



Mémoire de projet Fin d' Année En

Génie Informatique

Présenté par :

AL UOLANTI ABDELLAH

**La création d'une infrastructure pour héberger des
produits SaaS**

Stage effectué à : PAPERLESS



Encadré par :

M. Omar BEN HAMMOU

M. Mohamed CHRAYAH

Encadrant Professionnel

Encadrant universitaire

Année universitaire: 2024-2025

Dédicaces

Tout d'abord, je tiens à remercier Dieu de m'avoir donné la force et le courage nécessaires pour mener à bien ce modeste travail.

Je tiens également à exprimer ma gratitude envers mes encadrants et collègues pour leur précieuse aide et leurs conseils tout au long de ce projet.

Leurs encouragements et leurs contributions ont été essentiels à la réussite de ce travail.

Remerciements

Je tiens à remercier chaleureusement toutes les personnes qui ont contribué de près ou de loin à la réussite de ce projet.

Je souhaite également exprimer toute ma gratitude à Monsieur **Omar BEN HAMMOU** , mon encadrant professionnel. Son accompagnement technique, sa disponibilité et ses encouragements constants ont été des éléments clés pour surmonter les défis que j'ai rencontrés. Sa contribution a été déterminante pour mener à bien ce travail.

Je souhaite également adresser mes plus sincères remerciements à Monsieur **Mohamed CHRAYAH** mon encadrant, ainsi qu'à tous mes enseignants pour leur accompagnement tout au long de mon parcours académique.

Je suis particulièrement reconnaissante envers toute l'équipe de l'entreprise pour leur esprit d'équipe, leur collaboration, et leur soutien durant mon stage.

Enfin, je tiens à adresser mes remerciements les plus sincères à tous ceux qui, d'une manière ou d'une autre, ont apporté leur aide durant ce projet. Votre soutien, qu'il soit technique, moral ou intellectuel, a été indispensable, et je vous en suis profondément reconnaissant.

Résumé

Pendant mon stage de fin d'année chez Paperless IT Services, j'ai travaillé sur un projet ambitieux visant à créer une infrastructure pour héberger des produits SaaS en utilisant Docker, Ansible, et Python. Mon rôle consistait à concevoir, déployer, et automatiser cette infrastructure tout en garantissant une gestion optimisée des serveurs sous Linux et Windows.

Responsabilités :

- Conception et mise en place de l'architecture SaaS : Développement d'une infrastructure scalable et flexible pour l'hébergement de produits SaaS.
- Conteneurisation des applications avec Docker : Configuration des conteneurs Docker pour les services tels que Odoo et PostgreSQL, tout en gérant le cycle de vie des conteneurs à l'aide de Docker Compose.
- Automatisation des tâches d'infrastructure avec Ansible : Création et exécution de playbooks Ansible pour automatiser le déploiement des services, assurer la cohérence des configurations, et faciliter la gestion des serveurs.
- Développement de scripts Python : Automatisation des processus répétitifs, gestion dynamique des ports pour les conteneurs, et développement d'une application web pour simplifier la gestion des instances Docker.
- Configuration de services de surveillance : Intégration de Prometheus

pour la collecte des métriques et configuration de Grafana pour la visualisation en temps réel des performances des services et des infrastructures.

- Collaboration en équipe : Travailler en étroite collaboration avec des collègues et des encadrants pour s'assurer que les objectifs du projet sont atteints de manière efficace.

Résultats :

Grâce à l'automatisation mise en place avec Ansible et les scripts Python, la gestion des serveurs a été simplifiée, réduisant les tâches manuelles et les erreurs potentielles. La conteneurisation avec Docker a permis un déploiement rapide et efficace des applications SaaS, tandis que le service de surveillance basé sur Prometheus et Grafana a assuré une visibilité continue des performances et une gestion proactive des incidents.

Compétences acquises :

- Maîtrise de la conteneurisation et de l'orchestration avec Docker et Docker Compose.
 - Expertise dans l'automatisation des configurations et des déploiements avec Ansible.
 - Développement avancé de scripts Python pour l'automatisation et la gestion des infrastructures.
 - Expérience dans la configuration de services de surveillance avec Prometheus et Grafana.
 - Capacité à travailler en équipe multidisciplinaire et à collaborer efficacement pour la réussite du projet.
- Ce stage m'a permis de développer des compétences techniques essentielles tout en m'offrant l'opportunité de contribuer à un projet complexe et enrichissant.

Abstract

During my internship at Paperless IT Services, I worked on an ambitious project aimed at creating a robust infrastructure to host SaaS products using Docker, Ansible, and Python. My role involved designing, deploying, and automating this infrastructure while ensuring optimized server management on both Linux and Windows platforms.

Responsibilities:

- Design and implementation of the SaaS architecture: Developed a scalable and flexible infrastructure for hosting SaaS products.
- Containerization of applications with Docker: Configured Docker containers for services such as Odoo and PostgreSQL, managing the container lifecycle using Docker Compose.
- Automation of infrastructure tasks with Ansible: Created and executed Ansible playbooks to automate service deployment, ensure configuration consistency, and facilitate server management.
- Development of Python scripts: Automated repetitive processes, managed dynamic port allocation for containers, and developed a web application to simplify the management of Docker instances.
- Configuration of monitoring services: Integrated Prometheus for metrics collection and configured Grafana for real-time performance visualization of services and infrastructure.

- Team collaboration: Worked closely with colleagues and supervisors to ensure project objectives were achieved effectively.

Results:

Through the automation implemented with Ansible and Python scripts, server management was simplified, reducing manual tasks and potential errors. Containerization with Docker enabled fast and efficient deployment of SaaS applications, while the monitoring service based on Prometheus and Grafana provided continuous visibility into performance and proactive incident management.

Skills Acquired:

- Mastery of containerization and orchestration with Docker and Docker Compose.
 - Expertise in automating configurations and deployments with Ansible.
 - Advanced development of Python scripts for infrastructure automation and management.
 - Experience in configuring monitoring services with Prometheus and Grafana.
 - Ability to work in a multidisciplinary team and collaborate effectively for project success.
- This internship allowed me to develop essential technical skills while giving me the opportunity to contribute to a complex and rewarding project.

Liste des tableaux

Tableau 1: Listes des acronymes	11
--	-----------

Liste des figures

Figure 1: Références de l'organisme d'accueil.....	17
Figure 2: illustration Docker.....	21
Figure 3: illustration Ansible.....	21
Figure 4: Logo de Python	22
Figure 5: illustration Pometheus et Grafana	22
Figure 6 : Le Schéma de workflow de notre système	26
Figure 7: Diagramme de cas d'utilisation	30
Figure 8 : Diagramme d'Activité	31
Figure 9 : Configuration Smtip	32
Figure 10 : Installation package python	34
Figure 11 : Docker Desktop	35
Figure 12 : docker info	36
Figure 13 : Installation de docker-compose	36
Figure 14 : dockerfile de l'image Odoo.....	37
Figure 15 : Fichier de docker-compose du Container Odoo	38
Figure 16 : dockerfile de l'image Odoo	38
Figure 17 : fichier docker-compose du conteneurs du Production	39
Figure 18 : docker Network	40
Figure 19 : fichier ansible-playbook du Nginx.....	41
Figure 20 : fichier ansible-playbook du Odoo	41
Figure 21 : le script python pour Nginx.....	42
Figure 22 : le script python pour Odoo	43
Figure 23: Schéma de système du Supervision	43
Figure 24: l'interface Prometheus (DataSource)	44
Figure 25: Dashboard Grafana Illustrant les Performances de l'Hôte via Node Exporter	44
Figure 26: Visualisation des Performances des Conteneurs Docker dans	

Grafana.....	45
Figure 27: Alerte configuré	45
Figure 28: Notification d'Alerte via Gmail.....	45
Figure 29: Checkout page	47
Figure 30: L’instance Odoo sur le port 8072	47
Figure 31: Liste des Conteneurs Actifs sur le serveur ubuntu.....	48
Figure 32 : Les Conteneurs Actifs sur le serveur Windows.....	49

Liste des acronymes

Acronyme	Désignation
API	Application Programming Interface
SMTP	Simple Mail Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
CPU	Central Processing Unit
DB	Database
GUI	Graphical User Interface
SaaS	Software As a Service

Table des matières

Dédicace...	2
Remerciements...	3
Résumé	4
Abstract	6
Liste des Figures	9
Liste des Acronymes.....	11
CHAPITRE 1. Contexte Général du projet.....	15
I. Présentation de l'entreprise.....	16
1. Fiche technique	16
2. Secteur d'activité.....	17
3. Références	17
II. Cahier de charges	17
1. Contexte de projet	17
2. Objectifs de Projet.....	18
3. Périmètre de Projet.....	18
4. Spécifications techniques.....	19
5. Planning et livrables.....	20
6. Contraintes.....	20
7. Risques et plans de mitigation	21
III. Présentation des Technologies Utilisées.....	21
1. Docker.....	21
2. Ansible.....	22
3. Prometheus et Grafana	22
CHAPITRE 2. Spécification des exigences.....	23
I. Analyse du fonctionnement actuel.....	24
1. Gestion manuelle des applications et des serveurs	24
2. Complexité de l'infrastructure hybride	24
3. Limitations dans la gestion des conteneurs	24
4. Inefficacités dans le déploiement des applications	25

5. Manque de supervision centralisée des services	25
II. Recueil et spécifications des besoins	25
1. Besoins fonctionnels	25
2. Besoins non fonctionnels	26
3. Identification des problèmes à résoudre	27
4. Guide pour le développement et la mise en œuvre	27
Chapitre 3 Conception et Modélisation	28
I. Architecture du Système	29
1. Analyse du système	30
II. Automatisation et Orchestration.....	31
III. Gestion Dynamique des Ports	31
IV. Monitoring.....	31
Chapitre 4 Réalisation et Mise en oeuvre.....	33
I. Déploiement de l'Infrastructure	34
1. Configuration de l'Infrastructure Serveur	34
1.1 Détails sur la configuration des serveurs (Linux et Windows)	34
1.2 Installation des prérequis nécessaires	34
2. Mise en Place de l'Environnement de Conteneurisation	35
2.1 Installation et configuration de Docker	35
2.2 Déploiement des images Docker	36
3. Configuration du Réseau et des Services	39
II. Automatisation et Gestion des Ressources	40
1. Automatisation avec Ansible	40
2. Scripting Python pour la Gestion des Ports et des Déploiements.....	41
III. Supervision et Suivi des Services	43
1. Mise en Place de la Supervision avec Prometheus et Grafana.....	43
2. Collecte et Analyse des Données	46
IV. Tests de Fonctionnement et d'Optimisation	46
1. Tests de Fonctionnement des Services	46
2. Optimisation des Ressources.....	48
Références.....	51

Introduction générale

Dans un environnement technologique en constante évolution, la gestion de l'infrastructure informatique devient de plus en plus complexe. Les entreprises modernes recherchent des solutions efficaces pour héberger et déployer leurs applications, en mettant l'accent sur l'automatisation, la flexibilité et la scalabilité. Le modèle SaaS (Software as a Service) s'est imposé comme une solution incontournable pour répondre à ces exigences, en permettant aux organisations d'accéder à des logiciels via le cloud, sans avoir à gérer directement l'infrastructure sous-jacente.

