< Titre du  
Travail de Bachelor >

< Logo de l'entreprise si désiré >

< S'IL Y A LIEU, ajouter la mention suivante >

< TRAVAIL STRICTEMENT CONFIDENTIEL >

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.urkund.com](https://www.urkund.com/fr/student/) .

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< e > le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à < indiquer le lieu >, le < date >

< Votre prénom et votre nom >

< Signez la déclaration ici >

Remerciements

Si vous avez des remerciements à formuler, à l’entreprise ou à toute autre personne qui a pu vous aider dans la réalisation du travail.

Les remerciements sont rédigés dans le style « **Corps de texte** ».

Résumé

Le résumé ne doit pas dépasser une page. Il est rédigé dans le style « **Corps de texte** ».

Table des matières

[Déclaration i](#_Toc133158325)

[Remerciements ii](#_Toc133158326)

[Résumé iii](#_Toc133158327)

[Liste des tableaux v](#_Toc133158328)

[Liste des figures v](#_Toc133158329)

[2.1 De Monolitque à Microservices 4](#_Toc133158330)

[2.2 DevOps 6](#_Toc133158331)

[3.1 Le Cloud et l’infrastructure as Code 10](#_Toc133158332)

[3.2 Conteneurisation 11](#_Toc133158333)

[3.3 La sécurité 13](#_Toc133158334)

[4.1 Critères de choix de solution 18](#_Toc133158335)

[4.2 Continuous Integration 19](#_Toc133158336)

[4.2.1 Les étapes décortiquées 21](#_Toc133158337)

[4.2.1.1 Build 21](#_Toc133158338)

[4.2.1.2 Les Tests 21](#_Toc133158339)

[4.2.1.2.1 Comparatif des outils existants (non exhaustif) : 23](#_Toc133158340)

[4.2.2 Résumé Continuous Intégration 24](#_Toc133158341)

[4.3 CI/CD TOOLS 25](#_Toc133158342)

[4.3.1 OUTILS 25](#_Toc133158343)

[4.4 Continous Deployment 27](#_Toc133158344)

[4.4.1 TERRAFORM ET LE PROVISIONNEMENT DE RESSOURCES ET CONFIGURATION MANAGEMENT AVEC ANSIBLE 29](#_Toc133158345)

[4.4.1.1 TERRAFORM 30](#_Toc133158346)

[4.4.1.2 ANSIBLE 30](#_Toc133158347)

[4.4.1.3 KUBERNETES 30](#_Toc133158348)

[4.4.2 SCANNER DE VULNERABILITE 30](#_Toc133158349)

[4.5 CONTINUOUS MONITORING 31](#_Toc133158350)

[Bibliographie 33](#_Toc133158351)

[Annexe 1 : Titre de l’annexe 35](#_Toc133158352)

Liste des tableaux

[Tableau 1 : Titre du tableau 5](#_Toc381707012)

Liste des figures

[Figure 1 : Titre de la figure 5](file:///Y:\BODER\1-Cours\160-travail_de_bachelor\Modeles_TDB_HEG_2014.dotx#_Toc381707014)

# 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 découle deviennent de plus en plus élevé. Le temps ou la complexité d’un système était une excuse pour son mauvais fonctionnement sont révolus.

Mais dans une société qui évolue rapidement, maintenir et améliorer des programmes a créé une myriade de profession et 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 me mise en production de leur nouvelle version, la culture de développement à évoluer, le reste de la chaine de mise en service s’y est plier et de nouveaux outils sont nés. Des outils qui malheureusement semblent avoir 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’étudiant 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 trouvait 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 dit legacy que personne ne voulait toucher. L’environnement de production de celui-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 dit de DevOps trainaient par ci par là mais était plus utilise 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 puants.

Il est compréhensible que dans toute société ou les ressources sont très limitées, s’offrir les services d’un spécialiste du domaine ou d’octroyer du temps à un de ses employé pour que celui-ci puisse passer du temps sur la mise en place d’outils ou de procédures. Pourtant comme tout bon investissement, c’est avec le temps que leur valeur apparaitra.

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 appeler le monde du DevOps mais plus particulièrement les cycles d’intégrations et de déploiement 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 générer 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 a d’autre étudiants ou toute personne souhaitant le comprendre. Il vulgarise un grand nombre de concepts et est également pensé les pour être lu par des décideurs dans de petites entreprise. 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 net 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’explications de concepts et technologies importants sur lesquels est basé le monde du développement actuel. Enfin, je parlerai des étapes et outils lié à la réalisation d’un pipeline assurant la délivrance d’un logiciel propre et sécurisé. Cette dernière étape sera accompagnée d’un projet.

# Historique des pratiques de développement d’application

Afin de comprendre comment nous sommes arrivées 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 extrêmes corrélées.

Lorsque le développement d’application moderne à commencer à exister, l’ensemble des technologies d’un projet était utilisé ensemble pour fournir le service souhaité. Le terme ensemble est même quelque peu léger, les technologies devaient être configurée et utilisée 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 appeler « Waterfall » (en cascade) y est pour beaucoup. Le principe est simple, chaque étape du processus est dépendant de celui qui le précède. On réfléchit à l’ensemble des besoins du projet, on en déduit une architecture, on comme ensuite à travailler en général de la partie logique vers la partie interface utilisateur et enfin on test. Tout problèmes apparaissant plus loin dans la cascade des étapes ne pouvait que difficilement être adressé vu que l’ensemble du projet était déjà construit.

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’avérait compliqué de modifier ceux-ci en profondeur si le besoin venait à évoluer. L’utilisateur final réel du logiciel ne pouvait que très peu le tester avant d’en avoir une version complète. Une durée de plusieurs année pouvait être nécessaire avant d’obtenir une version fonctionnelle et les besoins avait souvent évolué entre temps.

