Dockerisation du projet Observo

MEMOIRE DE FIN D'ÉTUDES

Thomas FRITSCH

Promotion 2021 – 2023, UHA 4.0.5

UNIT SOLUTIONS

Master informatique, Parcours Informatique et Mobilité

Août 2023

Tuteur professionnel

Mr. Thibault VELLICUS

Tuteur pédagogique

Mr. Mounir ELBAZ

UNIVERSITÉ HAUTE-ALSACE

Formulaire d'information sur le plagiat

Dans le règlement des examens validé par la CFVU du 2 octobre 2014, le plagiat est assimilé à une fraude. Cette information est présentée à chaque étudiant de l'Université de Haute Alsace susceptible de rédiger un document long de type thèse, mémoire ou rapport de stage. Le formulaire signé devra obligatoirement être intégré au document.

Le plagiat consiste à reproduire un texte, une partie d'un texte, des données ou des images, toute production (texte ou image), ou à paraphraser un texte sans indiquer la provenance ou l'auteur.

Le plagiat enfreint les règles de la déontologie universitaire et il constitue une fraude. Le plagiat constitue également une atteinte au droit d'auteur et à la propriété intellectuelle, susceptible d'être assimilé à un délit de contrefaçon.

En cas de plagiat dans un devoir, dossier, mémoire ou thèse, l'étudiant sera présenté à la section disciplinaire de l'université qui pourra prononcer des sanctions allant de l'avertissement à l'exclusion.
Dans le cas où le plagiat est aussi caractérisé comme étant une contrefaçon, d'éventuelles poursuites judiciaires pourront s'ajouter à la procédure disciplinaire.

Je soussigné(e) FRITSCH Thomas

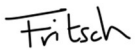
Etudiant(e) à l'Université de Haute Alsace en : UHA 4.0

Niveau d'études : UHA 4.0.5

Formation ou parcours : Master informatique mobile

Reconnaît avoir pris connaissance du formulaire d'information sur le plagiat.

Fait à Habsheim Le 28/07/2023 Signature :



Remerciements :

Je souhaite d'abord remercier Mr. **Thierry MOEBEL**, directeur d'UNIT SOLUTIONS pour avoir passé ces trois années d'alternance au sein de son entreprise. L'entreprise m'a proposé un excellent cadre de travail, ainsi qu'un projet très intéressant et enrichissant.

J'aimerais tout particulièrement remercier mon tuteur, Mr. **Thibault VELLICUS,** pour m'avoir guidé et donné l'ensemble des connaissances nécessaires pour travailler sur le projet.

Je remercie également un étudiant de l'UHA 4.0, Mr. **Tom WILLEMIN** avec qui j'ai pu travailler ces deux dernières années dans la même équipe.

Ensuite, je remercie l'ensemble des collaborateurs de l'entreprise pour leur aide et leur bienveillance.

Je remercie également l'ensemble de l'équipe pédagogique de **l'UHA 4.0**, Mr. Mounir ELBAZ, Mr. Daniel DA FONSECA, Mr. Florent BOURGEOIS, Mr. Pierre SCHULLER, Mme. Audrey BRUNSPERGER ainsi que les nombreux intervenants qui m'ont aidé et m'ont fait progresser durant ces années de formation.

Enfin, merci aux correcteurs pour m'avoir relu et donné leur avis sur ce manuscrit.

Sommaire :

Remerciements

Introduction 1

I. CONTEXTUALISATION 2

1. Mon parcours 2
2. Mes projets en formation 3

II. PRÉSENTATION DU PROJET 4

1. L'entreprise UNIT SOLUTIONS 4
2. Présentation du projet Observo 5
3. Les méthodes et outils de travail 8

III. L'ÉTAT DE L'ART 10

1. La virtualisation légère avec Docker 10
2. Intégration continue et déploiement continu avec Azure DevOps 14

IV. RÉALISATION 16

1. Introduction 16
2. Cahier des charges 18
3. L'architecture Docker 21
4. L'implémentation de cette architecture 24
5. Microsoft Azure DevOps 31
6. Problèmes restants 35
7. Etat actuel du projet 36

Conclusion 37

Bibliographie 39

Glossaire 40

Annexes

Résumé et mots clés

INTRODUCTION

Après l'obtention de ma licence professionnelle « Développeur informatique », j'ai voulu approfondir mes compétences et intégrer le cursus master de l'UHA 4.0.

Pendant ces deux années de formation au master, j'ai poursuivi mon alternance dans l'entreprise **UNIT SOLUTIONS** à Allschwil en Suisse dans laquelle j'ai déjà pu travailler à partir de ma troisième année de licence. J'ai pu continuer à contribuer sur le projet **Observo**. Il s'agit d'une application mobile pour permettre de collecter des données en faisant des inspections d'ouvrages d'art.

Dans ce mémoire je vous expliquerai en détail le projet et ma contribution au sein du projet Observo. Elle consiste à Dockeriser le projet Observo afin de créer un nouvel outil de gestion de travail collaboratif. C'est donc une étape transitionnelle importante pour la bonne avancée du projet.

J'ai réalisé la majorité de mon alternance en télétravail, car notre équipe est très productive et nous nous sommes bien adaptés grâce aux visio-conférences et aux outils collaboratifs utilisés dans notre équipe.

Dans ce mémoire, je vais commencer par vous détailler les compétences que j'ai acquises au cours de ma formation. Ensuite, je vous présenterai le projet auquel j'ai contribué. Puis je vous présenterai ma contribution pendant cette année d'alternance. Enfin, je conclurai ce mémoire en expliquant ce que cette alternance m'a apporté ainsi qu'à l'entreprise.

I. CONTEXTUALISATION

1. Mon Parcours

Après avoir obtenu mon Bac Scientifique, je me suis tourné vers une formation innovante : UHA 4.0. Cette formation m'a permis d'évoluer en toute autonomie tout en apprenant à travailler dans un milieu professionnel.

Il y a deux ans, j'ai conclu ma formation de licence professionnelle en obtenant celle-ci.

Depuis, j'ai pu continuer mon alternance dans le cadre du parcours master. En effet j'ai tellement évolué avec la formation de trois ans en licence et le parcours master m'a beaucoup intéressé afin d'obtenir un supplément de compétences.

Au cours de ces quelques années, j'ai pu réaliser plusieurs mois en entreprise. Cela m'a permis d'acquérir de nouvelles compétences. Pendant ces deux dernières années en alternance, j'ai énormément progressé dans les technologies *C#* [1] et *Xamarin* puisque l'entreprise travaille principalement avec ces technologies. Mais j'ai aussi pu professionnaliser mes habitudes de code, par exemple dans le langage *Angular* où le contexte professionnel nous impose de fournir un travail propre et parfaitement architecturé, pour une pérennisation sereine du projet.

[1] : Tous les mots en italique sont expliqués dans le glossaire à la fin du manuscrit

2. Mes projets en formation

Au cours de ma formation de master, j'ai eu l'occasion de travailler sur deux longs sujets en groupe. Nous appelons cela un « fil rouge » car il commence au mois de septembre et termine en janvier sans arrêt, et en parallèle aux autres cours des intervenants. J'ai donc pu travailler sur deux fils rouges, les voici :

- UHA 4.0.4 – 2021 à 2022 : Suivi de la concentration de CO2 d'une pièce

  Dans le cadre de la pandémie de la covid 19, nous avons eu pour but de faire une solution de suivi de la concentration en CO2 dans une pièce. Le CO2 est un très bon indicateur sur l'aération de la pièce. Nous devons donc en quelque sorte faire un assistant qui avertit les usagers de la salle s'il faut l'aérer pour limiter la propagation de la covid 19.

  Dans ce projet j'ai pris le rôle de scrum master et j'ai principalement travaillé sur le capteur connecté *Arduino*, l'enregistrement des données et le traitement de celles-ci pour en déduire l'heure à laquelle il faudra aérer grâce à la régression linéaire.

- UHA 4.0.5 – 2022 à 2023 : Plantes connectées

  Cette année, nous avons eu un sujet concernant l'utilisation de l'eau pour l'arrosage des plantes. L'idée était de suivre l'état d'une plante et de l'arroser au mieux possible. L'utilisation de capteurs d'humidité et de température nous a permis de savoir si la plante devait ou non être arrosée. De plus, une caméra nous a également permis de prendre des photos des feuilles qui ont la particularité de réagir au manque d'eau. Grâce à quelques algorithmes, nous avons essayé de connaitre l'état de la plante avec uniquement la vision de ses feuilles. En tant que scrum master de l'équipe, j'ai principalement travaillé le sujet des flux de données et des capteurs. J'ai créé un nouveau capteur d'humidité pouvant déterminer le taux d'humidité en profondeur dans la plante ce qui nous a permis d'avoir des mesures bien plus fiables.

Pendant les 6 mois de formation, j'ai également pu découvrir et m'initier à plusieurs compétences notamment : La synthèse d'image avec OpenGL, l'optimisation combinatoire, l'algorithmique géométrique, la business intelligence, la sécurité et les réseaux, le calcul massivement parallèle et la numérisation 2D/3D.

II. PRÉSENTATION DU PROJET

1. L'entreprise UNIT SOLUTIONS

UNIT SOLUTIONS est une petite entreprise localisée en Suisse à Allschwil dans le canton de Bâle Campagne.
Elle a été créée en 1986, et son chef d'entreprise est Mr. Thierry MOEBEL.
Elle comporte actuellement une vingtaine de collaborateurs. Son activité principale se concentre sur la création ainsi que sur la maintenance de logiciels informatiques. Ses activités se trouvent principalement en Suisse mais elle s'étend peu à peu à la France, à l'Allemagne et à d'autres pays européens.

Les domaines d'activité d'UNIT SOLUTIONS sont globalement basés sur la géolocalisation (Systèmes d'information géographiques ou SIG), mais également sur le développement d'applications web ou mobiles. L'informatique représente donc une partie très importante de l'activité de l'entreprise.

Les quatre projets principaux d'UNIT SOLUTIONS sont : InfKuba, Kuba, LangsamVerkehr et aussi Observo. Ces projets sont utilisés par des centaines d'utilisateurs principalement en Suisse, et également en France.

LangsamVerkehr est un projet permettant la gestion des sentiers de mobilité douce de la Suisse (chemins de randonnées et pistes cyclables).

InfKuba et Kuba sont deux projets ayant le même but : la gestion des ouvrages d'art (ponts, routes et tunnels). Kuba est l'ancien logiciel client lourd et infKuba est sa déclinaison en version web.

Observo est un projet qui peut être utilisé dans la suite logicielle infKuba, mais également sur des domaines d'application qui lui sont propres comme la gestion de petits ouvrages, et l'entretien des cours d'eau.

Au cours de cette alternance, je faisais partie du projet Observo. L'équipe est composée de développeurs, d'architectes, de testeurs, et de personnes chargées du support du projet. Dans l'entreprise nous privilégions le partage de connaissances entre les employés. J'ai donc eu l'occasion de d'échanger avec des collègues pour avancer plus facilement dans mon travail.

2. Présentation du projet Observo

En **2012**, le projet **Observo** a été créé pour satisfaire la demande de deux cantons suisses, celui d'Uri et celui de Zoug. Le chef du projet est Mr. Thibault VELLICUS.
Observo est un projet de développement d'application mobile initialement créé pour la plateforme Windows Phone. Observo permet à l'utilisateur de remplir des formulaires qu'il géolocalise sur une carte afin de réaliser de la collecte de données.

En **2015**, le projet a gagné en popularité, de nouveaux clients étant intéressés par le produit. Cependant, il n'avait pas été pensé pour être modulaire et dynamique. Ceci posa beaucoup de problèmes, car les informations des formulaires étaient stockées dans le code source de l'application. A chaque modification d'un formulaire, il fallait remettre en ligne l'application sur le store de Windows Phone. Cela entrainait une grosse perte de temps.

Pour pallier à ce problème, Observo a été complètement recréé en *C#* avec le Framework *Xamarin* (Une solution permettant de faire des applications mobiles de dernière génération, multiplateforme iOS et Android). Non seulement les formulaires peuvent être chargés dynamiquement depuis une API, mais en plus l'application est compatible avec Windows Phone, Android et iOS.
C'est le Framework *Xamarin* qui a été choisi car il permet de développer un seul code source pour les trois plateformes mobiles, mais également parce que le *C#* est le langage le plus présent dans l'entreprise.

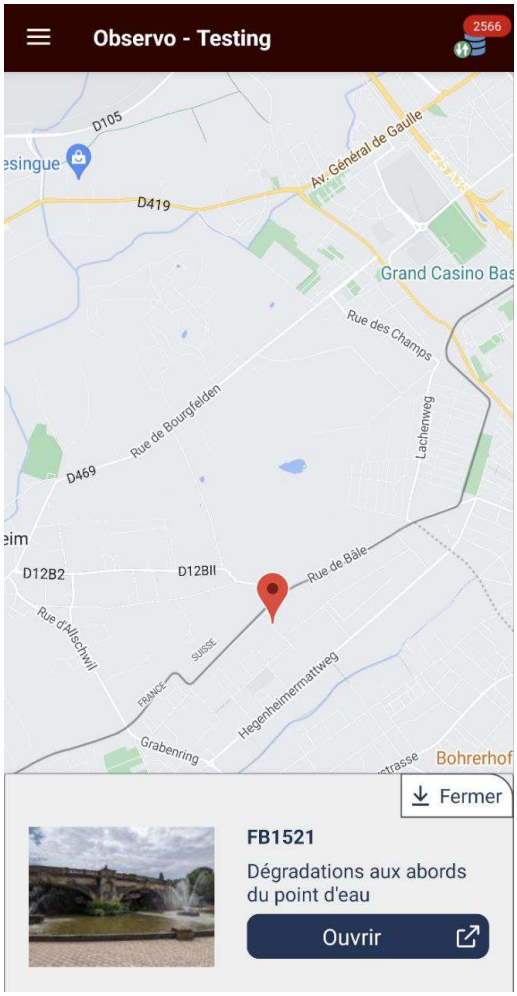
En **2020**, la plateforme Windows Phone a totalement été abandonnée suite à son abandon par Microsoft.
Depuis, le projet continue régulièrement d'être amélioré en y implémentant des nouvelles fonctionnalités, notamment en y ajoutant des modules pour générer les formulaires.
Il y a également plusieurs outils qui ont été créés autour d'Observo permettant par exemple d'importer et d'exporter des données, configurer les formulaires ou bien de recevoir des notifications.

