

Etude et mise en place d'un pipeline d'intégration et de déploiement orienté DevOps

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Timothée Molleyres

Conseiller au travail de Bachelor :

Ciaran Bryce, Professeur HES associé

Genève, 01.05.2023

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de Gestion

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre de Bachelor .

L'étudiant a envoyé ce document par email à l'adresse remise par son conseiller au travail de Bachelor pour analyse par le logiciel de détection de plagiat URKUND, selon la procédure détaillée à l'URL suivante : <https://www.orkund.com> .

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Vevey, le 01.05.2023

Molleyres Timothée

A handwritten signature in black ink, appearing to read 'Molleyres' followed by a stylized flourish.

Remerciements

Je souhaite remercier Monsieur Ciaran Bryce pour m'avoir suivi pendant ce travail ainsi que pour sa disponibilité et ses conseils.

Résumé

Ce travail vise à expliquer l'utilisation des outils DevOps et leur valeur ajoutée en prenant comme exemple central un pipeline d'intégration et de déploiement continu. Il est accompagné d'un projet fictif, reprenant une grande partie des outils proposés et les intègre dans un pipeline. De nombreuses parenthèses sont faites afin d'expliquer l'avantage que pourrait avoir l'adoption de ces outils par de petites entreprises.

Le travail commence par une introduction expliquant les motivations ayant conduit à sa réalisation ainsi que le contexte. La première partie explique l'évolution des méthodologies et des architectures liées au monde du développement, ceci afin de poser une trame ayant conduit à la situation actuelle.

La deuxième partie explique les concepts et technologies importants pour l'utilisation des outils, notamment, le cloud et l'infrastructure as a Service, la conteneurisation, les gestionnaires de code, ainsi que la sécurité. Ceci afin d'expliquer les synergies et utilités des différents outils intégrés au pipeline.

La troisième partie se focalise sur le pipeline et les cycles d'intégrations et de développement continu. Elle aborde les outils permettant de réaliser ces deux cycles. Elle décortique chacun des cycles afin de comprendre l'utilité et les outils liés à la résolution de chaque étape du cycle. Elle compare certains outils ayant la même utilité et propose des utilisations pour certains outils indépendamment du pipeline. Certaines étapes du pipeline sont réalisées dans le cadre du projet afin d'explicitier leur fonctionnement.

Une conclusion vient clore le travail avec un récapitulatif, un regard critique sur le travail et une note personnelle.

Table des matières

Déclaration	i
Remerciements	ii
Résumé	iii
Liste des tableaux	Error! Bookmark not defined.
Liste des figures.....	v
1. Introduction.....	1
2. Historique des pratiques de développement d'application	3
2.1 De Monolitique à Microservices	4
2.2 DevOps	7
3. Concepts et technologies	10
3.1 Le Mini-Projet	11
3.2 Le Cloud Computing et l'infrastructure as a Service.....	12
3.3 La sécurité	13
3.3.1 OWASP Top 10	15
3.4 Contrôleur de version	18
3.5 Conteneurisation.....	21
3.5.1 Docker	23
3.6 Résumé.....	30
4. L'outil au cœur du cycle d'automatisation.....	31
4.1 Critères de choix de solution	31
4.2 Continuous Integration.....	32
4.2.1 Les étapes décortiquées	33
4.2.2 Résumé Continuous Intégration	36
4.3 CI/CD TOOLS.....	37
4.3.1 OUTILS	38
4.4 Continous Deployment	40
4.4.1 Terraform pour provisionnement de ressources et Ansible pour le management de configuration	43
4.4.2 SCANNER DE VULNERABILITE	50
4.4.3 Continuous Monitoring	51
5. Conclusion	53
Bibliographie	54
Annexe 1: GitHub Repository	59

Liste des figures

Figure 1 : Waterfall vs Agile.....	4
Figure 2 : Le DevOps schématisé	8
Figure 3 : Schéma d'architecture du projet	11
Figure 4 : De la découverte à la publication d'une vulnérabilité	14
Figure 5 : Les 10 risques les plus répandus pour une application web.	16
Figure 6 : Création d'un dépôt Git sur Github	19
Figure 7: Initialisation de la connexion au dépôt distant.....	20
Figure 8 : La conteneurisation	23
Figure 9: Notre Dockerfile.....	25
Figure 10: Notre Dockercompose.....	28
Figure 11 : Le cycle d'intégration continu	32
Figure 12 : Notre jenkinsfile.....	40
Figure 13 : Le cycle de déploiement continu	41
Figure 14 : Zoom sur l'architecture du projet	43
Figure 15 : Le fonctionnement de Terraform imagé	45
Figure 16 : Notre fichier Terraform	46
Figure 17 : Kubernetes Architecture	49

1. Introduction

Dans un monde où les services informatiques se retrouvent à être le socle fonctionnel d'une grande partie de la société, les attentes et les standards qui en découlent deviennent de plus en plus élevés. Le temps où la complexité d'un système était une excuse pour son mauvais fonctionnement est révolu.

Mais dans une société qui évolue rapidement, maintenir et améliorer des programmes a créé une myriade de professions et de spécialisations. La tâche est d'autant plus ardue qu'il est attendu que tout se passe sans interruption de service pour les utilisateurs.

Aujourd'hui, les équipes dans les entreprises ayant embrassé le monde du DevOps n'ont plus que rarement à se soucier d'un jour de mise en production de leur nouvelle version, la culture de développement à évoluer, le reste de la chaîne de mise en service s'y est plié et de nouveaux outils sont nés. Des outils qui malheureusement semblent rencontrer des difficultés à trouver leur chemin dans les petites entreprises.

Après avoir fini ma formation de Bachelor en informatique de gestion, je me suis rendu compte que mon savoir se regroupait en plusieurs parties et me permettait d'avoir une bonne compréhension de l'univers informatique mais certains points restaient obscurs. Notamment, le fonctionnement de la mise en production du travail effectué par les développeurs. Je me suis bien rendu compte que pour un projet disposant d'un peu de complexité et de plusieurs personnes travaillant sur son contenu, il semblait difficilement envisageable de simplement prendre le code et de le déposer sur un serveur de production. Il n'était pas possible que notre monde actuel, tellement dépendant de ses outils numériques, se repose sur un processus aussi peu structuré et prompt à générer de nombreux problèmes.

Comme beaucoup d'étudiants j'ai travaillé durant mes études. Je me chargeais de la gestion de projet pour l'intégration d'outils informatiques nécessitant très souvent des Middleware. Même si certains de mes clients se trouvaient être des grosses entreprises, la majorité était également des PME. Je ne compte plus le nombre de fois où je me suis retrouvé face à des outils dits legacy que personne ne voulait toucher. L'environnement de production de ceux-ci était traité comme un volcan qu'on espérait ne jamais réveiller. Il existait également des projets dont la complexité rendait compliqué toutes modifications car personne ne semblait vraiment savoir comment il fonctionnait.

Je me suis également rendu compte que ce que je pensais impossible existait encore en beaucoup d'endroits. Quelques outils dits de DevOps traînaient par-ci par-là mais étaient plus utilisés dans le but d'automatiser le déploiement du code dans son environnement de production qu'autre chose. Le cycle de production modern de réalisation et déploiement d'un logiciel force les bonnes pratiques et évite la création de programmes dits « puants ».

Il est compréhensible que dans toute société où les ressources sont très limitées, s'offrir les services d'un spécialiste du domaine ou d'octroyer du temps à un de ses employés pour travailler sur la mise en place d'outils est complié. Pourtant, comme tout bon investissement, c'est avec le temps que leur valeur apparaîtra.

Le but de ce travail est de rendre compréhensif, à toute personne travaillant en lien avec le monde du développement, ce qui est aujourd'hui appelé le monde du DevOps mais plus particulièrement les cycles d'intégrations et de déploiements continus pour assurer la livraison d'un programme qualitatif et sûr. Il s'agit d'expliquer l'utilité et le fonctionnement des technologies utilisées afin de montrer la plus-value générée par leur mise en place. Ce travail sera accompagné d'un petit projet souhaitant montrer qu'il n'est pas si compliqué d'utiliser ces outils.

Ce travail est également rédigé dans le but de donner une vision plus claire de ce domaine à d'autres étudiants ou toute personne souhaitant le comprendre. Il vulgarise un grand nombre de concepts et est également pensé pour être lu par des décideurs dans de petites entreprises. J'espère qu'il leur donnera envie de fournir les ressources nécessaires à leur équipe pour mettre en place certains outils expliqué et ainsi amener une nette amélioration en termes de qualité, sécurité et maintenabilité de leurs applications.

Je commencerai par décrire comment nous sommes arrivés à la situation actuelle et d'où viennent les problèmes existants. Viendrons ensuite l'explication de concepts et technologies importants sur lesquels est basé le monde du développement actuel. Enfin, je parlerai des étapes et outils liés à la réalisation d'un pipeline assurant la délivrance d'un logiciel propre et sécurisé. Cette dernière étape sera accompagnée d'un projet.

2. Historique des pratiques de développement d'application

Afin de comprendre comment nous sommes arrivés dans la situation actuelle, il est important de revenir sur l'évolution du monde du développement logiciel.

Les pratiques de développement ainsi que les architectures des logiciels ont énormément évoluées ces dernières années. Il est important que comprendre que les deux sont extrêmement corrélées.

Lorsque le développement d'application moderne à commencer à voir le jour, toutes les technologies d'un projet étaient utilisées ensemble pour fournir le service souhaité. Le terme « ensemble » est même quelque peu léger, les technologies devaient être configurée et utilisées comme un seul procédé unifié. Cette façon de fonctionner était très lourde et comportait de nombreux désavantages. L'approche de gestion de projet pour le développement appelé « Waterfall » (en cascade) y est pour beaucoup. Le principe est simple : chaque étape du processus est dépendante de celle qui la précède. On réfléchit à l'ensemble des besoins du projet, on en déduit une architecture, on commence ensuite à travailler en général de la partie logique vers la partie interface utilisateur et enfin on teste. Ainsi tout problèmes apparaissant plus loin dans la cascade des étapes ne peut que difficilement être adressé étant donné que l'ensemble du projet a déjà été construit¹.

Cette façon de fonctionner rencontra assez rapidement un nombre de problèmes conséquents dans une société ayant des besoins évoluant rapidement. Bien que les logiciels livrés puissent fonctionner correctement, il s'avère compliqué de les modifier en profondeur si le besoin venait à évoluer. L'utilisateur final réel du logiciel ne peut que très peu le tester avant d'en avoir une version complète. Une durée de plusieurs années peut être nécessaire avant d'obtenir une version fonctionnelle et les besoins ont souvent évolués entre temps.

Les technologies évoluant rapidement, et les interconnexions entre les systèmes devenant de plus en plus nécessaires, avoir une application isolée et difficilement modifiable n'était pas une solution cohérente pour permettre de répondre aux besoins des utilisateurs.

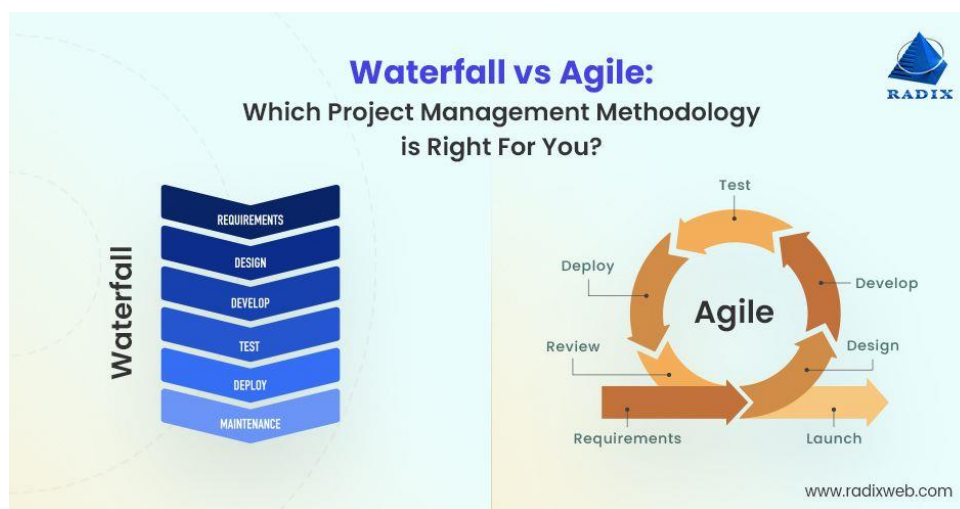
¹ HAMILTON, Thomas, 2023. « Agile vs Waterfall-Difference Between Methodologies »

Dans cette optique, les parties intégrantes d'un logiciel ont été peu à peu découplées et ont ainsi commencé à dialoguer entre elles par l'intermédiaire d'interfaces. C'est cette même philosophie qui a amené les pratiques à évoluer dans les mêmes directions pour l'ensemble des composants d'un logiciel. Dans le début des années 2000's, arriva le manifeste de l'agilité.

Dans cette optique d'acceptation du changement constant, le monde du développement devait s'adapter et avec lui les outils nécessaires apparurent.

L'agilité se focalisant sur les processus avec une création incrémentale et récursive, amenant avec elle le besoin de pouvoir délivrer ces changements au produit utilisé facilement et rapidement. Ce changement de paradigme força la façon d'architecturer et de développer les logiciels à changer. C'est ainsi que l'architecture générale des logiciels, leur cycle de déploiement et de vie ont fait peau neuve.

Figure 1 : Waterfall vs Agile



Source: PITALIYA, Sarrah, 2023. « Waterfall vs Agile: What to Know Before Choosing Your Development Method »

2.1 De Monolithique à Microservices

Comme énoncé précédemment, lorsque le développement logiciel a fait son apparition, une application était constituée d'un gros bloc et pouvait essentiellement être pensée comme un seul projet unique contenant l'ensemble du code et des services nécessaires pour faire marcher toutes les fonctionnalités du logiciel. Cette architecture porte le nom

de « Monolithe »². Cependant, avec un nombre de fonctionnalités croissant et une interconnexion avec d'autres logiciels, ce bloc unique posait de nombreuses limitations si le logiciel souhaitait évoluer.

Toutes les fonctionnalités et, par extension, le service qui se charge de les faire marcher sont fortement couplés. En effet, vu qu'il s'agit d'un seul projet, une modification sur un service peut avoir un effet péjorant sur un autre. Ce qui, en plus des bugs amène un niveau de complexité croissant au logiciel lorsque celui-ci grandit.

La mise en production d'un projet à l'architecture monolithique comportait également quelques obstacles qui avaient la fâcheuse tendance de se révéler qu'une fois le logiciel mis en production. En effet, le processus n'était pas bien compliqué, il s'agissait de packager le projet sous la forme d'un gros artefact au format souvent propre au langage et de le déposer sur le serveur de production et le tour est joué. Le problème étant que l'environnement du serveur de déploiement était configuré manuellement et il n'était pas du tout le même que celui du développement voire différent que celui de test. Tout cela à cause de modifications non répercutées, de versions d'OS différentes ou de quantité de ressources mise à disposition³.

Pour les équipes, il était courant de se retrouver à devoir tout lâcher pour régler un problème en production, surtout si celui-ci n'était pas répliquable dans d'autres environnements.

Les logiciels étant de plus en plus portés sur une utilisation à travers le web, le problème de pouvoir supporter la demande croissante des utilisateurs se trouver être problématique pour une application en Monolithe. En effet, une fois la capacité maximum d'un serveur atteint, il serait possible d'en faire tourner un second puis un troisième et ainsi de suite, puis d'équilibrer la charge entre les serveurs. Cette option pose de nombreux problèmes suivant le type d'applications notamment si des données doivent être disponibles pour chaque utilisateur, celui-ci devra toujours utiliser le même serveur. Cette architecture est également gourmande en ressources, car il est impossible d'augmenter les ressources pour un seul service du logiciel, il faut provisionner un serveur étant capable de tenir si tous les services sont utilisés au maximum de leur capacité en même temps.

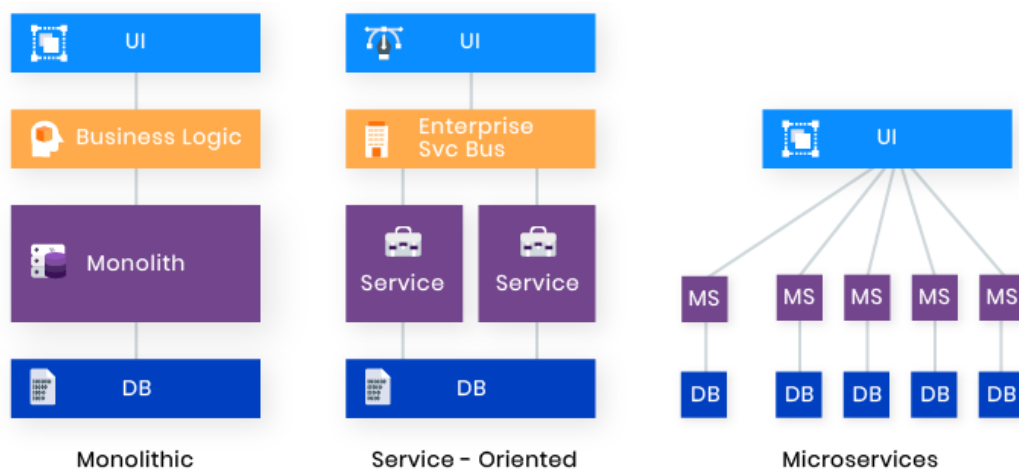
² HARRIS, Chandler, 2022. « Microservices vs. monolithic architecture »

³ HARRIS, Chandler, 2022. « Microservices vs. monolithic architecture »

Avec tous ces désavantages, il a rapidement été compris que cette architecture ne permettrait pas du tout de proposer des logiciels pouvant évoluer et s'interconnecter facilement, et encore moins qu'ils soient utilisables par un grand nombre d'utilisateurs simultanément.

