RÉPUBLIQUE DU BÉNIN
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ D'ABOMEY-CALAVI

INSTITUT DE FORMATION ET DE RECHERCHE EN INFORMATIQUE

BP 526 Cotonou Tel : +229 21 14 19 88
http://www.ifri-uac.bj Courriel : contact@ifri.uac.bj

MÉMOIRE

pour l'obtention du

Diplôme de Licence en Informatique

Option : Sécurité Informatique

Présenté par :
Calèbe **HOUNGBEME**

Implémentation d'un système de surveillance des microservices d'applications dans un environnement DevSecOps

Sous la supervision :
Dr Arnaud **AHOUANDJINOU**
Enseignant chercheur à l'UAC

Membres du jury :

KONNON Abel	Docteur	UNSTIM	Président
AFFOGNON Lionel	Ingénieur	IMSP	Examinateur
AHOUANDJINOU Arnaud	Docteur	IFRI	Rapporteur

Année Académique : **2023-2024**

Sommaire

Dédicace ii

Remerciements iii

Résumé iv

Abstract v

Liste des figures vi

Liste des tableaux vii

Liste des acronymes viii

Introduction 1

1 Etat de l'art 3

2 Mise en œuvre 11

3 Résultats et discussions 27

Conclusion 31

Bibliographie 32

Webographie 33

Table des matières 34

Dédicace

Je dédie ce travail à :

- Mon père HOUNGBEME Epiphane,
- Ma mère WUSA Edwige,
- Mes frères et mes sœurs, en particulier, Ephraïm, Eunice et Belvice.

Remerciements

La réalisation de ce mémoire a été rendue possible grâce à divers soutiens inestimables. Nous exprimons notre profonde gratitude à :

- Dieu pour son Amour, sa grâce et sa bonté infinie, qui m'ont permis de bénéficier de la vie, de la santé, de l'intelligence, des ressources et du temps nécessaires pour mener à bien ce mémoire ;
- Mon maître de mémoire, Dr Arnaud AHOUANDJINOU, pour sa confiance, sa disponibilité et son accompagnement précieux tout au long de ce travail ;
- Professeur Eugène C. EZIN, le directeur de notre institut, pour ses conseils avisés et pour l'excellence de la formation qu'il nous dispense ;
- Le Directeur Général de Pero Groupe, monsieur Ulriche Fiacre AIHOU, pour son soutien et sa confiance tout au long de cette expérience ;
- Tout le corps professoral de l'Institut de Formation et de Recherche en Informatique (IFRI), qui œuvre avec dévouement pour fournir les outils indispensables à la réussite de nos études universitaires.

Que tous ceux qui ont contribué, en particulier la start-up Pero Groupe, trouvent ici l'expression de mes sentiments les meilleurs.

Résumé

Le DevSecOps est une méthodologie qui fusionne les pratiques de développement d'applications, d'opérations informatiques et de sécurité pour garantir des déploiements et des livraisons rapides d'applications sécurisées. L'évolution vers les architectures basées sur des microservices a créé de nouveaux défis en matière de gestion et de surveillance, en raison de la multiplication de ces dernières. La surveillance de ces applications en architecture microservices est donc devenue essentielle pour assurer leur bon fonctionnement et leur sécurité. Dans le cadre de l'optimisation de l'environnement numérique de travail des institutions universitaires, ce mémoire se concentre sur la mise en place d'un système de surveillance des applications microservices avec Prometheus, en adoptant les principes et les pratiques de DevSecOps dans une CI/CD. Il explore en détail les concepts fondamentaux du DevSecOps, des microservices, ainsi que la surveillance des applications microservices dans un processus d'intégration continue et de déploiement continu. Des tests et mesures de performance sont effectués pour évaluer la disponibilité, les performances et les alertes des microservices. Les résultats montrent l'intérêt de la mise en place d'un système de surveillance des applications microservices pour garantir la sécurité et la disponibilité.

Mots clés : DevSecOps, Disponibilité, Microservice, Environnement Numérique de Travail, CI/CD.

Abstract

DevSecOps is a methodology that merges application development, IT operations, and security practices to ensure fast and secure application deployments and deliveries. The shift towards microservices-based architectures has introduced new challenges in management and monitoring, with their growing proliferation. Monitoring these microservice applications has thus become essential to ensure their proper functioning and security. As part of the optimization of the digital work environment for academic institutions, this thesis focuses on the implementation of a monitoring system for microservice applications with Prometheus, through the adoption of DevSecOps principles and practices in a CI/CD. It explores in detail the fundamental concepts of DevSecOps, microservices, and the monitoring of microservice applications in a continuous integration and continuous deployment process. Tests and performance measurements are conducted to assess the availability, performance, and alert systems of the microservices. The results are conclusive and demonstrate the importance of implementing a monitoring system for microservice-based applications to ensure security and availability.

Keywords: DevSecOps, Availability, Microservice, Digital Work Environment, CI/CD.

Liste des figures

1.1 Comparaison entre DevOps et DevSecOps [1] 4
1.2 Approche CI/CD 4
1.3 Comparaison entre Architecture Monolithique et Microservices [2] 5
1.4 Architecture de Aplus LMS 6
1.5 Architecture de Prometheus 9

