Le génie pour l'industrie

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS LOG792 PROJET DE FIN D'ÉTUDES EN GÉNIE LOGICIEL

ADAMCLOUD : PARTIE II
ORCHESTRATION DE DOCKER POUR LES ESSAIS D'ÉLASTICITÉ EN ÉTUDES GÉNOMIQUES

FRANÇOIS LANGELIER
LANF25108907

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur

ALAIN APRIL

MONTRÉAL, 23 AVRIL 2015
HIVER 2015

REMERCIEMENTS

En premier lieu, j'aimerais remercier mon professeur-superviseur Alain April qui m'a donné l'opportunité de travailler sur un projet que je trouve extrêmement intéressant et qui pourrait permettre d'améliorer la vie de plusieurs personnes éventuellement. Merci de m'avoir permis de faire partie de l'équipe.

Deuxièmement, merci à David Lauzon qui a su répondre à toutes mes questions lors de ce projet. Sans lui, je n'aurais pas pu avancer le projet comme je l'ai fait et il a su me guider vers les étapes les plus importantes à réaliser. Il a aussi mis beaucoup d'effort pour trouver des solutions à certaines embûches que j'ai eues tout au long du projet

Par la suite, j'aimerais remercier Sébastien Bonami, qui a travaillé sur la première partie du projet *Adamcloud*, et qui était toujours disponible pour répondre à mes questions sur le projet. De plus, je le remercie de la base du projet qu'il a établie avec ses recherches sur plusieurs technologies différentes et qui lui ont permises de trouver la meilleure solution.

ADAMCLOUD : PARTIE II
ORCHESTRATION DE DOCKER POUR LES ESSAIS D'ÉLASTICITÉ EN ÉTUDES GÉNOMIQUES

FRANÇOIS LANGELIER
LANF25108907

RÉSUMÉ

Ce rapport a été créé afin de documenter la partie II du projet *Adamcloud* réalisé pour le cours « LOG792 – Projet de fin d'études en génie logiciel » de l'École de technologie supérieure.

Le projet *Adamcloud* est basé sur l'utilisation d'outils de génomique du pipeline *Adam* développés par le *AMPLab* de l'*University of California, Berkeley.* De plus, il est aussi basé sur l'utilisation d'outils infonuagiques, de l'anglais « *cloud-computing* » d'où le nom du projet. L'objectif principal du projet est de réduire le temps alloué au séquençage d'un génome humain en utilisant les outils infonuagiques tout en gardant ce travail facile pour des personnes avec peu de connaissance en informatique.

Afin de réussir le séquençage, un environnement de conteneurs de *docker* est installé. Cet environnement comprend une grappe *Apache Spark* avec *Apache Hadoop* ainsi que les conteneurs pour les outils de génomique *Snap*, *Adam* et *Avocado*. Ensuite, les tâches de génomique sont exécutées.

Afin d'atteindre ces objectifs, des scripts d'orchestration ont été créés. Ces scripts permettent d'installer l'environnement de travail ainsi qu'exécuter les tâches afin de trouver la variance d'un génome humain. De plus, les images *dockers* utilisées pour le projet ont toutes été placées dans le DHR avec de la documentation pertinente.

Pour la troisième partie du projet *Adamcloud*, il faudrait que des tests de performance soient réalisés. De plus, l'intégration du projet *Adamcloud* avec les outils sous la tutelle de *docker*,

tel *docker-compose* et *docker-swarm*, ajouterait une valeur importante au projet et le rendrait encore plus facile à maintenir étant donné que ces outils sont supportés par *docker*.

TABLE DES MATIÈRES

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ADN	Acide désoxyribonucléique
AWS	*Amazon Web Services*
DHR	*Docker Hub Registry*
DNS	*Domain Name Server*
DO	*DigitalOcean*
HDFS	*Hadoop Distributed File System*
PFE	Projet de fin d'études
SGBD	Système de gestion de base de données
SSD	*Solid-State Drive*
SSH	Se*cure Shell*
VM	*Virtual Machine* – Machine Virtuelle

LISTE DES SYMBOLES ET UNITÉS DE MESURE

Ko	Kilooctect
Mo	Mégaoctet
Go	*Gigaoctet*
To	*Téraoctet*

INTRODUCTION

De plus en plus fréquemment, les médecins de la planète doivent analyser des échantillons sanguins. Que ce soit pour trouver la maladie dont est atteint un de leurs patients ou pour avoir un diagnostic plus précis, ces analyses sont de plus en plus nécessaires. De plus, de nouvelles technologies permettent de traiter ces analyses plus rapidement, c'est le cas du projet *Adamcloud*.

L'objectif principal du projet *Adamcloud* est de développer une architecture permettant d'améliorer le temps requis pour l'analyse d'un génome à partir d'un échantillon sanguin tout en gardant ce travail facile pour une personne ayant peu de connaissance informatique. Ce projet est possible grâce à de nouvelles technologies qu'on appelle les mégadonnées et par un outil de virtualisation appelé *Docker*.

Dans cette deuxième partie du projet, l'emphase a été placée sur les scripts d'orchestration afin de permettre la portabilité du projet et sa simplicité pour un utilisateur.

Ce document vous permettra de mieux comprendre le projet *Adamcloud*. Tout d'abord, une section de mise en contexte vous expliquera les outils utilisés dans le programme, puis suivra une section sur les scripts d'orchestration qui ont été développés pour le projet. Par la suite, les avancés faites dans le projet *Adamcloud* en ce qui concerne *Docker* seront abordées, puis une section sur les outils sous la tutelle du projet *Docker* permettant l'amélioration des outils d'orchestration suivra. Ensuite, les problèmes rencontrés seront explorés, avant de terminer avec les travaux futurs qu'il faudra réaliser pour la partie III et les parties subséquentes du projet *Adamcloud*.

CHAPITRE 1

MISE EN CONTEXTE

1.1 Mégadonnées

Les mégadonnées, en anglais « *big data* », sont un ensemble de systèmes qui ont pour but de remplacer les SGBD traditionnels, tels que *Oracle*, *MySQL*, *Microsoft SQL Server*, lorsque le volume des données recueilli par ces systèmes est simplement trop imposant pour un SGBD traditionnel. Les mégadonnées permettent d'améliorer les performances pour le traitement des données de par sa nature distribuée et facilement extensible, car cela permet d'utiliser des ressources sur plusieurs ordinateurs plutôt qu'un seul.

De nos jours, plusieurs compagnies les utilisent. Par exemple, plusieurs réseaux sociaux les utilisent pour enregistrer toutes les actions que font leurs utilisateurs sur leur site, par exemple « aimer une page », « publier un commentaire », « devenir ami avec une personne », « assister à un évènement ». Ils font ensuite des analyses à partir des informations qu'ils ont ainsi obtenues. Ils peuvent ensuite vendre ces renseignements à des compagnies de marketing ou faire de la publicité davantage personnalisée pour les utilisateurs. Cependant, ce n'est pas leur seule utilité. Plusieurs compagnies utilisent aussi ces projets afin de pouvoir traiter une grande quantité d'information afin de faire des calculs et des analyses concernant, par exemple, la durée de vie d'équipement industriel.

1.2 Génomique

La génomique [1] est l'étude du génome. Le génome est une représentation de l'ensemble de nos chromosomes, 46 dans le cas de l'humain, qui eux contiennent nos gènes. Nos gènes définissent l'individu ainsi que toutes ces caractéristiques. Il y a quatre grands domaines de recherche en génomique qui permettent à l'humain d'en apprendre davantage sur la génétique et l'évolution.

Tout d'abord, il y a l'exploration des fonctions associées aux gènes. Par exemple, on peut savoir quel gène en particulier contient l'information sur la couleur des cheveux des êtres humains. Le deuxième domaine permet de confirmer que l'humain est bel et bien un descendant du singe, c'est ce qu'on appelle la reconstruction des arbres phylogénétiques des espèces vivantes. Le troisième domaine est l'analyse de l'histoire évolutive des êtres vivants en lien avec leur écosystème. Par exemple, c'est cette fonction qui a permis de déterminer que les personnes dont les ancêtres provenaient de régions plus ensoleillées avaient une couleur de peau plus foncée que ceux dont leurs ancêtres provenaient de régions moins ensoleillées. Finalement, le quatrième domaine, soit celui qui est le plus intéressant pour le projet, est la compréhension des maladies liées aux gènes.

1.3 Adamcloud

1.3.1 Qu'est-ce qu'Adamcloud

Adamcloud est un projet de génomique qui permet de faire le séquençage et de trouver la variance d'un génome humain dans le but de diagnostiquer des maladies chez des patients. Le projet utilise des technologies de mégadonnées pour effectuer ces tâches.

Le génome informatisé d'un humain peut occuper facilement 100 Go de mémoire sur un ordinateur. De cette espace, environ 98% de l'information est identique chez tous les êtres humains, peu importe leur ethnicité [2] Ce 98% coûte très cher en entreposage et rend l'analyse d'un génome beaucoup plus long et ardu. Le projet *Adamcloud* vise à améliorer cette situation.

Le projet *Adamcloud* utilise des outils développés principalement par le *AMPLab* de *l'University of California, Berkeley* soit *Snap*, *Adam* et *Avocado*, ainsi que des outils de mégadonnées, *Apache Spark* et *Apache Hadoop*. De plus, l'outil de virtualisation par conteneur docker est aussi utilisé afin qu'*Adamcloud* puisse fonctionner convenablement.

1.3.2 Snap

À partir d'un échantillon sanguin, une machine spécialisée est en mesure de retirer le génome d'un patient et de l'écrire en fichier *.FASTQ*. Cependant, la machine n'est pas en mesure d'extraire le génome dans le bon ordre, il s'y retrouve dans un ordre aléatoire.

C'est pour cette raison que *Snap* existe. En effet, *Snap* permet d'aligner le génome en se basant sur le génome de référence. Cependant, avant de faire l'alignement de ce génome, il faut préalablement avoir indexé le génome de référence. Le génome de référence est le génome qui a été choisi par les experts dans le domaine et qui est utilisé comme standard pour déterminer la variance du génome des individus. En effet, puisque 98% du génome humain est identique pour tous, le génome de chaque individu peut être comparé au génome de référence afin que la variance de chacun en ressorte. Ce faisant, moins de temps et d'espace de stockage sont utilisés par le 98% d'ADN qui ne constitue pas une donnée utile à des fins de recherches en génomique.

1.3.3 Adam

Adam, quant à lui, permet de transformer un génome aligné, par exemple le fichier .sam que *Snap* produit, en un format *.adam*, puis de le transférer sur *HDFS*. Cette étape est cruciale, car elle permet l'utilisation d'une grappe *Apache Hadoop* pour exécuter des tâches sur un génome. Cela permet de réduire considérablement le temps de calcul, car ces calculs sont exécutés sur plusieurs ordinateurs à la fois et ont donc plus de ressource sont mises à leur disposition.

1.3.4 Avocado

Avocado est la dernière étape pour déterminer la variance d'un génome. En effet, cette tâche utilise les technologies de mégadonnées telles *Apache Hadoop* et *Apache Spark* pour déterminer la variance entre le génome aligné et le génome de référence. Les outils de

mégadonnées sont très utiles pour cette étape, car ils permettent de distribuer la tâche entre plusieurs ordinateurs différents.

1.4 Problématique

Au début du projet, lorsqu'on voulait rouler l'environnement d'*Adamcloud*, plusieurs étapes étaient nécessaires. Tout d'abord, il fallait avoir de bonnes connaissances informatiques, car pour l'installation de l'environnement, il fallait en premier lieu se connecter sur chacune des machines où l'environnement allait s'exécuter. Une fois connecté sur ces machines, il fallait télécharger le référentiel *GitHub*[3] contenant tous les *Dockerfiles* ainsi que les scripts *bash*. Ensuite, il fallait exécuter des commandes « *docker build* » pour toutes les images nécessaires, puis exécuter le bon script selon l'environnement et finalement faire certaines modifications manuelles sur certains conteneurs. De plus, il était impossible de faire ces actions sans utiliser un terminal et pour utiliser un terminal, il faut avoir un bon bagage informatique ou une volonté d'acier!