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

Dans cette optique, les parties intégrantes d’un logiciel ont été petit à petite découplé et ainsi commencer à dialoguer entre elle sans entre couplée 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 année 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 utiliser facilement et rapidement. Ce changement de paradigme força aussi la façon de construire les logiciels à changer et à s’adapter. C’est ainsi que l’architecture général des logiciels, leur cycle de déploiement et de vie on fait peau neuve.

## De Monolitque à Microservices

Comme énoncé précédemment, lorsque le développement logiciel à fait son apparition, une application était constituée d’un gros bloc et pouvait essentiellement être pensé 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 »\* .mais avec un nombre de fonctionnalités croissant et une interconnexion avec d’autres logiciel, ce bloc unique posait de nombreuses limitations si le logiciel souhaitait évoluer.

Toutes les fonctionnalités et par extension le service qui se charge de la faire marcher sont fortement couplée, 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 grandi.

La mise en production d’un projet à l’architecture monolithique comportait également quelques obstacles qui avait la fâcheuse tendance de se montrer 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 n’était pas du tout le même que celui de développement voir même différent de celui de test pour des questions de coût.

Il était donc monnaie courante de se retrouver pour les équipes à devoir tout lâcher pour régler un problème en production surtout si celui-ci n’était pas réplicable dans d’autres environnements.

Les logiciels étant de plus en plus porter 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 problème suivant le type d’applications notamment si des données doivent être disponible pour chaque utilisateur, celui-ci devra toujours utiliser le même serveur. Cette architecture est également gourmande en ressource, 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.

Avec tous ces désavantages, il a été rapidement compris que cette architecture ne permettrait pas du tout de proposer des logiciels pouvant évoluer et s’interconnecter facilement et encore moi qu’il soit utilisable par un grand nombre d’utilisateur simultanément.

Diagram

Description automatically generatedC’est pour cela que l’industrie à rapidement changer de cap est à commencer par découpler les services et les rendre suffisamment petit pour que leur compréhension, amélioration et maintenance soit facilité. C’est ainsi que les micro-services sont nés.

Figure httpsrubygarage.orgblogmonolith-soa-microservices-serverless

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.

Les microservices semblait résoudre beaucoup de problème sur le papier mais comme tout changement, il amenait avec lui de nouveaux problèmes. Avoir une myriade de petits services simplifiait grandement le développement de ceux-ci mais leur mise en production elle était devenue beaucoup plus complexe. En effet, chaque service demandait son lot de dépendances et configuration, 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.

## DevOps

Pendant de nombreuses années, les développeurs se chargeaient de coder le logiciel et les ingénieurs systèmes de mettre en place et maintenir les environnements de productions ou plus généralisé opérationnels.

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’à 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 eu fonctionnel sans trop se soucier de la suite. A l’inverse, les responsable des systèmes préparent des environnements bien pensés, et construit pour être sécure et maintenable. Lorsque le projet passe des mains des premiers au second, il arrivait très souvent que les dépendances ou configurations requises par le projet pour fonctionner ne soit pas les même que celle dont les environnements préparer 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 logicielles 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. »*

*https://azure.microsoft.com/fr-fr/resources/cloud-computing-dictionary/what-is-devops*

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 dédies, le pratique dites DEVOPS ont permis de soutenir l’évolution du développement et de la livraison de logiciel en permettant que l’ensemble des étapes se déroule sans accros et apporte une vraie plus-value au produit final.

Diagram

Description automatically generatedC’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 de milieu du développement et des opérations, chaque boucle comprenant les étapes de la vie d’une release logicielle lié à son milieu. Les professionnels liant leur activité à cette idéologie se chargeront de faire la jonction entre les deux mondes mais ne feront vraiment ni du développement ni de la maintenance de système.

Figure https://blog.ippon.fr/2020/07/29/le-devops-explique-a-mes-parents/

La situation actuelle qui ressort après cette évolution est mitigée, les grandes structures ont suivi la cadence est se retrouve 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 tomber face a des applications indisponibles même si cela arrive, le délai avant la reprise de service est en général court.

Pour les plus petites structures ou les organisations étatiques cependant, ce n’est pas le même constat. Mon expérience de plusieurs années m’a permis de dresser le portait 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. Celle-ci étant limitées, les choix technologiques se basait sur les connaissances d’une personne de l’équipe dans un domaine. Si cette personne venait à quitter l’entreprise, difficile de la remplacer. Cela couplé à des équipes dont le carnet des charges très remplie ne laissait pas beaucoup de place à la formation continue à amener 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 solution logiciel fait maison ou créer 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 indisponible sont problématique et pénible, les failles de sécurités peuvent être mortelles.

Je suis conscient que le travail pour résoudre ces problèmes ne se résume pas à la mise en place de quelques outils et l’adoption de certaines pratiques. Mais il ne suffit de pas grand-chose pour déjà aller vers le mieux, un pipeline de déploiement dans lequel les bons outils sont intégrer permet déjà de s’assurer que les programmes déployer ne souffrent pas de trop de faille de sécurité et qu’un bon nombre de bugs soit évités. Certaines technologies liées au monde du DevOps permettent également de facilité 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.

# 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 nécessaire au bon fonctionnement de notre pipeline. Certain juste utilisée individuellement amène une amélioration par rapport à l’état des choses actuelles 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 d’un certain nombre d’étapes. Chacune d’entre elle ayant pour but essayant de garantir au maximum que la version ne contient pas de bug, soit conforme à un niveau de qualité souhaité et surtout soit fonctionnelle.