2.1 Architecture de l'environnement 12
2.2 Récupération du code source de Aplus LMS depuis le dépôt 13
2.3 Vérification de l'état de nos microservices 13
2.4 Interface après connexion à Aplus LMS 14
2.5 Docker-compose de Prometheus et Grafana 15
2.6 Configuration des cibles à scrapper avec Prometheus 15
2.7 Connexion à Prometheus 16
2.8 Connexion à Grafana 16
2.9 Configuration de Postgre-exporteur et de cadvisor 17
2.10 Ajout du module dans les paramètres de Aplus front 18
2.11 Ajout du middleware de prometheus 18
2.12 Montage du code source de Aplus front 19
2.13 Ajout du middleware de prometheus 19
2.14 Création d'un point d'exposition des métriques 19
2.15 Visualisation de l'état des cibles de prometheus 20
2.16 Ajout d'un proxy et authentification 20
2.17 Création du middleware d'authentification 21
2.18 Vérification du fonctionnement de l'authentification 21
2.19 Création d'utilisateur pour les métriques 22
2.20 Mise à jour de la configuration de la stack des exporteurs 22
2.21 Visualisation du nombre de connexions 23
2.22 Mise en place des règles d'alertes au microservices 24
2.23 Présentation des alertes et de leurs états 25
2.24 Alert Manager pour l'envoi des alertes 25
2.25 Configuration de l'envoi des alertes par mail 26

3.1 Tableau de bord pour les microservices en cours d'utilisation 27
3.2 État des requêtes et de leur code d'erreur 28
3.3 Augmentation du nombre de connexions suite à une tentative de brute force 28
3.4 Réception des alertes par mail 29
3.5 Exemple d'alerte par mail 29

Liste des tableaux

1.1 Principaux microservices de Aplus LMS et leurs langages de programmation 7
1.2 Comparatif de quelques solutions de monitoring 8

Liste des acronymes

API :

Application Programming Interface 4, 5

CI/CD :

Continious Integration / Continious Deployement iv, v, 1, 2, 4

DevOps :

Developpements, Operationnels vi, 3, 4, 10, 34

DevSecOps :

Developpements, Sécurités, Operationnels iv, v, 2, 3, 10, 30, 34

ENT :

Environnement Numérique de Travail 1, 2, 11, 14

IFRI :

Institut de Formation et de Recherche en Informatique iii

LMS :

Learning Management System vi, vii, 5, 7, 12, 13, 35

SSH :

Secure Shell 14

VPS :

Virtual Private Server 11, 14

Introduction Générale

1 Contexte et justification

L'essor rapide des technologies de l'information et de la communication a transformé divers secteurs de notre société, et particulièrement l'éducation. Dans ce cadre, les institutions universitaires développent un espace numérique de travail qui offre des fonctionnalités avancées, ainsi que d'autres services essentiels pour les utilisateurs. Ces environnements numériques de travail, ou ENT, représentent une avancée significative dans la modernisation des institutions académiques, de part leurs architectures basées sur des microservices dédiés à la gestion des cours, des ressources pédagogiques, des évaluations et des communications [3].

2 Problématique

Dans cette quête d'un espace numérique de travail basé sur les architectures microservices, il est essentiel d'accorder une attention particulière à la gestion des ressources, à la sécurité, ainsi qu'à la continuité des services tout au long de la chaîne d'intégration et de déploiement continu (CI/CD). Bien que les solutions actuelles permettent de garantir, dans une certaine mesure, la sécurité, elles présentent des limites importantes, notamment en ce qui concerne la disponibilité des applications et la gestion efficace des ressources.

Des périodes d'indisponibilité fréquentes ont été observées dans plusieurs systèmes, souvent causées par des surcharges de ressources ou des défaillances. En outre, la multiplication des attaques par force brute sur ces environnements complexes accentue les vulnérabilités et compromet la stabilité des services. Ces problèmes montrent des lacunes qui empêchent d'avoir une vue d'ensemble en temps réel des applications et de prévoir efficacement les surcharges ou les tentatives d'intrusion.

Il devient alors impératif d'adopter une approche garantissant aussi une disponibilité accrue des services numériques. En mettant en œuvre cette approche, nous espérons non seulement résoudre les problèmes d'indisponibilité, mais aussi tirer parti d'améliorations significatives en matière de performance des ressources et de prévention, tant lors du développement que de la mise en production des ENTs [5].

3 Objectifs

Ce travail vise à implémenter un système de surveillance des conteneurs d'applications en architecture microservice appliqué aux environnements numériques de travail des institutions universitaires afin d'en assurer la disponibilité et la sécurité. Pour cela, nous devons atteindre les objectifs spécifiques suivants :

- Comprendre les principes et les pratiques du DevSecOps et du CI/CD dans le contexte du déploiement des applications microservices dans les conteneurs Docker.
- Explorer les différentes techniques et outils de surveillance des microservices conteneurisés, ainsi que leur applicabilité dans le cadre du CI/CD.
- Faire l'état de l'art des solutions de surveillance spécifiques, en mettant en avant ses avantages et ses limites par rapport aux autres solutions disponibles sur le marché.
- Implémenter un système de surveillance des microservices répondant aux exigences des ENT Universitaires.
- Développer des métriques et seuils de surveillance adéquats pour détecter les anomalies, vulnérabilités et erreurs potentielles dans les applications microservices.
- Tester et évaluer le système de surveillance à travers des scénarios de déploiement et des charges de travail représentatifs.
- Analyser les résultats obtenus et mesurer l'efficacité du système de surveillance pour la détection précoce des problèmes de sécurité, de disponibilité et de performance.

La mise en œuvre d'un système de surveillance des microservices pour les ENT vise à offrir de multiples avantages. Ces avantages incluent la réduction des temps d'indisponibilité, une gestion proactive des incidents, l'optimisation des performances en identifiant les goulets d'étranglement, ainsi qu'une diminution des coûts grâce à une meilleure prévention des problèmes de sécurité et de performance. Ces objectifs sont étroitement alignés sur les besoins des institutions universitaires pour garantir un ENT sécurisé et disponible.

4 Organisation du document

Ce document est structuré en trois chapitres. Le premier chapitre est consacré à un état de l'art portant sur les concepts liés au DevSecOps, à la présentation des architectures microservices, à la surveillance des applications microservices ainsi qu'à l'analyse et à la justification des choix techniques effectués. Le deuxième chapitre aborde la mise en œuvre de notre solution. Enfin, le troisième chapitre présente les tests réalisés, les résultats obtenus et les perspectives envisagées.