C'est pour cela que l'industrie a rapidement changé de cap et a commencé par découpler les services et les rendre suffisamment petits pour que leur compréhension, amélioration et maintenance soient facilitées. C'est ainsi que les micro-services sont nés.

Figure 1 : Monolithic vs Microservices



D, Anastasia et H, Dmytro, 2019. « Best Architecture for an MVP ».

Attention, cela ne veut pas dire qu'une architecture monolithique n'a pas sa place encore aujourd'hui. Suivant la complexité du projet et/ou la quantité d'utilisateurs voués à utiliser le service, une architecture plus simple peut tout à fait faire sens.

En apparence, les micro-services semblaient résoudre beaucoup de problèmes, mais comme tout changement, il en amenait avec lui de nouveaux. Avoir une myriade de petits services simplifiait grandement leur développement, mais leur mise en production, quant à elle, était devenue beaucoup plus complexe. En effet, chaque service demandait son lot de dépendances et configuration, et maintenir des environnements de développement, de test et de production propre à chaque service était un casse-tête. Cela couplé aux releases fréquentes imposées par les nouvelles normes de développement, le goulet d'étranglement qu'était la mise en production imposa un besoin d'amélioration et d'automatisation.

2.2 DevOps

Pendant de nombreuses années, les développeurs se chargeaient de coder le logiciel et les équipes opérationnels de mettre en place et maintenir les environnements de productions.

Comme très souvent lorsque deux rôles doivent s'appuyer sur le travail de l'un et de l'autre, l'intersection pose un problème, qui a la responsabilité de quoi et jusqu'où. En d'autres termes, les développeurs travaillant avec leur environnement local de développement ou un serveur dédié rempli avec mille et une dépendances, souvent dans le but de tests ou pour améliorer le projet, ils fournissent un projet pour eux fonctionnel sans trop se soucier de la suite. A l'inverse, les responsable des systèmes préparent des environnements bien pensés, et construits pour être sécurés et maintenables. Lorsque le projet passe des mains des premiers aux seconds, il arrivait très souvent que les dépendances ou configurations requises par le projet pour fonctionner ne soient pas les mêmes que celles dont les environnements préparés disposent. Surtout, lorsque l'on parle de développement continu, très souvent il peut même s'agir d'oubli de passage d'information.

Afin d'harmoniser le cycle de développement, il fallait joindre les deux mondes. C'est ainsi qu'est né le DEVOPS, jonction entre le monde du développement et des opérations. Le terme est souvent utilisé dans un sens très large englobant les pratiques de CI/CD, ainsi qu'un grand nombre de procédés et pratiques en lien avec la livraison de logiciel et les changements d'infrastructure résultant de leur utilisation.

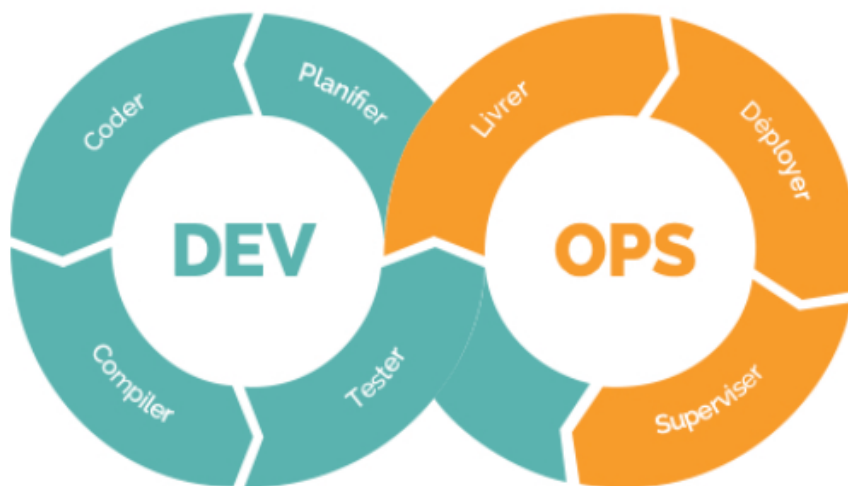
« Que signifie DevOps pour les équipes ? DevOps permet la coordination et la collaboration des rôles autrefois cloisonnés (développement, opérations informatiques, ingénierie qualité et sécurité) pour créer des produits plus performants et plus fiables. En adoptant une culture DevOps ainsi que des pratiques et outils DevOps, les équipes peuvent mieux répondre aux besoins des clients, accroître la confiance suscitée par les applications qu'elles développent, et atteindre plus rapidement les objectifs de leur entreprise. »⁴

Ces pratiques et façons de penser ont le mérite d'avoir grandement facilité le bon fonctionnement du monde informatique. En se basant sur des technologies comme la conteneurisation ainsi que des outils dédiés, les pratiques dites DEVOPS ont permis de soutenir l'évolution du développement et de la livraison de logiciel en permettant que l'ensemble des étapes se déroulent sans accroc et apportent une vraie plus-value au produit final.

⁴ AZURE, 2022. *Qu'est-ce que DevOps?*

C'est d'ailleurs pour ceci que le terme DEVOPS est souvent associé à une double boucle liée ou le symbole infini. Le croisement représentant l'intersection du développement et des opérations, chaque boucle comprenant les étapes de la vie d'une release logicielle liée à son milieu. Les professionnels liant leur activité à cette idéologie se chargeront de faire la jonction entre les deux mondes mais ne feront véritablement ni du développement, ni de la maintenance de système.

Figure 2 : Le DevOps schématisé



Source : PONTHEU, Theo, 2020. « Le DevOps expliqué à mes parents (et à nos collègues non-initiés »

La situation actuelle, qui est une conséquence cette évolution, est mitigée. Les grandes structures ont suivi la cadence et se retrouvent avec des architectures basées sur des micro-services, profitant de process d'intégration et de déploiement très bien calibrés. C'est pourquoi il est aujourd'hui très rare de rencontrer des applications indisponibles et même si cela arrive, le délai avant la reprise de service est en général court.

Cependant, le constat n'est pas le même les plus petites structures ou les organisations étatiques. Mon expérience de plusieurs années m'a permis de dresser le portrait suivant. Comme tout le monde, les PME ont tenté de prendre le train en marche, essayant d'adopter des concepts lorsque le budget le permettait. Une entrave à l'adoption de nouvelles pratiques est liée aux équipes à disposition. Celles-ci étant limitées, les choix technologiques se basent alors sur les connaissances d'une personne de l'équipe dans un domaine. Si cette personne venait à quitter l'entreprise, il est difficile de la remplacer. Cela couplé à des équipes dont le cahier des charges très remplie ne laissait pas

beaucoup de place à la formation continue a amené beaucoup de PME à se concentrer sur la réalisation du programme et à mettre de côté le reste.

Les architectures dans les PME sont variables mais beaucoup disposent encore de nombreuses ressources en interne, d'un grand nombre de solutions logicielles faites maison ou créées sur la base d'un logiciel mais ne pouvant plus évoluer avec ce dernier, ainsi que des processus de tests souvent très lacunaires ou inexistant.

C'est cet environnement qui est en partie responsable du manque de sécurité dans ces structures, car si les bugs fonctionnels et les services indisponibles sont problématiques et pénibles, les failles de sécurité peuvent être mortelles.

Je suis conscient que le travail demandé nécessaire à résoudre ces problèmes ne se résume pas à la mise en place de quelques outils et l'adoption de certaines pratiques. Mais il suffit de pas grand-chose pour déjà aller vers le mieux, un pipeline de déploiement dans lequel les bons outils sont intégrés permet déjà de s'assurer que les programmes déployés ne souffrent pas de trop de failles de sécurité et qu'un bon nombre de bugs soient évités. Certaines technologies liées au monde du DevOps permettent également de faciliter de nombreuses actions, réduire les erreurs humaines et assurer un suivi. En adoptant certains de ces outils, je suis persuadé que des économies de coûts seront réalisées et de nombreuses bonnes pratiques adoptées sans pour autant demander plus de travail.

3. Concepts et technologies

Avant de se lancer dans la réalisation du pipeline et de ses parties, je vais ici introduire le concept et expliquer certaines technologies et outils liés nécessaires au bon fonctionnement de notre pipeline. Certains, même utilisés individuellement, apportent une amélioration par rapport à la situation actuelle dans certaines entreprises.

Jusqu'à présent, le terme « mise en production » a été utilisé. Il est clair qu'avant qu'une version de logiciel puisse être mise à disposition pour son utilisation, celle-ci doit passer à travers un certain nombre d'étapes. Chacune d'entre elles ayant pour but de garantir au maximum que la version ne contienne pas de bug, qu'elle soit conforme à un niveau de qualité souhaité, et qu'elle surtout soit fonctionnelle.

C'est dans cet esprit qu'une chaîne d'actions en grande partie automatisés a vu le jour sous le nom de pipeline d'intégration et de distribution continues.

« Un pipeline CI/CD est une série d'étapes à réaliser en vue de distribuer une nouvelle version d'un logiciel. Les pipelines [d'intégration et de distribution continues \(CI/CD\)](#) désignent une pratique qui consiste à améliorer la distribution de logiciels ⁵»

Le mot « pipeline » est utilisé métaphoriquement, il s'agit d'un tuyau qui transportera notre nouvelle version du logiciel de la fin de sa création à sa mise en production. Comme nous le verrons, cette version sera minutieusement vérifiée à chaque étape avant qu'elle ne puisse pour continuer son chemin dans le tuyau.

Les termes intégration et distribution continues sont souvent couplés et acronymisés par leur version anglaise : CI/CD pour Continuous Integration /Continuous Delivery. L'acronyme englobe très souvent également Continuous Deployment(Déploiement continu).

La partie CI se focalisera sur la récupération du projet une fois le pipeline déclenché, ainsi sur que la partie test pré-déploiement.

La partie CD se focalisera sur l'envoi du projet dans ses différents environnements, ainsi que sur les tests post-déploiement.

Le pipeline dans son ensemble récupérera le projet mis à jour par le développeur, il construira l'environnement attaché pour vérifier sa bonne construction, passera un certain nombre de tests sur le code, déploiera ensuite le projet sur les différents

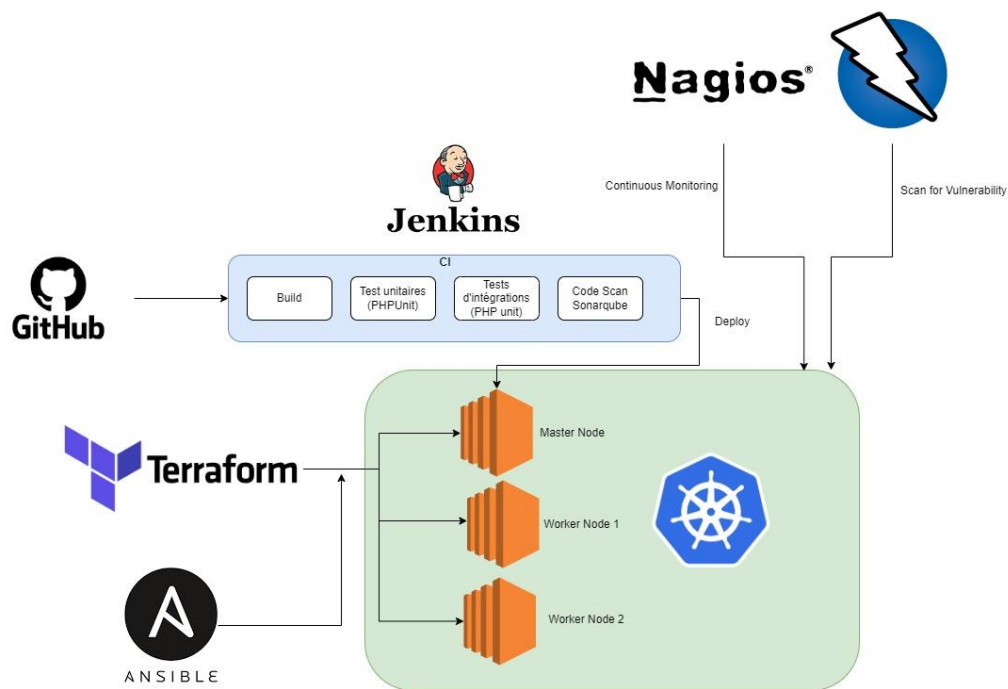
⁵ RED HAT, 2023. *What is a CI/CD pipeline?*

environnements. Le projet sera ensuite testé sous sa forme complète. Une fois validé, il sera mis en production. Le but est de fournir un feedback à chaque étape et ainsi permettre aux personnes responsables de régler les problèmes leur incombant.

Les étapes peuvent varier mais un certain nombre d'entre-elles sont toujours mises en place. Nous reviendrons sur chacune des parties plus précisément.

3.1 Le Mini-Projet

Figure 3 : Schéma d'architecture du projet



Voici l'architecture de notre mini-projet. Celui-ci a pour objectif de service d'exemple tout au long du travail. Toutes les parties seront expliquées mais ne seront pas forcément réalisées. Le but est de mettre en avant l'usage que pourraient avoir les outils sans trop tomber dans la complexité.

Pour l'instant aucun point de ce schéma n'a été abordé, mais au fur et à mesure qu'ils le seront. Une référence sera faite à ce schéma.

Les étapes d'installation des outils et de configuration des plug-ins nécessaires au fonctionnement ne seront pas décrites afin d'éviter de dupliquer de la documentation existante et d'alourdir ce travail.

3.2 Le Cloud Computing et l'infrastructure as a Service

De nombreuses entreprises disposent encore aujourd'hui de petits datacenters en interne, et ce pour différentes raisons. Premièrement, car certains outils aux besoins spécifiques y fonctionnent encore. Deuxièmement, car lorsque les serveurs se trouvent chez soi, il y a une impression de maîtrise et de sécurité. Ce genre d'infrastructure coûte cher et demande du personnel qualifié, surtout si les logiciels mis à disposition doivent être fonctionnels 24h/24. Sans compter que l'ajout de nouveaux outils génère un besoin d'augmentation des ressources ce qui peut mener à des coûts élevés même pour l'ajout d'outils peu gourmands. Les solutions d'offres de ressources dites cloud constituent une alternative qui, si elle est bien utilisée, peut réduire grandement les coûts.

Le cloud aujourd'hui est un terme à la mode, il a permis d'offrir tous les outils anciennement disponibles pour l'utilisateur uniquement via une installation et une configuration à un service accessible en ligne. Son autre composante est l'offre d'IAAS (Infrastructure as a service), autrement dit la mise à disposition d'infrastructure contre paiement, le prix étant basé uniquement sur l'utilisation des ressources⁶. Cela a une grande incidence sur le coût découlant de l'utilisation de certains outils. En effet, pendant de nombreuses années, les entreprises devaient provisionner leur infrastructure pour tenir la charge maximum dont elles avaient besoin et accepter que malheureusement, le reste du temps, une partie potentiellement conséquente de leurs ressources ne serait pas utilisée. Cela avait souvent pour conséquence que l'ajout d'un outil était équivalent à l'ajout d'une charge fixe, ce qui générait une réticence de la part du management. Les équipes de petites entreprises se limitaient donc souvent dans le panel de leurs outils. C'est pourquoi beaucoup d'outils de tests ainsi que leur utilisation automatique peinent à faire leur chemin.

Heureusement, grâce aux solutions cloud, deux possibilités s'offrent aux entreprises. La première est l'utilisation d'outils dits cloud, autrement dit tournant sur l'infrastructure de leur éditeur. Bon nombre de ces outils offre des prestations uniquement à l'utilisation, évitant ainsi la maintenance de l'outil et les charges fixes liées à son utilisation. Cela permet l'accès à des outils poussés et maintenus à jour même pour de petites structures. Il y a cependant un petit hiatus : certaines de ces solutions commencent à baser leur modèle non plus sur un coût à l'utilisation mais sur le paiement d'un tarif fixe, ce qui risque à nouveau de refroidir les utilisateurs disposant de plus petits moyens.

⁶ CLOUDFLARE, 2022. *Qu'est-ce que le cloud*

La seconde possibilité qui s'offre aux entreprises pour l'utilisation du cloud sont les outils open-source couplés à l'offre IAAS (infrastructure as a Service). Le gros avantage des outils open-sources est qu'ils sont gratuits, ce qui allège déjà une partie du problème. Reste celui de l'infrastructure nécessaire à son utilisation. L'offre IAAS gérée avec des outils d'automatisation de provisionnement de ressources est une excellente façon de limiter la facture. Nous reviendrons plus précisément sur ces outils plus loin. Bien pensés et configurés, ces outils permettent de mettre en place ou d'activer uniquement les ressources nécessaires au fonctionnement des outils au moment de leur utilisation. Avec le coût d'une instance d'uniquement quelques dizaines de centimes à l'heure, l'utilisation de ce type de solution permet une réelle économie de coût, et représente ainsi une option viable pour des petites structures.

3.3 La sécurité

Dans le monde actuel, la sécurité d'une application prend une place élevée dans les priorités émises lors de la conception. Il est extrêmement important de s'assurer que lors que des modifications sont apportées à l'application, celles-ci ne compromettent pas le travail effectué pour sa sécurité.