Mon stage de fin d'année chez Paperless IT Services m'a permis de plonger au cœur de ces enjeux, en travaillant sur la création d'une infrastructure destinée à héberger des produits SaaS. Ce projet se base sur l'utilisation de technologies avancées telles que Docker pour la conteneurisation, Ansible pour l'automatisation, et Python pour le développement de scripts et d'applications. L'objectif était de concevoir une infrastructure optimisée, capable de déployer des services de manière rapide et fiable, tout en simplifiant la gestion des serveurs.

Ce rapport a pour but de présenter l'ensemble des travaux réalisés durant ce stage, en commençant par l'étude des besoins, la mise en place de l'infrastructure, et l'automatisation des processus. J'explorerai également les défis rencontrés, les solutions apportées, ainsi que les résultats obtenus. Enfin, je partagerai les compétences acquises et les leçons tirées de cette expérience enrichissante dans le domaine de la gestion des infrastructures informatiques.

Chapitre 1 : Contexte général du projet

Ce chapitre présente une vue d'ensemble de l'entreprise et du projet sur lequel repose ce travail. Nous commencerons par une description de l'entreprise, en examinant son secteur d'activité, sa taille, sa position sur le marché, ainsi que ses objectifs stratégiques. Ensuite, nous passerons à une présentation détaillée du projet, en définissant ses principaux objectifs, les problématiques qu'il aborde, ainsi que les contraintes et les exigences auxquelles il doit répondre. Cette introduction posera les fondations pour une meilleure compréhension du contexte dans lequel le projet a été réalisé et préparera le terrain pour les développements à venir dans le rapport.

I. Présentation de l'entreprise

1. Fiche technique



- **Raison Sociale** : PAPERLESS
- **Date de Création** : 21 Octobre 2019
- **Nationalité** : Marocaine
- **Siège Social** : Av Hassan II Bureaux Nakhil 2ème ETG N°12,
Tétouan

- **Activité** : Programmation ,Conseil et autres Activités Informatique.
- **Forme Juridique** : S.A.R.L
- **Gérant** : Saad TEYAR
- **ICE** : N° 002341947000082
- **Registre Commercial** : N° 25925
- **Patente** : N°51103507
- **Identification Fiscale** : N°37747932
- **Capital** : 240000,00 MAD
- **Téléphone** : (+212) 05 39 70 00 78
- **Email** : contact@parperless.ma

2. Secteur d'activité

Il existe quatre catégories distinctes pour les services proposés :

- Consultation : Conseils d'experts, créativité, innovation, compétences analytiques, flexibilité et capacité à faire face aux défis et à la pression.
- IT Solutions : grâce à nos compétences en système et réseau informatique, nous

pouvons mettre en place des solutions adaptées à la bonne gestion de votre système d'information.

- Le développement peut inclure des applications Web, mobiles ou personnalisées. PAPERLESS vous accompagne à toutes les étapes, de l'analyse des besoins jusqu'aux quintessences du résultat.
- Externalisation : Nous mettons à votre disposition une équipe de professionnels talentueux et expérimentés entièrement dédiée à vous soutenir dans vos projets.

3. Références



Figure 1: Références de l'organisme d'accueil

II. Cahier de charges

1. Contexte du projet

Développement d'une infrastructure robuste et automatisée pour héberger des produits SaaS. Le projet s'inscrit dans le cadre de ton stage chez Paperless IT Services et vise à optimiser la gestion des serveurs sous Linux et Windows, en utilisant des technologies de conteneurisation et d'automatisation.

2. Objectifs du projet

Objectif principal : Mettre en place une infrastructure flexible et scalable capable de déployer et gérer des applications SaaS.

Sous-objectifs:

- Conteneuriser les applications SaaS, notamment Odoo, à l'aide de Docker.
- Automatiser les déploiements et configurations via Ansible.
- Développer des scripts Python pour l'automatisation des tâches répétitives.
- Intégrer un service de surveillance avec Prometheus et Grafana.

3.Périmètre de Projet

- Applications concernées : Odoo ERP, PostgreSQL.
- Environnements : Serveurs sous Linux et Windows.
- Technologies : Docker, Docker Compose, Ansible, Python, Prometheus, Grafana.

4. Spécifications techniques

- **Conteneurisation avec Docker :** Utiliser Docker pour encapsuler les applications Odoo et PostgreSQL, Créer et gérer des images Docker personnalisées pour Odoo et PostgreSQL, Déployer des conteneurs Docker sur des environnements Linux et Windows, Configurer les ports des conteneurs pour une gestion optimisée des ressources réseau.
- **Orchestration avec Docker Compose :** Utiliser Docker Compose pour organiser les conteneurs et gérer les services associés, Définir des configurations multi-conteneurs pour assurer l'intégration fluide entre Odoo et PostgreSQL, Faciliter le déploiement en définissant des configurations dans des fichiers docker-compose.yml.
- **Automatisation avec Ansible :** Créer des playbooks Ansible pour automatiser le déploiement des services Odoo et PostgreSQL, Assurer la cohérence des configurations sur les environnements Linux et Windows, Automatiser les mises à jour et les tâches de maintenance pour minimiser les interventions manuelles.
- **Développement de Scripts Python :** Automatiser les tâches répétitives, telles que la gestion dynamique des ports des conteneurs Docker, avec des scripts Python

Développer une application web pour gérer et surveiller les instances Docker de manière intuitive, Intégrer des fonctionnalités de planification pour arrêter les conteneurs automatiquement après un certain délai.

- **Gestion de la Surveillance avec Prometheus et Grafana :** Configurer Prometheus pour la collecte des métriques de performance des conteneurs et des services, Créer des tableaux de bord dans Grafana pour visualiser les métriques en temps réel et surveiller l'état de l'infrastructure, Mettre en place des alertes pour notifier les administrateurs en cas de dégradation des performances ou d'incidents.
- **Sécurisation de l'Infrastructure :** Configurer des politiques de sécurité pour les accès aux conteneurs et aux serveurs, Protéger l'accès aux ressources en utilisant des pare-feux et des règles de sécurité réseau spécifiques, Assurer la conformité des configurations de sécurité à travers les différents environnements.
- **Gestion des Environnements Hybrides :** Assurer la compatibilité des déploiements Docker et Ansible sur des environnements Linux et Windows, Optimiser la performance des serveurs en appliquant des configurations spécifiques pour chaque système d'exploitation, Automatiser la gestion des environnements hybrides pour minimiser les erreurs humaines.

5.Planning et livrables

- **Phase 1: Conception de l'infrastructure et planification du projet (1 semaine)**

Analyse des besoins, conception de l'architecture pour héberger des produits SaaS, et planification détaillée des étapes du projet.

- **Phase 2: Mise en place de l'architecture Docker et développement des scripts Python (3 semaines)**

Configuration des conteneurs Docker pour Odoo et PostgreSQL, développement des scripts Python pour l'automatisation des tâches, et gestion dynamique des ports pour les conteneurs.

- **Phase 3: Automatisation des déploiements avec Ansible (1 semaine)**

Création et exécution de playbooks Ansible pour automatiser le déploiement des services et la configuration des serveurs sous Linux et Windows.

- **Phase 4: Configuration du service de surveillance avec Prometheus et Grafana (1 semaine)**

Intégration de Prometheus pour la collecte des métriques et configuration de Grafana pour la visualisation des performances en temps réel.

- **Phase 5: Tests, optimisation et documentation (2 semaines)**

Test complet de l'infrastructure, optimisation des performances, rédaction de la documentation technique, et préparation du livrable final.

Livrables :

- Architecture détaillée de l'infrastructure SaaS.
- Conteneurs Docker configurés pour Odoo et PostgreSQL, avec scripts Python d'automatisation.
- Playbooks Ansible pour déploiement automatisé.
- Service de surveillance Prometheus et Grafana opérationnel.
- Documentation complète du projet, incluant les étapes de déploiement, les configurations, et les scripts.

6. Contraintes

- L'infrastructure doit être compatible avec les serveurs sous Linux et Windows.
- L'architecture doit être capable de s'adapter à une augmentation du nombre de services et d'utilisateurs.
- Implémentation des meilleures pratiques de sécurité pour protéger les services et les données.

7. Risques et plans de mitigation

Problèmes de compatibilité des versions : Tests réguliers sur différentes versions des outils utilisés.

Difficultés liées à la scalabilité : Mise en place d'une architecture modulaire permettant une évolution progressive.

Gestion des pannes : Implémentation de solutions de backup et de récupération en cas d'incidents.

III. Présentation des Technologies Utilisées

1.Docker

Docker est une plateforme de conteneurisation qui permet de déployer des applications dans des environnements isolés. En utilisant Docker, il est possible de créer des conteneurs légers qui encapsulent les applications et leurs dépendances, facilitant ainsi leur déploiement et leur gestion. Dans le cadre de ce projet, Docker est utilisé pour conteneuriser les services SaaS comme Odoo et PostgreSQL, permettant une gestion simplifiée de leur cycle de vie.

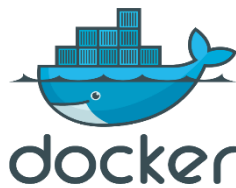


Figure 2: illustration Docker

2.Ansible :

Ansible est un outil d'automatisation qui permet de gérer la configuration et le déploiement des serveurs à grande échelle. Grâce à Ansible, il est possible de créer des "playbooks" qui définissent les étapes nécessaires pour configurer et déployer des services de manière cohérente et répétable. L'automatisation avec Ansible réduit le risque d'erreur humaine et améliore l'efficacité des opérations.