Par la suite, lorsque des modifications devaient être faites dans les *Dockerfiles* déjà existantes ou lorsqu'un transfert de connaissances sur les *Dockerfiles* devait être fait, c'était très compliqué, car peu de documentation concernant les *Dockerfiles* existe. De plus, le fonctionnement était plutôt compliqué à comprendre pour quelqu'un connaissant peu *docker*.

1.5 Objectifs

L'objectif établit dans le cadre de ce projet était de faciliter l'utilisation de l'ensemble de l'environnement pour une personne avec peu ou aucune formation en informatique. Afin d'accomplir cet objectif, deux fonctionnalités importantes étaient nécessaires.

Tout d'abord, il fallait que le projet ait une grande portabilité, c'est-à-dire qu'il peut fonctionner, peu importe l'environnement de travail, que ce soit sur un ordinateur personnel, une grappe d'ordinateur, un superordinateur ou bien sur le nuage informatique, par exemple avec *AWS*.

Deuxièmement, la facilité d'utilisation de l'environnement était aussi requise. L'exécution du moins de commandes possible pour lancer l'environnement complet et pour lancer les tâches de variance est nécessaire pour permettre au plus grand nombre de personnes d'utiliser le projet.

CHAPITRE 2

LES SCRIPTS D'ORCHESTRATION

2.1 Les scripts python

Afin de réduire la complexité de l'installation de l'environnement de travail d'*Adamcloud*, il a été convenu de faire des scripts facilement utilisables qui permettraient l'installation de tout l'environnement, et ce, peu importe l'architecture utilisée, que ce soit sur un seul ordinateur, une grappe d'ordinateur, un superordinateur ou bien sur le nuage informatique.

L'idée de départ était de modifier les scripts *bash* fait lors de la première étape du projet *Adamcloud* afin qu'il n'y ait qu'un seul script qui supporte tous les environnements. Après peu de temps, il est devenu clair qu'il était beaucoup plus compliqué de modifier les scripts *bash* pour nos besoins que de refaire les scripts en *Python*, un langage supportant plus facilement les fonctionnalités qui nous étaient nécessaires, comme la validation d'adresse IP et l'exécution de commandes sur des hôtes distants. C'est la principale raison expliquant la création des scripts en *Python* plutôt que la modification des scripts *bash* déjà existants. Étant donné que la version 2.7 de *Python* semblait être bien supportée par la communauté informatique et qu'elle était la version par défaut sur *Ubuntu 14.04*, c'est la version qui a été choisie pour faire les scripts.

Au début, il n'y avait qu'un seul script pour l'installation de l'ensemble de l'architecture. Cependant, après peu de temps, le script a été divisé en deux parties : *Orchestration.py*, permettant l'installation de l'environnement pour *Apache Hadoop* et *Apache Spark*, ainsi qu'*Adamcloud.py*, permettant l'installation de l'environnement pour les outils de génomique, soit *Snap*, *Adam* et *Avocado*. Le script a été divisé principalement pour des raisons de couplage. Ainsi, une personne voulant seulement utiliser *Apache Hadoop* et *Apache Spark* peut utiliser le script *Orchestration.py* sans être prise avec une des commandes pour l'installation des outils de génomique.

2.2 Les progiciels de pythons

Afin d'effectuer les opérations plus facilement avec les scripts, des tests avec le progiciel *docker-py*[4] ont été effectués. Cependant, ces tests n'ont pas été concluants. Bien que le progiciel permette d'utiliser les fonctionnalités de *docker*, il ne supportait pas les commandes de base en *bash* qui étaient essentielles pour l'installation de l'environnement. De surcroît, il n'ajoutait pas vraiment de valeur de plus qu'un progiciel capable d'envoyer des commandes *bash*. Donc, le progiciel *docker-py* n'a pas été utilisé puisqu'il aurait aussi fallu un progiciel pour envoyer des commande *bash*, et après un peu de recherche, c'est plutôt le progiciel *paramiko*[5], qui permet d'envoyer des commandes via *SSH*, qui a été utilisé, puisqu'il semblait bien fonctionner et permettait de remplir ces fonctions, soit envoyer des commandes via *SSH*, dont des commandes pour contrôler *docker*.

De plus, le module *getpass*[6] a aussi été nécessaire. Ce module permet d'entrer un mot de passe dans le terminal sans qu'il celui soit écrit directement à l'écran. Il était essentiel aux scripts, car lorsqu'on met en place l'environnement, certaines commandes nécessitent des droits d'administrateurs et le mot de passe est donc demandé.

2.3 Orchestration.py

Le premier script python qui a été fait pour le projet, *Orchestration.py*, voir l'Annexe II, permet l'installation d'un environnement *Apache Hadoop* et *Apache Spark* sur l'architecture désirée. Il y a trois principales commandes dans ce script afin d'installer l'environnement, **hdfsformat**, **hdfsrun** et **spark**. Pour exécuter le script, il faut spécifier le nom du script, l'environnement où on veut l'installer, la commande à exécuter, puis les paramètres de cette commande. Si le script est mal exécuté, une description de comment l'exécuter correctement apparaît.

2.3.1 hdfsformat

La commande **hdfsformat** lance la commande « *hdfs namenode -format* », qui permet de formater un environnement pour accueillir un *namenode*, dans un conteneur *docker* d'*Apache Hadoop*. La commande prend en paramètre seulement l'adresse IP de la machine hôte de *docker* où on veut installer le conteneur du *namenode* de *Apache Hadoop*.
Exemple : **orchestration.py macmini hdfsformat 192.168.1.100**

2.3.2 hdfsrun

La commande **hdfsrun** crée des conteneurs *Apache Hadoop* pour le *namenode*, le *secondarynamenode* et pour un ou plusieurs *datanodes*. La commande prend en paramètre les adresses IP des machines hôtes de *docker* où on veut installer les conteneurs selon l'ordre suivant, IP-namenode, IP-secondarynamenode, IP-datanode1, IP-datanode2.
Exemple : **orchestration.py macmini hdfsrun 192.168.1.100 192.168.1.100 192.168.1.101 192.168.1.102**
Cette commande installera le conteneur du *namenode* et du *secondarynamenode* sur l'instance de *docker* de la machine hôte qui se trouve à l'adresse IP 192.168.1.100, puis installera deux *datanodes*, un sur l'instance de *docker* de la machine hôte qui se trouve à l'adresse IP 192.168.1.101 et l'autre sur l'instance de *docker* de la machine hôte qui se trouve à l'adresse IP 192.168.1.102

2.3.3 spark

La commande **spark** fonctionne relativement comme la commande **hdfsrun,** mais pour les conteneurs d'*Apache Spark*. Elle crée des conteneurs pour le *spark-master* ainsi que les *spark-workers.* La commande prend en paramètre les adresses IP des machines hôtes de *docker* où on veut installer les conteneurs selon l'ordre suivant, IP-spark-master, IP-spark-worker1, IP-sparkworker2.
Exemple : **orchestration.py macmini spark 192.168.1.100 192.168.1.101 192.168.1.102**

Cette commande installera le conteneur du *spark-master* sur l'instance de *docker* de la machine hôte qui se trouve à l'adresse IP 192.168.1.100, puis installera deux *spark-workers*, un sur l'instance de *docker* de la machine hôte qui se trouve à l'adresse IP 192.168.1.101 et l'autre sur l'instance de *docker* de la machine hôte qui se trouve à l'adresse IP 192.168.1.102

2.4 Adamcloud.py

Le script *Adamcloud.py*, voir l'Annexe III, permet d'installer et d'exécuter des tâches pour les outils de génomique *Snap*, *Adam* et *Avocado*. Il y a quatre principales commandes dans ce script: **snapIndex**, **snapAlign**, **adam** et **avocado**. Pour exécuter le script, il faut spécifier le nom du script, l'environnement où on veut l'installer, la commande à exécuter, puis les paramètres de cette commande. Si le script est mal exécuté, une description expliquant comment l'exécuter correctement apparaîtra.

2.4.1 snapIndex

La commande **snapIndex** permet d'indexer le génome de référence afin de pouvoir l'utiliser pour faire l'alignement du génome. Elle crée temporairement un conteneur s*nap* et prend en paramètre le fichier du génome de référence et le nom du fichier où déposer le génome de référence indexé.
Exemple: **adamcloud.py snapIndex chr1.fa snap-index.chr1**
La commande indexera le génome de référence *chr1.fa* et déposera l'index dans le fichier *snap-index.chr1*.

2.4.2 snapAlign

La commande **snapAlign** permet d'aligner les acides désoxyribonucléiques d'une séquence d'ADN d'un génome généré par une machine spécialisée avec l'aide du génome de référence pour reconstruire un génome humain. La commande crée temporairement un conteneur *snap* et prend en paramètre le génome à aligner, le génome de référence indexé, avec **snapIndex**, et le nom du fichier du génome aligné, le fichier .*sam*.

Exemple : **adamcloud.py snapAlign SRR062634.fastq snap-index.chr1 SRR062634.sam**
La commande alignera le génome *SRR062634.fastq* avec le génome de référence indexée *snap-index.chr1* et déposera le génome aligné dans le fichier *SRR062634.sam.*

2.4.3 adam

La commande **adam** permet de transformer un génome aligné en un format qui lui permet d'être calculé sous HDFS de façon distribuée. La commande crée temporairement un conteneur *adam* et prend en paramètre les adresses IP des conteneurs du *spark-master* et du *hdfs*-namenode ainsi que le fichier du génome aligné *.sam*, et le fichier du génome aligné sous *HDFS*, le *.adam*.
Exemple : **adamcloud.py adam 192.168.1.100 192.168.1.100 SRR062634.sam SRR062634.adam**
La commande déposera le génome aligné *SRR062634.sam* sur le *HDFS* dont le *namenode* et le *spark-master* se situent à l'adresse IP 192.168.1.100 et le transformera ensuite en format pour *HDFS* dans le fichier *SRR062634.adam.*

2.4.4 avocado

La commande **avocado** permet de trouver la variance d'un génome humain par rapport au génome de référence. La commande crée temporairement un conteneur *avocado* et prend en paramètre les adresses IP des conteneurs du *spark-master* et du *hfds-namenode,* ainsi que le génome de référence, le fichier *.adam* qui contient le génome aligné sous *HDFS* dans un format distribué et finalement le dossier *avocado.*
Exemple : **adamcloud.py avocado 192.168.1.100 192.168.1.100 chr1.fa SRR062634.adam SRR062634.avocado.adam**
La commande trouvera la variance du génome aligné *SRR062634.adam* distribué sur le *HDFS* dont le *namenode* et le *spark-master* sont à l'adresse 192.168.1.100, à l'aide du génome de référence *chr1.fa* et déposera le fichier de variance dans le fichier *SRR062634.avocado.adam.*

CHAPITRE 3

DOCKER

3.1 Qu'est-ce que Docker?

Docker[7] est une alternative à la virtualisation traditionnelle. La principale différence entre *docker* et les VM traditionnelles réside dans le fait que docker utilise les *cgroups* et les *namespace* du *kernel linux* pour isoler les conteneurs et partager les ressources systèmes, plutôt que de complètement dupliquer le système d'exploitation. *Docker* isole donc les conteneurs avec des *namespaces* et donne accès aux ressources du système avec les *cgroups*.

3.2 Dockerfiles

Pour construire un conteneur, les *dockerfiles* sont utilisés. Il s'agit d'un fichier avec une série d'instructions qu'on appelle *Layers* qui décrivent comment le conteneur est créé. Voir l'Annexe IV pour les *dockerfiles* qui sont actuellement utilisés pour nos images *docker*. Pour plus d'informations sur le sujet, je vous conseille fortement de lire la documentation de *docker* à ce sujet[8].

Un *dockerfiles* commence toujours par l'instruction ***FROM*** qui précise sur quel docker image ce *dockerfiles* est basé. Par la suite, plusieurs instructions différentes peuvent être spécifiées. Parmi celles-ci, il y a ***ENV*** qui permet de créer une variable d'environnement, ***EXPOSE*** qui permet d'ouvrir un port sur le conteneur pour qu'une machine distante se connecte et puisse interagir avec ce qui se trouve sur le conteneur, ***RUN*** qui permet d'exécuter des commandes *bash* dans le conteneur, ***CMD*** qui permet d'avoir une commande par défaut qui sera exécutée lorsque le conteneur sera créé avec la commande « *docker run nom_de_l'image* ».

