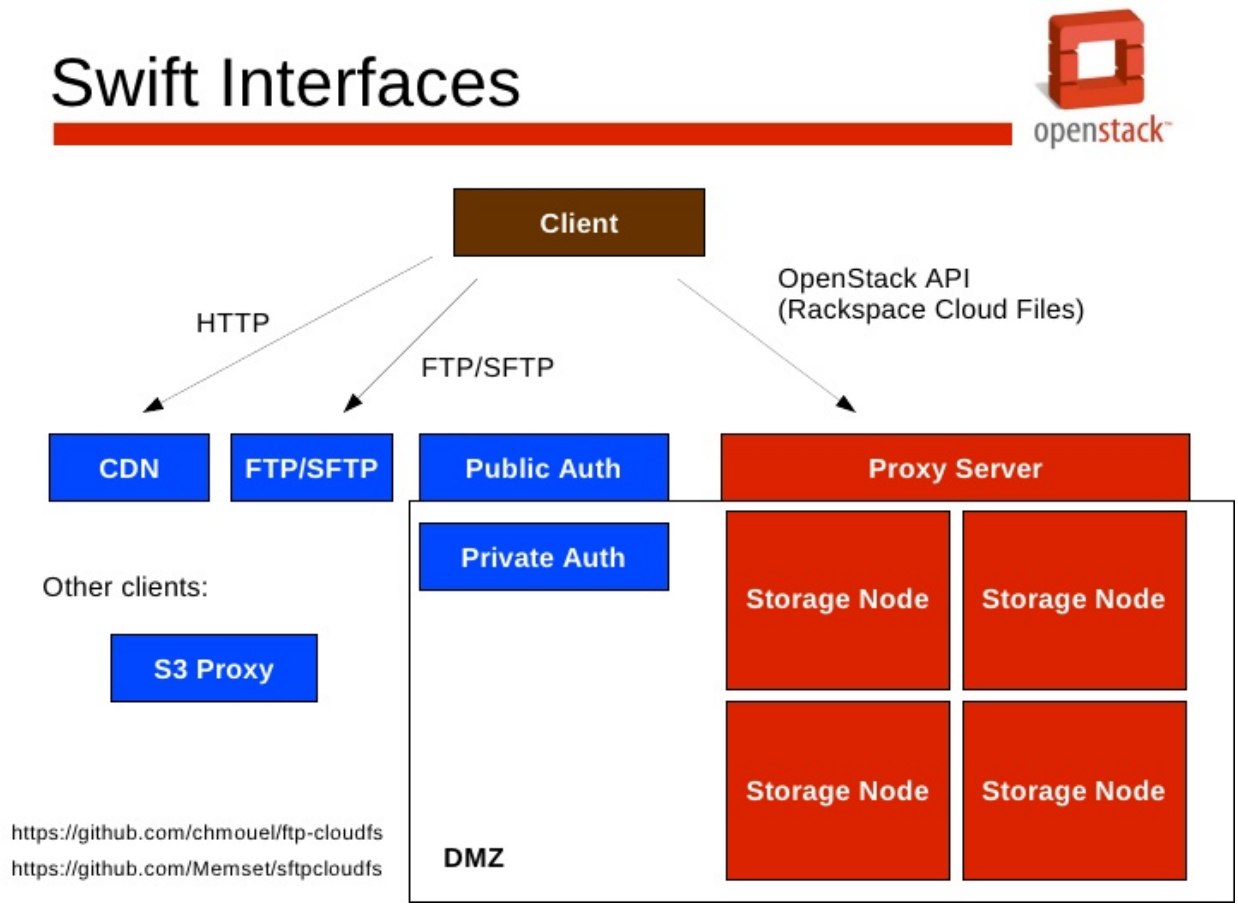
(a) Architecture conceptuelle des services OpenStack

OpenStack possède une architecture modulaire qui comprend de nombreux composants, nous ne nous intéresserons qu'à deux d'entre eux : *Swift* et *Cinder*

Stockage objet : Swift

Le stockage objet d'OpenStack s'appelle Swift. C'est un système de stockage de données redondant et évolutif[5]. Les fichiers sont écrits sur de multiples disques durs répartis sur plusieurs serveurs dans un Datacenter. Il s'assure de la réplication et de l'intégrité des données au sein du cluster. Le cluster Swift évolue horizontalement en rajoutant simplement de nouveaux serveurs. Si un serveur ou un disque dur tombe en panne, Swift réplique son contenu depuis des nœuds actifs du cluster dans des emplacements nouveaux. En août 2009, c'est Rackspace qui a commencé le développement de Swift, en remplacement de leur ancien produit nommé Cloud Files. Aujourd'hui c'est la société SwiftStack qui mène le développement de Swift avec la communauté.

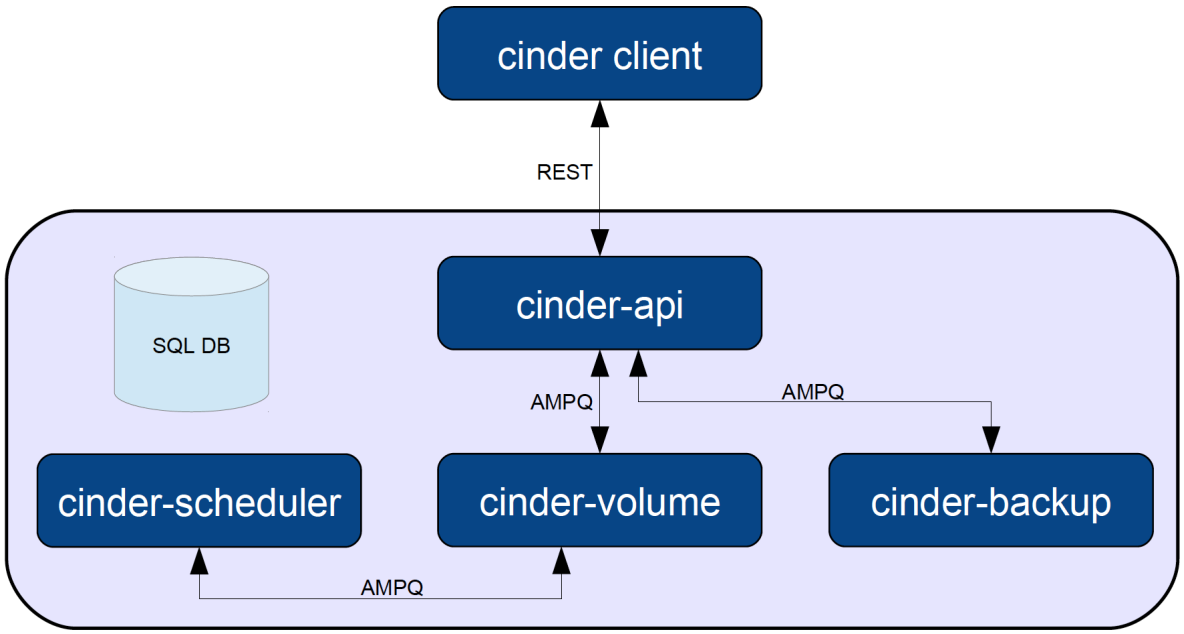
[5]Des noeuds et disques s'ajoutent à votre cluster en quelques minutes, sans aucune configuration. La capacité et les performances évoluent de manière linéaire, ce qui en fait une parfaite alternative aux dispositifs de stockage existants n'étant tout simplement plus conçus pour gérer l'ampleur considérable des données actuelles



(a) Architecture conceptuelle des services Swift

Stockage bloc : Cinder

Le service de stockage en mode bloc d'OpenStack s'appelle Cinder. Il fournit des périphériques persistants de type bloc aux instances OpenStack. Il gère les opérations de création, d'attachement et de détachement de ces périphériques sur les serveurs. En plus du stockage local sur le serveur, Cinder peut utiliser de multiples plateformes de stockage tel que Ceph, EMC (ScaleIO, VMAX et VNX), GlusterFS, Hitachi Data Systems, IBM Storage (Storwize family, SAN Volume Controller, XIV Storage System, et GPFS), NetApp, HP (StoreVirtual et 3PAR) et bien d'autres.
Le stockage en mode bloc est utilisé pour des scénarios performant comme celui du stockage de base de données, mais aussi pour fournir au serveur un accès bas niveau au périphérique de stockage. Cinder gère aussi la création d'instantanés (snapshots), très utile pour sauvegarder des données contenues dans les périphériques de type bloc. Les instantanés peuvent être restaurés ou utilisés pour créer de nouveaux volumes.



(a) Architecture conceptuelle des services Cinder

4.1.14 MongoDB

MongoDB est un système de gestion de base de données orienté documents, répartissable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données. Il est écrit en C++. Le serveur et les outils sont distribués sous licence SSPL, les pilotes sous licence Apache et la documentation sous licence Creative Commons. Il fait partie de la mouvance NoSQL.
Il est depuis devenu un des SGBD les plus utilisés, notamment pour les sites web de Craigslist, eBay, Foursquare, SourceForge.net, Viacom, pagesjaunes et le New York Times.

4.1.15 Thinkster

Thinkster est une société créée en 2013 fournissant des cours et des modèles pour construire des applications complètes à partir de rien en utilsiant les derniers frameworks disponibles.
Sur leur github : https://github.com/gothinkster/realworld , on peut trouver des modèles fonctionnels de FrontEnd et de BackEnd en de nombreux langages. Leurs cours permettent aussi de comprendre en détail les spécifications de leurs solutions et la communauté assez active sur Github met régulièrement leurs solutions au goût du jour.