C’est dans cet esprit qu’une chaine d’action en grande partie automatisé a vu le jour sous le nom de pipeline d’intégration et de distribution continue.

*« 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)*](https://www.redhat.com/fr/topics/devops/what-is-ci-cd)*désignent une pratique qui consiste à améliorer la distribution de logiciels »*

<https://www.redhat.com/fr/topics/devops/what-cicd-pipeline>

Le mot pipeline est utilisé car de manière image, nous allons avoir un tuyau qui transportera notre nouvelle version logicielle de la fin de sa création à sa mise en production. Comme nous le verrons ce projet sera triturer minutieusement à chaque étape d’avant d’être jugé acceptable pour continuer son chemin dans le tuyau.

Les termes intégrations 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 Deployement(Déploiement continu).

La partie CI se focalisera sur récupérer le projet une fois le pipeline déclencher ainsi que la partie test pré-déploiement.

La partie CD se focalisera sur pousser le projet test sur son environnement de production et 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 test sur le code, déploiera ensuite le projet sur les différents 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 dans un certain nombre d’entre-elles sont toujours mises en place. Nous reviendrons sur chacune des parties plus précisément.

## Le Cloud et l’infrastructure as Code

De nombreuses entreprises disposent de petits datacenters en interne encore aujourd’hui pour de nombreuses raisons, la première est souvent que certains outils aux besoin spécifiques y fonctionnent encore. La seconde est qu’une impression de maîtrise et de sécurité est ressentie lorsque les serveurs se trouvent chez soi. Ce genre d’infrastructure coûte cher et demande du personnel qualifié, surtout si les logiciels mis à disposition doivent l’être 24h/24. Sans compter que l’ajout de nouveaux outils, génère le 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 proposent une alternative qui si 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 est basé uniquement sur l’utilisation des ressources. Cela à une grande incidence sur le coût pouvant découler de l’utilisation de certains outils. En effet, pendant de nombreuses années, les entreprises devaient provisionner leur infrastructure pour tenir la charge maximum dont il avait besoin et accepter que malheureusement, le reste du temps une partie pouvant être conséquente de leurs ressources ne serait pas utilisées. Ce qui avait souvent pour incidence 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 limitait donc souvent dans le panel de leurs outils. C’est pourquoi beaucoup d’outils de testes ainsi que leur utilisation automatique peinent à faire leur chemin.

Heureusement, grâce aux solutions cloud deux possibilités s’offrent à eux. La première est l’utilisation d’outils dit cloud autrement dit tournant sur l’infrastructure de leur éditeur. Bon nombre de ces outils offre des prestations uniquement à l’utilisation. Evitant ainsi la maintenance de l’outil et les charges fixes liés à son utilisation. Cela donne l’accès à des outils poussé et maintenu à jour même à de petites structures. Un petit hiatus, nombre de ces solutions commence 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 moins de moyens.

La seconde est l’utilisation d’outils open-source couplé avec l’offre IAAS. 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 gérer avec des outils d’automatisation de provisionnement de ressources est une excellente façon de limitée la facture. Nous reviendrons plus précisément sur ces outils plus loin. Bien pensé et configurer, ces outils permettent de mettre en place ou d’activer uniquement les ressources nécessaires aux fonctionnement des outils voulu au moment où ils vont être utilisés. Avec le cout d’une instance d’uniquement quelques dizaines de centime à l’heure, l’utilisation de ce type de solution permet une réelle économie de coût et ainsi une option pour des petites structures.

## 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 la sécurisée.

Dans de nombreuses petites entreprise, la sécurité est souvent l’élément le plus laissé pour compte. Comme tous les frais liés à des situations potentielles, il est difficile de convaincre les décideurs ayant des ressources limitées de les mettre en haut de la liste de priorités. Certaines entreprises préfèrent même s’assurer financièrement contre les dommages que pourrait leur causer une attaque 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 ces produits. Pourtant, la perte ou le vole de données peux être synonyme de faillite.

Sécurité dans les PME Suisse Quote

Je vais tâcher d’expliquer comme les vulnérabilités sont traitées et répertoriées puis je listerais les plus communément découvertes sur des programmes et comment il est possible à l’aides des outils que nous verrons de les déceler.

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

Les CVE sont basé sur une organisation du même nom ayant mis en place un procédé de report et de suivi des vulnérabilités, celle-ci couvrent l’ensemble du monde informatique, des systèmes aux applications. Les CVE sont annoncé uniquement par des partenaires et entreprises reconnu afin d’assurer la véracité de l’information, le système est devenu la base du partage d’information lié à la sécurité. De nombreuses entreprises proposés même des récompenses pour l’annonce de vulnérabilité sur les produits. Permettant ainsi de profiter de la grande communauté présente autour du projet et assurer la sécurité de ses produits à moindre coût car même si certain « bug bounty » comme ils sont appelés peuvent atteindre plusieurs centaines de milliers de dollars, les sommes sont souvent beaucoup plus basses et ne représentent des sommes risibles pour de grandes entreprises. A titre d’exemple, Google aura payer un peu plus de 12 millions de dollars pour un équivalent de 2900 vulnérabilités reportées sur ses produits\*.

Diagram

Description automatically generated\*https://www.bleepingcomputer.com/news/security/google-paid-12-million-in-bug-bounties-to-security-researchers/

Figure 7httpswww.cve.orgAboutProcess#CVERecordLifecycle

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. Toutes vulnérabilité est loin d’être catastrophique et ne demande pas forcément d’y remédier instantanément.