Chapitre 1

Etat de l'art

Introduction

Dans ce chapitre, nous analysons les principes fondamentaux de la surveillance des microservices. Cela nous permet de mieux comprendre la problématique. Nous présentons ensuite les différentes catégories d'outils et de technologies utilisés dans la surveillance, le développement sécurisé et les opérations. Nous comparons également les différences entre ces outils pour sélectionner ceux adaptés à notre application. Enfin, nous examinons l'architecture de l'application et celle de son système de surveillance. Ces analyses constituent une base solide pour mettre en place notre solution et réaliser les tests.

1.1 Concepts fondamentaux

1.1.1 Du DevOps au DevSecOps

1.1.1.1 Le DevOps

Le **DevOps** promeut la collaboration entre développement logiciel et administration des infrastructures. Historiquement, la séparation des équipes entraînait des problèmes lors du passage en production. Le **DevOps** vise à automatiser et intégrer les processus, mettant l'accent sur l'autonomie, la communication et l'automatisation. Le processus comprend plusieurs étapes [4] :

1. **Planification** : Définition des objectifs et analyse des exigences.
2. **Développement** : Écriture et validation du code source.
3. **Intégration** : Intégration des modifications dans un référentiel avec tests.
4. **Déploiement** : Déploiement du code intégré sur les environnements.
5. **Surveillance** : Collecte des métriques pour évaluation des problèmes.
6. **Amélioration** : Optimisation des performances et de la fiabilité .

1.1.1.2 L'Intégration de la Sécurité dans le DevOps

Le **DevSecOps** intègre la sécurité dès les premières phases du développement, garantissant qu'elle est une priorité tout au long du cycle de vie des logiciels [2].

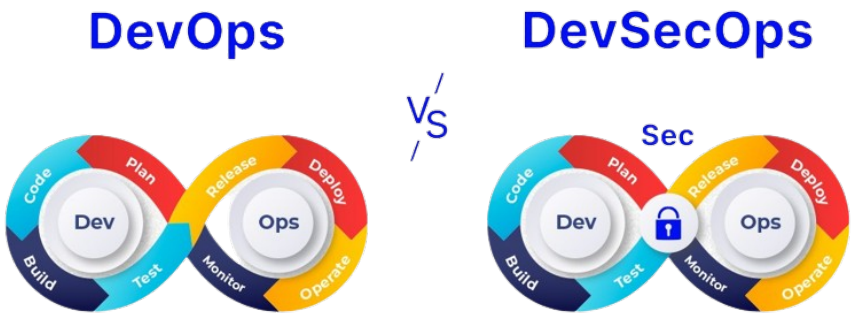


FIGURE 1.1 : Comparaison entre DevOps et DevSecOps [1]

1.1.1.3 La CI/CD

La CI/CD, ou intégration continue et déploiement continu, vise à rationaliser le cycle de développement. La CI implique l'automatisation de l'intégration et des tests, tandis que le CD englobe l'intégration, les tests et le déploiement des modifications. Ces pratiques reposent sur des pipelines CI/CD automatisés, favorisant la collaboration entre équipes et réduisant les bugs [5].

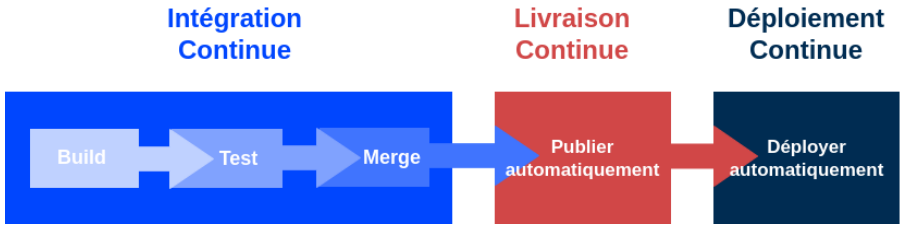


FIGURE 1.2 : Approche CI/CD

1.1.1.4 Les architectures Monolithiques et les architectures Microservices

Les architectures monolithiques et microservices représentent deux approches distinctes pour concevoir des applications logicielles [3].

Dans une architecture **monolithique**, l'application est une seule entité, regroupant toutes ses fonctionnalités dans un code source unique. Cette simplicité initiale peut se complexifier avec la croissance de l'application, rendant la mise à l'échelle difficile, car l'ensemble doit être mis à l'échelle plutôt que des parties spécifiques [6].

En revanche, les architectures **microservices** décomposent l'application en services indépendants, chacun responsable d'une fonctionnalité spécifique. Chaque microservice encapsule une fonctionnalité métier et communique avec d'autres via des APIs définies. Cela favorise la **flexibilité**, la **scalabilité** et la **résilience**, permettant aux services d'être mis à l'échelle indépendamment, mais requiert une gestion complexe des interactions entre services [6] .

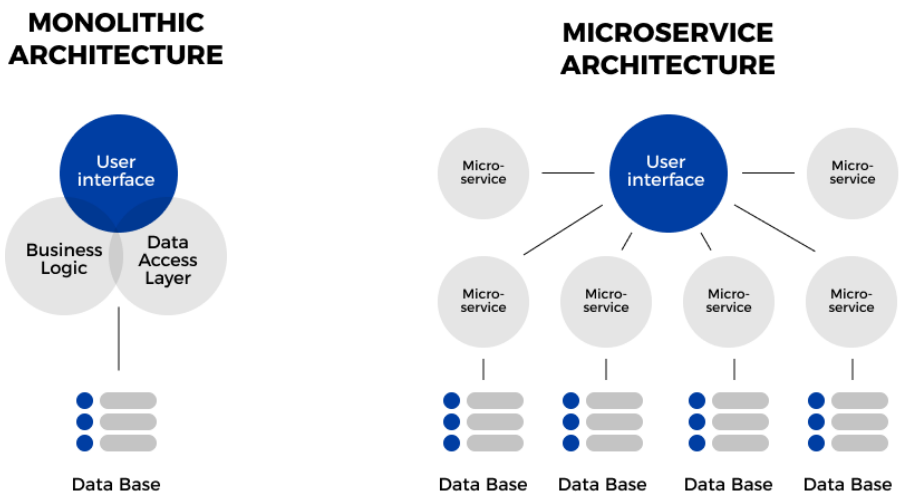


FIGURE 1.3 : Comparaison entre Architecture Monolithique et Microservices [2]

Avec l'adoption croissante des microservices, une surveillance appropriée est essentielle pour maintenir les performances, la fiabilité et la sécurité des applications distribuées.

