UNIVERSITÉ Clermont Auvergne

ISIMA
Institut Supérieur d'Informatique de Modélisation et de leurs Applications

RAPPORT DE STAGE

Mise en œuvre d’outils du cloud computing pour l’intégration d’un système d’informations scientifique

RÉALISÉ PAR:
Sammy GILLES

SOUS LA DIRECTION DE
Patrick MOREAU, Malika NASSIF et Vincent NEGRE

Pour l’obtention du Master 1 Informatique
Année universitaire 2017-2018

Structure d'accueil 5
- INRA 5
- MISTEA / LEPSE 5

Cahier des charges 6
- Contexte 6
  - Projet PHENOME 6
  - Système d'information PHIS 7
  - Infrastructure pour le déploiement 7
    - Cloud France Grille 7
    - Cloud EGI 8
- Etat de l'art 9
  - Les bases de données 9
  - Le web service 9
  - Application Web 9
- Problématique 10
- Analyse des besoins 10
- Choix des technologies 11
  - La documentation 11
  - La Virtualisation 11
  - Conteneurisation 12
  - Orchestration 16
- Choix de l'architecture 17
  - Analyse des conteneurs 17
  - Organisation des conteneurs 17
  - Architecture en local / sur cloud 19

Rapport technique 21
- Docker 21
  - Dockerfile 21
  - Gestion de conteneurs en réseau local 23
- Déploiement sur un environnement de type cloud 24
  - Gestion de conteneur sur cloud 25
  - Orchestration 26

Résultats 28
- Containerisation du système d'informations 28
- Déploiement en local 29
- Déploiement sur cloud et orchestration 29
- Documentation technique, formation utilisateur 30

Évolutions possibles 31
- Sécurité 31
- Replicas et shard Mongodb 31

Gestion de projet 33
- Démarche personnelle 33
- Planification 34

Conclusion 35

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui m'ont aidé lors de la réalisation de ce stage ainsi que lors de la rédaction de ce rapport.

Tout d'abord, mes remerciements vont à Patrick MOREAU, Malika NASSIF, Vincent NEGRE et Anne TIREAU, mes encadrants, qui m'ont fait confiance me laissant ainsi de grandes libertés tout au long de ce stage. Je tiens à les remercier aussi pour la qualité de leur suivi.

Je tiens ensuite à remercier toute l'équipe de MISTEA, qui m'a accordé une aide complémentaire à celle de mes encadrants en apportant leur expertise en cas de besoin. Je les remercie aussi du temps consacré au test utilisateur de mon travail.

Finalement, je tiens à remercier toute l'équipe des unités LEPSE[*] et MISTEA[*] de l'INRA[*] de Montpellier pour l'ambiance chaleureuse dans laquelle mon stage a pu s'effectuer.

GLOSSAIRE

UMR: Unité Mixte de Recherche.

INRA : Institut National de la Recherche Agronomique.

LEPSE : Laboratoire d'Ecophysiologie des Plantes sous Stress Environnementaux.

MISTEA: Mathématiques, Informatique et STatistique pour l'Environnement et l'Agronomie.

PHIS: Phenotyping Hybrid Information System.

Système d'information: un ensemble organisé de ressources qui permet de collecter, stocker, traiter et distribuer de l'information souvent grâce à un ordinateur. Dans mon cas, le système d'informations PHIS est constitué d'une base de données relationnelles, d'une base de données NoSQL, d'un triplet store et d'une interface web et de services Web permettant de déposer, consulter ou exporter les données.

Conteneur: Un conteneur est une enveloppe virtuelle qui permet de packager une application avec tous les éléments dont elle a besoin pour fonctionner : fichiers source, runtime, librairies, outils et fichiers. Ils sont packagés en un ensemble cohérent et prêt à être déployé sur un serveur et son OS. Contrairement à la virtualisation de serveurs et à une machine virtuelle, le conteneur n'intègre pas d'OS, il s'appuie directement sur le système d'exploitation du serveur sur lequel il est déployé.

VM: Virtual Machine, environnement informatique isolé du système hôte, répartition définitive des ressources physiques.

SE/OS: Système d'Exploitation (Operating System), ensemble de programmes qui gèrent l'utilisation des ressources de l'ordinateur (mémoire vive, disque dur, processeur) par les logiciels applicatifs.s.

Kernels/Noyaux: partie du système d'informations gérant les ressources de l'ordinateur et permettant aux différents composants matériels et logiciels de communiquer entre eux. établissant le lien entre les éléments physiques de la machine et le système d'exploitation.

Distribution GNU/Linux: terme désignant les différents systèmes d'exploitation GNU/Linux.

GNU/Linux: Système d'Exploitation open-source (libre de droit).

Cloud computing: concept consistant à exploiter la puissance de calcul ou de stockage de serveurs informatiques distants par l'intermédiaire d'un réseau (Internet).

TABLE DES FIGURES

Figure 1: Réseau PHENOME……………………………………………………………………….. 6
Figure 2: architecture du Système d'Information PHIS…………………………………………..... 7
Figure 3: Ressources Cloud France Grille Février 2018…………………………………………. 8
Figure 4: Containers VS VM……………………………………………………………………… 12
Figure 5: Position 1 de PHIS dans les containers………………………………………………… 18
Figure 6: Position 2 de PHIS dans les containers………………………………………………... 18
Figure 7: Architecture containers PHIS Locale…………………………………………………… 19
Figure 8: Architecture containers PHIS 1 VM……………………………………………………. 19
Figure 9: PHIS sur cloud sur plusieurs VM - sans orchestrateurs………………………………… 20
Figure 10: PHIS sur cloud sur plusieurs VM - avec orchestrateurs………………………………. 20
Figure 11: Architecture d'un cluster……………………………………………………………… 25
Figure 12: Shard Mongo………………………………………………………………………….. 31
Figure 13: Architecture mongo shard avec docker……………………………………………….. 32
Figure 14: Planification prévisionnelle…………………………………………………………... 34
Figure 15: Planification réalisé……...…………………………………………………………… 34

1. Structure d'accueil

1.1. INRA

Dans le cadre de mon Master 1 informatique à l'Université Clermont Auvergne (UCA), j'ai effectué mon stage à l'Institut National de la Recherche Agronomique (INRA) de Montpellier.

L'INRA est un établissement public de recherche en agronomie fondé en 1946, actuellement présidé par Philippe Mauguin, ayant le statut d'Établissement Public à caractère Scientifique et Technologique (EPST), et sous la double tutelle du ministère chargé de la Recherche et du ministère chargé de l'Agriculture.
Premier institut de recherche agronomique en Europe et deuxième dans le monde en nombre de publications en sciences agricoles et en sciences de la plante et de l'animal, l'INRA mène des recherches finalisées pour une alimentation saine et de qualité, pour une agriculture durable, et pour un environnement préservé et valorisé.

1.2. MISTEA / LEPSE

L'UMR Mathématiques, Informatique et STatistique pour l'Environnement et l'Agronomie (MISTEA) associe des (enseignants-)chercheurs du département Mathématiques et Informatique Appliquées (MIA) de l'INRA et du département Sciences pour les AgroBioProcédés de Montpellier SupAgro.

Les activités de l'UMR portent sur le développement de méthodes mathématiques, statistiques et informatiques pour l'analyse et l'aide à la décision des systèmes relevant de l'agronomie et de l'environnement, avec un accent particulier sur la modélisation, les systèmes dynamiques et les systèmes complexes.

Les travaux de l'UMR LEPSE visent à accompagner l'émergence d'une agriculture plus économe en eau et résiliente face aux changements climatiques. Ils étudient les réponses des plantes à la sécheresse et aux températures élevées et explorent leur variabilité génétique, entre ou au sein d'une même espèce. Ces réponses sont formalisées puis intégrées dans des modèles qui permettent d'évaluer les performances des modes de culture, des espèces, et des variétés actuelles ou à créer en fonction des scénarios climatiques du futur.

2. Cahier des charges

2.1. Contexte

2.1.1. Projet PHENOME

L'INRA conduit différents projets dans le domaine de l'agronomie. Il coordonne le projet PHENOME qui est un projet d'infrastructure nationale s'inscrivant dans le cadre des Investissements d'Avenir. L'objectif de PHENOME est de:

- construire des plates-formes fortement instrumentées sur sept sites en France, répondant aux besoins des principales espèces agronomiques et permettant de tester les contraintes environnementales majeures ;
- développer de nouveaux capteurs, méthodes d'analyse statistique et bases de données compatibles avec les millions de données qui seront générées ;
- diffuser ces nouvelles techniques et méthodes vers la communauté française de phénotypage (sociétés semencières, instituts techniques, recherche publique) ;
- faciliter l'émergence de PME françaises impliquées dans le développement de méthodes de phénotypage.

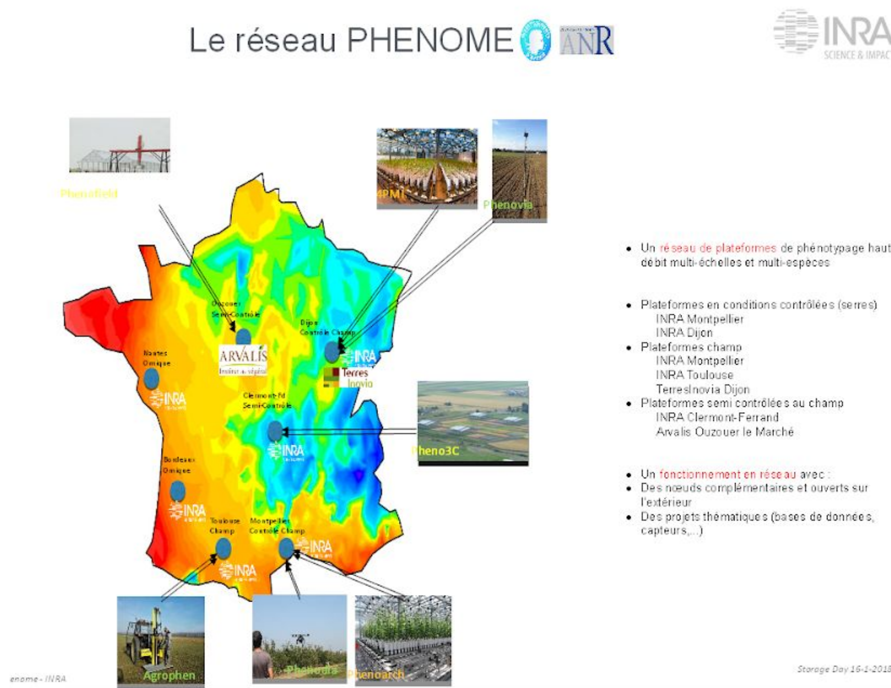

Figure 1: Réseau PHENOME

Mon stage s'effectue dans le cadre de ce projet PHENOME et plus précisément sur le système d'information PHIS (Phenotyping Hybrid Information System) développé par l'UMR Mathématiques, Informatique et STatistique pour l'Environnement et l'Agronomie (MISTEA) de l'INRA de Montpellier.

2.1.2. Système d'information PHIS

PHIS est un système d'information scientifique conçu pour gérer des données multi-échelles et multi-sources issues d'expérimentations en phénotypage haut-débit sur les plantes.

Le sujet de mon stage est d'étudier et de mettre en œuvre les techniques permettant d'empaqueter les différentes couches de PHIS et d'automatiser leur déploiement.

PHIS est composé de:

- 3 systèmes de bases de données différents : une base relationnelle SQL, une base NoSQL et un triplestore,
- des services Web,
- un serveur Web.

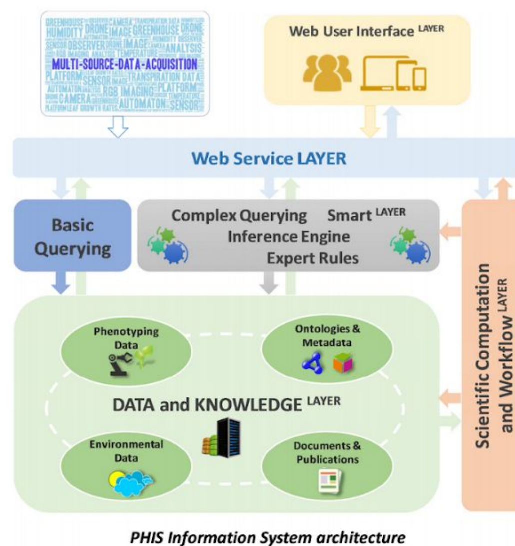

Figure 2: architecture du Système d'Information PHIS

2.1.3. Infrastructure pour le déploiement

Pour le déploiement de PHIS j'ai eu accès aux infrastructures de France Grille et de l'EGI, plus précisément au Cloud de ces derniers.
Ce sont tous deux des Clouds de type IaaS (Infrastructure As A Service).
Le déploiement sur ce type d'infrastructure permet au client de :

- de disposer des ressources dont il a besoin (calcul et stockage);
- la souplesse de ces ressources;
- gérer uniquement les applications déployées sans la gestion du serveur.

2.1.3.1. Cloud France Grilles

France Grilles est une structure française intégrée dans l'infrastructure européenne EGI.
L'infrastructure principale de France Grille est la grille, qui représente 95% de leur ressources. Dans le cadre du déploiement de PHIS c'est leur infrastructure Cloud qui sera utilisé. Elle est actuellement en expansion elle ne représente que 5% des ressources de France Grilles.