Les nombreux atouts d'Observo en font une application très flexible correspondant à plusieurs types de besoins (infrastructures routières, fluviales, mobilier urbain, dispositifs de sécurité, protection de l'environnement, etc...).

Par exemple, Observo peut servir à établir un diagnostic régulier d'un pont en y remplissant des formulaires. L'entreprise chargée de l'inspection, remplit un formulaire sur Observo. Un rapport ou un export de données peut être rapidement généré et envoyé à une entreprise chargée de faire les éventuelles réparations.

Voici un rapide aperçu d'un formulaire sur l'application Observo :

On a ici un formulaire très simple disposant de plusieurs onglets avec quelques champs. On peut distinguer un champ permettant de mettre des images ainsi que plusieurs champs texte pour argumenter l'observation.



L'inspection est géolocalisée sur une carte. Cela permet une bien meilleure visibilité des ouvrages et de leur état global. Il peut être matérialisé par un point, une ligne ou un polygone.

Structure du projet :

Les numéros présents dans le schéma ci-dessous correspondent aux exposants dans le texte explicatif en bas de page.

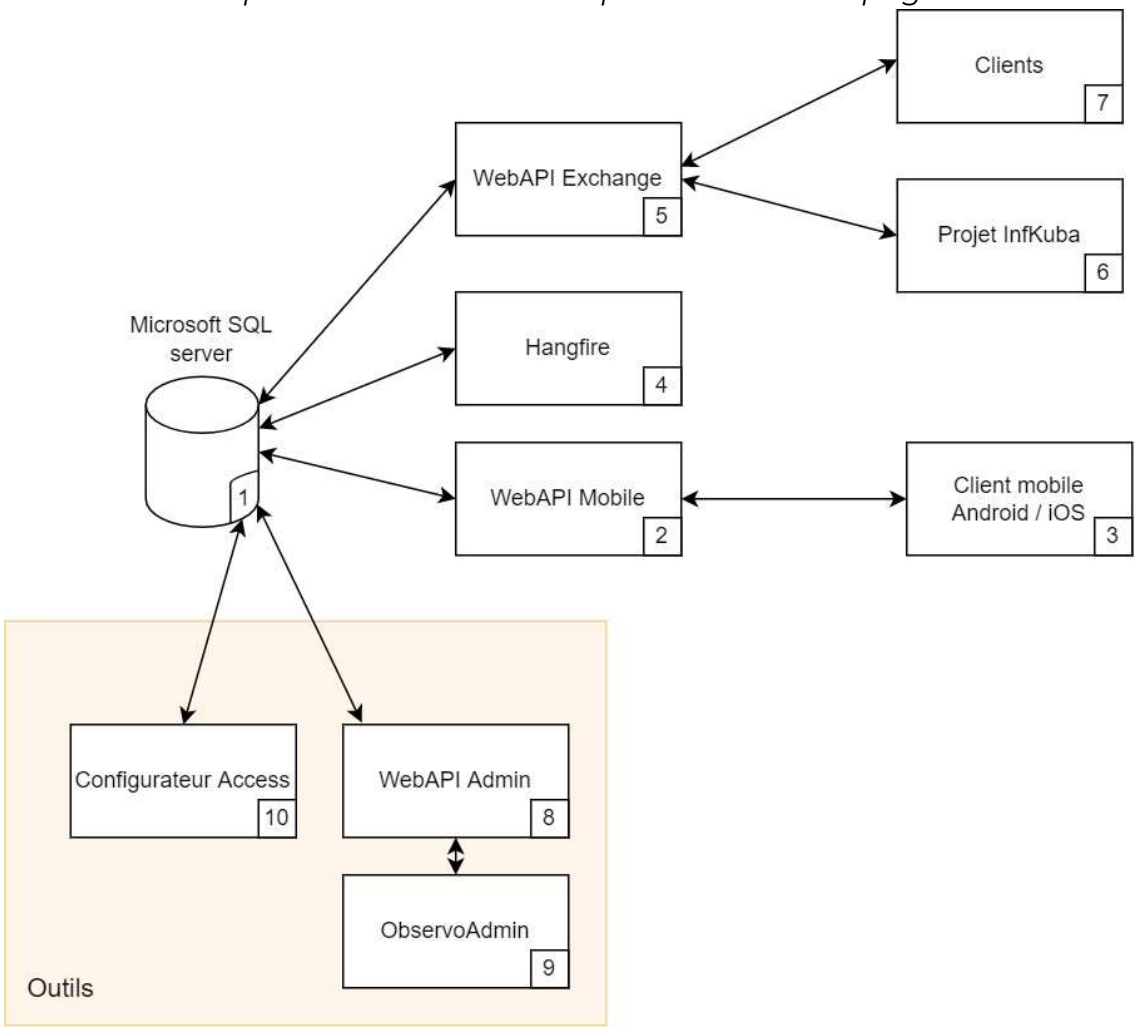


Figure 1 : Architecture simplifiée du projet Observo

Dans le projet, nous avons une base de données Microsoft SQL Server [1] qui représente le point central du projet.
Le projet *WebAPI* mobile [2] (serveur) s'occupe de l'envoi et de la réception des données avec l'application mobile [3].
Nous avons aussi le projet Hangfire [4] qui sert à générer les rapports, envoyer les mails et exécuter des taches récurrentes.
Un second projet *WebApi* Exchange [5] permet l'échange de données vers des projets externes comme le projet InfKuba [6]. Des clients [7] l'utilisent également pour récupérer les données de leurs inspections.
Un troisième projet *WebApi* Admin [8] sert cette fois-ci à faire fonctionner le l'outil d'administration web : ObservoAdmin [9].
Enfin, le configurateur Access [10] est l'ancien outil d'administration du projet qui est progressivement remplacé par ObservoAdmin.

Statistiques du projet :

- Nombre de clients : 40
- Nombre d'utilisateurs actifs (depuis 2022) : 492
- Pays : Suisse et France
- Nombres de formulaires : 146
- Nombre d'inspections sur le terrain (depuis 2022) : 6752
- Nombres d'ouvrages suivis : 132135
- Répartition des plateformes mobiles : iOS 60% / Android : 40%

3. Les méthodes et outils de travail

Tout au long de mon alternance, nous avons travaillé avec la méthodologie **Agile Scrum**. Notre équipe de développeurs était certes petite et facilement gérable, nous préférions appliquer cette méthode en prenant soin de revenir sur le travail fourni le *sprint* précédent pour améliorer le suivant. Nous étions trois développeurs à temps plein sur le projet, mettant en commun notre travail régulièrement et organisant des Daily (réunions quotidiennes). Tom et moi avons très régulièrement organisé des revues de code pour nous permettre d'améliorer la qualité de travail et d'échanger sur les bonnes pratiques de code. Chaque semaine, nous faisions donc une réunion inter projets mettant ainsi en commun nos réalisations de la semaine. Et depuis cette année, l'entreprise a décidé de faire une réunion mensuelle pour mettre en commun les problématiques d'architecture de chaque projet.

De plus, à chaque fin de *sprint,* nous faisons une « *sprint* rétrospective » pour nous permettre d'analyser sur les bonnes et mauvaises pratiques que nous avons mis en œuvre durant ces dernières semaines. Grâce à cela j'ai remarqué que notre travail est en général plus organisé et plus agréable pour l'ensemble des membres de l'équipe.

Depuis cette année, nous avons également un testeur (Stephan BORNOWSKI) à temps partiel sur notre projet. Il teste l'ensemble de nos tickets afin de correctement les réaliser et limiter les bugs dans l'application. Il nous aide à gagner beaucoup de temps, puisqu'auparavant nous devions réaliser les tests nous-même, et cela n'était pas aussi structuré que maintenant.

Nous avons utilisé plusieurs outils informatiques pour améliorer notre productivité et notre organisation en équipe. En voici la liste des principaux utilisés :

a. Confluence

Pour réaliser la documentation du projet et également se ressourcer sur les bonnes pratiques de l'entreprise, nous avons utilisé Confluence issu de la suite

de logiciels Atlassian. Confluence est un espace de travail qui permet de documenter les projets. Cela nous permet de retrouver l'ensemble des informations essentielles à la compréhension du projet et à son développement. J'y ai, à plusieurs reprises, documenté mes recherches et mon travail réalisé.

b. Jira

L'outil Jira (de la suite Atlassian) nous permet de suivre les tâches à faire et de découper les *sprints*. L'outil Jira nous sert à organiser les différentes tâches en cours. Nous avons découpé nos tâches en plusieurs sections nous permettant de facilement voir l'état actuel de la tâche en cours de chaque personne.
Ayant déjà beaucoup utilisé l'outil Jira, je le maitrise sans soucis en milieu professionnel.

c. Azure DevOps

Comme contrôleur de sources, nous utilisons *Microsoft Azure* DevOps. Nous l'utilisons pour le *CI/CD*, qui nous fait gagner énormément de temps pour la mise en production des nouvelles versions du projet. Nous aimerions très prochainement y intégrer l'exécution des tests unitaires dans le processus de *CI/CD* pour gagner en qualité de code. Dans ce manuscrit nous verrons que nous pouvons utiliser Azure DevOps avec *Docker*.

d. Visual studio

Nous utilisons Microsoft Visual Studio 2022 comme éditeur de code pour le *C#*. Il permet aussi d'utiliser l'émulateur Android pour le développement mobile. Visual Studio intègre également la connexion à un Mac pour l'émulation de la plateforme iOS. Nous travaillons sur Windows notamment pour avoir une meilleure compatibilité avec les outils Microsoft, mais aussi pour disposer de machines plus performantes pour l'émulation mobile.

5. Webstorm

Depuis l'arrivée du projet « ObservoAdmin », un outil web destiné à l'administration du projet (voir annexe 2), nous développons également avec l'IDE Webstorm de la suite Jetbrains. Webstrom est également utilisé par l'équipe du projet InfKuba dans l'entreprise, donc nous nous sommes automatiquement tournés vers ce logiciel. Il nous est très utile pour améliorer notre qualité de code et notre productivité. Il est bien plus adapté au développement du code *Angular* que l'IDE Visual Studio.

III. L'ÉTAT DE L'ART

1. La virtualisation légère avec *Docker*

a. Présentation de la plateforme

Docker est un outil de « virtualisation » de services *open source*. Il permet de déployer des applications dans des conteneurs. Le principe utilisé est la conteneurisation pour permettre d'être beaucoup plus léger que de la virtualisation comme les machines virtuelles.
La plateforme a été lancée en 2013 et son créateur est Solomon Hykes.
Depuis maintenant quelques années, *Docker* a pris beaucoup d'ampleur et est de plus en plus utilisé dans les entreprises.

b. Les principes de base

Docker a plusieurs principes fondamentaux, **les conteneurs**, des « boîtes » permettant de cloisonner les services. Ce qui est ingénieux, c'est que chaque conteneur n'a pas besoin d'avoir son propre système d'exploitation puisqu'il va se reposer sur celui de la machine hôte. On gagne alors énormément en efficience. De plus, chaque conteneur est autosuffisant, c'est-à-dire qu'il possède tout dont il a besoin pour fonctionner.
Ensuite nous pouvons évoquer les **images** *Docker*. Elles apparaissent comme une recette permettant la création de conteneurs.
Grâce à elles, nous pouvons détailler les étapes de création du conteneur et y ajouter de nombreux paramètres pour personnaliser son application. Ces instructions sont stockées dans des fichiers textes appelés des ***Dockerfile.*** La rapidité de création des images, et donc des conteneurs, constitue un point essentiel de *Docker*.
Une fois les images créées, elles peuvent être stockées dans des **registres** (registry en anglais). Ces « bibliothèques » d'images sont très utiles pour permettre le partage de celles-ci. Le plus gros registre communautaire est *Docker Hub*. En tant que développeur, nous pouvons envoyer nos images sur celui-ci et permettre à d'autres développeurs de les télécharger pour exécuter nos programmes.
Tout cela défini ce qu'est *Docker* et ses principes clés. Il a été conceptualisé pour être accessible au grand public. Cela fait de lui un outil très apprécié et facilement abordable, même pour une personne débutante.

c. Docker compose

Docker, à lui seul, est déjà un formidable outil. Il est souvent complété par un plugin qui peut s'avérer très utile lorsqu'on veut Dockeriser un projet complet. Le plugin le plus connu de *Docker est sûrement* ***Docker* compose.** Cet outil nous permet de programmer le démarrage de plusieurs services à la fois, et avec de nombreuses options. On peut ainsi gérer les réseaux, les *volumes* de stockage et les images de nos applications.

d. Utilisation et avantages de Docker

En tout premier lieu, *Docker* est utilisé pour permettre la gestion très simplement des services hébergés sur une machine. Il est également devenu un standard pour le partage de projet. Il n'est pas rare que nous retrouvions un fichier *Dockerfile* ou Docker-compose dans les sources d'un projet open-source sur *GitHub*.
Docker permet aussi de réduire de façon importante les coûts d'hébergement dans le cloud (grâce à l'utilisation de bien moins de ressources matérielles et à la mutualisation des serveurs).
Pour aller plus loin, *Docker* est aussi au cœur de deux gros outils *Docker* Swam et Kubertenes. Ils permettent l'orchestration des conteneurs, en automatisant et en simplifiant certaines actions. En résumé, un orchestrateur de conteneurs permet la création, la modification et la suppression des conteneurs en fonction de certains critères.

Par exemple : lorsque les bacheliers vont consulter leurs résultats du bac sur internet, un flux très important de personnes veut accéder à la même ressource dans un court laps de temps. Si aucun système de gestion des conteneurs n'est utilisé, il faudra soit prévoir suffisamment de services à l'avance (ce qui n'est pas toujours faisable car ce n'est pas toujours prévoyable), ou bien le service ne sera pas accessible par les internautes pour cause de saturation des serveurs. On utilise donc des services automatisés qui analysent la charge des serveurs pour démarrer plus ou moins de services à la volée. Cela permet donc une gestion avancée et automatisée des pics de charges tout en étant le moins coûteux possible, et permettre aux usagers d'accéder à la plateforme.