3.3 Les images et les conteneurs de docker

Le premier concept qu'il faut connaître dans docker est la différence entre une image *docker* et un conteneur. L'image *docker* est ce qui décrit un conteneur, il s'agit d'une série de *layers* qui sont décrits dans le *dockerfile*. Le conteneur peut être vu comme l'instance d'une image. Ainsi, plusieurs conteneurs identiques peuvent être créés à partir d'une même image, mais ceux-ci seront indépendants les uns des autres.

3.4 L'architecture des images

Avec *Docker*, toutes les images docker sont basées sur une autre image *docker*, définit par la commande « ***FROM*** ». De plus, la majorité de nos projets avaient les mêmes dépendances. Ce faisant, certaines images *docker* ont été créées uniquement pour être des images sur lesquelles les autres images *docker* se basent, plutôt que de dupliquer des commandes dans chacune des autres images. Par exemple, presque toutes les images ont besoin de *Java*, donc il y a une image *Java* qui a été créée. Voici une figure représentant l'architecture des images *docker* pour le projet *Adamcloud*.

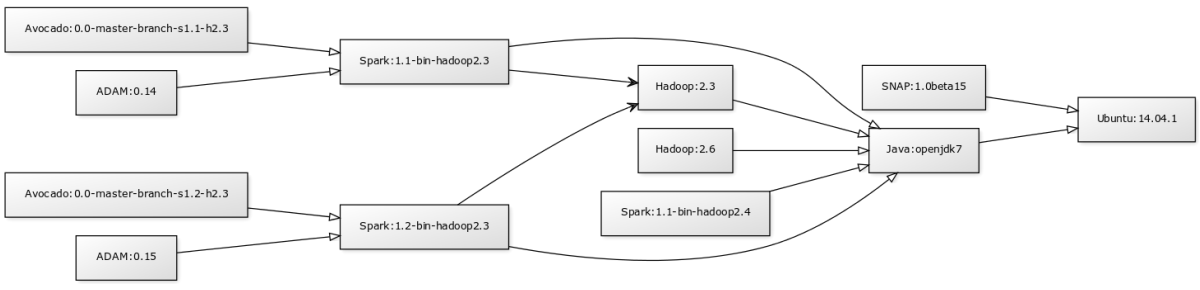

Figure 1 Arborescence des images docker

3.5 Docker Hub Registry

Pour construire une image de *Docker*, il faut télécharger son *dockerfile* et ensuite rouler la commander *« docker build »* pour créer l'image sur l'ordinateur hôte, ce qui permet par la suite de faire la commande « *docker run nom_de_l'image* » pour créer un conteneur.

Un des avantages du *Docker Hub Registry*, *DHR*, c'est qu'il nous permet de construire le conteneur directement avec « *docker run nom_de_l'image* », donc, il n'est plus nécessaire de télécharger le *dockerfile* et de faire un build. Le *DHR* permet ceci, car c'est un outil permettant d'avoir des référentiels ouverts, ou non, pour des *dockerfile*, c'est-à-dire qu'on enregistre une copie du *dockerfile* et par la suite *docker* est capable d'aller la récupérer par lui-même. Cependant, si on le désire, on peut aussi aller télécharger l'image sur l'hôte avec la commande « *docker pull nom_de_l'image* ». De plus, le *DHR* permet de synchroniser les référentiels avec un référentiel de *GitHub* pour permettre les « *Automated build* » des *dockerfiles*. Ainsi, dès qu'on publie une modification à notre *dockerfile* sur *GitHub*, un signal est envoyé au *DHR* afin que l'image soit reconstruite automatiquement. Une fois l'image reconstruite, la prochaine fois où la commande « *docker pull nom_de_l'image* » sera exécutée, la nouvelle version de l'image sera téléchargée. Si l'image n'a jamais été téléchargée sur l'ordinateur hôte en question, alors c'est l'image la plus récente qui le sera.

3.6 Documentation des *dockerfiles*

Étant donné que le *DHR* est à la base un référentiel ouvert et que tout le monde peut avoir accès à aux images s'y retrouvant et les utiliser, il était important d'avoir une documentation claire, concise et utile pour les images. C'est pour cette raison que des fichiers *README.md*, voir l'Annexe V, ont été ajoutés à tous les référentiels de *dockerfile* sur *GitHub*. Ces *README.md* contiennent toutes les mêmes sections afin qu'on retrouve les mêmes informations pour chaque image. Les sections sont :

- Les versions disponibles
- Quel est le projet sur l'image, par exemple « Qu'est-ce qu'Avocado »

- Qu'est-ce docker
- Les dépendances au projet
- L'image docker de base
- Comment utiliser l'image

3.7 Modification des *dockerfiles*

Au cours du projet, la modification de certains *dockerfiles* était nécessaire. Tout d'abord, l'image de *java* a été changée pour *openJDK7* plutôt qu'*oracleJDK7*. La version *oracleJDK7* avait été choisie comparativement à *openJDK7*, car dans les versions 6, la version d'*Oracle* était beaucoup plus stable. Cependant, pour les versions 7, la différence entre *Oracle* et *Open* n'était plus aussi grande, et étant donné que les projets de génomique du *AMPLab* utilisent la version *openJDK7*, la version de *java* pour le projet *Adamcloud* est passée à *openJDK7*. Suite à cette modification, les images utilisant l'image de *java* ont aussi dû être mise à jour.

Par la suite, l'image d'*Apache Spark* a aussi eu une mise à jour afin d'avoir une version plus récente, soit passer de la version 1.1.0 à 1.2.1. Étant donné que les images d'*Adam* et d*'Avocado* se basent sur l'image d*'Apache Spark*, celles-ci ont aussi dû être reconstruites.

Finalement, l'image d'*Adam* a aussi eu une mise à jour de la version 0.14.0 à 0.15.0. La principale raison de cette mise à jour est que la version 0.15.0 supportait l'utilisation d'un environnement *Apache Spark* externe.

CHAPITRE 4

OUTILS DE DOCKER

4.1 docker-compose

Docker-compose[9] est un outil sous la tutelle de *docker*. Anciennement, le projet s'appelait *fig*, mais il a été intégré comme outil officiel de *docker*. Cet outil permet d'utiliser un fichier de configuration afin de créer un environnement complet. Par exemple, si on désire un serveur web avec une base de données *MySQL*, on peut inscrire dans le fichier de configuration qu'on désire un conteneur pour *Tomcat*, un conteneur pour *MySQL* et qu'un lien entre les deux conteneurs soit créé, afin que *Tomcat* puisse accéder à la base de données. Par la suite, avec seulement une commande « *docker-compose up* », on peut créer l'environnement. De plus, il y a plusieurs autres fonctionnalités telles que « *docker-compose* », qui permettent l'extensibilité d'un conteneur. Par exemple, la commande « *docker-compose scale tomcat=2* » permettrait de créer un deuxième conteneur *Tomcat* en plus du premier, afin de faire de l'équilibrage de charges entre ces deux conteneurs.

4.2 docker-swarm

Docker-swarm[10] est aussi un outil sous la tutelle de *docker*. Cet outil permet de contrôler plusieurs instances de *docker* installées sur des hôtes différents, comme s'il ne s'agissait que d'une seule instance *docker*. Par exemple, le projet *Adamcloud* a besoin d'être fonctionnel, peu importe l'environnement. Dans un environnement comprenant plusieurs machines physiques, l'utilisation de *docker-swarm* permettrait d'alléger les configurations et les erreurs potentielles, car *docker* se croit sur une seule machine physique plutôt que sur plusieurs. De plus, cela permettrait aussi de ne pas avoir de problème de connexion entre les conteneurs qui se trouvent sur des hôtes différents. Donc, pour le projet, lorsque nous utiliserons des environnements distribués sur plusieurs hôtes physiques, il serait plus simple d'installer *docker-swarm* sur ces hôtes, plutôt que d'avoir à faire d'autres configurations manuelles afin que les conteneurs sur les différentes machines physiques puissent communiquer entre eux.

4.3 docker-machine

Docker-machine[11] est un outil sous la tutelle de *docker* qui a pour but de faciliter la configuration d'un environnement *docker* installé sous d'autres services. Par exemple, il permet de contrôler des instances de *docker* hébergées avec l'*AWS*, *DO* ou une instance à l'intérieur d'une *VM VirtualBox*[12].

4.4 Utiliser les outils ensemble

Ces trois outils sont tous compatibles entre eux[13], ce qui constitue leur fonctionnalité la plus intéressante. On peut donc utiliser *docker*-compose sur une *swarm* dont un ou plusieurs des nœuds est connecté via *docker-machine* sur *AWS* ou *DO* et dont d'autres nœuds peuvent être connectés localement. De plus, *docker-swarm* permet d'assigner des *Constraints* ou des *Affinities* sur les conteneurs[14]. Il pourrait y avoir des règles dans le *docker-compose* pour définir, par exemple, qu'un conteneur en particulier doit être sur un ordinateur avec un *SSD*, qu'il se situe sur un ordinateur de la côte Est américaine ou qu'il ne doit pas être sur la même machine physique qu'un autre *datanode*.

4.5 Les outils *docker* avec *Adamcloud*

Malheureusement, ces outils n'ont pas été implémentés dans le projet *Adamcloud*, principalement parce que le projet a besoin de beaucoup trop de configurations manuelles qui ne s'intègrent pas directement dans *docker-compose*. Pour régler ce problème, il serait possible de modifier ou créer de nouvelles images docker qui serviraient seulement de configuration. Par exemple, actuellement, il faut télécharger les bons fichiers d'environnement pour la configuration d'*Apache Hadoop*, ce qui est problématique. Par contre, il serait possible de créer une image qui exécute un script permettant de télécharger les configurations voulues dans un *docker-volume* pouvant être utilisé par les conteneurs d'*Apache Hadoop*.

CHAPITRE 5

PROBLÈMES RENCONTRÉS

5.1 Problèmes techniques

Les technologies utilisées pour ce projet sont toutes relativement nouvelles et peu matures. La première version d'*Apache Hadoop* date de décembre 2011[15], celle de *docker* de mars 2013[16] et le premier « *commit* » du projet *Adam* date d'avril 2013[17]. En informatique, les bogues sont communs dans tous les projets, on doit apprendre à vivre avec cela. Cependant, en mélangeant plusieurs projets immatures, on ne sait pas nécessairement d'où le bogue provient. Est-ce un bogue avec le projet 1? Est-ce un bogue avec le projet 2? Est-ce une mauvaise configuration? Est-ce un bogue causé par l'utilisation du projet 1 avec le projet 2? Plusieurs questions légitimes se posent et plus le nombre de projets immatures intégrés dans l'environnement est grand, plus le projet est complexe.

5.2 Problèmes organisationnels

La gestion des problèmes fut ma grande faiblesse au cours de ce projet. Lorsqu'un bogue prenait beaucoup de mon temps, je contactais mon superviseur pour savoir si mon problème lui sonnait une cloche. Cependant, au lieu d'attendre une réponse, je portais mon attention sur une autre section du projet afin de ne pas être bloqué et perdre trop de temps. À première vue, ça me semblait une bonne idée, cependant, elle a joué contre moi. Comme de fait, étant donné que j'étais concentré sur une autre partie du projet, je ne regardais plus les réponses qui m'étaient envoyées ou des questionnements pour recréer le problème que j'avais eu. Tout ça a fait perdre du temps à l'équipe, car j'ai mal fait le suivi de mes problèmes.

CHAPITRE 6

TRAVAUX FUTURS

6.1 À court terme

Pour les travaux futurs à court terme, il serait important de faire des tests de performance. Des tests de performance avaient été réalisés lors de la première partir du projet, mais malheureusement aucun n'a été fait pour cette partie. De plus, ces tests pourraient permettre d'avoir un tableau représentant le facteur de réplication des données de la grappe par rapport au nombre d'ordinateurs physiques disponible afin d'ajuster la grappe de *Apache Hadoop* pour améliorer les performances.

L'optimisation des images *docker* pourrait aussi être un objectif important à court terme. L'avantage en réduisant les images est principalement l'amélioration du temps de création de l'environnement. Plus l'image est petite, plus il sera rapide de la télécharger, et donc du temps sera sauvé lors de la création de l'environnement. De plus, une image plus petite est mieux conçue et plus simple à comprendre, c'est un facteur souvent recherché dans la communauté du code source libre.