4.1.16 LearnOCaml

Learn-OCaml est une plate-forme d'apprentissage du langage OCaml, proposant un toplevel directement sur le web, un environnement d'exercices, un répertoire de leçons et de didacticiels. Learn-OCaml a été écrit par OCamlPro et est sous licence MIT.

4.1.17 ReadTheDocs

ReadTheDocs est une énorme ressource sur laquelle des millions de développeurs s'appuient pour la documentation logicielle.
ReadTheDocs est pris en charge par la communauté. Ce sont les utilisateurs qui contribue au développement, au support et aux opérations. ReadTheDocs permet de rédiger une documentation à côté de votre code, avec tous les outils nécessaires. La documentation doit être écrite avec reStructuredText ou en Markdown.
À l'aide de webhooks, la documentation est automatiquement mise à jour, quelle que soit la version du logiciel.
La documentation publiée est hébergée de manière sécurisée et n'est disponible que pour les personnes internes d'une entreprise. L'hébergement du projet est gracieusement fourni par Microsoft Azure.

4.1.18 JsofOCaml

JsofOCaml est un compilateur de programmes de bytecode OCaml vers JavaScript.
JsofOCaml est fournit par Ocsigen, un outil de développement web et mobile, développé par le laboratoire français IRIF et par la société Be Sport SAS, utilisant des solutions nouvelles issues de la recherche sur les langages de programmation.
Il permet d'exécuter des programmes OCaml purs dans un environnement JavaScript tel que les navigateurs et Node.js. Il est facile à installer car il fonctionne avec une installation existante d'OCaml, sans qu'il soit nécessaire de recompiler une bibliothèque. Il est livré avec des liaisons pour une grande partie des API de navigateur. Selon les tests, les programmes générés s'exécutent généralement plus rapidement qu'avec l'interpréteur de bytecodes d'OCaml. Ce compilateur s'avère beaucoup plus facile à maintenir qu'un compilateur OCaml reciblé, car le bytecode fournit une API très stable.

4.1.19 Langages utilisés

Pour utiliser ces nombreux outils, une variété de langage était nécessaire, choisit en fonction du style de programmation et leur pertinence.

TypeScript

TypeScript est un langage de programmation libre et open source développé par Microsoft qui a pour but d'améliorer et de sécuriser la production de code JavaScript. C'est un sur-ensemble de JavaScript (c'est-à-dire que tout code JavaScript correct peut être utilisé avec TypeScript). Le code TypeScript est transcompilé en JavaScript, et peut ainsi être interprété par n'importe quel navigateur web ou moteur JavaScript. TypeScript a été cocréé par Anders Hejlsberg, principal inventeur de C#.

JavaScript

JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs avec l'utilisation (par exemple) de Node.js. C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont pas des instances de classes, mais qui sont chacun équipés de constructeurs permettant de créer leurs propriétés, et notamment une propriété de prototypage qui permet d'en créer des objets héritiers personnalisés. En outre, les fonctions sont des objets de première classe. Le langage supporte le paradigme objet, impératif et fonctionnel. JavaScript est le langage possédant le plus large écosystème grâce à son gestionnaire de dépendances npm, avec environ 500 000 paquets en août 2017.

OCaml

OCaml, anciennement connu sous le nom d'Objective Caml, est l'implémentation la plus avancée du langage de programmation Caml, créé par Xavier Leroy, Jérôme Vouillon, Damien Doligez, Didier Rémy et leurs collaborateurs en 1996. Ce langage, de la famille des langages ML, est un projet open source dirigé et maintenu essentiellement par l'Inria.

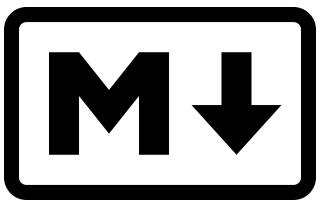
Html

L'HyperText Markup Language, généralement abrégé HTML, est le langage de balisage conçu pour représenter les pages web. C'est un langage permettant d'écrire de l'hypertexte, d'où son nom. HTML permet également de structurer sémantiquement et logiquement et de mettre en forme le contenu des pages, d'inclure des ressources multimédias dont des images, des formulaires de saisie et des programmes informatiques. Il permet de créer des documents interopérables avec des équipements très variés de manière conforme aux exigences de l'accessibilité du web. Il est souvent utilisé conjointement avec le langage de programmation JavaScript et des feuilles de style en cascade (CSS). HTML est inspiré du Standard Generalized Markup Language (SGML). Il s'agit d'un format ouvert.

Css

Les feuilles de style en cascade, généralement appelées CSS de l'anglais Cascading Style Sheets, forment un langage informatique qui décrit la présentation des documents HTML et XML. Les standards définissant CSS sont publiés par le World Wide Web Consortium (W3C). Introduit au milieu des années 1990, CSS devient couramment utilisé dans la conception de sites web et bien pris en charge par les navigateurs web dans les années 2000.

Markdown



Markdown est un langage de balisage léger créé en 2004 par John Gruber avec Aaron Swartz. Son but est d'offrir une syntaxe facile à lire et à écrire. Un document balisé par Markdown peut être lu en l'état sans donner l'impression d'avoir été balisé ou formaté par des instructions particulières.
Un document balisé par Markdown peut être converti en HTML, en PDF ou en autres formats. Bien que la syntaxe Markdown ait été influencée par plusieurs filtres de conversion de texte existants vers HTML dont Setext, atx, Textile, reStructuredText, Grutatext et EtText, la source d'inspiration principale est le format du courrier électronique en mode texte.

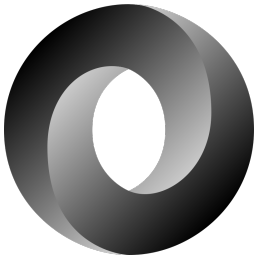
LaTeX

LaTeX est un langage et un système de composition de documents créé par Leslie Lamport en 1983. Il s'agit d'une collection de macro-commandes destinées à faciliter l'utilisation du processeur de texte TeX de Donald Knuth. Depuis 1993, il est maintenu par le LaTeX3 Project team. Le nom est l'abréviation de Lamport TeX. L'utilisation de LaTeX pour les formules mathématiques est très répandue.
Du fait de sa relative simplicité, il est devenu le langage privilégié pour les documents scientifiques employant TeX. Il est particulièrement utilisé dans les domaines techniques et scientifiques pour la production de documents de taille moyenne ou importante (thèse ou livre, par exemple). Néanmoins, il peut être aussi employé pour générer des documents de types variés (par exemple, des lettres, des transparents ou encore au hasard des rapports de stage...).

Bash

Bash (acronyme de Bourne-Again shell) est un interpréteur en ligne de commande de type script. C'est le shell Unix du projet GNU. Fondé sur le Bourne shell, Bash lui apporte de nombreuses améliorations, provenant notamment du Korn shell et du C shell. Bash est un logiciel libre publié sous licence publique générale GNU. Il est l'interprète par défaut sur de nombreux Unix libres, notamment sur les systèmes GNU/Linux.

JSON



JavaScript Object Notation (JSON) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple. Créé par Douglas Crockford entre 2002 et 2005, il est actuellement décrit par deux normes en concurrence : RFC 8259 de l'IETF et ECMA-404 de l'ECMA.

4.2 Les missions

Différentes missions m'ont été confiées durant le stage toute avec le même but : améliorer le fonctionnement de LearnOCaml. Mes missions s'organisent de deux manières bien distinctes : ma mission principale et mes missions secondaires. Il serait judicieux avant de rentrer dans les détails de mes missions de vous expliquer le fonctionnement de LearnOCaml et de Learn-OCaml-Essok.

4.2.1 LearnOCaml

LearnOCaml ou Learn-OCaml est une plate-forme d'apprentissage du langage OCaml. LearnOCaml est à destination des enseignants du monde entier. La platforme est développée en anglais. L'idée est que chaque enseignant possède son instance de LearnOCaml qu'il pourra personnaliser en fonction de ses besoins.
LearnOCaml fonctionne avec des instances, i.e. chaque personne voulant utiliser ce logiciel doit avoir son propre serveur. Chaque serveur est indépendant des autres sur son contenu, seule sa structure de base - son squelette - est identique.
Un enseignant peut écrire des exercices en OCaml à destination de ses étudiants et les ajouter à son instance de LearnOCaml. Les étudiants pourront alors y accéder et tenter de les résoudre, ce qui est complètement automatique, pour obtenir une note sur l'exercice avec des conseils pour s'améliorer.
Un enseignant peut aussi récupérer des exercices déjà tout faits sur des plateformes mise à disposition sur Internet pour économiser du temps.
Les forces de ce projet sont équitablement réparties : chaque enseignant pourra obtenir une note pertinente sur le travail de ses étudiants tout le long d'une année scolaire sans avoir à corriger un nombre incalculable de copie et chaque étudiant pourra rapidement voir sa progression dans l'apprentissage du langage OCaml avec un outil moderne et adapté à ses besoins.

4.2.2 Learn-OCaml-Essok

Learn-OCaml-Essok ou LearnOCamlEssok est un projet qui à pour objectif de gérer facilement et automatiquement toutes les instances de LearnOCaml à travers le monde.
Pour se faire, le projet se découpe en plusieurs parties :

- le FrontEnd qui est une ihm (interface homme machine, i.e. un site web), sur lequel les enseignants pourront "gérer" leurs instances de LearnOCaml.

- le BackEnd qui regroupe une API pour dialoguer avec le FrontEnd, des clients pour les différents services utilisés et une base de donnée.

Ici le BackEnd qualifie tous les services invisibles à l'utilisateur qui permettent le bon fonctionnement du FrontEnd.