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

* En plus d’être annoncé par une source fiable, elle doit être reconnu par l’autorité représentant le produit sur lequel elle a été découverte.
* Doit être solvable indépendamment d’autre bug, par exemple si une vulnérabilité est liée aux fonctionnements de plusieurs applications ou dépendances fonctionnant ensemble, une CVE sera créer pour chaque « codebase » autrement dit une collection de code source utilisée pour faire fonctionner un programme.

Une fois la solution trouver pour résoudre la vulnérabilité, celle-ci est ajouter 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 rechercher de vulnérabilité sur les systèmes et applications, de savoir quoi chercher et de proposer une résolution à ses utilisateurs.

OWASP TOP 10

Comme nous l’avons discuté, les vulnérabilités sont répertoriées quand il s’agit de problème lié à un produit utilisé. La très grande majorité des applications aujourd’hui sont basées ou utilises des produits existants les CVE permettent donc d’assurer que les outils sur lequel se repose notre application ne souffre 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 meilleurs pratiques évoluant constamment, notamment pour résoudre des soucis de sécurité, il est très fréquent que le code soit rempli de problèmes pouvant créant ainsi des vulnérabilités.

Le monde se dirigeant à pas de géant vers une digitalisation omniprésente, la sécurité des outils utilisés dans ce but est cruciale. 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. Ils proposent des outils permettant la mitigation des risques et vulnérabilités ainsi que du matériel et des ressources permettant de former, de garder le monde de la sécurité à jour.

Chart

Description automatically generatedOWASP édite toutes les quelques années une liste contenant les 10 risques les plus répandus permettant une attaque sur une application. Cette liste est la référence en termes de points devant être vérifiés par une équipe avant de rendre son application disponible à l’utilisation. Vu qu’il s’agit des plus connus, il est garanti que ce seront les premières portes d’entrée testées par tout entité souhaitant du mal à votre application. Comme vous l’aurez compris, nous parlons ici des CWE mentionné plus tôt.

Figure 8https://owasp.org/www-project-top-ten/

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 ou 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 réestives 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 adressé lors de tests sur le code.

Cryptographic Failures : Ici un grand nombre de scénario sont possible 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 encrypté ou utilisé tel quel et donc lisible. Ici une partie peut être vérifier sur le code, une autre sur le système, le reste dépendra des bonnes pratiques.

Injection : Il s’agit d’un des risques les plus connu qui semble heureusement être moins présent depuis la dernière mise à jour de la liste. Ici tout champs permettant à l’utilisateur d’y entrée ses informations ou services pouvant recevoir de l’information qui sera transmises pour effectuer un appel à un autre service. Toutes informations fournies par l’utilisateur et utilisé par l’application doit être vérifiée proprement. Il est sinon possible d’accéder à ou d’altérer des données. 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 relevé comme par exemple, des informations de connexion, pour accéder à un autre service étant mal protégés ou encore des informations sensibles reporté dans la génération de message d’erreur.

Security Misconfiguration : Comme mentionnée, 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’outils, ces ports n’étant pas utilisé, ils peuvent être utilisés comme une porte d’entrée. Cette partie peut être test 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 résolu par l’éditeur du service. Ici un scan sur le code et post déploiement sur le système permettra de relevé 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 relevé mais pour d’autres il s’agit de bonnes pratiques et standards.

Software and Data Integrity Failures: Il s’agit d’une catégorie important pour notre pipeline car lorsque nous faisons appel à des dépendances pour notre application, des téléchargement s’effectue sans vérifier la source et la validé des données téléchargées. Ici des logiciels dédiés pour la vérification de dépendances sont disponibles mais la mitigation demande de mettre en place des services n’étant pas nécessaire pour des petits projets. Dans le cas de notre projet, nous contenterons de nous baser sur des sources reconnues et des images de bases officielles.

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 données par l’utilisateur pour la récupérations d’informations, pour par exemple récupérer des données afin de pouvoir les utilisé dans le logiciel. Ici également difficile de tester mais certains points peuvent être relevé 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 sécure une fois validé 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 connaissances de cause, les collaborateurs auront tendances à régler les problèmes exposant leur organisation aux risques les plus élevés.

## Contrôleur de version

Cette technologie est la plus rependue, je n’ai croisé aucune organisation fonctionnant sans elle. Même si certaines ne l’utilise pas à son plein potentiel. Je ne vais pas m’attardé dessus, mais il n’est reste 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 marché pour une bonne raison, historiquement le logiciel effectue le travail parfaitement. Il existe un grand nombre de concurrent qui chacun représente une petite part du marché, les avantages mis en avant par chaque concurrent est souvent lié aux autres produits qu’ils proposent et souvent dans une idée de garder l’ensemble des ressources utilisées par une entreprise au prêt d’un seul fournisseur.

Dans notre cas git, rempli l’ensemble de nos critères de sélections. Afin de comprendre ce qui va constituer le trigger, nous allons nous attarder sur quelques actions que le logiciel permet d’effectuer.

Comme nous l’avons 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 sens de faire passer tous ces tests à notre projet uniquement lorsque nous amenons des modifications à celui-ci.

Comme Git fonctionne-il : le principe 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 git appel un repository. Cela n’est rien d’autre qu’un dossier héberger 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 contribuer. Vu que chacun travaille sur le même code, git proposes quelques actions et outils permettant à tout le monde de travailler ensemble facilement.

Chacun va travailler sur sa version du code mais ne vas pas la partager avec les autres avant d’être sûr d’avoir correctement effectuer 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 garder à part. La branche contenant votre projet principal mis en production s’appellera par défaut master ou main suivant les envies de Microsoft. Vous pouvez également disposez 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 souhaiter et ainsi fusionner les deux.