1.1.2 Surveillance des Microservices

La surveillance des microservices consiste à collecter, analyser et visualiser les données générées par ces services afin de détecter rapidement les problèmes potentiels. Son objectif est de garantir que le système répond à des exigences telles que la disponibilité, la fiabilité et la tolérance aux pannes [7]. Elle fournit également des informations exploitables aux développeurs et équipes opérationnelles pour faciliter l'amélioration continue et la prise de décision. Les principaux composants de la surveillance des microservices comprennent la **collecte de données** provenant de journaux et de métriques, l'**agrégation et le stockage** dans des bases, ainsi que la **visualisation** à travers des tableaux de bord. De plus, des **alertes** peuvent être configurées sur des seuils prédéfinis, tandis que des outils de **diagnostics et dépannage** aident à identifier les causes profondes des problèmes [7].

1.1.3 Docker et la Conteneurisation

Docker est une plateforme qui simplifie le déploiement d'applications en utilisant des conteneurs. Ces conteneurs encapsulent le code et les dépendances dans un environnement isolé, favorisant la portabilité. En effet, Docker se distingue par sa conteneurisation, offrant une utilisation efficace des ressources.

1.1.4 Présentation de l'ENT Aplus LMS

Aplus LMS est un système de gestion de l'apprentissage open source interopérable et extensible, qui offre aux enseignants une plateforme flexible leur permettant de personnaliser entièrement leur matériel d'apprentissage en combinant différents types de contenu de manière adaptable.

Une information importante à noter est que **Aplus LMS est l'ENT d'apprentissage actuellement utilisée à l'Université d'Aalto et à l'Université de Tampere**. Cela en fait une solution éprouvée dans des environnements universitaires de renom [8] .

Aplus LMS est conçu pour être extensible. L'écosystème comprend plusieurs microservices qui communiquent entre eux par des APIs. Les responsabilités sont réparties à l'aide d'interfaces afin que les différents services et modules puissent être modifiés et repensés individuellement.

1.1.4.1 Fonctionnalité de l'application Aplus LMS

Aplus LMS propose une large gamme de fonctionnalités pour les enseignants et les étudiants. Les enseignants peuvent créer et évaluer divers devoirs, développer du contenu interactif et organiser des cours de manière flexible. Les étudiants, quant à eux, peuvent suivre des cours en ligne, s'évaluer mutuellement, soumettre des devoirs, travailler en groupe, demander de l'assistance et suivre leur progression via des outils intégrés. Aplus LMS prend également en charge les formats de fichiers ouverts, le contrôle de version avec Git, la détection de plagiats, ainsi que des évaluations manuelles et automatiques. Son architecture flexible permet l'intégration de nouveaux services d'exercices personnalisés.

1.1.4.2 Architecture et Composants Principaux

L'architecture de Aplus LMS contient plusieurs microservices disposant de plusieurs fonctions variées :