4.2.3 Objectif principal : développer LearnOCamlEssok

Présentation

Ma mission principale était, à l'origine, de réaliser le FrontEnd pour Learn-OCaml-Essok. J'ai opté pour TypeScript comme langage de programmation et Angular 8 comme framework pour le dévélopper.

Rapidement, je me suis rendu compte que j'avais besoin du BackEnd pour réaliser des tests. J'ai alors commencé le développement du BackEnd. Le problème était mon manque de connaissances sur le sujet, je n'avais aucune idée de comment faire une application complète tout seul et à partir de rien. Pour palier ce problème, je me suis dirigé vers des solutions basiques et libres toutes prêtes disponibles sur github.

Thinkster a été d'un grand secours car il proposait exactement ce dont j'avais besoin : des FrontEnd et des BackEnd communiquant entre eux de manière fonctionnelle et sécurisée, le code étant sous licence MIT [6].
Thinkster propose des solutions dans un panel de langages différents et par continuité vis à vis de mon travail déjà effectué j'ai choisi un FrontEnd en Angular et un Backend en JavaScript.

Bien évidemment, les solutions proposées par Thinkster ne répondent pas aux spécifications de Learn-OCaml-Essok et sont justes mises à disposition pour créer une base communiquante entre un FrontEnd et un BackEnd. Heureusement c'est tout ce qu'il me fallait pour démarrer le développement en bonne et due forme de LearnOCamlEssok.
Mon travail à donc consisté à développer le FrontEnd et le BackEnd sur cette base déjà existante en supprimant les fonctionnalités inutiles au projet, en rajoutant celles qui n'existaient pas encore et en modifiant celles qui pouvaient servir.

Par la suite un serveur correspondra à une instance de LearnOCaml. Un enseignant, donc ici un utilisateur, peut posséder plusieurs instances de LearnOCaml, par exemple une pour chacune de ses classes, le nombre maximal d'instance se fera au cas par cas.

[6] Massachusetts Institute of Technology, la licence donne à toute personne recevant le logiciel (et ses fichiers) le droit illimité de l'utiliser, le copier, le modifier, le fusionner, le publier, le distribuer, le vendre et le "sous-licencier" (l'incorporer dans une autre licence). La seule obligation est d'incorporer la notice de licence et de copyright dans toutes les copies.

4.2.4 Le FrontEnd

Présentation

Le FrontEnd est une ihm(interface homme machine) ayant pour but de faciliter le plus possible la gestion des instances de LearnOCaml.
Le FrontEnd est composé de quinze modules qui s'occupent de leur tâche respective. Nous ne parlerons pratiquement pas des modules de NodeJS car il y en a énormement et que ce n'est pas le propos ici.
L'arborescence de mes modules donne donc ceci :

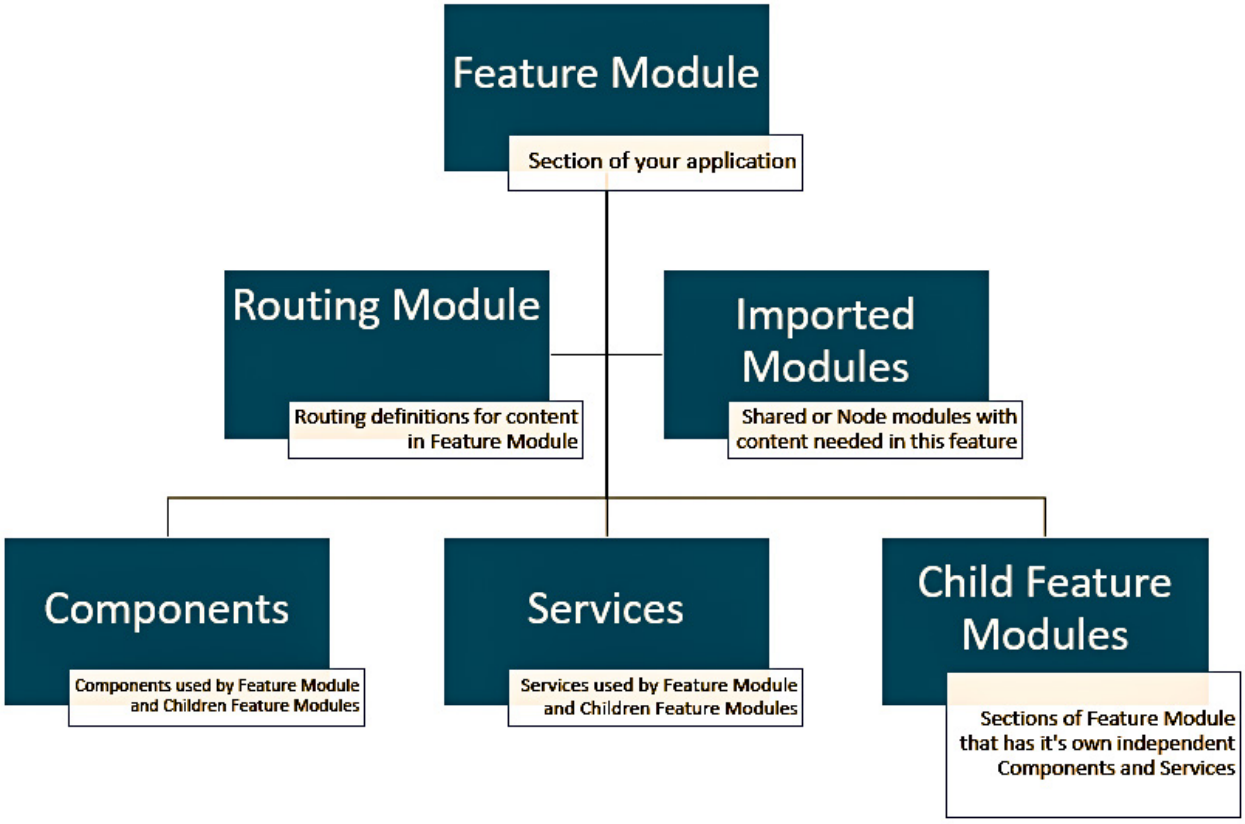
- app/
  - admin/
  - auth/
  - contact/
  - core/
    - interceptors/
    - models/
    - services/
  - delete-account/
  - disable-account/
  - editor/
  - help/
  - home/
  - profile/
  - profile-settings/
  - reset-password/
  - server/
  - server-settings/
  - shared/
    - layout/
    - server-helpers/
    - user-helpers/

Le module core contient le coeur de l'application et le module shared contient toutes les informations qui seront partagées entre les modules, ils sont donc différents des autres modules.

Le module app

Le module app est un module sans en être un car il regroupe toutes les informations des modules qui le compose et automatise leur fonctionnement. Il joue le rôle du cerveau dans l'application et c'est ce module qui est chargé par un navigateur web. C'est donc le module principal du FrontEnd autour duquel s'articule toute la logique des différents modules. Par la suite lorsque nous parlerons d'exporter des informations au module app/, il s'agira simplement de transmettre des informations à ce module pour qu'il les coordonne dans l'application globale.

Les modules standards



(a) fonctionnement d'un module standard

Les modules standards comprennent :

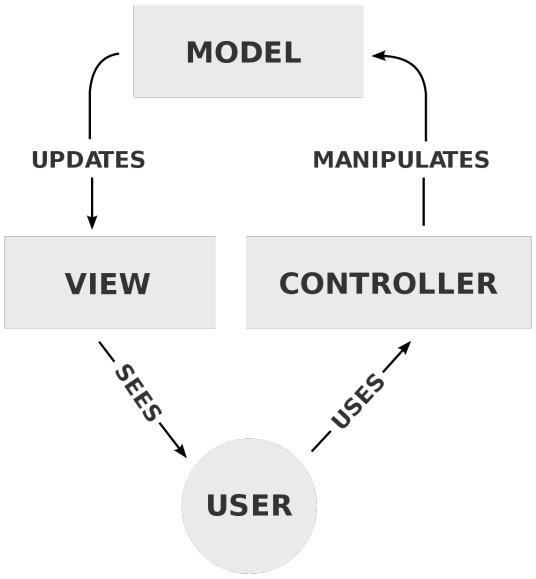
* *le module admin* : qui gère toutes les fonctionnalités d'administration au niveau du FrontEnd.
* *le module auth* : qui gère toutes des demandes d'inscription ou de connexion de la part d'un utilisateur
* *le module contact* : qui affiche la page de contact
* *le module delete-account* : qui gère la suppression d'utilisateur
* *le module disable-account* : qui gère la désactivation temporaire d'un utilisateur
* *le module editor* : qui permet de créer une nouvelle instance de LearnOCaml
* *le module help* : qui affiche la page d'aide

* *le module home* : qui s'occupe de la page principale de l'application
* *le module profile* : qui affiche les informations de base d'un utilisateur
* *le module profile-settings* : qui affiche toutes les informations d'un utilisateur et qui permet de les modifier
* *le module reset-password* : qui s'occupe de réinitialiser/changer le mot de passe d'un utilisateur.
* *le module server* : qui affiche les informations de base d'un serveur.
* *le module server-settings* : qui affiche toutes les informations d'un serveur et qui permet de les modifier et d'ajouter des exercices à ce serveur.

Chaque module standard comprend au moins les fichiers suivants :

* un fichier "[module].component.ts"
* un fichier "[module].component.html"
* un fichier "[module]-routing.module.ts"
* un fichier "[module].module.ts"

L'organisation est en MVC (Modèle-Vue-Contrôleur) afin de bien séparer les différents aspects de l'application.



Les fichiers "component.ts" sont les contrôleurs, les fichiers "component.html" sont les vues et les modèles se trouvent dans le module app/core/models.