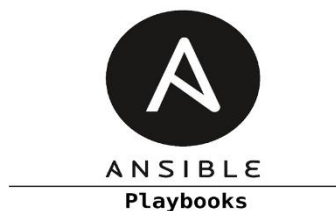


Figure 3: illustration Ansible

3. Python :

Python est utilisé dans ce projet pour développer des scripts d'automatisation qui facilitent la gestion des conteneurs et des services. Les scripts Python permettent, par exemple, de gérer dynamiquement les ports utilisés par les conteneurs, d'automatiser les tâches répétitives et de créer des outils web pour simplifier l'administration des instances SaaS.



Figure 4: Logo de Python

4. Prometheus et Grafana :

Pour surveiller et visualiser les performances de l'infrastructure, Prometheus est utilisé pour collecter des métriques et Grafana pour afficher ces données sous forme de tableaux de bord interactifs. Ce service de surveillance permet une visibilité en temps réel des performances des services SaaS et une gestion proactive des incidents.



Figure 5: illustration Prometheus et Grafana

Chapitre 2 : Spécification des exigences

*Ce chapitre comprend deux parties : **Analyse du fonctionnement actuel** et **Recueil et spécifications des besoins**. Dans la première partie, l'accent est mis sur l'évaluation détaillée du système actuel, en soulignant ses limites et inefficacités. La seconde partie se concentre sur la collecte des besoins fonctionnels et non fonctionnels, nécessaires pour développer une solution automatisée qui améliorera l'efficacité du processus.*

I. Analyse du fonctionnement actuel

L'infrastructure actuelle repose sur une gestion manuelle des applications et des serveurs dans un environnement hybride, comprenant à la fois des systèmes Linux et Windows. Cette approche présente plusieurs défis, notamment en matière de standardisation, d'efficacité, et de maintenance.

1. Gestion manuelle des applications et des serveurs :

Actuellement, les déploiements et les configurations de services sont réalisés manuellement, ce qui augmente les risques d'erreurs humaines. Cette méthode demande un investissement important en termes de temps et d'efforts, surtout lorsque le nombre de services à gérer croît. Le manque de standardisation des processus rend difficile la reproductibilité des déploiements, ce qui peut entraîner des incohérences entre les environnements.

2. Complexité de l'infrastructure hybride :

L'utilisation simultanée de serveurs Linux et Windows crée une complexité supplémentaire dans la gestion des configurations et des environnements. Chaque système possède ses propres spécificités, rendant plus difficile l'intégration et la coordination des services. Les équipes doivent posséder des compétences diversifiées pour gérer les deux types de systèmes, ce qui peut ralentir les opérations en cas de manque de ressources ou de connaissances spécialisées.

3. Limitations dans la gestion des conteneurs :

La conteneurisation est utilisée pour héberger des services tels qu'Odoo et PostgreSQL, mais la gestion manuelle des conteneurs pose des problèmes d'efficacité. Le déploiement et la maintenance des conteneurs nécessitent une intervention humaine régulière, augmentant le risque de conflits de configuration.

L'absence d'une gestion automatisée des ports complique le déploiement simultané de plusieurs instances d'une même application sur un même serveur.

4. Inefficacités dans le déploiement des applications :

Le déploiement des applications SaaS est ralenti par l'absence de processus automatisés. Chaque étape, du développement à la mise en production, doit être réalisée manuellement, ce qui prolonge les délais et réduit la flexibilité.

Les mises à jour des applications nécessitent également une intervention manuelle, augmentant les risques d'indisponibilité des services pendant les opérations de maintenance.

5. Manque de supervision centralisée des services :

La surveillance des performances des services n'est pas centralisée, ce qui rend difficile la détection proactive des problèmes. Les incidents peuvent passer inaperçus jusqu'à ce qu'ils affectent gravement les utilisateurs finaux.

L'absence d'une vue d'ensemble des performances rend difficile l'optimisation des ressources et la gestion des incidents en temps réel.

Ces limitations et inefficacités démontrent la nécessité de moderniser l'infrastructure actuelle. L'automatisation des tâches répétitives, l'intégration de solutions de surveillance centralisée, et l'optimisation des processus de déploiement sont essentielles pour améliorer l'efficacité, la fiabilité et la flexibilité de l'infrastructure dans son ensemble.

II. Recueil et spécifications des besoins

Dans cette section, les exigences du projet sont définies pour concevoir une solution automatisée et efficace. Les besoins fonctionnels et non fonctionnels sont identifiés afin de répondre aux défis actuels et de préparer l'infrastructure pour les besoins futurs.

1. Besoins fonctionnels :

Automatisation des déploiements : Le besoin principal est de mettre en place un système d'automatisation pour le déploiement des services SaaS. Cela comprend l'utilisation d'Ansible pour orchestrer le déploiement des conteneurs Docker, garantissant des

configurations cohérentes et reproductibles. Chaque déploiement doit être rapide, fiable, et nécessiter peu d'intervention humaine.

Gestion des ports : Une gestion dynamique des ports est nécessaire pour permettre l'exécution simultanée de plusieurs instances des mêmes applications, comme Odoo et Nginx, sans conflits. Les ports doivent être alloués de manière automatisée et optimisée pour maximiser l'utilisation des ressources.

Supervision continue des services : Un système de surveillance centralisé doit être mis en place pour assurer une visibilité en temps réel sur les performances des services et de l'infrastructure. L'intégration de Prometheus pour la collecte des métriques et de Grafana pour la visualisation est essentielle pour détecter rapidement les problèmes et optimiser les ressources.

Intégration d'APIs : La solution doit inclure des points d'API permettant l'intégration avec d'autres systèmes tiers, facilitant ainsi l'interaction et l'extension des services SaaS hébergés.

2. Besoins non fonctionnels :

Fiabilité et disponibilité : L'infrastructure doit garantir une haute disponibilité des services, minimisant les interruptions et assurant la continuité des opérations. Cela inclut la redondance des services critiques et la mise en place de mécanismes de récupération en cas de défaillance.

Scalabilité : La solution doit être capable de s'adapter facilement à une augmentation de la demande. Cela implique une architecture capable de gérer une croissance du nombre de services et d'utilisateurs sans compromettre les performances.

Sécurité : Des mesures de sécurité doivent être intégrées à tous les niveaux, de la configuration des conteneurs à la gestion des accès. Cela inclut la sécurisation des communications, la gestion des identités et des accès, et la mise en place de politiques de sécurité pour protéger l'infrastructure et les données.

Performance : L'infrastructure doit être optimisée pour assurer des temps de réponse rapides et une utilisation efficace des ressources. Les performances des applications doivent être surveillées en permanence pour identifier et corriger les goulots d'étranglement.

3. Identification des problèmes à résoudre :

Complexité des déploiements manuels : L'un des principaux problèmes est la complexité et le temps requis pour les déploiements manuels. La solution doit simplifier et accélérer ces processus grâce à l'automatisation.

Conflits de ports : La gestion manuelle des ports entraîne des conflits lors du déploiement de plusieurs instances d'applications. Une gestion automatisée des ports doit être mise en place pour éviter ces conflits.

Absence de supervision centralisée : L'infrastructure actuelle manque de supervision centralisée, rendant difficile la détection proactive des problèmes. Un système de surveillance en temps réel doit être implémenté pour pallier ce manque.

4. Guide pour le développement et la mise en œuvre :

Le développement de la solution SaaS doit être orienté par ces besoins fonctionnels et non fonctionnels. Docker sera utilisé pour la conteneurisation des applications, Ansible pour l'automatisation des déploiements, et Python pour le développement de scripts spécifiques et l'intégration des APIs.

La solution doit être conçue de manière modulaire, permettant des mises à jour et des évolutions futures sans perturber les services existants. L'infrastructure doit également être documentée de manière détaillée pour faciliter la maintenance et les améliorations continues.

- Ces besoins constituent la base du projet et guideront toutes les étapes du développement et de la mise en œuvre de la solution automatisée. Ils permettront d'assurer une infrastructure plus efficace, évolutive, et adaptée aux exigences actuelles et futures.

Chapitre 3 : Conception et Modélisation

Le chapitre Conception et modélisation aborde les aspects techniques essentiels de la solution d'infrastructure pour l'hébergement des produits SaaS. Ce chapitre détaillera l'architecture de l'infrastructure, la conteneurisation des applications, l'automatisation des déploiements, ainsi que le développement de scripts Python pour la gestion des ports et la supervision des services. Nous examinerons également les mécanismes de surveillance mis en place, et la manière dont ils s'intègrent dans l'ensemble du système. Ce chapitre fournira les fondations techniques pour créer une infrastructure scalable, automatisée, et adaptée aux besoins futurs des applications SaaS.

I. Architecture du Système

Le système vise à fournir une infrastructure robuste et automatisée pour le déploiement et la gestion des applications conteneurisées, en utilisant des technologies telles que Docker, Ansible, et Python. L'architecture est conçue pour héberger des produits SaaS, avec une attention particulière à la flexibilité, à l'évolutivité, et à la gestion efficace des ressources.

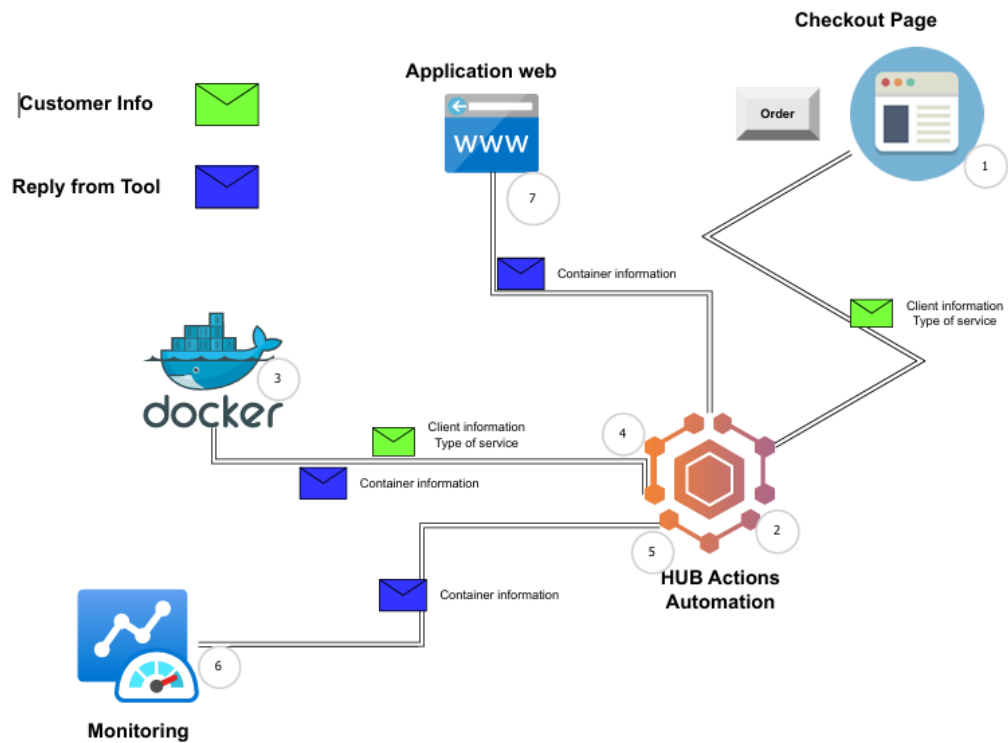


Figure 6 : Le Schéma de workflow de notre système

1. Analyse du système

- Le diagramme ci-dessous montre comment les différentes parties interagissent pour permettre l'automatisation et l'optimisation de la gestion d'infrastructure.