Dans de nombreuses petites entreprises, la sécurité est souvent l'élément le plus laissé pour compte. Comme tous les frais liés à la protection contre des situations pouvant se produire mais n'étant pas certaines, il est difficile de convaincre les décideurs de fournir suffisamment de ressources pour s'en protéger. Certaines entreprises préfèrent même s'assurer financièrement contre les dommages que pourrait leur causer une attaque plutôt que d'essayer de s'en protéger correctement. Un effort est souvent fait sur la sécurisation de l'infrastructure de la société mais rarement sur ses produits. Pourtant, la perte ou le vol de données peut être synonyme de faillite. Selon une étude récente (Hausmann, 2022), de trop nombreuses sont les PME suisse ne disposant pas des outils miniums mis en place pour assurer leur sécurité.⁷

Je vais tâcher d'expliquer comment les vulnérabilités sont traitées et répertoriées, ensuite je listerai les plus communément découvertes sur des programmes et comment il est possible, à l'aide des outils que nous verrons, de les déceler.

⁷ HAUSMANN, Roger, 2022. « Les PME ne prennent pas suffisamment au sérieux la cybersécurité ».

Dans le monde de l'informatique, les vulnérabilités sont appelées CVE pour « Common Vulnerability and Exposure », les faiblesses pouvant générer une vulnérabilité sont nommées CWE pour « Common Weakness Enumeration »⁸.

Les CVE sont basés sur une organisation du même nom ayant mis en place un procédé de report et de suivi des vulnérabilités, ces dernières couvrent l'ensemble du monde informatique, des systèmes aux applications. Les CVE sont annoncés uniquement par des partenaires et entreprises reconnus afin d'assurer la véracité de l'information, le système étant devenu la base du partage d'information lié à la sécurité. De nombreuses entreprises proposent même des récompenses pour l'annonce de vulnérabilité sur les produits. Permettant ainsi de profiter de la grande communauté présente autour du projet et d'assurer la sécurité de ses produits à moindre coût. En effet même si certains « bug bounty » comme ils sont appelés, peuvent atteindre plusieurs centaines de milliers de dollars, les sommes sont souvent beaucoup plus basses et elles sont des sommes risibles pour de grandes entreprises. A titre d'exemple, Google a payé un peu plus de 12 millions de dollars pour un équivalent de 2'900 vulnérabilités reportées sur ses produits⁹.

Figure 4 : De la découverte à la publication d'une vulnérabilité



Source: CVE, 2023. CVE Numbering Authority CAN Rules [en ligne].

L'idée est de centraliser le suivi et d'accorder à chaque vulnérabilité une note indiquant le degré de risque qu'encourt une application et son système si rien n'est entrepris pour remédier à la vulnérabilité. Toute vulnérabilité est loin d'être catastrophique et ne requiert pas forcément d'y remédier instantanément.

Afin d'assurer un suivi pas trop compliqué, une CVE doit respecter quelques critères.

⁸ CVE, 2023. *CVE Numbering Authority CAN Rules*

⁹ IONUT, Ilascu, 2022. « Google paid \$12 million in bug bounties to security researchers »

- En plus d'être annoncé par une source fiable, elle doit être reconnue par l'autorité représentant le produit sur lequel elle a été découverte.
- Elle doit être solvable indépendamment d'autres bugs, par exemple si une vulnérabilité est liée aux fonctionnements de plusieurs applications ou dépendances fonctionnant ensemble, une CVE sera créée pour chaque « codebase », autrement dit une collection de code source utilisée pour faire fonctionner un programme¹⁰.

Une fois la solution trouvée pour résoudre la vulnérabilité, celle-ci est ajoutée aux informations de la CVE. C'est sur cette base de données que l'ensemble de l'industrie fonctionne, permettant aux logiciels dédiés à la recherche de vulnérabilité sur les systèmes et applications de savoir quoi chercher et de proposer une résolution à ses utilisateurs.

3.3.1 OWASP Top 10

Comme discuté précédemment, les vulnérabilités sont répertoriées lorsqu'il s'agit d'un problème lié à un produit utilisé. La très grande majorité des applications aujourd'hui sont basées ou utilisent des produits existants, les CVE permettent donc d'assurer que les outils sur lesquels se reposent notre application ne souffrent pas de vulnérabilités connues. Mais lors de la réalisation d'une application, ce qui en fait un produit est le travail réalisé par les équipes de développement. Ces équipes vont produire du code. Les standards et meilleures pratiques évoluant constamment, notamment pour résoudre des soucis de sécurité, il est très fréquent que le code soit rempli de problèmes pouvant ainsi engendrer des vulnérabilités.

Le monde se dirigeant à grands pas vers une digitalisation omniprésente, la sécurité des outils utilisés dans ce but est cruciale. L'une des fondations les plus connue active dans ce milieu s'appelle The Open Worldwide Application Security Project (OWASP). Il s'agit de la plus grande communauté travaillant sur les aspects liés à la sécurité. Elle propose des outils permettant la mitigation des risques et vulnérabilités ainsi que du matériel et des ressources de formation, permettant de garder le monde de la sécurité à jour.

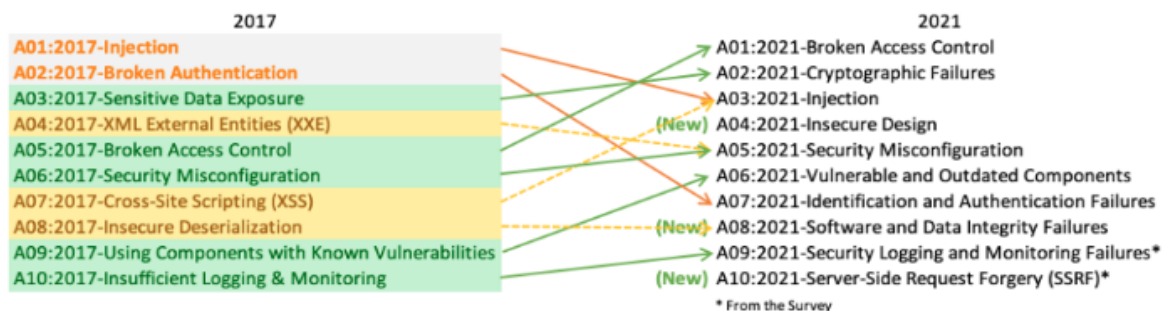
¹⁰ CVE, 2023. *CVE Numbering Authority CAN Rules*

OWASP édite toutes les quelques années une liste contenant les 10 risques permettant des attaques sur une applications les plus répandus. Cette liste est la référence en termes d'éléments devant être vérifiés par une équipe avant de rendre son application disponible à l'utilisation. Etant donné qu'il s'agit des plus connus, il est garanti que ce seront les premières portes d'entrée testées par toute entité souhaitant du mal à votre application. Comme vous l'aurez compris, nous parlons ici des CWE mentionnés plus tôt.

Figure 5 : Les 10 risques les plus répandus pour une application web.

Top 10 Web Application Security Risks

There are three new categories, four categories with naming and scoping changes, and some consolidation in the Top 10 for 2021.



Source : OWASP, 2021. OWASP Top Ten [en ligne].

Comme nous pouvons le voir, les risques les plus présents ont quelques peu évolués ces dernières années. Nous allons nous attarder sur la liste actuelle, nous reviendrons à chaque étape sur les outils permettant de déceler ces risques et autres vulnérabilités.

Broken Acces Control : il s'agit de cas où l'application ne définit par bien ce qui peut être accessible à chaque type d'utilisateur. Très souvent cela est dû à des règles pas assez restrictives que ce soit sur l'appel ou l'accessibilité de certaines ressources. Typiquement, lors de l'utilisation d'API FrontEnd-Back end, la possibilité de récupérer des informations ou pire de les modifier. Ces erreurs sont liées en grande partie au code et devront donc être adressées lors de tests sur le code.

Cryptographic Failures : ici un grand nombre de scénarios sont possibles mais le problème vient de l'accessibilité à l'information lors de son transit entre services en utilisant des protocoles non sécurisés ou dépassés. Il peut également s'agir d'informations sensibles n'étant pas encryptées ou utilisées telles quelles et donc lisibles. Ici, une partie peut être vérifiée sur le code, une autre sur le système, le reste dépendra des bonnes pratiques.

Injection : Il s'agit d'un des risques les plus connus qui semble heureusement être moins présent depuis la dernière mise à jour de la liste. Ici tout champ permettant à l'utilisateur d'entrer des informations ou services pouvant recevoir de l'information qui sera transmise un autre service devrait être vérifiée. Toutes informations fournies par l'utilisateur et utilisé par l'application doivent être contrôlées proprement. Il est sinon possible d'accéder aux données ou de les altérer. Ici, c'est le code qui devra être testé.

Insecure Design : cette catégorie est beaucoup plus large et beaucoup plus difficile à tester, certaines choses peuvent tout de même être relevées comme des informations de connexion, pour accéder à un autre service étant mal protégé ou encore des informations sensibles reportées dans la génération de message d'erreur.

Security Misconfiguration : comme mentionné, les applications développées aujourd'hui s'appuient sur de nombreux outils. L'avantage amené par l'utilisation d'outils peut créer certains risques liés à la configuration de ces outils, par exemple l'ouverture de ports par défaut sur l'outil, ces ports n'étant pas utilisés, ils peuvent être utilisés comme une porte d'entrée. Cette partie peut être testée post-déploiement sur l'infrastructure.

Vulnerable and Outdated Components : cette catégorie s'explique d'elle-même, l'utilisation de services n'étant pas à jour laisse des bugs et vulnérabilités ayant été résolus par l'éditeur du service. Ici, un scan sur le code et post-déploiement sur le système permettra de relever convenablement les problèmes potentiels.

Identification and Authentication Failures : cette partie couvre tous les problèmes liés à une identification trop faible, l'utilisation de valeur par défaut, l'utilisation de valeur de session après une durée trop longue. Certains de ces soucis peuvent être relevés mais pour d'autres il s'agit de bonnes pratiques et standards.

Software and Data Integrity Failures: il s'agit d'une catégorie importante pour notre pipeline, car lorsque nous faisons appel à des dépendances pour notre application, des téléchargements s'effectuent sans vérifier la source et la validité des données téléchargées¹¹. Ici, des logiciels dédiés pour la vérification de dépendances sont disponibles mais la mitigation demande de mettre en place des services n'étant pas nécessaires pour des petits projets. Dans le cas de notre projet, nous nous contenterons de nous baser sur des sources reconnues et des images de bases officielles.

¹¹ OWASP, 2021. *OWASP Top Ten*

Security Logging and Monitoring Failures : la difficulté de tester les problèmes liés à ce groupe de risques est élevée. Même si le Continuous Monitoring, sur lequel nous reviendrons, offre quelques possibilités.

Server-Side Request Forgery(SSRF): il s'agit d'un type de risque très spécifique à l'utilisation d'url transmises par l'utilisateur pour la récupération d'informations, pour par exemple récupérer des données afin de pouvoir les utiliser dans le logiciel¹². Ici , il est également difficile de tester mais certains points peuvent être relevés sur le code.

Grâce à cette liste, nous allons pouvoir nous efforcer d'ajouter tous les outils nécessaires permettant de s'assurer que notre application soit sécurisée une fois validée par notre pipeline. Du moins les développeurs disposeront de toutes les informations nécessaires leur permettant de sécuriser l'application. Il peut être souhaitable que le pipeline bloque le déploiement en fonction des vulnérabilités détectées.

Le but n'est pas de proposer des outils qui vont complexifier la vie des développeurs ou des membres de l'équipe des opérations mais plutôt de leur donner toutes les informations pour leur permettre de régler les vulnérabilités jugées les plus dangereuses. Chaque organisation est responsable de ses choix mais en connaissance de cause, les collaborateurs auront tendance à régler les problèmes exposant leur organisation aux risques les plus élevés.

3.4 Contrôleur de version

Cette technologie est la plus répandue, je n'ai croisé aucune organisation fonctionnant sans elle. Même si certaines ne l'utilisent pas à son plein potentiel. Je ne vais pas m'attarder dessus, mais il n'est resté pas moins que c'est le départ du cycle de vie du déploiement logiciel et le point déclencheur de notre pipeline.

Le choix d'un logiciel de contrôle de version n'est pas un choix fondamentalement important pour ce que nous essayons de réaliser avec notre projet, c'est pourquoi nous ne nous attarderons pas trop longuement sur le choix de celui-ci.

Git représente la plus grande part du marché¹³ pour une bonne raison, historiquement le logiciel effectue le travail parfaitement. Il existe plusieurs concurrents chacun représente une petite part du marché par rapport à git, les avantages mis en avant par chaque concurrent sont souvent liés aux autres produits qu'ils proposent et souvent dans

¹² OWASP, 2021. *OWASP Top Ten*

¹³ HECHT, Lawrence, 2019. « I Don't Git it: Tracking the source Collaboration Market »

une idée de garder l'ensemble des ressources utilisées par une entreprise auprès d'un seul fournisseur.

Dans notre cas, Git, remplit l'ensemble de nos critères de sélection. Afin de comprendre ce qui va constituer le trigger, nous allons nous attarder sur quelques actions que le logiciel permet d'effectuer.

Comme précédemment discuté, le but d'un pipeline est de s'assurer que notre projet, et donc son code, soit bien aux standards désirés avant de le rendre accessible à nos utilisateurs. Il ne fait donc pas sens de faire passer tous ces tests à notre projet uniquement lorsque nous amenons des modifications à celui-ci.


Le principe de fonctionnement de Git est relativement simple. Vous disposez de votre dossier de travail localement mais vous souhaitez pouvoir travailler sur votre projet avec d'autres personnes. Vous allez donc mettre votre code à leur disposition via ce que Git appelle un repository. Cela n'est rien d'autre qu'un dossier hébergé soit sur l'un de vos serveurs soit par la solution cloud proposée par Git, Github. Une fois votre dossier de projet disponible pour vos partenaires de travail, l'intérêt est que chacun puisse y contribuer. Vu que chacun travaille sur le même code, Git propose quelques actions et outils permettant à tout le monde de travailler ensemble facilement.

Nous allons créer un dépôt sur Github pour notre projet.

Figure 6 : Création d'un dépôt Git sur Github

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *  BloopT / Repository name *

✔ Demo is available.

Great repository names are short and memorable. Need inspiration? How about [symmetrical-telegram](#)?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.

Une fois cette action effectuée, nous allons nous placer dans le dossier contenant notre projet et y effectuer les actions suivantes en ligne de commande.

Figure 7: Initialisation de la connexion au dépôt distant

```
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/BloopT/Demo.git
git push -u origin main
```

Dans l'ordre : Nous initialisons d'abord notre dossier comme allait être lié avec un dépôt distant. Cela va créer une structure permettant à Git de reconnaître le dossier et d'y apporter les modifications souhaitées. Ensuite nous ajoutons un fichier (mais nous pourrions tous les ajouter), qui nous permettra de faire notre premier commit. Le commit enregistre les changements effectués dans votre répertoire uniquement sur les fichiers ajoutés et depuis le dernier commit. Ici il s'agit du premier, il n'y aura que notre fichier Readme. Il va ensuite créer la branche principale appelée main. S'ensuit le liage entre notre dépôt local et le distant. Enfin, nous poussons l'information du dépôt local au distant.

Dans une équipe, chacun va travailler sur sa version du code mais ne va pas la partager avec les autres avant d'être sûr d'avoir correctement effectué son travail. Il travaillera sur ce que l'on appelle une branche. Il s'agit du même principe que pour votre projet principal mais qui sera gardé à part. La branche contenant votre projet principal mis en production s'appellera par défaut master ou main selon les envies de Microsoft. Vous pouvez également disposer d'une branche à jour mais uniquement pour des tests. Une fois qu'un des contributeurs a fini sa tâche, il va vouloir rapatrier son travail sur le projet de test par exemple. Il va dans ce cas faire une action portant le nom de merge. Il va rapatrier son code sur celui de la branche souhaitée et ainsi fusionner les deux.

C'est ce qui suit qui va être important pour nous. Pour l'instant, notre contributeur travaille localement et veut désormais partager son code avec le reste de l'équipe, le mettre en production ou le tester avec le travail des autres contributeurs. Il va donc devoir envoyer son travail sur le répertoire distant partagé. C'est cette action qui sera notre trigger, en effet, avant de déployer notre projet avec ces nouveautés, nous allons nous assurer que tout ce qui fonctionnait le fait toujours. Il va donc passer à travers notre

pipeline. Cette action, appelée par Git « Push » (comme lors de l'initialisation), envoie le code sur le dépôt distant et va signaler que notre processus doit être lancé.

Il est important de préciser que Git ne fait pas que les quelques actions mentionnées mais bien plus et que les actions ont été simplifiées dans les explications, Git n'étant pas d'une grande importance dans notre projet. Nous nous limiterons aux concepts importants pour comprendre le fonctionnement de bout en bout pour la mise en production de notre projet.

3.5 Conteneurisation

La conteneurisation règle de nombreux problèmes liés aux différents environnements nécessaires à la conception et à la vie d'un programme. De nombreuses entreprises travaillent encore avec de la configuration manuelle sur leurs serveurs. Les machines virtuelles se sont bien implantées, les outils de configurations aussi mais la conteneurisation a facilité énormément de choses. Il s'agit d'un outil dont la prise en main n'est pas trop compliquée, comme nous le verrons avec Docker. Couplé avec d'autres outils, le gain en coût et en économie de temps est énorme. Rien que pour les environnements de développement, pour les projets jusqu'à une complexité moyenne, il n'est plus nécessaire d'y dédier un serveur, tout peut être monté en une commande sur la machine du développeur.