De plus, *Docker* est aussi un bon élève en termes de sécurité. Il nous permet de cloisonner les applications en créant plusieurs réseaux virtuels. La gestion des mots de passe est aussi réalisée par des fichiers d'environnement séparés et sécurisés. Par exemple, le service « Key Vault » de *Microsoft Azure* permet le

stockage sécurisé de fichiers de configuration pour leur utilisation dans les services Azure cloud Docker de Microsoft.
Concernant les performances, *Docker* est surtout efficient. Il reste très léger comparé à de la virtualisation lourde comme le sont les machines virtuelles. D'après mes tests et mon expérience, les performances sont très faiblement dégradées.

e. Comparaison de Docker avec la virtualisation matérielle

Dans le livre « Docker et conteneurs » de Pierre-Yves Cloux, Thomas Garlot et Johann Kohler, il est expliqué ce que *Docker* apporte en plus par rapport à de la virtualisation matérielle.
Les informations suivantes sont principalement fondées sur les propos de ce livre.
Dans le cas où nous souhaitons héberger trois services nous pouvons voir sur le schéma ci-dessous qu'avec la virtualisation matérielle nous avons un impact matériel conséquent. Chaque service est encapsulé dans une machine virtuelle nécessitant un *OS ce qui a pour conséquence u*ne forte consommation de ressources, notamment de *RAM*.

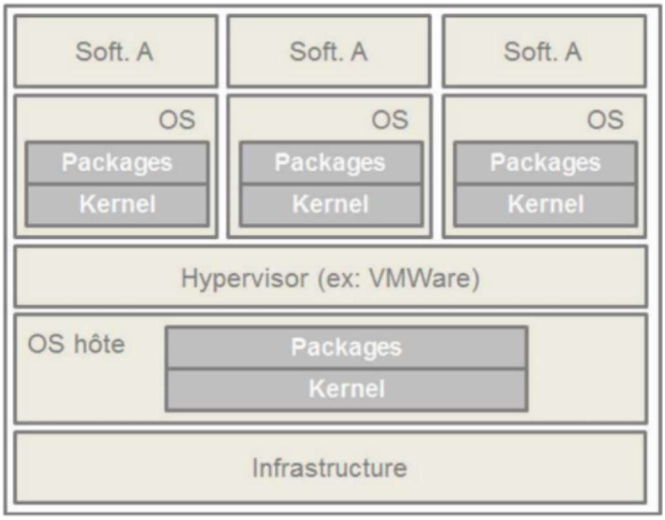


Figure 2 : Virtualisation matérielle : trois VM sur un même hôte [1]

Docker a été créé pour corriger ces importants problèmes de ressources. Cela est nécessaire car en entreprise on héberge parfois plusieurs centaines de services. Dans le schéma ci-dessous, nous pouvons observer que chaque service a un impact bien plus faible. Un seul *OS* est alors nécessaire et les conteneurs partagent le *kernel* hôte.

[1] : Cloux, P., Garlot, T., Kohler, J. (2022). Docker et conteneurs – p.6

Le nombre de conteneurs qu'un serveur hôte peut héberger est alors bien plus élevé qu'avec des machines virtuelles.

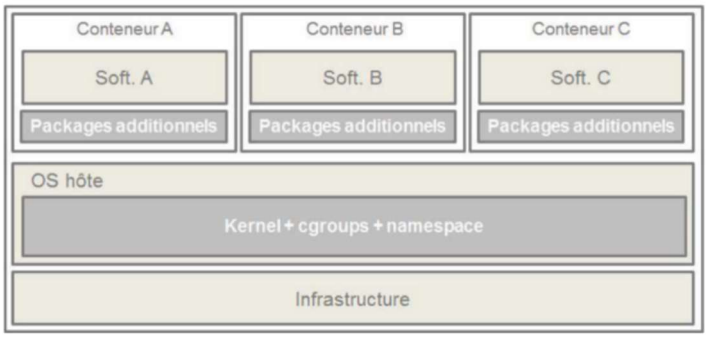


Figure 3 : Architecture à conteneur [1]

f. Conclusion

En conclusion, *Docker* est une plateforme de virtualisation de services avec de nombreux avantages. Elle est encore peu utilisée par les petites entreprises car elle est une technologie assez récente. Le manque de connaissances et le cap à franchir pour l'utiliser sont probablement trop importants. Cependant, ces nombreux avantages et ces rares inconvénients en font un outil très à la mode, qui apporte une excellente solution dans de nombreux cas.

[1] : Cloux, P., Garlot, T., Kohler, J. (2022). Docker et conteneurs – p.6

2. Intégration continue et déploiement continu avec Azure DevOps

a. Introduction

Microsoft Azure DevOps est un outil d'intégration continu et de déploiement continu (*CI/CD*) mais pas seulement ; Il permet la gestion de projets, du code source, des tests et des fichiers de build. Ses nombreux atouts en font un outil très polyvalent et complet. De plus, DevOps intègre beaucoup de fonctionnalités pour utiliser *Docker*.

b. Les composants d'Azure DevOps

Les principaux composants de l'outil Azure DevOps sont :

Azure Boards : Un ensemble d'outils agiles pour la gestion de projet. Il est possible de créer des tickets, de les planifier et de gérer une équipe de développement.

Azure Pipelines : Toute une partie de l'outil est consacrée au *CI/CD*. C'est dans cette section que nous pouvons générer des fichiers de build de notre projet, et de les déployer sur de nombreux supports (Serveur local, cloud, IaaS (Infrastructure-as-a-Service)).

Azure Repos : Un contrôleur de sources permettant de stocker, et gérer l'ensemble du code de son projet.

Azure Tests : Tester votre projet dans sa globalité avec des d'outils manuels ou automatiques.

Azure Artéfacts : Héberger les paquets générés à la suite des pipelines *CI/CD*. On peut aussi les historiser et les partager entre équipes.

c. Azure DevOps avec Docker

Comme cité précédemment, Azure DevOps s'adapte très bien à *Docker*. Il fournit quelques commandes prédéfinies pour par exemple stocker les images *Docker* générées sur une Registry dans le cloud Azure. Ceci permet alors à d'autre personnes comme les développeurs de les récupérer. Nous utilisons la version « *self-hosted* » de DevOps, ainsi nous avons la possibilité de

nous connecter en réseau local et sécurisé aux machines où l'on va déployer nos conteneurs. Dans la version cloud, c'est encore plus simple et accessible, seulement quelques clics suffisent à déployer notre application sur le cloud Azure et d'y accéder depuis internet. Depuis les versions 2020 et 2022, Microsoft met *Docker* plus en avant car il a décidé d'ajouter encore de nouvelles fonctionnalités et surtout l'intégration de Kubernetes et *Docker* Swarm pour les plus grosses infrastructures.

d. Conclusion

En conclusion DevOps est un excellent outil multifonctions et spécialisé dans le *CI/CD*. Son intégration avec *Docker* offre la possibilité de faire le pas plus facilement vers une telle architecture virtualisée, et de façon automatique. Cela permet par exemple à des petites entreprises d'avoir des outils puissants rapidement et sans compétences particulières.

II. RÉALISATION

1. Introduction

Depuis quelques temps, nous rencontrons des difficultés d'organisation dans l'équipe de configuration d'Observo, composée actuellement de trois personnes.

Les formulaires représentent la base du projet Observo. Chaque client fait une demande pour avoir ses propres formulaires afin de l'adapter à son activité. Il est fréquent qu'un client demande quelques modifications de ses formulaires. Il est nécessaire que nous ayons une équipe qui réponde à ce besoin. En effet, cette équipe travaille sur les nombreuses configurations de formulaires, et a des difficultés à communiquer pour organiser les versions de chaque formulaire.

Deux outils permettent de faire l'édition des formulaires : un ancien outil sous *Microsoft Access,* et un nouvel outil web ObservoAdmin. Ces deux outils ne communiquent que par la base de données, et il arrive alors que deux personnes travaillent en simultané sur un même formulaire sans s'en rendre compte. Cela est contraignant, car l'une des deux personnes perd son travail. Une solution doit alors permettre le travail collaboratif concernant l'édition de formulaires.

Nous avons mesuré l'importance de ce problème et avons essayé de le corriger. Nous avons élaboré une solution : création d'un petit projet orchestrateur de configurations. Nous l'avons appelé ObsRepo [ɔpsʁipo] (Observo – Repository). Celui-ci va se positionner en tant que chef d'orchestre d'environnements Observo.

Son fonctionnement nous permettra de régler ce problème car :

- Il a la connaissance de tout ce qui se passe sur chaque environnement.
- Il bloque l'édition simultanée d'un formulaire ce qui évite d'avoir des conflits.
- Il historise chaque configuration.

Grâce à lui, tout ce qui se passe à un endroit va alors prévenir les autres configurateurs.

Voici un schéma simplifié du positionnement de ce projet parmi les nombreux environnements Observo à l'heure actuelle :

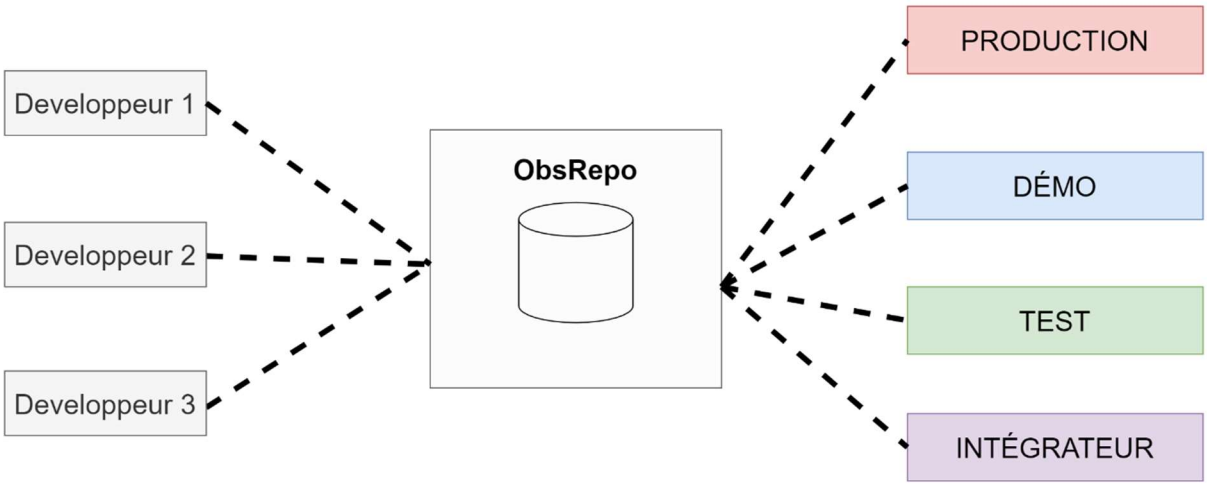


Figure 4 : Schéma simplifié du positionnement d'ObsRepo parmi les environnements Observo

Comprendre les environnements utilisés dans Observo :

Observo est un peu plus complexe que de la simple édition d'un formulaire sur un environnement de production. Nous disposons exactement de quatre environnements dits « de production », et ils ont chacun un rôle important :

- La production :
  C'est sur cet environnement que les clients se connectent et renseignent les données réelles pour leur travail.
- L'environnement de démonstration :
  L'environnement de démonstration est mis à la disposition du client pour effectuer des tests et des formations à ses ouvriers.
- L'environnement de test :
  Sur cet environnement nous procédons à plusieurs tests avant son déploiement en production pour le client. C'est à ce moment que le testeur de l'équipe vérifie les développements réalisés.
- L'environnement intégrateur :
  C'est sur cet environnement que nous créons et modifions les environnements. Les configurations de formulaires ne sont alors pas toujours terminées. Après l'arrivée de ObsRepo c'est cet environnement qui sera l'unique base pour éditer une configuration d'un formulaire.

Les environnements de développement à la gauche du schéma sont là pour indiquer qu'un développeur pourra faire des tests. Il pourra aussi

potentiellement se rattacher à ObsRepo pour débugger une configuration de formulaire sur son environnement. Cela facilitera aussi le transfert de configurations en interne.

Grâce à cette architecture, chaque action passe par ce projet central qui écrit des « logs » dans un fichier, et transmet les messages aux autres environnements.
On a également choisi d'historiser les configurations au sein de ce projet pour permettre de restaurer une ancienne version, en cas d'erreur d'un configurateur.

Un problème majeur se présente, pour pouvoir développer ce nouveau projet central. En effet, nous devons disposer de **plusieurs copies d'Observo**. Pourquoi ?
ObsRepo a pour but de **gérer plusieurs environnements Observo** car il fait de l'import-export de données.
Pour permettre le développement et les tests du projet ObsRepo, il nous faut une solution simple pour créer des *instance*s à notre volonté sur un même ordinateur (surtout lors du développement). Lorsque nous développerons l'outil ObsRepo, il faudra que le développeur puisse à la fois avoir **une** *instance* du projet d'ObsRepo, et à minima **trois** environnements complet du projet Observo (comprenant : *Web Apis,* Outil admin web, base de données, …). **Sinon le développement de l'outil ObsRepo ne sera pas possible**.

La problématique suivante se pose :
« Comment permettre la multi-instanciation du projet Observo sur une unique machine ? »

Après réflexion, nous avons naturellement choisi d'utiliser *Docker* pour répondre à cette problématique. Grâce à lui, nous pourrons disposer de plusieurs environnements sur un même serveur. Dans ce manuscrit, nous verrons les raisons de ce choix, l'analyse que j'ai faite pour intégrer *Docker* dans Observo, et l'explication de mon travail réalisé.

2. Le cahier des charges

Avant de commencer, nous avons élaboré un premier cahier des charges.
Celui-ci reste temporaire, car nous n'avons pas le recul suffisant pour anticiper les éventuelles modifications une fois la première version d'ObsRepo réalisée.