L'utilisation des outils de la communauté *docker*, tel *docker-compose* et *docker-swarm*, serait judicieuse, puisqu'ils nous obligeraient à avoir des images *docker* selon les standards. De plus, il n'y aurait plus de script à maintenir étant donné que la communauté *docker* est déjà derrière ceux-ci. L'utilisation de ces outils permettrait d'enlever un degré de complexité au projet *Adamcloud*, puisqu'ils sont déjà compatibles avec *docker* et plusieurs les utilisent tous les jours et donc les mettent au défi.

Finalement, le problème principal que j'ai détecté avec *docker* est le lien existant entre les conteneurs. *Docker* est fait pour avoir un conteneur X qui connaît un conteneur Y et d'interagit avec lui. Cependant, le conteneur Y, bien qu'ils puissent répondre aux requêtes

envoyées par X, ne connaît pas l'existence de ce dernier. Dans une très grande partie des situations, il n'y a pas de problème. Par exemple, pour un serveur web avec une base de données, le conteneur de la base de données n'a pas besoin de connaître le serveur web. Cependant, dans d'autres situations, comme la nôtre, les conteneurs ont besoin de connaître les autres conteneurs. Par exemple, les *spark-worker* et les *datanodes* ont besoin de connaître les conteneurs *Adam et Avocado* qui leur demandent du travail. Pour cette raison, il faut ajouter manuellement les conteneurs des outils de génomique dans les *spark-workers* et les *datanodes*. Une solution possible pour pallier à ce problème serait peut-être l'utilisation d'un conteneur contenant un serveur *DNS*. Par la suite, tous les conteneurs pourraient s'enregistrer sur lui et comme cela, chaque conteneur pourrait connaître l'existence des autres conteneurs et interagir avec eux si cela s'avère nécessaire. Aucun test n'a été fait avec des *DNS*, mais il s'agirait d'une bonne piste à explorer qui pourrait sans doute régler des problèmes.

6.2 À moyen terme

À moyen terme, il serait sans doute nécessaire de faire des tests avec des services d'hébergement sur le nuage, par exemple avec *AWS*. Ces services permettent de louer des serveurs avec une tarification à l'heure et supportent *docker*. De plus, le projet *1000 Genomes*[18] est hébergé sur *AWS*, donc lors du téléchargement d'un génome pour effectuer une tâche, on sauverait beaucoup de temps puisque la connexion serait locale.

Un autre objectif à moyen terme serait d'effectuer la validation des résultats. Il faudrait valider l'ensemble de la transformation d'un génome avec le pipeline du *AMPLab*. Il est impossible pour nous de le valider, mais des généticiens seraient en mesure de le faire.

6.3 À long terme

À long terme, il serait possible de faire rouler le projet dans des hôpitaux afin de faire des tests avec de vrais patients et aussi pour avoir des diagnostics plus rapides de leur examen.

L'installation de l'environnement dans un cadre réel serait sans aucun doute très intéressante, surtout si on améliore la situation de certains patients en ayant un diagnostic plus rapide.

CONCLUSION

Le projet *Adamcloud* est un projet de grande envergure. Ce rapport représente seulement la partie II du projet et je ne suis pas convaincu que la moitié du développement du projet a été atteinte. Il s'agit d'un projet de grande ampleur qui utilise plusieurs technologies non matures, dont *Apache Hadoop, Apache Spark, docker*, *Snap, Adam* et *Avocado*, et cela donne une immense courbe d'apprentissage pour les personnes travaillant sur le projet. De plus, étant donné qu'il s'agit de PFE, ces personnes ont seulement quatre mois pour monter cette immense courbe et faire leur projet.

Ceci dit, le projet a une importance cruciale et pourrait, à un certain point, aider à sauver des vies et faire avancer la médecine. Par exemple, le séquençage de génome sur *AWS* permettrait d'obtenir des résultats plus rapidement, car les coûts d'utilisation d'*AWS* seraient moindres, puisque l'utilisation de leurs serveurs peut être facturée à l'heure plutôt qu'au mois. Donc, si on a simplement besoin des serveurs pour quelques heures, seulement ces heures seront facturées. Ces coûts en moins pourraient permettre l'utilisation de plus de serveurs pour réduire encore plus le temps des tâches de séquençages. De plus, puisque c'est simplement la variance qui serait sauvegardée, il serait possible de créer une grande base de données recueillant les variances des génomes. Il pourrait ensuite être possible de faire des recherches et possiblement pouvoir trouver des liens entre certains gènes et certaines maladies par exemple.

RECOMMANDATIONS

Les outils *docker-swarm* et *docker-compose* sont très puissants et risquent de le devenir encore davantage dans les années à venir avec tout le mouvement de la communauté informatique qui se dirige vers l'utilisation de *docker* au dépens des outils de virtualisation traditionnels. Les outils de docker pourraient éventuellement permettre de facilement construire l'environnement complet d'*Adamcloud*. C'est pourquoi je crois que l'utilisation de ces outils serait essentielle dans le projet Adamcloud et devrait y être intégrée assez rapidement.

LISTE DE RÉFÉRENCES

[1] **Wikipedia**, Génomique
Internet, consulté le 22 janvier 2015.
http://fr.wikipedia.org/wiki/G%C3%A9nomique

[2] **Live science**, Gorillas & Humans closer than thought, genome sequencing reveals
Internet, consulté le 7 avril 2015.
http://www.livescience.com/18892-gorillas-humans-gene-sequence.html

[3] **GitHub**, GELOG/adamcloud
Internet, consulté le 5 marsl 2015.
https://github.com/GELOG/adamcloud

[4] **GitHub**, docker/docker-py
Internet, consulté le 20 mars 2015.
https://github.com/docker/docker-py

[5] **Paramiko**, Welcome to Paramiko!
Internet, consulté le 20 mars 2015.
http://www.paramiko.org/

[6] **Python documentation**, 15.10. getpass – Portable password input
Internet, consulté le 20 mars 2015.
https://docs.python.org/2/library/getpass.html#module-getpass

[7] **Docker**, Build, Ship and Run Any App, Anywhere
Internet, consulté le 20 mars 2015.
https://www.docker.com/

[8] **Docker**, Dockerfile Reference
Internet, consulté le 5 avril 2015.
https://docs.docker.com/reference/builder/

[9] **Docker**, Docker Compose
Internet, consulté le 5 avril 2015.
https://docs.docker.com/compose/

[10] **Docker**, Docker Swarm
Internet, consulté le 5 avril 2015.
https://docs.docker.com/swarm/

[11] **Docker**, Docker Machine
Internet, consulté le 5 avril 2015.
https://docs.docker.com/machine/

[12] **VirtualBox**, Oracle VM VirtualBox
Internet, consulté le 6 avril 2015.
https://www.virtualbox.org/

[13] **Docker**, ORCHESTRATING DOCKER WITH MACHINE, SWARM AND COMPOSE
Internet, consulté le 5 avril 2015.
https://blog.docker.com/2015/02/orchestrating-docker-with-machine-swarm-and-compose/

[14] **Docker**, SCALING DOCKER WITH SWARM
Internet, consulté le 5 avril 2015.
https://blog.docker.com/2015/02/scaling-docker-with-swarm/

[15] **Wikipedia**, Apache Hadoop
Internet, consulté le 10 avril 2015.
http://en.wikipedia.org/wiki/Apache_Hadoop

[16] **Wikipedia**, Docker (software)
Internet, consulté le 10 avril 2015.
http://en.wikipedia.org/wiki/Docker_%28software%29

[17] **GitHub**, Initial commit - bigdatagenomics/adam@e9d835d
Internet, consulté le 10 avril 2015.
https://github.com/bigdatagenomics/adam/commit/e9d835d14ba72d1950006a2c28eafd58ac9e94d4

[18] **1000 Genomes**, Home | 1000 Genomes
Internet, consulté le 10 avril 2015.
http://www.1000genomes.org/

BIBLIOGRAPHIE

Wikipedia, *Docker (software)*
Internet, consulté le 10 avril 2015.
http://en.wikipedia.org/wiki/Docker_%28software%29

Wikipedia, *Big data*
Internet, consulté le 10 avril 2015.
http://fr.wikipedia.org/wiki/Big_data

AMPLab – UC Berkeley
Internet, consulté le 12 avril 2015.
https://amplab.cs.berkeley.edu/

Computer Science Division – UC Berkeley, SNAP Sequence Aligner
Internet, consulté le 12 avril 2015.
http://snap.cs.berkeley.edu/

Big Data Genomics, ADAM
Internet, consulté le 12 avril 2015.
http://bdgenomics.org/projects/adam/

Big Data Genomics, Avocado
Internet, consulté le 12 avril 2015.
http://bdgenomics.org/projects/avocado/

Spark, Overview
Internet, consulté le 20 avril 2015.
https://spark.apache.org/docs/latest/

Apache Hadoop, Welcome to Apache Hadoop!
Internet, consulté le 12 avril 2015.
http://hadoop.apache.org/

Google Drive, ADAMCLOUD UNE INFRASTRUCTURE PORTABLE POUR LE TRAITEMENT DE DONNÉES GÉNOMIQUE
Internet, consulté le 12 avril 2015.
https://drive.google.com/file/d/0BzArSBBvDRxDcWJkSmtzUnozSkk/view?usp=sharing

GitHub, Build software better, together.
Internet, consulté le 12 avril 2015.
https://github.com/

Amazon Web Services (AWS), Cloud Computing Services
Internet, consulté le 12 avril 2015.
http://aws.amazon.com/

DigitalOcean, SSD Cloud Server, VPS server, Simple Cloud Hosting | DigitalOcean
Internet, consulté le 12 avril 2015.
https://www.digitalocean.com/

Python.org, Welcome to Python.org
Internet, consulté le 12 avril 2015.
https://www.python.org/

Ubuntu, The leading OS for PC, tablet, phone and cloud | Ubuntu
Internet, consulté le 12 avril 2015.
http://www.ubuntu.com/

Docker Documentation, Container
Internet, consulté le 12 avril 2015.
http://docs.docker.com/terms/container/

Docker Documentation, Image
Internet, consulté le 12 avril 2015.
http://docs.docker.com/terms/image/

Docker Hub Registry, Repositories of Docker Images
Internet, consulté le 12 avril 2015.
https://registry.hub.docker.com/

ANNEXE I

COMPTES RENDUS HEBDOMADAIRES

Lundi 19 janvier 2015

Ce qui a été fait

- Lecture du PFE de Sébatien Bonami
-

Élément(s) bloquant(s)

- *Aucun*

Plan d'action

-

Lundi 26 janvier 2015

Ce qui a été fait

- Installation de l'environnement sur mon ordinateur personnel (Ubuntu, docker, ...)
- Installation du cluster fait par Sebastien Bonami pour un environnement pseudo-distribue
-

Élément(s) bloquant(s)

- Dans le DockerFile de snap-adam-avodaco-spark, le path pour télécharger maven de l'université Dalhousie retourne un 404…
    - J'ai fait un fix temporaire, je l'ai forcé à utiliser la version 3.2.5 à la place de 3.2.3
-

Plan d'action

-

Lundi 9 février 2015

Ce qui a été fait

- Installation des MAC-minis
- Configuration des MAC-minis pour y avoir accès plus facilement et des configurations
    - Création de clés publiques-privé et copie de celles-ci dans les autre MAC-minis (Plus besoin de mot de passe pour ce connecter d'un MAC à un autre)
    - Ajout des MAC-minis dans leurs hosts files.
    - Installation de vim pour éditer le texte.
    - Téléchargement des répertoires de GitHub dans les Mac mini
- Tester le pipeline en environnement pseudo-distribué sur mon laptop personnel

Élément(s) bloquant(s)

- Certains MAC-mini ont un problème au démarrage.
    - J'ai réussi à les démarrer en les partant avec les options avancées et en les démarrant selon les configurations de la dernière fois qu'il avait marché.
- Certains MAC-mini n'étaient pas capables de ce connecter sur le réseau.
    - Après quelque recherche Google, j'ai remarqué qu'ils avaient 2 interfaces réseau en conflit (eth0 et eth1) car ils essayaient d'utiliser le même MAC Address. J'ai déactivé eth1.
- Lorsque j'ai testé le pipeline sur l'environnement pseudo-distribué, ça n'a pas fonctionné par un manque de mémoire vive.