Comme énoncé précédemment, l'un des problèmes dont souffrait le cycle de développement logiciel était la différence de configuration pouvant exister entre les différents environnements à travers lesquels le logiciel passait lors de son déploiement. Une première tentative aura été d'utiliser des machines virtuelles pour régler ce problème¹⁴.

Une machine virtuelle (abrégé VM pour Virtual Machine) offre l'avantage d'être très portable en comparaison avec la configuration d'un serveur qui sera plus difficilement copiable et réutilisable. C'est pourquoi elle semblait être une bonne façon de fournir à tous les métiers un artefact pouvant être modifié et réutilisé pour chacune des étapes.

Même si une VM est un outil excellent et qui permettrait de régler une bonne partie des problèmes liés aux différences d'environnement, elle souffre encore de certains problèmes. Une machine virtuelle doit avoir des ressources précises liées à son hôte qui lui sont allouées pour sa consommation. A cause de son OS et ses dépendances elle

¹⁴ FARCIC, Viktor, 2016 *The DevOps 2.0 Toolkit, Automating the Continuous Deployment Pipeline with Containerized Microservice*

a déjà beaucoup un poids et une consommation non négligeable, ce qui-là rend volumineuse et longue à mettre en marche. Sans compter qu'une VM dédiée à un environnement de développement sera configurée pour gérer les besoins des développeurs mais ceux-ci voudront tester de nouvelles choses, ajouter des librairies et autres dépendances, ce qui, comme avec les serveurs de développement, créera sur le terme un monstre contenant plus que ce qu'il n'en faudrait. Il est tout à fait possible de faire attention et logger tout changements apporter afin de pouvoir les supprimer si besoin. Mais cela n'est que peu automatisable et très prompt aux erreurs humaines. Sans compter que dans un monde de micro-services, il est très important de pouvoir compter sur des environnements légers, au risque de voir ses coûts exploser et ses performances amoindries.

Un concept est venu régler ces soucis, la conteneurisation. Le principe est simple : il s'agit d'un outil permettant d'automatiser la création d'un environnement possédant le minimum pour faire fonctionner chaque application. En parlant de minimum, il s'agit bien sûr de tout ce dont le programme a besoin pour accomplir ses tâches, même si, comme nous allons le voir, il faudra comme pour une machine virtuelle un système opérateur comme base pour y installer le reste. L'avantage de cette manière de faire un paquet avec tout ce qui est nécessaire, est que celui-ci est autonome, il pourra donc être installé et fonctionné indépendamment de l'infrastructure ou de l'OS de son hôte. Il pourra fonctionner comme à l'intérieur d'une bulle et ne laissera rentrer et sortir que ce qu'on lui demande.

Cette bulle portera le nom de conteneur, le nom se rapporte à un conteneur de transport et convient à merveille car notre conteneur est fait pour être léger et transportable. Comme nous le verrons, ce concept convient parfaitement aux micro-services et à l'utilisation de ressources sur demande, car notre conteneur ne contient en général rien de persistant. Il peut être utilisé et ainsi créer son environnement, y faire tourner le programme lié et ensuite être détruit. Il est donc parfait pour créer rapidement de nouvelles instances de supports si un service a une demande accrue et, à l'inverse, les détruire si ce n'est plus le cas.

Figure 8 : La conteneurisation



Source : SHAAN, Ray, 2019. *What is Containerization?* Medium[en ligne].

[Consulté le 30 mars 2023]. Disponible à l'adresse :

<https://medium.com/hackernoon/what-is-containerization-83ae53a709a6>

Dans notre cas, il permet de fournir un environnement évolutif et léger à l'ensemble de notre chaîne de production. Il permettra donc à nos développeurs de modifier leur environnement comme bon leur semble tout en pouvant le passer à la production sans que cela alourdisse ou complexifie le projet.

La conteneurisation a permis de grandement réduire les ressources demandées pour faire fonctionner des projets, c'est aussi cela qui a accéléré la transition en direction du cloud. La majorité des providers de IaaS profite de la flexibilité qu'offre l'ensemble de la chaîne de déploiement pour mutualiser les ressources à des prix attractifs.

3.5.1 Docker

Git n'aura plus d'incidence pour le reste de notre projet contrairement à Docker qui a une importance capitale. Comme déjà mentionné dans la partie de ce travail sur la conteneurisation, il va nous permettre de s'assurer que notre environnement de développement sera le même que notre environnement de production ou du moins que notre environnement dans lequel nous testons le projet est le même que celui de production. Cela est crucial pour assurer que l'ensemble du projet soit fonctionnel. Pour cela les tests que nous ferons subir à notre projet dans notre pipeline seront déterminant.

Afin de construire notre environnement facilement partout où nous en aurons besoin, de faire passer ces tests à notre projet ou de le déployer, il est nécessaire d'avoir un outil afin de construire nos containers et de les configurer. Nous irons plus loin dans les détails de cet outil mais ce qui est important de comprendre à cette étape est que notre outil réalisant les tests devra utiliser un environnement dans lequel les effectuer.

Encore une fois, nous nous trouvons dans une situation où le moindre problème de configuration entre notre environnement de test et celui de développement peut rapidement engendrer de plus gros problèmes. La conteneurisation règle ce souci et va nous permettre de construire notre environnement et d'y faire tourner nos tests. Si nos développeurs travaillent dans un environnement également mis en place via un outil de conteneurisation, le moindre changement effectué se fera via le fichier de configuration de l'environnement conteneurisé. Le fichier pourra ainsi être passé avec le reste du projet afin d'être utilisé lors des prochaines étapes du pipeline.

Il existe plusieurs outils de conteneurisation, le plus connu est Docker. Ici, le choix est facile : il existe certes d'autres outils mais le plus complet et surtout le plus communément utilisé est Docker. Le second argument en faveur de ce choix est très important : tout nouvel outil ayant à travailler avec de la containerisation va chercher à être compatible avec le leader du marché. Docker domine le marché et a défini les standards de l'industrie.

Comment Docker fonctionne-t-il : le but étant de fournir un fichier de configuration qui contiendra l'ensemble des actions nécessaires à la création et à la configuration d'un environnement souhaité. Le but n'étant pas de configurer l'environnement direct mais bien de construire par-dessus, comme pour une machine virtuelle, il faut un élément permettant de communiquer avec le système opérateur de l'hôte. Dans le cas de Docker, il s'agit de Docker Engine. Docker Engine, à l'instar d'un hyperviseur pour une VM, va permettre d'utiliser les ressources de l'hôte et d'utiliser ses services pour, par exemple, communiquer avec l'extérieur ou rendre accessible les services installés dans le container.¹⁵ Pour que cela fonctionne dans l'environnement, il est important de préciser que Docker doit être installé sur son hôte, ce qui, lorsque l'on souhaite automatiser nos actions, demandera l'utilisation d'un outil de configuration de ressources sur lequel nous reviendrons plus tard.

Lorsque notre fichier de configuration sera prêt, celui-ci sera construit(build) et comme pour une VM, donnera naissance à une image. Une image n'est rien d'autre que la

¹⁵ DOCKER, 2023. *Docker Engine*

représentation d'un environnement sous forme d'un fichier ce fichier pouvant être ensuite monté afin de rendre l'environnement utilisable.

Le but de Docker n'est pas seulement de pouvoir construire ces environnements mais également d'avoir la possibilité de se baser sur des images existantes et d'y rajouter sa surcouche. Un peu comme un jeu de poupées russes, il est possible de rajouter plusieurs surcouches les unes sur les autres. Cela permet à une communauté de mettre à disposition des images contenant des configurations simples sur lesquelles il sera possible de construire des déclinaisons en fonction des besoins.

3.5.1.1 Dockerfile

Comment va se présenter notre fichier de configuration : celui-ci s'appelle Dockerfile voici un exemple utilisé dans notre projet.

Figure 9: Notre Dockerfile

```
FROM php:8.1-apache
WORKDIR /var/www/html

RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

RUN curl -O https://phar.phpunit.de/phpunit-10.0.19.phar

RUN chmod +x phpunit-10.0.19.phar && mv phpunit-10.0.19.phar /usr/local/bin/phpunit

RUN phpunit --version

RUN apt-get update && apt-get install -y \
    && docker-php-ext-install pdo pdo_mysql \
    && docker-php-ext-enable pdo pdo_mysql

COPY . /var/www/html/
EXPOSE 80
```

FROM va nous permettre d'aller récupérer une image existante et sur laquelle nous souhaitons commencer à construire. Cette image se trouve dans un dépôt similaire au principe proposé par Git. Le service en ligne contenant les images mises à disposition par la communauté s'appelle Docker Hub. Si nous venions à construire notre fichier, Docker téléchargerait l'image choisie et la monterait. Cela donnera une base sur laquelle la suite de la configuration sera effectuée. Ici notre projet sera écrit en PHP, nous nous basons donc sur l'image PHP de docker, avec un tag explicitant la version (8.1) ainsi que (apache) pour spécifier que nous voulons une image contenant un serveur apache.

WORKDIR va simplement spécifier l'emplacement dans lequel nous allons travailler. C'est donc dans ce dossier que les commandes qui suivent seront exécutées.

COPY va nous permettre de dupliquer le dossier du projet dans l'emplacement de notre environnement prévu pour les faire fonctionner. Dans notre projet, nous allons dupliquer

l'ensemble de notre dossier où se trouve notre Dockerfile (la racine du projet) dans l'emplacement de travail que nous avons défini. Le premier paramètre concerne la partie locale, ici « . » symbolise tout ce que le dossier contient, le second argument est la destination de copie dans notre conteneur.

RUN va permettre de lancer des commandes Shell par default, ou autre si précisé et ainsi va finir la configuration de l'environnement. Il est possible de lancer autant de commandes que souhaité. Dans notre exemple, nous allons lancer les commandes Shell permettant d'aller récupérer et installer les différences dépendances dont notre projet à besoin. En l'occurrence, le premier de la liste est le composer PHP, que nous allons récupérer puis installer, suivi de PHP Unit (pour nos tests) et enfin le code de l'interface permettant à PHP de dialoguer avec notre base de données MySQL.

CMD est unique à chaque fichier de configuration. Il est le point d'entrée du service que l'on souhaite faire fonctionner dans le conteneur, il permettra de lancer le service pour que celui-ci soit fonctionnel une fois le conteneur prêt.

EXPOSE indiquera quel port ouvrir sur notre conteneur, permettant ainsi l'accès au service. Comme nous l'avons discuté, le conteneur est initialement une boîte hermétiquement fermée, si nous souhaitons que notre service puisse être disponible à l'utilisation, il est essentiel d'ouvrir une porte d'accès. Dans notre projet, nous allons simplement ouvrir le port 80 qui est le port HTTP permettant ainsi l'accès direct depuis un navigateur.

Il est important de préciser que Docker va créer une image temporaire à chaque étape, appliquant ainsi les avantages de la conteneurisation à son propre procédé. L'ensemble des étapes de travail sur l'image s'effectue bien sûr dans un container créé pour l'occasion.

Il existe d'autres mots-clés d'instruction dans Docker que nous n'aborderons pas ici. Une fois notre image créée, nous allons la stocker sur un répertoire de la même manière que pour GIT. Le but de cette action est de pouvoir récupérer notre image pour l'utiliser partout y compris dans un projet contenant plusieurs services. Docker offre une bibliothèque en ligne s'appelant DockHub, elle fonctionne de manière similaire à GitHub.

Nous disposons maintenant d'une image disponible de la même manière que celle que nous avons utilisée pour notre base (php). Nous allons pouvoir l'appeler de la même manière comme nous le verrons dans le chapitre suivant.

3.5.1.2 Dockercompose

Les micro-services dominent aujourd'hui le monde de l'architecture logiciel. Une application ne sera que très rarement composée d'un seul service. Nous souhaitons que ces services soient découplés mais, suivant les besoins, il est souvent nécessaire qu'ils soient disponibles les uns pour les autres. L'idée est donc de combiner plusieurs Dockerfile en un seul fichier afin de n'avoir à monter que celui-ci et que les services soient configurés pour fonctionner ensemble.

Il est important de préciser que si nos services sont dans des conteneurs dispatchés et ainsi techniquement indépendants, cela n'en fait pas une architecture de micro-services. Si les services sont déployés indépendamment sur des clusters(groupes) d'instances différentes, il s'agira de micro-services. Dans le cas de multiple conteneur sur une instance, on simule du micro-service mais dans une architecture monolithique.

Un fichier Dockercompose va permettre de construire plusieurs conteneurs dans un conteneur mère. Une fois monté, il construira un conteneur qui contiendra les conteneurs de chaque service présent dans le fichier¹⁶.

¹⁶ DOCKER, 2023. *Docker Compose*

Voici le fichier Dockercompose de notre projet :

Figure 10: Notre Dockercompose

```
version: '1.0'
services:
  php:
    image: tmolleyres/apachewithcomposerandphpunit1.0:latest
    volumes:
      - ./my-website:/var/www/html
    environment:
      TZ: Europe/Zurich
      PUID: ${UID:-9999}
      PGID: ${GID:-9999}
    ports:
      - 80:80
    links:
      - db
    depends_on:
      - sonarqube
  db:
    image: mysql:latest
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: admin
      MYSQL_DATABASE: my_database
    ports:
      - 3306:3306
  sonarqube:
    image: sonarqube
    ports:
      - 9000:9000
    volumes:
      - sonarqube_data:/opt/sonarqube/data
  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080

volumes:
  sonarqube_data:
```

La version est annoncée en début de fichier et sert uniquement à garder un historique. C'est la partie suivante qui nous intéresse, celle nommée « services ». Nous nous retrouvons ici avec une application complète découpée en services, chacun découpé l'un de l'autre en termes d'environnement mais étant reliés et fonctionnant dans le même conteneur.

Ici, la syntaxe et les éléments sont proches de ceux d'un Dockerfile. Nous avons notre liste de services, chacun disposant d'un nom et étant lié à une image. Tout d'abord, le nom n'est pas présent que pour une question de clarté car, suivant le contexte, il fera office de lien d'accès au service. Par exemple, pour la connexion à la base de données depuis l'un des autres conteneurs.¹⁷ Ensuite, l'image, pour que notre Dockercompose puisse fonctionner, il doit pouvoir accéder aux images demandées, s'il s'agit d'images génériques de services, par exemple MySQL pour le service « db », il ne s'agira que de MySQL (docker hub a simplifié le nom des images les plus communément utilisées).

Dans le cas où le projet demanderait l'utilisation d'une image créée sur mesure, il est important que celle-ci soit hébergée sur un dépôt, afin que les différents serveurs puissent venir la récupérer pour créer les environnements lors des différentes étapes de notre pipeline. Une image hébergée sur Docker hub aura un nom suivant la structure : `votre_profil/nom_de_votre_image`. Enfin, un dernier élément important pour récupérer la bonne image est, le tag. Il s'agit du texte suivant le nom de l'image et précédé de « : ». Par défaut, celui-ci sera « latest », il s'agira de l'image la plus récente mise à disposition sur le dépôt de ce nom. Mais si nous souhaitons faire évoluer notre environnement, la dernière image pourrait être créée pour des tests par un développeur et ne devrait donc pas être utilisée à des fins de production, dans ce cas pour éviter tout problème on utilisera un tag spécifique.

Le fichier Dockercompose de notre projet contient quatre services, le premier et notre serveur web pour y faire tourner notre projet PHP. Il reprend notre image Dockerfile que nous avons hébergée sur notre DockerHub. Afin de la récupérer nous utilisons la structure expliquée au paragraphe précédent. Dans ce service, nous y fixons des variables d'environnement afin d'assurer son bon fonctionnement, exposons le port 80 du conteneur et l'exposons sur le port 80 de la machine locale, et enfin, nous l'attachons aux autres services. Pour les 3 autres services, il s'agit de du même principe.

Nous avons 4 services chacun dans son conteneur mais dialoguant entre eux si souhaité. MySql sera notre service de base de données, adminer le gestionnaire lié et enfin Sonarqube est installé afin de tester le projet.

L'un des problèmes de la conteneurisation est la persistance des données. Comme nous l'avons discuté, les environnements sont construits puis détruits au gré des besoins, mais que faire si nous avons des données qui sont persistantes et qui doivent être conservées même une fois l'environnement détruit ? Lorsqu'il s'agit de données

¹⁷ DOCKER, 2023. *Docker Compose*

centralisées qui n'ont rien à faire dans des containers, le service est gardé à part sur des instances fixes et dupliquées pour assurer la stabilité du bien précieux qu'est la donnée. Dans une plus petite mesure, Docker a créé un service annexe aux conteneurs, permettant de sauvegarder les données. Ce service porte le nom de volume. Il n'est pas dans le conteneur, ce qui lui permet de continuer à exister même lorsque le conteneur est détruit et, ce qui permet aux données d'être toujours disponibles sur la machine même si un nouvel environnement venait à remplacer l'ancien, du moment que la référence sur le volume est la même. Dans notre projet, cela est purement à des buts fonctionnels, aucune donnée n'a besoin d'être persistée.

Avec Docker, nous allons pouvoir créer un environnement portable disposant de tous nos services qui va accompagner notre projet tout au long de son cycle de vie et lui permettre d'être testé, de fonctionner ou encore d'être mis à l'échelle en fonction du besoin sans presque plus jamais avoir à se soucier qu'il dispose de toutes les dépendances et configurations dont il aurait besoin.

3.6 Résumé

Nous avons posé les bases des technologies nécessaires à la construction de notre pipeline. Nous avons discuté l'importance de la sécurité ainsi que les vulnérabilités les plus courantes pour des logiciels et comment nous allons pouvoir les identifier dans notre pipeline. Toutes les pièces du puzzle ne sont pas nécessaires, cependant, afin de pouvoir arriver à un résultat répondant aux attentes, le plus d'outils seront intégrés, meilleur sera le résultat.

