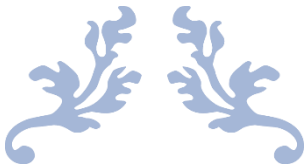


DOCUMENT D'ARCHITECTURE

08 AVRIL 2020

AUTOMATISATOR
Rue Pierre de Maupertuis, 35170 Bruz

TABLE DES MATIERES

1 GÉNÉRALITÉS

1.1 Description du document

Ce document décrit les composants techniques mis en œuvre pour la réalisation de la solution Automatisator, la façon dont ils ont été intégrés dans la solution réalisée, ainsi que la façon dont ils s'articulent les uns autour des autres. Il a pour but d'expliciter les orientations choisies pour la phase de réalisation d'Automatisator.

1.2 Présentation globale de la solution Automatisator

1.2.1 Objectifs

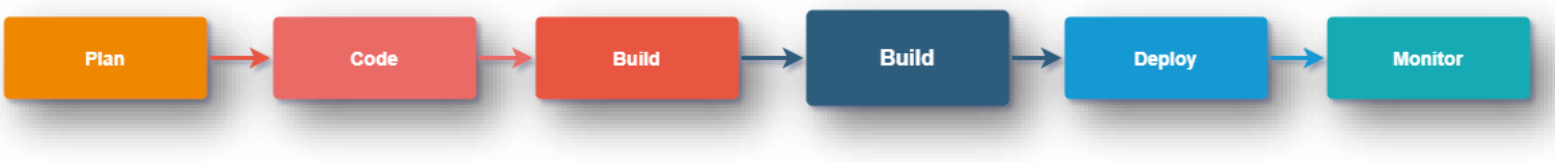
Automatisator est une solution clé en main de chaîne DevSecOps pour le développement, la production et la maintenance de l'ensemble de l'exposition. Elle implique la création d'une culture de « Security as Code » avec une collaboration continue et flexible entre les équipes de développement, production, et les équipes de sécurité. Elle implique l'utilisation du mouvement DevSecOps, elle se concentre sur la création de nouvelles solutions pour des processus de développement logiciel complexe dans un cadre agile et sécurisé.

L'architecture logicielle d'Automatisator s'appuie sur des composants libres et gratuits, dont la robustesse, la qualité et la pérennité ont été soigneusement étudiés, et l'attention a été portée sur la possibilité de répartir ses outils sur plusieurs machines physiques, afin de permettre une souplesse suffisamment grande.

De plus un certain nombre de règles de validation doivent pouvoir être appliqué aux données envoyées pour vérifier qu'elles respectent les normes de conformité de l'entreprise ou simplement qu'elles soient cohérentes entre elles. Ces règles doivent pouvoir être paramétrées.

1.2.2 Processus

La solution Automatisator sera donc articulée autour du processus suivant :



- **Plan :** Organisation et de vos tâches et configuration de vos infrastructures.
- **Code :** Développement, révision et merge de code sur des outils de collaboration.
- **Build et Test :** Outils d'intégration continue (CI) pour compiler votre code et exécuter des tests continus (tests unitaires) minimisant ainsi le risque de bugs et de vulnérabilités.
- **Deploy :** Déploiement continu de votre application en production, le gitlab-ci sera également utilisé pour déployer vos codes quand toutes les vérifications de la partie Test seront satisfaites.
- **Monitor :** Surveillance des métriques et des logs de votre application afin de découvrir l'impact sur l'expérience de l'utilisateur.

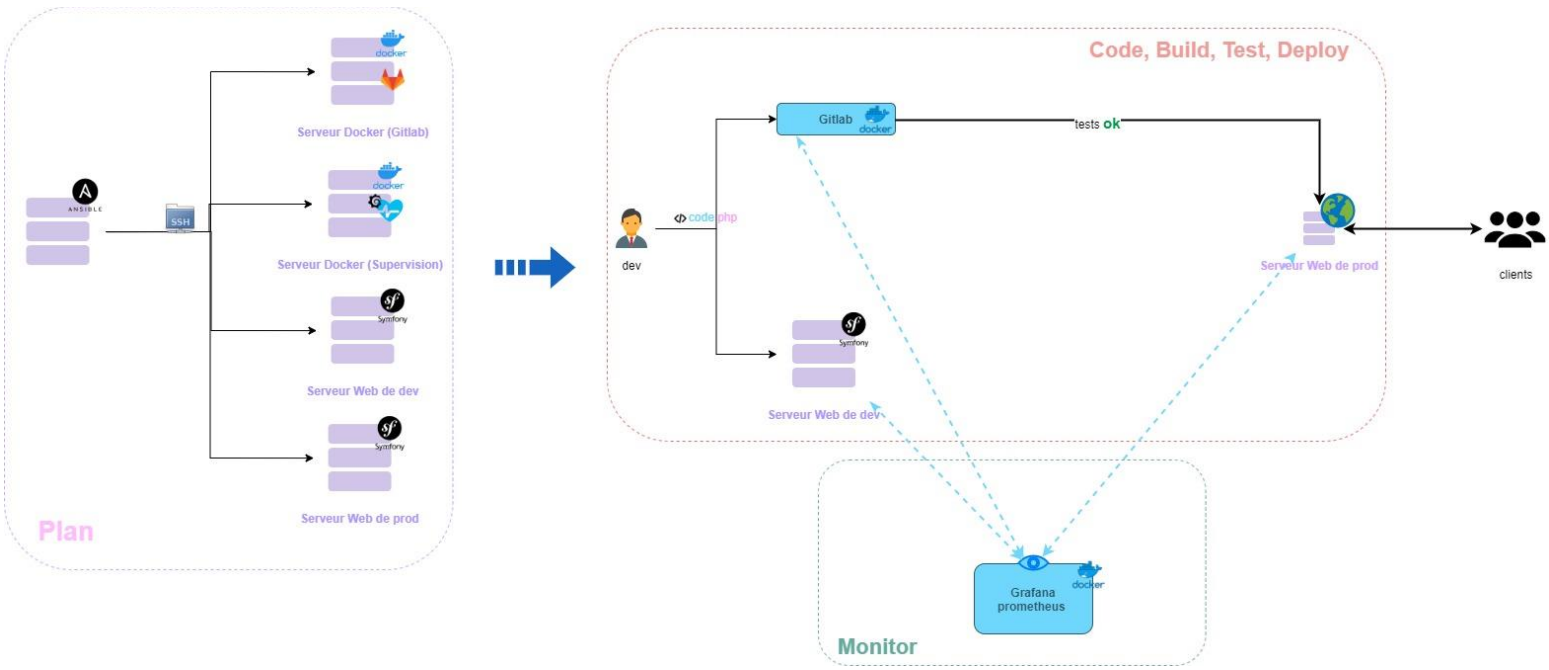
La solution technique conçue doit donc bien évidemment prendre en compte ces contraintes mais également permettre au maximum l'automatisation de ces étapes.

1.2.3 Principe de la solution

Le principe retenu pour la conception de la solution est la centralisation des processus depuis un minimum d'outils open source et la facilitation d'exportation des données vers une autre infrastructure. Le but est de fournir aux différentes équipes la meilleure expérience, réactivité et collaboration et d'éviter de gérer plusieurs outils rendant ainsi leurs maintenances complexes.

1.3 Détails de la solution Automatisator

1.3.1 Architecture Globale



1.3.2 Outils des étapes du processus

1.3.2.1 Plan

La partie Plan sera organisée depuis l'outil d'automatisation de configuration Ansible. L'objectif est de pouvoir configurer et orchestrer automatiquement un ensemble de machines avant de pouvoir les utiliser. Cette automatisation garantit un bon niveau de rapidité, sécurité et d'assurer la conformité avec un ensemble de règles de déploiements.