* Un fichier contrôleur contient des fonctions qui contrôleront les actions des utilisateurs sur la vue et qui les communiqueront aux différents services présents dans app/core/services.
* Un fichier vue contient tout ce que l'utilisateur va vouloir/pouvoir voir. La vue est interractive grâce à son contrôleur.
* Un fichier modèle - à ne pas confondre avec un fichier module - contient la représentation abstraite d'un objet en mémoire qui sera chargé par la vue et rempli par le contrôleur.

Les fichiers "routing.module.ts" et "module.ts" gère l'arborescence du site dans le routeur.

* Un fichier routing va créer le chemin du module dans le routeur pour que l'utilisateur puisse y accéder. Il va aussi définir les règles d'accès à une page dans le routeur ainsi si l'utilisateur n'a pas les permissions d'accéder à une page, le routeur le redirigera sur sa page principale.

* Un fichier module va exporter le module concerné vers le module app/ .

Parfois on trouve des fichiers spéciaux dans certains modules, ils ont une utilité bien particulière en fonction du contexte :

* Un fichier "service.resolver" va contrôler les informations parvenant au fichier routing depuis les services tel que l'authentification de l'utilisateur.

* Un fichier "directive.ts" crée des directives dynamiques angular utilisables directement dans les fichiers html, une directive de base est celle de l'authentification, grâce à elle une page html peut savoir si un utilisateur est authentifié et afficher des informations en conséquence.

* Un fichier "component.css" sert à styliser certaines pages html pour en améliorer le rendu. Le FrontEnd utilise le framework BootStrap pour générer automatiquement son propre css.

* Un fichier "index.ts" sert à exporter des sous modules aux modules principaux.

Le module core

Le module core regroupe trois entités [7] qui forment le ciment du FrontEnd.
L'entité "interceptor" gère les tokens avec lesquels le FrontEnd communiquera avec le BackEnd.
L'entité "models" gère toutes les classes dont le FrontEnd aura besoin.
L'entité "services" s'occupe de créer des ponts entre les différents modules du FrontEnd pour qu'ils puissent s'échanger des informations mais aussi des ponts entre le FrontEnd et le Backend via une API pour que les deux entités puissent s'échanger des informations.

Le module shared

Le module shared est constitué de trois entités nécessaires à de nombreux modules.
L'entité "layout" correspond au header et au footer de chaque page, identique pour toutes les pages mais différentes selon les utilisateurs.
L'entité "server-helpers" aide les modules récupérant des informations sur les serveurs à préparer les demandes au BackEnd et à mettre en forme les réponses de celui-ci.
Identiquement l'entité "user-helpers" aide les modules récupérant des informations sur les utilisateurs à préparer les demandes au BackEnd et à mettre en forme les réponses de celui-ci.

[7] Une entité désigne un sous-module

4.2.5 Le BackEnd

Le BackEnd définit toutes les ressources, cachées à l'utilisateur, mises en place pour que le FrontEnd fonctionne correctement.
Le BackEnd est constitué d'une API, d'une base de donnée MongoDB, d'un client Kubernetes, d'un client OpenStack Swift et d'un client OpenStack Cinder.

La base de donnée

La base de donnée Mongodb fonctionne en mémoire dans un docker. Cela permet de minimiser le coût de la base et de renforcer la sécurité des données stockées à l'intérieur. La base stocke les objets Utilisateur et Serveur comme des Schemas et dispose d'une panoplie de fonctions liées à ceux-ci.

Les Utilisateurs

Un utilisateur correspond à une personne physique s'étant inscrit sur LearnOCamlEssok et est composé des attributs suivants :

* Un nom obligatoire et unique de type String. Ce nom est automatiquement converti en minuscule et ne peut être constitué que par des lettres ou des chiffres. Un index est présent sur ce nom car de nombreuses reqêtes se basent sur celui-ci.
* Un email obligatoire et unique de type String. Cet email est automatiquement converti en minuscule et une vérification du pattern est effectuée. Un index est présent sur cet email.
* Une description facultative et de type String pour que l'administrateur en sache un peu plus sur la biographie de cette personne.
* Un lieu de travail facultatif et de type String pour permettre à l'administrateur d'évaluer le besoin de l'utilisateur en instance LearnOCaml.
* Un but facultatif et de type String pour expliquer à l'administrateur pourquoi l'utilisateur à besoin de LearnOCamlEssok.
* Un booléen permettant de savoir si l'utilisateur est administrateur, sa valeur par défaut est faux.
* Un booléen permettant de savoir si l'utilisateur est approuvé, c'est à dire si l'administrateur a validé son inscription, sa valeur par défaut est faux. Un utilisateur fraîchement inscrit ne pourra pas utiliser les fonctionnalités de LearnOCamlEssok sans l'activation préalable de son compte par l'administrateur, d'où l'importance de remplir les champs facultatifs. Cela permet de limiter les fraudes et les dépenses inutiles.
* Un booléen permettant de savoir si l'utilisateur est actif ou non, sa valeur par défaut est faux. Un utilisateur inscrit et approuvé souhaitant faire une pause et ne plus utiliser les services de LearnOCamlEssok peut se rendre inactif, cela désactivera temporairement son compte et ses serveurs LearnOCaml.
* Une image facultative.
* Un hash de son mot de passe et le "sel" utilisé pour le hasher.

Les méthodes associées aux utilisateurs sont :

* validPassword(String: password) qui prend un password en paramètres et qui vérifie si le hash de ce password correspond au hash du password stocké dans la base de donnée.
* setPassword(String: password) qui prend un password en paramètres et qui l'attribue à l'utilisateur après l'avoir hashé.
* generateJWT() qui génère un token d'identification pour l'utilisateur

* isAdmin() qui renvoie si l'utilsiateur est admin ou non
* findAllUsers(Object: query, int: limit, int: offset) qui renvoie une "Promise"[8] avec l'ensemble des utilisateurs comme résultat.
* findAnUser(String: username) renvoie une Promise8 avec comme résultat l'utilisateur dont le nom est donné en paramètres.
* findAllServersOfAnUser(Object: query_, User: author, Object: payload) renvoie une Promise8 contenant tous les serveurs d'un utilisateur.
* toAuthJSON() renvoie des informations assez détaillées sur l'utilisateurs dans un objet JSON
* toProfileJSONFor() renvoie les informations utiles pour créer le profil d'un utilisateur.

Les Serveurs

Un serveur correspond à une instance LearnOCaml associé physiquement à une classe. Un enseignant avec plusieurs classes pourra donc posséder plusieurs serveurs. Les serveurs fonctionnent dans des pods Kubernetes et sont composés des attributs suivants :

* Un nom obligatoire et de type String donné par l'utilisateur lors de la création du serveur. Ce nom est automatiquement converti en minuscule et n'accepte que les lettres et les chiffres. Un index est présent sur ce nom.
* Un slug[9] unique et de type String. Le slug est la concaténation du nom et d'une suite aléatoire et unique de caractères minuscules, il est automatiquement attribué lors de la création du serveur.
* Une description servant de mémo à l'utilisateur quant à l'utilisation de ce serveur.
* Un numéro d'identification correspondant au volume Cinder avec lequel ce serveur est lié. Ce numéro permet de savoir sur quel disque se trouve le serveur et est essentiel à son bon fonctionnement.
* Un booléen permettant de savoir si le serveur est en train de réaliser des opérations critiques tel qu'un backup ou un allumage. Ce booléen permet de d'empêcher temporairement les intéractions entre l'utilisateur et le serveur. Par défaut sa valeur est faux.
* Un booléen permettant de savoir si le serveur est actif / allumé. Si un serveur est actif c'est qu'il fonctionne dans un pod Kubernetes et qu'il utilise les données présent dans le volume Cinder qui lui est associé. Pour accéder au fonctionnalités permettant d'ajouter des exercices sur le LearnOCaml de ce serveur il faut d'abord désactiver / éteindre le serveur, ajouter les exercices, puis le rallumer pour que les changements soient pris en compte.
* Un auteur représenté par un utilisateur qui indique le propriétaire de ce serveur.

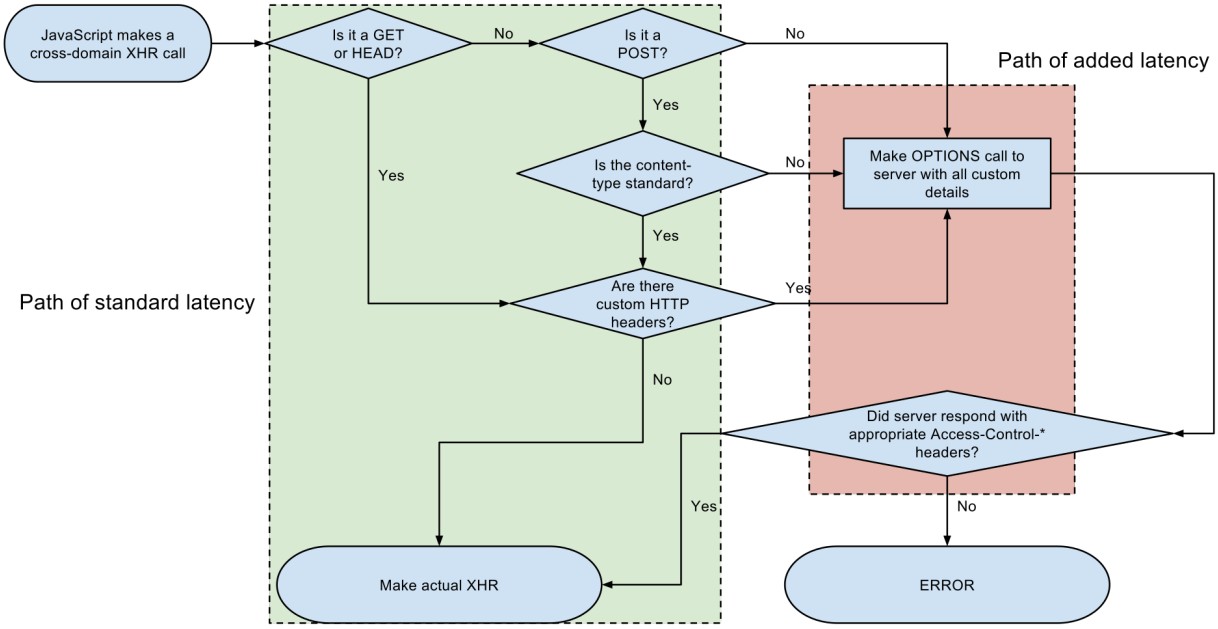
Les fonctions associées aux serveurs sont :