4. L'outil au cœur du cycle d'automatisation

Dans l'optique d'offrir un outil flexible et centralisé, le pipeline n'a pas besoin d'être complet, ni d'être extrêmement restrictif. Pour une petite structure, il peut n'inclure que quelques tests vitaux dans les deux cycles mais il est important que cet outil permette d'assurer la délivrance d'un produit sécurisé et un environnement à jour fonctionnel.

Il est clair qu'un pipeline sera différent d'un projet à un autre. Ce qui peut vite devenir une quantité de travail conséquente si le nombre de projets est élevé. C'est pourquoi il est préférable de rester sur une architecture aussi proche de l'agnostique que possible. Certes, certaines parties sont propres aux langages utilisés, en utilisant la conteneurisation, l'environnement étant packagé avec le projet, une grande partie du pipeline pourra ainsi être réutilisé pour d'autres projets.

Il serait même intéressant d'aborder l'idée d'un pipeline qui puisse être utilisable peu importe le langage utilisé pour le projet. Cela n'est pas aussi compliqué qu'il puisse le paraître mais cela demande de se reposer sur une grande quantité de scripts Bash ou autre suivant l'OS. Il n'en est pas moins que cette possibilité est attirante surtout dans une idée de micro-services, qui pour une liberté de conception et dans le but d'être aussi performants que possible, seront dans un grand nombre de cas développés dans des langages différents. Nous reviendrons un peu plus en détail sur cette possibilité au fur et à mesure que nous avançons dans la description du pipeline.

4.1 Critères de choix de solution

Afin d'avoir une structure de pipeline qui soit portable sur n'importe quelle architecture ou hébergeur, il est préférable d'éviter des solutions développées par les revendeurs de solution d'hébergement. N'étant pas une société disposant de moyens conséquents, nous nous concentrerons sur des solutions open sources. Elles ont l'avantage d'être gratuites et d'être supportées par une communauté, ce qui nous offrira également une base de support gratuite.

Le pipeline que nous allons essayer de mettre en place ne sera sûrement pas le plus complet, mais nous essaierons de garder sa complexité de mise en place et d'utilisation à un degré raisonnable pour éviter de se retrouver avec un monstre dont nous ne maîtrisons pas la complexité.

Nous nous assurerons également d'utiliser des solutions qui ont prouvé qu'elles allaient survivre dans le temps, qui sont reconnues dans et par la communauté du monde informatique et qui sont communément utilisées.

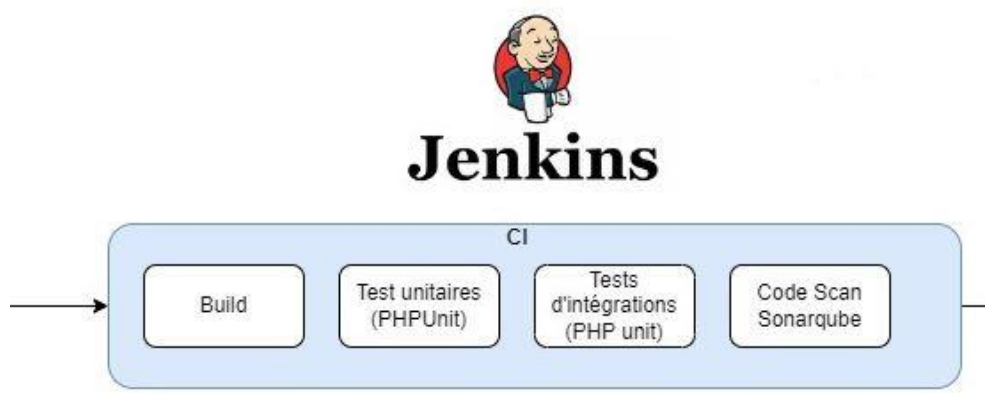
Afin d'éviter de tomber dans la complexité de services distribués, nous allons rester sur un déploiement limité avec une possibilité de répondre à l'augmentation de la demande mais dans une limite. La mise en place du dit pipeline demande déjà de toucher un grand nombre de technologies et y ajouter une capacité à pouvoir supporter une augmentation de charge conséquente requiert encore une autre expertise et risque de générer plus de problèmes qu'en régler dans notre cas.

Dans les chapitres à venir, nous allons décortiquer chaque étape de notre pipeline et faire un choix technologique pour régler le problème si nécessaire.

L'idée finale est d'avoir un pipeline automatisable disposant de toutes les étapes nécessaires à amener un projet jusqu'à son déploiement.

4.2 Continuous Integration

Figure 11 : Le cycle d'intégration continu



La première étape consiste à déclencher notre pipeline, c'est une action qui signalera le besoin de déclencher les étapes, le déclencheur (en anglais Trigger) vient très souvent d'une action réalisée sur le dépôt contenant le code de notre projet. En parlant de dépôt, il s'agit bien sûr d'une base de données liée à un logiciel de contrôle de version. Dans notre cas, un répertoire Git hébergé sur GitHub

Selon les avis, le trigger ne fait pas toujours parti du pipeline, je suis d'avis que le trigger fait partie de la solution complète et, par conséquent, doit être abordé.

Une fois notre trigger déclenché, notre projet va être de le récupérer sur son répertoire. Avant de discuter de qu'est ce qui va le récupérer et ce qu'il va en faire, nous allons devoir nous attarder sur ce que notre projet doit contenir afin de pouvoir être mis en

production. Comme discuté nous pourrions juste le packager et remplacer le projet existant sur notre serveur de production. Mais nous avons bien compris que ce n'est pas la meilleure façon de fonctionner et c'est dans cette optique que nous avons abordé la conteneurisation.

4.2.1 Les étapes décortiquées

Nous allons nous arrêter ici sur les étapes de notre pipeline et les outils qui seront nécessaires à son bon fonctionnement. Comme nous l'avons discuté, si l'une des étapes venait à ne pas se dérouler de manière satisfaisante, le processus n'ira pas plus loin et le développeur ayant envoyé son nouveau code vers le dépôt distant sera averti du souci et pourra travailler sur la résolution du problème sans que ces changements n'impactent les différents environnements de tests ou de production¹⁸.

4.2.1.1 Build

La première étape peut sembler triviale mais n'est pas moins essentielle particulièrement suivant les langages utilisés pour construire le projet.

Nous avons ici 2 parties, vu que nous avons fait le choix de conteneuriser notre environnement, il est important de voir si celui-ci va pouvoir se construire correctement. Si ce n'est pas le cas, inutile d'essayer d'aller plus loin, rien ne marchera correctement.

La seconde partie de cette étape dépend des langages utilisés. En effet, afin de pouvoir être utilisés par une machine certains langages doivent être compilés et ainsi transformés en fichier exécutables. Cette étape est cruciale pour tous les langages demandant à être compilés avant de pouvoir être exécutés, il s'agit du test le plus simple permettant de savoir si le code contient des erreurs pouvant empêcher son exécution.

4.2.1.2 Les Tests

Une fois que notre environnement, et si besoin notre programme, est monté et prêt à fonctionner, il s'agit de s'assurer que tout fonctionne correctement et que les standards du produit proposé soient bons.

Première étape, les tests unitaires, ils sont la base de la vérification du bon fonctionnement d'un programme. Un test va s'occuper de valider le bon fonctionnement d'un morceau de code. D'une manière assez pragmatique, si l'on veut tester une fonction, il nous faudra écrire plusieurs tests s'assurant que la fonction remplit bien ce qu'on attend d'elle suivant les informations qu'on lui fournit où qui existent déjà dans le

¹⁸ IBM, 2020. *What is Continuous Integration?*

projet. Et à l'inverse, qu'elle n'altère pas certaines données. Le but est simple, s'assurer qu'en cas de modification de la fonction ou d'un autre morceau de code auquel celle-ci ferait appel, les résultats attendus restent les mêmes.

Viennent ensuite les tests d'intégrations, en effet, il est rare que notre application fonctionne en autarcie sans autres services, qu'il s'agisse de l'utilisation d'une base de données, de l'appel à une API externe ou encore l'appel d'une fonction appartenant à un autre service, notre application ne fonctionnera pas correctement si ces besoins ne sont pas également fonctionnels. Il est donc nécessaire de réaliser des tests utilisant ces services, soit en appelant directement le service mais il est plus sage d'effectuer ces tests directement sur les parties du code de l'application y faisant appel.¹⁹

Ces deux premiers jeux de tests sont souvent réalisés par les développeurs, ils sont adjacents au code et ne requièrent pas d'utilisation d'outils externes particuliers. Chaque langage dispose de son propre outil de test, celui-ci est souvent en extension et installé dans l'environnement. Ils permettent de mettre en place une structure pouvant être run de la même manière que du code et d'obtenir un retour indiquant quels tests sont réussis ou non.

Un dernier test effectué directement sur le code, s'appelle la revue du code. Un logiciel externe va se charger de scanner le code à la recherche de code déprécié ou pouvant générer des problèmes de sécurité. Ici se trouve un grand nombre d'outils disponibles permettant de scanner le code et de donner un retour. Ce type de test est particulièrement utile sur des systèmes complexes, il est difficile de tester l'entièreté du code, sans compter qu'il est également difficile d'anticiper tous les scénarii possibles. Les outils vont s'appuyer sur les tests réalisés précédemment, ils ne sont en aucun cas une manière de remplacer les tests unitaires et d'intégrations. Il s'agit ici de tests dits statiques car le code n'est en aucun cas exécuté.

Les outils vont généralement tester et proposer des améliorations concernant :

La refactorisation du code, qu'il s'agisse de code dupliqué voire même partiellement dupliqué, en proposant une ou plusieurs fonctions couvrant l'entier du besoin réduisant ainsi la complexité du projet. Sur le même principe, lorsqu'une partie du code peut être jugée trop compliquée et donc difficile à maintenir, l'outil peut proposer une alternative.

¹⁹ KIM, Gene, HUMBLE, Jez, DEBOIS, Patrick et WILLIS John, 2021. *The DevOps Handbook How to Create World-Class Agility, Reliability, and Security in Technology Organizations*

Par exemple dans le cas de patterns utilisés pouvant générer des problèmes ou de fonctions pouvant être poussées à l'erreur grâce au passage d'informations problématiques en paramètres.

Et enfin, dans un souci d'assurer la sécurité de l'application, l'outil testera le code et l'ensemble de l'application pour toutes CWE ou CVE. Qu'il s'agisse de s'assurer qu'il ne soit pas possible de réaliser d'injection SQL, d'attaques du type cross-site Scripting ou encore de Buffer Overflow. Cela permettra de garder l'application à jour contre un bon nombre de vulnérabilités connues pouvant être exploitées à cause d'un mauvais code ou d'une mauvaise configuration. Il s'agit ici de la partie test de sécurité pré-déploiement, une autre partie se fait post-déploiement, l'application n'étant pas du tout dans le même état, tout ne peut pas être testé.

4.2.1.2.1 Comparatif des outils existants (non exhaustif) :

Le choix d'un outil d'inspection de code va dépendre de chaque projet. Tous les outils ne gèrent pas forcément le langage que vous utiliserez dans votre projet. Il est également important de prendre en compte les outils avec lesquels vous allez devoir intégrer ce nouvel outil. De la même manière, il peut être intéressant d'avoir un seul outil pour le test dynamique de votre code ainsi que pour les tests statiques. Le dernier paramètre à prendre en compte correspond à votre hébergement de l'outil, si vous travailler dans un environnement fermé, utiliser un outil uniquement disponible en version cloud peut demander de la configuration supplémentaire et des risques de problèmes dans l'intégration. A l'inverse, il peut être judicieux de se démettre du maintien d'un serveur dédié à un outil supplémentaire.

Voici une liste de 3 outils ayant chacun leurs particularités :

Les 3 outils vont offrir :

- Une analyse du code avec une vision de la couverture des tests effectués.
- Une recherche d'optimisation de code via la détection de code dupliqué ou de fonctions mal optimisées.
- La détection de vulnérabilités ou de problèmes pouvant amener à des vulnérabilités.
- Plus d'une 20aines de langages supportés.
- Sont intégrables avec la majorité des cloud provider, gestionnaires de dépôts de code et outils de gestion de pipeline CI/CD.

Codacy propose de personnaliser les règles de vérification de qualité du code. En effet comme tous les outils de reporting, la quantité d'informations peut être très élevée. Suivant le niveau de standards émis par la société, il se peut que toute l'information

retournée ne soit pas utile. En effet, certains points sont cruciaux, notamment au niveau des performances et de la sécurité, mais d'autres peuvent être plus secondaires et l'entreprise peut les juger trop chronophage. En choisissant les règles, vous amenez vos équipes à se focaliser sur les types de problèmes que vous jugez les plus importants. Disponibles en version cloud ou on-premise.

Sonarqube fait partie de la suite Sonar, visant à couvrir tous les besoins en analyse de code, il s'agit du membre de la liste disposant de la couverture du plus grand nombre de langages ainsi que de la compatibilité avec le plus d'outils. Il s'agit également d'un des outils les plus utilisés dans sa catégorie²⁰, il dispose d'une version gratuite déjà très qualitative et est disponible en version cloud ou on-premise.

Veracode convient autant à l'analyse statique que dynamique du code avec un accent mis sur la sécurité. Il propose une option de scan liée à l'analyse de la composition du logiciel, ce qui permet de refactoriser des morceaux jugés complexes en plus petits morceaux moins complexes, simplifiant ainsi la maintenance du logiciel. Il est disponible uniquement en version cloud²¹.

Certains des avantages listés sont également présents dans les autres logiciels listés mais de manière moins poussée ou en tout cas moins mis en avant par le concepteur.

4.2.2 Résumé Continuous Intégration

Nous avons maintenant fait le tour de l'ensemble des points à mettre en place dans la partie d'intégration continue de notre pipeline. Avec ces étapes et leur outils associés, les développeurs de l'application devraient pouvoir recevoir toutes les informations nécessaires à la réalisation du code pour un logiciel respectant les attentes et standards demandés. Cette partie du pipeline ne couvre certes pas tous les tests mais peut être utilisée très fréquemment par les développeurs. Il est important de noter que suivant le langage et le projet, une grande partie des tests mentionnés peuvent être réalisés par le développeur dans son environnement de développement, voire même avec l'aide de son IDE directement. Il n'empêche que lorsqu'une nouvelle version est prête, il n'en est pas moins une obligation de s'assurer que rien n'a été oublié et d'avoir une trace du passage des tests et leur retour en cas de problème.

L'approche se basant sur une grande quantité de tests pour assurer une qualité élevée au produit obtenu a fait son chemin dans le monde du développement, au point de créer

²⁰ APPCIRCLE, 2022. *How to integrate Sonarqube into your CI/CD Workflow*

²¹ Codacy. *Wikipédia : l'encyclopédie libre*

le « Test Driven Développement » consistant à écrire les tests avant le code et ainsi de s'assurer d'emblée que le code effectuera bien ce qui est attendu de lui. Cette façon de faire est bonne mais extrêmement chronophage, il est clair qu'elle sera évitée par de nombreuses petites entreprises. Cependant, les tests ne sont pas la partie à oublier, quelques tests unitaires sur des fonctions clés ainsi que des tests fonctionnels offrent déjà une bonne base, sans oublier le scanner de code. A nouveau, le but n'est pas d'astreindre mais de donner toutes les informations et éviter les catastrophes.

4.3 CI/CD TOOLS

Afin de faciliter la réalisation de ces tests, d'avoir un suivi sur leur retour et d'automatiser toute la chaîne du pipeline, il faut un outil permettant d'orchestrer la chaîne d'événement et d'arrêter le processus lorsque cela est nécessaire.

Les outils de CI/CD sont le cœur du pipeline, ils vont permettre d'automatiser toutes les étapes énumérées jusqu'à présent et de déployer le projet dans ces différents environnements que sont l'environnement de test/QA, de Staging et enfin celui de développement.

Ils sont suffisamment souples pour permettre d'effectuer différents types de tests et de déployer dans différents environnements en fonction de la source souhaité²². Typiquement, un pipeline peut être prévu en fonction d'une branche d'un dépôt de code pour tester en profondeur certains aspects mais ne demande pas l'entièreté du pipeline lié à la production économisant ainsi du temps et des ressources.

Ces outils sont très souvent associés au terme DevOps car ils sont la jonction entre les équipes de développement et de production. Il s'agit des outils assurant que l'attente des deux parties soit réalisée et permet ainsi de s'assurer que le logiciel passé dans la chaîne ne manquera pas à ses obligations.

Pour récapituler, nous attendons de notre outil de CI/CD, qu'il soit synchronisé avec notre dépôt de code et réagisse lorsqu'une nouvelle version est déposée sur celui-ci. Une fois qu'il reçoit le signal, l'outil va récupérer notre projet sur le dépôt. Il va se charger de construire l'environnement grâce à Docker, si besoin il construira également notre programme et le mettra en fonction dans l'environnement qu'il a construit. Une fois le programme prêt, il exécutera les tests prévus unitaires, fonctionnels et de qualité de

²² KIM, Gene, HUMBLE, Jez, DEBOIS, Patrick et WILLIS John, 2021. *The DevOps Handbook How to Create World-Class Agility, Reliability, and Security in Technology Organizations*

code afin de s'assurer que tout est aux normes attendues. Si c'est le cas, il se chargera de déployer notre projet dans les environnements souhaités.