C’est ce qui suit qui va être important pour nous. Pour l’instant notre contributeur travail localement et veut maintenant partager son code avec le reste de l’équipe et 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 qui envoie le code sur le dépôt distant va signaler que notre processus doit être lancer.

Il est important de préciser que git ne fait pas que les quelques actions mentionner mais bien plus et que les actions ont été simplifier 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.

## Conteneurisation

La conteneurisation règle de nombre problème lié aux différents environnements nécessaire à la conception et vie d’un programme. De nombreuses entreprise travail 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 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.

Comment il l’a été énoncé, 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 problème.

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

Même si les VM(s) sont un outil excellent et qui permettrait de régler une bonne parti des problèmes liés aux différences d’environnement, elles souffrent encore de certains problèmes. Une machine virtuelle doit avoir des ressources précises lié à son hôte qui lui sont allouée à sa consommation, de pars son OS et ses dépendances elle a déjà beaucoup un poids et une consommation non négligeable. 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 qu’il n’en faut. Il est tout à fait possible de faire attention et logger tout changements apporter afin de pouvoir les supprimer si besoin. Mais il n’en reste que cela n’est pas très automatisable et très prompt aux erreurs humaines. Sans compter que dans un monde de micro-service, il est très important de pouvoir compter sur des environnements légers au risque de voir ses coups exploser et ses performances amoindries.

Un concept est venu régler ces soucis, la conteneurisation. Le principe est assez simple, Un outil permettant d’automatiser la création d’un environnement possédant le minimum pour faire fonctionner chaque application. En parlant de minium, 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 ce qui est nécessaire, c’est que celui-ci est autonome, il pourra donc être installer et fonctionner 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 rapportant à un conteneur de transport convient à merveille car notre conteneur est fait pour être léger et transportable.

Graphical user interface, application

Description automatically generatedComme nous le verrons ce concept convient parfaitement aux micros-services et à l’utilisation de ressources sur demande. Car notre conteneur ne contient en général rien de persistant. Il peut être utiliser et ainsi créer son environnement, y faire tourner le programme lié et ensuite être détruit. Il est donc parfait pour rapidement créer des nouvelles instances de supports si un service est en demande accru et à l’inverse, les détruire si ce n’est plus le cas.

Figure httpsmedium.comhackernoonwhat-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 de qu’offre l’ensemble de chaine de déploiement pour mutualiser les ressources à des prix attractifs.

### DOCKER

Git n’aura plus d’incidence pour le reste de notre projet autant docker lui est d’une importance capitale. Comme nous l’avons discuté en parlant de 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 ou nous en auront besoin, qu’il de faire passer ces tests à notre projet ou de le déployer, un outil pour construire nos containers et les configurer sera nécaissaire. 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 effectués.

Encore une fois, nous nous trouvons dans une situation ou le moindre problème de configuration entre notre environnement de test est celui de développement pose rapidement des problèmes. La conteneurisation règle ce souci et va nous permettre de construire notre environnement et d’y faire tourner nos tests. Si nos développeurs travail dans un environnement également mis en place via un outil de conteneurisation, le moindre changement effectuer se fera via le fichier de configuration de l’environnement conteneuriser. Le fichier pourra ainsi être passé avec le reste du projet pour être utilisé lors des étapes du pipeline.

Il existe quelques outils de conteneurisation mais le plus connu est Docker. Ici le choix est simple il existe certes d’autres outils mais le plus complet et surtout le plus communément utilisé est docker. Le second argument est très important car 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 nécessaire à la création est 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’hôtes et d’utiliser ses services pour par exemple communiquer avec l’extérieur ou rendre accessible les services installés dans le container. Pour que cela fonctionne dans l’environnement, il est important de préciser que Docker doit être installer 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 revendrions plus tard.

Lorsque notre fichier de configuration sera prêt, celui-ci sera construit(build) est comme pour une VM donnera naissance à une image. Une image n’est rien d’autre que la représentation d’un environnement sous forme d’un fichier. Ce fichier pouvant ensuite être monter pour 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 basé 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 lesquels il sera possible de construire des déclinaisons en fonctions des besoins.

DOCKERFILE

Graphical user interface, text, application

Description automatically generatedComment va se présenter notre fichier de configuration : Celui-ci s’appelle Dockerfile voici un exemple simple donné par Docker.

Figure 5https://docs.docker.com/get-started/02\_our\_app/

FROM va nous permettre d’aller récupérer une image existant 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 dispositions par la communauté s’appelle Docker Hub. Si nous venions à construire notre ficher Docker téléchargerait l’image choisie et la monterait. Cela donnera une base sur laquelle la suite de la configuration sera effectuée.

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.

RUN va permettre de lancer des commandes Shell par default ou autre si préciser et ainsi fini la configuration de l’environnement. Il est possible de lancer autant de commande que souhaité.

CMD lui est unique à chaque fichier de configuration. Il est le point d’entrer du service que l’on souhaite faire fonctionner dans le conteneur, Il permettra de lancer l’service pour que celle-ci soit fonctionnelle 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 de base une boite hermétiquement fermée, si nous souhaitant que notre service soit disponible pour être utilisé, il est essentiel d’ouvrir une porte d’accès.

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 créer pour l’occasion.

Il existe d’autres mot clé d’instruction dans docker, certain sur lesquels nous reviendrons dans la partie projet.

DOCKER COMPOSE :

Les micro-services domine aujourd’hui le monde de l’architecture logicielle. Une application ne sera que très rarement composée d’un seul service. Nous souhaitons que ces services soit découplé mais suivant les besoins, ils ont souvent besoins d’être disponibles les uns pour les autres. L’idée est donc de combinée plusieurs DockerFile en un seul fichier afin de n’avoir à monter que celui-ci et que les services soient configurés pour fonctionner ensemble.