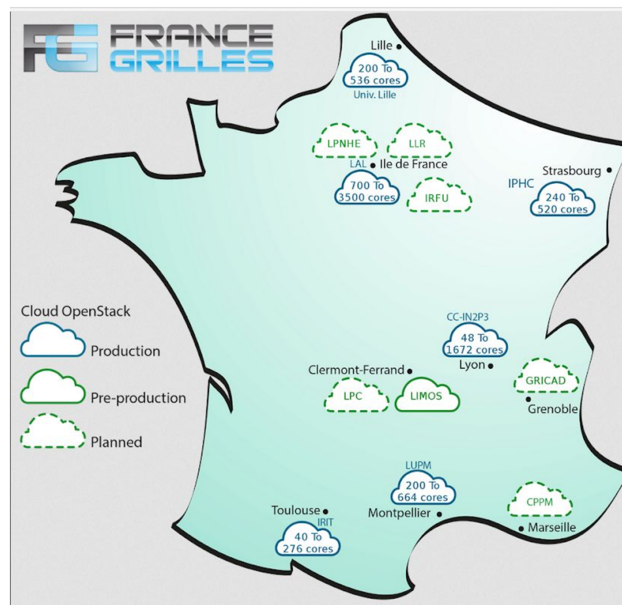

Figure 3: Ressources Cloud France Grille Février 2018

Ce Cloud est intégré à la fédération de clouds européens EGI.

2.1.3.2. Cloud EGI

C’est une fédération de divers Cloud Européens dont l’infrastructure est destiné à la recherche.
Grâce à cette fédération de nombreux clouds, l’accés à l’EGI permet de disposer de nombreuse ressources partout en Europe.

2.2. Etat de l’art

2.2.1. Les bases de données

- La base de donnée SQL sert à la gestion :
  - de la position géospatiale des objets agronomiques (plantes, parcelle, capteur, etc.),
  - des comptes utilisateur, avec la liste des utilisateurs et des groupes,

- La base de donnée NoSQL sert quant à elle à la gestion des grands volumes de données, des documents et fichiers comme les images drône, les publications scientifiques, etc… produits par les utilisateurs sur PHIS.

- Le triplestore est utilisé pour le stockage des métadonnées au format standardisé (RDF) et des connaissances (ontologies) de PHIS, il permet de faire du contrôle du vocabulaire et du raisonnement simple sur les données de PHIS.

Ces 3 systèmes de base de données utilisent chacun une technologie différente à savoir : PostgreSQL pour la base SQL, MongoDB pour la base NoSQL et RDF4J pour le triplestore.

2.2.2. Le web service

PHIS s’appuie sur un web service qui sert d’interface aux différentes sources de données décrites précédemment et fournit une API interopérable d’accès aux données.

Ce web service REST est implémenté en Java et implémente toute la communication/gestion des bases de données mais aussi l’exploitation et de ces données.

Lors de l’installation de PHIS il est nécessaire d’adapter la configuration du web service afin d’assurer la communication avec les différentes bases de données. A chaque déploiement il faut donc reconstruire l’archive de l’application Web avant son déploiement au sein du serveur d’application Tomcat.

2.2.3. Application Web

PHIS fournit une application Web qui permet d’utiliser le Web service. Cette application est implémentée en PHP à l’aide du framework Yii2 et s'exécute avec le serveur Web Apache2. Elle implémente une interface permettant une utilisation de tous les services de PHIS.

Il en va de même que pour le Web service, il est nécessaire pour cette application que certains fichiers de configurations soient adaptés pour permettre la connexion au service. Il est donc indispensable de reconstruire le projet avant son déploiement avec Apache.

2.3. Problématique

PHIS résulte d'une communication entre pas moins de 5 services différents. Le déploiement de PHIS passe par l'installation et la configurations de ces 5 services. Comme pour toute installation ces différents services ont des dépendances avec des librairies et packages ainsi qu'avec les versions de ces derniers pouvant entraîner des conflits avec d'autre packages, librairies ou applications présentes sur le système. De plus ces dépendances peuvent évoluer avec les mises à jour.

Il est donc essentiel de simplifier le plus possible ce déploiement. En plus de simplifier le déploiement, il est nécessaire de rendre chacun des services de PHIS indépendants (modularisation) afin de pouvoir aisément poursuivre l'évolution de PHIS même en phase de production. Aussi il est important de pouvoir distribuer facilement PHIS à des collaborateurs du projet.

Par ailleurs les besoins applicatifs sont susceptibles d'évoluer au cours du projet : ajout de nouvelles instances du système d'informations pour des partenaires, ajout de nouvelles ressources afin de répondre à une montée en charge du système, ….

Il est donc nécessaire de concevoir un système capable de s'adapter aux besoins applicatifs le plus rapidement possible. Le déploiement doit donc être scalable.

Mon travail au cours de ce stage sera d'analyser et mettre en oeuvre un système permettant de déployer rapidement et simplement PHIS sur différents services de Cloud, en faisant attention à la robustesse et à l'évolutivité de ce dernier.

2.4. Analyse des besoins

- Simplification par l'amélioration de l'existant:

Au moment de mon arrivée, une documentation sur le déploiement de PHIS existe mais n'est plus à jour et n'a pas d'instruction distincte pour une installation locale et pour un déploiement en réseau.
C'est pourquoi, une première simplification de déploiement consiste en la rédaction d'une documentation sur l'installation locale. Cette dernière est mise à disposition sur github.

- Utilisation et mise en place de nouveaux outils

Actuellement aucun outil d'automatisation du déploiement n'existe. Déployer PHIS se fait uniquement par l'installation et la configuration manuelle des différents services qui le compose.
Une partie du travail consiste en la recherche et l'analyse de technologies permettant un déploiement simplifié/automatisé de PHIS. J'ai ensuite mis en oeuvre les technologies retenues.

2.5. Choix des technologies

2.5.1. La documentation

Afin d’être la plus universelle possible cette documentation est en anglais.
De plus elle vient s’ajouter à un travail de documentation, sur l’utilisation de github, déjà réalisé par d’autres membres de l’équipe, pour cela elle est réalisée avec mkdocs qui permet une mise en page rapide d’un document et qui est naturellement interprété par github.

Pour cette documentation la stratégie que j’ai retenu est de guider l’utilisateur depuis l’installation du plus simple outil d’aide au déploiement jusqu’à la configuration de PHIS en passant, bien sûr, par l’installation des dépendances. Le tout afin de minimiser les erreurs pouvant être commises durant le déploiement. Toutefois, il existe de nombreuses possibilités d’erreurs, d’exceptions liées au système sur lequel le déploiement est effectué. Ces erreurs sont liées aux différentes mises à jour des outils … Pour palier à cela une partie portant uniquement sur les erreurs identifiées durant l’installation est présente dans la documentation.

2.5.2. La Virtualisation

Bien que pratique la documentation ne fait que guider, elle ne réduit pas le temps de déploiement et ne l’automatise pas.
Dans le cas de PHIS, un des principes pouvant simplifier son déploiement est la virtualisation, c’est à dire la création d’environnements virtuels dans un environnement hôte.
La virtualisation consiste à isoler du système hôte une ou plusieurs applications afin de les placer dans un environnement qui leur est propre et adapté.
Il existe deux principes de virtualisation :

- les Machines Virtuelles (VM);
- la conteneurisation.

Les VM contiennent un Système d’Exploitation (SE/OS) complet avec ses pilotes, fichiers binaires ou bibliothèques, ainsi que l'application elle-même (“VM invitée”). Chaque VM s'exécute sur un hyperviseur, qui s'exécute à son tour sur un système d'exploitation hôte, qui lui-même fait fonctionner le matériel du serveur physique. dans un autre OS en lui attribuant des valeurs physiques de la machine hôte. Elle a pour avantage de permettre l’installation de tous les environnements sur tous les OS, mais pour désavantage d’attribuer des valeurs physiques fixes (nombre de CPUS, mémoire vive, espace disque) pour chaque VM invitée. La virtualisation induit un certain gaspillage des ressources liées à la couche d’hypervision et au déploiement d’un système d’exploitation isolé sur chaque VM invitée.

La conteneurisation consiste à créer des petits environnements Linux dans un OS Linux, isolant ainsi cet environnement de l’OS hôte mais pas du noyau Linux. Par conséquent il n’y a pas d’attribution de valeurs physiques et le container consomme uniquement les ressources qui lui sont nécessaires. Cette approche réduit le gaspillage des ressources car chaque conteneur ne renferme que l'application et les fichiers binaires ou bibliothèques associés. On utilise donc le même système d'exploitation hôte pour plusieurs conteneurs, au lieu d'installer un OS pour chaque VM invitée.

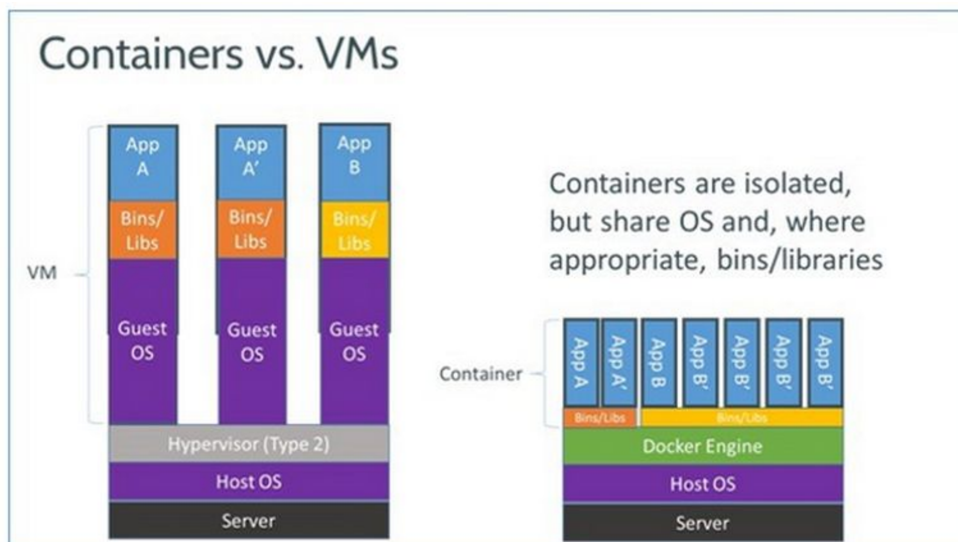

Figure 4: Containers VS VM

L'objectif étant de faciliter le déploiement d'une seule application dans n'importe quel environnement, la conteneurisation s'est imposée comme la solution idéale.

2.5.3. Conteneurisation

Pour choisir la technologie de conteneurisation la mieux adaptée j'ai réalisé un état de l'art des technologies existantes et je les ai comparées les unes aux autres.

Actuellement on peut identifier sur le marché 5 solutions dominantes qui sont les suivantes :

- Docker
- Singularity
- Shifter
- LXC/LXD
- Rocket

Shifter étant très similaire à Singularity, je ne lui réserverai pas de partie dans l'analyse des solutions en sachant que toutes les informations sur Singularity sont valables pour Shifter.

Afin d'avoir rapidement une idée de l'utilité de chacune de ces solutions voici un tableau comparatif:

	LXC/LXD	**Singularity**	**Docker**	**Rocket**
Architecture	Monolithique	Monolithique	Micro-service	Micro-service
Droit/Sécurité	Utilisateur régulier	Utilisateur régulier	Root	Utilisateur régulier Namespace cgroup
Images	Format spécifique à LXD	Singularity compatibilité Docker	Docker LXC	ACI Docker
Distribution hôte	Ubuntu	La majorité des Linux	la majorité des Linux (Windows)	Linux Red Hat et dérivés Linux Debian et dérivés

Distribution Linux pour les images container	La majorité des Linux			CoreOS Les distributions les plus populaires de Linux
Cadre d'utilisation	Isolation de toute une application dans un environnement Linux complet (idée similaire à une VM) Souvent utilisé dans le but de résoudre des problèmes de compatibilité	Isolation de toute une application, généralement utilisé dans le calcul scientifique pour sa gestion des ressources Cloud sur HPC	Microservice : Un container contient un seul processus, c'est la connexion de plusieurs containers entre eux qui forme l'application	
Position sur le marché	Ancienneté (2008) Le seul a s'apparenter à une VM Isolé mais communauté active Canonical	Dominant dans la recherche et le calcul scientifique Très utilisé sur les HPC Soutenu par de grands noms du hardware comme Intel, HP. Principal concurrent : Shifter	Dominant sur le marché public Utilisé par de nombreux géants du service en ligne (Netflix, Spotify…) De plus en plus isolé	De plus en plus populaire Utilisé par quelques start up (blablacar…) Très soutenu par les grandes entreprises : Google, RedHat Racheté par Red Hat en Mai 2018
Documentation/ Complexité d'utilisation	Peu documenté, la majorité des informations sont postées sur des forum de la communauté Ubuntu	Documentations sur le site officiel et tutoriels fournis par les entreprises soutenant le projet (site d'Intel) Avec la documentation simple à mettre en place	Documentation officielle payante Par sa position dominante de nombreux tutoriels et documents amateurs existent Par la communauté et les documents disponibles Docker est la solution la plus simple à utiliser	Peu de documents, Un peu difficile par le manque de documentations mais très modulable et compatible avec des outils annexes
Orchestration	Kubernetes	Kubernetes	Docker swarm Kubernetes	Kubernetes

- Architecture :

Cette ligne désigne l'architecture à adopter pour l'application déployée avec la solution (désignée par la colonne). Pour une architecture dite Monolithique, l'application au sein du container est complète, c'est à dire que toutes ses dépendances et ses services doivent être dans le même container. L'installation de l'application correspond au déploiement du container.
Pour une architecture en micro-services l'application est divisée en plusieurs containers isolant chacun un service de l'application et communiquant entre eux. C'est ce réseau de container qui constitue l'application. Installer l'application revient donc à installer l'ensemble des containers qui la compose.

- Droits/Sécurité

Cette ligne porte sur les droits du container sur le système hôte, cela revient à étudier les droits d'une application sur le système suivant quel utilisateur lance l'application.
On remarquera la différence « utilisateur régulier » et « root »:

- Utilisateur régulier signifie que les containers peuvent être lancés depuis un utilisateur régulier du système hôte, par conséquent le container aura les mêmes droits sur le système hôte que ce dernier. Généralement un utilisateur régulier peut lire et écrire dans le répertoire « home » de l'utilisateur qui a lancé le container mais peut uniquement lire dans les fichiers et répertoires en amont.
- Utilisateur root signifie que le container peut uniquement être lancé par l'utilisateur root, par conséquent le container possède les droits de modifier tous les dossiers/fichiers de l'OS hôte.