Plan d'action

- Je vais tester le pipeline sur l'architecture distribué des MAC-minis

Lundi 23 février 2015

Ce qui a été fait

- Modification des scripts dans Adamcloud pour le nouvel environnement
- Création des docker repos dans le docker hub registry

    - Snap
    - Adam
    - Avocado
    - java
    - spark
    - hadoop
- Tester un chromosome avec l'architecture complète

Élément(s) bloquant(s)

- Un problème avec les conteneurs de weave
    - Lors que le script de création des conteneurs roulait, les conteneurs weave s'arrêtaient presque automatiquement…
    - C'est probablement en effaçant tous les conteneurs et images sur les machines que j'ai causé le problème.
    - J'ai réinstallé weave (même si je ne l'avais pas désinstaller) en suivant les instructions dans le rapport de Sébastien puis la fois suivante quand j'ai essayé de roule le script pour créer les conteneurs, l'image de weavetool été téléchargée.
- Corruption d'un Mac mini 1
    - En essayant de faire un sauvegarde du répertoire /docker-volume (afin de le vider), j'ai fait la commande **mv *** en passant être dans le répertoire /docker-volume, mais j'étais dans /
    - J'ai perdu environ 5 heures à régler le problème (qui n'est toujours pas réglé…)
- Problème de l'indexation de snap
    - Lorsque j'essaie d'indexer un chromosome avec snap, ça ne fonctionne pas, le conteneur est arrêté et ne veut pas partir...

Plan d'action

- Pour le Mac mini, je vais voir demain avec David

- Pour snap, je vais essayer de partir un conteneur avec l'image de base de snap et de refaire manuellement les opérations dans le docker file.
- Lorsque je vais avoir réussi à tester l'architecture complète, je vais compléter les milestone 2 et 3 qui sont sur Github.

Lundi 2 mars 2015

Ce qui a été fait

- Le milestone 0.2
    - Issue #1 **Publish images on the Docker Hub Registry**
    - Issue #2 **Produce a clear and concise documentation for each Dockerfile**
    - Issue #3 **Show the Dockerfile for each released version of the image**

Élément(s) bloquant(s)

-

Plan d'action

-

Lundi 9 mars 2015

Ce qui a été fait

- Réparer le Mac mini 1 dont j'avais corrompu il y a 2 semaines.
- Dans le milestone 0.3
    - Issue #10: **Optimize orchestration scripts**
        - Beaucoup de recherche sur docker swarm
        - Configuration du cluster de Mac mini pour le mettre dans le swarm
        - Commencement du docker-compose.yml
        -
    -

Élément(s) bloquant(s)

- Ce qu'on avait présentement utilisation weave afin que les conteneurs puissent communiquer entre eux s'il était sur des machines physiques différentes. Lorsqu'on

veut utiliser weave avec les conteneurs, il faut utiliser la commande ***weave*** à la place de la commande ***docker***. Docker-compose (anciennement connu sous le nom de fig) utilise la commande ***docker*** afin de créer facilement un environnement de conteneur docker, il est donc impossible d'utiliser weave avec docker-compose.

- La complexité de la configuration du fichier docker-compose selon les scripts de création déjà existant.

Plan d'action

- Pour régler le problème de weave et docker-compose, on peut utiliser docker-swarm qui permet de contrôler un cluster de docker comme une seule instance.

Lundi 16 mars 2015

Ce qui a été fait

- Dans le milestone 0.3
    - Issue #10: **Optimize orchestration scripts**
        - Recherche sur docker swarm
        - Faire fonctionner docker swarm avec le cluster de mac mini
        - Travail sur le script bash pour exécuter l'environnement
        -
    -

Élément(s) bloquant(s)

- La complexité du script bash avec les connexions au différent host

Plan d'action

- Faire le script en python qui contient déjà des outils pour communiquer avec docker et pour bash. Il contient probablement aussi des outils afin de travailler avec des hôtes distants. De plus, les commandes sont beaucoup plus simples et intuitives que du bash.

Lundi 23 mars 2015

Ce qui a été fait

- Dans le milestone 0.3
    - Issue #10: **Optimize orchestration scripts**
        - Script python pour exécuter l'environnement selon les paramètres reçus
            - Fonctionne pour snap
            - Fonctionne pour adam
    -

Élément(s) bloquant(s)

- docker-py, outils pour contrôler pi docker dans python, semble avoir des fonctionnalités intéressantes pour le projet, par exemple contrôlé un hôte docker à distance, mais semble être plus compliquer a utiliser que les commands lines
- Avocado ne semble pas fonctionner
    - Il y a des config particuliers à faire selon l'environnement et ça semble plutôt complexe

Plan d'action

- Faire fonctionnaire avocado
    - Suite aux problèmes, il serait peut-etre plus d'installer/faire un script wrapper dans le conteneur qui lui ferait les changements selon l'environnement et appellerai la tache de avocado
- Aucun test avec spark ou hadoop n'a été fait encore, il faudrait les faires pour pouvoir setuper l'environnement avec plusieurs ordinateurs
    - Pour hadoop, il faudra peut-être aussi avoir un genre de wrapper comme décrit si haut pour avocado...
- Voir si on utilise le docker-py ou non, si oui l;implémenter…
- optimiser le script, documenter et faire un readme
- upgrader l'image de spark a 1.3?

Lundi 30 mars 2015

Ce qui a été fait

- Dans le milestone 0.3
    - Issue #10: **Optimize orchestration scripts**
        - Script python pour lancer le cluster de hadoop et spark
            - hadoop marche

Élément(s) bloquant(s)

- L'exécution des commandes sur des hosts distants
    - Paramiko.py permet d'exécuter des commandes sur un host distant par SSH

Plan d'action

- Faire fonctionner spark, snap, adam, avocado

- Lundi 6 avril 2015

Ce qui a été fait

- Orchestration scripts
    - hadoop
        - format
        - setup
    - spark
        - run

Élément(s) bloquant(s)

- La transformation dans adam sur un cluster spark fait une erreur et la job est tue, mais les informations semble toutes être présente dans le dossier adam
- Les spark-workers semblent avoir besoin de connaitre adam lorsque la tache de adam Est lance (pareille pour avocado)

Plan d'action

Lundi 13 avril 2015

Ce qui a été fait

- Orchestration script
    - Spark
- Fichier expliquant comment rouler les taches snap adam et avocado
- Mise a jour des docker images de spark, adam et avocado pour les changements dans openjdk7
-

Élément(s) bloquant(s)

- Avocado ne semble pas fonctionner, en local ou en distribuer...

Plan d'action

-

Lundi 20 avril 2015

Ce qui a été fait

- Préparation du rapport
- Préparation de l'Oral

Élément(s) bloquant(s)

-

Plan d'action

ANNEXE II

ORCHESTRATION.PY

```
__author__ = 'flangelier'

import sys
import re
import paramiko
import getpass

adamcloud_base_path = 'https://raw.githubusercontent.com/GELOG/adamcloud/master/'
hadoop_host_volume_path = '/hdfs-data'
hadoop_image = 'gelog/hadoop'
hadoop_image_tag = '2.3.0'
spark_image = 'gelog/spark'
spark_image_tag = '1.2-bin-hadoop2.3'
sudo_passwords = {}


def is_valid_hostname(hosts):
    failed = []
    for host in hosts:
        # Validating only IPv4 because we need to add the IP in the containers hosts file
        if not re.compile('^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-
4][0-9]|[01]?[0-9][0-9]?)$').match(host):
            failed.append(host)
            continue
        # if len(host) > 255:
        #     return False
        # if host[-1] == ".":
        #     host = host[:-1]  # strip exactly one dot from the right, if present
        # allowed = re.compile("(?!-)[A-Z\d-]{1,63}(?<!-)$", re.IGNORECASE)
        # return all(allowed.match(x) for x in hostname.split("."))

    if len(failed) > 0:
        print "There aren't valid IPv4", failed
        return False
    return True


def execute_over_ssh(host, command):
    print 'host: %s cmd: %s' % (host, command)
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(host)
        chan = ssh.get_transport().open_session()
        if 'sudo' in command:
            chan.get_pty()
            chan.exec_command(command)
            received = chan.recv(1024)
            # If the password entered is bad, it will loop until the end of time...
            while '[sudo] password for adamcloud:' in received:
                password = sudo_passwords.get(host, None)
                if password is None:
                    print received
                    password = getpass.getpass()
                    sudo_passwords[host] = password
                stdin = chan.makefile('wb', -1)
                stdin.write(password + '\n')
                received = chan.recv(1024)
```

```
        else:
            chan.exec_command(command)
            received = chan.recv(1024)

        print(received)

        ssh.close()
    except paramiko.AuthenticationException as e:
        help('Connection to %s failed, make sure you pushed your public key on that host!' %
host)
        return False
    return True


def hdfs_configure(env, hosts):
    print 'Configuring hdfs for', hosts
    failed = []
    for host in hosts:
        command = 'sudo mkdir -p ' + hadoop_host_volume_path + '/conf'
        if not execute_over_ssh(host, command):
            failed.append(host)
            continue

        base_command = 'sudo wget
https://raw.githubusercontent.com/GELOG/adamcloud/{0}/env/{1}/{2} -O ' \
                       + hadoop_host_volume_path + '/conf/{2}; '
        hadoop_tag = hadoop_image_tag

        if not execute_over_ssh(host, base_command.format(hadoop_tag, env, 'core-site.xml')):
            failed.append(host)
            continue
        if not execute_over_ssh(host, base_command.format(hadoop_tag, env, 'yarn-site.xml')):
            failed.append(host)
            continue
        if not execute_over_ssh(host, base_command.format(hadoop_tag, env, 'mapred-
site.xml')):
            failed.append(host)
            continue
        if not execute_over_ssh(host, base_command.format(hadoop_tag, env, 'hdfs-site.xml')):
            failed.append(host)
            continue
    if len(failed) > 0:
        print 'Failed to configure hosts:', failed
        return False
    return True


def hdfs_format(env, host, alsoConfigure=True):
    print 'Formating HDFS on host', host

    if alsoConfigure and not hdfs_configure(env, [host]):
        return False

    command = 'docker run --rm -ti --name hdfs-setup -h hdfs-setup -v {}:/data {}
{}'.format(hadoop_host_volume_path,
                                                                                hadoop_image + ':'
+ hadoop_image_tag,
                                                                                'hdfs namenode -
format')
    if not execute_over_ssh(host, command):
        return False
    return True


def hdfs_run(env, namenode_host, secondarynamenode_host, datanodes_host):
    print 'hdfs run'
    print 'nn', namenode_host
```

```
    #-v /hdfs-data/log:/usr/local/hadoop/logs/

    command = 'docker run -d --name hdfs-namenode -h hdfs-namenode -p 9000:9000 -p
50070:50070 -v /adamcloud:/adamcloud -v {}:/data {} {}'.format(
        hadoop_host_volume_path,
        hadoop_image + ':' + hadoop_image_tag,
        'hdfs namenode'
    )
    #hdfs namenode
    if not execute_over_ssh(namenode_host, command):
        print 'Failed to setup namenode on', namenode_host
        return False

    print 'snn', secondarynamenode_host

    command = 'docker run -d --name hdfs-secondarynamenode -h hdfs-secondarynamenode -p
50090:50090 --link=hdfs-namenode:hdfs-namenode -v /adamcloud:/adamcloud -v {}:/data {}
{}'.format(
        hadoop_host_volume_path,
        hadoop_image + ':' + hadoop_image_tag,
        'hdfs secondarynamenode'
    )
    #hdfs secondarynamenode
    if not execute_over_ssh(secondarynamenode_host, command):
        print 'Failed to setup secondary namenode on', secondarynamenode_host
        return False

    index = 1
    failed = []
    for datanode_host in datanodes_host:
        print 'dn', datanode_host

        command = 'docker run -d --name hdfs-datanode{0} -h hdfs-datanode{0} -p
5008{0}:5008{0} --link=hdfs-namenode:hdfs-namenode --link=hdfs-secondarynamenode:hdfs-
secondarynamenode -v /adamcloud:/adamcloud -v {1}:/data {2} {3}'.format(
            index,
            hadoop_host_volume_path,
            hadoop_image + ':' + hadoop_image_tag,
            'hdfs datanode'
        )
        #hdfs datanode
        if not execute_over_ssh(datanode_host, command):
            failed.append(datanode_host)
            continue

    if len(failed) > 0:
        print 'Failed to setup datanodes:', failed
        return False
    return True


def spark_run(env, master_host, workers_host):
    print 'spark setup'
    if len(workers_host) > 9:
        help('You can only have up to 9 workers.')
        return False
    print 'master', master_host
    command = 'docker run -d --name spark-master -h spark-master -p 8080:8080 -p 7077:7077
{0} {1}'.format(
        spark_image + ':' + spark_image_tag,
        'spark-class org.apache.spark.deploy.master.Master'
    )
    if not execute_over_ssh(master_host, command):
        print 'Failed to setup spark master on', master_host
        return False

    print 'workers', workers_host
```

```
    index = 0
    failed = []
    for worker_host in workers_host:
        index += 1
        command = 'docker run -d --name spark-worker{0} -h spark-worker{0} -p 808{0}:808{0} -
-link=hdfs-namenode:hdfs-namenode --link=spark-master:spark-master --link=adam:adam --
link=avocado:avocado {1} {2}'.format(
            index,
            spark_image + ':' + spark_image_tag,
            'spark-class org.apache.spark.deploy.worker.Worker spark://spark-master:7077'
        )
        if not execute_over_ssh(worker_host, command):
            failed.append(worker_host)
            continue

    if len(failed) > 0:
        print 'Failed to setup workers:', failed
        return False
    return True


def help(error=''):
    print
    print error
    print
    print 'usage: env service [service-params]'
    print
    print 'env              the environment you want to setup:'
    print '                      local'
    print '                      macmini'
    print '                      aws'
    print 'service          the service to execute:'
    # print '                    hdfsconfigure [host1 [host2 [...]]]'
    print '                      hdfsformat nn-host'
    print '                      hdfsrun nn-host snn-host dn1-host [dn2-host [dn3-host
[...]]]'
    print '                      spark master-host worker1-host [worker2-host [worker3-host
[...]]]'


def validate_env(env):
    if env == 'local':
        print 'local'
    elif env == 'macmini':
        print 'macmini'
    elif env == 'aws':
        print 'aws'
    else:
        help('Environment "' + env + '" is not valid.')
        return False
    return True


def init():
    try:
        args_iterator = iter(sys.argv)
        next(args_iterator)  # Skip the path
        env = next(args_iterator)
        service = next(args_iterator)

        if not validate_env(env):
            return
        ips = list(args_iterator)
        if not is_valid_hostname(ips):
            print 'Mauvais ip'

        if service == 'hdfsconfigure':
```

```
            if not hdfs_configure(env, ips):
                print
                print 'Error: Configuring HDFS failed.'
        elif service == 'hdfsformat':
            print 'hdfssetup [host]'
            if not hdfs_format(env, ips[0]):
                print
                print 'Error: Formating HDFS failed.'
        elif service == 'hdfsrun':
            print 'hdfsrun nn-host snn-host dn1-host [dn2-host [dn3-host [...]]]'
            if not hdfs_run(env, ips[0], ips[1], ips[2:]):
                print
                print 'Error: Running HDFS failed.'
        elif service == 'spark':
            print 'spark spark-master-host worker1-host [worker2-host [worker3-host [...]]]'
            if not spark_run(env, ips[0], ips[1:]):
                print
                print 'Error: Running Spark failed.'
        else:
            help('Service "' + service + '" is not valid.')
            return
    except (IndexError, StopIteration):
        help('Some arguments are missing.')
        return


init()
```