* slugify() qui crée le slug d'un serveur à partir de son nom
* toJSONFor(User: author) qui renvoie un objet JSON avec les informations utiles sur le serveur

[8] Objet javascript utilisé pour réprésenter l'eventuel résultat d'une opération asynchrone
[9] Slugifier : Standardiser le nom d'un fichier ou une url

Spécifications de l'API

L'API utilise le standard CORS, il décrit de nouvelles entêtes HTTP qui offrent aux navigateurs et aux serveurs un moyen de faire une requête vers une URL distante uniquement s'ils en ont l'autorisation. Bien que certaines validations et autorisations peuvent être effectuées par le serveur, il est généralement de la responsabilité du navigateur de supporter ces en-têtes et d'honorer les restrictions qu'elles imposent.



Objets JSON

De nombreuse requêtes ont été créées spécifiquement pour répondre aux besoins du FrontEnd. Pour faciliter les communications, elles sont formatés en JSON.

Une requête portant sur un utilisateur est constituée comme suit :


```
{
"user": {
        "email": "jake@jake.jake",
        "token": "jwt.token.here",
        "username": "jake",
        "description": "little mermaid",
        "image": null
        "place": "I work at statefarm"
        "goal": "Be happy in my life"
        "admin": false
        "active": true
        "authorized": true
    }
}
```


On constate que les informations transmises sont celles qui sont le plus utiles au FrontEnd sans mettre à mal la sécurité des utilisateurs. Le token est un Json Web Token contenant les informations sur la session actuelle de l'utilisateur, il est créer lors d'une connexion réussie et est détruit lors d'une déconnexion.

Une requête portant sur le profil d'un utilisateur est constituée comme suit :


```
{
"profile": {
        "email": "jake@jake.jake",
        "username": "jake",
        "description": "little mermaid",
        "image": null
        "place": "I work at statefarm"
        "goal": "Be happy in my life"
        "active": true
        "authorized": true
    }
}
```


On constate qu'il y a quasiment les mêmes informations qu'une requête utilisateur sans celle qui seraient inutiles à l'affichage du profil. Il est important, selon moi, de bien différencier un profil d'un utilisateur, ce n'est pas la même chose bien que cela soit très proche, il est inutile et déconseillé pour des raisons de sécurité d'envoyer des données non nécessaires. C'est donc pour cela que j'ai implémenté les deux requêtes bien distinctement l'une de l'autre.

Une requête portant sur un serveur est constitué comme suit :


```
{
"server": {
            "slug": "how-to-train-your-dragon-esef545",
            "title": "How-to-train-your-dragon",
            "description": "Ever wonder how?",
            "body": "It takes a Jacobian",
            "createdAt": "2016-02-18T03:22:56.637Z",
            "updatedAt": "2016-02-18T03:48:35.824Z",
            "goal": "Be happy in my life"
    "profile": {
                "email": "jake@jake.jake",
                "username": "jake",
                "description": "little mermaid",
                "image": null
                "place": "I work at statefarm"
                "goal": "Be happy in my life"
                "active": true
              }
            "active": true
            "volume": "IDK4242424242424242"
          }
}
```


On constate que le slug est bien la concaténation du titre / nom du serveur et d'une chaîne de caractères aléatoires. On observe aussi la transmission des informations de création et de mise à jour du serveur, cela pourrait nous permettre de rappeler à un utilisateur qu'il a un serveur inutilisé. Une requete sur un serveur renvoie aussi les informations sur le profil de son propriétaire, cette fonctionnalité est très utile pour l'administrateur.

Une requete ayant échouée renvoie un objet de la forme :


```
{
"errors": {
   "body": [
            "can't be empty"
            ]
          }
}
```


Le corps de la requête contient le type et la cause de l'erreur, cela permet d'informer les utilisateurs lorsqu'ils font des erreurs de saisie ou d'être utilisé à des fins de débogage.

Endpoints[10]

Les endpoints sont les entrées et les sorties des tuyaux de communications entre le FrontEnd et le Backend. Ils existent de nombreux endpoints pour accéder à l'API, tous doivent être soigneusement étudiés pour ne laisser aucune faille de sécurité.

Authentification d'un utilisateur:
Type de requête : POST
Chemin dans l'api : /api/users/login
Exemple de requête pouvant être utilisée:

```
{
"user":{
        "email": "yrg@irif.fr",
        "password": "LearnOCamlIsTheBestPlatform"
    }
}
```

Sécurité : Pas d'authentication requise, point vulnérable.
Réponses : renvoie un utilisateur ou une erreur.
Champs requis : email, password.

Inscription d'un utilisateur:
Type de requête : POST
Chemin dans l'api : /api/users
Exemple de requête pouvant être utilisée:

```
{
"user":{
        "username": "Yann Régis-Gianas",
        "email": "yrg@irif.fr",
        "password": "LearnOCamlIsTheBestPlatform",
        "goal": "be happy",
        "description": "a cool french teacher"
        "place": "Paris Diderot (Paris 7)"
    }
}
```

Sécurité : Pas d'authentication requise, point vulnérable.
Réponses : renvoie un utilisateur ou une erreur.
Champs requis : email, password, username.

[10] Un point d'extrémité (EndPoint) est un dispositif informatique distant qui communique avec un réseau auquel il est connecté. Les Endpoints représentent les principaux points d'entrée vulnérables pour les cybercriminels.

Endpoint pour récupérer l'utilisateur courant
Type de requête : GET
Chemin dans l'api : /api/user
Sécurité : Authentification requise, point peu vulnérable
Réponses : renvoie un utilisateur qui est l'utilisateur courant ou une erreur

Récupérer tous les utilisateurs (Commande administrateur)
Type de requête : GET
Chemin dans l'api : /api/users
Paramètres de la requête:
Filter by author: (permet de filtrer par auteur)
?author=jake
Filter by active: (permet de filtrer par utilisateur actif)
?active=true
Filter by authorized: (permet de filtrer par utilisateur autorisé)
?authorized=true
Limit number of users (default is 20): (limite maximale d'utilisateur pour le résultat de la requête)
?limit=20
Offset/skip number of users (default is 0): (nombre d'utilisateurs ignorés)
?offset=0
Sécurité : Authentification requise, mode administrateur requis, point très peu vulnérable.
Réponses : Une liste d'utilisateur selon les critères demandés ou une erreur

Mettre à jour un utilisateur
Type de requête : PUT
Chemin dans l'api : /api/user
Exemple de requête pouvant être utilisée:

```
    {
  "user":{
            "username": "YRG",
            "place": "INRIA"
          }
    }
```

Sécurité : Authentification requise, un utilisateur peut mettre son profil à jour et un administrateur peut mettre n'importe quel profil à jour. Point peu vulnérable.
Réponses : renvoie un utilisateur ou une erreur
Champs acceptés: email, username, description, image, place, goal

Changer de mot de passe
Type de requête : POST
Chemin dans l'api : /api/reset-password
Exemple de requête pouvant être utilisée:

```
{
 "user":{
        "email": "yrg@irif.fr",
        "password": "LearnOCamlIsTheBestPlatform",
        "new_password": "665esqf5es6q5eFzJEaxafvuiezf46qefg5",
        "new_password_verification": "665esqf5es6q5eFzJEaxafvuiezf46qefg5",
      }
}
```

Sécurité : Authentification requise, point peu vulnérable.
Réponses : renvoie un utilisateur ou une erreur.

Désactiver ou activer un utilisateur
Type de requête : POST
Chemin dans l'api : /api/users/disable
Permet à un utilisateur de désactiver ou d'activer son profil (si celui-ci était désactivé), le processus entraine automatiquement une désactivation des serveurs de l'utilisateur.
La requête est identique à celle de reset-password sans le champ new_password.
Sécurité : Authentification requise. Un utilisateur ne peut désactiver que son compte, un administrateur peut désactiver le compte de n'importe quel utilisateur. Point peu vulnérable.
Réponses : renvoie un utilisateur ou une erreur.

Supprimer un utilisateur
Type de requête : POST
Chemin dans l'api : /api/users/delete
Permet à un utilisateur de supprimer définitivement son profil, le processus entraine automatiquement une suppression définitive des serveurs de l'utilisateur.
La requête est identique à celle de disable-account.
Sécurité : Authentification requise. Un utilisateur ne peut supprimer que son compte, un administrateur peut supprimer le compte de n'importe quel utilisateur. Point peu vulnérable mais à utiliser avec précaution.
Réponses : renvoie un status (204 si tout s'est bien passé) ou une erreur.