1.3.2.2 Code/Build/Test/Deploy

La partie Code/Build/Test/Deploy sera accordée à une application unique nommée Gitlab intégrant sa propre solution d'intégration et déploiement continu nommée Gitlab-Ci. Il gérera donc l'ensemble du cycle de vie DevOps qui permet aux équipes de mieux travailler ensemble et d'apporter plus de valeur aux clients, plus rapidement.

Il y parviendra en raccourcissant la durée de votre cycle DevSecOps, en reliant les silos et les étapes et en vous supprimant le travail manuel. Cela signifie également un flux de travail uni qui réduit la friction des activités traditionnellement distinctes comme les tests de sécurité des applications.

1.3.2.3 Monitor

Afin de garantir une bonne qualité de ces services, la partie Monitor sera gérée par la solution Grafana et Prometheus. Cette étape supervisera les différents serveurs et services inclus dans la solution Automatisator.

1.3.3 Liste des serveurs

Serveur	vCore(s)	RAM	Stockage
Serveur Ansible	1 vCore	4 Go	40 Go SSD
Serveur Docker (Gitlab)	2 vCores	8 Go	80 Go SSD
Serveur Docker (Prometheus et Grafana)	1 vCore	4 Go	40 Go SSD
Serveur Web de dev	2 vCores	8 Go	80 Go SSD
Serveur Web de prod	2 vCores	8 Go	80 Go SSD

1.3.3.1 Choix des ressources des serveurs

1.3.3.1.1 Serveur Web

En vue du nombre de visiteurs estimés qui sont de 100 000 visiteurs par an, nous avons opté pour des serveurs web avec 8 Go de Ram et 2vCores, l'application web pourra donc supporter la charge de travail estimée pour ce nombre de visiteurs. Il est recommandé que le serveur web de développement soit identique à celui de la production afin de simuler la même situation que dans l'environnement de production.

1.3.3.1.2 Serveur Ansible

4 Go de RAM est recommandées par Ansible pour une charge de travail modéré (100 jobs max simultanés).

1.3.3.1.3 Gitlab

1 cœur et 4 Go de mémoire peuvent prendre en charge jusqu'à 100 utilisateurs, mais l'application peut être un peu plus lente en raison de l'exécution de tous les travailleurs et tâches d'arrière-plan sur le même cœur. 8 Go de RAMs et 2 cœurs sont le nombre minimum recommandé par Gitlab afin de prendre en charge jusqu'à 100 utilisateurs.

1.3.3.1.4 Prometheus et Grafana

Prometheus et Grafana restent une solution de monotoring qui consomme très peu de ressources, 4 go de mémoire et 1 cœur suffisent pour superviser les serveurs et ressources nécessaires au bon fonctionnement de la solution Automatisator.

1.3.3.2 Services et caractéristiques inclus dans les serveurs

Évolutivité	Ascendante via l'espace client
Protection incluse	100 Mbps - Trafic illimité
IP	Anti-DDoS
Gestion	Jusqu'à 16 IPs géolocalisées
Sauvegarde	Manuelle (Snapshot)
Reboot et réinstallation	Illimités, depuis l'espace client
Disques additionnels	Possibilité d'en rajouter
Accès	root

1.3.4 Hébergement

Étant donné l'importance accordé à l'écologie, la solution devra être exploitée dans un datacenter ayant une empreinte Carbone très réduite. De ce fait la solution doit être hébergée en France. Le choix s'est porté vers l'entreprise OVH sur l'un de ces 15 datacenters situés en France.

1.3.4.1 Réduction d'énergies

Dans l'optique d'un processus de réduction de la consommation d'énergie, l'ensemble des serveurs des datacenters seront refroidis avec un système de watercooling, ce qui permet de disperser 70 % de la chaleur émise par le processeur des serveurs.

Les climatiseurs y sont remplacés par un système de ventilation naturelle assurant, grâce à des allées chaudes et froides, la régulation des températures. Cette prouesse permet d'atteindre un PUE (Power Usage Effectivess, un indicateur de référence mal standardisé, ndlr) compris entre 1 et 1,2. Cela est possible grâce à une tour creuse, destinée à être traversé par l'air extérieur afin de réguler la température des serveurs. Une fois réchauffé par les machines, l'air est évacué en face arrière des serveurs, par la « cheminée ».

1.3.5 Garanties et SLA

Disponibilité (SLA)	99.95 %
Niveau 1 sans nécessité de diagnostic personnalisé Exemple : serveur indisponible	
Intervention (GTI)	1 heure
Rétablissement (GTR)	GTI + 2 heures
Niveau 2 avec diagnostic personnalisé Exemple : fonctionnement anormal du serveur (sauf configuration software)	
Intervention (GTI)	12 heures

1.3.6 Remboursement

Support incidents niveau 1	
Pénalité par heure de retard	5 %
Pénalité maximale par mois	100 %

Réseau	
Pénalité par heure de retard	5 %
Pénalité maximale par mois	100 %

2 CHOIX DES SOLUTIONS

2.1 Ansible

Ansible est un outil ou une plate-forme d'automatisation open source, utilisé pour les tâches informatiques telles que la gestion de la configuration, le déploiement d'application, l'orchestration interservices et l'approvisionnement. L'automatisation est cruciale de nos jours, avec des environnements informatiques trop complexes et qui doivent souvent évoluer trop rapidement pour que les administrateurs systèmes et les développeurs puissent suivre s'ils devaient tout faire manuellement.

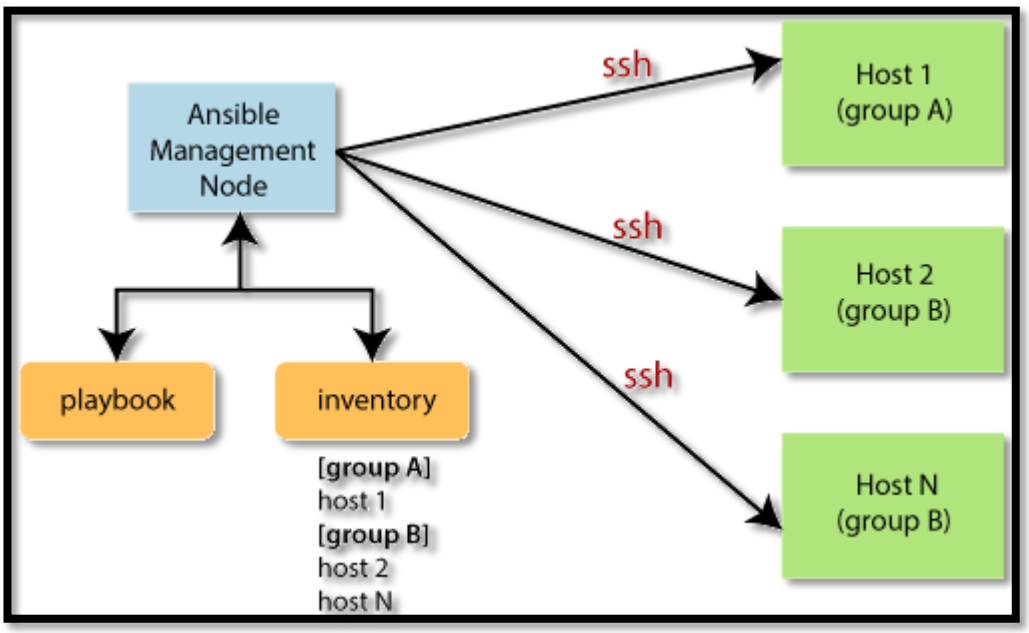
L'automatisation simplifie les tâches complexes, non seulement en rendant les tâches des développeurs plus faciles à gérer, mais en leur permettant de concentrer leur attention sur d'autres tâches qui ajoutent de la valeur à une organisation. En d'autres termes, cela libère du temps et augmente l'efficacité. Et Ansible, comme indiqué ci-dessus, se hisse rapidement au sommet du monde des outils d'automatisation.