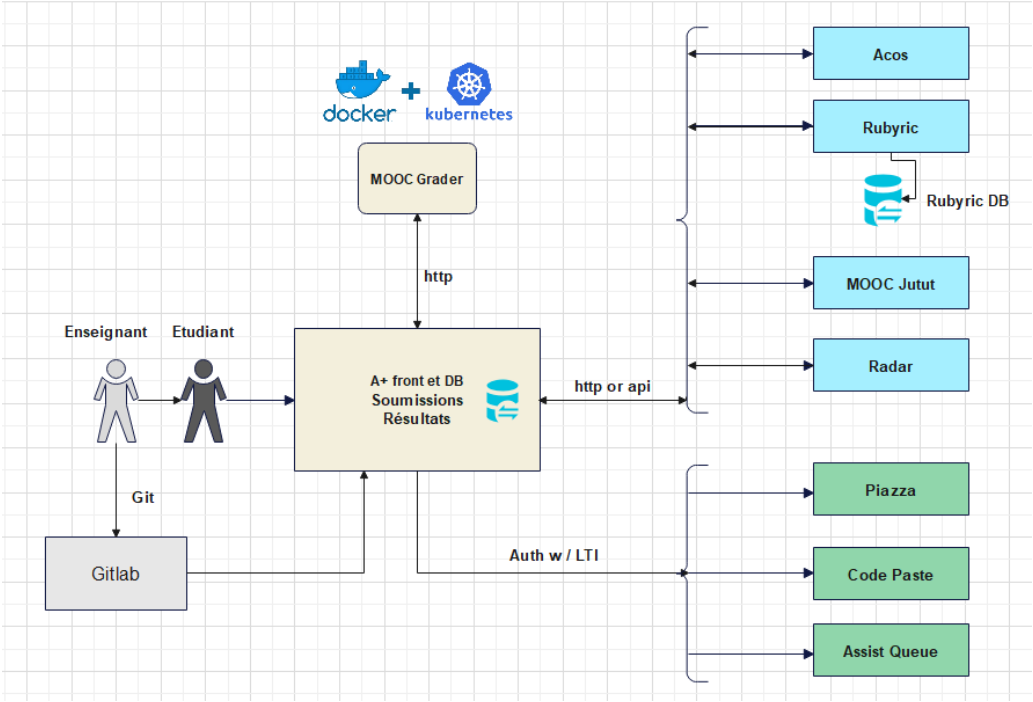


FIGURE 1.4 : Architecture de Aplus LMS

- **Le Portail Aplus** : Il constitue l'interface principale pour les utilisateurs de l'ENT, responsable de l'authentification des utilisateurs, de la centralisation des définitions de cours, de la récupération du matériel pédagogique, et de l'enregistrement des soumissions et des notes dans la base de données ;
- **Le Serveur Aplus Radar** : Il gère l'analyse automatique de similarité pour détecter les cas de plagiat dans les soumissions des étudiants ;
- **Mooc Grader (Évaluateur)** : Il fournit un cadre de notation automatique pour divers types d'affectations, avec une intégration Docker pour exécuter le code de notation fourni par le personnel enseignant ;
- **Le Serveur Aplus ACOS** : Il héberge du contenu d'apprentissage intelligent pour une distribution flexible à travers différents systèmes de gestion de l'apprentissage ;

- **Serveur MOOC Jutut** : Il offre une fonctionnalité interactive de rétroaction et de journal d'apprentissage ;
- **Le Serveur Rubyrique** : Il fournit une évaluation manuelle basée sur des rubriques pour les soumissions de fichiers avec un guide d'évaluation ;
- **Le Serveur Rubyricdb** : Représente le serveur de base de données central pour stocker les informations pertinentes liées aux évaluations .

D'autres microservices, appelés utilitaires, peuvent être ajoutés selon les besoins. Nous pouvons citer les utilitaires tels que Neuvontajono, Koodisäilö, la plateforme d'évaluation par les pairs, l'Aide à la notation (Grading Helper), Piazza, et Assist Queue. Ces microservices interagissent entre eux à travers des interfaces bien définies, favorisant la flexibilité et l'extensibilité du système.
Chaque service de L'ENT Aplus LMS est indépendant et spécialisé dans une fonction spécifique. Le tableau suivant montre les langages ou technologies des composants principaux des microservices :

TABLE 1.1 : Principaux microservices de Aplus LMS et leurs langages de programmation

Microservices	Aplus front	Gradder	Acos	Jutut	Raddar	Rubyric
Languages	Python (Django)	Python (Django)	Javascript (Express js)	Python (Django)	Python (Django)	Ruby

1.1.5 Outils de Surveillance des Microservices

Dans un environnement conteneurisé, dynamique et distribué, la surveillance des microservices revêt une importance cruciale pour garantir la disponibilité, les performances et la fiabilité des applications. Plusieurs solutions sont disponibles pour répondre à ce besoin croissant. Voici un aperçu succinct de quelques-unes de ces solutions qui jouent un rôle essentiel dans la surveillance des microservices :

- **Prometheus** : Un système de surveillance open-source conçu pour surveiller les environnements cloud natifs, y compris les microservices.
- **Sysdig** : Une plateforme de surveillance et de sécurité pour les environnements cloud et conteneurisés.
- **ELK Stack** : Elasticsearch, Logstash, Kibana est une suite logicielle open-source utilisée pour la collecte, le stockage et la visualisation de données.
- **New Relic** : Une plateforme de surveillance des performances des applications qui fournit une visibilité sur les applications, l'infrastructure et l'expérience utilisateur.
- **Datadog** : Une plateforme de surveillance cloud-native qui offre une visibilité sur l'ensemble de l'environnement informatique, des applications aux infrastructures.

1.1.6 Analyse des outils de surveillance des microservices

Dans le cadre de notre projet, une étape essentielle consiste à sélectionner les outils de surveillance des microservices les plus appropriés pour notre environnement. Cette sélection a été guidée par une

analyse approfondie des solutions disponibles sur le marché, en tenant compte de nos besoins spécifiques en matière de surveillance des applications microservices dans un environnement conteneurisé. Nous avons dressé un tableau comparatif des principales solutions de surveillance des applications microservices, présentant leurs caractéristiques et fonctionnalités clés, ainsi que leur pertinence pour notre projet. Voici ci-dessous le dit tableau :

TABLE 1.2 : Comparatif de quelques solutions de monitoring

Solutions	**Prometheus**	**Sysdig**	**ELK Stack**	**Datadog**
Installation	Facile	Facile	Complexe	Facile
Évolutivité	Oui	Oui	Oui	Oui
Surveillance	Systèmes, conteneurs et applications	Conteneurs et Cloud	Systèmes, et conteneurs	Cloud
Compatibilité avec Docker	Hautement compatible	Compatible	Compatible	Compatible
Sécurité	Moyenne	Avancée	Avancée	Moyenne
Support	Communautaire	Communautaire	Communautaire	Inclus
Suivi des performances	Très élevée	Moyenne	Moyenne	Moyenne
Métriques personnalisables	Facile	Facile	Complexe	Moyennement Facile
Gestion des journaux	Nécessite une configuration avancée	Intégrée	Hautement performante	Supportée
Portabilité	Windows, Linux	Linux, Windows	Windows, Linux	Version cloud
Licence	Open source	Open source	Open source	Propriétaire
Tarification	Gratuite	Version cloud payante	Version cloud payante	Payante
Avantages	Configuration facile, Traitement en temps réel.	Surveillance puissante des conteneurs et du cloud.	Analyse et visualisation flexible des logs	Surveillance complète

1.1.7 Motivation du Choix et Présentation

1.1.7.1 Présentation de Prometheus

Prometheus est une plateforme open source de surveillance et d'alerte conçue pour les environnements cloud natifs et les architectures basées sur les microservices. Son modèle *pull* lui permet de récupérer les métriques des cibles exposant un point de terminaison HTTP, puis de stocker ces données dans une base de séries chronologiques. Cette approche modulaire et flexible permet d'interroger les métriques à l'aide de PromQL et de configurer des alertes, facilitant ainsi la surveillance des microservices avec fiabilité, évolutivité et simplicité d'utilisation.