ANNEXE III

ADAMCLOUD.PY

```
__author__ = 'flangelier'

import sys
import re
import paramiko
import getpass

snap_image = 'gelog/snap'
snap_image_tag = 'latest'

adam_image = 'gelog/adam'
adam_image_tag = 'latest'

avocado_image = 'gelog/avocado'
avocado_image_tag = 'latest'

sudo_passwords = {}


def is_valid_hostname(hosts):
    failed = []
    for host in hosts:
        # Validating only IPv4 because we need to add the IP in the containers hosts file
        if not re.compile('^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-
4][0-9]|[01]?[0-9][0-9]?)$').match(host):
            failed.append(host)
            continue
        # if len(host) > 255:
        #     return False
        # if host[-1] == ".":
        #     host = host[:-1]  # strip exactly one dot from the right, if present
        # allowed = re.compile("(?!-)[A-Z\d-]{1,63}(?<!-)$", re.IGNORECASE)
        # return all(allowed.match(x) for x in hostname.split("."))

    if len(failed) > 0:
        print "There aren't valid IPv4", failed
        return False
    return True


def execute_over_ssh(host, command):
    print 'host: %s cmd: %s' % (host, command)
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(host)
        chan = ssh.get_transport().open_session()
        if 'sudo' in command:
            chan.get_pty()
            chan.exec_command(command)
            received = chan.recv(1024)
            # If the password entered is bad, it will loop until the end of time...
            while '[sudo] password for adamcloud:' in received:
                password = sudo_passwords.get(host, None)
                if password is None:
                    print received
                    password = getpass.getpass()
                    sudo_passwords[host] = password
```

```
                stdin = chan.makefile('wb', -1)
                stdin.write(password + '\n')
                received = chan.recv(1024)
        else:
            chan.exec_command(command)
            received = chan.recv(1024)

        print(received)

        ssh.close()
    except paramiko.AuthenticationException as e:
        help('Connection to %s failed, make sure you pushed your public key on that host!' %
host)
        return False
    return True


def snap_index(fasta, snap):
    host = '127.0.0.1'
    print 'Running snap index on host', host

    sub_command = 'snap index /adamcloud/{0} /adamcloud/{1}'.format(fasta, snap)
    command = 'docker run --rm -ti --name snap -h snap -v /adamcloud:/adamcloud {}
{}'.format(
                                                                        snap_image + ':' +
snap_image_tag,
                                                                        sub_command)
    if not execute_over_ssh(host, command):
        return False
    return True


def snap_align(fastq, snap, sam):
    host = '127.0.0.1'
    print 'Running snap align on host', host

    sub_command = 'snap single /adamcloud/{0} /adamcloud/{0} -o /adamcloud/{0}'.format(snap,
fastq, sam)
    command = 'docker run --rm -ti --name snap -h snap -v /adamcloud:/adamcloud {}
{}'.format(
                                                                        snap_image + ':' +
snap_image_tag,
                                                                        sub_command)
    if not execute_over_ssh(host, command):
        return False
    return True


def adam(spark_master, hdfs_namenode, sam, adam):
    host = '127.0.0.1'
    print 'Running Adam on host', host

    sub_command = 'adam-submit --master spark://spark-master:7077 transform hdfs://hdfs-
namenode:9000/{0} hdfs://hdfs-namenode:9000/{1}'.format(sam, adam)
    command = 'docker run --rm -ti --name adam -h adam -p 4040:4040 -v /adamcloud:/adamcloud
--link=spark-master:spark-master --link=hdfs-namenode:hdfs-namenode {} {}'.format(
                                                                        adam_image + ':' +
adam_image_tag,
                                                                        sub_command)
    if not execute_over_ssh(host, command):
        return False
    return True

def avocado(spark_master, hdfs_namenode, fasta, adam, avocado):
    host = '127.0.0.1'
    print 'Running Avocado on host', host
```

```
    sub_command = 'avocado-submit --master spark://spark-master:7077 hdfs://hdfs-
namenode:9000/{0} /adamcloud/{1} /adamcloud/{2} /usr/local/avocado/avocado-sample-
configs/basic.properties'.format(adam, fasta, avocado)
    command = 'docker run --rm -ti --name avocado -h avocado -v /adamcloud:/adamcloud --
link=spark-master:spark-master --link=hdfs-namenode:hdfs-namenode {} {}'.format(
                                                                        avocado_image +
':' + avocado_image_tag,
                                                                        sub_command)
    if not execute_over_ssh(host, command):
        return False
    return True

def help(error=''):
    print
    print error
    print
    print 'usage: env service [service-params]'
    print
    print 'env              the environment you want to setup:'
    print '                      local'
    print '                      macmini'
    print '                      aws'
    print 'service          the service to execute:'
#    print '                       link_to_swarm swarm_host swarm_salve1 [swarm_salve2
[swarm_salve3 [...]]]'
    print '                      snapIndex fasta snap'
    print '                      snapAlign fastq snap sam'
    print '                      adam spark-master hdfs-namenode sam adam'
    print '                      avocado spark-master hdfs-namenode fasta adam avocado'


def init():

    try:
        args_iterator = iter(sys.argv)
        next(args_iterator)  # Skip the path
        service = next(args_iterator)

        if service == 'snapIndex':
            if not snap_index(next(args_iterator), next(args_iterator)):
                print
                print 'Error: Indexing failed.'
        elif service == 'snapAlign':
            if not snap_align(next(args_iterator), next(args_iterator), next(args_iterator)):
                print
                print 'Error: Alignment failed.'
        elif service == 'adam':
            ips = list(args_iterator[:2])
            if not is_valid_hostname(ips):
                print 'Mauvais ip'
            if not adam(ips[0], ips[1], next(args_iterator), next(args_iterator)):
                print
                print 'Error: Adam failed.'
        elif service == 'avocado':
            ips = list(args_iterator[:2])
            if not avocado(ips[0], ips[1], next(args_iterator), next(args_iterator),
next(args_iterator)):
                print
                print 'Error: Avocado failed.'
        else:
            help('Service "' + service + '" is not valid.')
            return
    except (IndexError, StopIteration):
        help('Some arguments are missing.')
        return
init()
```

ANNEXE IV

LES IMAGES *DOCKER*

Java

```
# Pull base image.
FROM ubuntu:14.04.1

MAINTAINER Francois Langelier

ENV JAVA_VERSION 7u75-2.5.4-1~trusty1
ENV JAVA_HOME /usr/lib/jvm/jdk

# Install Java.
RUN \
    apt-get update && \
    DEBIAN_FRONTEND=noninteractive apt-get install -y openjdk-7-jdk=$JAVA_VERSION && \
    ln -s /usr/lib/jvm/java-7-openjdk-amd64 /usr/lib/jvm/jdk && \
    rm -rf /var/lib/apt/lists/*
```

Snap

```
# Builds from an official Docker image
FROM ubuntu:14.04.1


# Installing Make, G++, and wget
# (wget is required to download the snap package)
RUN apt-get update && \
    DEBIAN_FRONTEND=noninteractive apt-get install -y make g++ wget && \
    rm -rf /var/lib/apt/lists/*


# Environment variables
ENV ZLIB_VERSION   1.2.8
ENV SNAP_VERSION   1.0beta.15
ENV SNAP_HOME               /usr/local/snap


# Installing ZLIB
RUN wget http://zlib.net/zlib-$ZLIB_VERSION.tar.gz  && \
    tar -xzf /zlib-$ZLIB_VERSION.tar.gz -C /usr/local/  && \
    rm /zlib-$ZLIB_VERSION.tar.gz  && \
    ln -s /usr/local/zlib-$ZLIB_VERSION /usr/local/zlib  && \
    cd /usr/local/zlib  && \
    ./configure  && \
    make
# Docker's RUN command prepends 'sh -c' to the RUN instruction, which caues the error
#
#   /bin/sh: 1:  : not found
#
# This forces us to do the 'make install' in a separate RUN instruction
# Using the brackets does not prepend 'sh -c'
#
RUN ["make", "-C", "/usr/local/zlib", "install"]
```

```
# Installing Snap
RUN wget http://snap.cs.berkeley.edu/downloads/snap-$SNAP_VERSION-linux.tar.gz  && \
    tar -xzf /snap-$SNAP_VERSION-linux.tar.gz -C /usr/local/  && \
    rm /snap-$SNAP_VERSION-linux.tar.gz  && \
    ln -s /usr/local/snap-$SNAP_VERSION-linux /usr/local/snap  && \
    ln -s /usr/local/snap/snap /usr/bin/snap


# Define the default command
ENTRYPOINT ["snap"]
```