4.3.1 OUTILS

Nous allons nous intéresser à 3 outils de CI/CD parmi les plus courants sur le marché. Le monde technologique tournant autour du DevOps étant devenu très à la mode pour de bonne raisons, il existe un très grand nombre d'outils. La majorité des plus grands hébergeurs dispose de leurs propres outils de CI/CD, voire même de l'ensemble des outils nécessaires à la réalisation de l'ensemble des étapes du pipeline. Ces outils sont souvent très bons, notamment ceux proposés par Amazon Web Services et permettent la mise en place d'un pipeline rapidement et efficacement avec peu de configurations dites système. Cependant, je pense, qu'il est important de rester agnostique et de se baser sur des outils open-source ne liant pas la configuration à un hébergeur. Le vent tourne vite dans le milieu technologique on ne sait jamais trop de quoi demain sera fait et ne pas pouvoir se permettre de migrer facilement son infrastructure peut s'avérer coûteux.

La majorité des outils se basent sur un fichier de configuration importable avec le projet ou une création basée sur GUI permettant de configurer les étapes.

Travis CI offre des workflows permettant de construire rapidement un pipeline. Son avantage principal tourne autour de son environnement en GUI permettant de configurer facilement les fonctionnalités. Le fichier de configuration utilise YAML pour la description (même type de format que Docker). Il propose une matrice de tests pour différents environnements. Il dispose d'une intégration avec la majorité des cloud providers. Il rencontre cependant des limites en termes de customisation et sur de gros projets. C'est une bonne option pour commencer. Une caractéristique cependant, il n'est disponible qu'en version cloud.

GitLab était d'abord une extension à GitHub pour gérer des dépôts de codes. Il a évolué en un outil CI/CD disposant de nombreuses fonctionnalités adjacentes liées aux travaux collaboratifs sur des projets de développement. Qu'il s'agisse de gestion de projet ou de documentation, GitLab offre de nombreuses options en dehors de son usage CI/CD. GitLab ne se n'est pas arrêté là, il intègre un scanneur de code, ainsi qu'un outil de monitoring. Il permet ainsi l'intégration de nombreux outils ainsi que tous les grands cloud providers. Son fichier de configuration est également en YAML. Il est disponible en version cloud et On-premise. Comme Travis CI, il peut rencontrer certaines limitations

en termes de compatibilité avec d'autres outils. Il est important de noter que GitLab devient rapidement payant en dehors d'une utilisation pour petits projets.

Jenkins est actuellement le leader du marché, il s'agit d'un projet open-source disposant d'un très grand nombre de plug-ins développés par la communauté. Cela lui donne l'avantage de se rendre compatible avec énormément d'outils et la quasi-totalité des langages. Il permet un logging et reporting extensif des informations retournées par le pipeline. Jenkins, ne propose pas un pipeline de base quasi fonctionnel de la même manière que pourrait le faire GitLab. La mise en place d'un pipeline est cependant facilitée par son énorme librairie de plug-ins. La communauté maintient très proprement les plug-ins et offre un support gratuit. Jenkins est disponible en version On-premise ce qui demandera à leurs utilisateurs d'avoir une instance configurée correctement pour pouvoir l'utiliser.

Circle CI est compatible avec un peu moins de langages que ses contreparties proposées. Son fichier de configuration est également au format YAML. Son avantage se trouve sur ses mises à jour relativement fréquentes et son support intégrée. Il est le choix d'entreprises plus grandes, souhaitant un GUI plus poussé et un support rapide. Il propose également la possibilité d'exécuter en parallèle différents conteneurs et ainsi environnements ce qui n'est pas le cas de Jenkins, mais également le cas de GitLab. Circle CI propose différents prix dont une version gratuite, mais le prix évolue très rapidement en cas de besoin de support ou d'un peu plus de puissance de calcul pour éviter un temps d'exécution trop long.

Pour notre petit projet, nous allons utiliser Jenkins installé sur une instance AWS. Nous y avons installé les différents plug-ins liés aux outils que nous allons utiliser et rentrer les informations nécessaires dans chaque plug-in. Nous allons ensuite démarrer un projet de type pipeline que nous allons lier à notre répertoire Git et choisir de lancer le pipeline en cas d'action de type push sur le répertoire de la branche main.

Notre installation étant uniquement sur une instance peu puissante, le déroulement du pipeline peut prendre un temps conséquent. Il est possible de distribuer les tâches sur plusieurs instances et ainsi d'augmenter la vitesse de réalisation. Nous testerons uniquement sur un serveur avec une base Linux, mais il est également possible de distribuer le même pipeline sur différents serveurs avec différents OS si cela est nécessaire.

Afin de définir les étapes de notre pipeline, nous allons utiliser une jenkinsfile qui sera intégrée dans le projet. La voici :

Figure 12 : Notre jenkinsfile

```
pipeline {
  agent any

  stages {
    stage('Checkout') {
      steps {
        checkout([$class: 'GitSCM', branches: [[name: '*/master']],
          userRemoteConfigs: [[url: 'https://github.com/BloopT/TBPhpMySQL.git']]])
      }
    }

    stage('Build Docker Images') {
      steps {
        sh 'docker-compose build'
      }
    }

    stage('Start Docker Compose Environment') {
      steps {
        sh 'docker-compose up -d'
        sleep 60
      }
    }

    stage('Run PHP Unit Tests') {
      steps {
        sh 'docker-compose run --rm run --rm php vendor/bin/phpunit tests/ConnexionBddTest.php'
      }
    }

    stage('Run SonarQube Analysis') {
      steps {
        withSonarQubeEnv(installationName: 'sonarqube'){
          sh './mvn clean org.sonarsource.scanner.maven:sonar-maven-pugin:39.0.2155:sonar'
        }
      }
    }

    stage('Stop Docker Compose Environment') {
      steps {
        sh 'docker-compose down'
      }
    }
  }
}
```

Les étapes définissant notre pipeline vont fonctionner comme suit :

Nous allons d'abord récupérer le contenu de notre répertoire git, puis monter notre docker-compose, ensuite mettre en place notre environnement avec le projet à l'intérieur. Nous laissons un peu de temps au système pour monter le docker-compose. Ensuite nous lançons les tests directement dans le conteneur. L'étape suivante par le même procédé va lancer une analyse avec sonarqube sur notre projet depuis un serveur externe. Enfin, une fois les tests passés, nous pouvons stopper notre conteneur.

4.4 Continous Deployment

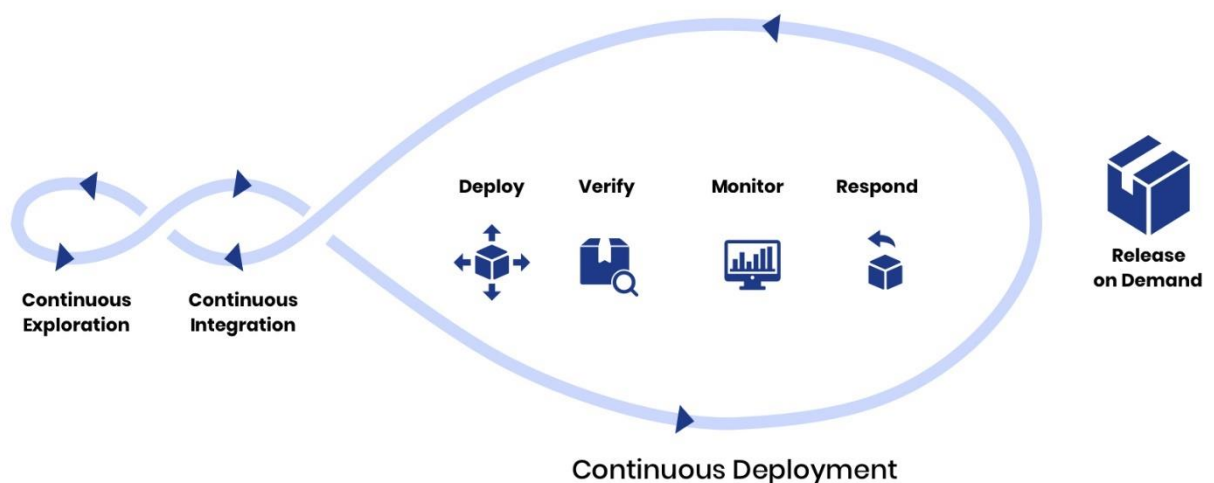
Comme il l'a été mentionné, l'acronyme CD peut représenter deux termes. Le premier est continuous delivery, le second, continous deployment. Le second étant l'extension du premier. Continuous delivery représente la possibilité de déployer une nouvelle version

d'un logiciel à n'importe quel moment²³. Cela permet aux développeurs de déployer sur l'environnement souhaité leur travail à chaque nouvelle fonctionnalité, aussi petite soit-elle et sans que cela n'impacte l'utilisateur. Cependant, certaines validations restaient liées à une intervention humaine, la plus classique étant la mise en production. Continuous deployment pousse le principe un peu plus loin, en automatisant l'ensemble de la chaîne de déploiement²⁴. Les tests automatisés sont jugés suffisamment complets pour assurer que, dans le cas où le logiciel en passe la totalité, il peut directement être déployé dans l'environnement souhaité, voire même en production.

Le continuous deployment est l'extension du delivery dans le sens où si delivery est suffisamment bien fait, il se transforme en continuous deployment.

Nous avons énoncé les étapes nécessaires à un bon cycle d'intégration continu, où le but était de fournir toutes les informations nécessaires aux développeurs pour garantir que leur travail respecte les attentes de l'entreprise. Ici, il s'agira de donner toutes les informations aux équipes opérationnelles pour que celles-ci s'assurent que leur architecture et ses systèmes soit pérenne et sécurisée.

Figure 13 : Le cycle de déploiement continu



Source : KAUTURI, Srividya et VITUCCI, Carmen. *An Introduction to Continuous Deployment*. 2022.

²³ KAUTURI, Srividya et VITUCCI Carmen, 2022. « An Introduction to Continuous Deployment »

²⁴ Source : KAUTURI, Srividya et VITUCCI, Carmen. *An Introduction to Continuous Deployment*. 2022.

Le déploiement est première la partie du processus, le but étant de mettre en fonctionnement notre application dans un environnement donné. Il existe plusieurs types d'environnement, cela dépend des besoins de l'entreprise et de ses équipes concernant les phases de validation des modifications apportées.

L'environnement de Développement, l'avantage de la virtualisation et par extension de la conteneurisation, est de pouvoir offrir à ses développeurs un environnement configuré de la même manière que celui dans lequel il sera déployé. Cependant, cela peut avoir des limites, suivant la complexité du projet : le nombre de micro-services, des projets trop volumineux ou encore des ressources disponibles uniquement sur un Cloud privé virtuel. C'est pourquoi il peut être nécessaire d'offrir un environnement de travail à ses développeurs.

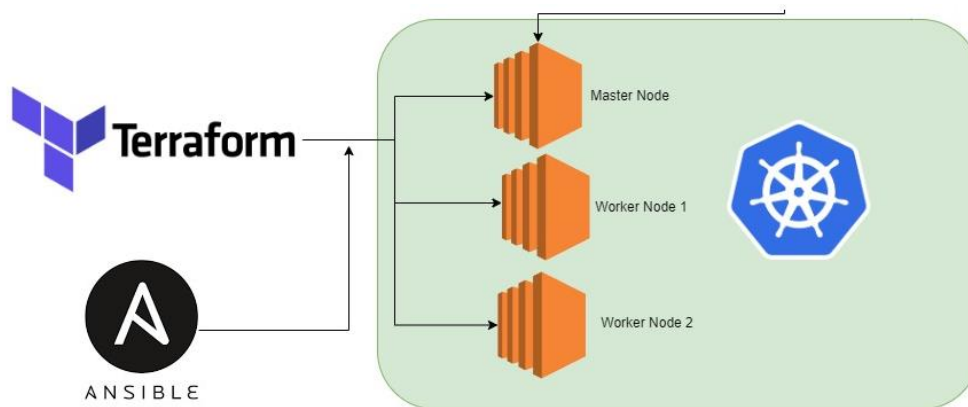
L'environnement QA ou quality assurance est purement un environnement de test qui ne se veut proche de la production. Même si un grand nombre de tests peuvent être automatisés, il n'en est pas moins que s'il s'agit de fonctionnalité complète, il est possible de tester le code mais plus difficilement l'utilisation humaine. C'est pourquoi cet environnement est crucial pour les personnes travaillant à valider les modifications. Suivant les tailles des projets et entreprises, il se peut que l'environnement de Staging et QA ne soit qu'un.

L'environnement de Staging : l'utilité de cet environnement est hybride, très souvent c'est l'environnement mis à disposition du client afin que celui-ci puisse utiliser les nouvelles fonctionnalités et donner un feedback à l'équipe de développement. Il est crucial dans une méthodologie agile. Il sert également d'environnement dans lequel il est possible de voir les effets des modifications apportées sur une partie d'un logiciel et vérifier son intégration hors code. Il s'agit de l'environnement de plus proche de la production.

L'environnement de production est celui que l'utilisateur final aura à disposition. Ici, pas d'erreur possible : si les modifications arrivent jusque-là, c'est que le projet est jugé stable et ses fonctionnalités fiables. Il n'en est pas moins possible que des bugs persistent, il est difficile d'envisager toutes les possibilités d'utilisation, de problèmes entre les composants ou de tentatives d'intrusion sur un logiciel. C'est pourquoi encore aujourd'hui, des bugs et crashes arrivent sur des logiciels mis en production, même parmi les plus grandes sociétés du monde informatique.

4.4.1 Partie du mini-projet

Figure 14 : Zoom sur l'architecture du projet



L'idée pour notre projet fictif était d'offrir un environnement modulable et réutilisable. Les outils proposés permettent de mettre en place automatique une architecture pouvant supporter un projet. L'idée est de déployer un conteneur packager directement dans le nœud maître d'un cluster Kubernetes et ensuite le laisser le déployer sur les nœuds travailleurs.

Le projet n'ira pas jusque-là, nous nous contenterons de générer automatique une instance sur AWS et d'y exécuter les commandes en SSH afin de déployer notre projet. Le reste des outils sera expliqué afin de suivre le fil conducteur du projet fictif.

4.4.2 Terraform pour provisionnement de ressources et Ansible pour le management de configuration

Nous avons énoncé les outils de provisionnement de ressources un peu plus tôt. Ceux-ci servent à mettre en place les ressources nécessaires au fonctionnement d'un service. Ils peuvent être utilisés de nombreuses manières différentes, mais nous allons nous intéresser à leur utilisation pour provisionner des ressources sur les plateformes de revendeurs d'IAAS.

L'idée est simple, vous avez besoin d'un de vos outils de tests ou n'importe quel service. Plutôt que de devoir créer une instance chez votre fournisseur et ensuite vous y connecter et la configurer, ce type de logiciel va vous permettre de créer ses instances automatiquement, y installer tout ce dont vous avez besoin, au point d'avoir tous les services fonctionnels, et que vous n'ayez plus qu'à les utiliser. Une fois fini avec votre tâche, il ne vous restera plus qu'à lancer une autre commande et détruire ces instances. Cela va permettre d'économiser en temps de travail ainsi qu'en coûts liés à l'utilisation

de ressources. Cela peut être utilisé même pour les environnements de Staging et QA, voire même de développement si un projet se trouve à maturité et cet environnement ne sera remonté que en cas de besoin d'évolution ou de correction de bug.

L'un des autres problèmes liés aux coûts des ressources est celui d'être sûr d'en avoir assez pour satisfaire tous les utilisateurs. Pour la partie de déploiement de notre pipeline proposé, nous avons besoin de plusieurs outils. Un outil pour provisionner nos environnements, ainsi qu'un outil pour assurer sa stabilité et, si besoin, sa scalabilité (mise à l'échelle) en cas d'augmentation ou diminution de l'utilisation de notre logiciel. Nous allons utiliser Terraform pour le provisionnement et Kubernetes abrégé K8s pour assurer le reste de nos besoins. Un dernier outil sera nécessaire pour configurer proprement les environnements que nous allons provisionner et gérer, il s'agira d'un outil de management de configuration nommé Ansible.

4.4.2.1 Terraform

Terraform est l'un des outils de provisionnement open source les plus connus. Il permet plusieurs actions très intéressantes pour notre utilisation et un grand nombre que nous n'aborderons pas.

Tout d'abord, vu que nous allons provisionner des ressources sur un IAAS provider (dans notre cas AWS), il est important de savoir que Terraform peut être utiliser depuis n'importe quelle machine du moment qu'elle a accès à internet. Il peut être intéressant de l'installer sur la même machine que votre outil CI/CD afin de déclencher un script de provisionnement en fonction du pipeline qui aurait été lancé.²⁵ Typiquement, pour créer les environnements souhaités. Une autre option est de l'avoir sur une machine mutualisée comme une VM interne à votre entreprise, et de laisser les développeurs les exécuter en fonction des besoins.