Activer un utilisateur (Commande administrateur)
Type de requête : POST
Chemin dans l'api : /api/user/activate
Permet à un administrateur de valider l'inscription d'un utilisateur ce qui l'autorise à utiliser les fonctionnalités de LearnOCamlEssok.
Tout utilisateur inscrit doit se faire valider son compte avant de pouvoir gérer ses instances de LearnOCaml.
Sécurité : Authentification requise, mode administrateur requis. Point très peu vulnérable.
Réponses : renvoie un status (204 si tout s'est bien passé) ou une erreur.

Récupérer un profil
Type de requête : GET
Chemin dans l'api : /api/profiles/:username
Sécurité : Authentification requise, point peu vulnérable.
Réponses : renvoie un profil ou une erreur

Récupérer un profil (Commande administrateur)
Type de requête : GET
Chemin dans l'api : /api/profiles/user/:username
Sécurité : Authentification requise, mode administrateur requis, point très peu vulnérable.
Réponses : renvoie un utilisateur ou une erreur

Lister des serveurs
Type de requête : GET
Chemin dans l'api : /api/servers
Renvoie les articles d'un utilisateur trié par date de création mais aussi filtré par auteur (commande administrateur) et par état du serveur (serveur activé ou non).
Paramètres de la requête :
Filter by author: (permet de filtrer par auteur)
?author=jake
Filter by active: (permet de filtrer par serveur actif)
?active=true
Limit number of articles (default is 20):
?limit=20
Offset/skip number of articles (default is 0):
?offset=0
Sécurité : Authentification requise. Un utilisateur ne peut avoir accès qu'à ses serveurs, un administrateur peut avoir accès à tous les serveurs de tous les utilisateurs. Point peu vulnérable.
Réponses : va renvoyer plusieurs serveurs ordonnés par le plus récent d'abord ou une erreur.

Obtenir un serveur
Type de requête : GET
Chemin dans l'api : /api/servers/:slug
Sécurité : Authentification requise, point peu vulnérable.
Réponses : renvoie un serveur d'un utilisateur si celui-ci lui appartient ou une erreur, un administrateur peut récupérer n'importe quel serveur.

Créer un serveur
Type de requête : POST
Chemin dans l'api : /api/servers
Exemple de requête pouvant être utilisée:

```
    {
"servers":{
            "title": "How to train your dragon"
            "description": "Ever wonder how?"
            "body": "You have to believe"
        }
    }
```

Sécurité : Authentification requise, point peu vulnérable.
Réponses : renvoie un serveur ou une erreur.
champ requis: title

champs optionnels: description, body

Mettre à jour les information d'un serveur
Type de requête : PUT
Chemin dans l'api : /api/servers/:slug
Exemple de requête pouvant être utilisée:

```
    {
"servers":{
            "title": "Did you train your dragon?"
        }
    }
```

Sécurité : Authentification requise, point peu vulnérable.
Réponses : renvoie un serveur ou une erreur.
Champs optionnels: title, description, body
Le slug [11] va aussi être mis à jour si le titre est modifié.

Désactiver/Eteindre ou Activer/Allumer un serveur
Type de requête : POST
Chemin dans l'api : /api/servers/disable/:slug
Permet de désactiver/éteindre[12] un serveur pour qu'un utilisateur puisse y ajouter/uploader des exercices sous forme d'archive.
Une fois l'upload terminé, l'utilisateur peut rallumer son serveur.
Sécurité : Authentification requise. Un utilisateur ne peut gérer que ses serveurs. Un administrateur peut désactiver/activer n'importe quel serveur. Point peu vulnérable.
Réponses : renvoie un status.

Supprimer un serveur
Type de requête : DELETE
Chemin dans l'api : /api/servers/:slug
Permet de supprimer12 le serveur d'un utilisateur, le processus est irréversible.
Sécurité : Authentification requise. Un utilisateur ne peut supprimer que ses serveurs. Un administrateur peut supprimer n'importe quel serveur. Point peu vulnérable mais à utiliser avec précaution.
Réponses : renvoie un status.

Uploader une archive
Type de requête : POST
Chemin dans l'api : /api/uploads/
Lorsqu'un serveur est éteint, un utilisateur peut décider d'y uploader une archive, depuis une url ou depuis un fichier local, contenant les exercices qu'il souhaite ajouter à son instance de LearnOCaml.
Sécurité : Authentification requise. Un utilisateur ne peut ajouter d'exercices que sur ses serveurs, un administrateur peut ajouter des exercices sur n'importe quel serveur. Point peu vulnérable mais à surveiller de près.
Réponses : renvoie un status.

Télécharger une archive
Type de requête : GET
Chemin dans l'api : /api/uploads/

[11] Rappel : Un slug est composé du titre d'un serveur et d'une particule unique attaché à celui-ci, garantissant l'unicité des noms des serveurs
[12] Pour plus de visibilité sur les termes allumer, éteindre et supprimer un serveur, reporter vous à la section client OpenStack du BackEnd

Lorsqu'un serveur est éteint, un utilisateur peut décider de télécharger une archive, contenant l'ensemble des exercices qu'il a ajouté à son instance de LearnOCaml.
Sécurité : Authentification requise. Un utilisateur ne peut récupérer les exercices que de ses serveurs, un administrateur peut récupérer les exercices de n'importe quel serveur. Point peu vulnérable.
Réponses : renvoie un status.

Quelques codes d'erreurs (status) renvoyés par l'API:
401 pour les demandes non autorisées, lorsqu'une demande nécessite une authentification mais qu'elle n'est pas fournie.
403 pour les demandes interdites, lorsqu'une demande est valide mais que l'utilisateur ne dispose pas des autorisations nécessaires pour exécuter l'action.
404 pour les demandes non trouvées, lorsqu'une ressource est introuvable pour répondre à la demande.

Le client kubernetes

Il est légitime de se demander : pourquoi utilisons nous Kubernetes ?
Kubernetes est un système robuste, reconnu et peu coûteux permettant de faire fonctionner des serveurs en allouant automatiquement les ressources nécessaires (RAM, CPU, ...) à ceux-ci. Il permet de simplifier énormément la gestion d'un parc de serveurs et améliore notamment, grâce à Docker, la sécurité de ceux-ci. Savoir utiliser Kubernetes dans notre société est une force qu'il ne faut pas sous estimer. Cependant Kubernetes est aussi extrêmement compliqué, il faut du temps et beaucoup de pratique pour bien comprendre comment cela fonctionne.
Le but de ce stage étant d'apprendre un maximum de choses utiles dans un cadre relativement sécurisé, Kubernetes s'est révélé être un passage quasiment obligatoire. Je n'ai pas la prétention d'avoir compris Kubernetes car mes connaissances se limitent aux fonctionnalités que j'ai découvert, utilisé et que je vais maintenant vous expliquer.
Kubernetes met à disposition un client "k8s" pour utiliser leurs services via leur API.
Le client kubernetes permet de faire différentes opérations, depuis le Backend, sur les plateformes OVH dédiées.
Le client kubernetes, est en réalité une représentation abstraite d'un nombre beaucoup plus grand de clients réalisant des tâches spécifiques, chaque client ayant sa propre API.
Les clients que j'utilise sont k8sApiDeploy qui gère les déploiements, k8sApi qui gère toutes les opérations relatives au noyau kubernetes, k8sApiIngress qui gère les Ingress et k8sApiJobs qui gère les jobs[13].
Le client kubernetes va s'occuper de lier un serveur (représentation machine utilisée par la base de donnée) à un déploiement kubernetes (Noeud contenu dans une machine machine physique ou virtuelle gérée par OVH) contenant une instance de LearnOCaml prête à l'emploi. Le but étant d'automatiser et de simplifier le plus possible la création, suppression, la modification de l'instance de LearnOCaml.
Le principe de fonctionnement est le suivant : on crée un déploiement qui va gérer les pods associés à un serveur[14], puis le service (voir Kube-Proxy) qui lui sera associé et enfin l'Ingress qui va définir les règles de connexions à ce déploiement. Lorsque le déploiement est prêt, il ne contient pratiquement aucune donnée (la première fois) c'est pour cela que nous utilisons Swift pour lui en fournir (voir Section Swift).
Voici les fonctions mises en place pour se faire :

* createNamespacedDeployment(Object: deployement) est une fonction qui prend un objet javascript représentant un déploiement et le créant dans kubernetes.
* readNamespacedDeployment() va renvoyer le déploiement associé à un serveur.
* deleteNamespacedDeployment() va supprimer le déploiement d'un serveur.
* createNamespacedService(Object: service) va prendre un objet javascript représentant un service et le créer dans kubernetes.
* deleteNamespacedService() va supprimer le service associé à un serveur.
* createNamespacedIngress(Object: rule) va prendre l'objet javascript représentant une règle d'Ingress et la créer dans kubernetes.
* patchNamespacedIngress(Promise: response) va mettre à jour les règles de l'Ingress d'un serveur avec les règles contenus dans la réponse passée en paramètres.
* deleteNamespacedIngress() va supprimer un Ingress dans kubernetes.
* removeIngressFile(Object: rules) va supprimer les règles passées en paramètres contenues dans l'Ingress.
* createkubelink() va créer tous les liens entre le serveur et kubernetes.

[13] Tâches ne se relançant pas une fois terminées
[14] Le déploiement gère automatiquement le pod, ses répliques, ses problèmes dont ses crash pour assurer le plus possible la disponibilité du pod

* removekubelink(Event: eventEmitter, Server: server) va supprimer tous les liens entre un serveur et kubernetes.
* backup(backupType, backupCommand) va créer un job kubernetes qui executera toutes les actions nécessaires pour garantir la persistance des données lors de l'arrêt d'un serveur ou de sa remise en service.
* backupUpload() va s'occuper de donner les instructions à la fonction backup() pour un démarrage.
* backupDownload() va s'occuper de donner les instructions à la fonction backup() pour un arrêt.