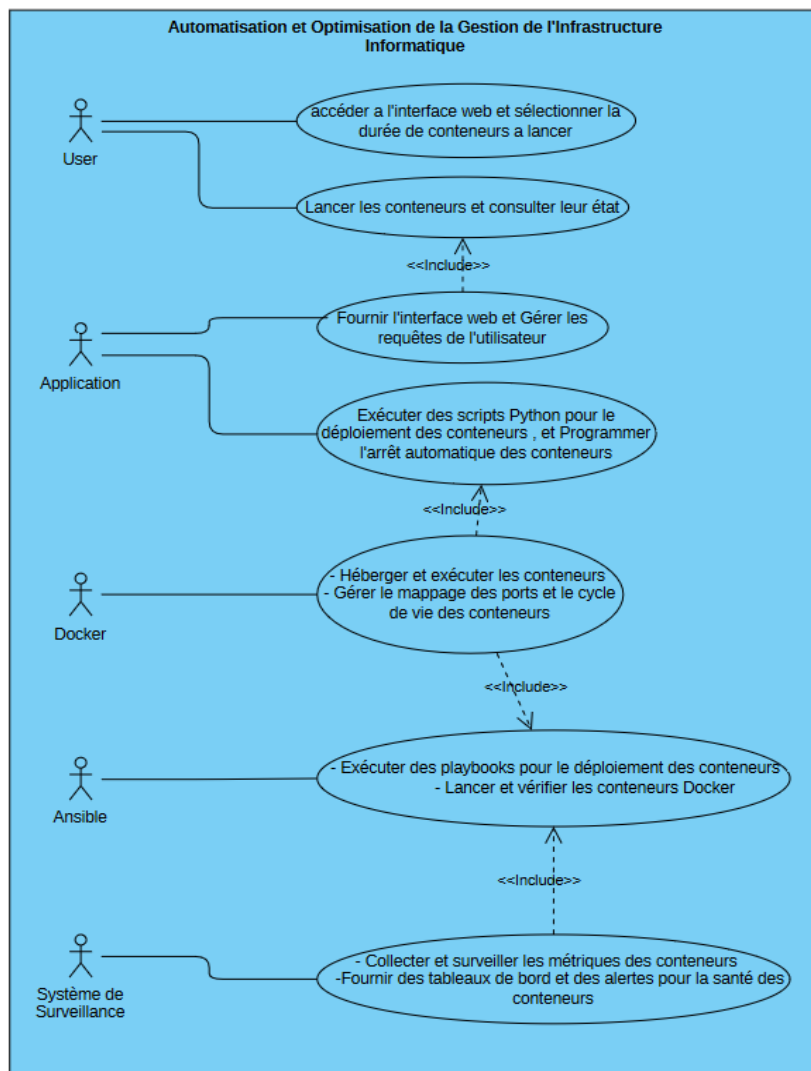


Figure 7: Diagramme de cas d'utilisation

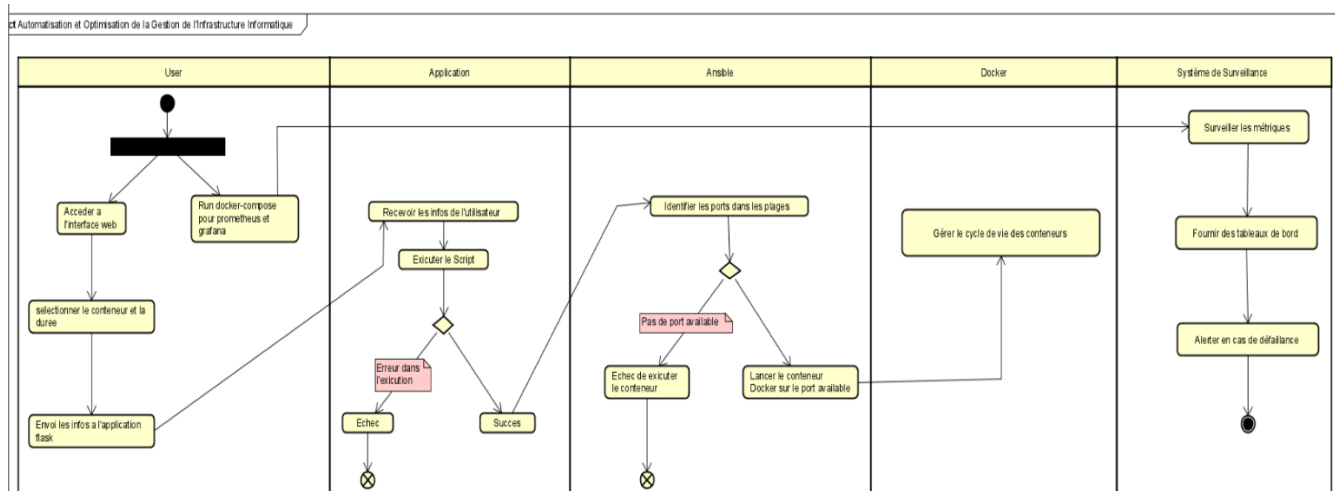


Figure 8 : Diagramme d'Activité

- Le diagramme ci-dessus montre le flux de travail ou les processus à l'intérieur de système. Il illustre la séquence des activités ou des actions, souvent liées à des cas d'utilisation spécifiques.

II. Automatisation et Orchestration

Ansible : Utilisé pour automatiser le déploiement, la configuration, et la gestion des conteneurs Docker. Des playbooks spécifiques sont créés pour le déploiement de Nginx, Odoo, et PostgreSQL, ainsi que pour la gestion dynamique des ports.

Docker Compose : Facilite l'orchestration des services Docker, permettant de gérer plusieurs conteneurs avec une configuration centralisée.

III. Gestion Dynamique des Ports

Script Python : Gère la disponibilité des ports et attribue dynamiquement les ports aux conteneurs en fonction de la disponibilité. Cela permet une gestion flexible et évite les conflits de ports.

IV. Monitoring

- **Prometheus** :

Rôle : Collecte des métriques de performance et des données de monitoring à partir des conteneurs et des services. Prometheus est configuré pour surveiller les différents aspects des services déployés et stocker ces données pour une analyse ultérieure.

Configuration : Les jobs de scraping sont définis dans le fichier prometheus.yml pour recueillir les métriques nécessaires.

- **Grafana :**

Rôle : Visualise les données collectées par Prometheus à l'aide de tableaux de bord interactifs. Grafana est utilisé pour créer des graphiques et des alertes basées sur les métriques collectées, offrant ainsi une vue en temps réel de la santé et des performances du système.

Configuration SMTP : Configuré pour envoyer des alertes par email en cas de dépassement de seuils définis.

```
##### SMTP / Emailing #####
[smtp]
enabled = true
host = smtp.gmail.com:587
user = abdellahloulanti234@gmail.com
# If the password contains # or ; you have to wrap it with triple quotes. Ex ""#password;""
password = ""rofounmfxfnxchv""
;cert_file =
;key_file =
skip_verify = true
from_address = abdellahloulanti234@gmail.com
from_name = Grafana
# EHLO identity in SMTP dialog (defaults to instance_name)
ehlo_identity = smtp.gmail.com
# SMTP startTLS policy (defaults to 'OpportunisticStartTLS')
starttls_policy = OpportunisticStartTLS
# Enable trace propagation in e-mail headers, using the 'traceparent', 'tracestate' and (optionally) 'baggage' fields
# (see)
enable_tracing = false

[smtp.static_headers]
# Include custom static headers in all outgoing emails
;Foo-Header = bar
;Foo = bar

[emails]
```

Figure 9 : Configuration Smtip

- L'architecture du système est conçue pour offrir une solution scalable et automatisée pour le déploiement des applications SaaS. En utilisant Docker pour la conteneurisation, Ansible pour l'automatisation, Python pour la gestion dynamique des ports, et Prometheus avec Grafana pour le monitoring, le système assure une gestion efficace.

Chapitre 4 : Réalisation et Mise en œuvre

Le chapitre 4, *Réalisation et Mise en œuvre*, se focalise sur l'application pratique des exigences et des spécifications définies précédemment pour développer une infrastructure SaaS efficace. Il couvre les différentes étapes du déploiement, de la configuration, et de l'implémentation des serveurs, des conteneurs Docker, ainsi que des services associés. Ce chapitre met en lumière comment les concepts architecturaux ont été intégrés dans un cadre opérationnel. Il souligne l'importance de l'automatisation des processus à l'aide d'Ansible et de Python, la gestion optimisée des ressources, et l'intégration des divers services. Nous examinerons en détail les mesures prises pour transformer la vision architecturale en une solution fonctionnelle, garantissant une gestion efficace et une performance optimale des produits SaaS déployés.