2.1.1 Avantages d'Ansible

- **Gratuit** : Ansible est un outil open source.
- **Simple** : aucune compétence en code spécial n'est nécessaire pour utiliser les playbooks d'Ansible.
- **Puissant** : Ansible vous permet de modéliser des workflows informatiques même très complexes.
- **Flexible** : vous pouvez orchestrer tout l'environnement d'application quel que soit l'endroit où il est déployé. Vous pouvez également le personnaliser en fonction de vos besoins.
- **Sans agent** : vous n'avez pas besoin d'installer d'autres logiciels ou ports de pare-feu sur les systèmes clients que vous souhaitez automatiser. Vous n'avez pas non plus à mettre en place une structure de gestion distincte.
- **Efficace** : Parce que vous n'avez pas besoin d'installer de logiciel supplémentaire, il y a plus de place pour les ressources d'application sur votre serveur.

L'un des avantages d'Ansible par rapport à ses concurrents c'est qu'il est principalement agentless, nul besoin donc d'installer des agents sur vos différents serveurs cibles ce qui rend sa mise en place plus accessible et facile à administrer.

2.1.2 Architecture Fonctionnelle d'Ansible

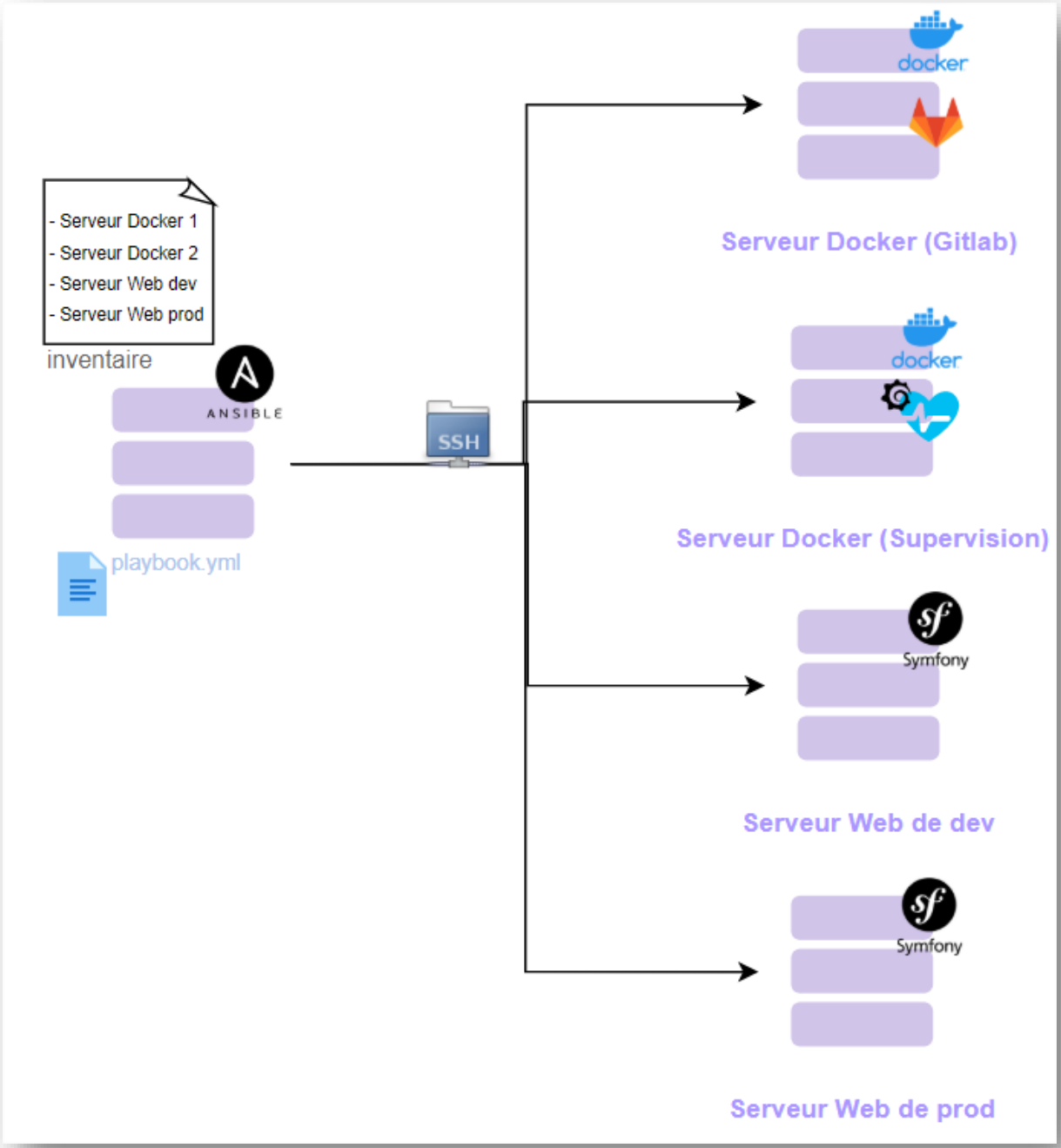


Dans Ansible, il existe deux classes de serveurs : le **nœud maître** (master) et les **nœuds esclaves** (slaves). Le nœud maître est une machine sur laquelle est installé l'outil Ansible et les nœuds esclaves sont les machines cibles sur lesquelles le nœud maitre ordonne l'exécution de tâches Ansible.

Ansible fonctionne en se connectant à vos nœuds en SSH et en y poussant de petits programmes, appelés **modules**. Ces modules sont définis dans un fichier nommé le **Playbook**. Le nœud maître, se base sur un fichier d'inventaire qui fournit la liste des hôtes sur lesquels les modules Ansible doivent être exécutés.

2.1.3 Architecture globale d'Ansible

Ansible sera donc utilisé dans la partie Plan du processus DevSecOps afin de configurer les différents serveurs de la solution Automatisator.



1. Ansible lit les informations sur les machines que vous souhaitez gérer à partir du fichier d'inventaire.
2. Ansible exécute les modules Ansible sur les machines distantes via le protocole SSH. Voici ce qui est attendu du playbook Ansible :

- Installer le moteur docker sur le serveur qui hébergera le conteneur gitlab.
- Installer le moteur docker sur le serveur qui hébergera le conteneur de supervision (Prometheus et Grafana).
- Installer et configurer le serveur web pour l'environnement de développement et de production.

2.2 Docker

Les applications Gitlab, Grafana et Prometheus ont été conteneurisées avec la solution Docker dans le but de faciliter leur déploiement et leur exportation vers d'autres infrastructures. Ces outils seront ainsi isolés du système hôte afin de permettre à vos équipes de conditionner ces outils avec toutes leurs dépendances, et de les expédier dans un package unique.

Ces images docker seront créées, exécutées et paramétrées automatiquement en lisant les instructions du fichier docker-compose, cet outil définit et exécute des applications Docker multi-conteneurs. Avec ce fichier docker-compose, vous utilisez un fichier YAML pour configurer les services de votre application. Ensuite, avec une seule commande, vous créez et démarrez tous les services à partir de votre configuration.