S'agissant d'un sujet de transition à la création du gros projet ObsRepo, nous ne savons pas comment le développement d'ObsRepo influencera sur ce cahier des charges. C'est pourquoi nous serons sûrement amenés à réaliser une version 2 après le commencement du développement d'ObsRepo.
Voici la liste non exhaustive des principaux points-clés attendus :

- Garder la même rapidité d'exécution des déploiements serveur que la solution actuelle.
- Avoir à minima les mêmes performances des services (API, Sites web, Base de données).
- Avoir une interface et configuration simple, accessible à tous les collaborateurs.
- Avoir une architecture modulable et évolutive.
- Modifier le moins possible le projet Observo pour éviter les effets de bord et avoir une rétrocompatibilité complète avec l'ensemble des outils satellites d'Observo. Cela comprend :
    a. L'application mobile Observo
    b. L'interface d'administration web ObservoAdmin
    c. L'outil de génération de rapports
    d. Les projets de l'entreprise : infKuba et LV2
    e. Les échanges de données avec les clients via l'API Exchange
- Le temps de développement nécessaire doit être au maximum d'un mois, car le budget est limité pour ce type de sujet de transition. Nous aurons à nouveau un peu de temps disponible après le commencement d'ObsRepo.

Pourquoi utiliser Docker ?

En premier lieu, il faut préciser qu'une solution de virtualisation est nécessaire car lancer plusieurs instances hors *Docker* n'est que très difficilement faisable voir infaisable. Observo est configuré avec beaucoup de dépendances : base de données par instance, données de fonctionnements (configurations, images, logos, …) et par conséquent il faudrait tout gérer à la main et faire énormément de configurations. De plus il sera aussi très probable que nous devions démarrer des projets dans des versions différentes pour tester la rétrocompatibilité et dans ce cas l'utilisation d'environnements virtuels sera obligatoire.
Nous avons donc choisi l'écosystème Docker pour répondre à cette problématique. Il a fallu confirmer l'utilisation de *Docker,* et donc aussi comparer les solutions disponibles pour répondre à notre problématique.

J'ai retenu quatre solutions possibles :

- Docker
  Docker est bien évidement la solution la plus polyvalente dans ce genre de problème. J'ai déjà pu auparavant beaucoup tester l'écosystème. Il convient parfaitement grâce à sa légèreté, sa performance et son adaptabilité.
- LXC (Linux containers)
  LXC est une solution de plus bas-niveau que *Docker*. Nous utilisons des ordinateurs sous Windows, et cette solution nous a paru plus compliquée et chronophage. De plus, aucune personne de l'entreprise n'a déjà eu l'occasion de travailler avec LXC.
- La virtualisation matérielle (via VirtualBox ou VMWare)
  Cette solution est la plus probable face à *Docker*. La virtualisation matérielle correspond à quasiment tous les critères que nous avons : temps de développement, performances, facilité d'utilisation, même machine, …. Mais un gros souci se pose : l'utilisation en ressources matérielles. En effet, comme expliqué précédemment dans l'état de l'art de *Docker*, ce type de virtualisation est peu efficient puisqu'il virtualise plusieurs services et *OS* inutiles. Nos machines de développement ne sont pas assez puissantes pour faire fonctionner plusieurs environnements Observo en simultané sur machines virtuelles.
- L'utilisation de plusieurs machines physiques
  Cette solution d'avoir un PC par environnement n'a bien évident pas été retenue du fait que le budget ne permet pas à chaque développeur d'avoir 2 ou 3 machines. Même en se partageant les machines cela ne serait pas une solution fiable. L'hébergement sur serveur ne serait lui aussi pas amélioré ayant plusieurs environnements sur un même serveur.

C'est pourquoi nous avons retenu *Docker*, une solution universelle et polyvalente, qui possède de nombreux atouts : rapidité, efficience, simplicité et sécurité.

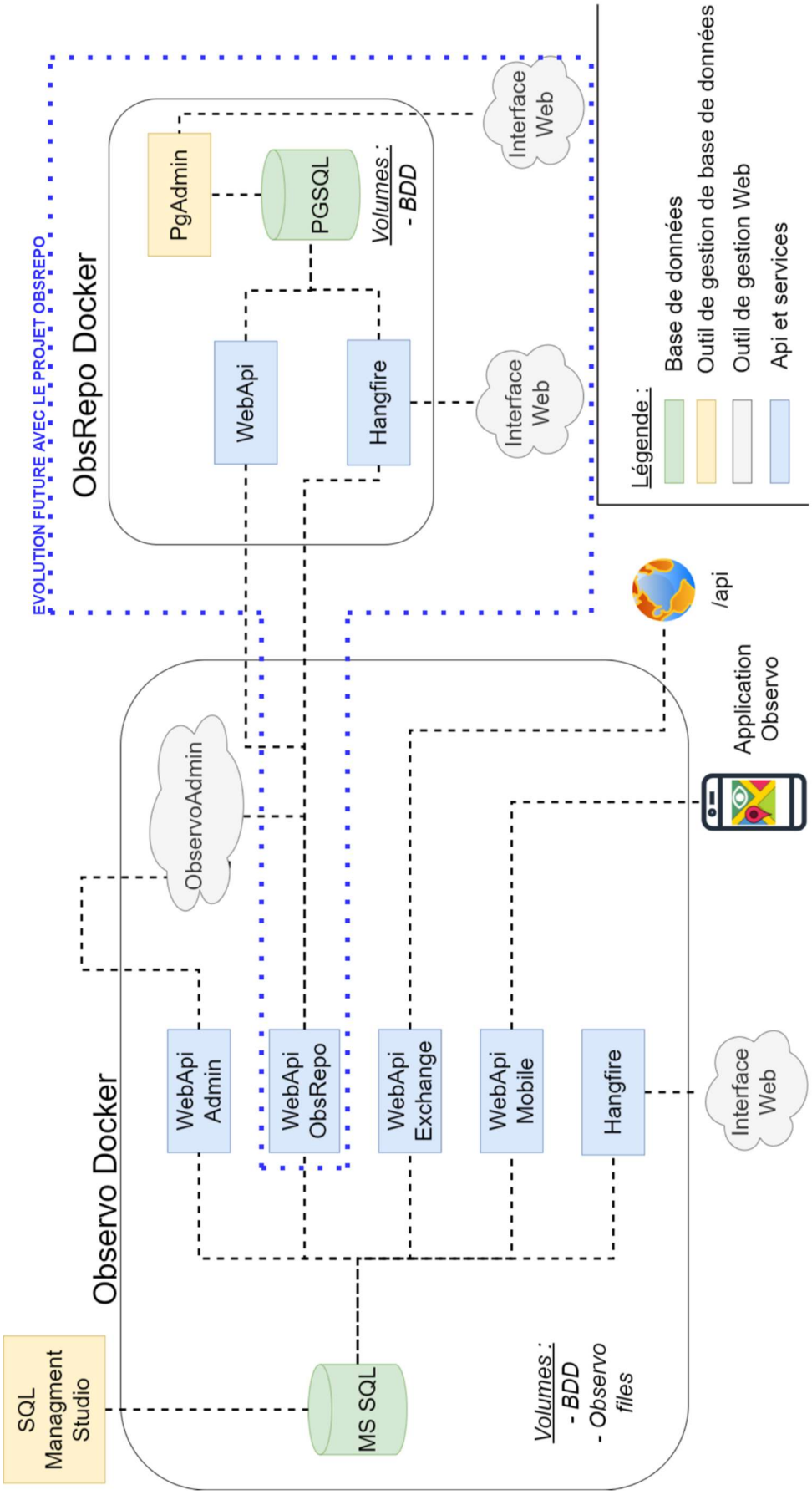
Suite à l'élaboration du concept et de la validation de l'utilisation de *Docker* avec mon équipe pour répondre à cette problématique, j'ai ainsi commencé à travailler sur un concept d'architecture pouvant convenir. C'est un travail très important pour correctement préparer mon travail. Dans la partie suivante nous allons donc parler de cette architecture *Docker*.

3. L'architecture Docker

Ensuite j'ai réalisé un travail de recherche très important pour déterminer l'architecture optimale pour la conteneurisation du projet.
L'architecture d'un projet sous *Docker* peut être faite de plusieurs manières. Grâce à mon expérience et à mes recherches, j'ai réalisé une architecture pour le projet.

L'ensemble de l'architecture a été exposée à un architecte logiciel de l'entreprise, Benjamin SCHUDEL. Il a beaucoup de notions de *Docker* et d'azure cloud. Nous avons réalisé plusieurs réunions, environ une toutes les deux semaines, pour échanger et valider mon travail sur *Docker*.
L'entreprise met en avant les collaborations inter-projet pour permettre de mieux faire avancer chacun d'eux. Cela m'a été très utile et réconfortant qu'une personne expérimentée reconnaisse et approuve mon travail.
C'est donc avec son approbation que nous avons validé cette architecture ci-dessous.

Voici le schéma de l'architecture :



Dans le paragraphe suivant, je vous explique pas à pas les choix réalisés dans la conception de ce schéma d'architecture.

Dans un premier lieu, on peut constater que les projets Observo et ObsRepo sont séparés. Dans le langage *Docker,* on peut appeler cela des « stack ». Il s'agit de groupes de services représentant le plus souvent des projets. La partie ObsRepo (en pointillé bleu foncé) n'est pas traitée dans ce manuscrit, car il s'agit d'un développement futur non nécessaire à la résolution de la problématique.

Les lignes pointillées représentent les nombreuses connexions réseau entre les services.
Concernant les services, au niveau d'Observo, nous avons la base de données Microsoft SQL Server qui est présente dans *Docker*. Il est indiqué de ne pas *Docker*iser de base de données et encore moins de fichiers de données. En effet, cela pourrait poser des problèmes de performances et de perte de données. Cependant, nous utilisons dans un premier cas cette architecture pour créer des environnements de développement et de test. La problématique de la puissance et de la pérennité des données n'est donc pas le cas. Dans une éventuelle évolution vers l'utilisation de *Docker* en production, nous pourrons bien évidemment séparer la base de données.
Une connexion sécurisée vers la base de données est permise pour l'administrer avec l'outil SQL Server Management Studio (SSMS).
Les 5 *Web Api* viennent ensuite se connecter à la base de données. Le fait qu'elles soient découpées nous aide à cloisonner et à sécuriser les différents accès aux projets. Chaque service aura donc sa propre image pour être ainsi arrêtée et redémarrée indépendamment des autres services.
Les fichiers du projet (images, rapports, données externes, …) doivent pouvoir être accessibles et partagés entre les services. C'est pourquoi j'ai prévu deux *volumes* de stockages partagés en mode bridge : les fichiers sont stockés sur la machine hôte, et les services y ont accès en lecture et écriture via ce volume *Docker*.
Le réseau est un point très important dans une architecture *Docker*, non seulement pour la sécurité mais aussi pour permettre un accès à chaque service dans tous les cas de figure. Sans trop détailler, voici une liste des différents numéros de ports devant être ouverts vers l'extérieur :

- 1433 : Base de données
- 8002 : *Web Api* Admin
- 8000 : *Web Api* Mobile
- 8001 : *Web Api* Exchange
- 8005 : *Web Api* ObsRepo
- 8004 : Hangfire
- 8003 : ObservoAdmin

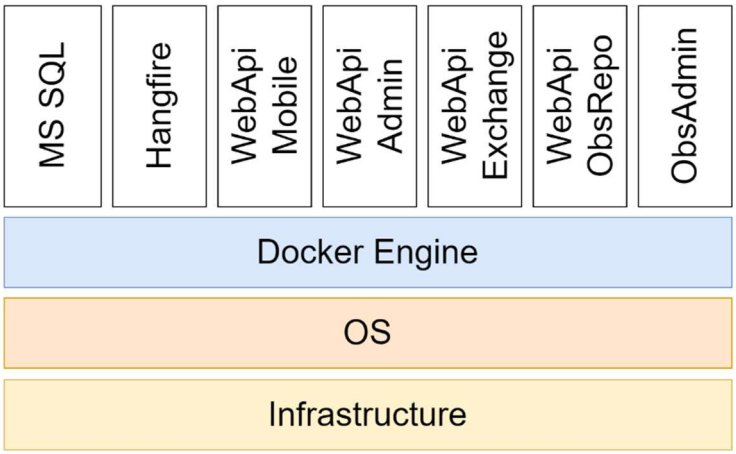


Figure 5 : Schéma des couches pour la virtualisation avec Docker

Ce schéma représente les différentes couches sur lesquelles les conteneurs *Docker* fonctionnent. Tout en dessous on retrouve l'infrastructure, c'est le matériel physique (serveur, pc, cloud). Puis on a *l'OS* (Windows, MacOs ou *Linux*) pour faire fonctionner *Docker* sur la machine. Ensuite on a le moteur *Docker* (*Docker* Engine) qui se charge de l'exécution des conteneurs *Docker*. Et enfin, au dernier niveau on a les conteneurs avec chacun leur service.

4. L'implémentation de cette architecture

Dans cette partie, je vais vous expliquer comment j'ai procédé pour mettre en place cette architecture *Docker*.

a. L'environnement de développement

En premier lieu, il m'a fallu préparer mon environnement de développement pour réaliser mon travail. Je réalise une petite documentation des outils nécessaires. Cela permettra aussi aux autres développeurs de préparer leur environnement pour travailler sur le projet.

Nos machines fonctionnent sous Windows 10 et 11. Étant très fortement dépendants des technologies de Microsoft (Office, Visual Studio, SQL Server, ...), nous ne pouvons pas changer de système d'exploitation pour passer sur *Linux*. Il m'a donc fallu trouver une solution simple pour exécuter un moteur *Docker* sur Windows. Heureusement, depuis quelques années, Microsoft travaille sur l'intégration du noyau *Linux* dans Windows : cela s'appelle WSL. Dans sa version 2, nous avons un réel système d'exploitation en CLI (Command Line Interface) de *Linux,* sous la distribution de notre choix. J'ai choisi Ubuntu car c'est la plus répandue et elle est conseillée par Microsoft. Les

performances sont réputées bonnes depuis la version 2 ce qui me permet de m'orienter vers cette solution.

Ensuite, pour permettre l'exécution de *Docker* sur nos machines il existe un logiciel développé par Docker, Inc. Il s'appelle « Docker Desktop », et nous permet d'utiliser les commandes *Docker* et Docker-compose sur nos machines Windows disposant de WSL 2.

J'ai cependant rencontré un problème avec cette façon de travailler avec *Docker* sur Windows. La consommation de *RAM* de la machine est très importante, tellement importante que les 32Go dont nous disposons ne suffisent plus pour travailler confortablement. Des ralentissements et des disfonctionnements apparaissent et nous font perdre du temps. Ce problème vient de WSL2, qui recréé un noyau *Linux* à côté de Windows. WSL2 ainsi que *Docker* Desktop consomment donc facilement 20Go de *RAM*.