1. Déploiement de l'Infrastructure

1. Configuration de l'Infrastructure Serveur

1.1 Détails sur la configuration des serveurs (Linux et Windows) :

- **Serveur Ubuntu :**

Pour commencer, le serveur Ubuntu, qui sera utilisé comme hôte principal pour les conteneurs Docker, doit être configuré avec les dernières mises à jour et les packages

nécessaires. Cela inclut l'installation de packages essentiels tels que curl, git, python et d'autres outils requis pour le bon fonctionnement de Docker. De plus, les utilisateurs et les permissions doivent être configurés correctement pour assurer la sécurité du système.

```
abdellah@DESKTOP-7N8JG03:/mnt/c/Users/HP/dockerImages$ sudo apt-get install python3
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.10.6-1~22.04.1).
python3 set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 13 not upgraded.
```

Figure 10 : Installation package python

- **Serveur Windows:**

Dans un environnement Windows, la configuration du serveur est également essentielle. Si des tâches spécifiques doivent être exécutées sous Windows, il est crucial d'installer les mises à jour nécessaires et de configurer les permissions utilisateurs. Windows peut être utilisé pour des besoins spécifiques, tels que l'exécution de certaines applications ou services qui ne sont pas compatibles avec Linux.

1.2 Installation des prérequis nécessaires :

Une fois les serveurs configurés, l'installation des prérequis est une étape clé. Pour Ubuntu, cela inclut l'installation de Docker et de Docker Compose, les principaux outils utilisés pour la gestion des conteneurs.

Sur le serveur Windows, l'installation de Docker Desktop permet de gérer les conteneurs de manière similaire à Linux. Il est important de s'assurer que les outils et scripts nécessaires sont compatibles avec l'environnement Windows et bien configurés pour l'intégration avec le serveur Ubuntu.

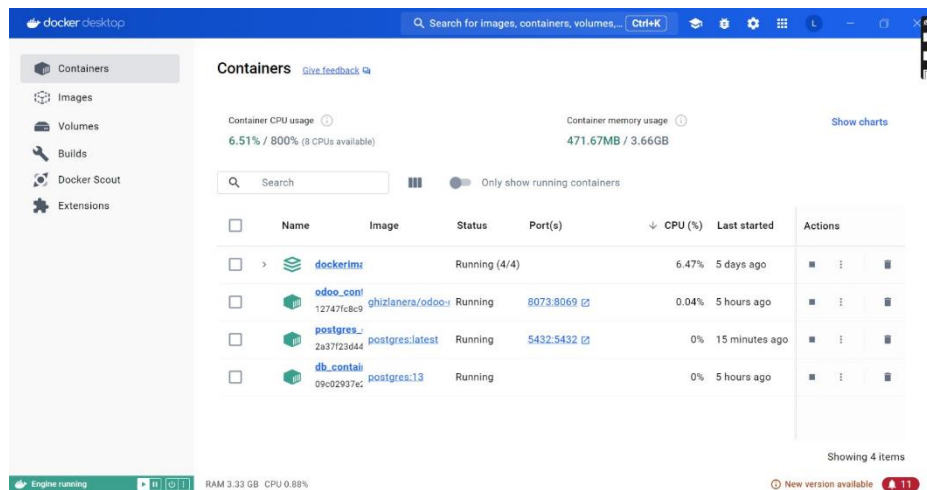


Figure 11 : Docker Desktop

2. Mise en Place de l'Environnement de Conteneurisation

2.1 Installation et configuration de Docker :

Docker est l'outil principal pour la conteneurisation des applications dans ce projet. Après l'installation de Docker sur le serveur Ubuntu, il est essentiel de vérifier son bon fonctionnement. Cette vérification est faite en exécutant des commandes de base telles que `docker --version` pour s'assurer que la version installée est correcte, et `docker info` pour obtenir des informations détaillées sur l'installation.

```

abdellah@DESKTOP-7N8JG03:/mnt/c/Users/HP/dockerImages$ docker --version
Docker version 26.1.4, build 5650f9b
abdellah@DESKTOP-7N8JG03:/mnt/c/Users/HP/dockerImages$ docker info
Client:
Version:      26.1.4
Context:      default
Debug Mode:   false
Plugins:
buildx: Docker Buildx (Docker Inc.)
  Version:    v0.14.1-desktop.1
  Path:       /usr/local/lib/docker/cli-plugins/docker-buildx
compose: Docker Compose (Docker Inc.)
  Version:    v2.27.1-desktop.1
  Path:       /usr/local/lib/docker/cli-plugins/docker-compose
debug: Get a shell into any image or container (Docker Inc.)
  Version:    0.0.32
  Path:       /usr/local/lib/docker/cli-plugins/docker-debug
dev: Docker Dev Environments (Docker Inc.)
  Version:    v0.1.2
  Path:       /usr/local/lib/docker/cli-plugins/docker-dev
extension: Manages Docker extensions (Docker Inc.)
  Version:    v0.2.24
  Path:       /usr/local/lib/docker/cli-plugins/docker-extension
feedback: Provide feedback, right in your terminal! (Docker Inc.)
  Version:    v1.0.5
  Path:       /usr/local/lib/docker/cli-plugins/docker-feedback
init: Creates Docker-related starter files for your project (Docker Inc.)
  Version:    v1.2.0
  Path:       /usr/local/lib/docker/cli-plugins/docker-init
sbom: View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc.)
  Version:    0.6.0
  Path:       /usr/local/lib/docker/cli-plugins/docker-sbom
scout: Docker Scout (Docker Inc.)
  Version:    v1.9.3
  Path:       /usr/local/lib/docker/cli-plugins/docker-scout
Server:
Containers: 10
Running: 10
Paused: 0

```

Figure 12 : docker info

Docker Compose, un outil supplémentaire pour gérer des configurations multi-conteneurs, est ensuite installé. Il permet de définir et de déployer plusieurs conteneurs simultanément. Là encore, une vérification de l'installation est nécessaire pour confirmer que Docker Compose fonctionne correctement.

```

abdellah@DESKTOP-7N8JG03:/mnt/c/Users/HP/dockerImages$ sudo curl -L "https://github.com/docker/compose/releases/download/v2.27.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
[sudo] password for abdellah:
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
  0     0    0     0    0     0      0  0 --:--:--  0:00:09 --:--:--  0
99 60.1M  99 60.0M    0     0  111k    0  0:09:11  0:09:10  0:00:01 26557

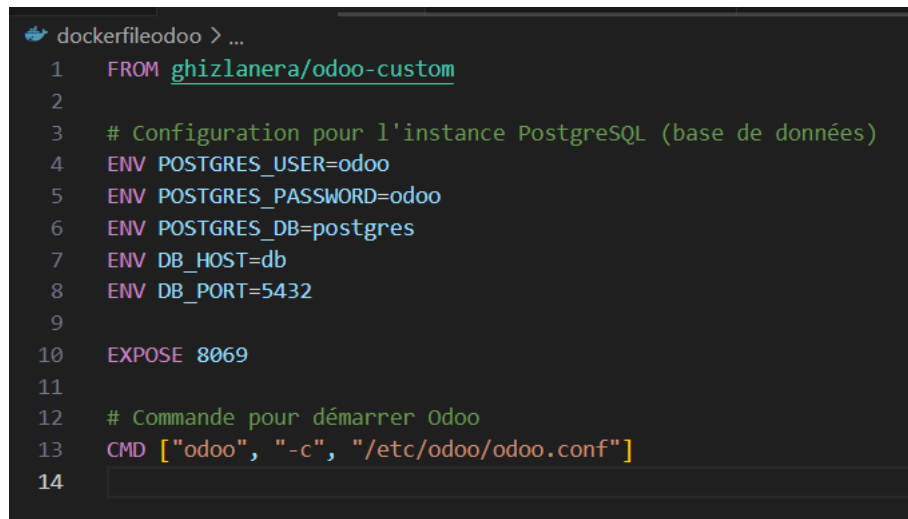
```

Figure 13 : Installation de docker-compose

2.2 Déploiement des images Docker :

- Odoo:

Une fois Docker installé et configuré, le déploiement des conteneurs Odoo et PostgreSQL peut commencer. Pour Odoo, l'image Docker personnalisée **ghizlanera/odoo-custom:latest** est téléchargée et déployée. La configuration de ce conteneur nécessite la définition de volumes pour persister les données et de variables d'environnement spécifiques pour configurer Odoo selon les besoins du projet. Une fois le conteneur déployé, son état est vérifié pour s'assurer qu'il fonctionne correctement.

A screenshot of a code editor showing a Dockerfile for Odoo. The file is named 'dockerfileodoo' and is located in a directory named '...'. The code is as follows:

```
1 FROM ghizlanera/odoo-custom
2
3 # Configuration pour l'instance PostgreSQL (base de données)
4 ENV POSTGRES_USER=odoo
5 ENV POSTGRES_PASSWORD=odoo
6 ENV POSTGRES_DB=postgres
7 ENV DB_HOST=db
8 ENV DB_PORT=5432
9
10 EXPOSE 8069
11
12 # Commande pour démarrer Odoo
13 CMD ["odoo", "-c", "/etc/odoo/odoo.conf"]
14
```

Figure 14 : dockerfile de l'image Odoo

De manière similaire, le conteneur PostgreSQL est déployé à partir de l'image Docker postgres:13. Les volumes pour la base de données et les variables d'environnement, comme les utilisateurs et les mots de passe, sont configurés avant de lancer le conteneur. Une vérification du bon fonctionnement est ensuite effectuée.

```

1 |
2 version: '3.8'
3
4 services:
5   odoo:
6     image: ghizlanera/odoo-custom
7     depends_on:
8       - db
9     environment:
10      - POSTGRES_USER=odoo
11      - POSTGRES_PASSWORD=odoo
12      - POSTGRES_DB=postgres
13      - DB_HOST=db
14      - DB_PORT=5432
15     ports:
16       - "8328:8069"
17     restart: always
18
19   db:
20     image: postgres:13
21     environment:
22       POSTGRES_USER: odoo
23       POSTGRES_PASSWORD: odoo
24       POSTGRES_DB: postgres
25     volumes:
26       - odoo-db-data:/var/lib/postgresql/data
27     restart: always
28
29 volumes:
30   odoo-db-data:

```

Figure 15 : Fichier de docker-compose du Container Odoo

- **Nginx:**

L'image Docker officielle **nginx:latest** est utilisée pour garantir une compatibilité maximale avec les applications et services web. Pour déployer Nginx, le conteneur est configuré avec les volumes nécessaires pour persister les fichiers de configuration et les logs. De plus, les ports sont mappés de manière appropriée pour permettre l'accès au serveur web à partir du réseau externe.

```

Dockerfile > FROM
1  FROM nginx:latest
2
3  # Copiez le fichier de configuration nginx.conf dans le conteneur
4  COPY nginx.conf /etc/nginx/nginx.conf
5
6  # Exposez le port 80 pour le service
7  EXPOSE 80
8
9  CMD ["nginx", "-g", "daemon off;"]
10