Comme pour beaucoup des outils proposés, Terraform va utiliser un fichier de configuration sur lequel il va se baser pour effectuer ses tâches. Ses scripts ont la particularité d'être écrit de manière déclarative, ce qui revient à annoncer ce que l'on souhaite avoir à la fin du processus, à l'inverse du reste des outils utilisés qui ont une forme impérative qui, elle, annonce ce qui va être fait sous forme de commandes²⁶. De la même manière que Docker, Terraform va se baser sur le fichier de configuration présent à l'emplacement où est exécutée la commande de lancement. Ce qui permet de simplement se déplacer au bon endroit pour lancer la configuration, dans cette idée, il

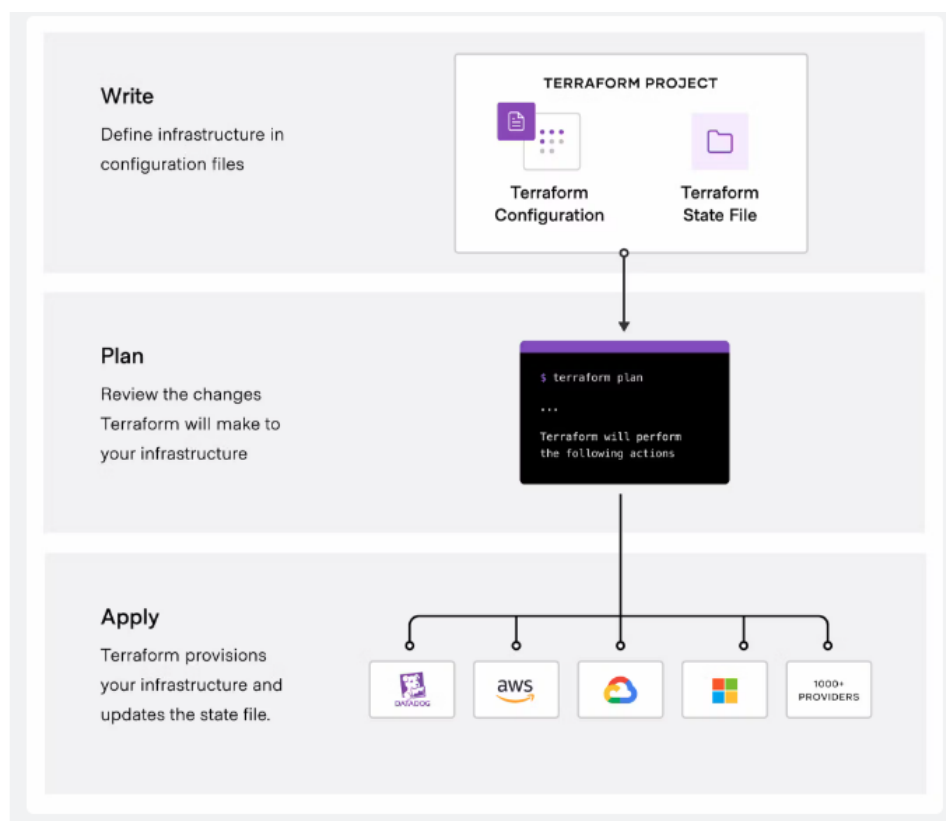
²⁵ Source: HASHICORP, 2022. What is Terraform?

²⁶ Source: HASHICORP, 2022. What is Terraform?

suffit de préparer l'ensemble du dossier voulu pour chaque déploiement et de simplement modifier le chemin du script de déplacement dans le script pour l'exécuter.

Terraform va proposer plusieurs actions utiles, la première est le « plan » qui va permettre au logiciel de préparer son plan d'exécution pour arriver à l'état (la configuration) qu'on lui a demandé. L'avantage avec ce plan, c'est que si vous avez déjà vos ressources en cours d'exécution et que vous avez modifié votre script, celui-ci va simplement regarder ce qu'il doit modifier. Cela permet par exemple de rajouter une instance pour un nouveau service sans perturber le reste. Pour que le programme aille récupérer l'état actuel, la commande « refresh » est utilisée. Enfin pour appliquer le plan qui a été prévu, il faut utiliser « apply », à savoir que apply exécutera automatiquement les 2 commandes précédentes, donc pas besoin de le faire manuellement ou dans un script. Une fois que vous n'avez plus besoin de votre ressource, vous pouvez utiliser « destroy ».

Figure 15 : Le fonctionnement de Terraform imagé



Source: HASHICORP, 2022. What is Terraform?

Ce type d'outils permet une réelle économie sur des ressources aux besoins temporaires. En automatisant correctement l'exécution, vous pouvez vous démettre des

ressources ou environnements inutiles même pendant certaines tranches horaires, typiquement la nuit, période durant laquelle vos équipes ne travaillent pas.

Notre script Terraform :

Figure 16 : Notre fichier Terraform

```
provider "aws" {
  region = "eu-central-1"
}

resource "aws_key_pair" "my-key-pair" {
  key_name     = "my-key-pair"
  public_key   = file("${path.module}/ansible/ssh/my-key.pub")
}

resource "aws_security_group" "my-security-group" {
  name_prefix = "my-security-group"

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port = 80
    to_port   = 80
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "my-instance" {
  ami           = "ami-0c94855ba95c71c99"
  instance_type = "t2.micro"
  key_name      = aws_key_pair.my-key-pair.key_name
  security_groups = [aws_security_group.my-security-group.name]

  connection {
    type     = "ssh"
    user     = "ubuntu"
    private_key = file("${path.module}/ansible/ssh/my-key.pem")
    host     = self.public_ip
  }

  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update",
      "sudo apt-get install -y python3-pip",
      "pip3 install ansible boto boto3"
    ]
  }

  provisioner "local-exec" {
    command = "sleep 30"
  }
}
```

Nous allons rapidement passer en revue ce qui se passe dans ce script. Tout d'abord, nous annonçons le provider : AWS. Il est possible de provisionner différentes ressources chez différents providers dans le même plan. Nous précisons la région d'AWS où doit être créée notre instance.

Commence ensuite le provisionnement de ressource, la première est une clé côté AWS qui devra matcher celle que nous avons préalablement importée.

Ensuite, nous créons un groupe de sécurité qui sera appliqué à notre instance. Nous lui donnons un nom et les droits d'entrée sur les ports 22 pour SSH et 80 pour http. Nous autorisons les entrées depuis n'importe quelle IP pour les 2 ports.

Cette configuration n'est pas très sécurisée, mais pour notre projet démo cela ne pose pas de problème.

Durant l'étape suivante, nous créons notre instance de type t2.micro (une taille liée à une quantité de ressources). Nous choisissons comme image, Linux Ubuntu précisée grâce à l'ami. Le tout autorisé grâce à notre clé liée au compte ayant la permission de créer des instances sur AWS.

Enfin, nous nous connectons à l'instance en SSH et y installons Python et le module Ansible nécessaire à l'interaction avec AWS. C'est d'ailleurs ici qu'un problème se pose avec la partie Ansible de ce projet, un souci de configuration reste dans mon projet. C'est pourquoi la partie Ansible du projet ne sera pas présentée dans ce travail.

De nombreuses améliorations peuvent être ajoutées dans ce script notamment avec le stockage de l'IP public de l'instance créée afin de pouvoir la réutiliser dans le script Ansible sans devoir la copier manuellement dans le fichier de configuration.

4.4.2.2 ANSIBLE

Ansible est un outil d'automatisation de tâches typiquement de configuration. Il permet également comme Terraform de provisionner des ressources tout comme Terraform permet d'automatiser certaines tâches, mais il ne s'agit pas de leur point fort²⁷.

Ansible va donc permettre d'automatiser des tâches comme la configuration d'un environnement, l'installation d'une mise à jour ou un reboot. Il s'agit ici toujours de tâches manuelles répétitives qui peuvent générer des problèmes si elles ne sont pas faites correctement.

²⁷ GEERLING, Jeff, 2020. « How useful is Ansible in a Cloud-Native Kubernetes Environment? »

Ansible a le grand avantage de ne pas demander d'installation du côté client, ce qui veut dire qu'il n'y a pas besoin de se connecter à une machine après son provisionnement, y installer un client et ensuite lancer la configuration. Un simple accès SSH suffit pour permettre à Ansible d'effectuer son travail sur la machine. Comme Terraform, il peut être utilisé depuis une machine en-dehors de l'environnement cloud.

Ici aussi nous avons un fichier de configuration qui va être utilisé pour réaliser les tâches sur la machine cliente. Ces fichiers s'appellent des Playbooks, terme venant du milieu sportif pour désigner le livre contenant toutes les stratégies sur le terrain pour une équipe.

Même si nous n'avons pas besoin de configurer notre machine nouvellement provisionnée avec Terraform car tout ce dont nous avons besoin se trouve dans le container préparé, il faut tout de même installer docker ou K8s et la manière la plus propre reste d'utiliser Ansible.

4.4.2.3 KUBERNETES

Nous avons plusieurs fois abordé dans ce travail le concept de micro-services et son rôle dans l'architecture moderne de logiciels. La conteneurisation a réglé le problème de la création d'environnement léger et portables, mais quand le nombre de services est grands et que ceux-ci doivent pouvoir réagir à la demande, il devient difficile de gérer les besoins manuellement.

Kubernetes est un outil dit d'orchestration de conteneurs. L'idée est de gérer tous les conteneurs de services dans le but de les maintenir opérationnel, de s'assurer que la charge de demande est bien répartie entre les différentes instances du même service et enfin augmenter ou diminuer le nombre d'instances de chaque service en fonction du besoin²⁸.

L'outil est compliqué à prendre en main, il est cependant très puissant. Il est utilisé uniquement pour des environnements de production, il ne serait d'ailleurs pas d'une grande utilité dans un environnement de test.

Afin d'assurer les fonctionnalités que nous avons énoncées, K8s utilise différentes entités. Tout d'abord, les nœuds (nods), chaque nœud représente une machine. Chaque groupe de nœud est géré par un nœud maître. Celui-ci disposera d'une API et sera responsable de la gestion des autres nœuds de son groupe appelé « slaves » (mais nous utiliserons travailleur).

²⁸ IBM, 2019. *What is Kubernetes*

Les kubelets : sur chaque nœud un kubelet est opérationnel, sa tâche est de monitorer et gérer ce qui se passe dans les nœuds travailleurs, plus précisément tout ce qui touche aux pods fonctionnant dans leur nœud.

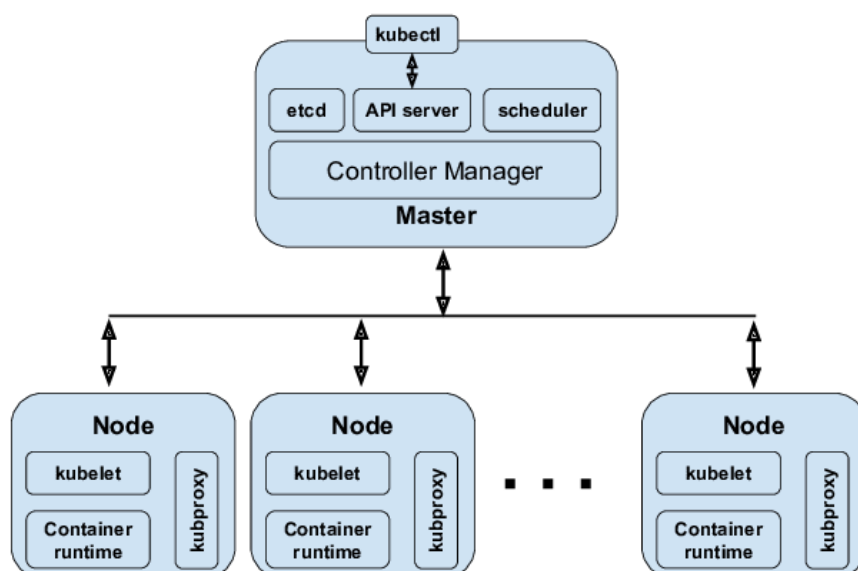
Les pods : ce sont des groupes d'un ou plusieurs conteneurs qui partagent le même environnement et sont gérés (orchestrés) ensemble. Chaque pod a une adresse IP permettant d'y accéder.

L'utilisateur fournit un template de fonctionnement écrit en yaml au nœud maître et laisse celui-ci dispatcher la création ou modification des pods dans chacun de ses nœuds.

Une fois les pods déployées, Kubernetes va permettre via un de ses services de mettre à disposition l'ensemble des pods, du même type, déployé sur les différents nœuds comme un seul et même service. Permettant ainsi de faciliter la répartition de la charge entre chaque pod.

Pendant de nombreuses années, lors de la création de ressources ou leurs modifications, traquer les changements sur l'ensemble des ressources était une tâche laissée aux outils de service « registry » et « discovery ». Ceux-ci avaient la tâche de garder à jour une liste de l'ensemble des machines et services disponibles afin de pouvoir y accéder et les utiliser. Ils devaient cependant être configurés sur chaque machine. Kubernetes gère cela nativement. Si un pod venait à être détruit et

Figure 17 : Kubernetes Architecture



SAHA, Pankaj, BELTRE, Angel, YOUNG, Andres, et GRANT, Ryan, 2019.
« Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container
Orchestration Mechanisms »

reconstruit, car il souffrait d'un problème, son adresse IP sera automatiquement stockée et mise à disposition.

Kubernetes dispose de nombreux autres services, mais leur explication en détail n'est pas nécessaire. Mais il est important de comprendre que tout ne se fait pas tout seul et doit être configuré. Mais une fois fait, le système va se maintenir et réagir au changement selon les règles que nous lui auront éditées²⁹.

Dans notre projet, l'idée était d'avoir une instance master et 2 instances de travail. Notre projet aurait ainsi été de manière identique sur chacun des nœuds assurant son bon fonctionnement, la répartition de la charge ainsi que sa scalabilité en cas de besoin.

4.4.3 SCANNER DE VULNERABILITE

Les scanners de codes décrits dans les chapitres précédents font les tests effectués directement sur la logique d'une application. Le but étant de prévenir toutes erreurs ou mauvaises utilisations pouvant amener à une vulnérabilité. Le scanner de vulnérabilité va travailler sur votre infrastructure et votre logiciel mais maintenant que celui-ci est fonctionnel et que le code est en utilisation, il ne va plus s'agir de prévenir les vulnérabilités, mais de lister toutes les existantes !

Ce n'est pas parce que vous aurez patchez toutes les recommandations du scanner de code qu'il n'y aura pas de vulnérabilités. Ici, le système dans son ensemble est testé, qu'il s'agisse de la configuration du serveur, ou des outils et dépendances y fonctionnant tout est lié.

L'outil ne demande pas une grande configuration, la première étape consiste à être sûr que l'ensemble des ressources de votre infrastructure soit trouvé par le logiciel. Il se chargera ensuite de relever les programmes et dépendances installées ainsi que la configuration de chaque machine, permettant de relever l'ensemble des vulnérabilités ou risques découverts. Enfin, un rapport sera retourné, permettant aux équipes d'effectuer leur travail. En quelques clics, ou automatisé au sein d'un pipeline, un scanner de vulnérabilité permet autant au management de se rendre compte des soucis dont pourrait souffrir leur installation qu'à leur équipe ne pouvant ainsi plus ignorer les vulnérabilités existantes. La sécurité semble malheureusement tomber dans les problèmes qui sont ignorés tant qu'ils ne sont pas révélés et accompagnés d'avertissements³⁰.

²⁹ BEN, 2019. « How Kubernetes Can Improve your CI/CD Pipeline ».

³⁰ ZAP, 2023. *Getting Started*

Les scanners de vulnérabilités sont un outil vraiment important dans la lutte pour sécuriser une entreprise. Certes, une grande partie sont payants et peuvent atteindre des prix élevés, même en payant à l'utilisation. Cependant, il en existe des gratuits, même s'ils ne fournissent pas le même travail, ils permettent déjà de corriger une partie importante des vulnérabilités, surtout pour des petites structures. Ils n'ont pas forcément besoin d'être intégrés à un pipeline et ainsi exécutés à chaque déploiement. Mais une utilisation régulière est importante, de nouvelles vulnérabilités sont découvertes chaque jour et mieux vaut ne pas laisser les portes ouvertes trop longtemps, surtout depuis que les attaques de type ransomware se multiplient et même sur les PME.

OWASP ZAP est un scanner de vulnérabilité gratuit développé et maintenu par la communauté d'OWASP. Il est orienté pour les applications web et sera donc moins précis sur la partie infrastructure³¹. Il a le mérite d'être simple d'utilisation et de déjà donner un grand nombre d'informations. Il peut être installé localement sur une machine ou intégré dans un pipeline. Il n'offre malheureusement pas de version cloud mais cela peut être une bonne chose, vous permettant de le faire fonctionner depuis l'intérieur de votre infrastructure, ainsi que l'extérieur. Il n'est également pas le plus simple à intégrer en fonction de l'outil CI/CD choisi. Il dispose même d'une version plug-in pour navigateur web permettant de tester rapidement un logiciel via son url d'accès.

Avec ce genre d'outils, il devient difficile de se cacher derrière l'excuse de la complexité. Vu les failles présentes aujourd'hui dans une PME moyenne, même un simple scan avec un outil de ce type ferait une grande différence.

4.4.4 Continuous Monitoring

Dans l'esprit de proposer un cycle de vie complet aux déploiements d'un logiciel, il reste une étape qui n'a pas été abordée, sa maintenance. Une fois un logiciel testé et correctement déployé, il n'en reste pas moins que des bugs ou problèmes de tous types peuvent subvenir. Il existe aujourd'hui un type d'outil tombant dans la catégorie du continuous monitoring (surveillance continue). Ce type d'outil a plusieurs utilités, et sa valeur sera complète uniquement si le reste de l'ensemble du cycle a été correctement réalisé³². Si cela n'est pas le cas, cet outil peut être utilisé simplement comme surveillant, annonçant lorsqu'un service ne fonctionne plus correctement, permettant ainsi de déclencher une procédure de relance ou de le faire manuellement. Pour une structure plus mature, l'outil de CM devient quelque peu le poste de contrôle de l'ensemble des services.

³¹ ZAP, 2023. *Getting Started*

³² SHARIF, Arfan, 2022. « What is Continuous Monitoring »

Il permettra de déceler et anticiper un certain nombre de problèmes, comme bloquer l'IP de la source d'une attaque en force brute sur un service, provisionner plus de ressources en cas de saturation ou d'espace de stockage arrivant à capacité.