Hadoop

```
# Building the image using my Oracle JDK 7
FROM gelog/java:openjdk7

MAINTAINER Francois Langelier

ENV WGET_VERSION       1.15-1ubuntu1.14.04.1
# Setting HADOOP environment variables
ENV HADOOP_VERSION 2.3.0
ENV HADOOP_INSTALL /usr/local/hadoop
ENV PATH $PATH:$HADOOP_INSTALL/bin
ENV PATH $PATH:$HADOOP_INSTALL/sbin
ENV HADOOP_MAPRED_HOME $HADOOP_INSTALL
ENV HADOOP_COMMON_HOME $HADOOP_INSTALL
ENV HADOOP_HDFS_HOME $HADOOP_INSTALL
ENV HADOOP_COMMON_LIB_NATIVE_DIR $HADOOP_INSTALL/lib/native
ENV YARN_HOME $HADOOP_INSTALL
ENV HADOOP_CONF_DIR /data/conf

# Installing wget
RUN \
    apt-get update && \
    apt-get install -y wget=$WGET_VERSION && \
    rm -rf /var/lib/apt/lists/*

# Installing HADOOP
RUN wget http://archive.apache.org/dist/hadoop/core/hadoop-$HADOOP_VERSION/hadoop-
$HADOOP_VERSION.tar.gz && \
    tar -zxf /hadoop-$HADOOP_VERSION.tar.gz && \
    rm /hadoop-$HADOOP_VERSION.tar.gz && \
    mv hadoop-$HADOOP_VERSION /usr/local/hadoop && \
    mkdir -p /usr/local/hadoop/logs

# Creating symlink for HADOOP configuration files
VOLUME /data
# Copying default HADOOP configuration files
ADD https://raw.githubusercontent.com/GELOG/docker-ubuntu-
hadoop/$HADOOP_VERSION/$HADOOP_VERSION/core-site.xml $HADOOP_CONF_DIR/core-site.xml
ADD https://raw.githubusercontent.com/GELOG/docker-ubuntu-
hadoop/$HADOOP_VERSION/$HADOOP_VERSION/yarn-site.xml $HADOOP_CONF_DIR/yarn-site.xml
ADD https://raw.githubusercontent.com/GELOG/docker-ubuntu-
hadoop/$HADOOP_VERSION/$HADOOP_VERSION/mapred-site.xml $HADOOP_CONF_DIR/mapred-site.xml
ADD https://raw.githubusercontent.com/GELOG/docker-ubuntu-
hadoop/$HADOOP_VERSION/$HADOOP_VERSION/hdfs-site.xml $HADOOP_CONF_DIR/hdfs-site.xml

CMD ["hdfs"]
```

Spark

```
# Inspired by these images:
# https://github.com/sequenceiq/docker-spark
# https://github.com/apache/spark/tree/master/docker
```

```
# For building this image, See https://github.com/GELOG/docker-ubuntu-java
FROM gelog/java:openjdk7

MAINTAINER Francois Langelier

ENV WGET_VERSION       1.15-1ubuntu1.14.04.1
ENV SPARK_VERSION      1.2.1
ENV SPARK_BIN_VERSION  $SPARK_VERSION-bin-hadoop2.3
ENV SPARK_HOME         /usr/local/spark
ENV PATH               $PATH:$SPARK_HOME/bin

# Installing wget
RUN \
    apt-get update && \
    apt-get install -y wget=$WGET_VERSION && \
    rm -rf /var/lib/apt/lists/*

# Installing Spark for Hadoop
RUN wget http://d3kbcqa49mib13.cloudfront.net/spark-$SPARK_BIN_VERSION.tgz && \
    tar -zxf /spark-$SPARK_BIN_VERSION.tgz -C /usr/local/ && \
    ln -s /usr/local/spark-$SPARK_BIN_VERSION $SPARK_HOME && \
    rm /spark-$SPARK_BIN_VERSION.tgz


# Default action: show available commands on startup
CMD ["spark-submit"]
```

Adam

```
# For building this image, See https://github.com/GELOG/docker-ubuntu-spark
FROM gelog/spark:1.2-bin-hadoop2.3

# Installation instructions: https://github.com/bigdatagenomics/adam/releases

# Environment variables
ENV MAVEN_VERSION       3.2.3
ENV MAVEN_HOME          /usr/local/apache-maven
ENV MAVEN_OPTS          -Xmx512m -XX:MaxPermSize=128m
ENV ADAM_VERSION        0.15.0
ENV ADAM_HOME           /usr/local/adam
ENV PATH                $ADAM_HOME/bin:$MAVEN_HOME/bin:$PATH


# Installing Maven
RUN wget http://archive.apache.org/dist/maven/maven-3/$MAVEN_VERSION/binaries/apache-maven-
$MAVEN_VERSION-bin.tar.gz && \
    tar -xzf /apache-maven-$MAVEN_VERSION-bin.tar.gz -C /usr/local/ && \
    ln -s /usr/local/apache-maven-$MAVEN_VERSION $MAVEN_HOME && \
    rm /apache-maven-$MAVEN_VERSION-bin.tar.gz
# Verify Maven installation with: mvn -version

# Installing ADAM
RUN wget https://github.com/bigdatagenomics/adam/archive/adam-parent-$ADAM_VERSION.tar.gz &&
\
    tar -xzf /adam-parent-$ADAM_VERSION.tar.gz -C /usr/local/ && \
    ln -s /usr/local/adam-adam-parent-$ADAM_VERSION $ADAM_HOME && \
    rm /adam-parent-$ADAM_VERSION.tar.gz

EXPOSE 4040

# Updates the hadoop version in Adam's pom.xml file
# FIXME: Why does ADAM requires the Hadoop version to be specified,
#        if it is already specified in Spark's installation
```

```
RUN ADAM_HADOOP_VERSION=2.3.0 && \
    sed -i -e "s/\(<hadoop.version>\).*\(<\/hadoop.version>\)/\1$ADAM_HADOOP_VERSION\2/g"
$ADAM_HOME/pom.xml

# FIXME: Investigate why Adam takes 11min to build
RUN cd $ADAM_HOME && \
    mvn clean package -DskipTests

# Default action: prints Adam's options
# See also: adam-submit, adam-shell, adam-pyspark
CMD ["adam-submit"]
```

Avocado

```
# For building this image, See https://github.com/GELOG/docker-ubuntu-spark
FROM gelog/spark:1.2-bin-hadoop2.3

# Environment variables
ENV MAVEN_VERSION       3.2.3
ENV MAVEN_HOME          /usr/local/apache-maven
ENV MAVEN_OPTS          -Xmx512m -XX:MaxPermSize=128m
ENV AVOCADO_VERSION     0.0.0-master-branch
ENV AVOCADO_HOME        /usr/local/avocado
ENV PATH                $AVOCADO_HOME/bin:$MAVEN_HOME/bin:$PATH


# Installing Maven
RUN wget http://archive.apache.org/dist/maven/maven-3/$MAVEN_VERSION/binaries/apache-maven-
$MAVEN_VERSION-bin.tar.gz && \
    tar -xzf /apache-maven-$MAVEN_VERSION-bin.tar.gz -C /usr/local/ && \
    ln -s /usr/local/apache-maven-$MAVEN_VERSION $MAVEN_HOME && \
    rm /apache-maven-$MAVEN_VERSION-bin.tar.gz
# Verify Maven installation with: mvn -version

# Installing Avocado
# FIXME: should pull from official source once the bug is fixed
RUN wget https://github.com/bigdatagenomics/avocado/archive/master.tar.gz && \
    tar -xzf /master.tar.gz -C /usr/local/ && \
    rm /master.tar.gz && \
    ln -s /usr/local/avocado-master $AVOCADO_HOME && \
    cd $AVOCADO_HOME && \
    mvn package

# FIXME: it takes 25min to build the image
# FIXME: the virtual image size is 1.07 GB (310 MB for this layer)

# Default action: print Avocado's options
CMD ["avocado-submit"]
```

ANNEXE V

LES README.MD DES DOCKERFILES

Snap

```
# Supported tags and respective `Dockerfile` links
- [`1.0beta.15`/Dockerfile](https://github.com/GELOG/docker-ubuntu-
snap/tree/1.0beta.15/Dockerfile)

# What is Snap ?
SNAP is a new sequence aligner that is 3-20x faster and just as accurate as existing tools
like BWA-mem, Bowtie2 and Novoalign. It runs on commodity x86 processors, and supports a rich
error model that lets it cheaply match reads with more differences from the reference than
other tools. This gives SNAP up to 2x lower error rates than existing tools (in some cases)
and lets it match larger mutations that they may miss. SNAP also natively reads BAM, FASTQ,
or gzipped FASTQ, and natively writes SAM or BAM, with built-in sorting, duplicate marking,
and BAM indexing.

SNAP was developed by a team from the UC Berkeley AMP Lab, Microsoft, and UCSF.

[http://snap.cs.berkeley.edu/](http://snap.cs.berkeley.edu/)

# What is Docker?
Docker is an open platform for developers and sysadmins to build, ship, and run distributed
applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging
tool, and Docker Hub, a cloud service for sharing applications and automating workflows,
Docker enables apps to be quickly assembled from components and eliminates the friction
between development, QA, and production environments. As a result, IT can ship faster and run
the same app, unchanged, on laptops, data center VMs, and any cloud.

[https://www.docker.com/whatisdocker/](https://www.docker.com/whatisdocker/)

## What is a Docker Image?
Docker images are the basis of containers. Images are read-only, while containers are
writeable. Only the containers can be executed by the operating system.

[https://docs.docker.com/terms/image/](https://docs.docker.com/terms/image/)

# Dependencies
* [Install Docker](https://docs.docker.com/installation/)

# Base Docker image
* [Ubuntu 14.04 LTS](https://registry.hub.docker.com/_/ubuntu/)

# How to use this image?
### 1) Get the reference genome (or chromosome) and unzip it.
    mkdir /data/
    wget -O /data/chr1.fa.gz
http://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chr1.fa.gz
    gzip -d /data/chr1.fa.gz
### 2) Index the reference genome (or chromosome)
    docker run --rm=true -ti -v /data:/data gelog/snap index /data/chr1.fa /data/snap-
index.chr1
### 3) Get a genome (or chromosome)
    wget -O /data/SRR062634.filt.fastq.gz
ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data/HG00096/sequence_read/SRR062634.filt.fastq.gz
    gzip -d /data/SRR062634.filt.fastq.gz

### 4) Align the genome (or chromosome) with Snap
    docker run --rm=true -ti -v /data:/data gelog/snap single /data/snap-index.chr1/
/data/SRR062634.filt.fastq -o /data/SRR062634.sam
```

Java

```
# Supported tags and respective `Dockerfile` links
- [`openjdk7`/Dockerfile](https://github.com/GELOG/docker-ubuntu-
java/blob/openjdk7/1.7u75/Dockerfile)
- [`oraclejdk7`/Dockerfile](https://github.com/GELOG/docker-ubuntu-
java/blob/oraclejdk7/Dockerfile)

# What is Java?
Java is a programming language and computing platform first released by Sun Microsystems in
1995. There are lots of applications and websites that will not work unless you have Java
installed, and more are created every day. Java is fast, secure, and reliable. From laptops
to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java
is everywhere!

[https://www.java.com/en/download/faq/whatis_java.xml](https://www.java.com/en/download/faq/w
hatis_java.xml)

# What is Docker?
Docker is an open platform for developers and sysadmins to build, ship, and run distributed
applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging
tool, and Docker Hub, a cloud service for sharing applications and automating workflows,
Docker enables apps to be quickly assembled from components and eliminates the friction
between development, QA, and production environments. As a result, IT can ship faster and run
the same app, unchanged, on laptops, data center VMs, and any cloud.

[https://www.docker.com/whatisdocker/](https://www.docker.com/whatisdocker/)

## What is a Docker Image?
Docker images are the basis of containers. Images are read-only, while containers are
writeable. Only the containers can be executed by the operating system.

[https://docs.docker.com/terms/image/](https://docs.docker.com/terms/image/)

# Dependencies
* [Install Docker](https://docs.docker.com/installation/)

# Base Docker image
* [Ubuntu 14.04 LTS](https://registry.hub.docker.com/_/ubuntu/)

# How to use this image?
    docker run --rm -ti gelog/java:openjdk7

#### Run `java`
    docker run --rm -ti gelog/java:openjdk7 java

#### Run `javac`
    docker run --rm -ti gelog/java:openjdk7 javac
```