De ce fait, dans le cas d'une exportation des données vers une autre infrastructure, il suffit de sauvegarder les fichiers docker-compose.yaml dans un outil de sauvegarde et de les exécuter dans une machine. Cependant, le seul prérequis est que la machine cible qui hébergera ces images doit au préalable posséder le moteur docker, cette étape a été automatisée depuis un playbook Ansible.

2.3 Gitlab

2.3.1 Pourquoi Gitlab et non Jenkins

Jenkins est l'un des outils d'automatisation de génération open source et de développement de CI/CD le plus populaire au monde. Il tire sa flexibilité incroyable de l'incorporation de capacités de ses centaines de plugins disponibles, lui permettant de prendre en charge la construction, le déploiement et l'automatisation de tout projet.

Jenkins intègre nativement le serveur CI/CD Jenkins, Kubernetes, Helm et d'autres outils pour offrir un pipeline CI/CD avec les meilleures pratiques intégrées, telles que l'utilisation de GitOps pour gérer les environnements. En revanche, GitLab fournit déjà plus que ce que Jenkins espère faire évoluer, en fournissant une application unique entièrement intégrée pour l'ensemble du cycle de vie DevOps. Plus que les objectifs de Jenkins, GitLab fournit également la planification, le SCM (git), l'empaquetage, la publication, la configuration et la surveillance (en plus du CI/CD sur lequel Jenkins se concentre).

De plus L'extension de la fonctionnalité native de Jenkins se fait via des plugins. Les plugins sont coûteux à maintenir, à sécuriser et à mettre à niveau. En revanche, GitLab est un noyau ouvert et n'importe qui peut apporter des modifications directement à la base du code, qui une fois fusionnée serait automatiquement testée et maintenue à chaque modification.

De plus, Gitlab est également un réel outil de collaboration et de gestions de projets (agiles), permettant ainsi de se passer de l'intégration d'autres outils de collaboration (ex : Jira).

La plateforme open source Gitlab a été retenue puisque celle-ci propose ainsi de nombreuses fonctionnalités permettant de centraliser, sécuriser automatiser plusieurs étapes du processus DevSecOps. Cet outil permet en plus une réelle gestion de vos projets informatique. Voici dans le chapitre quelques-unes des fonctionnalités proposées par Gitlab.

2.3.2 Quelques fonctionnalités du Gitlab

Ci-dessous quelques fonctionnalités qui justifient le choix de la solution Gitlab et son adaptation au processus Code,Build,Test et Deploy de la chaine DevSecOps.

2.3.2.1 Événements d'audit

2.3.2.1.1 Journaux d'audit

Pour maintenir l'intégrité de votre code, GitLab donne aux administrateurs la possibilité de visualiser toutes les modifications apportées au sein du serveur GitLab dans un système de journal d'audit avancé, afin que vous puissiez contrôler, analyser et suivre chaque changement.

2.3.2.2 Gestion de la conformité

2.3.2.2.1 Gestion des informations d'identification

Les administrateurs GitLab et les propriétaires du groupe sont responsables de la sécurité globale de leur instance et de leurs groupes. Gardez une trace de toutes les informations d'identification PAT et SSH qui peuvent être utilisées pour accéder à votre environnement. Voyez quand les informations d'identification expirent et gérez les stratégies de rotation.

2.3.2.2.2 Tableau de bord de conformité

La gestion de la conformité au sein de GitLab est plus facile avec une vue agrégée de toutes les activités du projet. Affichez l'état de conformité de votre groupe de manière simple et rapide. Repérez facilement les projets non conformes et prenez des mesures en connaissance de cause pour résoudre tout problème.

2.3.2.3 Authentification et autorisation

2.3.2.3.1 Rôles d'utilisateur granulaires et autorisations flexibles

Gérez l'accès et les autorisations avec cinq rôles et paramètres utilisateurs différents pour les utilisateurs externes. Définissez les autorisations en fonction du rôle des utilisateurs, plutôt que d'accéder en lecture ou en écriture à un référentiel. Ne partagez pas le code source avec des personnes qui n'ont besoin que d'accéder au suivi des problèmes.

2.3.2.3.2 Accès au serveur

Vous avez un contrôle total sur le serveur / l'instance, vous pouvez donc installer des logiciels supplémentaires (détection d'intrusion, surveillance des performances, etc.) et afficher les fichiers journaux sur le serveur lui-même. Le système de journal avancé de GitLab signifie que tout est enregistré et vous offre un accès facile à une multitude d'informations sur les fichiers journaux.

2.3.2.3.3 Tags et branches protégés

Autorisations granulaires pour les tags et les branches que vous souhaitez protéger.

2.3.2.3.4 Intégration AD / LDAP

Synchronisez les groupes, gérez les clés SSH, gérez les autorisations, l'authentification et plus encore. Vous pouvez gérer une instance GitLab entière via l'intégration LDAP / AD. Liez plusieurs serveurs LDAP à GitLab pour l'authentification et l'autorisation et utilisez des groupes LDAP pour créer ou supprimer des administrateurs de votre instance GitLab. GitLab peut s'intégrer aux mécanismes d'authentification et d'autorisation (LDAP / AD), à plusieurs services tiers, CI/CD et à d'autres outils tels que les outils ALM, PLM, Agile et Automation.

2.3.2.3.5 Restreindre l'accès par adresse IP et Liste blanche IP

Limitez l'accès au niveau du groupe au trafic entrant adhérant à un sous-réseau d'adresse IP, en gardant votre code sécurisé.

La liste blanche IP définit des adresses de réseau IP sécurisées à partir desquelles les clients peuvent accéder au serveur de référentiel et interagir avec lui. Cela permet d'empêcher des tiers indésirables d'accéder à votre compte même s'ils ont acquis l'adresse e-mail et le mot de passe d'un membre de l'équipe.

2.3.2.3.6 Authentification à deux facteurs appliquée (2FA)

L'authentification à deux facteurs sécurise votre compte en exigeant une deuxième confirmation, en plus de votre mot de passe. Cette deuxième étape signifie que votre compte reste sécurisé même si votre mot de passe est compromis. La possibilité d'appliquer 2FA offre une sécurité supplémentaire en s'assurant que tous les utilisateurs l'utilisent.

2.3.2.4 Gestion des flux de valeur

2.3.2.4.1 Espace de travail Analytics

L'espace de travail Analytics permet d'agréger les analyses dans GitLab, afin que les utilisateurs puissent afficher les informations de plusieurs projets et groupes en un seul endroit.

2.3.2.4.2 Analyses de productivité

Productivity Analytics fournit des graphiques et des rapports pour aider les responsables de l'ingénierie à comprendre la productivité des équipes, des projets et des groupes afin qu'ils puissent découvrir les modèles et les meilleures pratiques pour améliorer la productivité globale. L'objectif initial de Productivity Analytics est le MR et le temps qu'il faut pour fusionner les MR.

2.3.2.4.3 Perspectives

Créez des graphiques alimentés par des tags pour visualiser des données telles que les problèmes créés/fermés pour une période donnée, le temps moyen de fusion des demandes de fusion et bien plus encore.

2.3.2.4.4 Analyse de révision de code

Trouvez les bottlenecks dans votre processus de révision de code en comprenant depuis combien de temps les demandes de fusion ouvertes sont en cours de révision.