- Images

Sur cette ligne figure le format des images containers prises en compte par la solution. La plupart des solutions possèdent un format spécifique mais les images Docker sont soit directement compatibles soit convertibles.
Seul Rocket utilise des images dans un format développé indépendamment du projet, bien que actuellement Rocket est le seul compatible avec ce format. L'objectif est de faire de ce modèle un standard.

- Distribution hôte

Cette ligne renseigne sur les distributions pouvant accueillir la solution. Elles sont pour la plupart compatibles avec presque toutes les distributions Linux. Seul LXC/LXD, développé par Canonical, se veut uniquement compatible avec Ubuntu et ses dérivés mais il peut être porté sur d'autres distributions avec des difficultés.

Docker est compatible Windows, mais installer Docker sous Windows consiste à créer une VM Linux sur Windows et installer Docker sur cette VM. Effectivement les containers isolent des systèmes Linux mais utilisent un noyau Linux déjà présent sur la machine, il est donc obligatoire qu'un noyau Linux soit disponible sur l'hôte.

- Distribution Linux pour les images container

Sur cette ligne figurent les distributions Linux pour lesquelles il existe une image container au format de la solution. La majorité des solutions étant compatibles avec les images Docker, et Docker possédant la quasi totalité des distributions Linux, il est donc aisé d'obtenir une image pour toutes les distributions quelque soit la solution de conteneurisation.

- Cadre d'utilisation

Cette ligne est liée à l'architecture à adopter pour les applications déployées dans les containers. En effet, l'utilisation de l'application et son architecture sont étroitement liées. On retrouve alors les deux grandes familles Monolithique et micro-service.

On peut toutefois remarquer que LXC/LXD a été conçu dans une problématique de compatibilité des applications avec le système hôte.

Singularity et Shifter sont des solutions optimisées pour les HPC, Singularity est d'ailleurs très utilisé pour les calculs scientifiques et pour sa gestion des ressources.

Pour les micro-services Docker et rocket sont très similaires, la différence étant principalement commerciale. Ils consistent tous deux à créer un réseau de container qui constitue l'application. Ces solutions sont donc principalement utilisées pour déployer des applications réunissant différents services et non des calculs, qui généralement sont parallélisés mais sans dépendance externe.

- Position sur le marché

Ce point est important aussi dans le cas présent, il s'agit ici de déterminer la solution la plus pérenne et la plus universelle afin de ne pas devoir recommencer en fonction de l'évolution du marché. Sur ce point là Docker semble être le mieux positionné. Effectivement, une image Docker est compatible avec les autres solutions. Par sa popularité, c'est l'une des solutions la plus documentée avec de nombreuses formations disponibles à travers différents supports. De plus c'est une solution soutenue par de grands groupes ce qui lui assure une certaine pérennité. Docker est de plus en plus pointé du doigt pour ses failles de sécurité, notamment l'obligation d'être utilisateur root, et subit la concurrence de solutions libres (Docker est libre aussi sur son développement, mais tout l'univers autour est payant et très fermé) comme Rocket.

Rocket quant à lui bénéficie de la concurrence que se livre Google et Docker. Google soutient le projet Rocket et produit l'orchestrateur Kubernetes. Rocket par ses soutiens peut rapidement devenir lui aussi un standard. Rocket attire par sa modularité et sa liberté.

Par sa position sur le marché Singularity confirme sa position d'outil scientifique, principalement soutenu par des entreprises proches de la recherche et les fournisseurs HPC.

LXC/LXD bénéficie de son ancienneté et se positionne en tant qu'utilitaire système plus que comme outil d'optimisation.

- Documentation/Complexité d'utilisation

Plus une solution est populaire plus elle est documentée augmentant ainsi sa facilité d'utilisation. Sur ce point, une fois de plus, Docker se positionne très bien avec de nombreuses ressources (tutoriel, documentation officiel…) bien que la plupart des documentations officielles soient payantes. Singularity, par son côté scientifique, dispose de bonnes documentations, généralement fournies par les HPC et les entreprises de la recherche comme Intel.

Les solutions les plus complexes s'avèrent donc être Rocket et LXC/LXD qui sont des projets libres et très peu documentés. De plus les communautés qui s'y intéressent sont trop peu importantes pour que des formations aient lieu.

- Orchestration

Dernier point de cette comparaison, il n'en est pas moins important: l'orchestration. Elle permet de gérer les containers. L'orchestration est particulièrement utile dans les architectures en micro-service, c'est l'orchestration qui va assurer une bonne communication entre les différents containers qui composent l'application.

Mais l'orchestrateur sert aussi à la gestion des ressources dans les processus parallélisés, il trouve donc une utilité pour les solutions du type Singularity dans les calculs haute performance.

Kubernetes s'impose par sa grande compatibilité avec toutes les solutions, seul Docker propose une alternative, Docker swarm, mais reste compatible Kubernetes.

Par cette analyse, on peut souligner l'importance de Docker sur le marché.

De plus dans le cas de PHIS, la meilleure solution de conteneurisation serait en micro-service. Comme dit précédemment PHIS se compose de plusieurs services de bases de données un web service et un ou plusieurs clients.

La solution de conteneurisation retenue est Docker. En effet, en plus d'être la solution la plus universelle, avec la compatibilité des images, elle est aussi la solution la plus rapide à mettre en place.

2.5.4. Orchestration

L'orchestration est l'outil le plus important pour la phase de production. C'est lui qui va permettre non seulement l'automatisation du déploiement sur les serveurs mais aussi lui qui va veiller au maintien des différents services déployés.

A l'image d'un script l'orchestrateur se présente sous forme de fichiers décrivant les tâches que doit effectuer l'orchestrateur et les conditions dans lesquels elles doivent s'appliquer.

Ainsi l'orchestrateur peu s'occuper de vérifier la disponibilité des services déployé dans les conteneurs. Cette vérification peut se faire autant au niveau de la présence du conteneur que au niveau de du bon déroulement du service dans le conteneur. En cas de problème l'orchestrateur sait quoi faire, arrêter/relancer le conteneur, déploiement sur une autre VM du cluster...

Outre le maintien qui est néanmoins la plus grande utilité de l'orchestrateur, l'orchestrateur trouve aussi son rôle lors du déploiement. Il est possible par son biais d'automatiser le déploiement en imposant des conditions sur la machine de déploiement de tel ou tel conteneur.

J'ai étudié 3 outils orchestration compatibles avec les containers Docker: :

- des scripts
- Kubernetes
- Swarm

- Les scripts:

Les scripts consistent simplement à automatiser le déploiement à l'aide d'instructions shell. C'est un outil simple, facile à mettre en oeuvre. Par contre il ne permet pas d'assurer une continuité de services en cas de dysfonctionnement d'un container. En effet utiliser un script pour le maintien du réseau de conteneur consistera à créer des fonctions de tests d'existence et de vie sur les conteneurs mais aussi créer des fonctions afin de rétablir le réseau en cas de problème sur l'un des conteneurs. La création de fonctions ne présente aucun avantage par rapport à l'utilisation des outils déjà existant tel que Kubernetes et Swarm.

- Kubernetes:

Kubernetes est un orchestrateur développé par Google. il est largement le plus utilisé sur le marché. Il laisse à l'utilisateur une grande liberté sur la gestion de son réseau de conteneurs tout en proposant de nombreuses fonctions pré-codées. Il présente aussi de nombreux autres avantages, comme sa compatibilité avec tous les types de conteneur qui existent ainsi que sa robustesse.

Kubernetes, est donc un outil très pratique pour l'orchestration mais sa mise en place est compliquée. Bien qu'il existe de nombreux outils annexes développés dans le but de simplifier cette mise en place, tel que Minikube, pour pouvoir bénéficier de tous les avantages de Kubernetes, notamment sa robustesse, l'infrastructure à mettre en place est complexe et grossit très rapidement avec le nombre de conteneurs à gérer.

- Swarm

Swarm est l'orchestrateur développé par Docker, il est uniquement dédié à la gestion de conteneurs Docker. Il est très simple et il est très rapide à mettre en place il permet une utilisation complète des outils fournis par Docker. Ses principaux défauts sont le manque de compatibilité et le contrôle sur ce que fait et doit faire l'utilisateur. En effet, Swarm limite beaucoup plus les possibilités de l'utilisateur.

2.6. Choix de l'architecture

2.6.1. Analyse des conteneurs

La première étape de la conteneurisation consiste à identifier tous les services à conteneriser. Dans PHIS, on identifie rapidement 5 services:

- PostgreSQL
- MongoDB
- rdf4j
- web service
- serveur web

RDF4J est avant tout un service qui se déploie avec le même service que le web service de PHIS. Ainsi RDF4J et le web service de PHIS utilisent un seul et même conteneur qui fournira le service Tomcat.

A cette analyse rapide viennent s'ajouter les services secondaires mais nécessaires tels que le service sftp qui permet de faire transiter les fichiers et documents uploader par le web service vers la base mongo.

Cela fait donc 5 services différents à conteneriser.

- PostgreSQL:
  Le conteneur du service postgres nécessite une ouverture du port 5432 ainsi que PostGis pour les besoins de PHIS.

- MongoDB:
  Le conteneur du service mongo nécessite une ouverture du port 27017.

- sftp:
  Le conteneur sftp a besoin d'ouvrir le port 22.

- Tomcat:
  Le conteneur Tomcat a besoin d'ouvrir le port 8080.

- Apache HTTP:
  Le conteneur apache nécessite l'ouverture du port 80

2.6.2. Organisation des conteneurs

Une fois les conteneurs identifiés et les images créées, le déploiement d'une application à l'aide des outils Docker ressemble à un puzzle, l'important étant alors de gérer et garantir la connexion entre les différents conteneurs.

De plus, il est important de prendre en compte que le stockage dans un conteneur n'est pas persistant lors de la disparition du conteneur. Pour cela, il est possible de monter des volumes, stockés physiquement sur la machine hôte, dans le conteneur. Les données stockées dans ce volume persistent même après la disparition du conteneur. Un même volume peut être monter dans plusieurs conteneurs, il peut ainsi servir de zone d'échange entre ces derniers.

Plusieurs architectures sont envisageables en fonction des besoins:

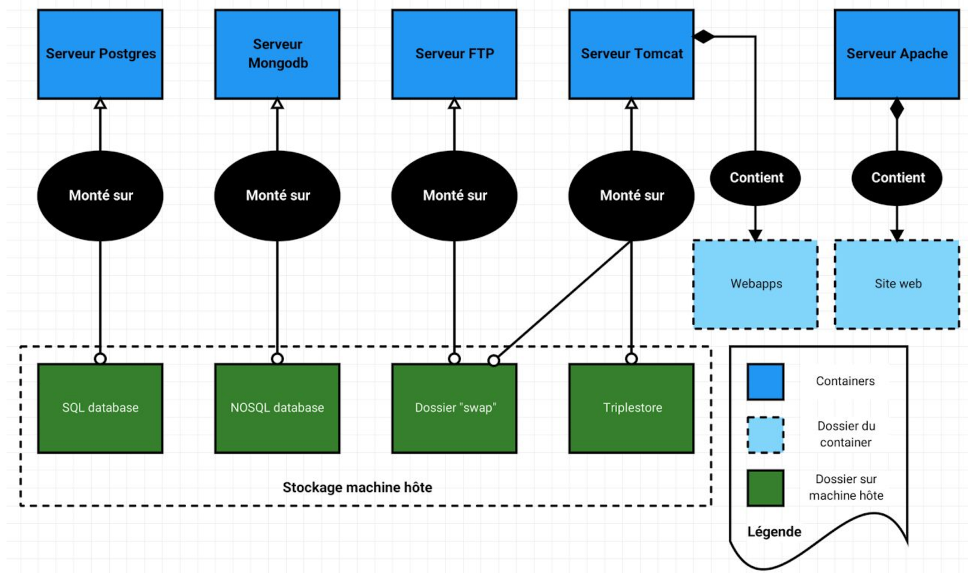

Figure 5: Position 1 de PHIS dans les containers

Dans le cadre d'un déploiement rapide, lors de l'exécution d'un seul script, cette architecture s'avère la plus adaptée. L'ensemble des exécutables sont contenus dans les conteneurs.
Pour l'utilisateur : 1 conteneur = 1 service DE PHIS et non 1 conteneur = 1 service utilisé PAR PHIS.

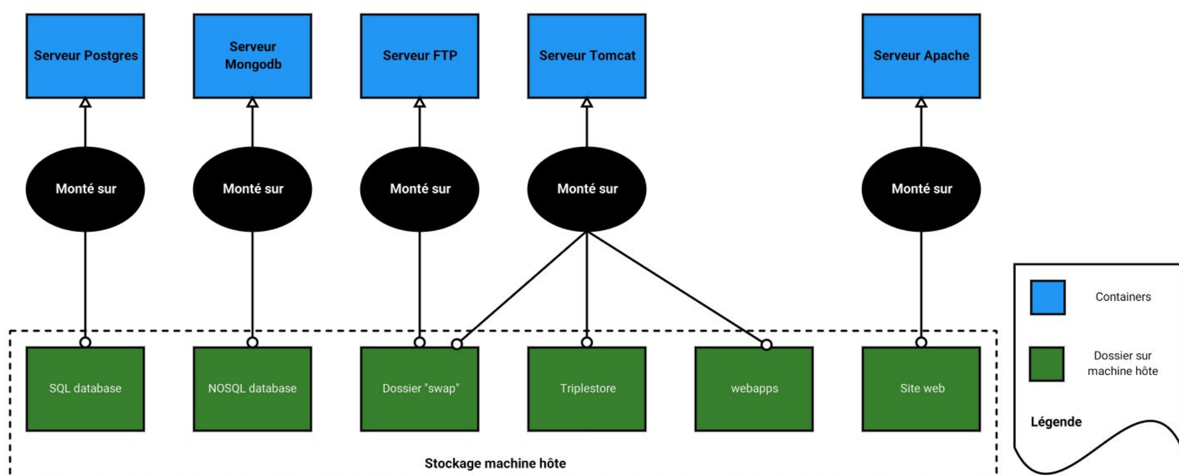

Figure 6: Position 2 de PHIS dans les containers

Au contraire cette architecture contenerisé uniquement les services Postgres - Mongo - SFTP - Tomcat - Apache. PHIS se trouve alors sur la machine hôte et est déployé par les services contenerisés par le biais de montage au sein de ces derniers.