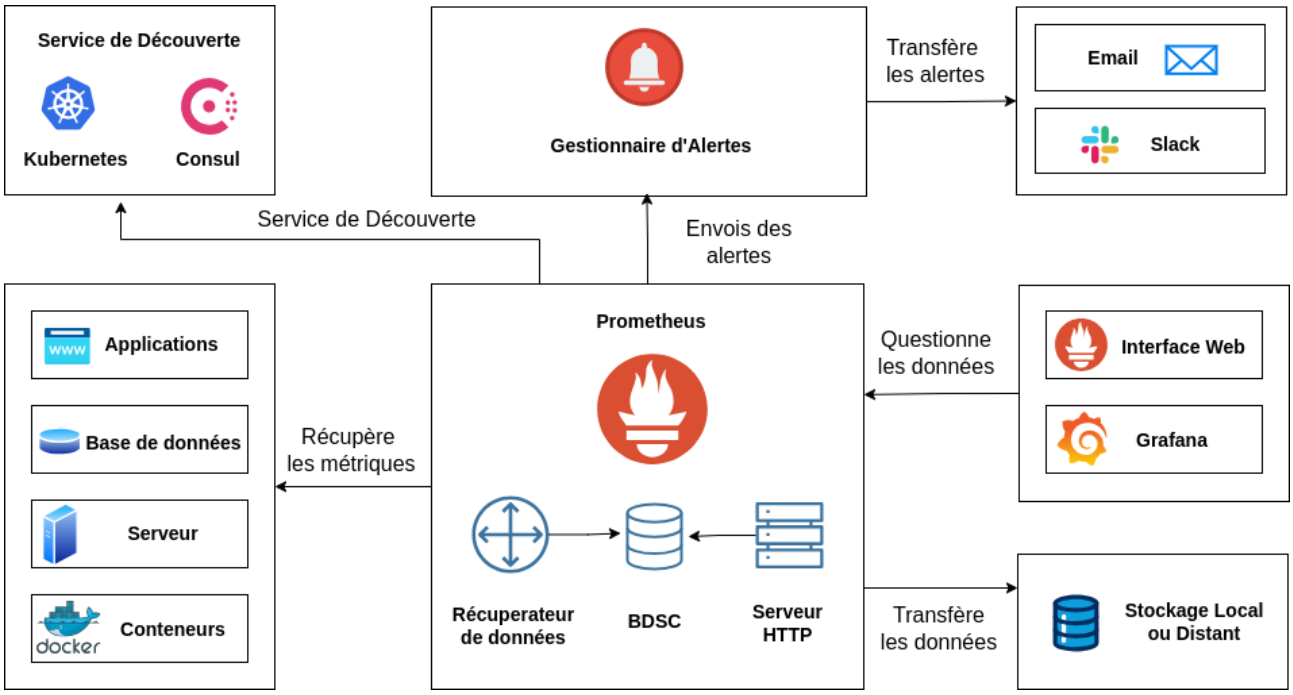


FIGURE 1.5 : Architecture de Prometheus

De ce diagramme, nous comprenons que le **serveur Prometheus** constitue le cœur du système. Les **exportateurs** collectent des données spécifiques à surveiller et les rendent disponibles à Prometheus. L'**interface utilisateur web** permet de visualiser ces données. Le **gestionnaire d'alertes** envoie des notifications en cas de problèmes ou de dépassements de seuils. Enfin, le **service de découverte** identifie et ajoute dynamiquement des cibles à surveiller [9].

Prometheus fonctionne selon un modèle en trois étapes : collecte, stockage et interrogation des métriques. Il **collecte les métriques** activement via des exportateurs configurés. Les données sont ensuite **stockées** dans une base de séries temporelles, prêtes à être analysées avec PromQL. Grafana complète cette solution en offrant une interface graphique pour créer des tableaux de bord personnalisés, permettant une surveillance et une analyse efficaces des données [1].

1.1.7.2 Motivation du Choix

La sélection de Prometheus pour la surveillance de l'environnement numérique de travail d'architecture microservices repose sur plusieurs avantages clés. Tout d'abord, Prometheus se distingue par sa facilité d'installation et d'utilisation. De plus, en tant qu'outil open source, Prometheus est non seulement accessible à tous, mais bénéficie également d'une large communauté de support. Combiné à Grafana, il offre une solution de surveillance robuste et flexible pour les environnements de microservices.

Prometheus excelle dans sa capacité à surveiller les systèmes et les applications en temps réel, avec des fonctionnalités flexibles de collecte de métriques personnalisables, ce qui en fait un choix idéal pour un environnement dynamique tel que celui des applications microservices. Sa compatibilité élevée avec Docker garantit également une intégration transparente.

Bien que certains inconvénients existent, tels que la nécessité d'une configuration supplémentaire pour la gestion de la sécurité, ces défis peuvent être surmontés avec une planification et une expertise appropriées.

En résumé, la combinaison de la facilité d'utilisation et de l'intégration avec Grafana fait de Pro-

metheus un choix convaincant pour répondre aux besoins de surveillance des espaces numériques de travail [10].

1.2 Choix des Métriques

Pour garantir des performances optimales, une disponibilité anticipée et une expérience utilisateur de qualité, nous surveillons plusieurs métriques personnalisées sur l'ENT Aplus, notamment :

- **la tentative de connexion**,
- **le taux de succès des requêtes**,
- **les requêtes critiques**,
- **le taux de disponibilité global du système**,
- **la charge du serveur de base de données**,
- **le nombre de requêtes SQL par unité de temps**,
- **le nombre de microservices en cours d'exécution**,
- **l'utilisation de la CPU et de la mémoire par microservice**,
- **le taux de saturation des ressources**.

Ces métriques couvrent un large éventail de domaines, allant de la performance et la disponibilité des microservices à la sécurité, la scalabilité et les coûts.