2.3.2.5 Suivi du temps

Le suivi du temps dans GitLab permet à votre équipe d'ajouter des estimations et d'enregistrer le temps passé sur les problèmes et les demandes de fusion.

- Mettre à jour et suivre le temps passé sur un problème ou une demande de fusion.
- Mettre à jour et suivre le temps estimé requis sur un problème ou une demande de fusion.

2.3.2.6 Gestion des et suivit des projets

2.3.2.6.1 Listes de tâches

- Utilisez des listes de tâches dans les problèmes, les demandes de fusion et les épopées pour gérer les tâches et suivre leur achèvement à l'aide de cases à cocher.
- Attribuez des emoji dans les numéros, les demandes de fusion et les épopées, pour des communications plus expressives et pour indiquer les votes positifs et négatifs.

- Téléchargez des pièces jointes (y compris des images) aux problèmes, aux demandes de fusion et aux épopées, pour communiquer des idées au-delà du texte.
- Affichez l'activité du système pour afficher un historique des modifications apportées aux problèmes, aux demandes de fusion et aux épopées. Filtrer uniquement par les commentaires ou l'historique uniquement.
- Suivez les changements de titre dans l'activité système des problèmes, des demandes de fusion et des épopées.
- Lorsqu'un utilisateur est mentionné dans ou affecté à un problème ou une demande de fusion, il sera inclus dans les Todos de l'utilisateur, ce qui rend le flux de travail de développement plus rapide et plus facile à suivre.
- Annulez n'importe quel commit ou une seule demande de fusion de l'interface utilisateur de GitLab, en cliquant sur un bouton.

2.3.2.6.2 Verrouiller la discussion

Verrouillez la discussion continue dans un problème ou une demande de fusion en tant que rôle principal ou supérieur, pour éviter tout abus, spam ou collaboration improductive.

2.3.2.6.3 Notifications personnalisées

Soyez avertis par e-mail, Slack ou autre chaque fois qu'un changement ou une demande de fusion est modifié.

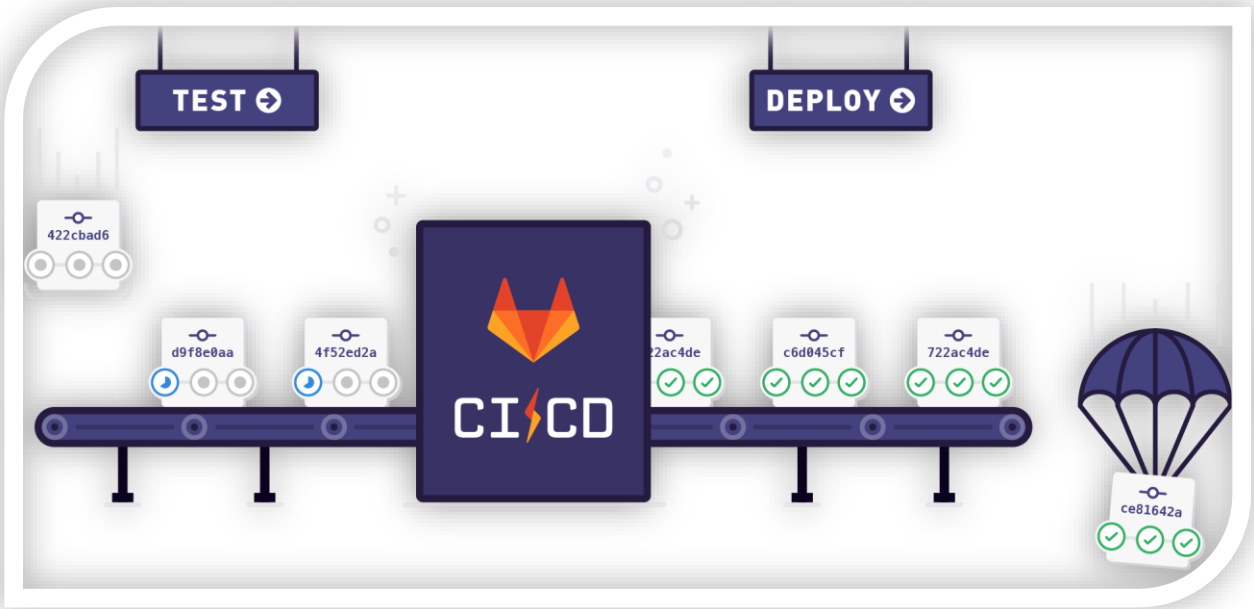
2.3.2.6.4 Documentation de projet basée sur le wiki

Un système de documentation distinct appelé Wiki est intégré directement dans chaque projet GitLab. Chaque wiki est un référentiel Git distinct.

2.3.2.7 Pipeline DevOps

Capable d'établir une visibilité sur le pipeline DevOps de bout en bout afin que toute l'équipe soit au courant de l'état du pipeline et puisse contribuer au succès global. Les caractéristiques spécifiques comprennent : la visibilité sur l'état du pipeline.

2.3.2.8 Intégration continue (CI)



GitLab a intégré l'intégration continue / livraison ne continue, gratuitement, pas besoin de l'installer séparément. Utilisez-le pour créer, tester et déployer votre site Web (pages GitLab) ou votre application Web. Les résultats du travail sont affichés sur les demandes de fusion pour un accès facile.

2.3.2.8.1 Tableau de bord des pipelines CI/CD

Visualisez l'historique et l'état actuel des pipelines à travers les projets et les groupes dans un tableau de bord unique qui peut être personnalisé pour chaque utilisateur.

2.3.2.8.2 Déclenchement planifié de pipelines

Vous pouvez faire fonctionner vos pipelines selon un calendrier dans un environnement de type cron.

2.3.2.8.3 Exécuter des travaux CI/CD sous Windows

GitLab Runner prend en charge Windows et peut exécuter des tâches en mode natif sur cette plate-forme. Vous pouvez créer, tester et déployer automatiquement des projets Windows en utilisant PowerShell ou des fichiers batch.

2.3.2.8.4 Test de sécurité des applications statiques

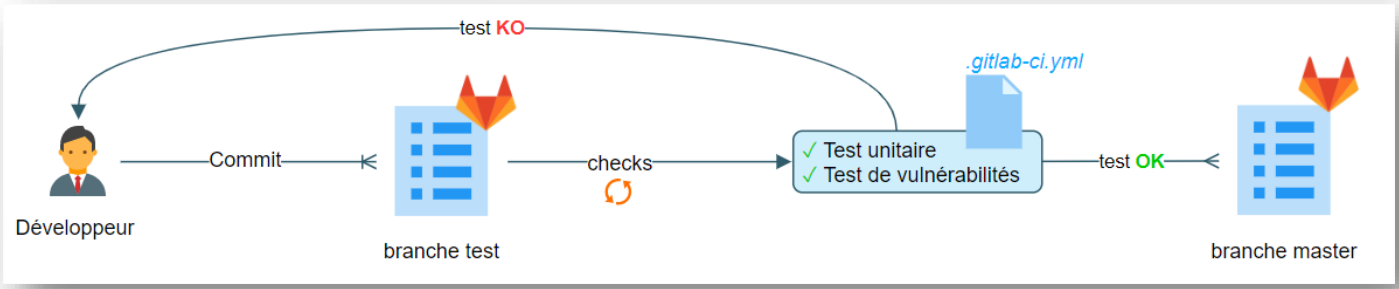
GitLab permet d'exécuter facilement des tests de sécurité des applications statiques (SAST) dans les pipelines CI/CD, vérification du code source vulnérable ou des bugs de sécurité bien connus dans les bibliothèques incluses par l'application. Les résultats sont ensuite affichés dans la demande de fusion et dans la vue Pipeline.