2.6.3. Architecture en local / sur cloud

Local:

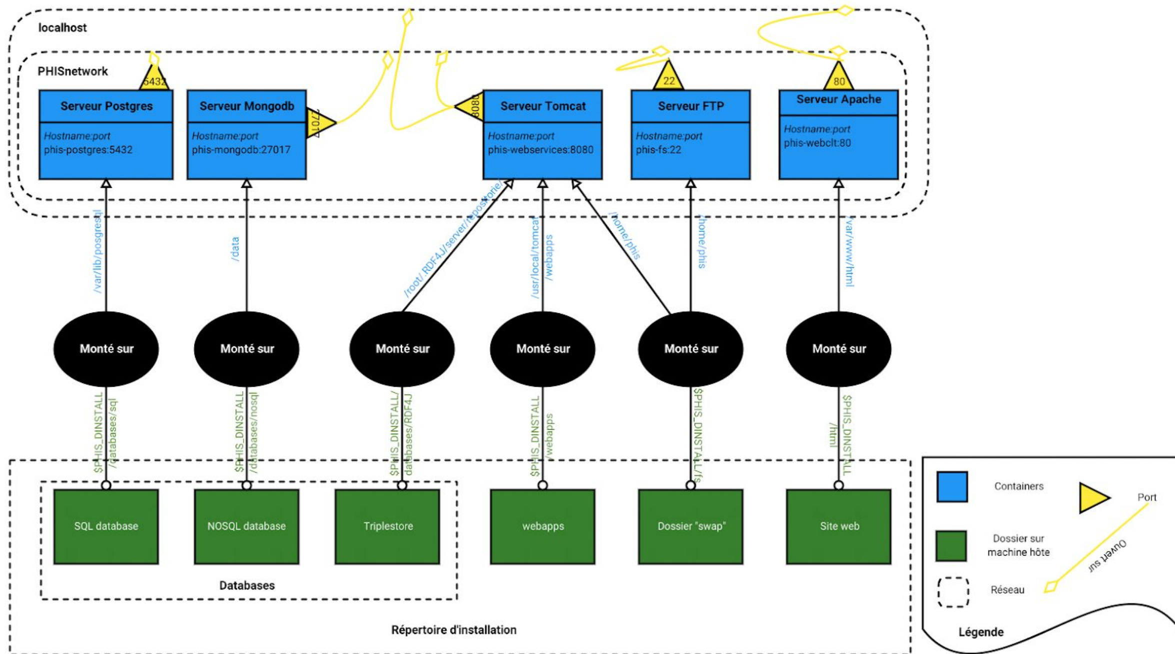

Figure 7: Architecture containers PHIS Locale

Ainsi en créant un sous réseau pour les conteneurs PHIS, les communications entre les conteneurs se font dans ce sous réseau. L'utilisation de PHIS se fait par la connexion à ce sous-réseau évitant ainsi d'éventuel conflit pour un port déjà utilisé (postgres déjà installé etc…).

Cloud:

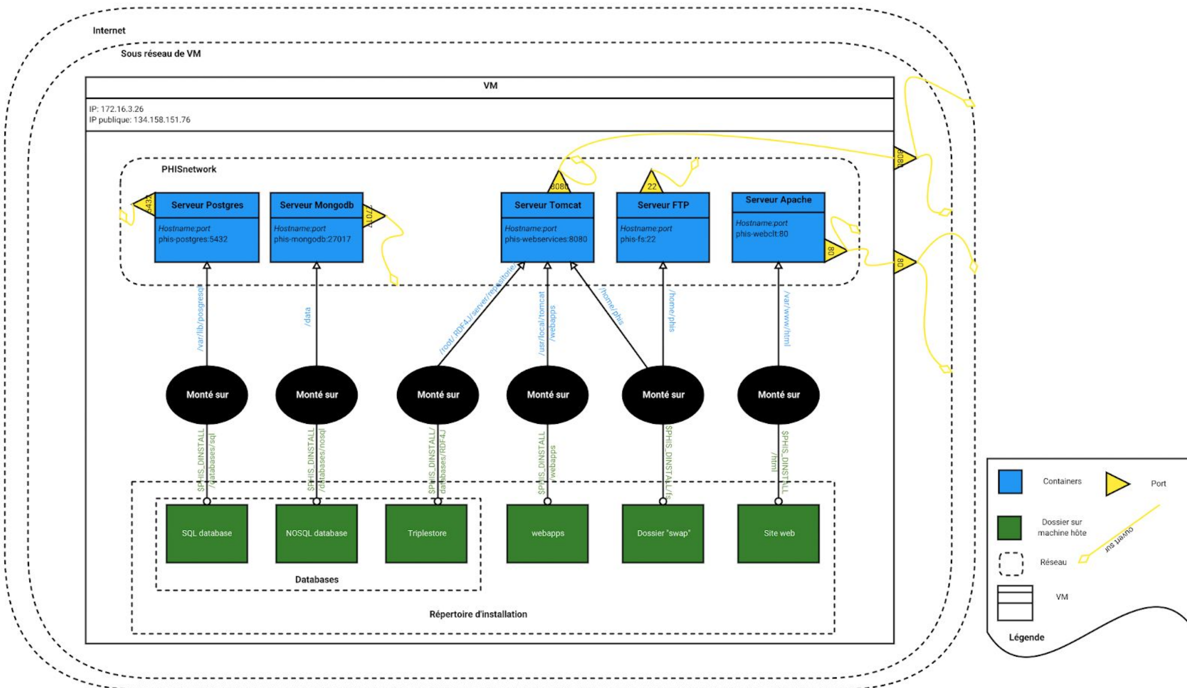

Figure 8: Architecture containers PHIS 1 VM

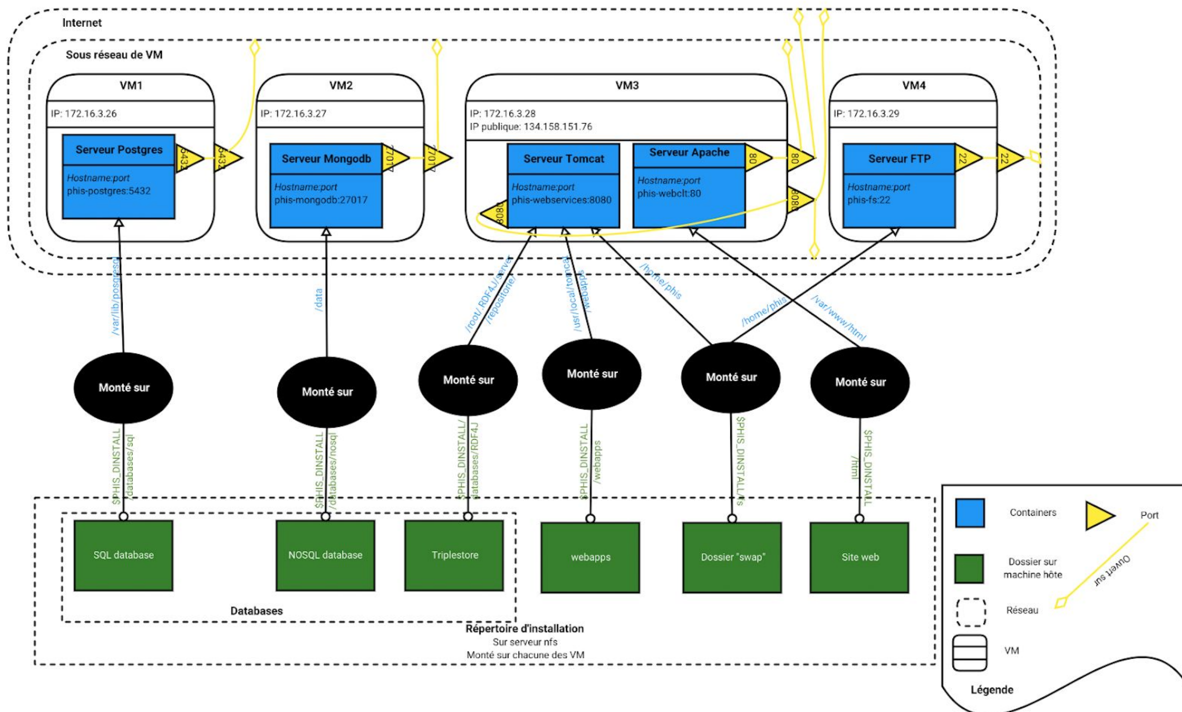

Figure 9: PHIS sur cloud sur plusieurs VM - sans orchestrateur

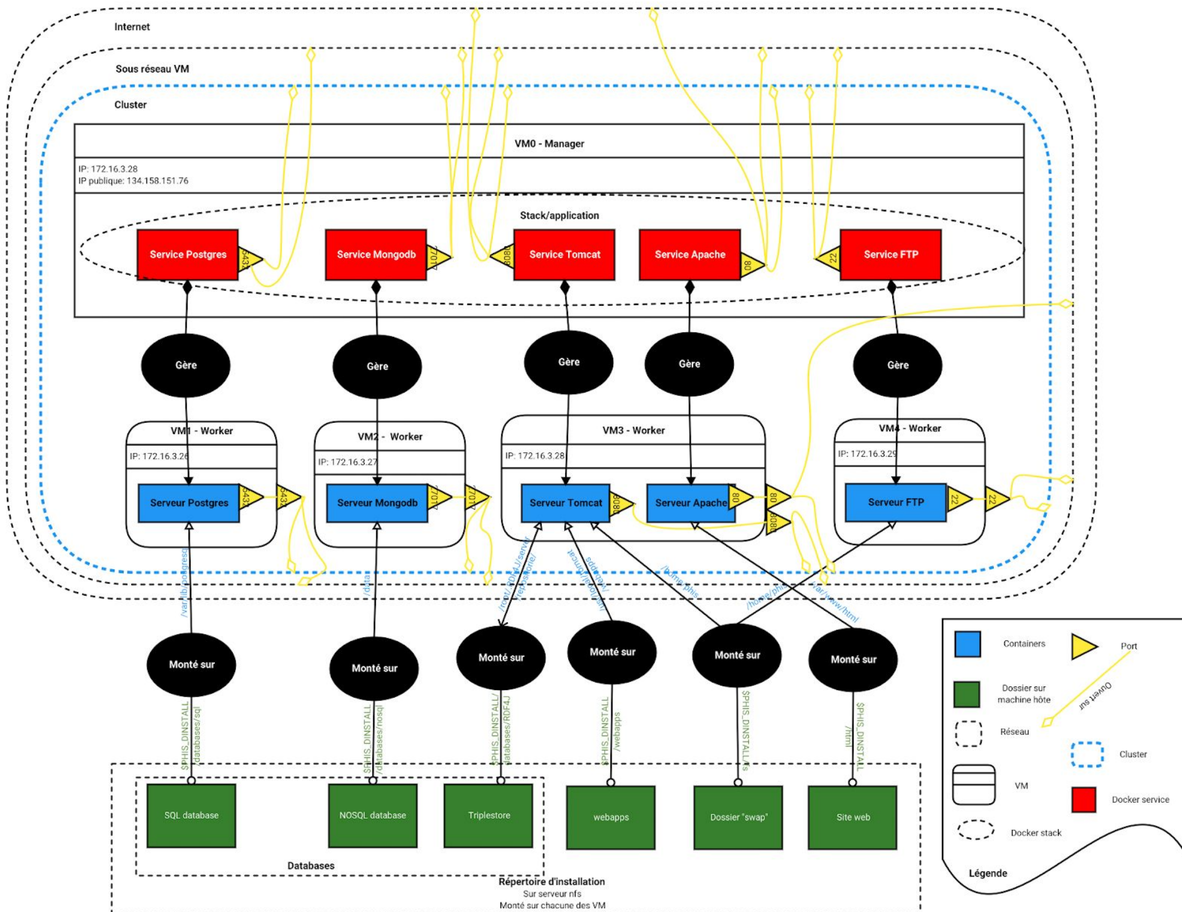

Figure 10: PHIS sur cloud sur plusieurs VM - avec orchestrateurs

Les deux architectures précédentes sont un déploiement de PHIS sur le Cloud en utilisant plusieurs VM. L'utilisation de VM formant un sous-réseau permet de n'ouvrir que 2 ports (80 pour le serveur web et 8080 pour l'application web) d'une seule VM pour que PHIS soit accessible.

3. Rapport technique

3.1. Docker

Docker fournit un ensemble de logiciels et de services tous utilisés dans la conteneurisation. L'écosystème Docker va de la construction d'une image conteneur à l'orchestration de conteneur en passant par le stockage de projet en ligne sur le Dockerhub.

3.1.1. Dockerfile

Le Dockerfile est un format de fichier utilisé par Docker pour construire une image de conteneur. C'est un fichier texte qui sera interprété par l'outil build de Docker lors de la construction de l'image.
Le Dockerfile s'écrit en suivant une règle très simple:

- INSTRUCTION argument : dans un Dockerfile une ligne non vide commence soit par # marquant alors le commentaire, soit par une instruction écrite en majuscule.