J'ai réussi à trouver une solution très intéressante. Nous pouvons ainsi paramétrer la consommation maximale de *RAM* de WSL2 que j'ai donc définie sur 12Go, et je n'ai quasiment plus aucun souci à ce niveau. Si cela posait à nouveau problème, nous avons tout de même prévu d'augmenter la capacité de mémoire vive à 64Go pour un plus grand confort de travail.

Enfin, concernant l'environnement de développement, j'ai préconisé à l'équipe d'utiliser un très bon outil. Il s'agit de « Portainer » que j'utilise à titre personnel depuis quelques temps. Il nous permet de savoir et de tout réaliser concernant *Docker* : démarrer, arrêter, statistiques, logs, … des conteneurs, images, *volumes* et réseaux. Le tout, sous forme d'une interface graphique web elle-même, est hébergé dans *Docker*. (voir un aperçu de l'interface Portainer en Annexe 1)

Figure 6 : Récapitulatif des outils de développement utilisés

b. Les fichiers et scripts de configuration

Un projet, devant fonctionner avec *Docker*, nécessite d'avoir plusieurs fichiers de configurations. On pourrait les comparer à des plans, permettant à *Docker* de construire les différentes briques de l'architecture.

En premier lieu, on peut s'intéresser aux ***Dockerfile*** des fichiers texte qui sont spécifiques à chaque service *Docker* (une API, un site Web, une base de données, etc.).

Le *Dockerfile* aura pour but d'énoncer les étapes pour la création d'une image *Docker*.

Voici un exemple pour une *Web API* sous *C# .NET Core* :

Généralement elle se base sur une image déjà existante et préparée pour le service nécessaire. Elle est ensuite paramétrée. Voici une image fournie par Microsoft avec la version *.NET*7 d'ASP *.NET* .C'est sur cette image que fonctionnera le service. On expose le port 80 : c'est sur ce port que le service fonctionnera.

```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
ENV DockerBuild=true
...
```

Figure 7 : Début du fichier Dockerfile d'exemple

La deuxième partie est une image dédiée au build qui utilise le SDK de Microsoft. Et je considère la variable d'environnement « *Docker*Build » comme « vraie » pour un problème que je vais vous expliquer prochainement.

C'est ensuite que nous copions les références des projets pour que l'image possède bien les sources nécessaires au démarrage du service. Le projet Observo est sur ce point assez complexe, car il a une grande quantité de références à d'autres sous-projets. J'ai finalement réussi à trouver comment configurer chaque dépendance pour positionner correctement chacun des fichiers. Les fichiers de configurations (environnements, mots de passes, url, etc..) sont également copiés dans l'image. Cela n'est pas satisfaisant pour

l'instant, mais je cherche une meilleure solution. Dans l'image ci-dessous, chaque opération « COPY » est un lien de référence avec un autre sous-projet.

```
...
COPY ["ObservoForm.Server/WebApi/ObservoForm.Server.WebApi.Admin/ObservoForm.Server.WebA
pi.Admin.csproj", "ObservoForm.Server.WebApi.Admin/"]
RUN dotnet restore "ObservoForm.Server.WebApi.Admin/ObservoForm.Server.WebApi.Admin.cspr
oj"
COPY ./Config/Global/appsettings.json ./ObservoForm.Server/WebApi/ObservoForm.Server.Web
Api.Admin/appsettings.json
COPY ./Config/Development/appsettings.env.json ./ObservoForm.Server/WebApi/ObservoForm.S
erver.WebApi.Admin/appsettings.env.json
COPY ./Config/Development/appsettings.env.tfh.json ./ObservoForm.Server/WebApi/ObservoFo
rm.Server.WebApi.Admin/appsettings.env.Development.json
COPY ./Config/Development/log4net.tfh.config ./ObservoForm.Server/WebApi/ObservoForm.Ser
ver.WebApi.Admin/log4net.config
COPY ./ObservoForm.Server/ObservoForm.Server.CommonCore/ ./ObservoForm.Server/ObservoFor
m.Server.CommonCore/
COPY ./ObservoForm.Server/WebApi/ ./ObservoForm.Server/WebApi/
COPY ./ObservoForm.Common/ ./ObservoForm.Common/
COPY ./Unit/ ./Unit/
COPY ./Config/ ./Config/
...
```

Figure 8 : Suite du fichier Dockerfile : Références

Pour finir, il y a encore des commandes de build et de configuration réseau. La commande « COPY » ci-dessous va mettre les fichiers créés dans l'image « finale » qui est celle que nous avons préparée au début.

```
...
WORKDIR "/src/ObservoForm.Server/WebApi/ObservoForm.Server.WebApi.Admin"
RUN dotnet build "ObservoForm.Server.WebApi.Admin.csproj" -c Debug -o /app/build

FROM build AS publish
RUN dotnet publish "ObservoForm.Server.WebApi.Admin.csproj" -c Debug -o /app/publish /p:
UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "ObservoForm.Server.WebApi.Admin.dll", "--environment=Developmen
t"]
...
```

Figure 9 : Fin du fichier Dockerfile d'exemple

A la fin, on indique un paramètre d'environnement « développement », car il s'agit d'un *Dockerfile* de développement (dev.*Dockerfile*).

En testant si mon *Dockerfile* fonctionne, j'ai rencontré un problème : celui du changement d'*OS* pour un même fichier .csproj (le fichier qui définit le projet *C#* avec les dépendances, packages, ressources, ...). Dans notre fichier, nous avons une commande de copie de fichiers de configurations qui utilise la commande « xcopy ». Cette commande est conçue pour être utilisée dans Windows, mais n'est pas reconnue sur *linux*. Or mon *Dockerfile* est utilisé sur *Linux*.

Voulant absolument garder la compatibilité avec le debug sous Windows, j'ai dû trouver une solution. Et après plusieurs heures de recherches, j'ai trouvé qu'un fichier .csproj peut contenir des conditions et des variables. J'ai ajouté dans le fichier *Dockerfile* la variable « *Docker*Build » pour indiquer que je compile sous *linux* et désactive donc la commande de copie avec une condition. Sous *Docker,* la copie sera réalisée avec un script. Grâce à ce système utile, j'ai pu rendre le fichier compatible Windows et *Linux,* et éviter la duplication du fichier.

Ensuite, une fois nos services définis avec les *Dockerfile*. Une « extension » de *Docker* : ***Docker*-compose**, permet de coordonner le tout. C'est également un fichier texte (.yml) qui va ordonner le démarrage des services. On préfère ce moyen de démarrer les services car il donne un contexte et clarifie les opérations, surtout dans notre cas où l'on doit démarrer plusieurs *instance*s d'un même service.

Voici le début du fichier *Docker*-compose.dev.yml (fichier complet en annexe 4) :

```yaml
version: '3.4'

x-services-volume:
    - &observo-data-storage
        type: bind
        source: /mnt/c/DATA/ObservoDocker/pool${POOL_NUMBER}/Observo
        target: /app/Observo
    - &observo-sources-storage
        type: bind
        source: /mnt/c/projects/ObservoCross
        target: /app/ObservoCross

services:
  admin_webapi:
    container_name: observo_admin_webapi_pool${POOL_NUMBER}
    image: observo_admin_webapi:${OBSERVO_VERSION}
    build:
      context: .
      dockerfile: ObservoForm.Server/WebApi/ObservoForm.Server.WebApi.Admin/dev.Dockerfi
le
    ports:
      - "${WEBAPI_ADMIN_PORT}:80"
    environment:
      ConnectionStrings:ConnectionString: ${CONNECTION_STRING}
      FilesRepositoriesConfig:FrontendRepositoryPath: /app/Observo
      FilesRepositoriesConfig:BackendRepositoryPath: /app/Observo
      FilesRepositoriesConfig:BackendLocalRepositoryPath: /app/ObservoCross
    depends_on:
      - sql-server-db
    volumes:
      - *observo-data-storage
      - *observo-sources-storage
    networks:
      - observo-network
… (95 lignes)
```

Figure 10 : Fichier docker-compose.dev.yml d'exemple

Dans ce fichier, j'ai dû trouver un moyen de modifier dynamiquement certains paramètres comme les *volumes* de stockage des données. Après quelques recherches, j'ai découvert une nouvelle manière de déclarer les *volumes*. Il *s'agit de volumes* partagés entre les services avec une possibilité de dynamiser les liens vers les dossiers dans mon cas :

`source: /mnt/c/DATA/ObservoDocker/pool${POOL_NUMBER}/Observo`

J'ai la variable « *pool* » qui change et qui modifie l'emplacement de mon dossier source pour le stockage des données de mes Apis.

Les fichiers *Dockerfile* et Docker-compose ont un défaut dans mon cas. Ils sont statiques, donc ne peuvent pas être adaptés à notre usage multi-instances. Heureusement, j'ai trouvé comment définir des variables d'environnements pour modifier partiellement ceux-ci ; Les ports, versions,

pools, paramètres de connexions sont dynamisés et viennent d'un fichier. env défini au lancement du *Docker* compose.

Le fichier. env est réalisé de cette manière :

```
POOL_NUMBER=1
SQL_SERVER_PORT=1433
WEBAPI_MOBILE_PORT=8000
WEBAPI_EXCHANGE_PORT=8001
WEBAPI_ADMIN_PORT=8002
OBSERVO_ADMIN_PORT=8003
HANGFIRE_PORT=8004
WEBAPI_OBSREPO_PORT=8005
```

Figure 11 : Exemple de fichier de variables d'environnement

Finalement, j'ai choisi de faire un **script Bash** pour pallier ce manque de dynamisme. J'ai simplifié le plus possible ce script et je l'ai placé dans le fichier *Make*. Je peux alors préparer les bons fichiers d'environnement avant le démarrage des services.

Comme nous le verrons par la suite, il a plusieurs utilités plutôt simples comme l'exécution des commandes de build, le démarrage des services, la copie de fichiers et l'initialisation des données (base de données, fichiers du projet). Chaque commande *Make* exécutée fait donc référence à ce script.

c. Préparation du serveur *Docker*

Pendant mon travail sur ce projet, j'ai aussi eu l'occasion de me placer en tant qu'administrateur système et réseaux.

Avec l'aide de Laurent VONAU, un collègue du service du support, nous avons configuré le serveur permettant le déploiement des services *Docker*. Ce serveur nous permettrait d'héberger l'environnement de test d'ObsRepo, et de faire un premier essai de la nouvelle architecture *Docker*.

J'ai conseillé l'installation de « Portainer », un outil *open source* permettant d'avoir une interface graphique de tout l'univers *Docker* (Démarrage, arrêt, logs, statistiques des conteneurs, *volumes*, réseaux, images, …)

J'ai aussi préparé une registry *Docker* privée sur le serveur pour permettre le stockage des images générés Azure DevOps.

L'ensemble de l'environnement et des configurations Docker nécessaires étant prêtes, je continue mon travail en réalisant l'intégration et le déploiement automatique du projet.

5. Microsoft Azure DevOps

C'est dans la prochaine section de ce manuscrit que nous verrons le déploiement automatique du projet Observo. Nous verrons comment j'ai préparé le système puis imaginé les différents schémas et scripts de déploiement.

Nous utilisons Azure DevOps pour le déploiement *Docker* depuis déjà trois ans. Nous stockons notre code, déployons nos projets uniquement avec Azure DevOps qui est auto-hébergé dans l'entreprise. C'est pourquoi j'ai continué à utiliser ce système pour mon travail sous *Docker*. Comme c'est la première fois que nous utilisons *Docker* dans l'entreprise, il a fallu préparer la connexion du serveur *Linux,* que nous venons de créer, et DevOps. Nous allons évoquer cela dans la prochaine partie.

a. L'agent de build DevOps

Pour déployer sur un serveur prenant en charge *Docker*, j'ai dû chercher dans la documentation d'Azure DevOps. En effet, ayant déjà travaillé sur du déploiement *Docker* à titre personnel, je connaissais les principes du déploiement automatique.

Pour réaliser ce travail, j'ai suivi pas à pas la documentation très détaillée des étapes sur le site de Microsoft.

Le fonctionnement est plutôt simple, un service dit « agent » doit être présent sur un serveur possédant l'environnement nécessaire au build du projet. Dans mon cas, il me faut un serveur avec le système d'exploitation *Linux* ainsi que *Docker* et Docker-compose.

Étant donné que nos serveurs sont pour la plupart sur le système d'exploitation Windows Server, nous devions trouver une distribution *linux* apte à faire cette tâche. Nous avons choisi *linux* Debian 11 pour sa stabilité et sa réputation pour une utilisation en tant qu'*OS* pour serveur. Afin de réaliser ce premier point, j'ai à nouveau demandé à mon collègue Laurent de m'assister à la création de ce serveur. Nous avons décidé de créer cet agent dans un environnement sécurisé et partagé, dans le but d'être un jour utilisé par les autres projets de l'entreprise.

Grâce à la documentation du projet, et malgré quelques petites difficultés nous avons installé un agent de build *Docker* sous *linux*. C'est sur ce serveur que nous enverrons les fichiers sources (code source du projet), et que nous

les compilerons pour y récupérer des images *Docker*. Nous pourrons réutiliser ces images par la suite pour le déploiement sur les serveurs de production.

Cet agent doit également être connecté pour être utilisable depuis Azure DevOps, directement en réseau interne à l'entreprise.

C'est l'agent qui initie la connexion vers DevOps, cette connexion se fait en *HTTP*. Une fois cette connexion établie, l'agent est prêt à être utilisé et s'ajoute automatiquement dans les agents disponibles au build. Les caractéristiques de l'agent sont également évaluées afin de permettre certaines options comme la compilation, pour un environnement *Docker*.

b. Le déploiement automatique avec DevOps

Une fois le serveur d'hébergement finalisé, nous allons nous attarder sur la partie qui va finaliser mon travail.

Le déploiement avec l'outil Azure DevOps a été un sujet assez complexe puisque j'ai dû chercher à de nombreuses reprises comment réaliser ce que je voulais faire. La version que nous utilisons est : Azure DevOps 2019 auto-hébergé. Il n'intègre donc pas toutes les fonctionnalités *Docker* nécessaires pour simplifier mon travail. J'ai donc dû trouver des solutions alternatives telles que les scripts de déploiement.