Conclusion

Ce chapitre a établi une base solide pour comprendre l'implémentation d'un système de surveillance des applications microservices dans un environnement DevSecOps. Nous avons présenté les principes du DevOps, son évolution vers le DevSecOps, ainsi que les concepts de la conteneurisation et de la CI/CD. Nous avons également défini les limites de notre application et comparé plusieurs solutions. Enfin, nous avons sélectionné les outils Grafana et Prometheus pour la surveillance de l'environnement.

Chapitre 2

Mise en œuvre

Introduction

Dans ce chapitre, nous avons présenté en détail l'architecture de surveillance de l'Environnement Numérique de Travail (ENT). Cette analyse nous a permis de mieux comprendre les éléments constitutifs de cet environnement. Nous avons également procédé à la mise en place et à la configuration de la solution de surveillance pour garantir la disponibilité et la stabilité de l'ENT. Ces étapes ont été essentielles pour assurer un fonctionnement optimal et prévenir d'éventuelles interruptions de service.

2.1 Environnement de surveillance

Afin de mieux monitorer les microservices de l'ENT, nous avons réalisé l'utilisation de deux serveurs virtuels : le premier est le serveur de supervision et le second est le VPS qui héberge l'Environnement de Travail Numérique Aplus.

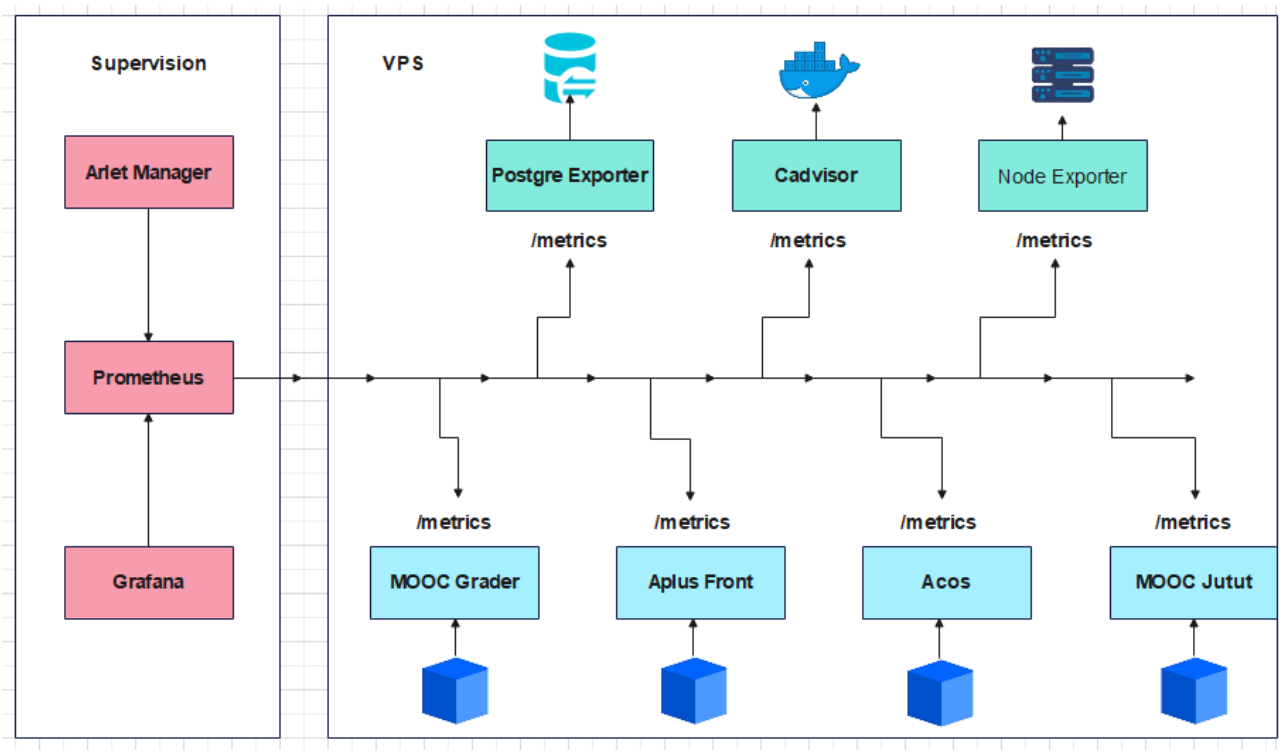


FIGURE 2.1 : Architecture de l'environnement

L'architecture est structurée autour de plusieurs composants clés pour la supervision et le suivi des microservices. Ces composants comprennent le gestionnaire d'alertes, Prometheus et Grafana pour la supervision, qui sont donc installés sur le serveur de surveillance, ainsi que les exportateurs (PostgreSQL Exporter, cAdvisor et Node Exporter) et l'ENT, hébergés sur le VPS, pour le suivi des métriques.

Le fonctionnement est assez linéaire : les exportateurs et les microservices collectent les métriques depuis les environnements et les exposent au format Open Metrics sur la route /metrics afin que Prometheus puisse récupérer les métriques. Ce dernier analyse et stocke ces données. Grafana peut alors accéder à ces informations stockées pour la visualisation et l'analyse détaillée. Le gestionnaire d'alertes supervise l'ensemble du processus, s'assurant que toutes les anomalies ou problèmes potentiels sont identifiés en temps réel.