```

Figure 16 : dockerfile de l'image Odoo

- **Conteneurs de production:**

Pour assurer une surveillance et une gestion efficaces de mon infrastructure, j'ai intégré Prometheus, Grafana, cAdvisor, et Node Exporter dans mon projet. Prometheus collecte des métriques détaillées sur les performances des services et des conteneurs, tandis que Grafana offre une interface visuelle pour analyser ces données via des tableaux de bord interactifs. cAdvisor fournit des métriques spécifiques aux conteneurs Docker, et Node Exporter surveille les performances des serveurs hôtes. Cette combinaison d'outils permet une surveillance complète, une détection rapide des problèmes, et une optimisation continue des performances de l'infrastructure.

```

docker-compose.yml
1  version: '3.8'
2
3  services:
4    prometheus:
5      image: prom/prometheus
6      ports:
7        - "9090:9090"
8      volumes:
9        - C:/Users/HP/prometheus-2.53.1.windows-amd64/prometheus.yml:/etc/prometheus/prometheus.yml
10
11    grafana:
12      image: grafana/grafana
13      ports:
14        - "3001:3000"
15      environment:
16        - GF_SECURITY_ADMIN_PASSWORD=admin
17
18    node-exporter:
19      image: prom/node-exporter
20      ports:
21        - "9100:9100"
22
23    cAdvisor:
24      image: google/cadvisor:latest
25      ports:
26        - "8089:8080"
27      volumes:
28        - /:/rootfs:ro
29        - /var/run:/var/run:rw
30        - /sys:/sys:ro
31        - /var/lib/docker:/var/lib/docker:ro
32      restart: always
33

```

Figure 17 : fichier docker-compose du conteneurs du Production

3. Configuration du Réseau et des Services

Pour que les services conteneurisés puissent communiquer efficacement entre eux et avec le monde extérieur, une configuration réseau adéquate a été mise en place. Cela a impliqué la création de sous-réseaux Docker spécifiques pour isoler les différents services tout en leur permettant de communiquer de manière sécurisée. Des règles de pare-feu ont été établies pour restreindre l'accès aux seuls ports nécessaires, garantissant ainsi que seuls les services légitimes pouvaient être atteints.

```
abdelah@DESKTOP-7N8JG03:/mnt/c/Users/HP/dockerImages$ docker network ls
NETWORK ID          NAME                DRIVER             SCOPE
ecd047ebce48        bridge              bridge             local
118ee1bb2430        dockerimages_default bridge             local
1ca1a8b5fd76        host                host               local
74291f444db2        none                null               local
44bf4c3b6b88        odoo_network        bridge            local
abdelah@DESKTOP-7N8JG03:/mnt/c/Users/HP/dockerImages$
```

Figure 18 : docker Network

En parallèle, les groupes de sécurité ont été configurés pour définir des règles d'accès spécifiques pour chaque service. Cela a permis d'assurer une isolation entre les différentes parties de l'infrastructure tout en autorisant les communications nécessaires. Enfin, des règles de gestion des accès ont été mises en place pour s'assurer que seuls les utilisateurs autorisés pouvaient interagir avec les services critiques, renforçant ainsi la sécurité globale du système.

II. Automatisation et Gestion des Ressources

1. Automatisation avec Ansible

L'une des pierres angulaires du projet a été l'automatisation des déploiements et des configurations à l'aide d'Ansible. Ansible a été choisi pour sa simplicité et son efficacité dans la gestion des configurations d'infrastructure. Des playbooks spécifiques ont été créés pour automatiser le déploiement des conteneurs Odoo, PostgreSQL et Nginx, ce qui a permis de standardiser et de simplifier le processus de déploiement. Chaque playbook a été conçu pour exécuter des tâches répétitives, telles que l'installation des packages, la configuration des services, et le démarrage des conteneurs.

Grâce à cette automatisation, il a été possible de déployer rapidement de nouvelles instances de services, réduisant ainsi le temps nécessaire pour mettre en place l'infrastructure. De plus, les playbooks Ansible ont permis de garantir la cohérence des configurations à travers tous les environnements, minimisant ainsi les risques d'erreurs humaines. Cette approche a considérablement amélioré l'efficacité opérationnelle du projet.

```
playbooks > ! nginx_playbook.yml
1  ---
2  - hosts: localhost
3    tasks:
4      - name: Run Nginx container
5        docker_container:
6          name: "nginx_container_{{ next_port }}"
7          image: "{{ image_name }}"
8          state: started
9          ports:
10             - "{{ port }}:80"
11
```

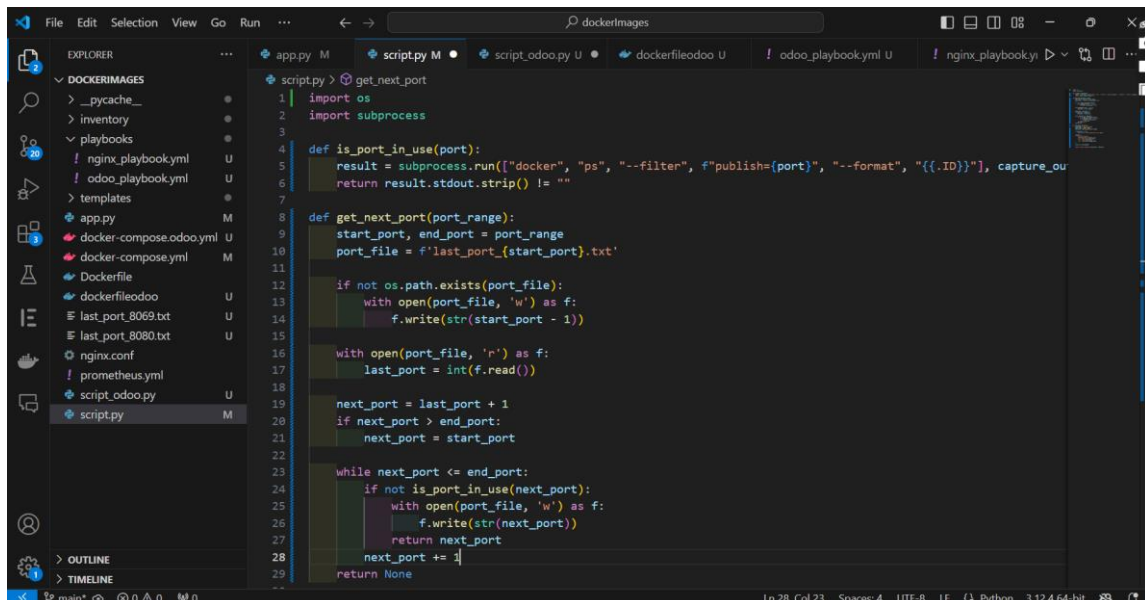
Figure 19 : fichier ansible-playbook du Nginx

```
playbooks > ! odoo_playbook.yml
1  ---
2  - hosts: localhost
3    tasks:
4      - name: Run PostgreSQL container
5        docker_container:
6          name: "postgres_container"
7          image: "postgres:13"
8          state: started
9          ports:
10             - "5432:5432"
11          env:
12             POSTGRES_USER: "odoo"
13             POSTGRES_PASSWORD: "odoo"
14             POSTGRES_DB: "postgres"
15
16      - name: Run Odoo container
17        docker_container:
18          name: "odoo_container_{{ next_port }}"
19          image: "ghizlanera/odoo-custom:latest"
20          state: started
21          ports:
22             - "{{ port }}:8069"
23          env:
24             POSTGRES_USER: "odoo"
25             POSTGRES_PASSWORD: "odoo"
26             POSTGRES_DB: "postgres"
27             DB_HOST: "postgres_container" # Utilise le nom du conteneur PostgreSQL
28          links:
29             - "postgres_container:db" # Lien vers le conteneur PostgreSQL avec alias 'db'
30
```

Figure 20 : fichier ansible-playbook du Odoo

2. Scripting Python pour la Gestion des Ports et des Déploiements

En complément de l'automatisation avec Ansible, des scripts Python ont été développés pour gérer les aspects spécifiques du déploiement, tels que l'allocation dynamique des ports et la gestion des déploiements. Ces scripts ont permis de résoudre des défis spécifiques liés à l'utilisation des ports pour les conteneurs Odoo et Nginx, en s'assurant que les ports étaient correctement alloués et disponibles pour chaque instance. Les scripts Python ont également été utilisés pour automatiser des tâches complexes non couvertes par Ansible, telles que la gestion des intervalles de temps pendant lesquels les conteneurs devaient rester actifs. En intégrant ces scripts dans les playbooks Ansible, il a été possible de créer une solution cohérente et entièrement automatisée pour le déploiement et la gestion des conteneurs.



```
1 import os
2 import subprocess
3
4 def is_port_in_use(port):
5     result = subprocess.run(["docker", "ps", "--filter", f"publish={port}", "--format", "{{.ID}}"], capture_output=True)
6     return result.stdout.strip() != ""
7
8 def get_next_port(port_range):
9     start_port, end_port = port_range
10    port_file = f'last_port_{start_port}.txt'
11
12    if not os.path.exists(port_file):
13        with open(port_file, 'w') as f:
14            f.write(str(start_port - 1))
15
16    with open(port_file, 'r') as f:
17        last_port = int(f.read())
18
19    next_port = last_port + 1
20    if next_port > end_port:
21        next_port = start_port
22
23    while next_port <= end_port:
24        if not is_port_in_use(next_port):
25            with open(port_file, 'w') as f:
26                f.write(str(next_port))
27            return next_port
28        next_port += 1
29    return None
```

The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like DOCKERIMAGES and files like app.py, script.py, and various Docker-related files. The code editor displays a Python script named `script.py` with the following content:

```

30
31 def run_nginx_container():
32     image_name = "my-nginx-image"
33     port_range = (8080, 8090)
34     next_port = get_next_port(port_range)
35
36     if next_port is None:
37         return None, "No available ports for Nginx."
38
39     command = [
40         "ansible-playbook",
41         "playbooks/nginx_playbook.yml",
42         "-e", f"next_ports={next_port}",
43         "-e", f"port={next_port}",
44         "-e", f"image_name={image_name}"
45     ]
46
47     subprocess.run(command)
48
49     return f"nginx_container_{next_port}", next_port
50