1. Build pipeline

Ensuite, il m'a fallu créer les deux pipelines de build et release.

Voici le schéma du pipeline de build :

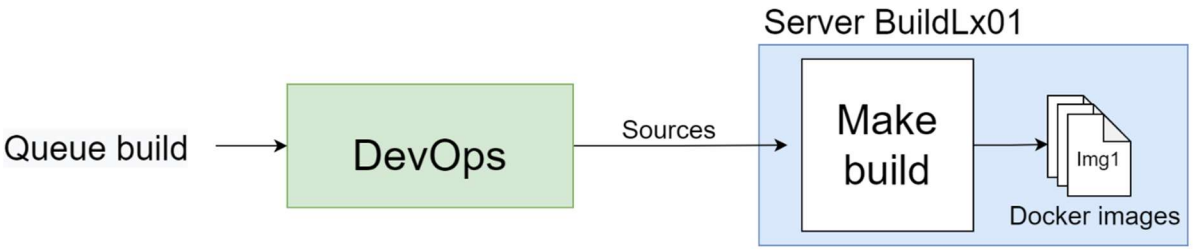


Figure 12 : Schéma du pipeline de build dans DevOps

Le pipeline peut démarrer avec un build automatique (déploiement automatique la nuit), ou bien manuellement quand quelqu'un force un build.

DevOps va préparer et exécuter cette demande et envoyer les sources (code, paramètres) à l'agent de Build. Celui-ci exécute le fichier *Make* build qui fait les tâches suivantes :

- Prépare les fichiers d'environnement

On combine plusieurs fichiers d'environnement préparés en fonction des paramètres choisis. Par exemple, si l'on choisit le *pool* 1, on va mettre les ports en lien avec ce *pool* choisi.

- Exécution de la commande Docker-compose build

On lance la commande de build *Docker*. Celle-ci va créer l'ensemble des images et les stocker sur le serveur.

- On envoie les images sur la Registry Docker située sur l'autre serveur de déploiement pour les utiliser lors du pipeline de release.

```
C'est quoi Make ?

« Make est un petit outil qui fonctionne avec un fichier « Makefile ». C'est un fichier
conçu pour faciliter l'exécution de petits scripts. Souvent combiné aux projets il perme
t de rapidement savoir comment compiler et éxécuter le projet. »
```

2. Release pipeline

L'étape finale de tout ce travail est le déploiement sur le serveur. Nous allons utiliser les images générées par le pipeline de build pour les démarrer sur le serveur de déploiement.

Voici le schéma du pipeline de release :

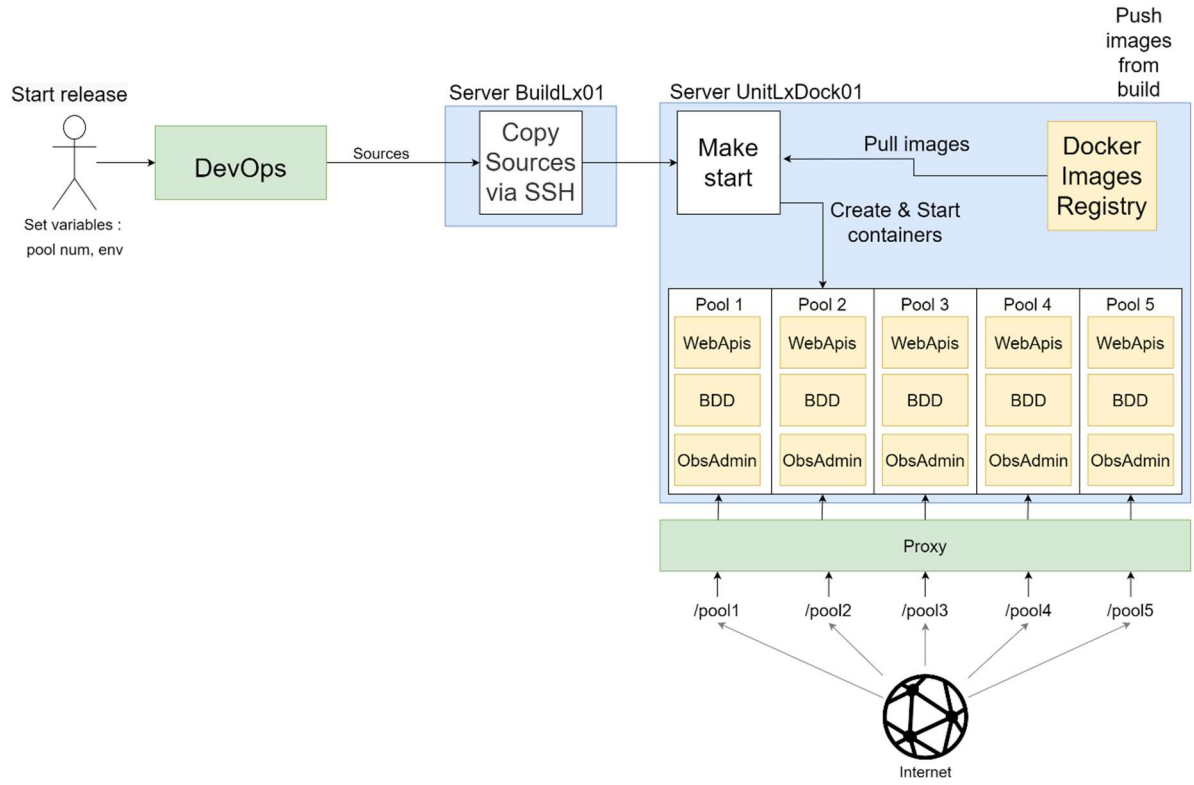


Figure 13 : Schéma du pipeline de release dans DevOps

Celui-ci est plus complexe que le pipeline de Build puisqu'il comporte la partie d'hébergement.

Le build peut être demandé par deux manières différentes. Soit un utilisateur le lance manuellement, soit un processus automatique le lance au cours de la nuit.

Lors de cette exécution du build, il est nécessaire de renseigner des variables pour exprimer le besoin de l'environnement voulu.

L'utilisateur a le choix entre les environnements Intégrateur, Test, Démonstration ou Production lors du déploiement.

DevOps ramène à nouveau les sources pour disposer des scripts et des fichiers de configurations sur le serveur de « build ». Celui ne fait que de copier les fichiers sources de DevOps vers le serveur de déploiement.

Ensuite, nous démarrons le script *Make* start. Voici ce que fait précisément ce script :

- *Pull* des images depuis la Registry Docker

C'est ici que l'on récupère le travail réalisé par le pipeline de build. Les images sont téléchargées depuis la Registry Docker.

- Exécution de Docker-compose up

On lance la commande de démarrage des conteneurs. Les images sont utilisées et chaque conteneur démarre en fonction de sa configuration.

- On actualise les données si besoin

Une fois les services exécutés, on laisse le choix à l'utilisateur de repartir d'un environnement « propre » de la production, cela permet d'actualiser de temps en temps les données des environnements de tests.

L'utilisation de deux pipelines se justifie, puisque nous voulons déployer une multitude d'*instances* avec les mêmes fichiers de build. Seules quelques variables changent pour modifier le *pool* et l'environnement. (Schéma complet en annexe 3)

Le choix du fonctionnement par *pool* a été réfléchi par Laurent VONAU et moi. Laurent ayant beaucoup d'expérience dans le domaine de l'administration système et réseau, je me suis adressé à lui concernant le choix des ports et des *volumes* de stockages. Après notre réunion, nous en avons conclu que l'utilisation de « *pool* » des emplacements prédéfinis sont plus appropriés à

notre problématique. Pour imager cela, on pourrait voir cela comme des places de parkings (le *pool*) (dans notre cas on le limite à cinq *pools* pour l'instant). Chaque place à des prises avec la connexion vers l'extérieur (ports réseau et stockages). On vient y garer des voitures (*instance* d'Observo) pour les faire fonctionner sur une place et utiliser les ressources de la place.

Et donc on peut au final avoir cinq *pools* de déploiement permettant de démarrer plusieurs *instances* d'Observo sur le même serveur et avec des configurations différentes. Le tout bien évidement accessible depuis nos machines sur le réseau local de l'entreprise, permet d'effectuer les tests sur le projet.

6. Problèmes restants

Finalement, le plus gros problème que j'ai rencontré se trouve au niveau du projet Hangfire. C'est un service de file d'attente de job, et nous l'utilisons principalement pour la génération des rapports d'inspection. Le module de génération de rapport est une technologie assez ancienne et malheureusement dépréciée. Il s'agit de faire le rendu avec Microsoft Word et de faire un export PDF. Cela pose problème car les conteneurs *Docker* fonctionnant sur *Linux* la suite Microsoft Office n'est pas disponible.

La solution que j'ai trouvée est la suivante : créer un micro-service qui fonctionne sous Windows et qui se charge uniquement de la génération des rapports. Celui-ci serait connecté à Hangfire (hébergé dans *Docker*), et il serait partagé entre les environnements. Cela nous simplifierait la mise à jour du moteur de génération de formulaires.

Après quelques jours de réflexion, nous avons analysé le pour et le contre de la mise en place de cette solution. Nous avions finalement remarqué que ce n'est pas la totalité des rapports qui sont générés avec Office, mais seulement une minorité (moins de 5 sur plusieurs dizaines). Nous avons donc conclu que mettre en place une telle architecture n'était pas nécessaire pour s'adapter à une technologie vieillissante. La génération par Office n'était apparemment même plus supportée par Microsoft depuis la version Office 2016. La personne s'occupant de ce générateur de rapports est Mr. Thierry Moebel, le chef d'entreprise. Il nous a averti qu'il utilise à ce jour une nouvelle librairie issue de la suite *Syncfusion*, et qu'il pourra très certainement mettre à jour les derniers rapports utilisant encore l'ancien système.

Ce problème sera donc corrigé prochainement.

Un second problème concerne plutôt l'architecture de *Docker*. Je n'ai pas trouvé le moyen de créer une seule image *Docker* pour l'ensemble des services dans Observo. Cette image nous apporterait plusieurs avantages. Nous pourrions *versionner* l'entièreté du projet en une seule fois, et également partager le tout le plus simplement possible. Cette façon de générer l'image serait en effet plus adaptée à notre usage. Je pense donc encore faire quelques recherches pour trouver une solution. Cependant, il en résulte quand même un avantage d'avoir une image par service. Cela nous permettrait éventuellement de déployer avec une certaine *scalabilité* nos services et d'utiliser un outil comme Kubernetes pour *orchestrer* nos conteneurs.

Concernant ce problème, j'ai demandé à Benjamin SCHUDEL de me conseiller pour corriger ce problème dès que possible.

7. État actuel du projet

Actuellement mon travail est en production, mais uniquement pour l'environnement de test. Je fais encore quelques modifications assez couramment pour l'améliorer. Cela fonctionne bien, mais il manque encore de dynamisme notamment lors de la configuration dans DevOps.

Une mise à jour de DevOps est prévue fin août. C'est pourquoi nous attendons cette nouvelle version pour finir correctement la configuration.

En parallèle, j'ai par ailleurs commencé à utiliser *Docker* sur mon environnement de développement, et j'ai pu correctement démarrer plusieurs fois Observo sur celui-ci. Cela nous fait gagner déjà beaucoup de temps de développement, et surtout cela rend le développement d'ObsRepo possible.

Nous avons hâte de finir totalement ce travail pour l'utiliser en production, réduire nos couts d'hébergement, et pourquoi pas d'utiliser cette solution dans les autres projets de l'entreprise.

CONCLUSION

Afin de conclure ce mémoire, je suis très content d'avoir réalisé ces trois dernières années de ma formation de Master dans l'entreprise Unit Solutions. Depuis maintenant trois ans, j'évolue sur le projet Observo en abordant des points très divers et variés. Cette année, j'ai apprécié mon travail sur le déploiement *Docker* qui m'a permis de découvrir de nombreux aspects du déploiement d'un projet.

De même, j'ai pu grandement approfondir mes connaissances sur le projet Observo, ainsi que sur ses technologies. L'entreprise me fournit un cadre de travail idéal pour progresser et appliquer mes connaissances dans un milieu professionnel.

Ce travail conséquent, et qui certes relève plus du domaine de l'architecture que celui du développement, nous a tout de même permis de gagner un temps précieux lors de nos tests et développements futurs. J'ai ainsi pu répondre à la problématique du déploiement multi-*instance*s, et ainsi permettre le début du développement du projet satellite ObsRepo. Celui-ci nous a fait également gagner du temps et de l'organisation dans l'équipe du projet.

Nous utilisons maintenant mon travail dans un cadre de tests et de développement, et il sera prévu d'utiliser cette solution également pour la production dans quelques mois.

Je suis ravi d'avoir continué à contribuer sur le projet Observo. J'apprécie beaucoup ce projet en pleine effervescence.

D'années en années, je gagne beaucoup en autonomie et en rapidité, ce qui me permet également de me positionner en tant qu'aide pour le support du projet et en tant qu'assistant du chef de projet.

Depuis maintenant deux ans j'ai aussi pu travailler avec Tom WILLEMIN, et j'ai eu l'occasion de beaucoup collaborer avec lui. Nous avons beaucoup échangé concernant notre savoir, et cela nous a permis à tous deux d'améliorer notre qualité de travail.

Mon tuteur m'a également très bien accompagné, ce qui m'a permis d'être très efficace dans mon travail et de prendre plus de responsabilités dans ce projet.

Je suis très impatient de continuer à poursuivre mon travail pour Unit Solutions durant les années à venir.

Retour de l'entreprise :

« Pour sa dernière année d'alternance, Thomas a su une nouvelle fois relever les défis qui lui ont été proposés. Le bilan global de son alternance est au-dessus de nos attentes.

Il a su acquérir les compétences nécessaires pour dockeriser le projet Observo, élément indispensable au projet pour les développements en cours et améliorer nos processus de travail.
Thomas a su monter l'architecture en complète autonomie tout en respectant les standards exigés.
La qualité de la documentation élaborée a notamment permis d'expliquer, comprendre et valider son travail avec les différents acteurs du projet.

Nous sommes heureux de pouvoir compter Thomas comme collaborateur à l'issue de sa formation en Master. » **Mr. Thibault VELLICUS, Chef du projet Observo**

Bibliographie :

Cloux, P., Garlot, T., Kohler, J. (2022). *Docker* et conteneurs - 3e éd.: Architectures, développement, usages et outils. (n.p.): Dunod.