Exemple d'un Dockerfile :

```
FROM Ubuntu:16.04

COPY ./init_files/index.html /var/www/html/index.html

RUN ["apt-get","update"]
RUN ["apt-get","install",“-y”,"apache2"]
RUN ["chown","-R","www-data:www-data","/var/www/html"]
RUN ["chown","-R","755","/var/www/html"]

EXPOSE 80

ENTRYPOINT ["apachectl","-k","start","-DFOREGROUND"]
```

Les instructions:

- FROM :
  - indique l'image sur laquelle se base la nouvelle image
  - format: FROM <nom de l'image>
- COPY :
  - copie des fichiers depuis un fichier de l'OS hôte vers l'image
  - format: COPY <chemin relatif fichier à copier> <chemin absolu destination du fichier>
- RUN :
  - effectue une commande, compatible avec l'image d'origine, au moment de la construction de l'image
  - format: RUN [“commande”,“un”,“argument”,“par”,“case”] (format recommandé) ou RUN commande à effectuer (ancien format)

- ENTRYPOINT :
  - commande à effectuer au démarrage du container
  - format: idem RUN

- EXPOSE :
  - expose/ouvre un port du container
  - format: EXPOSE <n° port>

- ENV :
  - définit une variable d'environnement dans l'image
  - format: ENV <NOM_VARIABLE> <VALEUR_PAR_DEFAUT> #la valeur peut être vide, la variable est alors nulle

- VOLUME :
  - définit un dossier comme un point de montage d'un volume externe
  - format: VOLUME chemin/absolu/du/dossier

Il existe de nombreuses autres instructions, leurs descriptions sont disponibles sur le site de Docker https://docs.Docker.com/develop/develop-images/Dockerfile_best-practices/.

Informations complémentaires:

Le <nom de l'image> répond aux formats suivants:
→ <dossier_hub>/<nom de l'image>:<version> qui fait alors référence à une image sur le hub, avec:
<dossier hub>: le nom du dossier du Docker hub dans lequel se trouve l'image: généralement le nom d'utilisateur Docker du créateur
<nom de l'image>: nom de l'image sur laquelle se base le Dockerfile, dans l'exemple Ubuntu
<version>: la version de l'image, ne pas renseigner la version est équivalent à renseigner latest, ce qui correspond à la dernière version de l'image mise en ligne.
→ <nom de l'image>:<version> qui fait alors référence à une image présente sur le système hôte (obtenue grâce à Docker pull ou en la construisant) ou à une image officielle du Docker hub. De même la version peut ne pas être renseignée, donnant alors la dernière version de l'image.

Il ne peut y avoir qu'un seul ENTRYPOINT par conteneur, en effet l'entrypoint doit correspondre à un service qui reste actif indéfiniment, il doit donc correspondre au lancement du service pour lequel le conteneur est construit, dans l'exemple un serveur Apache.

Une fois le Dockerfile écrit il est nécessaire d'utiliser l'outil build de Docker afin de construire l'image:

Commande build stockage local:
Docker build (--no-cache) -t <nom_image>:<version> <chemin_Dockerfile>

Cette commande construit et stocke l'image dans le répertoire local de Docker sous le nom <nom_image>:<version>

Commande build stockage hub:
Docker build (--no-cache) -t <nom_utilisateur_Docker>/<nom_image>:<version> <chemin_Dockerfile>

Docker push <nom_utilisateur_Docker>/<nom_image>:<version>

Comme précédemment la première commande stocke l'image dans le répertoire local de Docker. La deuxième commande sert à télécharger cette image sur le hub Docker de l'utilisateur.

3.1.2. Gestion de conteneurs en réseau local

Les outils Docker permettent de lancer et gérer des conteneurs sur une même machine physique ou non.
Les conteneurs lancés par Docker sur une même machine appartiennent à un sous-réseau créé automatiquement ou par l'utilisateur, Docker fournit différents outils pour gérer ce réseau.

- Docker run:

Docker run -d nom_utilisateur_Docker>/<nom_image>:<version>

Cette commande construit un container à partir de l'image renseignée. Le container est lancé en daemon. Dans la cas où aucun nom d'utilisateur Docker n'est renseigné alors l'image est soit sur le dépôt officiel Docker soit présente localement.

Les options:
-d → container en daemon
-i →intéractif
-t → allocation d'un TTY
-it → option -t et -i cumulé = container intéractif, avec un TTY réservé
--rm → container supprimé une fois son exécution terminée
--name → permet de nommer le container
-v, --volume <dossier_host>:<point_de_montage_container> → monter un dossier de la machine hôte en lieu et place d'un dossier du container
-p <port_host>:<port_container> → connecte un port du container sur un port de la machine hote
--ip <adresse_ip> → attribue une adresse ip au container
--hostname <nom> → attribue un nom reconnu par les autres containers du sous-réseau (par défaut nom du container)

--network <nom network> → le container appartient à un sous réseau défini
<cmd> → ajouter une commande à la fin de la commande run permet que celle ci soit exécutée en tant que ENTRYPOINT

- Docker exec

Docker exec <-d -i -t> <nom_container> <cmd>

La commande exec permet d'effectuer une commande dans un container en cours de fonctionnement, de même que pour run elle possède les options -i -d et -t respectivement intéractive, daemon, TTY.

- Docker stop/start

Docker stop <nom_container>

La commande permet l'arrêt d'un container

Docker start <-i -d (-t)> <nom_container>

La commande permet de relancer un container déjà exécuté, au redémarrage le container effectue à nouveau l'ENTRYPOINT.

- Docker ps

Docker ps -a

Liste des containers existant sur la machine

- Docker network

Docker network create <option> <name>

La commande crée un sous réseau dont le subnet, la passerelle réseau (gateway) … peut être attribué par l'utilisateur grâce aux options de la commande. La création d'un sous-réseau permet à l'utilisateur d'être maître sur l'adressage des conteneurs mais aussi permet de faciliter la communication entre les conteneur de ce sous réseau, apportant ainsi des avantages de sécurité et de gestion.

Les options:
--subnet : désigne le segment d'ip disponible pour ce sous-réseau
--gateway
https://docs.Docker.com/engine/reference/commandline/network_create/#options

Les sous réseaux Docker se gèrent avec des commandes utilitaire de la commande ***Docker network***. Comme vu précédemment la création se fait à l'aide de la commande ***create***, de nombreuses commandes existent comme ***ls*** pour lister les réseaux Docker présents sur la machine, ***rm*** pour supprimer un réseau etc…
https://docs.Docker.com/engine/reference/commandline/network/

Ces commandes servent à la création et à la gestion de conteneurs présents sur une même machine ou plutôt appartenant à un même service Docker. Effectivement, il est possible de créer un serveur Docker tel que tous les conteneurs puissent être gérés depuis ce serveur (même s'ils sont lancés sur différentes machines).

3.2. Déploiement sur un environnement de type cloud

PHIS n'est pas destiné à un déploiement local mais à un déploiement sur des infrastructures en ligne du type Cloud.
Afin de pouvoir tester la portabilité de mes réalisations Docker et de mettre en place une procédure de déploiement de PHIS sur cloud il a été mis à ma disposition un accès aux serveurs France Grille ainsi qu'aux serveurs EGI.

J'ai tout d'abord créer des VM afin de pouvoir déployer PHIS sur celles-ci, d'abord manuellement, puis de manière automatisée.

Comme vu lors de l'analyse des architectures possibles, il y a plusieurs architectures possibles pour PHIS sur le cloud. L'objectif étant de permettre une orchestration des conteneurs, il est conseillé pour la robustesse de l'application déployée d'utiliser plusieurs machines organisées en Cluster.
Pour des raisons de simplicité de mise en place et de gain de temps, il sera question d'un cluster Swarm.

3.2.1. Gestion de conteneur sur cloud

Il est question ici d'utiliser un cluster Swarm et par conséquent les outils de l'orchestrateur du même nom.

L'architecture minimale est une machine (VM ou physique) désignée alors comme la machine manager. L'architecture minimale pour un cluster n'apporte pas grand chose par rapport à un déploiement classique sur une seule machine qui ne serait pas attribuée à un cluster, si ce n'est la possibilité d'utiliser les outils de gestion automatique des conteneurs proposés par Swarm.

Une autre architecture possible est alors une machine manager, et une ou plusieurs machines ditent workers, ainsi les outils de l'orchestrateur permettent à la VM manager de lancer et gérer des conteneurs sur les VM worker, la présence de plusieurs workers est alors conseillée pour la robustesse de l'application. Effectivement disposer de plusieurs workers permet de garder l'application disponible même en cas de "chute" de l'une des VM worker.

Sur la même idée que les workers, l'architecture conseillée est plusieurs machines managers et plusieurs machines workers ainsi l'application déployé reste disponible quelque soit la machine rencontrant un problème.

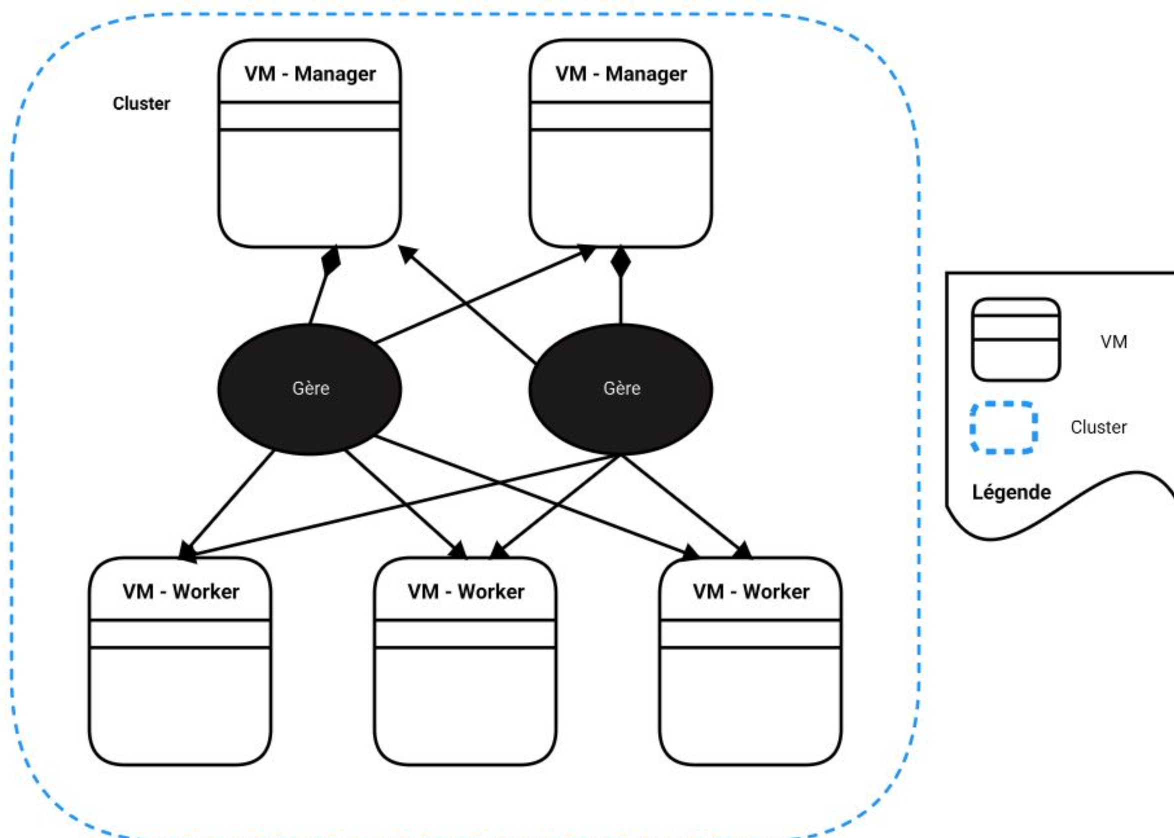

Figure 11: Architecture d'un cluster

*NB: A l'initialisation du cluster une seule machine est désignée comme manager, toutes les autres machines rejoignent le cluster en tant que worker, on utilise par la suite la commande **Docker node promote** pour promouvoir une machine worker en machine manager.*

Sur le manager exécuter:

Docker swarm init --advertise-addr <ip_VM_Manager>

Sur les workers:

Docker swarm join --token <token_fournis_à_l'initialisation_du_manager> <ip_manager>:2377

- Docker node

Node est l'outil Docker qui permet la gestion des noeuds d'un cluster. On retrouve les commandes classiques de gestion comme ***Docker node ls, rm, ps*** … c'est cet outil qui permet de désigner un noeud en tant que manager ou au contraire transformer un noeud manager en noeud worker grâce aux commandes ***promote*** et ***demote.***

- Docker service

Un service Docker est un outil qui peut être utilisé uniquement dans un cluster, sur le noeud manager.
Un service se construit de manière très similaire à un container.
Un service est un outil qui va lancer des containers sur les différents noeuds du cluster. À partir des options renseignées lors du ***Docker service create*** il va créer des containers.
Un service nécessite donc toutes les options d'un container mais aussi des options sur les réplicas de containers à créer, comme: le nombre de réplicas, les conditions de lancement/déploiement de ces réplicas…

Docker service create:

Docker service create <options_similaire_à_run> <--replicas> <image_Docker>

Cette commande permet de créer un service pour la gestion de containers, tous créés à partir de l'image et des options passées en paramètres.

Docker service update:

Cette commande permet de faire évoluer l'ensemble des conteneurs gérés par ce service: ajout d'options, retrait d'options, adaptation du nombre de réplicas…

3.2.2. Orchestration

Dans l'objectif de rester compatible avec un éventuel cluster Kubernetes, l'orchestration réalisée utilise un fichier Docker "compose" décrivant les services déployés sur le cluster.
Ce fichier Docker "compose" s'utilise avec les outils Docker, à la seule condition d'être sur un cluster qu'il soit swarm ou kubernetes. Ainsi bien que n'exploitant pas totalement les capacités disponibles lors d'un déploiement sur un cluster Kubernetes, cela permet de ne pas rendre la réalisation complètement dépendante à un cluster Swarm.

- fichier Docker "compose"

Un fichier Docker "compose" utilise le format yaml. Il permet à l'aide d'instructions de décrire des containers et des services Docker.

Les instructions utilisées sont directement liées aux arguments disponibles lors de l'utilisation de la commande Docker run ou Docker service create, ainsi le Docker "compose" est une mise en page de l'architecture de l'application.