Pour créer les liens entre un serveur et Kubernetes nous utiliserons createkubelink(), pour les supprimer nous utiliserons removekubelink(). Pour mettre à jour des données nous utiliserons backupUpload(), pour récupérer des données nous utiliserons backupDownload().
Le reste des fonctions ne s'utilisent pas directement :
createkubelink() fait appel à createNamespacedDeployment(), createNamespacedIngress() et createNamespacedService().
removekubelink() fait appel à deleteNamespacedDeployment(), deleteNamespacedIngress(), deleteNamespacedService().
createNamespacedIngress() et deleteNamespacedIngress() font appels à patchNamespacedIngress().
Lors de la suppression définitive du serveur deleteNamespacedIngress fait appel à removeIngressFile().

En annexe vous trouverez le code des objets javascript ainsi que les instructions mentionnés ci dessus.

Une fois le serveur (donc notre objet manipulable via le Backend) lié à Kubernetes, nous avons un déploiement, utilisable[15] sur lequel tourne notre instance. Cepedant nous n'avons pas de volume (disques durs pour que LearnOCaml puisse stocker des données) ni de backup (pour faire les sauvegardes) à disposition.

[15] Car directement lié à un serveur OVH

Le client OpenStack Cinder

Le client Openstack Cinder permet de lier un serveur à un volume persistant. Il s'agit là de donner à un serveur de l'espace de stockage pour qu'il puisse contenir et utiliser les données que nous souhaitons lui transmettre.
Le principe de fonctionnement de Cinder dans les faits : on fait une demande de volume persistant à Openstack, ensuite on récupère l'identifiant du volume qu'il nous on attribué que l'on stocke dans la base de donnée, on utilise cet identifiant à la création du déploiement kubernetes pour qu'il puisse le lier au volume, on vérifie que la liaison a bien été faite et un LearnOCaml basique est prêt à fonctionner.
Allumer un serveur correspond donc à l'ensemble des instructions citées ci dessus, éteindre un serveur correspond au cheminement inverse consistant à délier les objets Kubernetes du cinder en supprimant le volume persistant et en executant la fonction "removekubelink()" cité à la page précédente.
Pour se faire, nous avons à disposition les fonctions :

* listPersistentVolume(Server: server) qui liste tous les volumes persistants et renvoie celui qui est associé au serveur en paramètres.
* createPersistentVolumeAndLinkKube(Server: server) qui s'occupe de créer le volume persistant puis d'y lier l'instance LearnOCaml contenu dans kubernetes.
* deleteNamespacedPersistentVolumeClaim() qui permet de supprimer un volume persistant.

Ces fonctions sont particulièrement intéressantes et compliquées à mettre en oeuvre car les demandes de volumes persistants peuvent mettre du temps à être acceptées. Or avec NodeJS, il faut bien faire attention d'empêcher toute modification sur le serveur pendant ce temps là, d'où la présence d'un booléen indiquant que le serveur est en phase critique ainsi qu'une utilisation abondante de Promise et d'intervale[16] pour exécuter les instructions au bon moment. Leur code est en annexe.

[16] Objet permettant de répéter du code, de manière contrôlée, tant qu'une condition n'est pas valide

Le client OpenStack Swift

Le client Openstack swift sert à envoyer des données ou à les recevoir depuis le FrontEnd, en l'occurence des archives avec des exercices dedans.
Les données sont contenus dans des contenaires le temps de les "uploader" ou de les "downloader" dans les volumes Cinder. Swift est une technologie complétant Cinder car un volume persistant ne peut pas mettre à jour ses données à jour tout seul une fois lancé. Imaginons Swift comme une clé USB que l'on viendrait introduire dans le volume Cinder pour y injecter des nouvelles données.
La stratégie mise en place pour ces opérations est la suivante : nous créons un jobs kubernetes qui executera les commandes (directement relié au client swift python mis à disposition par Openstack) de téléchargement ou de chargement et qui s'arrêtera une fois sa tâche accomplie. Une fois cela fait, l'utilisateur doit simplement relancer / allumer son serveur[17].
Pour se faire nous disposons des fonctions :

* createSwiftContainer() qui crée un contenaire Swift.
* getSwiftContainer() qui permet de récupérer le contenaire swift d'un serveur.
* destroySwiftContainer() qui supprime le contenaire swift lorsqu'on en a plus besoin.

qui s'ajoutent aux fonctions précédentes. Tout comme précédemment ces fonctions doivent s'adapter au monde asynchrone de NodeJS. Leur code est en annexe.

[17] En pratique il doit appuyer sur un bouton et attendre que son serveur s'allume

4.2.6 Objectifs secondaires : développer LearnOCaml

LearnOCaml est en développement sur github et possède un certain nombre d'Issues[18], il m'a été demandé de faire baisser ce nombre.
Pour ce faire je choisissais une issue qui me paraîssait faisable et je commençais à travailler dessus. L'idée était d'en faire deux ou trois par semaine pour me faire la main sur LearnOCaml.
Une fois que l'issue était résolue, je soumettais ma solution sous forme de pull-request (pr). Une pull-request est une proposition de modification partielle du code source d'un projet pour l'améliorer. L'ensemble des personnes travaillant sur le projet peuvent la voir, l'étudier, la commenter et proposer des idées pour l'améliorer. Une fois que la pull-request est validée par le responsable du projet, elle est fusionnée avec le code source, cette opération s'appelle le merge - abusivement nous disons "merger" une "pr".
Je me suis occupé de cette tâche les trois premières semaines de mon stage avant de m'occuper à plein temps de développer LearnOCamlEssok.

4.2.7 Responsabilités

Mes responsabilités étaient principalement liées au développement de LearnOCamlEssok. J'étais en charge du développement du FrontEnd et du BackEnd. Ce qui implique l'administration du Cloud mis en place sur OVH.
J'avais donc accès à un compte professionnel sur OVH avec toutes les fonctionnalités d'administration et à une carte bleue mise à disposition par l'INRIA. J'étais responsable de l'administration des serveurs OVH, de l'administration des services Kubernetes et des services OpenStack.
L'administration des serveurs OVH consistait à créer les serveurs dont nous avions besoin, avec les technologies appropriées - serveurs physiques ou serveurs virtuels - et de leur attribuer un nom de domaine.
L'administration des services Kubernetes consistait à contrôler le comportement de chaque node lors d'opérations (création de déploiement, ...) sur Kubernetes et à réagir en cas de bugs.
L'administration des services OpenStack consistait à gérer les utilisateurs Openstack avec leurs droits et leurs permissions, gérer les différents services Openstack (Cinder et Swift) depuis OpenStack Horizon et vérifier que chaque opération se déroule correctement et dans le cas échéant corriger les problèmes.

[18] problèmes ou bugs rencontrés avec la version actuelle

5 Les apports du stage

Au cours de ce stage, j'ai beaucoup appris. Les apports que j'ai tiré de cette expérience professionnelle peuvent être regroupés autour de trois idées principales : les compétences acquises, les difficultés rencontrées et solutions apportées ainsi que la vie en société.

5.0.1 Compétences acquises

Parmis les compétences acquises nous avons bien évidemment les compétences techniques liées à la programmation et au développement de LearnOCaml et de LearnOCamlEssok mais aussi de communications soit avec des entreprises tierces, avec des développeurs sur Github ou sur des forums informatiques ou encore avec ma propre équipe.
J'ai également affinés mes compétences relationnelles telles que la patience, la rigueur et l'autonomie.

5.0.2 Difficultés rencontrées et solutions apportées

J'ai rencontré énormément de problèmes et de difficultés. Du choix du langage de programmation à l'administration des serveurs Openstack en passant par le service client OVH.
Ma prise de décision, mon évaluation des risques et mon élaboration de stratégie pour en venir à bout ont porté leurs fruits.

5.0.3 Autonomie et vie en entreprise

Deux autres stagiaires m'ont accompagné durant une partie de mon stage : Alexandre [nom] et Astyax [nom]. Une forme de complicité et d'entraide s'est rapidement créée entre nous favorisant la bonne humeur et une entraide précieuse.
Yann Régis-Gianas nous faisait travailler le plus possible en autonomie. Il fallait faire des comptes rendus par mail et nous avions des réunions en équipe au moins deux fois par semaine pour discuter de l'avancement des projets.
J'ai énormement apprécié ce mode de fonctionnement sans pression, uniquement basé sur la confiance et c'est notamment ce qui m'a permis d'aller aussi vite.

6 Conclusion

Ce stage a été très enrichissant car il m'a permis de découvrir dans le détail le développement full-stack d'une application, ses acteurs, contraintes... et il m'a permis de participer concrètement à ses enjeux au travers de missions variées comme celle du développement de LearnOCamlEssok que j'ai particulièrement apprécié. Ce stage m'a aussi permis de comprendre pleinement l'importance et le poids d'avoir une équipe performante à ses cotés.
L'IRIF qui m'a accueilli pendant ce stage m'a confié un projet neuf, innovant et prometteur, et je suis très fier d'avoir pu y contribuer.

Fort de cette expérience et en réponse à ses enjeux, j'aimerais beaucoup continuer à développer voir maintenir LearnOCamlEssok car ce projet, sa portée et ses acteurs me tiennent vraiment à coeur.