```

Figure 21 : le script python pour Nginx

The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like DOCKERIMAGES and files like app.py, script.py, and various Docker-related files. The code editor displays a Python script named `script.py` with the following content:

```

1 import os
2 import subprocess
3
4 def is_port_in_use(port):
5     result = subprocess.run(["docker", "ps", "--filter", f"publish={port}", "--format", "{{.ID}}"], capture_output=True)
6     return result.stdout.strip() != ""
7
8 def get_next_port(port_range):
9     start_port, end_port = port_range
10    port_file = f'last_port_{start_port}.txt'
11
12    if not os.path.exists(port_file):
13        with open(port_file, 'w') as f:
14            f.write(str(start_port - 1))
15
16    with open(port_file, 'r') as f:
17        last_port = int(f.read())
18
19    next_port = last_port + 1
20    if next_port > end_port:
21        next_port = start_port
22
23    while next_port <= end_port:
24        if not is_port_in_use(next_port):
25            with open(port_file, 'w') as f:
26                f.write(str(next_port))
27            return next_port
28        next_port += 1
29    return None
30

```

```

30
31 def run_odoo_container():
32     image_name = "my-odoo-image"
33     port_range = (8069, 8079)
34     next_port = get_next_port(port_range)
35
36     if next_port is None:
37         return None, "No available ports for Odoo."
38
39     command = [
40         "ansible-playbook",
41         "playbooks/odoo_playbook.yml", # Chemin mis à jour
42         "-e", f"next_port={next_port}",
43         "-e", f"port={next_port}",
44         "-e", f"image_name={image_name}"
45     ]
46
47     subprocess.run(command)
48
49     return f"odoo_container_{next_port}", next_port
50

```

Figure 22 : le script python pour Odoo

III. Supervision et Suivi des Services

1. Mise en Place de la Supervision avec Prometheus et Grafana

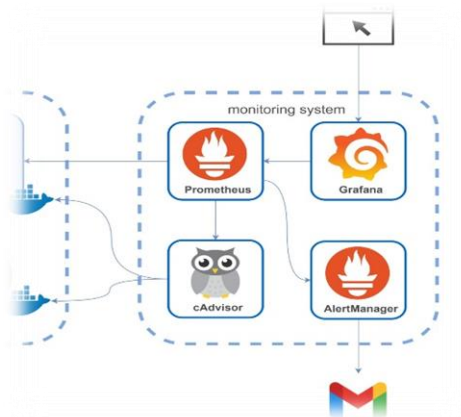


Figure 23: Schéma de système du Supervision

La supervision des services a été un aspect essentiel pour assurer leur bon fonctionnement et détecter les éventuels problèmes de performance. Pour cela, Prometheus et Grafana ont été déployés pour surveiller l'état des conteneurs et collecter des métriques en temps réel. Prometheus a été configuré pour extraire des métriques telles que l'utilisation du CPU, de la mémoire, et des disques, tandis que Grafana a été utilisé pour visualiser ces données sous forme de tableaux de bord interactifs.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://cadvisor:8080/metrics	UP	instance="cadvisor:8080" job="cadvisor"	9.886s ago	389.126ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://node-exporter:9100/metrics	UP	instance="node-exporter:9100" job="node-exporter"	2.864s ago	35.987ms	

Figure 24: l'interface Prometheus (DataSource)

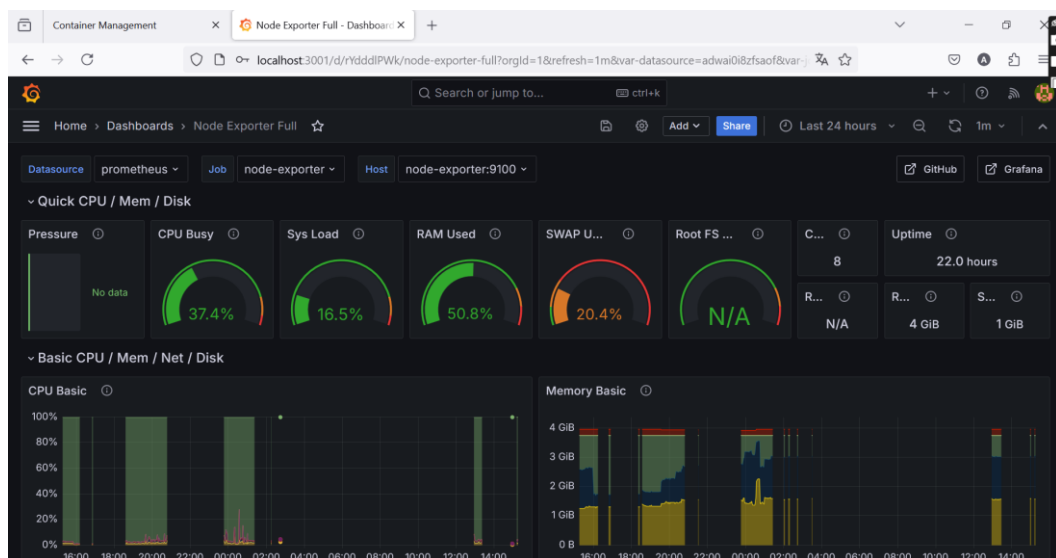


Figure 25: Dashboard Grafana Illustrant les Performances de l'Hôte via Node Exporter

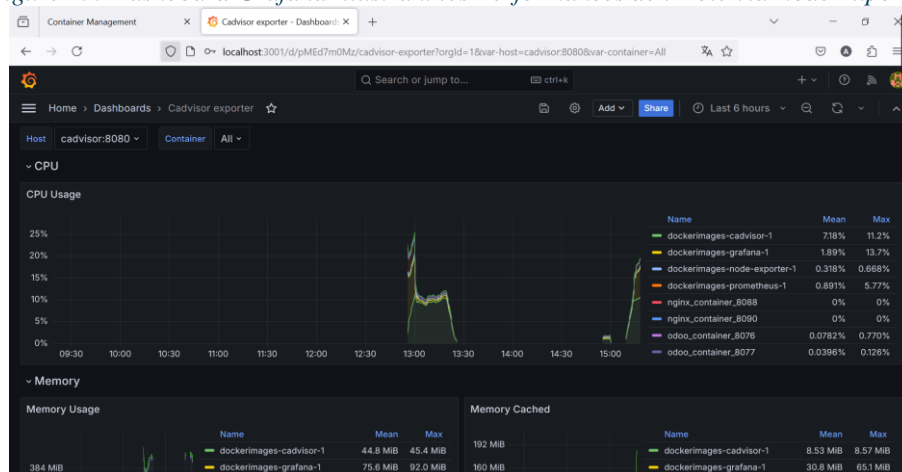


Figure 26: Visualisation des Performances des Conteneurs Docker dans Grafana

Grâce à cette configuration, il a été possible de suivre en temps réel l'état de santé des services et de réagir rapidement en cas d'anomalies. Des alertes ont également été configurées pour notifier les administrateurs en cas de dépassement de seuils critiques, permettant ainsi de prévenir les pannes avant qu'elles ne se produisent.

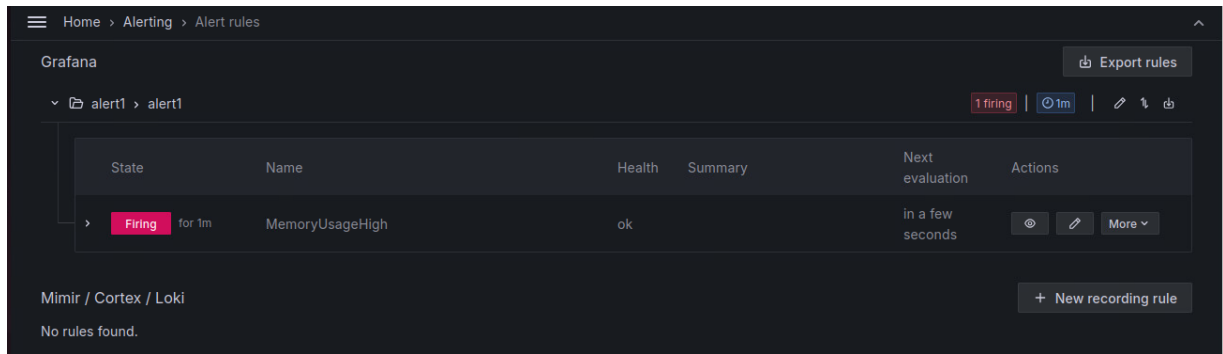


Figure 27: Alerte configuré

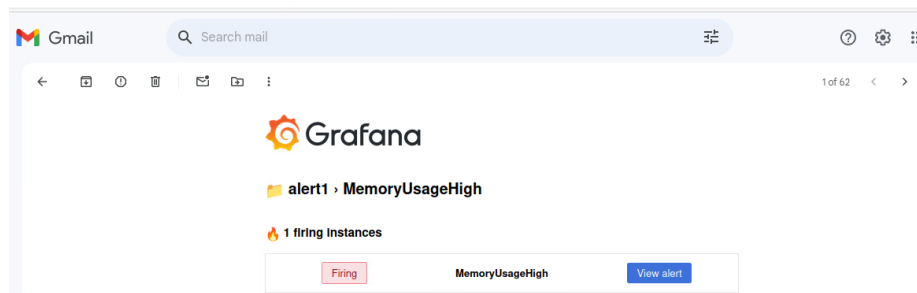


Figure 28: Notification d'Alerte via Gmail

2. Collecte et Analyse des Données

La collecte des métriques via Prometheus a permis de disposer d'une quantité importante de données sur le comportement des services. Ces données ont été analysées pour identifier les éventuelles optimisations à apporter, tant au niveau de la configuration des conteneurs que de l'utilisation des ressources. L'analyse a permis de détecter des goulots d'étranglement potentiels et de prendre des mesures pour améliorer les performances globales de l'infrastructure. Par exemple, des ajustements ont été faits sur l'allocation des ressources pour certains conteneurs, permettant de réduire l'utilisation du CPU tout en maintenant des performances optimales. Cette approche proactive a permis d'améliorer la résilience de l'infrastructure et de garantir une qualité de service élevée.

IV. Tests de Fonctionnement et d'Optimisation

1. Tests de Fonctionnement des Services

Les tests de fonctionnement ont été réalisés pour vérifier que chaque service déployé fonctionnait correctement et que les communications entre les conteneurs étaient optimales. Ces tests ont inclus des vérifications de la connectivité entre Odoo et PostgreSQL, ainsi que des tests de performance pour s'assurer que les services pouvaient supporter la charge prévue. Des outils de test ont été utilisés pour simuler des charges de travail et vérifier que les conteneurs répondaient correctement. Les résultats des tests ont permis d'identifier et de résoudre rapidement les problèmes, assurant ainsi que les services étaient prêts pour une utilisation en production.