Grubor, S. (2017). Deployment with *Docker*: Apply Continuous Integration Models, Deploy Applications Quicker, and Scale at Large by Putting *Docker* to Work. United Kingdom: Packt Publishing.

Chcomley. (2023, 20 juillet). Présentation d'Azure DevOps - Azure DevOps. Microsoft Learn. https://learn.microsoft.com/fr-fr/azure/devops/user-guide/what-is-azure-devops?toc=%2Fazure%2Fdevops%2Fserver%2Ftoc.json&bc=%2Fazure%2Fdevops%2Fserver%2Fbreadcrumb%2Ftoc.json&view=azure-devops-2022

Unit Solutions : https://unit.solutions/

Microsoft : https://www.microsoft.com/fr-fr

Glossaire :

Terme	Définition
.NET Core	Framework Microsoft en grande partie basé sur le langage *C#*, facilite la programmation en proposant un environnement de développement.
Angular	Framework client web qui est basé sur le langage Typescript et maintenu par Google.
Arduino	Plateforme d'initiation à électronique simple pour la réalisation de petits projets électroniques interactifs.
C#	C# est un langage de programmation crée par Microsoft et utilisé pour une multitude de projet avec la plateforme *.NET*
CI/CD	Le CI/CD où l'intégration et déploiement continu est un processus d'intégration du code dans les flux de production DevOps.
Docker	Docker est un outil qui aide à créer et exécuter des applications dans des environnements virtuels.
Docker Hub	Plateforme de partage d'images *Docker* ouverte au public.
Dockerfile	Fichier *Docker* qui comprend une liste d'instructions pour la création d'images *Docker*.
GitHub	Plateforme créée en 2007 principalement utilisée par les développeurs pour héberger du code et le partager.
HTTP	HyperText Transfer Protocol est un protocole pour le transfert de requêtes entre client et serveur sur le web.
Instance	Une « copie » d'un élément qui peut se paramétrer pour agir différemment que la source.
Kernel	*Kernel* ou noyau est le cœur d'un système d'exploitation d'un ordinateur.
Linux	Système d'exploitation *open source* et gratuit principalement utilisé pour des serveurs.
Make	Petit outil qui permet l'exécution de commandes simples pour la création automatique d'un exécutable à partir du code source.
Microsoft Access	Microsoft Access est un outil permettant de faire de la gestion avancée de tableaux de données avec de la connexion dynamique à des bases de données.
Microsoft Azure	Plateforme cloud offrant toutes les solutions cloud nécessaires à une entreprise (hébergement, stockage, réseau, …)
Multi-instanciation	Fait d'avoir plusieurs *instances*.
Open source	Logiciel libre pouvant être utilisé et modifié gratuitement par le public.
Orchestrer	Contrôler un ensemble d'éléments comme un chef d'orchestre dirige les musiciens.
OS	« Operating system » (système d'exploitation en français)

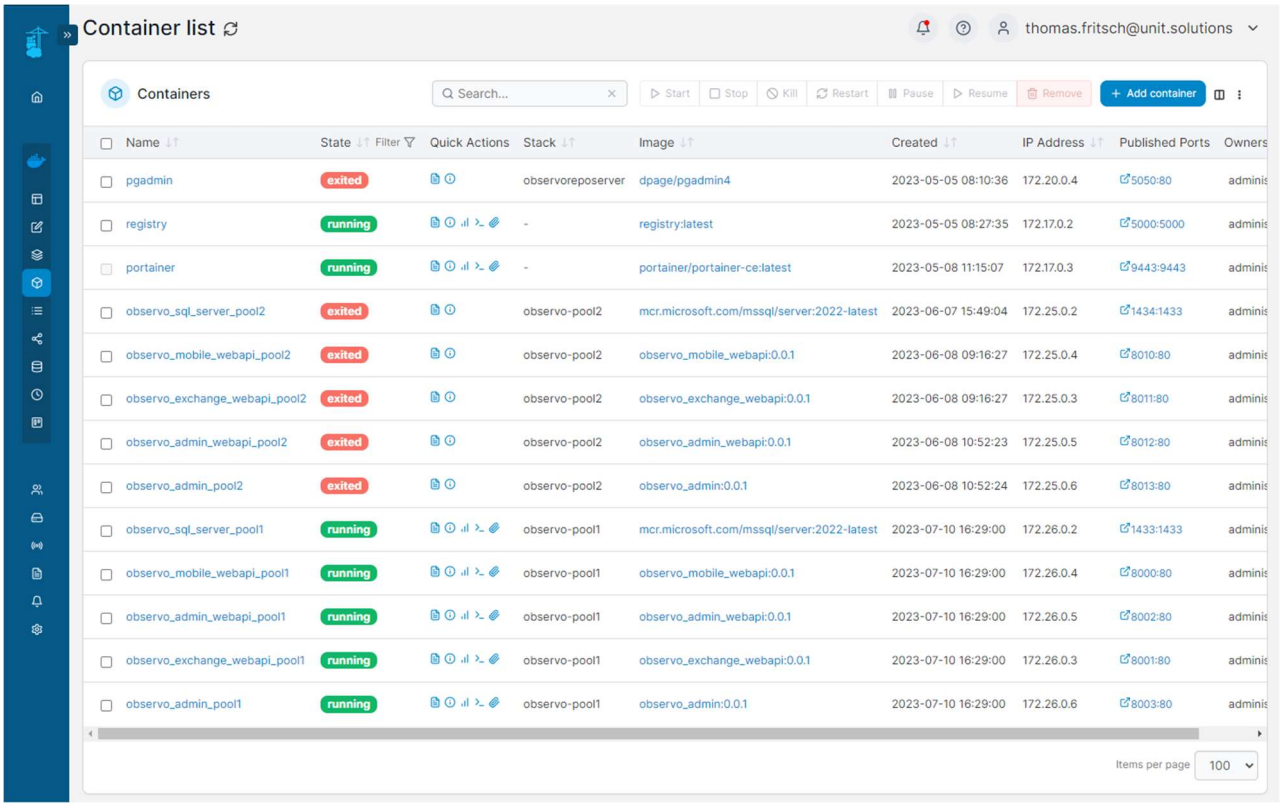
	est l'ensemble des programmes et services qui permettent l'utilisation des ressources matérielles par un utilisateur.
Pool	Un pool est un ensemble de ressources communautaires vouées à être utilisées génériquement entre plusieurs services.
Pull	Action de récupérer du code, fichiers de build, images d'un serveur.
RAM	La RAM est la mémoire vive d'un ordinateur permettant le travail sur plusieurs logiciels en parallèle.
Scalabilité	Capaciter à un produit de pouvoir s'adapter vers des tailles supérieures ou inférieures.
Self-hosted	Possibilité pour un logiciel d'être hébergé sur son propre serveur. Cela a souvent l'avantage d'avoir un coût plus faible de l'hébergement.
Sprint	Un sprint (présent dans la méthode agile) est un lapse de temps de quelques semaines pendant lequel une équipe de développeurs contribue à l'avancement d'un projet.
Syncfusion	Fournisseur de composants et bibliothèques de développement pour les entreprises.
Versionner	Action d'appliquer un numéro de version à un projet.
Virtualisation	Processus de création et utilisation d'un outil, d'un programme, d'un service sous forme virtuelle dans un environnement isolé.
VM	Abréviation de Machine Virtuelle, il s'agit d'un ordinateur que l'on virtualise pour avoir un environnement séparé de la machine hôte.
Volumes (Docker)	Dans *Docker* un volume est un élément de stockage des fichiers. C'est par un volume que le conteneur accède aux fichiers de la machine hôte.
WebApi	Une web API (Application Programming Interface) est un service faisant interface entre un serveur et un projet, elle peut être utilisée selon un standard REST API.
Xamarin	Xamarin est un Framework *C#* permettant de faire des applications mobiles avec un code partagé entre les plateformes Android, iOS et Windows Phone.

Annexes :

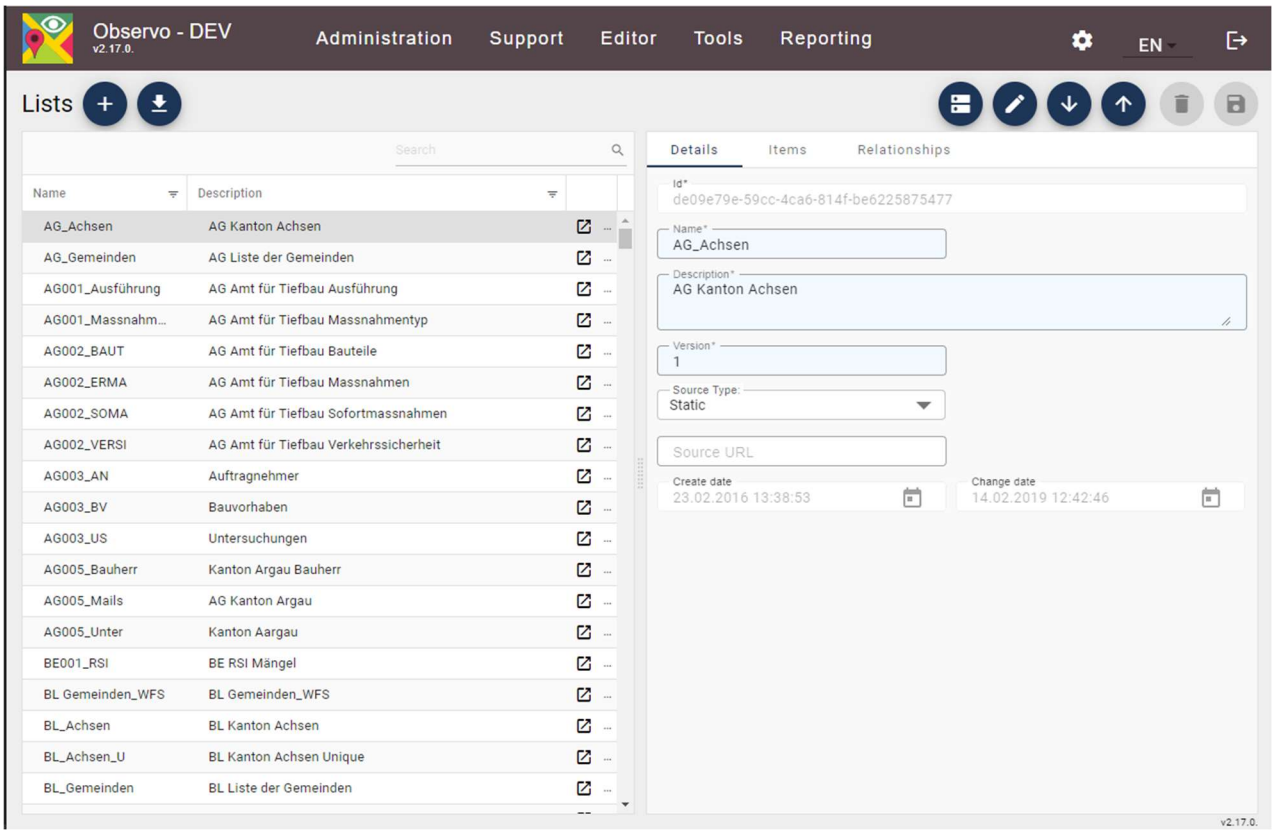
Table des annexes :

n°1. Aperçu de Portainer 1
n°2. Aperçu d'ObservoAdmin 2
n°3. Schéma des pipelines de Build et Release d'Azure DevOps 2
n°4. Fichier *Docker*-compose.dev.yml complet 3
n°5. Pipeline de déploiement avec le choix du pool dans Azure DevOps 5
n°6. Étapes du pipeline de déploiement dans Azure DevOps 5

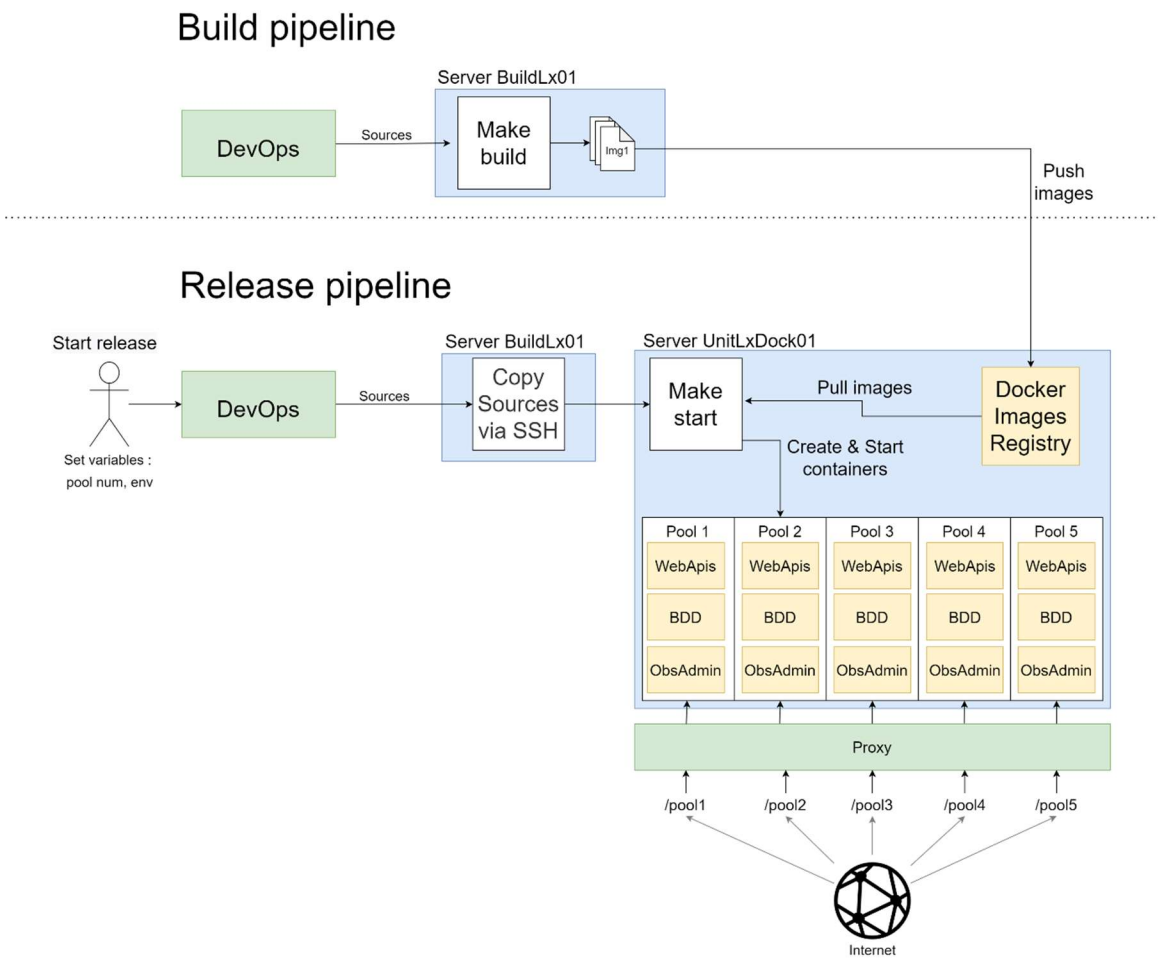
Annexe n°1 – Aperçu de Portainer :