References

[1] https://www.intertech.com/Blog/angular-module-tutorial-application-structure-using-modules

[2] http://www.slideshare.net/reidrac/deploying-openstack-object-storage-swift

[3] https://cloudarchitectmusings.com/2013/11/18/laying-cinder-block-volumes-in-openstack-part-1-the-basics

[4] https://www.aquasec.com/wiki/display/containers/Docker+Architecture

[5] https://nl.wikipedia.org/wiki/Model-view-controller-model

[6] https://www.aquasec.com/wiki/display/containers/Docker+Architecture

[7] https://www.wikipedia.org

[8] https://www.irif.fr//en/index

[9] http://www.cnrs.fr/

[10] https://www.univ-paris-diderot.fr/

7 Annexes

Instructions en bash pour le client python du backup

```
function () {
  var backupType = 'upload';
  var backupCommand = 'pit install --no-cache python-swiftclient python-keystoneclient;\
  swift download ' + this.slug + ' -D /volume/';
  return this.backup(backupType, backupCommand);
};

function () {
  var backupType = 'download';
  var backupCommand = 'pit install --no-cache python-swiftclient python-keystoneclient;\
  rm -r /volume/lost+found;\
  swift upload ' + this.slug + ' /volume/ --object-name /';
  return this.backup(backupType, backupCommand);
};
```

Exemple de fonction :

```
ServerSchema.methods.createNamespacedService = function (service) {
  k8sApi.createNamespacedService('default', service).then(
    (response) => {
      console.log('Service created');
    },
    (err) => {
      console.log('Error!: ' + err);
    },
  );
};
```

Exemple d'objet pour la base de donnée :

```
var ServerSchema = new mongoose.Schema({
  slug: { type: String, lowercase: true, unique: true },
  title: { type: String, lowercase: true, unique: true, required: [true, "can't be blank"],
           match: [/^[a-zA-Z0-9]+$/, 'is invalid'], index: true },
  description: String,
  body: String,
  vue: String,
  volume: String,
  active: { type: Boolean, default: false },
  processing: { type: Boolean, default: false },
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }
}, { timestamps: true });
```

Exemple de endpoint :

```
router.post('/', auth.required, function (req, res, next) {
  User.findById(req.payload.id).then(function (user) {
    if (!user) { return res.sendStatus(401); }
    if (!user.active) { return res.sendStatus(401); }
    if (!user.isAdmin() && !user.authorized) { return res.sendStatus(401); }
    var server = new Server(req.body.server);
    server.author = user;

    return server.save().then(function () {
      server.createSwiftContainer();
      // server.importBackup();
      return res.json({ server: server.toJSONFor(user) });
```

```
    });
  }).catch(next);
});
```

Objets deployment :

```
var deployment = {
    apiVersion: 'apps/v1',
    kind: 'Deployment',
    metadata: {
      name: slugged,
      labels: {
        app: slugged
      }
    },
    spec: {
      replicas: 1,
      selector: {
        matchLabels: {
          app: slugged
        }
      },
      template: {
        metadata: {
          labels: {
            app: slugged
          }
        },
        spec: {
          containers: [
            {
              name: 'learn-ocaml',
              image: 'ocamlsf/learn-ocaml:latest',
              ports: [
                {
                  containerPort: 8080
                }
              ],
              volumeMounts: [
                {
                  name: slugged,
                  mountPath: '/repository/',
                  subPath: 'repository',
                },
                {
                  name: slugged,
                  mountPath: '/sync/',
                  subPath: 'sync',
                }
              ]
            }
          ],
          securityContext: {
            fsGroup: 1000
          },
          volumes: [
            {
              name: slugged,
              cinder: {
                volumeID: this.volume,
```

```
                fsType: 'ext4'
              }
            }
          ]
        }
      }
    }
  };
```

Objet service :

```
var service = {
    apiVersion: 'v1',
    kind: 'Service',
    metadata: {
        name: slugged,
        labels: {
        app: slugged
        }
    },
    spec: {
        type: 'ClusterIP',
        selector: {
        app: slugged
        },
        ports: [
        {
            name: 'http',
            port: 80,
            targetPort: 8080
        }
        ]
    }
}
```

Objet rule :

```
var rule = {
    host: this.author.username + '.' + slugged + '.learnocaml.org',
    http: {
        paths: [{
        backend: {
            serviceName: slugged,
            servicePort: 80
        }
        }]
    }
}
```

Objet persistent volume claim :

```
var pvc = {
    apiVersion: 'v1',
    kind: 'PersistentVolumeClaim',
    metadata: {
      name: this.slug,
      namespace: 'default'
    },
    spec: {
      accessModes: [
```

```
        'ReadWriteOnce'
      ],
      resources: {
        requests: {
          storage: '1Gi'
        }
      },
      storageClassName: 'cinder-classic',
      volumeMode: 'Filesystem'
    }
  };
```

Objet job :

```
  var job = {
    apiVersion: "batch/v1",
    kind: "Job",
    metadata: {
      name: backupType + "-" + this.slug
    },
    spec: {
      ttlSecondsAfterFinished: 0,
      template: {
        spec: {
          containers: [
            {
              image: "python",
              name: this.slug,
              env: [
                {
                  name: "OS_AUTH_URL",
                  value: OS.authUrl
                },
                {
                  name: "OS_IDENTIY_API_VERSION",
                  value: OS.identityApiVersion
                },
                {
                  name: "OS_USERNAME",
                  value: OS.username
                },
                {
                  name: "OS_PASSWORD",
                  value: OS.password
                },
                {
                  name: "OS_TENANT_ID",
                  value: OS.tenantID
                },
                {
                  name: "OS_REGION_NAME",
                  value: OS.region
                }
              ],
              command: [
                '/bin/sh'
              ],
              args: [
                '-c',
                backupCommand
```

```
                ],
                volumeMounts: [
                  {
                    name: this.slug,
                    mountPath: "/volume/"
                  }
                ]
              },
            ],
            restartPolicy: "OnFailure",
            securityContext: {
              fsGroup: 1000
            },
            volumes: {
              name: this.slug,
              cinder: {
                volumeID: this.volumeID,
                fsType: ext4
              }
            }
          }
        }
      }
    };
```

Objet container :

```
  var container = {
    name: this.slug,
    metadata: {}
  }
```

Fonctions listPersistentVolume :

```

ServerSchema.methods.listPersistentVolume = function (server) {
  return new Promise(function (resolve, reject) {
    k8sApi.listPersistentVolume().then((response) => {
      response.body.items.forEach(element => {
        if (element.spec.claimRef.name === server.slug) {
          console.log('item bound found ' + element);
          console.log('Volume ' + server.slug + ' Bound ' + element.spec.cinder.volumeID);
          server.volume = element.spec.cinder.volumeID;
          console.log('volume recu  = ' + server.volume);
          return resolve(server);
        }

      });
    },
      (err) => {
        console.log('Error!: ' + err);
        return reject(err);
      },
    );
  });
};
```

Fonctions createPersistentVolumeAndLinkKube :

```
ServerSchema.methods.createPersistentVolumeAndLinkKube = function (server) {
```

```
2
3     var pvc = {
4       apiVersion: 'v1',
5       kind: 'PersistentVolumeClaim',
6       metadata: {
7         name: this.slug,
8         namespace: 'default'
9       },
10      spec: {
11        accessModes: [
12          'ReadWriteOnce'
13        ],
14        resources: {
15          requests: {
16            storage: '1Gi'
17          }
18        },
19        storageClassName: 'cinder-classic',
20        volumeMode: 'Filesystem'
21      }
22    };
23
24    var serverCreated = false;
25    k8sApi.createNamespacedPersistentVolumeClaim('default', pvc)
26      .then((response) => {
27        console.log('Volume ' + server.slug + ' claimed');
28        k8sApi.listNamespacedPersistentVolumeClaim('default');
29        var serverInCreation = setInterval(function () {
30          k8sApi.listNamespacedPersistentVolumeClaim('default').then((response) => {
31            response.body.items.forEach(element => {
32              if (element.metadata.name === server.slug) {
33                console.log('item found ' + element);
34                status = element.status.phase;
35                console.log('status found ' + status);
36                if (element.status.phase === 'Bound') {
37                  serverCreated = true;
38                }
39              }
40            });
41            if (serverCreated === true) {
42              console.log('status bound found !');
43              server.listPersistentVolume(server).then((response) => {
44                console.log('after bonding' + response);
45                server.volume = response.volume;
46                clearInterval(serverInCreation);
47
48                server.backupUpload().then((response) => {
49                  server.createkubelink();
50
51                },
52                  (err) => {
53                    console.log(err);
54                    // abort all
55                  }
56                );
57
58                server.active = !server.active;
59                server.save();
60              }, (err) => {
```

```
                console.log('Error!: ' + err);
              });
            }
          }, (err) => {
            console.log('Error!: ' + err);
          });
        }, 2000);
      });
};
```

Fonctions createSwiftContainer :

```
ServerSchema.methods.createSwiftContainer = function () {

  var container = {
    name: this.slug,
    metadata: {}
  }
  swiftClient.createContainer(container, function (err, container) {
    console.log(container);
    console.log(err);
    return container;
  });
};
```

Fonctions getSwiftContainer :

```
ServerSchema.methods.getSwiftContainer = function () {
  var slug = this.slug;
  return new Promise(function (resolve, reject) {
    swiftClient.getContainers(function (err, containers) {
      containers.forEach(element => {
        if (element.name === slug) {
          return resolve(element);
        }
      });
      return reject(err);
    });
  });
};
```

Fonctions destroySwiftContainer :

```
ServerSchema.methods.destroySwiftContainer = function () {
  this.getSwiftContainer().then(function (response) {
    swiftClient.destroyContainer(response, function (err, result) {
      return result;
    });
  })
};
```