Dans le cadre de l'orchestration, l'instruction la plus utile est *deploy* qui avec d'autres sous-instructions permet de gérer les réplicas de conteneurs: notamment le nombre de réplicas pour chacun des services, les conditions de redémarrage des conteneurs en cas d'arrêt de ces derniers, les conditions de déploiement des containers (par exemple imposer qu'un conteneur soit lancé uniquement sur les noeuds de type manager), ou encore des conditions de ressource comme la mémoire disponible sur le noeud etc…

Une fois le fichier "Docker compose" écrit, dans les conditions de PHIS, il sera utilisé en combinaison de l'outil ***Docker stack*** qui permet de définir un pile de conteneurs/services comme formant une application. Ainsi l'utilisation d'une pile permet de rassembler tous les conteneurs au sein d'un même sous-réseau (comme l'utilisation d'un ***Docker network*** dans un déploiement local), de plus la suppression de tous les conteneurs est simplifiée par la suppression de la pile, aussi on trouvera l'avantage sur la gestion des conteneurs et des services avec les commandes ***ls ps*** et ***services***. La pile vient compléter la portabilité des travaux réalisés sur un cluster Kubernetes avec la possibilité de prendre en compte des fichiers de configuration Kubernetes.

4. Résultats

4.1. Containerisation du système d'informations

Mon premier travail sur le déploiement de PHIS par container a été de conteneriser tout le système d'information avec un conteneur par service.

J'ai créé 5 Dockerfile décrivant 5 images Dockers à savoir:

- PostgreSQL:

L'image se base sur l'image Docker officielle postgres qui se trouve sur le Dockerhub officiel. L'adaptation de cette dernière à PHIS consiste alors en une installation de PostGIS.

- MongoDB:

Il existe une image officielle mongodb, mais afin de réellement adapter le container aux besoins de PHIS, l'image est construite à partir d'une image Ubuntu 16.04 sur laquelle est installée MongoDB, configurée pour une utilisation avec PHIS, lors du build de l'image.
J'ai choisi la distribution Ubuntu 16.04 car c'est la distribution utilisée par les développeurs du système d'informations PHIS.

- Sftp:

Il n'existe pas d'image officielle Docker d'un service sftp, afin d'éviter d'éventuels problèmes avec l'utilisation d'images amateurs, il est conseillé de construire soi-même l'image container. De même que pour MongoDB l'image se base sur Ubuntu 16.04 sur laquelle il est installé, à l'aide du Dockerfile, un serveur sftp configuré pour les besoins de PHIS.

- Tomcat:

De même que pour MongoDB une image officielle Tomcat existe mais afin de mieux contrôler la configuration de Tomcat et son adaptation à PHIS, j'ai favorisé la construction d'une image Docker basée sur un Ubuntu 16.04 sur laquelle le Dockerfile installe et configure un serveur Tomcat.

- Apache HTTP:

Il existe aussi une image officielle Apache HTTP, mais toujours dans l'optique d'un meilleur contrôle de l'installation et de la configuration de ce dernier j'ai choisi de construire une image Docker à partir de Ubuntu 16.04 en installant Apache, PHP 7.0 et les autres dépendances du serveur web de PHIS.

En plus de réaliser un Dockerfile par service il est important de souligner la nécessité d'isoler ces Dockerfiles dans un dossier différent pour chacun d'entre eux afin de mieux organiser les fichiers à placer dans l'image Docker lors du build.

Les Dockerfiles de ces images sont disponibles dans les annexes.

4.2. Déploiement en local

Une fois le système contenerisé, l'objectif est d'automatiser son déploiement en local.
Pour cela j'ai organisé tous les fichiers nécessaires au déploiement (Dockerfile pour la construction des images, fichiers d'initialisation, web service de PHIS, client web…) dans un dossier. Cette organisation me permet non seulement de faciliter la maintenance de PHIS et son évolution par quelqu'un n'ayant pas travaillé sur Docker, mais aussi de réaliser un script, disponible en annexe, de déploiement de PHIS en local.

Ce script se décompose en 4 étapes:

- La construction du système de fichiers nécessaires au réseau de conteneurs à savoir:
  un dossier d'installation de PHIS qui contient les bases de données, les fichiers logs (pour faciliter l'analyse d'erreur en cas de problème), les dossiers racines des applications webs de Tomcat et du site web pour apache afin de faciliter l'évolution de PHIS sans nécessairement reconstruire les images et recommencer le déploiement.

- La construction des images Docker:
  Cela permet de faire facilement évoluer les images, et évite la dépendance au Dockerhub ou à la présence des images sur la machine hôte.

- La création d'un sous réseau afin d'isoler les conteneurs du reste de la configuration réseau de l'hôte pour éviter d'éventuels conflits sur l'utilisation des ports ou autre.

- Le lancement des conteneurs avec le montages des volumes, l'attribution d'ip, l'association des ports (pas vraiment nécessaire pour une utilisation en local), …

4.3. Déploiement sur cloud et orchestration

De même que pour un déploiement local l'objectif est d'automatiser et simplifier le déploiement de PHIS sur le cloud. Les images Dockers restent les mêmes que pour le déploiement local.
Ainsi pour le déploiement de PHIS sur une seule machine sans orchestrateur, le déploiement se fait par un script très similaire au script d'un déploiement local, à la différence qu'il est nécessaire d'associer les ports du web service et du serveur web avec les ports de l'ip publique de la machine hôte, les autres ports restent fermés pour des raisons de sécurité, les conteneurs appartenant tous au même sous réseau les requêtes entre containers peuvent se faire sans l'ouverture des ports de la machine hôte.

Le travail le plus conséquent pour le cloud est d'automatiser le plus possible le déploiement de PHIS sur un réseau de machine. Pour cela, j'ai réalisé 7 scripts (en partie disponibles en annexe). La nécessité de ces nombreux scripts vient surtout de la volonté de la portabilité sur toutes les distributions Linux. Ainsi l'automatisation s'organise de la manière suivante:

- un script automatisant l'installation de Docker et sa configuration sur la machine destinée à être manager, uniquement compatible Ubuntu 16.04 et supérieur ainsi que les autres distributions basées sur cette dernière;
- un script automatisant l'installation et la configuration d'un serveur nfs pour la disponibilité du répertoire d'installation sur toutes les machines du réseau;
- un script pour l'installation de Docker à exécuter sur toutes les machines workers;

- un script d'installation et de configuration d'un client nfs à exécuter sur toutes les machines du réseau (excepté celle qui héberge le système de fichier);
- un script automatisant la construction et la mise à jour des images Docker sur le Docker Hub;
- un script de déploiement de PHIS, regroupant comme pour une installation locale la construction du système de fichiers, et le déploiement des conteneurs, mais dans ce cas ci il s'agit en réalité du déploiement d'une pile Docker de services, initialisée à partir du fichier Docker compose. Services qui vont par la suite déployer les conteneurs PHIS;
- un script regroupant les scripts précédents, par conséquent uniquement compatible Ubuntu.

Cette organisation permet de réellement automatiser/guider le déploiement de PHIS de A à Z sur un réseau de machines Ubuntu. Elle permet aussi de déployer PHIS sur des machines autre que Ubuntu mais dans ce cas là l'installation de Docker et de nfs n'est pas garantie par les scripts réalisés, l'utilisateur doit alors installer et initialiser Docker et nfs par lui-même et ensuite utiliser le script de déploiement qui est compatible avec toutes les distributions.

L'orchestration est décrite dans le fichier Docker compose (disponible en annexe), elle oblige la présence d'un réplica pour chacun des conteneurs, ainsi que l'obligation de l'existence de ce réplica. C'est à dire que quelque soit la raison d'arrêt du conteneur l'orchestrateur Swarm le relancera, garantissant ainsi la disponibilité de l'application.

Afin de vérifier le bon fonctionnement du déploiement, j'ai réalisé un ensemble de tests qui sont:

- ➔ l'ajout de l'instruction HEALTHCHECK CMD aux Dockerfiles des images conteneurs, cette instruction s'utilise avec une commande qui sera exécutée tous les x temps (paramétrable avec des arguments comme --interval=5s de l'instruction) dans le conteneur. Un test est ensuite réalisé sur la valeur retour de la commande, en fonction du résultat du test un statut healthy ou unhealthy est attribué au conteneur. Par exemple dans le cas d'un test à l'aide de la commande curl, si curl retourne 200 le container se verra attribuer le statut healthy. Dans les autres cas unhealthy signale un problème avec le service déployé dans le conteneur. Ce statut est visible avec la commande Docker ps mais chaque changement de statut d'un conteneur génère un événement pouvant être utilisé par l'orchestrateur si présent.
- ➔ l'écriture de scripts permettant d'automatiser des tests sur le web service de PHIS et sur le client web de ce dernier. Ces scripts réalisent des appels aux web services via la commande curl pour ensuite vérifier le code retourné par curl, si c'est un code d'erreur il est retourné vers l'utilisateur qui devra vérifier les logs afin de déterminer la cause de l'erreur.
- ➔ la réalisation d'une procédure à effectuer afin de tester les différents services de PHIS. C'est une procédure qui guide l'utilisateur sur l'utilisation de l'interface sur l'application web de PHIS afin que ce dernier vérifie que tout est fonctionnel.

4.4. Documentation technique, formation utilisateur

Dans le but de faciliter la transmission de mon travail réalisé durant mon stage, j'ai réalisé des petites documentations (sous forme de fichier texte README) à propos de chacune de mes méthodes de déploiement afin de permettre à des personnes n'ayant pas travaillées sur le projet de l'utiliser, et de le faire évoluer.
En complément de cette documentation, j'ai réalisé deux formations auprès des informaticiens des unités MISTEA et du LEPSE (développeurs du système d'exploitations PHIS et administrateurs système). La première formation a eu lieu le 03 Août. C'est une formation qui s'est déroulée sur 3h avec 8 informaticiens (https://twitter.com/PHISphenomics/status/1025323122810146816) durant

lesquelles j'expliquais comment utiliser Docker et ses différents outils et aussi comment déployer PHIS à partir des outils que j'ai mis en place. La deuxième formation se déroulera fin Août.

5. Évolutions possibles

5.1. Sécurité

Docker n'est pas réputé pour la sécurité. De plus, du fait des volumes partagés entre plusieurs containers, le dossier d'installation est ouvert à tous les utilisateurs. Pour cela une recherche sur une configuration alternative pour l'accès au fichiers serait intéressante afin de garantir la sécurité des données. Je cherche actuellement à résoudre ce défaut, par l'utilisation de volumes Dockers à la place de dossiers, mon objectif serait de garder la propreté et la simplicité de l'arborescence des dossiers et fichiers stockés tout en garantissant la sécurité des données.

5.2. Replicas et shard MongoDB

MongoDB propose aussi une répartition de la charge par réplication du service, et même un système appelé Shard. C'est une méthode appréciée des administrateurs de bases de données NoSQL. C'est pourquoi il serait intéressant d'étudier une architecture de containers permettant l'utilisation des replicas MongoDB au sein d'un cluster orchestré.

L'architecture des réplications et des shards MongoDB est la suivante:

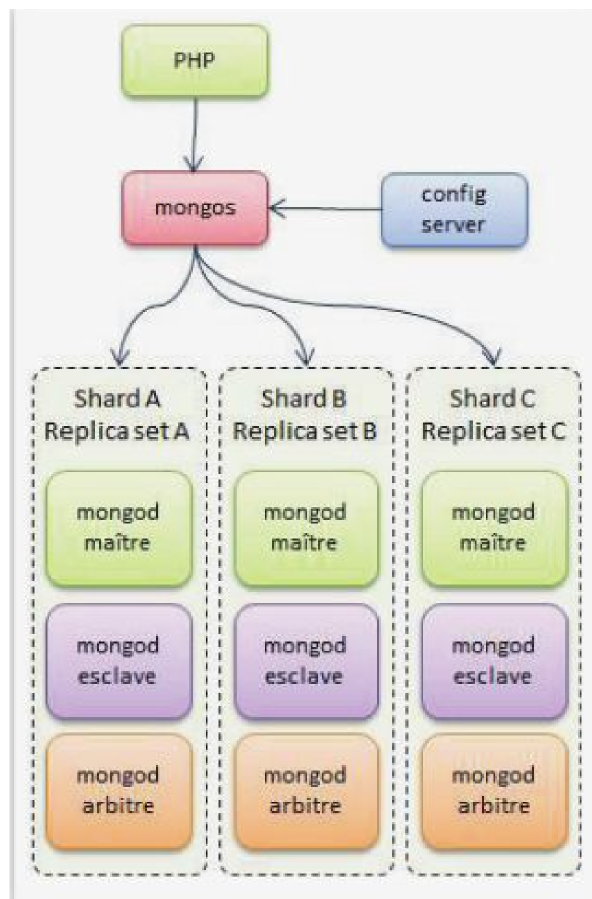

Figure 12: Shard MongoDB

L’architecture suivante semble intéressante à étudier/tester:

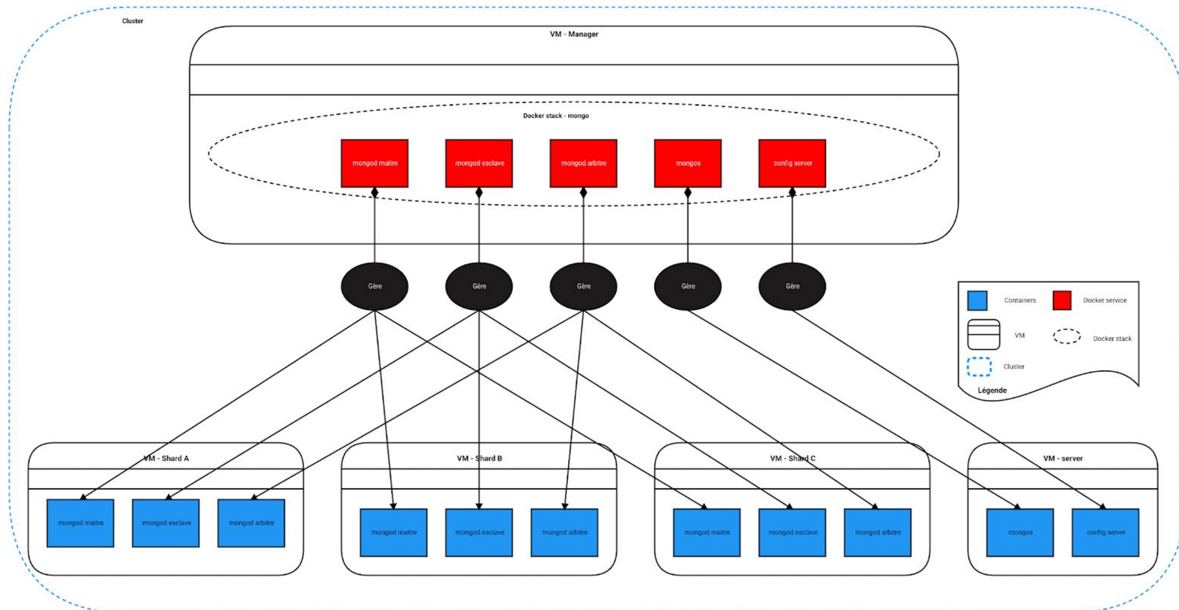

Figure 13: Architecture MongoDB shard avec Docker

Cette architecture présente l’avantage de déployer facilement autant de shards que l’on veut. En effet, il suffit ici d’avoir un nombre de VM dans le cluster égale au nombre de shards voulu + au moins 2 (une manager + une pour le server de configuration et le serveur mongos), renseigner à l’orchestrateur le nombre de réplicas voulu pour les services de la pile, ici mongod maître, mongod esclave et mongod arbitre doivent avoir tous trois un nombre de réplicas égal au nombre de shards voulu , avec comme condition de déploiement 1 seul container de chaque type par VM.