Annexe n°2 – Aperçu d'ObservoAdmin :



Annexe n°3 – Schéma des pipelines de Build et Release d'Azure DevOps :

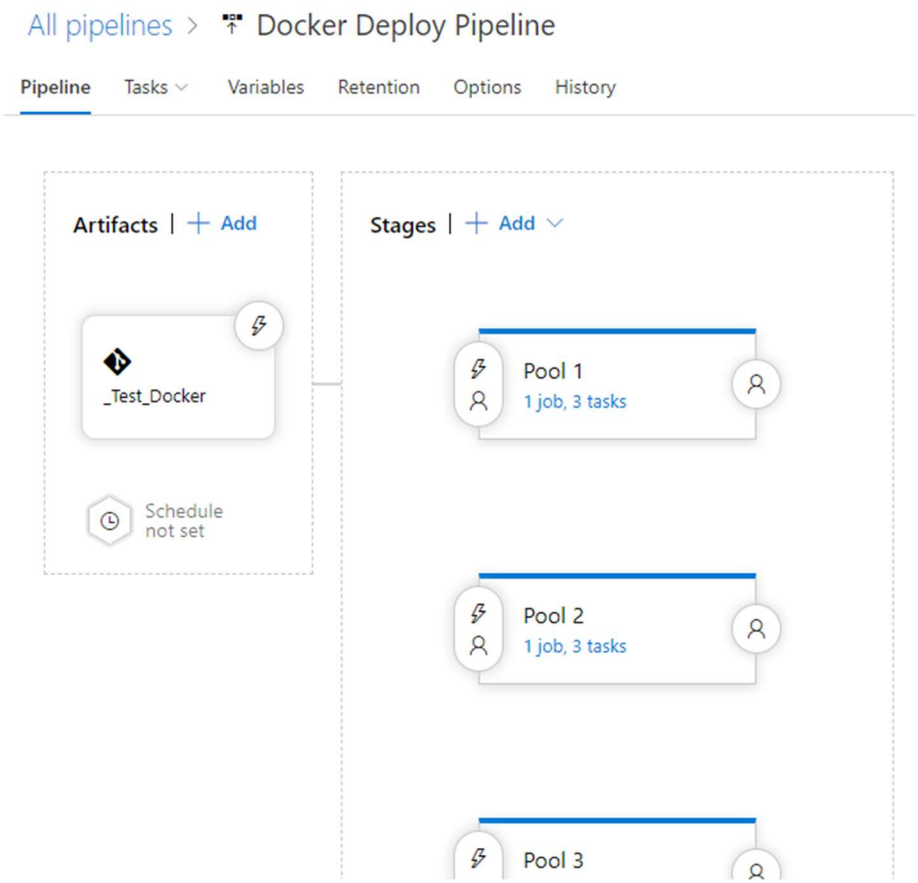


Annexe n°4 – Fichier *Docker*-compose.dev.yml :

```yaml
version: '3.4'

x-services-volume:
    - &observo-data-storage
        type: bind
        source: /mnt/c/DATA/ObservoDocker/pool${POOL_NUMBER}/Observo
        target: /app/Observo
    - &observo-sources-storage
        type: bind
        source: /mnt/c/projects/ObservoCross
        target: /app/ObservoCross

services:
  mobile_webapi:
    container_name: observo_mobile_webapi_pool${POOL_NUMBER}
    image: observo_mobile_webapi:${OBSERVO_VERSION}
    build:
      context: .
      dockerfile: ObservoForm.Server/WebApi/ObservoForm.Server.WebApi.Mobile/dev.Dockerf
ile
    ports:
      - "${WEBAPI_MOBILE_PORT}:80"
    environment:
      ConnectionStrings:ConnectionString: ${CONNECTION_STRING}
      FilesRepositoriesConfig:FrontendRepositoryPath: /app/Observo
      FilesRepositoriesConfig:BackendRepositoryPath: /app/Observo
      FilesRepositoriesConfig:BackendLocalRepositoryPath: /app/ObservoCross
    depends_on:
      - sql-server-db
    volumes:
      - *observo-data-storage
      - *observo-sources-storage
    networks:
      - observo-network

  admin_webapi:
    container_name: observo_admin_webapi_pool${POOL_NUMBER}
    image: observo_admin_webapi:${OBSERVO_VERSION}
    build:
      context: .
      dockerfile: ObservoForm.Server/WebApi/ObservoForm.Server.WebApi.Admin/dev.Dockerfi
le
    ports:
      - "${WEBAPI_ADMIN_PORT}:80"
    environment:
      ConnectionStrings:ConnectionString: ${CONNECTION_STRING}
      FilesRepositoriesConfig:FrontendRepositoryPath: /app/Observo
      FilesRepositoriesConfig:BackendRepositoryPath: /app/Observo
      FilesRepositoriesConfig:BackendLocalRepositoryPath: /app/ObservoCross
    depends_on:
      - sql-server-db
    volumes:
      - *observo-data-storage
      - *observo-sources-storage
    networks:
      - observo-network
```

```yaml
exchange_webapi:
    container_name: observo_exchange_webapi_pool${POOL_NUMBER}
    image: observo_exchange_webapi:${OBSERVO_VERSION}
    build:
      context: .
      dockerfile: ObservoForm.WebApi/ObservoForm.Server.WebApi.Exchange/dev.Dockerfile
    ports:
      - "${WEBAPI_EXCHANGE_PORT}:80"
    environment:
      ConnectionStrings:ConnectionString: ${CONNECTION_STRING}
      FilesRepositoriesConfig:FrontendRepositoryPath: /app/Observo
      FilesRepositoriesConfig:BackendRepositoryPath: /app/Observo
      FilesRepositoriesConfig:BackendLocalRepositoryPath: /app/ObservoCross
    depends_on:
      - sql-server-db
    volumes:
      - *observo-data-storage
      - *observo-sources-storage
    networks:
      - observo-network

  obsRepo_webapi:
    container_name: observo_obsrepo_webapi_pool${POOL_NUMBER}
    image: observo_obsrepo_webapi:${OBSERVO_VERSION}
    build:
      context: .
      dockerfile: ObservoForm.WebApi/ObservoForm.Server.WebApi.ObsRepo/dev.Dockerfile
    ports:
      - "${WEBAPI_OBSREPO_PORT}:80"
    environment:
      ConnectionStrings:ConnectionString: ${CONNECTION_STRING}
      FilesRepositoriesConfig:FrontendRepositoryPath: /app/Observo
      FilesRepositoriesConfig:BackendRepositoryPath: /app/Observo
      FilesRepositoriesConfig:BackendLocalRepositoryPath: /app/ObservoCross
    depends_on:
      - sql-server-db
    volumes:
      - *observo-data-storage
      - *observo-sources-storage
    networks:
      - observo-network
  observo_admin:
    container_name: observo_admin_pool${POOL_NUMBER}
    image: observo_admin:${OBSERVO_VERSION}
    build:
      context: ObservoForm.Angular/.
      dockerfile: Dockerfile
    ports:
      - "${OBSERVO_ADMIN_PORT}:80"
    depends_on:
      - admin_webapi
    networks:
      - observo-network
  sql-server-db:
    container_name: observo_sql_server_pool${POOL_NUMBER}
    image: mcr.microsoft.com/mssql/server:2022-latest
    ports:
      - "${SQL_SERVER_PORT}:1433"
    environment:
      MSSQL_SA_PASSWORD: "**********"
      ACCEPT_EULA: "Y"
    volumes:
      - observo-bdd:/var/opt/mssql
    networks:
      - observo-network
volumes:
  observo-bdd:
networks:
  observo-network:
    driver: bridge
```

Annexe n°5 – Pipeline de déploiement avec le choix du pool dans Azure DevOps :



Annexe n°6 – Étapes du pipeline de déploiement dans Azure DevOps :

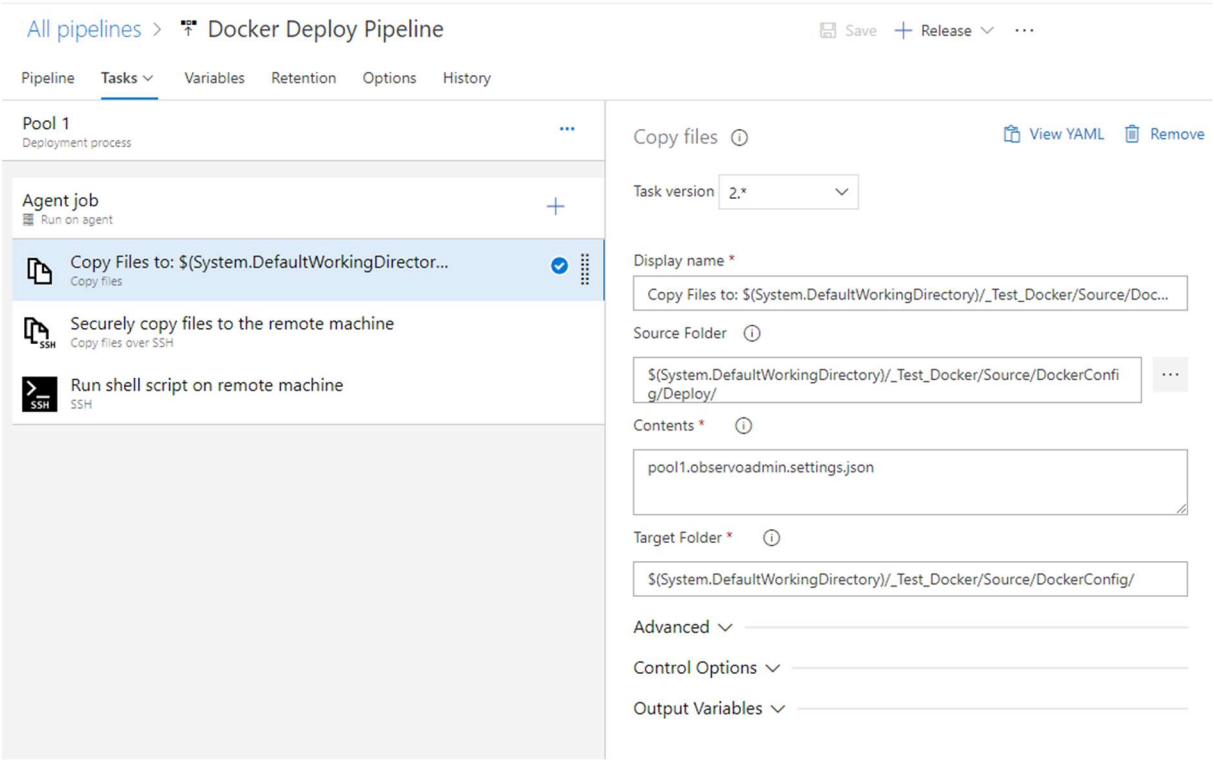


Table des matières :

Remerciements

Introduction 1

I. CONTEXTUALISATION 2

1. Mon parcours 2
2. Mes projets en formation 2

II. PRÉSENTATION DU PROJET 4

1. L'entreprise UNIT SOLUTIONS 4
2. Présentation du projet Observo 4
3. Les méthodes et outils de travail 8
   a. Confluence 8
   b. Jira 9
   c. Azure DevOps 9
   d. Visual Studio 9
   e. Webstorm 9

III. L'ÉTAT DE L'ART 10

1. La virtualisation légère avec *Docker* 10
   a. Présentation de la plateforme 10
   b. Les principes de base 10
   c. Docker compose 11
   d. Utilisation et avantages de Docker 11
   e. Comparaison de Docker avec la virtualisation matérielle 12
   f. Conclusion 13
2. Intégration continue et déploiement continu avec Azure DevOps 14
   a. Introduction 14
   b. Les composants d'Azure DevOps 14
   c. Azure DevOps avec Docker 14
   d. Conclusion 15

IV. RÉALISATION 16

1. Introduction 16
2. Cahier des charges 18
3. L'architecture Docker 21
4. L'implémentation de cette architecture 24
   a. L'environnement de développement 24
   b. Les fichiers et scripts de configuration 26
   c. Préparation du serveur Docker 30
5. Microsoft Azure DevOps 31
   a. L'agent de build DevOps 31
   b. Le déploiement automatique avec DevOps 32
      i. Build pipeline 32
      ii. Release pipeline 33
6. Problèmes restants 35

7. Etat actuel du projet 36
Conclusion 37
Bibliographie 39
Glossaire 40
Annexes
Résumé et mots-clés

RÉSUMÉ

Ce mémoire détaille ma seconde et dernière année de master en alternance dans l'entreprise Unit Solutions. J'ai pu travailler sur une application mobile durant ces neuf mois d'entreprise, au sein de l'équipe de développement d'Observo.

Mon travail a consisté à réaliser une architecture Docker du projet, dans le but de pouvoir créer une multitude d'environnements, en peu de temps. En effet, grâce à cela, nous pouvons développer et tester le projet dans le cadre du développement d'un nouvel outil satellite : ObsRepo.

Dans ce mémoire, vous allez donc découvrir comment j'ai analysé, préparé et créé cette architecture de déploiement sous Docker et Azure DevOps.

Mots–clés :

- Docker
- Azure DevOps
- Déploiement continu
- Multi-instanciation
- Architecture