2.3.2.9 Conclusion

Gitlab est aujourd'hui une solution adaptée aux entreprises. En effet, les projets de développement sont bien souvent effectués en mode projet au sein des entreprises. Les contributeurs à ces projets peuvent être internes ou externes aux entreprises.

2.3.3 Architecture Gitlab

2.3.3.1 Architecture globale



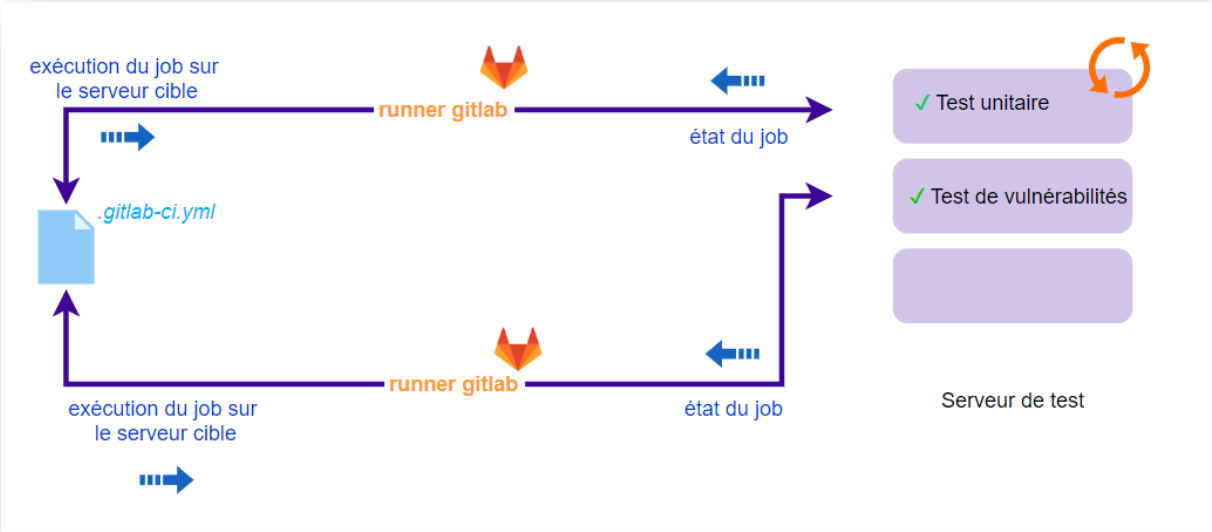
1) Depuis son poste, un développeur a la possibilité d'accéder et modifier à la branche de test.
2) Les développeurs poussent leurs changements du code afin de versionner leurs codes (commit).
3) À la suite du commit, un pipeline se déclenche intégrant des tests unitaires et de tests de vulnérabilités.
4) Dans le cas où les tests sont réussis les modifications sont également déployées dans la branche de master, dans le cas contraire les modifications sont annulées, aucune modification n'est soumise à la branche de master et le développeur doit revoir son code afin de régler son problème.

2.3.3.2 Architecture d'intégration continue

GitLab CI/CD est un outil puissant intégré à GitLab qui vous permet d'appliquer toutes les méthodes continues (intégration continue et déploiement) à votre logiciel sans aucune application ou intégration tierce.

Le code utilisé dans l'environnement est stocké dans le référentiel Git dans GitLab. Les développeurs poussent les changements de code. Pour chaque poussée vers le référentiel, vous pouvez créer un ensemble de scripts pour générer et tester votre application automatiquement, ce qui réduit les risques d'introduction d'erreurs dans votre application.

Cette pratique est connue sous le nom d'intégration continue, pour chaque changement soumis à une application (même aux branches de développement) des tâches sont exécutées automatiquement en continu, garantissant que les changements introduits passent tous les tests, directives et normes de conformité au code que vous avez établies pour votre application.



Les pipelines GitLab CI/CD sont configurés à l'aide d'un fichier YAML appelé *".gitlab-ci.yml"* dans chaque projet. Ce fichier est exécuté lors de chaque commit et définit la structure et l'ordre des pipelines et détermine :

- Quoi exécuter en utilisant GitLab Runner.
- Quelles décisions prendre quand des conditions spécifiques sont rencontrées. Par exemple, lorsqu'un processus réussit ou échoue.

Le runner GitLab est un agent installé sur un serveur cible et utilisé pour exécuter vos jobs et renvoyer les résultats à GitLab. Il est utilisé conjointement avec GitLab CI, le service d'intégration continue open source inclus avec GitLab qui coordonne les travaux. Idéalement, le runner GitLab ne doit pas être installé sur la même machine que GitLab, nous avons donc décidé de l'intégrer dans une machine de test destiné aux développeurs.

Ci-dessous plus de détails sur les jobs à exécuter :

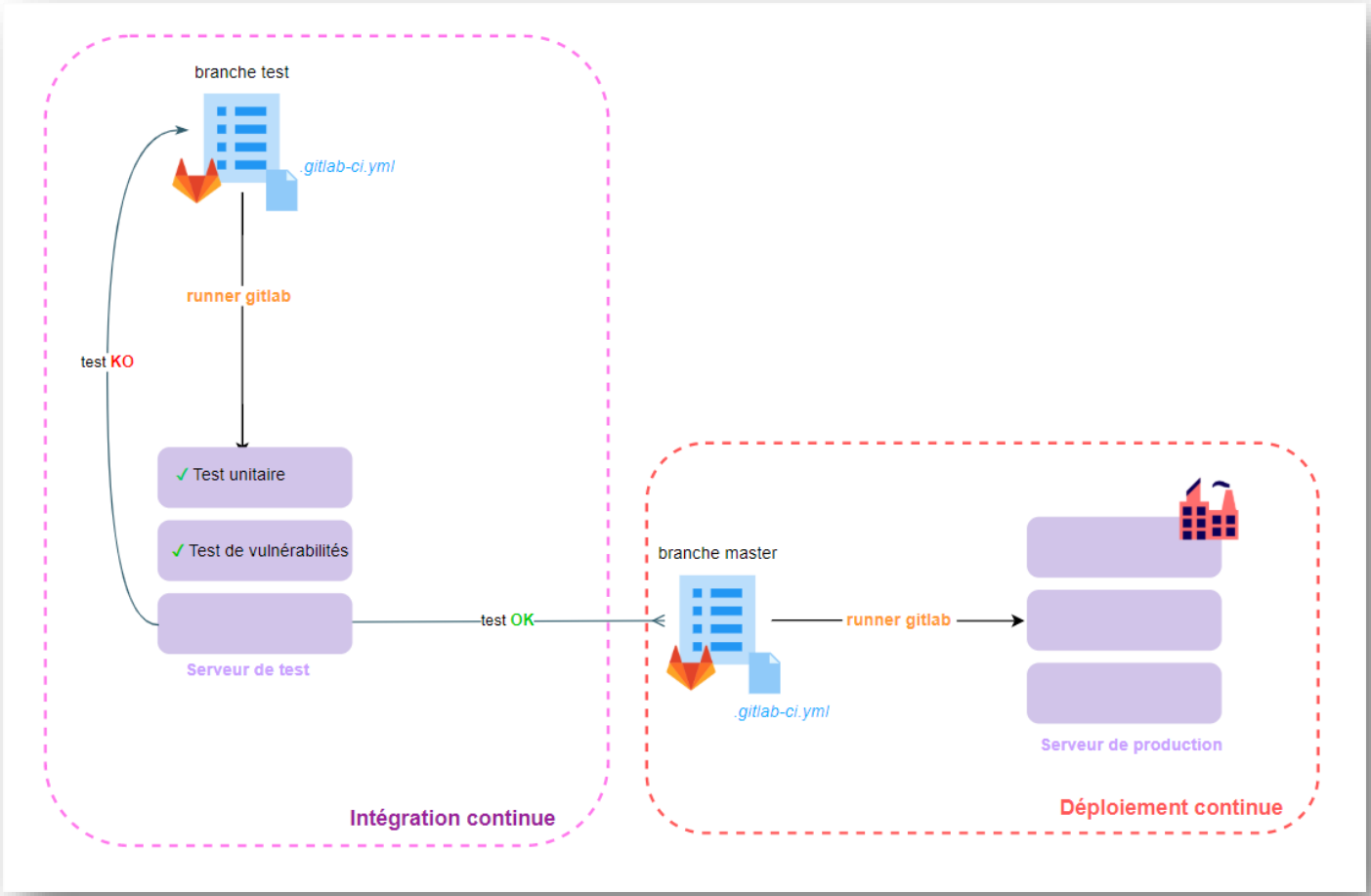
- **Test unitaire :** L'application client est codée sous le framework php symfony et l'outil standard de facto pour les tests en PHP est PHPUnit. Ce framework est l'outil open source des tests unitaires dédié au langage de programmation PHP. Il est basé sur l'idée que les développeurs devraient être capables de trouver rapidement des erreurs dans leur code nouvellement validé et d'affirmer qu'aucune régression de code ne s'est produite dans d'autres parties de la base de code. Il utilise des assertions pour vérifier que le comportement du composant spécifique