Un fichier dockercompose va permettre de faire tout cela. Une fois monter, il construira un conteneur qui contiendra les conteneurs de chaque service présent dans le fichier.

Graphical user interface, text

Description automatically generated with medium confidenceSa structure est du type :

Figure 6https://www.freecodecamp.org/news/what-is-docker-compose-how-to-use-it/

La version est annoncée en début de fichier est 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 découplé 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 proche de ceux d’un dockerfile. Nous avons notre liste de services, chacun disposant d’un nom et lié à une image. Tout d’abord, le nom n’est pas présent que pour une question de clarté, 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. Ensuite l’image, pour que notre docker compose puisse fonctionner, il doit pouvoir accéder aux images demandées, s’il s’agit d’image génériques de services par exemple Node.js pour le premier service, il ne s’agira que de node (docker hub à simplifier le nom des images les plus communément utilisées).

Dans le cas où le projet demanderait l’utilisation d’une image créer sur mesure, il est important que celle-ci soit hébergée sur un dépôt afin que les différents servers puissent venir la récupérer afin 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, 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 pourra être créer pour des tests par un développeur et ne devra donc pas être utilisé à des fins de production, on utilisera un tag spécifique évitant ainsi tout problème.

Un des problèmes de la conteneurisation est la persistance des données. Comme nous l’avons discuté. Les environnements sont construits puis détruit au grès du besoin, mais que faire si nous avons des données qui sont persistante est que nous devons conserver même une fois l’environnement détruit. Lorsqu’il s’agit de données 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, ce qui permets aux données d’être toujours disponibles sur la machine même si un nouvel environnement venant à remplacer l’ancien du moment que la référence sur le volume est la même.

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 tester, 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.

DOCKER DANS NOTRE PROJET

## Conclusion

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 comme nous allions 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, les plus d’outils seront intégrées, meilleur sera le résultat.

# L’outil au cœur du cycle d’automatisation

Dans l’optique d’offrir un outil flexible est 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 pour si le nombre de projet est élevé. C’est pourquoi il est préférable de reste sur une architecture aussi proche de l’agnostique que possible. Certes certain partie sont propre au langages utilisés, en utilisant la conteneurisation, l’environnement étant packager avec le projet, une grande partie du pipeline pourra ainsi être réutiliser pour d’autre 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 complique qu’il puisse le paraitre mais 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 performant que possible seront dans un grand nombre de cas développé 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.

## Critères de choix de solution

Afin d’avoir une structure de pipeline qui soit portable sur n’importe quel architecture ou hébergeur, il est préférable d’éviter des solutions développer 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 gratuite et supporter 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 surement pas le plus complet possible mais nous essaierons de sa complexité de mise en place et d’utilisation à un degré raisonnable pour éviter de se retrouver avec un monstre dont nous ne maitrisons pas la complexité.

Nous nous assurerons également d’utiliser des solutions qui ont prouvé quelle aller survivre dans le temps, qui sont reconnu dans par la communauté du monde informatique et qui sont utiliser communément.

Afin d’éviter de tomber dans la complexité de services distribué, 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 y ajouter une capacité à pouvoir support une augmentation de charge conséquente requière encore une autre expertise et risque de générer plus de problèmes qu’il n’en règle 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. Afin que celui-ci soit vraiment utilisable dans le monde actuel, il faut qu’il soit portable sur le cloud.

## Continuous Integration

Diagram

Description automatically generated

La première étape est pour déclencher notre pipeline 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 cas un répertoire Git héberger 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 est pas conséquent doit être abordé.

Une fois notre trigger déclencher, notre projet va être récupérer sur son répertoire. Avant de discuter 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 pour 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 avions aborder la conteneurisation.

### Les étapes décortiquées

Nous allons nous arrêter ici sur les étapes de notre pipeline et les outils qui seront nécessaire à son bon fonctionnement. Comme nous l’avons discuté, si une des étapes venant à ne pas se déroulée 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 ses changement n’impact les différents environnements de tests ou de production.

#### Build

La première étape peut sembler triviale mais n’en n’ait pas moins essentiel 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, certain langage afin de pouvoir être utilisé par une machine doivent être compliés et ainsi transformer en fichier exécutables. Cette étape est cruciale pour tous les langages demandant à être compliés 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.

#### Les Tests

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

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, s’il l’on veut tester une fonction, il nous faudra écrire plusieurs tests s’assurant que la fonction remplie bien ce qu’on attend d’elle suivant les informations qu’on lui fournit où qui existent déjà dans le projet. Et à l’inverse qu’elle n’altère pas certaines données. Le but est simple, s’assurer que 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 appartement 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 directement mais il est plus sage d’effectuer ces tests directement sur les parties du code de l’application y faisant appel.

Ces deux premiers jeux de tests sont souvent réalisés par les développeurs, ils sont adjacents au code et ne requière pas d’utilisation d’outils externes particulier. Chaque langage dispose de son propre outil de test, celui-ci est souvent en extension et installer 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 réussi ou non.

Un dernier test effectué directement sur le code, appeler la revue du code. Un logiciel externe va se charger de scanner le code à la recherche de code déprécier ou pouvant générer des problèmes de sécurité. Ici se trouve un grand nombre d’outils disponible 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’anticipé tous les scénarii possibles. Les outils vont s’appuyer sur les tests réalisés précédemment, ils ne sont en aucun une manière de remplacer les tests unitaires et d’intégrations. Il s’agit ici de tests dit 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 voir 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 compliqué et donc difficile à maintenir, l’outil peut proposer une alternative.