6. Gestion de projet

6.1. Démarche personnelle

Mon stage s'est déroulé dans deux bâtiments de l'INRA - Montpellier, les locaux de MISTEA qui regroupe les développeurs du projet PHIS et les locaux du LEPSE où je fus durant la majeur partie de mon stage et dans lequel se trouvent les administrateurs systèmes et bases de données.

Ainsi mon stage s'est divisé en deux périodes la première d'une durée d'environ 3 semaines durant laquelle je me trouvais dans les locaux de MISTEA afin de réaliser un premier déploiement de PHIS en bénéficiant de l'aide des développeurs en cas de difficultés. Durant cette période j'ai réalisé une documentation sur le déploiement, elle me fut bénéfique pour la suite, ayant plusieurs fois durant mon stage eu à refaire des configurations.
La deuxième période du mois de mai au mois d'août s'est effectuée dans les locaux du LEPSE. J'ai pu bénéficier de l'aide de l'administrateur système Vincent NEGRE lors de blocage sur des configurations ou des utilisations d'outils mis à ma disposition, notamment les accès aux clouds.

La mission qui m'a été confiée durant ce stage utilisait des technologies nouvelles pour l'équipe au sein de laquelle je me trouvais. En conséquence, j'ai bénéficié durant mon stage d'une période d'auto-formation et de recherche autour de ces technologies. Les décisions sur les technologies à utiliser se sont faites suite à ces recherches en discussion entre les différents encadrants et moi-même sous forme de bilan des recherches.

C'est aussi cet apport de nouvelles technologies qui justifie la réalisation de formations de l'équipe à ces technologies.

Pour le suivi des travaux, une réunion hebdomadaire est organisée par l'équipe de MISTEA afin que chacun des membres de l'équipe explique ses réalisations de la semaine. Un point, avec mes encadrants Vincent NEGRE, Malika NASSIF et Patrick MOREAU est organisé toutes les deux semaines en complément de la réunion MISTEA.

6.2. Planification

Prévisionnelle:

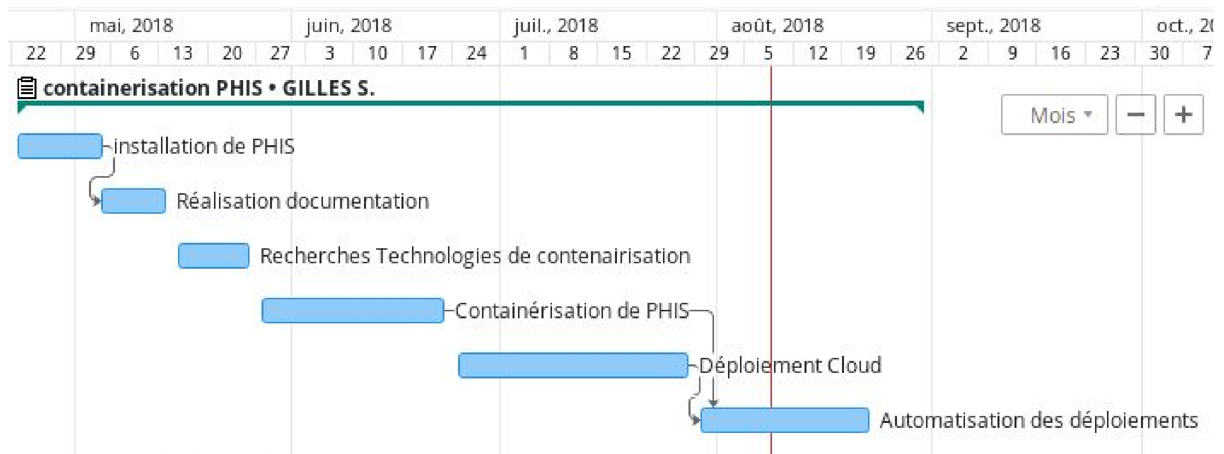

Figure 14: Planification prévisionnelle

État au 09 Août:

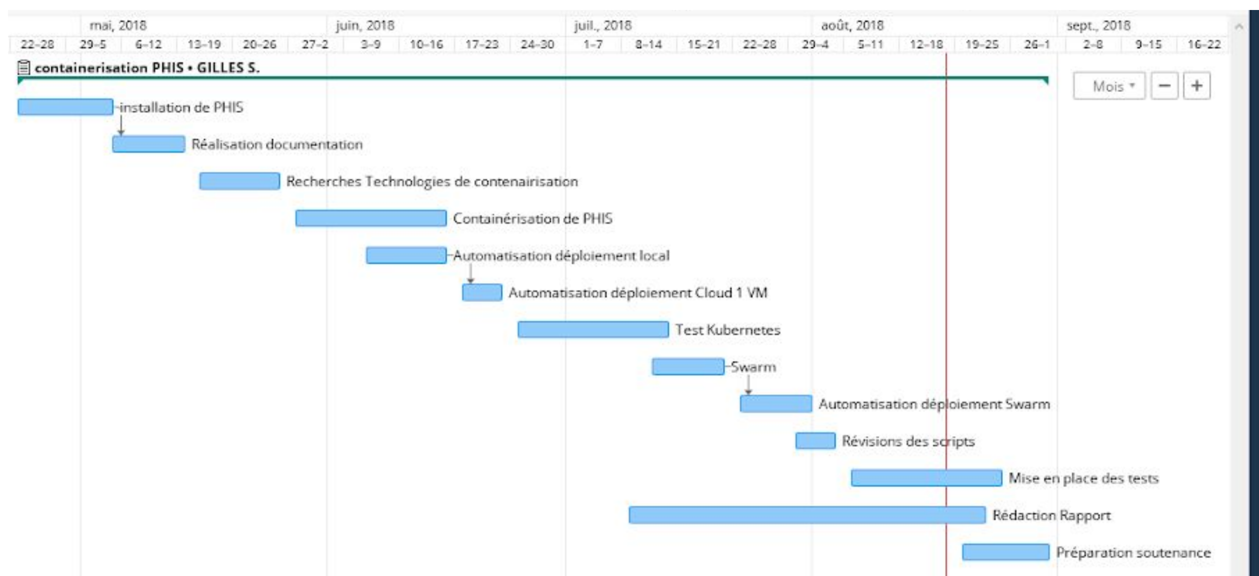

Figure 15: Planification réalisée

7. Conclusion

7.1 Gains personnels

Ce stage m'a apporté beaucoup de choses, allant de la découverte de la vie professionnelle à l'apprentissage de nombreuses technologies informatiques.
Ce stage m'a donc apporté beaucoup de connaissances sur l'utilisation de Docker et ses différents éléments mais aussi complété mon apprentissage d'outils comme GitHub et m'a permis de comprendre la configuration des différents logiciels utilisé comme Postgres, Mongo, Tomcat et Apache.

7.2 Technique

En conclusion sur le sujet du stage, je dirai que il reste encore beaucoup d'évolutions possibles sur les outils créés. Malgré tout ces outils sont créés et fonctionnels.
PHIS peut être désormais déployé en 15 minutes pour un déploiement local ou sur un cloud composé d'une unique machine. 20 à 30 minutes pour un déploiement dans un cluster (le temps de déploiement varie fortement avec le nombre de machines composant le cluster) .

Bibliographie/Sitographie

Formation Docker: https://docs.Docker.com

Documentation Docker: https://docs.Docker.com/engine/

Kubernetes:
https://kubernetes.io/docs/home/?path=users&persona=app-developer&level=foundational
https://www.projectatomic.io/blog/2017/09/running-kubernetes-on-fedora-atomic-26/

Mongodb: https://docs.mongodb.com/manual/

Tomcat: http://tomcat.apache.org/

TABLE DES ANNEXES

Annexe 1 - 2: Dockerfiles…………………………………………………………… 37 - 38
Annexe 3: Scripts d'installation local…………………………………………………. 39
Annexe 4: Scripts de désinstallation.………………………………………………….. 40 - 42
Annexe 5: Scripts de déploiement PHIS sur cluster Swarm…………………………… 43
Annexe 6: Docker-compose de PHIS sur cluster………………………………………. 44
Annexe 7: Programme formation………………………………………………………. 45

Dockerfiles

```
#phis-ws.dimg
FROM ubuntu:16.04

#Ajout de tomcat déjà configuré
COPY ./tomcat /usr/local/tomcat

#Définition du dossier de travail comme étant le dossier principal de Tomcat
WORKDIR /usr/local/tomcat

#Ajout des bibliothèques Java pour tomcat et phis
RUN apt-get update
RUN apt-get install -y openjdk-8-jdk openjdk-8-jre

#Permettre l'accés aux fichiers de configurations de Tomcat
VOLUME /usr/local/tomcat/conf

#Permettre l'accés aux fichiers logs
VOLUME /usr/local/tomcat/logs
VOLUME /home/tomcat/phis2ws/logs
VOLUME /root/.RDF4J/Server/logs
VOLUME /root/.RDF4J/Workbench/logs

#Teste la santé du container
HEALTHCHECK --interval=5s --timeout=3s --retries=3 \
CMD curl -f http://127.0.0.1:8080 || exit 1

#Exposition du port
EXPOSE 8080

#Lancement du service Tomcat lors de la création d'un container
ENTRYPOINT ["/usr/local/tomcat/bin/catalina.sh","run"]
```

Dockerfile Pour le webservice de PHIS

```
#phis-webclt.dimg
FROM ubuntu:16.04

#Définition du répertoire de travail
WORKDIR /var/www/html/

#Installation de apache2 ainsi que php7
RUN apt-get update
RUN apt-get install -y apache2 php7.0 libapache2-mod-php7.0

#Ajout du client web phis au répertoire racine de apache2
COPY ./phis-webapp /var/www/html/phis-webapp

#Configuration pour l'accés au client phis
RUN cp /var/www/html/phis-webapp/index.php /var/www/html/index.php


#Permettre l'accés aux logs
VOLUME /var/log/apache2

#Exposition du port 80
EXPOSE 80

#teste de la santé du container
HEALTHCHECK --interval=5s --timeout=3s --retries=3 \
CMD curl -f http://127.0.0.1:80 || exit 1

#Lancement du serveur apache lors du lancement d'un container
ENTRYPOINT ["apachectl","-d","/etc/apache2/","-f","apache2.conf","-e","info","-DFOREGROUND"]
```

Dockerfile pour le serveur Apache-PHP de l’application web de PHIS

```
#phis-mongodb.dimg
FROM ubuntu:16.04

COPY ./entrypoint.sh /bin/entrypoint.sh
#Ajout de mongodb
RUN apt-get update

RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
RUN echo "deb http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.2 multiverse" > /etc/apt/sources.list.d/mongodb-org-3.2.list

RUN apt-get update
RUN apt-get install -y mongodb-org

#Permettre l'accés aux logs
VOLUME /var/log/mongodb
VOLUME /data/db

ENV BINDIP 172.18.0.3
#Exposition du port
EXPOSE 27017

#Teste la santé du container
HEALTHCHECK --interval=5s --timeout=3s --retries=3 \
CMD curl -f http://127.0.0.1:27017/test || exit 1

#Lancement de mongodb au démarrage des containers
ENTRYPOINT ["entrypoint.sh"]
```

Dockerfile du serveur Mongo pour PHIS

```
#phis-postgres.dimg
FROM postgres:10

#Ajout de postgis
VOLUME /var/log/postgresql
EXPOSE 5432

RUN apt-get update
RUN apt-get -y install postgresql-10-postgis-2.4
```

Dockerfile du serveur Postgres pour PHIS

```
#phis-fs.dimg

FROM ubuntu:16.04

COPY ./init_files_fs /init_files

#Utilisateur phis
RUN newusers /init_files/users

#ajout du serveur sftp
RUN apt-get update
RUN apt-get install -y openssh-sftp-server

#Création des dossiers pour phis
RUN mkdir -p /home/phis/documents/instance
RUN mkdir -p /home/phis/images
RUN mkdir -p /home/phis/www/layers

#Gestion des droits sur ces dossiers
RUN chown -R phis:phis /home/phis
RUN chmod -R 775 /home/phis

#Configuration du serveur sftp

RUN rm /etc/ssh/sshd_config
RUN cp /init_files/sshd_config /etc/ssh/
RUN /etc/init.d/ssh restart

#Ajout de apache2 pour l'accés aux layers
RUN apt-get install -y apache2

#configuration de apache2
RUN rm /etc/apache2/apache2.conf
RUN rm /etc/apache2/sites-available/*
RUN cp /init_files/apache2.conf /etc/apache2/apache2.conf
RUN cp /init_files/000-default.conf /etc/apache2/sites-available/000-default.conf
RUN cp /init_files/default-ssl.conf /etc/apache2/sites-available/default-ssl.conf
RUN chmod 775 -R /home/phis/www/

#Ouverture des ports
EXPOSE 80
EXPOSE 22

RUN cp /init_files/entrypoint.sh /bin/

#Lancement du service apache2 ET du serveur sftp
```