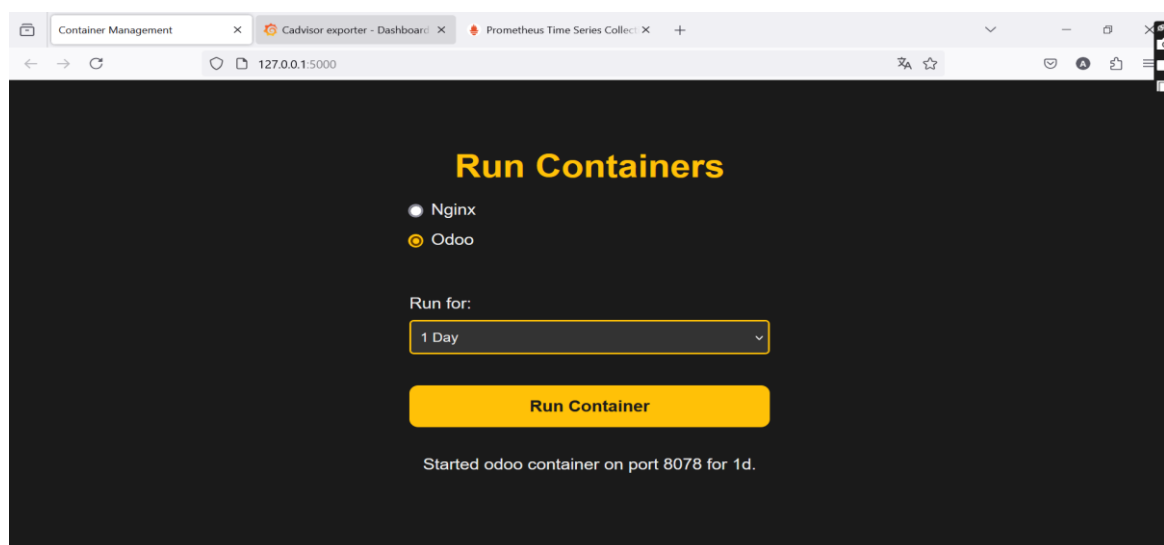


Figure 29: Checkout page

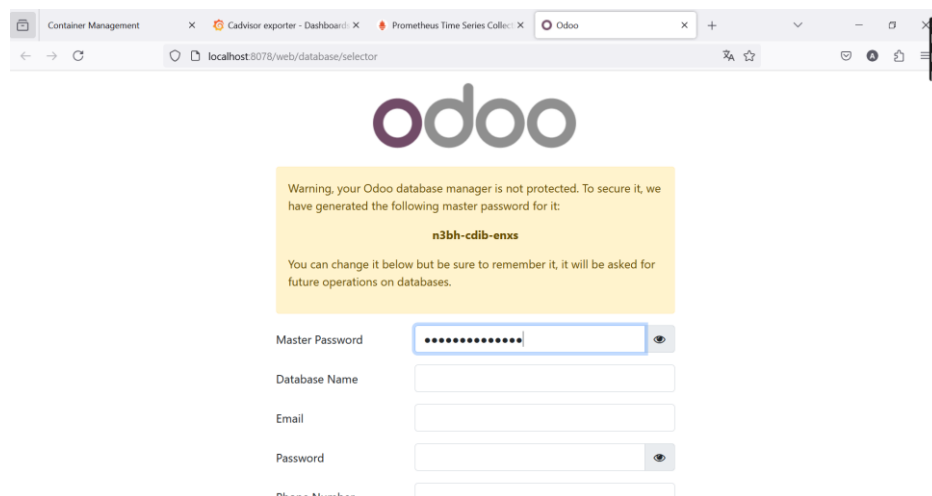


Figure 30: L'instance Odoo sur le port 8078

```

abdellah@DESKTOP-7N8JGO: X + v
abdellah@DESKTOP-7N8JGO3: /mnt/c/Users/HP/dockerImages$ docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
18315473c0e0	ghizlanera/odoo-custom:latest	"/entrypoint.sh odoo..."	About a minute ago	Up About a minute	8071-8072/tcp, 0.0.0.0:8070->8069/tcp	odoo_container_8078 ✓
da94d4f178e0	ghizlanera/odoo-custom:latest	"/entrypoint.sh odoo..."	14 hours ago	Up 14 hours	8071-8072/tcp, 0.0.0.0:8077->8069/tcp	odoo_container_8077
3ad5c3ec50c4	ghizlanera/odoo-custom:latest	"/entrypoint.sh odoo..."	14 hours ago	Up 14 hours	8071-8072/tcp, 0.0.0.0:8076->8069/tcp	odoo_container_8076
8c9bc6048e31	postgres:13	"docker-entrypoint.s..."	14 hours ago	Up 14 hours	0.0.0.0:5432->5432/tcp	postgres_container
5ada77afd3c5	my-nginx-image	"/docker-entrypoint..."	15 hours ago	Up 15 hours	0.0.0.0:8090->80/tcp	nginx_container_8090
c6105897837a	my-nginx-image	"/docker-entrypoint..."	15 hours ago	Up 15 hours	0.0.0.0:8088->80/tcp	nginx_container_8088
cb0924ed51da	prom/prometheus	"/bin/prometheus --c..."	5 days ago	Up 5 days	0.0.0.0:9090->9090/tcp	dockerimages-prometheus-1
790fb95ab699	google/cadvisor:latest	"/usr/bin/cadvisor -..."	5 days ago	Up 5 days	0.0.0.0:8089->8080/tcp	dockerimages-cadvisor-1
66ab561babb3	grafana/grafana	"/run.sh"	5 days ago	Up 5 days	0.0.0.0:3001->3000/tcp	dockerimages-grafana-1
66eff2b98699	prom/node-exporter	"/bin/node_exporter"	5 days ago	Up 5 days	0.0.0.0:9100->9100/tcp	dockerimages-node-exporter-1

Figure 31: Liste des Conteneurs Actifs dans le serveur ubuntu

Container CPU usage: 9.44% / 800% (8 CPUs available)

Container memory usage: 531.84MB / 3.66GB

Search:

Only show running containers: ☒

Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
dockerim		Running (4/4)		9.27%	5 days ago	
postgres_8c9bc6048	postgres:13	Running	5432-5432	0.1%	1 minute ago	
odoo_con_3ad5c3ec5	ghizlanera/odoo-	Running	8076-8069	0.04%	1 minute ago	
odoo_con_da94d4f17	ghizlanera/odoo-	Running	8077-8069	0.04%	28 seconds ago	

Figure 32 : Les Conteneurs Actifs dans le serveur Windows

2. Optimisation des Ressources

Enfin, des efforts d'optimisation ont été entrepris pour améliorer l'efficacité de l'infrastructure. Cela a inclus des ajustements dans les configurations des conteneurs pour réduire l'utilisation des ressources tout en maintenant des performances élevées. Des optimisations ont également été apportées aux scripts et aux playbooks pour réduire les temps d'exécution et améliorer la réactivité des services. Ces optimisations ont permis de maximiser l'efficacité de l'infrastructure, réduisant les coûts tout en augmentant la fiabilité et la performance des services. Les résultats obtenus ont montré une amélioration significative de l'utilisation des ressources, permettant de mieux répondre aux besoins du projet.

Conclusion générale

En conclusion, ce projet de fin d'année marque une étape significative dans mon parcours académique et professionnel. Il a constitué une opportunité précieuse pour approfondir mes compétences en administration d'infrastructure, en développement d'applications, et en automatisation des processus à travers l'utilisation de Docker, Ansible, et Python.

Au cours de ce projet, j'ai conçu et mis en œuvre une solution complète pour la gestion de conteneurs et la provision d'infrastructure basée sur Docker. J'ai intégré des fonctionnalités avancées telles que le déploiement automatisé de conteneurs Odoo et PostgreSQL, la gestion dynamique des ports, ainsi que la configuration et la surveillance de l'infrastructure avec Prometheus, Grafana, cAdvisor et Node Exporter. Cette approche a permis de créer une infrastructure flexible et évolutive, adaptée aux besoins de gestion des applications SaaS.

L'objectif principal de ce projet était d'améliorer l'efficacité et la flexibilité des déploiements d'applications en automatisant la création et la gestion des conteneurs. J'ai atteint cet objectif en développant des scripts et des playbooks Ansible pour l'automatisation des tâches, en configurant des outils de surveillance pour assurer la performance et la disponibilité des services, et en mettant en place des pratiques de gestion des ports et des réseaux adaptés.

Les tests approfondis réalisés ont confirmé le bon fonctionnement des conteneurs et la fiabilité de l'infrastructure mise en place. J'ai vérifié que chaque conteneur était correctement déployé et que les services étaient accessibles comme prévu. Ces validations ont démontré l'efficacité et la robustesse de la solution développée.

Ce projet a été une occasion précieuse pour développer mes compétences en gestion de l'infrastructure informatique, en intégration de systèmes, et en résolution de problèmes complexes. Il m'a également permis de mettre en pratique mes connaissances théoriques et d'acquérir une expérience précieuse en gestion de projet et en collaboration technique.

Je suis reconnaissant d'avoir eu l'opportunité de travailler sur ce projet comme stage de fin d'année . Cette expérience a renforcé mes compétences techniques et ma capacité à relever des défis complexes de manière autonome. Je suis fier des résultats obtenus et convaincu que cette solution apportera des avantages significatifs à toute entreprise cherchant à optimiser ses déploiements d'applications conteneurisées.

En somme, ce projet représente un jalon important dans mon développement professionnel, consolidant mes compétences en administration d'infrastructure et en développement de solutions automatisées. Je suis reconnaissant pour cette opportunité et confiant que cette expérience sera un atout précieux pour ma future carrière.

Références

Docker documentation : <https://docs.docker.com/get-started/overview/>

Docker-compose documentation : <https://docs.docker.com/compose/>

Python documentation : <https://docs.python.org/3/>

Ansible Documentation : [Using Ansible playbooks — Ansible Community Documentation](#)

Grafana Documentation : [Technical documentation | Grafana Labs](#)

General documentation : <https://chat.openai.com/>