En cas de patterns utilisé pouvant générer des problèmes ou des fonctions pouvant être poussées à l’erreur en cas de passage d’informations problématique en paramètres.

Et enfin, dans un souci d’assurer la sécurité de l’application, l’outils 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 vulnérabilité connues pouvant être exploité 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.

##### 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 que pour les tests statiques. Le dernier paramètre à prendre en compte correspond à votre l’hébergement de l’outils, si vous travailler dans un environnement fermé, utilisé 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 outils 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 dupliquer ou de fonctions mal optimisées.

-La détection de vulnérabilité ou de problème pouvant amener à des vulnérabilités.

-Plus d’une 20aines de langages supportés.

-Sont intégrable avec la majorité des cloud provider, gestionnaire 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 tout outils de reporting, la quantité d’information 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 performance et de la sécurité mais d’autre peuvent être plus secondaire et l’entreprise 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. Disponible 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 langage ainsi que de la compatibilité avec le plus d’outils. Il s’agit également d’un des outils les plus utilisé dans sa catégorie, il dispose d’une version gratuite déjà très qualitative et est disponible en version cloud ou on-prem.

Veracode convient autant à l’analyse statique que dynamique du code avec un accent mis sur la sécurité. Il propose une option de scan lié à l’analyse de la composition du logiciel ce qui permet de refactorisé jugé complexe en plus petits morceaux moins complexes simplifiant ainsi la maintenance du logiciel. Il est disponible uniquement en version cloud.

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.

PHPUNIT ET SONARQUBE DANS NOTRE PROJET

### 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 devrait pouvoir recevoir toutes les informations nécessaires à la réalisation du code pour un logiciel respectant les attentes et standards demandé. Cette partie du pipeline ne couvre certes pas tous les tests mais peut être utilisé 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 voir 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 à fait son chemin dans le monde du développement, au point de créer le « Test Driven Développement » consistant à écrire les testes avant le code et ainsi s’assurer d’entrée 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 nombre petites entreprises. Cependant les tests ne sont pas la partie à oublier, quelques tests unitaires sur des fonctions clé ainsi que des testes 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.

## CI/CD TOOLS

Afin de faciliter la réalisation de ces tests, d’avoir un suivi sur leur retour et d’automatisé toute la chaine du pipeline, il faut un outil permettant d’orchestrer la chaine 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 différent type de tests et de déployer dans différents environnements en fonction de la source souhaité. Typiquement, un pipeline peut être prévu en fonction d’une branche d’un dépôt de code pour tester en profondeur certain aspect 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 est réalisé 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’outils 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 prévu unitaires, fonctionnel et de qualité de 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.

### OUTILS

Nous allons nous intéresser à 3 outils de CI/CD parmi les plus courant sur le marché. Le monde technologique tournant autour du DevOps étant devenu très au monde pour de bonne raisons, il existe un très grand nombre d’outils. La majorité des plus grands hébergeurs disposent de leurs propres outils de CI/CD voir 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 proposé par Amazon Web Services et permettent la mise en place d’un pipeline rapidement et efficacement avec peu de configuration dites système. Je pense cependant, qu’il est important de rester agnostique et de se baser sur des outils open-source et ne liant pas la configuration à un hébergeur. Le vent tourne vite dans le milieu technologique on ne sait jamais trop de quoi est fait demain et ne pas pouvoir se permettre de migrer son infrastructure facilement peut s’avérer coûteux.

La majorité des outils se base 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 différent environnement. 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 exertion à GitHub pour gérer des dépôts de codes. Il a évolué en un outil CI/CD disposant de nombreuse fonctionnalité adjacente lié aux travails 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 ce n’est pas arrêté là, il intégrer un scanneur de code, ainsi qu’un outil de monitoring. Il permet 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 développer 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-in. 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 ces 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 intégrer. Il est le choix d’entreprises plus conséquente, souhaitant un GUI plus poussé et un support rapide. Il propose également la possibilité d’exécuter en parallèle différent conteneurs et ainsi environnement 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 puissance de calcul pour éviter un temps d’exécution trop long.

JENKINS DANS NOTRE PROJET

## Continous Deployment

Comme il l’a été mentionnée, l’acronyme CD peut représenter deux termes. Le premier est continuous delivery, le second, continous deployment. Le second étant l’extension du premier. Continous delivery représente la possibilité de déployer une nouvelle version d’un logiciel à n’importe quel moment. Cela permet aux développeurs de déployer sur l’environnement souhaité leur travail à chaque nouvelle fonctionnalité aussi petite soit-elle 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 deployement pousse le principe un peu plus loin, en automatisant l’ensemble de la chaîne de déploiement. Les tests automatisés sont jugés suffisamment complets pour assurer que dans le cas où le logiciel en passe la totalité, il peut directement être déployer dans l’environnement souhaité voir même en production.

Le continous deployement est l’extenstion du delivery dans le sens ou si delivery est suffisamment bien fait, il se transforme en continous deployement.

Nous avons énoncé les étapes nécessaire un bon cycle d’intégration continu ou le but était de fournir toutes les informations nécessaires aux développeurs pour garantir que leur travail respectait les attentes de l’entreprise, ici il s’agira de donner toutes les informations aux équipes opérationnels pour que celles-ci s’assures que leur architecture et ses systèmes est pérenne et sécurisée.

Diagram

Description automatically generated

Figure httpswww.qentelli.comthought-leadershipinsightscontinuous-deployment#pipeline-of-cd

Le déploiement est première 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 configurer de la même manière que celui dans lequel il sera déployé. Cependant, cela peut avoir ces 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 automatisé, 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 pour 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 stagne 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 possible de voir les effets des modifications apporté 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, difficile d’envisagé 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 crash arrives sur des logiciels mis en production même parmi les plus grandes sociétés du monde informatiques.