Dockerfile pour le serveur FTP pour PHIS

Outils déploiement local

```
containerphis
  cloud
  local
    diaphen
    dimg
    phis-webapp
    phis2-ws
    webapps
    installPhis.sh
    README
    uninstallPhis.sh
```

```
14  alias docker="sudo docker"
15
16  PHIS_DINSTALL=/home/$USER/.phis
17  SQL=$PHIS_DINSTALL/databases/postgres
18  NOSQL=$PHIS_DINSTALL/databases/mongodb
19  RDF4J=$PHIS_DINSTALL/databases/rdf4j
20  FS=$PHIS_DINSTALL/fileserver
21
22  LOGS=$PHIS_DINSTALL/logs
23
24
25  #Making repositories to save Phis data
26  mkdir -p $PHIS_DINSTALL
27  sudo mkdir -p $SQL
28  sudo mkdir -p $NOSQL
29  sudo mkdir -p $RDF4J
30  sudo cp -r ./diaphen $RDF4J/
31
32  #Making File Server repositories
33  sudo mkdir -p $FS/documents/instance
34  sudo mkdir -p $FS/images
35  sudo mkdir -p $FS/www/layers
36
37  #Making Logs repositories
38  sudo mkdir -p $LOGS/postgresql
39  sudo mkdir -p $LOGS/mongodb
40  sudo mkdir -p $LOGS/tomcat
41  sudo mkdir -p $LOGS/rdf4j/Server
42  sudo mkdir -p $LOGS/rdf4j/Workbench
```

```
containerphis
  cloud
  local
    diaphen
    dimg
    phis-webapp
    phis2-ws
    webapps
    installPhis.sh
    README
    uninstallPhis.sh
```

```
46  #Putting Phis executables in their directory
47  sudo mkdir -p $PHIS_DINSTALL/html
48  sudo cp -r ./webapps $PHIS_DINSTALL/
49  sudo cp -r ./phis-webapp $PHIS_DINSTALL/html/
50  sudo cp ./phis-webapp/index.php $PHIS_DINSTALL/html/
51
52  #Making a Volume where you can access to the Tomcat configuration
53  #Make a symbolic link from Phis directory to this Volume
54  docker volume create conf
55  ln -s /var/lib/docker/volumes/conf/_data $PHIS_DINSTALL/tomcat_conf
56
57  #All containers user need read and write rigths to Phis directories
58  sudo chmod -R 777 $PHIS_DINSTALL
59
60  #With "-b" arguments this script build docker images
61  #from dockerfiles in directory dimg
62
63  if [ "$1" = "-b" ]
64  then
65      if [ "$2" = "--no-cache" ]
66      then
67        NOCACHE = $2
68      else
69        NOCACHE = ""
70      fi
71      docker build -t $NOCACHE postgres-postgis.dimg ./dimg/postgres/
72      docker build -t $NOCACHE mongodb.dimg ./dimg/mongodb/
73      docker build -t $NOCACHE phis_fs.dimg ./dimg/fileserver/
74      docker build -t $NOCACHE phis_ws.dimg ./dimg/ws/
```

containerphis
- cloud
- local
  - diaphen
  - dimg
  - phis-webapp
  - phis2-ws
  - webapps
  - installPhis.sh
  - README
  - uninstallPhis.sh

```
75      docker build -t $NOCACHE phis_webclt.dimg ./dimg/webclt/
76      #When a build process doesn't end correctly
77      #docker doesn't remove intermediate containers
78      #With arguments "-b" AND "-rm" this script remove ALL containers
79      #already on the host
80      if ["$2" = "-rm" || "$3" = "-rm" ]
81      then
82        docker stop $(docker ps -a -q)
83        docker rm $(docker ps -a -q)
84      fi
85  fi
86
87
88  #Make a docker network to connect all Phis containers
89  docker network create   --driver=bridge \
90        --subnet=172.18.0.0/16 \
91        --gateway=172.18.0.1 \
92        phis-network
93
94  #Postgres server container
95  #Save data on the SQL folder
96  #Redirect logs files to the correct Phis host directory
97  #This container belongs to the Phis network
98  #Container ip on Phis network
99  #Connect host port to the container open port
100 #Container Docker image (in local, so it is necessary building before, arg "-b")
101 docker run  --name phis-postgres \
102     --volume $SQL:/var/lib/postgresql/data \
103     -v $LOGS/postgresql:/var/log/postgresql \
104     --network phis-network \
```

containerphis
- cloud
- local
  - diaphen
  - dimg
  - phis-webapp
  - phis2-ws
  - webapps
  - installPhis.sh
  - README
  - uninstallPhis.sh

```
105     --ip="172.18.0.2" \
106     -p 5434:5432 \
107     -d postgres-postgis.dimg
108
109
110 #Mongo server container
111 #Save data on the NOSQL folder
112 #Redirect logs files to the correct Phis host directory
113 #This container belongs to the Phis network
114 #Container ip on Phis network
115 #Connect host port to the container open port
116 #Container Docker image (in local, so it is necessary building before, arg "-b")
117 docker run  --name phis-mongodb \
118     --volume $NOSQL:/data/db \
119     -v $LOGS/mongodb:/var/log/mongodb \
120     --network phis-network \
121     --ip="172.18.0.3" \
122     -p 27027:27017 \
123     -d mongodb.dimg
124
125 #File server container
126 #Mount swap folder
127 #This container belongs to the Phis network
128 #Container ip on Phis network
129 #Connect host ports to the container open ports
130 #Container Docker image (in local, so it is necessary building before, arg "-b")
131 docker run  --name phis-fs \
132     --network phis-network \
133     --volume $FS:/home/phis \
134     --ip="172.18.0.6" \
```

```
cloud
local
  diaphen
  dimg
  phis-webapp
  phis2-ws
  webapps
  installPhis.sh
  README
  uninstallPhis.sh
```

```
135      -p 24:22 \
136      -p 84:80 \
137      -d phis_fs.dimg
138
139  #Web services container
140  #Save data on the RDF4J folder
141  #Mount swap folder
142  #Redirect logs files to the correct Phis host directory
143  #Connect conf volume to the tomcat configuration folder
144  #This container belongs to the Phis network
145  #Container ip on Phis network
146  #Connect host port to the container open port
147  #Container Docker image (in local, so it is necessary building before, arg "-b")
148  docker run  --name phis-webservices \
149      --volume $RDF4J:/root/.RDF4J/server/repositories \
150      --volume $FS:/home/phis \
151      --volume $PHIS_DINSTALL/webapps:/usr/local/tomcat/webapps \
152      -v $LOGS/rdf4j/Workbench:/root/.RDF4J/Workbench/logs \
153      -v $LOGS/rdf4j/Server:/root/.RDF4J/Server/logs \
154      -v $LOGS/tomcat:/usr/local/tomcat/logs \
155      -v $LOGS/phis:/home/tomcat/phis2ws/logs \
156      -v conf:/usr/local/tomcat/conf \
157      --network phis-network \
158      --ip="172.18.0.4" \
159      -p 8088:8080 \
160      -d phis_ws.dimg
```

```
  phis2-ws
  webapps
  installPhis.sh
  README
  uninstallPhis.sh
```

```
161
162  #Web client container
163  #Mount client executable folder
164  #Redirect logs files to the correct Phis host directory
165  #This container belongs to the Phis network
166  #Container ip on Phis network
167  #Connect host port to the container open port
168  #Container Docker image (in local, so it is necessary building before, arg "-b")
169  docker run  --name phis-webclt \
170      --volume $PHIS_DINSTALL/html:/var/www/html \
171      -v $LOGS/apache2:/var/log/apache2 \
172      --network phis-network \
173      --ip="172.18.0.5" \
174      -p 88:80 \
175      -d phis_webclt.dimg
176
177  #Quickly build SQL database
178  #Temporary container:
179  #When container process ending docker remove the container
180  docker run --rm --network phis-network -it sammy2104/buildphisdb.dimg
181
182  echo "Phis: 172.18.0.5:80"
183
```

local/installPhis.sh 183:1

Script de déploiement de PHIS en local

Outils déploiement cloud

```
#!/bin/sh

if [ $# -lt 2 ]
then
  echo "$0 needs 2 arguments: $0 <install_directory> <Manager_IP_address>"
  exit 1
fi

./imgupdate.sh

PHIS_DINSTALL=$1
SQL=$PHIS_DINSTALL/databases/postgres
NOSQL=$PHIS_DINSTALL/databases/mongodb
RDF4J=$PHIS_DINSTALL/databases/rdf4j
FS=$PHIS_DINSTALL/fileserver

LOGS=$PHIS_DINSTALL/logs

export PHIS_DINSTALL SQL NOSQL RDF4J FS LOGS
#Making repositories to save Phis data
mkdir -p $PHIS_DINSTALL
mkdir -p $SQL
mkdir -p $NOSQL
mkdir -p $RDF4J
cp -r ./diaphen $RDF4J/

#Making File Server repositories
mkdir -p $FS/documents/instance
mkdir -p $FS/images
mkdir -p $FS/www/layers
```

```
#Making Logs repositories
mkdir -p $LOGS/postgresql
mkdir -p $LOGS/mongodb
mkdir -p $LOGS/tomcat
mkdir -p $LOGS/rdf4j/Server
mkdir -p $LOGS/rdf4j/Workbench
mkdir -p $LOGS/phis
mkdir -p $LOGS/apache2

#Putting Phis executable in their directory
mkdir -p $PHIS_DINSTALL/html
cp -r ./webapps $PHIS_DINSTALL/
cp -r ./phis-webapp $PHIS_DINSTALL/html/
cp ./phis-webapp/index.php $PHIS_DINSTALL/html/

#All containers user need read and write rigths to Phis directories
chmod -R 777 $PHIS_DINSTALL

docker swarm init --advertise-addr $2
echo "Please copy-paste the docker command previously obtained (docker join ...\
) on every worker node and press a key when it done"

read ok

docker stack deploy --compose-file phis-compose.yml phis

rm $PHIS_DINSTALL/webapps/phis2ws/WEB-INF/lib/postgis-stubs-1.3.3.jar
docker service update --restart-delay 5s phis-webservices
docker run --rm --network phis-network -it postgresclt.dimg
```

Script déploiement de PHIS sur un cluster Swarm

Project | phis-compose.yml

```
containerphis
  cloud
    multiVM
      manager
        diaphen
        dimg
        phis-webapp
        webapps
        deploy.sh
        imgupdate.sh
        initNFS.sh
        installDocker.sh
        manager_ubuntu.sh
        phis-compose.yml
        uninstallPhis.sh
      phis2-ws
      worker
      manager.tar.gz
      README
      worker.tar.gz
    uneVM
    uneVM.tar.gz
  local
```

```yaml
version: "3"
services:
    phis-postgres:
        image: sammy2104/postgres-postgis.dimg
        volumes:
            - "$SQL:/var/lib/postgresql/data"
            - "$LOGS/postgresql:/var/log/postgresql"
        ports:
            - "5432:5432"
        deploy:
            replicas: 1
            restart_policy:
              condition: any
    phis-mongodb:
        image: sammy2104/mongodb.dimg
        environment:
            - BINDIP="172.16.3.24"
        volumes:
            - "$NOSQL:/data/db"
            - "$LOGS/mongodb:/var/log/mongodb"
        ports:
            - "27017:27017"
        deploy:
            replicas: 1
            restart_policy:
              condition: any
    phis-fs:
        image: sammy2104/phis_fs.dimg
        volumes:
            - "$FS:/home/phis"
        ports:
            - "22:22"
            - "88:80"
        deploy:
            replicas: 1
            restart_policy:
              condition: any
    phis-webservices:
        image: sammy2104/phis_ws.dimg
        volumes:
            - "$RDF4J:/root/.RDF4J/server/repositories"
            - "$FS:/home/phis"
            - "$PHIS_DINSTALL/webapps:/usr/local/tomcat/webapps"
            - "$LOGS/rdf4j/Workbench:/root/.RDF4J/Workbench/logs"
            - "$LOGS/rdf4j/Server:/root/.RDF4J/Server/logs"
            - "$LOGS/tomcat:/usr/local/tomcat/logs"
            - "$LOGS/phis:/home/tomcat/phis2ws/logs"
        ports:
            - "8080:8080"
        deploy:
            replicas: 1
            restart_policy:
              condition: any
    phis-webapp:
        image: sammy2104/phis_webclt.dimg
        volumes:
            - "$PHIS_DINSTALL/html:/var/www/html"
            - "$LOGS/apache2:/var/log/apache2"
        ports:
            - "80:80"
        deploy:
            replicas: 1
            restart_policy:
              condition: any

```

cloud/multiVM/manager/phis-compose.yml 1:1

Docker-compose pour les services PHIS sur cluster Swarm

Programme Formation

- L'installation de Docker
- L'architecture obtenue par la contenerisation
- La construction des images Docker
- Le lancement d'un container
- Le déploiement de PHIS automatisé par un script en local
- Initialisation d'un cluster Swarm
- Le déploiement de PHIS automatisé par des scripts sur un cloud
- Comment faire évoluer PHIS simplement grâce à ce que j'ai réalisé

Formation disponible en suivant le lien:
https://docs.google.com/document/d/1TroJEPFQ8ZI5O0ODblWp69N18PUQSKr_AUeV8dBTxeo/edit?usp=sharing