Ici le plus simple reste de synchroniser manuellement l'outil avec les ressources que vous souhaitez monitorer et mettre en place des filtres sur le type et le niveau d'information que vous souhaitez recevoir. Ce type d'outils vous permettra de créer un Dashboard vous donnant ainsi une vue d'ensemble en temps réel de ce qui est en train de se passer sur vos différents produits.

5. Conclusion

Nous avons fait le tour de notre double boucle, ce qui nous a permis de traiter tous les éléments importants pour ce travail. Les outils dits DevOps offrent une réelle valeur ajoutée lorsqu'ils sont utilisés. Ils n'ont pas besoin d'être tous utilisés, comme nous l'avons discuté, même l'intégration de quelques outils, peut avoir un impact significatif.

Les petites entreprises ont tout à gagner en utilisant ces outils car elles sont souvent débordées par les avancées technologiques et leur adoption. La mise en place de l'utilisation automatique des outils DevOps dans un pipeline offre l'avantage de forcer les standards et de maintenir à jour les standards de l'industrie dans l'écosystème de l'entreprise. Cela se traduira par des produits beaucoup plus qualitatifs une fois finis.

La sécurité est en train de devenir une préoccupation majeure même pour de petites structures, les scanners de code, le scanner de vulnérabilité et les outils de monitoring permettront de fermer de nombreuses portes jusqu'à présent laissées ouvertes. Bien que cela ne remplace pas un audit, ces outils offrent la possibilité d'améliorer la sécurité de manière significative.

Je suis convaincu que l'utilisation d'outils DevOps peut contribuer à réduire les coûts liés aux ressources informatiques dans une entreprise. Bien que les salaires restent la première charge, l'infrastructure est également une charge importante. L'automatisation des déploiements et l'adaptation des ressources en fonction du besoin, permettront une économie de coût ainsi qu'une réduction des besoins. Dans une époque où la consommation d'énergie est une préoccupation majeure, mutualisé l'ensemble des ressources et n'utiliser que ce qui est nécessaire devient une priorité.

J'espère que ce travail permettra à certains de s'informer et pour d'autres de générer des idées. L'objectif était d'éviter de tomber dans une rédaction technique trop poussée afin de rendre les concepts accessibles à tous. Les outils discutés permettent bien plus que ce qui a été présenté. Je suis enthousiaste à l'idée de suivre l'évolution du monde DevOps dans les années à venir, d'autant plus que l'adoption du cloud accélérera également l'adoption des outils DevOps. Je pense que le monde informatique continuera à se transformer à un rythme effréné vers des niveaux de performance toujours plus élevé mais souvent toujours plus complexe, ce qui nécessitera toujours plus de personnes prêtes à travailler à son bon fonctionnement.

Bibliographie

ANKIT, Kumar, 2022. « *Staging Environment vs QA: What are the Differences* ». *Techpluto* [en ligne]. 6 décembre 2022 [Consulté le 26.04.2023]. Disponible à l'adresse : <https://www.techpluto.com/staging-environment-vs-qa-what-the-difference/>

APPCIRCLE, 2022. *How to integrate Sonarqube into your CI/CD Workflow* [en ligne]. [Consulté le 24.03.2023]. Disponible à l'adresse : <https://blog.appcircle.io/article/how-to-integrate-sonarqube-into-your-ci-cd-workflow>

APPSECENGINEER, 2021. DAST with Jenkins: Scan & Reporting with OWASP ZAP [enregistrement vidéo], *YouTube* [en ligne]. 18 août 2021. [Consulté le 23 avril 2023]. Disponible à l'adresse : https://www.youtube.com/watch?v=kFFYweAz-s8&list=LL&index=3&ab_channel=AppSecEngineer

ATKINSON, Brandon et EDWARDS, Dallas, 2018. *Generic Pipelines Using Docker The DevOps Guide to Building Reusable, Platform Agnostic CI-CD Frameworks*. New York: Apress Media LLC. ISBN 978484236543

AWS, 2023. *Continuous delivery: creating a staging environment* [en ligne]. [Consulté le 20.03.2023]. Disponible à l'adresse : <https://docs.aws.amazon.com/whitepapers/latest/practicing-continuous-integration-continuous-delivery/continuous-delivery-creating-a-staging-environment.html>

AWS, 2023. *Microservices* [en ligne]. [Consulté le 10.03.2023]. Disponible à l'adresse : <https://aws.amazon.com/fr/microservices/>

AZURE, 2022. *Qu'est-ce que DevOps?* [en ligne]. [Consulté le 04.03.2023]. Disponible à l'adresse : <https://azure.microsoft.com/fr-fr/resources/cloud-computing-dictionary/what-is-devops>

BEN, 2019. « How Kubernetes Can Improve your CI/CD Pipeline ». *Stackify* [en ligne]. 1er décembre 2019 [Consulté le 21.04.2023]. Disponible à l'adresse : <https://stackify.com/kubernetes-improve-ci-cd-pipeline/>

CODACY. *Wikipédia : l'encyclopédie libre* [en ligne]. Dernière modification de la page le 25 mars 2023 à 16:08. [Consulté le 17 avril 2023]. Disponible à l'adresse : <https://en.wikipedia.org/wiki/Codebase>

CLOUDBEESTV, 2021. How to Integrate SonarQube With Jenkins [enregistrement vidéo], *YouTube* [en ligne]. 24 août 2021. [Consulté le 20 avril 2023]. Disponible à l'adresse : https://www.youtube.com/watch?v=KsTMy0920go&ab_channel=CloudBeesTV

CLOUDFLARE, 2022. *Qu'est-ce que le cloud* [en ligne]. [Consulté le 10.03.2023]. Disponible à l'adresse : <https://www.cloudflare.com/fr-fr/learning/cloud/what-is-the-cloud/>

CVE, 2023. *CVE Numbering Authority CAN Rules* [en ligne]. [Consulté le 15.03.2023]. Disponible à l'adresse : <https://www.cve.org/ResourcesSupport/AllResources/CNARules>

DEVOPS JOURNEY, 2022. Learn Jenkins! Complete Jenkins Course – Zero to Hero [enregistrement vidéo], *YouTube* [en ligne]. 23 juin 2022. [Consulté le 1 avril 2023]. Disponible à l'adresse :

https://www.youtube.com/watch?v=l5k1ai_GBDE&t=1s&ab_channel=TechWorldwithNa
[na](#)

D, Anastasia et H, Dmytro, 2019. « Best Architecture for an MVP ». *Rubygarage* [en ligne]. 18 avril 2019 [Consulté le 01.03.2023]. Disponible à l'adresse : <https://rubygarage.org/blog/monolith-soa-microservices-serverless>

DOCKER, 2023. *Docker Desktop* [en ligne]. [Consulté le 22.03.2023]. Disponible à l'adresse : <https://docs.docker.com/desktop/>

DOCKER, 2023. *Docker Engine* [en ligne]. [Consulté le 22.03.2023]. Disponible à l'adresse : <https://docs.docker.com/engine/>

DOCKER, 2023. *Docker Compose* [en ligne]. [Consulté le 22.03.2023]. Disponible à l'adresse : <https://docs.docker.com/compose/>

DOMINGUS, Justin et ARUNDEL, John, 2019. *Cloud Native DevOps with Kubernetes*. New York: O'Reilly Media Inc. 356 p. ISBN 9781098116781

EDUREKA!, 2019. DevOps Tutorial for Beginners [enregistrement vidéo], *YouTube* [en ligne]. 30 juin 2019. [Consulté le 20 mars 2023]. Disponible à l'adresse : https://www.youtube.com/watch?v=hQcFE0RD0cQ&t=12071s&ab_channel=edureka%21

FARCIC, Viktor, 2016 *The DevOps 2.0 Toolkit, Automating the Continuous Deployment Pipeline with Containerized Microservice*. Birmingham: Packt Publishing. 462p ISBN 9781785289194

GEERLING, Jeff, 2020. « How useful is Ansible in a Cloud-Native Kubernetes Environment? ». *Red Hat Ansible* [en ligne]. 14 janvier 2020 [Consulté le 21.04.2023]. Disponible à l'adresse : <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith>

GOOGLE CLOUD, 2023. *Qu'est-ce que Kubernetes ?* [en ligne]. [Consulté le 20.04.2023]. Disponible à l'adresse : <https://cloud.google.com/learn/what-is-kubernetes?hl=fr#:~:text=Kubernetes%20automates%20operational%20tasks%20of,it%20easier%20to%20manage%20applications>.

HARRIS, Chandler, 2022. « Microservices vs. monolithic architecture ». *Atlassian* [en ligne]. 2022 [Consulté le 11.03.2023]. Disponible à l'adresse : <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith#:~:text=A%20monolithic%20architecture%20is%20a,monolith%20architecture%20for%20software%20design>.

HECHT, Lawrence, 2019. « I Don't Git it: Tracking the source Collaboration Market ». *The NewStack* [en ligne]. 2019 [Consulté le 21.03.2023]. Disponible à l'adresse : <https://thenewstack.io/i-dont-git-it-tracking-the-source-collaboration-market/>

HASHICORP, 2019. La bonne voie vers DevOps avec Terraform et Ansible [enregistrement vidéo], *YouTube* [en ligne]. 21 mai 2021. [Consulté le 2 avril 2023]. Disponible à l'adresse : https://www.youtube.com/watch?v=AsPIKWF1y_M&ab_channel=HashiCorp

HASHICORP, 2022. *What is Terraform?* [en ligne]. [Consulté le 06.04.2023]. Disponible à l'adresse : <https://developer.hashicorp.com/terraform/intro>

HASHICORP, 2022. *Build Infrastructure* [en ligne]. [Consulté le 12.04.2023]. Disponible à l'adresse : <https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-build>

HAMILTON, Thomas, 2023. « Agile vs Waterfall-Difference Between Methodologies ». *GURU99* [en ligne]. 11 mars 2023 [Consulté le 15.03.2023]. Disponible à l'adresse : <https://www.guru99.com/waterfall-vs-agile.html>

HAUSMANN, Roger, 2022. « Les PME ne prennent pas suffisamment au sérieux la cybersécurité ». *Swisscom b2b mag* [en ligne]. 6 juillet 2022 [Consulté le 17.03.2023]. Disponible à l'adresse : <https://www.swisscom.ch/fr/b2bmag/securite/pme-cybersecurite-protection/>

IBM, 2023. *What is continuous deployment ?* [en ligne]. [Consulté le 22.03.2023]. Disponible à l'adresse : <https://www.ibm.com/topics/continuous-deployment#:~:text=Continuous%20deployment%20is%20a%20strategy,directly%20to%20the%20software's%20users.>

IBM, 2019. *What is Kubernetes ?* [en ligne]. [Consulté le 17.04.2023]. Disponible à l'adresse : <https://www.ibm.com/topics/kubernetes#:~:text=Kubernetes%20is%20an%20open%20source,IBM%20Newsletter%20Explore%20Kubernetes%20optimization>

IBM, 2020. *What is Continuous Integration?* [en ligne]. [Consulté le 17.04.2023]. Disponible à l'adresse : <https://www.ibm.com/topics/continuous-integration>

IBM TECHNOLOGIES, 2021. Ansible vs Terraform : quelle est la différence ? [enregistrement vidéo], *YouTube* [en ligne]. 28 mai 2021. [Consulté le 2 avril 2023]. Disponible à l'adresse : https://www.youtube.com/watch?v=rx4Uh3jv1cA&ab_channel=IBMTechology

IONUT, Ilascu, 2022. « Google paid \$12 million in bug bounties to security researchers » *BleepingComputer* [en ligne]. 22 février 2022 [Consulté le 22.03.2023]. Disponible à l'adresse : <https://www.bleepingcomputer.com/news/security/google-paid-12-million-in-bug-bounties-to-security-researchers/>

JENKINS, 2023. *Pipeline Steps Reference* [en ligne]. [Consulté le 07.03.2023]. Disponible à l'adresse : <https://www.jenkins.io/doc/pipeline/steps/>

KARUTURI, Srividya et VITUCCI Carmen, 2022. « An Introduction to Continuous Deployment ». *Qentelli* [en ligne]. 2022 [Consulté le 22.03.2023]. Disponible à l'adresse : <https://www.qentelli.com/thought-leadership/insights/continuous-deployment#pipeline-of-cd>

KIM, Gene, HUMBLE, Jez, DEBOIS, Patrick et WILLIS John, 2021. *The DevOps Handbook How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Second Edition. Portland, OR: IT Revolution Press LLC. 480p. ISBN 978195058402

OWASP, 2022. *OWASP Top Ten* [en ligne]. [Consulté le 17.03.2023]. Disponible à l'adresse : <https://owasp.org/www-project-top-ten/>

OWASP, 2022. *Who is the OWASP Foundation?* [en ligne]. [Consulté le 17.03.2023]. Disponible à l'adresse : <https://owasp.org>

PONTHIEU, Theo, 2020. « *Le DevOps expliqué à mes parents (et à nos collègues non-initiés)* » *Ippon Positive Technology* [en ligne]. 29 juillet 2020 [Consulté le 04.03.2023]. Disponible à l'adresse : <https://blog.ippon.fr/2020/07/29/le-devops-explique-a-mes-parents/>

PITALIYA, Sarrah, 2023. « *Waterfall vs Agile: What to Know Before Choosing Your Development Method* » *Radix* [en ligne]. 27 mars 2023 [Consulté le 06.04.2023]. Disponible à l'adresse : <https://radixweb.com/blog/waterfall-vs-agile>

RED HAT, 2023. *What is a CVE?* [en ligne]. [Consulté le 15.03.2023]. Disponible à l'adresse : <https://www.redhat.com/en/topics/security/what-is-cve>

RED HAT, 2023. *What is a container registry?* [en ligne]. [Consulté le 01.04.2023]. Disponible à l'adresse : <https://www.redhat.com/en/topics/cloud-native-apps/what-is-a-container-registry>

RED HAT, 2023. *What is a CI/CD pipeline?* [en ligne]. [Consulté le 02.03.2023]. Disponible à l'adresse : <https://www.redhat.com/fr/topics/devops/what-cicd-pipeline>

SARAV, Ak, 2023. « *Terraform AWS Example-Create EC2 instance with Terraform* ». *DevOpsJunction.com* [en ligne]. 6 janvier 2023 [Consulté le 13.04.2023]. Disponible à l'adresse : <https://www.middlewareinventory.com/blog/terraform-aws-example-ec2/>

SAHA, Pankaj, BELTRE, Angel, YOUNG, Andres, et GRANT, Ryan, 2019. « *Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms* ». *ResearchGate* [en ligne]. novembre 2019 [Consulté le 20.04.2023]. Disponible à l'adresse : https://www.researchgate.net/publication/336889240_Enabling_HPC_Workloads_on_Cloud_Infrastructure_Using_Kubernetes_Container_Orchestration_Mechanisms

SHARIF, Arfan, 2022. « *What is Continuous Monitoring* ». *CrowdStrike* [en ligne]. 21 décembre 2022 [Consulté le 20.04.2023]. Disponible à l'adresse : <https://www.crowdstrike.com/cybersecurity-101/observability/continuous-monitoring/#:~:text=Continuous%20monitoring%20is%20an%20approach,time%20to%20address%20them%20quickly>.

SHARMA, Sanjeev, 2017. *The DevOps Adoption Playbook. A Guide to Adopting DevOps in a Multi-Speed IT Enterprise*. Illustrated. New York: John Wiley & Sons. 416 p. ISBN 978119308744

SONAR, 2023. *Clean code for teams and enterprises with {SonarQube}* [en ligne]. [Consulté le 18.04.2023]. Disponible à l'adresse : <https://www.sonarsource.com/products/sonarqube/>

STACKSHARE, 2023. *Codacy vs sonarqube* [en ligne]. [Consulté le 15.04.2023]. Disponible à l'adresse : <https://stackshare.io/stackups/codacy-vs-sonarqube>

STOREY, Dave, 2021. « *Terraform for Beginners* ». *ITNEXT* [en ligne]. 1er septembre 2023 [Consulté le 28.04.2023]. Disponible à l'adresse : <https://itnext.io/terraform-for-beginners-dd8701c1ebdd>

TECHWORLD WITH NANA, 2020. *Terraform explained in 15 mins* [enregistrement vidéo], *YouTube* [en ligne]. 4 juillet 2020. [Consulté le 30 mars 2023]. Disponible à l'adresse :

https://www.youtube.com/watch?v=l5k1ai_GBDE&t=1s&ab_channel=TechWorldwithNana

VAKUL, Gotra, 2023. « *Staging Environment vs Test Environment: Differences You Should Know* ». *TestSigma* [en ligne]. 27 avril 2023 [Consulté le 28.04.2023]. Disponible à l'adresse : <https://testsigma.com/blog/staging-environment-vs-test-environment/>

VEHENT, Julien, 2018. *Securing Devops- Safe Services in the Cloud*. New York: MANNING. ISBN 9781617294136

VERACODE, 2023. *For Developers & Security* [en ligne]. [Consulté le 19.04.2023]. Disponible à l'adresse : <https://www.veracode.com/why-veracode/for-outcomes>

ZAP, 2023. *Getting Started* [en ligne]. [Consulté le 18.03.2023]. Disponible à l'adresse : <https://www.zaproxy.org/getting-started/>

ZAP, 2021. *ZAPping the OWASP Top 10(2021)* [en ligne]. [Consulté le 18.03.2023]. Disponible à l'adresse : <https://www.zaproxy.org/docs/guides/zapping-the-top-10-2021/>

Annexe 1: GitHub Repository

<https://github.com/BloopT/TBPhpMySql>