- ou "unité" - testé se comporte comme prévu. Ses assertions seront spécifiées par l'équipe de développement.
- Test de vulnérabilités : Un point qu'il semble important de soulever est là différence entre « Static Application Security Testing » (SAST) et « Dynamic Application Security Testing » (DAST) :
    - SATS : solution de sécurité qui permet de vérifier que le code ne comporte pas de problème qui pourrait inclure une faille de sécurité, son travail est donc d'analyser le code fourni et d'en renvoyer un résultat.
    - DATS : entre en jeu après l'intervention du SATS, en effet, une fois que l'application ou le site web est déployé, il peut être sujet à diverses attaques venant de l'extérieur, le rôle du DATS est donc d'empêcher toute intervention malveillante venant de l'extérieur.

  Lors de notre intégration continue, seul l'aspect SATS sera utilisé car nous n'avons que du code à analyser. L'outil de scan Gemnasium sera utilisé dans le cadre du projet Automatisator. Cet outil offre une base de données plus grande et des algorithmes avancés, et trouve plus de vulnérabilités avec moins de faux positifs. Ainsi le code des développeurs sera protégé et détectera les vulnérabilités des logiciels open source avant que les attaquants puissent exposer une organisation à des menaces telles que les injections de logiciels malveillants, et les violations de données.

2.3.3.3 Architecture de déploiement continu

Le déploiement continu est également une étape supplémentaire au-delà de l'intégration continue, similaire à la livraison continue. La différence est qu'au lieu de déployer votre application manuellement, vous la configurez pour qu'elle soit déployée automatiquement. Il ne nécessite aucune intervention humaine pour que votre application soit déployée.



Il existe deux types de branches :

- **Branche test :** accessible et géré par les équipes de développement et les équipes de sécurités. On y trouve la dernière version de l'application mise en environnement de test.
- **Branche master :** accessible et géré par les équipes d'opérations. On y trouve la dernière version de l'application mise en environnement de production.

Une fois les tests validés dans la partie d'intégration continue, les changements sont ensuite déployées également dans la branche master et dans le serveur de production à l'aide de l'agent runner gitlab installé dans le serveur de test et du fichier *".gitlab-ci.yml"* intégré dans la branche master.

2.3.4 Image Docker Gitlab

La plateforme Gitlab sera déployée via une image docker.

2.3.4.1 Les volumes

Trois volumes ont été créés :

Emplacement hôte	Emplacement dans le conteneur	Usage
`/srv/gitlab/data`	`/var/opt/gitlab`	Pour stocker les données d'application
`/srv/gitlab/logs`	`/var/log/gitlab`	Pour stocker les journaux
`/srv/gitlab/config`	`/etc/gitlab`	Pour stocker les fichiers de configuration Gitlab

2.3.4.2 Mappage des ports

Il a été convenu de mapper les ports comme suit :

Port conteneur	Port hôte	Explication
443	443	Accès à gitlab via le port https
80	80	Accès à gitlab via le port http
22	22	Port ssh

2.4 Supervision

La dernière étape du processus DevSecOps est la partie Monitor. Cette étape est essentielle, car elle fournit des informations cruciales qui vous aideront à garantir la disponibilité du service et des performances optimales, en identifiant les problèmes spécifiques de livraisons et de comprendre l'impact sur les utilisateurs finaux.

La norme ISO 7498/4 définit les principales fonctions que doivent implanter les systèmes de monitoring. Ci-dessous des fonctions qui ont été défini par l'ISO 7498/4.

2.4.1 ISO 7498/4

2.4.1.1 Gestion des performances

Elle doit pouvoir évaluer les performances des ressources du système et leur efficacité. Elle comprend les procédures de collecte de données et de statistiques. Elle doit aboutir à l'établissement de tableaux de bord. Les informations recueillies doivent aussi permettre de planifier des évolutions.

2.4.1.2 Gestion des configurations

La gestion de configuration permet d'identifier, de paramétrer et de contrôler les différents équipements. Les procédures requises pour gérer une configuration sont :

- La collecte d'informations
- Le contrôle d'état
- La sauvegarde historique de configurations de l'état du système.

2.4.1.3 Gestion des anomalies

La gestion des fautes permet la détection, la localisation et la correction d'anomalies passagères ou persistantes. Elle doit également permettre le rétablissement du service à une situation normale.

2.4.1.4 Gestion de la sécurité

La gestion de la sécurité contrôle l'accès aux ressources en fonction des politiques de droits d'utilisations établies. Elle veille à ce que les utilisateurs non autorisés ne puissent accéder à certaines ressources protégées.

2.4.2 Le but de la solution choisie

Le but de la solution est donc de repérer et de mettre en place une solution de monitoring optimale pour les serveurs et de pouvoir détecter et interpréter facilement les causes et origines des problèmes rencontrés afin de les résoudre le plus rapidement possible.