Hadoop

```
# Supported tags and respective `Dockerfile` links
- [`2.6.0`/Dockerfile](https://github.com/GELOG/docker-ubuntu-hadoop/tree/2.6.0/Dockerfile)
- [`2.3.0`/Dockerfile](https://github.com/GELOG/docker-ubuntu-hadoop/tree/2.3.0/Dockerfile)

# What is Hadoop?
The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed
computing.

The Apache Hadoop software library is a framework that allows for the distributed processing
of large data sets across clusters of computers using simple programming models. It is
designed to scale up from single servers to thousands of machines, each offering local
computation and storage. Rather than rely on hardware to deliver high-availability, the
library itself is designed to detect and handle failures at the application layer, so
delivering a highly-available service on top of a cluster of computers, each of which may be
prone to failures.
```

```
[http://hadoop.apache.org/](http://hadoop.apache.org/)

# What is Docker?
Docker is an open platform for developers and sysadmins to build, ship, and run distributed
applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging
tool, and Docker Hub, a cloud service for sharing applications and automating workflows,
Docker enables apps to be quickly assembled from components and eliminates the friction
between development, QA, and production environments. As a result, IT can ship faster and run
the same app, unchanged, on laptops, data center VMs, and any cloud.

[https://www.docker.com/whatisdocker/](https://www.docker.com/whatisdocker/)

## What is a Docker Image?
Docker images are the basis of containers. Images are read-only, while containers are
writeable. Only the containers can be executed by the operating system.

[https://docs.docker.com/terms/image/](https://docs.docker.com/terms/image/)

# Dependencies
* [Install Docker](https://docs.docker.com/installation/)

# Base Docker image
* [gelog/java:openjdk7](https://registry.hub.docker.com/u/gelog/java/)

# How to use this image?
### Formating the namenode
    docker run -d --name hdfs-namenode -h hdfs-namenode gelog/hadoop:2.3.0 hdfs namenode -
format
### Starting the namenode
    docker run -d --name hdfs-namenode -h hdfs-namenode -p 9000:9000 -p 50070:50070
gelog/hadoop:2.3.0 hdfs namenode
### Starting a secondary namenode
    docker run -d --name hdfs-secondarynamenode -h hdfs-secondarynamenode -p 50090:50090 --
link=hdfs-namenode:hdfs-namenode gelog/hadoop:2.3.0 hdfs secondarynamenode
### Starting a datanode
    docker run -d --name hdfs-datanode1 -h hdfs-datanode1 -p 50081:50081 --link=hdfs-
namenode:hdfs-namenode --link=hdfs-secondarynamenode:hdfs-secondarynamenode
gelog/hadoop:2.3.0 hdfs datanode
```

Spark

```
# Apache Spark

[![dockeri.co](http://dockeri.co/image/gelog/spark)](https://registry.hub.docker.com/u/gelog/
spark/)

[![stars](https://img.shields.io/github/stars/apache/spark.svg)
![forks](https://img.shields.io/github/forks/apache/spark.svg)
![issues](https://img.shields.io/github/issues/apache/spark.svg)
](https://github.com/apache/spark)

## Supported tags and respective `Dockerfile` links
- [`1.2-bin-hadoop2.3`/Dockerfile](https://github.com/GELOG/docker-ubuntu-
spark/blob/master/1.2-bin-hadoop2.3/Dockerfile)
- [`1.1-bin-hadoop2.4`/Dockerfile](https://github.com/GELOG/docker-ubuntu-
spark/blob/master/1.1-bin-hadoop2.4/Dockerfile)
- [`1.1-bin-hadoop2.3`/Dockerfile](https://github.com/GELOG/docker-ubuntu-
spark/blob/master/1.1-bin-hadoop2.3/Dockerfile)

## What is Spark ?
Apache Spark is a fast and general-purpose cluster computing system. It provides high-level
APIs in Java, Scala and Python, and an optimized engine that supports general execution
graphs. It also supports a rich set of higher-level tools including [Spark
SQL](https://spark.apache.org/docs/latest/sql-programming-guide.html) for SQL and structured
data processing, [MLlib](https://spark.apache.org/docs/latest/mllib-guide.html) for machine
learning, [GraphX](https://spark.apache.org/docs/latest/graphx-programming-guide.html) for
```

```
graph processing, and [Spark Streaming](https://spark.apache.org/docs/latest/streaming-
programming-guide.html).

[https://spark.apache.org/docs/latest/](https://spark.apache.org/docs/latest/)

## What is Docker?
Docker is an open platform for developers and sysadmins to build, ship, and run distributed
applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging
tool, and Docker Hub, a cloud service for sharing applications and automating workflows,
Docker enables apps to be quickly assembled from components and eliminates the friction
between development, QA, and production environments. As a result, IT can ship faster and run
the same app, unchanged, on laptops, data center VMs, and any cloud.

[https://www.docker.com/whatisdocker/](https://www.docker.com/whatisdocker/)

### What is a Docker Image?
Docker images are the basis of containers. Images are read-only, while containers are
writeable. Only the containers can be executed by the operating system.

[https://docs.docker.com/terms/image/](https://docs.docker.com/terms/image/)

## Dependencies
* [Install Docker](https://docs.docker.com/installation/)

## Base Docker image

| Branch               | Base Image      | Description               |
| -------------------- | --------------- | ------------------------- |
| master               | [gelog/java:openjdk7](https://registry.hub.docker.com/u/gelog/java/)
| Spark pre-built for Hadoop |
| spark-for-hadoop     | "             " | Spark pre-built for Hadoop (dev branch) |
| spark-from-source    | scala:2.10.4    | Spark built from source |

Note: currently the spark-from-source image takes quite a while to build, and generates 2.3
GB of virtual size.

The recommended branch for general use is **master**.

## How to use this image?

### Spark Master
    docker run -d --name spark-master -h spark-master -p 8080:8080 -p 7077:7077 \
    gelog/spark:1.2-bin-hadoop2.3 spark-class org.apache.spark.deploy.master.Master
### Spark Worker
    docker run -d --name spark-worker1 -h spark-worker1 --link=hdfs-namenode:hdfs-namenode --
link=spark-master:spark-master \
    gelog/spark:1.2-bin-hadoop2.3 spark-class org.apache.spark.deploy.worker.Worker
spark://spark-master:7077
```

Adam

```
# Supported tags and respective `Dockerfile` links
- [`0.15.0`/Dockerfile](https://github.com/GELOG/docker-ubuntu-
adam/blob/master/0.15.0/Dockerfile)
- [`0.14.0`/Dockerfile](https://github.com/GELOG/docker-ubuntu-
adam/blob/master/0.14.0/Dockerfile)

# What is Adam ?
ADAM provides both an application programming interface (API) and a command line interface
(CLI) for manipulating genomic data on a computing cluster. ADAM operates on data stored
inside of [Parquet](http://www.parquet.io/) with the [bdg-
formats](http://bdgenomics.org/projects/bdg-formats/) schemas, using [Apache
Spark](http://spark.apache.org/), and provides scalable performance on clusters larger than
100 machines.

ADAM is on [Github](https://github.com/bigdatagenomics/adam). Quick start guides are
available for [running ADAM on EC2](https://github.com/bigdatagenomics/adam/wiki/Running-
```

```
ADAM-on-EC2), and for [building ADAM for specific CDH
releases](https://github.com/bigdatagenomics/adam/wiki/Running-ADAM-on-CDH-4-or-5).

http://bdgenomics.org/projects/adam/

# What is Docker?
Docker is an open platform for developers and sysadmins to build, ship, and run distributed
applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging
tool, and Docker Hub, a cloud service for sharing applications and automating workflows,
Docker enables apps to be quickly assembled from components and eliminates the friction
between development, QA, and production environments. As a result, IT can ship faster and run
the same app, unchanged, on laptops, data center VMs, and any cloud.

https://www.docker.com/whatisdocker/

## What is a Docker Image?
Docker images are the basis of containers. Images are read-only, while containers are
writeable. Only the containers can be executed by the operating system.

https://docs.docker.com/terms/image/

# Dependencies
* [Install Docker](https://docs.docker.com/installation/)

# Base Docker image
* [gelog/spark:1.1.0-bin-hadoop2.3](https://registry.hub.docker.com/u/gelog/spark/)

# How to use this image?
### 1) Get the aligned file of a genome (or chromosome)
#### 1.1) Get it from [Snap](https://github.com/GELOG/docker-ubuntu-snap)
#### 1.2) Download it from an external source (Warning: This file is 14.5 GB)
    mkdir /data/
    wget -O /data/HG00096.mapped.ILLUMINA.bwa.GBR.low_coverage.20120522.bam
ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data/HG00096/alignment/HG00096.mapped.ILLUMINA.bwa.G
BR.low_coverage.20120522.bam
### 2) Transform the genome (or chromosome) with Adam
#### 2.1) From [Snap](https://github.com/GELOG/docker-ubuntu-snap)
    docker run --rm=true -ti -v /data/:/data gelog/adam adam-submit transform
/data/SRR062634.sam /data/SRR062634.adam
#### 2.2) From an external source
    docker run --rm=true -ti -v /data/:/data gelog/adam adam-submit transform
/data/HG00096.mapped.ILLUMINA.bwa.GBR.low_coverage.20120522.bam /data/HG00096.adam
```

Avocado

```
# Supported tags and respective `Dockerfile` links
- [`0.0.0-master-branch-s1.2-h2.3`/Dockerfile](https://github.com/GELOG/docker-ubuntu-
avocado/blob/master/0.0.0-master-branch-s1.2-h2.3/Dockerfile)
- [`0.0.0-master-branch-s1.1-h2.3`/Dockerfile](https://github.com/GELOG/docker-ubuntu-
avocado/blob/master/0.0.0-master-branch-s1.1-h2.3/Dockerfile)

# What is Avocado ?
avocado is a distributed pipeline for calling variants, and is built on top of [Apache
Spark](http://spark.apache.org/) and the [ADAM API](http://bdgenomics.org/projects/adam/).
avocado provides a highly configurable pipeline that can be used for the alignment,
processing, and variant calling of genomes/exomes/targets. We are currently in the process of
hardening avocado for clincial use, and expanding the avocado pipeline so that it can triage
processing steps based on genomic complexity.

avocado is on [Github](https://github.com/bigdatagenomics/avocado), and is in active
development.

[http://bdgenomics.org/projects/avocado/](http://bdgenomics.org/projects/avocado/)

# What is Docker?
Docker is an open platform for developers and sysadmins to build, ship, and run distributed
applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging
```

```
tool, and Docker Hub, a cloud service for sharing applications and automating workflows,
Docker enables apps to be quickly assembled from components and eliminates the friction
between development, QA, and production environments. As a result, IT can ship faster and run
the same app, unchanged, on laptops, data center VMs, and any cloud.

[https://www.docker.com/whatisdocker/](https://www.docker.com/whatisdocker/)

## What is a Docker Image?
Docker images are the basis of containers. Images are read-only, while containers are
writeable. Only the containers can be executed by the operating system.

[https://docs.docker.com/terms/image/](https://docs.docker.com/terms/image/)

# Dependencies
* [Install Docker](https://docs.docker.com/installation/)

# Base Docker image
* [gelog/spark:1.1.0-bin-hadoop2.3](https://registry.hub.docker.com/u/gelog/spark/)

# How to use this image?
### 1) Get the adam file of a genome (or chromosome)
#### 1.1) Get it from [Adam](https://github.com/GELOG/docker-ubuntu-adam)

### 2) Get the reference genome (or chromosome) and unzip it.
    mkdir /data/
    wget -O /data/chr1.fa.gz
http://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chr1.fa.gz
    gzip -d /data/chr1.fa.gz
### 3) Find the variation of the genome (or chromosome) with Avocado
    docker run -ti --rm --name client-genomics -v /data:/data gelog/avocado /bin/bash
    avocado-submit /data/SRR062634.adam /data/chr1.fa /data/SRR062634.avr
/usr/local/avocado/avocado-sample-configs/basic.properties
```