### TERRAFORM ET LE PROVISIONNEMENT DE RESSOURCES ET CONFIGURATION MANAGEMENT AVEC ANSIBLE

Nous avons énoncé les outils de provisionnement de ressources un peu plus tôt. Ceux-ci servent à mettre en places les ressources nécessaires au fonctionnement d’un service. Ils sont peuvent être utilisé de nombreuse manière différente mais nous allons 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, voir 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.

Un des autres problèmes liés aux coûts des ressources est 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 nous 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 nommée Ansible.

#### 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 lui peu 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. Typiquement pour créer les environnements souhaités. Une autre option et de l’avoir sur une machine mutualisée comme une VM interne à votre entreprise et de laisser les développeurs les exécutés 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 utilisé qui ont une forme impérative qui elle annonce ce qui va être fait sous forme de commandes. De la même manière que Docker, Terraform va se baser sur le fichier de configuration présent à l’emplacement où est exécuté la commande de lancement. Ce qui permet de simplement se déplacer au bon endroit pour lancer la configuration, dans cette idée, il suffit de préparer l’ensemble de 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 modifier 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 ».

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 environnement inutiles même pendant des tranches horaires, typiquement la nuit période durant laquelle vos équipes ne travailles pas.

Notre script Terraform :

#### 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 fort.

Ansible va donc permettre d’automatiser des tâches comme la configuration d’un environnement, l’installation 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.

Ansible à 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 utiliser pour réaliser les tâches sur la machine cliente. Ces fichiers s’appel 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 provisionner avec Terraform car tout ce dont nous avons besoin se trouve dans le container préparé, il faut tout de même installer docker ou K8s.

La partie utilisant Ansible de notre projet.

#### KUBERNETES

### SCANNER DE VULNERABILITE

Les scanner de codes décrits dans les chapitres précédents sont les tests effectués directement sur la logique d’une application. Le but étant de prévenir toutes erreurs ou mauvaise utilisation pouvant amener à une vulnérabilité. Le scanner de vulnérabilité lui 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 sont trouvées par le logiciel. Il se chargera ensuite de relevé 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 que à leur équipe ne pouvant plus ignorer les vulnérabilités existantes. La sécurité semble malheureusement tomber dans les problèmes qui sont ignoré tant qu’ils ne sont pas révélés et accompagné d’avertissements.

Les scanner de vulnérabilités sont un outil vraiment important dans la lutte pour sécuriser une entreprise. Certes, une grande partie sont payant 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 intégré 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 multiplie et même sur les PME.

OWASP ZAP est un scanner de vulnérabilité gratuit développer et maintenant par la communauté d’OWASP. Il est orienté pour les applications web et sera donc moins précis sur la partie infrastructure. Il a le mérite d’être simple d’utilisation et de déjà donner un grand nombre d’information. Il peut être installer localement sur une machine ou intégrer dans un pipeline. Il n’offre malheureusement pas de version cloud mais cela peut être une bonne chose, vous permutant 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 choisis. Il dispose même d’un version plug-in pour navigateur web permettant de tester rapidement un logiciel via son url d’accès.

Avec ce genre d’outil. 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 de ce type ferait une grande différence.

## 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 tester et correctement déployer, 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 continuos monitoring (surveillance continue). Ce type d’outil à plusieurs utilités et sa valeur sera complète uniquement si le reste de l’ensemble du cycle a été correctement réalisé. Si cela n’est pas le cas, cet outil peut être utiliser 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.

Il permettra de déceler et anticiper un certain nombre de problème, comme boquer l’IP de la source de 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.

# Conclusion

A rédiger

Bibliographie

Le style de paragraphe appliqué dans les références doit être « **références bibliographiques** » : Arial 11 points, interligne simple, justifié, espace en dessus 6 pt, espace en dessous 6 pt. Voici un exemple :

FOREST, David, 2011. *Droit des données personnelles*. Paris : Gualino, 2011. Droit en action. ISBN 9782297015028.

<https://www.redhat.com/en/topics/security/what-is-cve>

<https://www.cve.org/ResourcesSupport/AllResources/CNARules>

<https://owasp.org>

<https://www.zaproxy.org/getting-started/>

<https://www.sonarsource.com/>

<https://www.codacy.com/>

<https://www.veracode.com/>

<https://en.wikipedia.org/wiki/Codebase>

https://www.crowdstrike.com/cybersecurity-101/observability/continuous-monitoring/#:~:text=Continuous%20monitoring%20is%20an%20approach,time%20to%20address%20them%20quickly.

Securing Devops- Safe Services in the Cloud

Generic Pipelines Using Docker\_ The DevOps Guide to Building Reusable, Platform Agnostic CI\_CD Frameworks(Book)

The DevOps Handbook\_ How to Create World-Class Agility, Reliability, and Security in Technology Organizations(Book)

Cloud Native DevOps with Kubernetes(Book)

The DevOps Adoption Playbook. A Guide to Adopting DevOps in a Multi-Speed IT Enterprise(Book)

The DevOps 2.0 Toolkit\_ Automating the Continuous Deployment Pipeline with Containerized Microservice (Book)

Pipeline as Code - Continuous Delivery with Jenkins, Kubernetes, and Terraform (Vidéos)

Annexe 1 : Titre de l’annexe

Votre texte ou le document mis en annexe.

Évitez de mettre deux annexes sur la même page. Faites plutôt un saut de page puis indiquez le numéro de la nouvelle annexe et son titre.