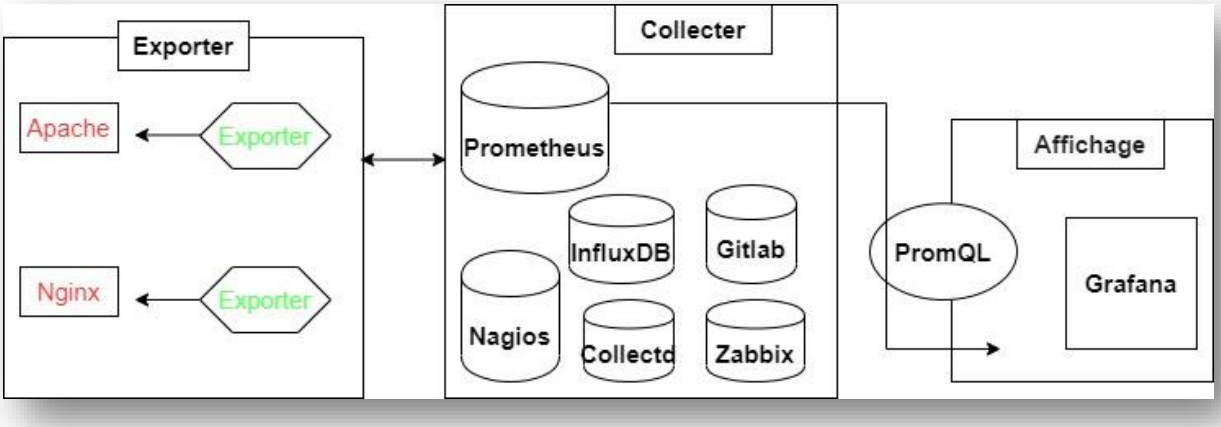
Nous proposons la mise en place d'un outil de monitoring avec l'aide de la norme ISO 7498 qui assure les fonctionnalités suivantes :

- Gestion des droits utilisateurs
- Surveiller les différents services (Docker, Apache, ssh etc …)
- Déclencher des alertes lors de détections des pannes
- Vérifier la disponibilité des équipements en surveillant les ressources et les performances systèmes (CPU, Espace disque, RAM, bande passante etc.)

- Générer des graphes et des rapports
- Gestion du listing des destinataires des alertes ainsi et évaluation de la pertinence de ces dernières afin d'éviter le spam
- Disposer d'une interface graphique compréhensible facilitant l'interaction entre l'utilisateur et le logiciel.
- Interface d'administration web protégée par un système d'authentification

2.4.3 Architecture principale des composants

Voici le schéma de l'infrastructure que nous allons utiliser pour réaliser ce projet :



2.4.4 Prometheus

Prometheus sera l'outil de supervision choisi en tant que collecter (collecteur). Il sera le cœur de notre système. Ce serveur collecte les métriques de plusieurs nœuds et les stocke localement. Le serveur Prometheus fonctionne sur le principe du scraping, c'est-à-dire en invoquant les points de terminaison des métriques des différents nœuds qu'il est configuré pour surveiller. Il recueille ces métriques à intervalles réguliers et les stocke localement. Ces métriques sont extraites de nœuds qui exécutent des exporters (exportateurs) spécifiques, ici sont utilisés les exportateurs de type Apache et Nginx.

2.4.5 Outil de visualisation : Grafana

Les modèles d'architecture d'applications actuelles entrainent l'utilisation de plusieurs serveurs, de faire du déploiement rapide et évolutif. Il est nécessaire d'avoir une vision détaillée de la supervision sur tous ces serveurs et toutes ces applications, mais aussi une vision globale de l'état des services. Il existe de nombreux systèmes de supervision, propriétaires ou open source.

Pour la réalisation de ce projet, il était au préalable instruit d'utiliser des outils Open source, dont le choix de Grafana contrairement à kibana qui permet aussi de créer des Dashboard. Cependant nous avons jugé nécessaire de choisir Grafana en tenant compte de notre architecture de monitoring et de sa performance avec l'outil de collecte des métriques Prometheus.

2.4.6 Quoi superviser ?

2.4.6.1 Ressources

L'usage du cpu, ram, nombre de processus (était des processus : actif, en attente, erreur …) et l'usage de disques sont collectés depuis tous les serveurs de la solution Automatisator. Ils permettront de gérer la capacité planifiée et d'anticiper les incidents relatifs à la plateforme.

2.4.6.2 Ports et Services

Les services et ports importants, tel que le daemon docker ou le port ssh seront supervisés également par Prometheus.

2.4.6.3 Serveur Web

Enfin, l'objectif principal de cette solution de monitoring est de superviser les serveurs web est de savoir s'ils répondent bien aux requêtes des utilisateurs, dans un temps satisfaisant mais également d'observer quelle est l'utilisation des ressources système, pour permettre le suivi, et un diagnostic en cas d'incident.

2.4.6.3.1 Gestion des requêtes

Elle permet de déterminer avec précision :

- Le temps de réponse
- Le temps entre la requête utilisateur, et la réponse du serveur : la latence des requêtes
- Le taux et le type d'erreur HTTP (page non existante 404, droit d'accès 403, page déplacée 301/302, erreur serveur 500
- Mesurer le débit du temps de latence (réponse lente) par rapport à la moyenne et maximum

2.4.6.3.2 Les logs

Afin de gérer efficacement un serveur Web, il est nécessaire d'obtenir des commentaires sur l'activité et les performances du serveur ainsi que sur tout problème pouvant survenir. Le serveur HTTP Apache ou Nginx propose des fonctionnalités de journalisation souples et très complètes dont :

- Les erreurs de log
- Les erreurs d'accès

Ces informations permettent de faire une comparaison des données pour savoir que le trafic est normal, élevé ou faible.

2.4.6.3.3 Taille de stockage

Dans le monitoring d'un serveur web, l'une des caractéristiques les plus importantes à surveiller est la quantité de RAM car :

- Elle permet de comprendre le nombre d'utilisateurs supportés par la RAM du serveur
- D'ajuster le nombre de processus
- Le nombre de RAM consommée par processus

2.4.6.3.4 Les utilisateurs actif et inactif (busy and idle workers)

Monitorer le nombre d'utilisateurs actif et inactif permet de déterminer les soucis de configuration du serveur plus vite. Le nombre d'utilisateurs inactifs approche ou atteint zéro, et les demandes peuvent être mises en file d'attente en attendant d'être traitée par un utilisateur disponible. Une demande en file attente, attend que les demandes les plus anciennes soient traitées et ce qui permet de déterminer que le temps de réponse le plus court du serveur.

Pour éviter ce type de problème, il est conseillé d'adapter le nombre de connexions simultanées par utilisateur en :

- Augmentant la RAM
- Mettre à niveau le serveur pour équilibrer les charges

2.4.6.3.5 Le scoreboard status du serveur

Il s'occupe des demandes reçues et indique dans un tableau de bord tout en gardant une trace de chaque connexion :

- Connexion entrante
- Connexion en attente
- Connexion complète

2.4.6.3.6 Les métriques à collecter

L'explication des métriques et les seuils spécifiques à monitorer à partir du serveur

- Request per second
- Bytes per second
- Bytes per request
- Uptime
- CPU Usage

- Network Bandwidth
- Disk Usage
- Load

2.4.7 Image Docker Prometheus

2.4.7.1 Les volumes

Emplacement hôte	Emplacement dans le conteneur	Usage
`./grafana/grafana.db`	`/etc/grafana/grafana.ini`	Configuration par défaut
`./grafana/provisioning/dashboards`	`/etc/grafana/provisioning/dashboards`	Conservation les tableaux de bord
`./grafana/provisioning/datasources`	`/etc/grafana/provisioning/datasources`	Configuration par défaut Conservation les tableaux de bord Conservation les sources de données
`./prometheus/prometheus.yml`	`/etc/prometheus/prometheus.yml`	Pour stocker les métriques collectées et configurer les targets

2.4.7.2 Mappage des ports

Port conteneur Port host Explication

Port conteneur	Port hôte	Explication
9000	9000	Port pour accéder à l'interface graphique de prometheus
3000	3000	Port pour accéder à l'interface graphique de Grafana

9100	9100	Port du Node Exporter
9117	9117	Port d'Apache Exporter (à ouvrir vous utilisez Apache)
9113	9113	Port d'Nginx Exporter numéro 1 (à ouvrir vous utilisez Nginx)
9114	9114	Port d'Nginx Exporter numéro 2 (à ouvrir vous utilisez Nginx)