2.2 Installation et Configuration de l'ENT Aplus LMS

Pour faire l'installation et la configuration de l'ENT, nous allons commencer par la connexion SSH sur le VPS. Ensuite, nous allons faire une mise à jour et une installation de Docker et de Docker-compose, puisque l'ENT Aplus tourne sur Docker. Il nous suffira juste d'entrer les commandes suivantes :

```
sudo apt update && (wget https://get.docker.com | sh)
sudo apt install docker-compose
sudo usermod -aG docker $USER &&  rm index.html
```

Après l'installation, nous allons fermer notre session SSH puis nous effectuons une nouvelle connexion afin de vérifier si les installations sont effectives. Nous allons alors créer un répertoire

mémoire pour cloner les fichiers nécessaires à la mise en place de l'ENT depuis le dépôt GitHub du projet accessible ici avec la commande suivante :

```
mkdir memoire && cd memoire &&
git clone https://github.com/apluslms/aplus-manual.git
```

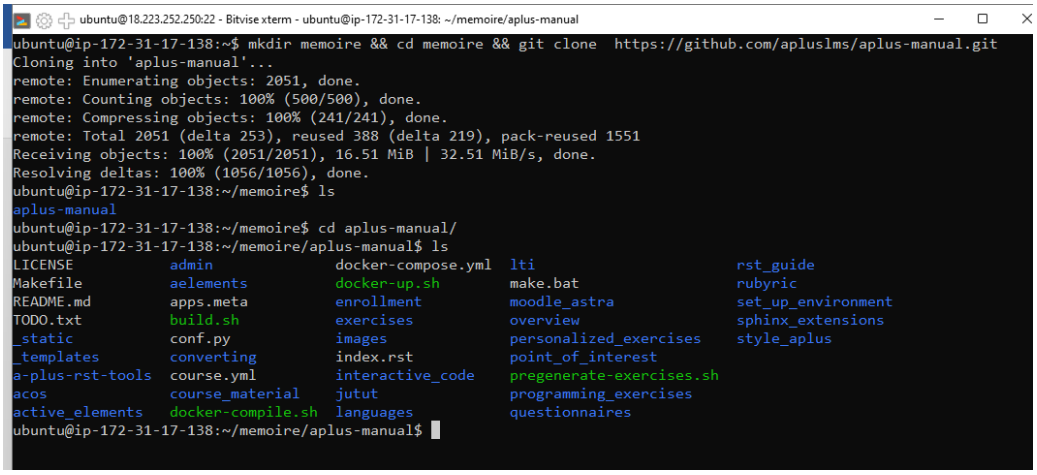


FIGURE 2.2 : Récupération du code source de Aplus LMS depuis le dépôt

Afin d'installer quelques fichiers manquants et les compiler, nous entrons les commandes suivantes :

```
git submodule update --init && ./docker-compile.sh
```

Enfin, nous exécutons le fichier `./docker-up.sh` pour lancer le système A-plus, puis il sera disponible sur le navigateur à l'adresse du serveur et sur le port 8000.



FIGURE 2.3 : Vérification de l'état de nos microservices

Par défaut, des utilisateurs sont créés et mis à notre disposition :

- `root:root`
- `teacher:teacher`
- `student:student`
- `assistant:assistant`

Nous avons donc accès à des fonctionnalités différentes en fonction des droits des utilisateurs auquel on se connecte. Par exemple, le compte du professeur a les droits de modifier les cours, de créer des cours, de récupérer les soumissions, d'assigner les notes et plein d'autres actions. Les comptes étudiants peuvent effectuer des actions mais celles-ci sont limitées.

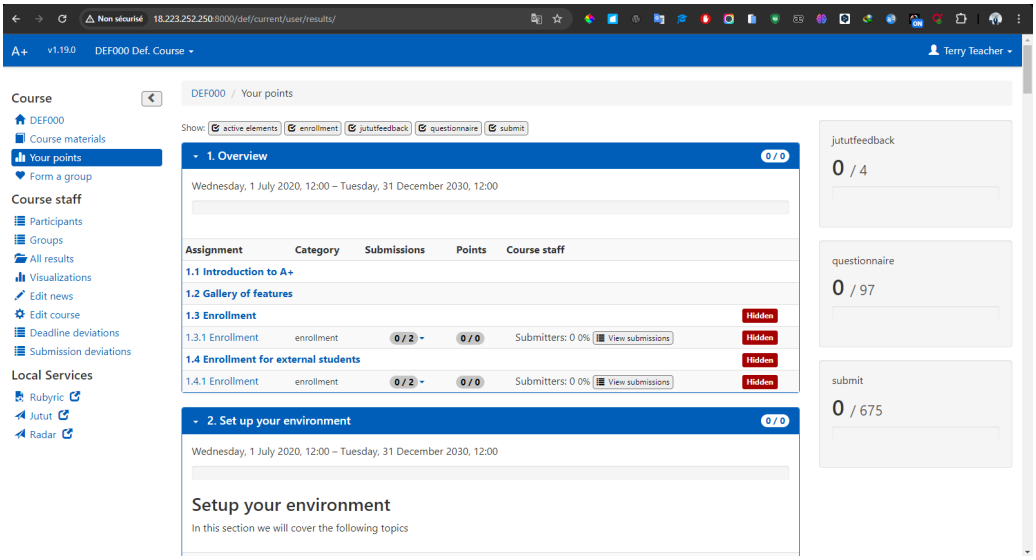


FIGURE 2.4 : Interface après connexion à Aplus LMS

2.3 Installation et Configuration de la Solution de Monitoring

La mise en place de la solution de monitoring se fera en deux étapes : la première étape est celle de l'installation de la solution de monitoring (Prometheus et Grafana) sur le serveur de monitoring et la seconde étape concerne la mise en place des exporteurs sur les différents points du serveur VPS pour l'ENT.

2.3.1 Installation et Configuration de Prometheus et Grafana

Nous allons nous connecter au serveur de surveillance par SSH, installer Docker et Docker-compose, puis installer la version 2.51.1 de Prometheus et la version 10.2.2 de Grafana avec Docker-compose. Ensuite, nous allons récupérer les fichiers de configuration avec la commande :

```
mkdir memoire && cd memoire &&
git clone git@github.com:Tob229/ent-aplus-monitoring.git
```

Nous avons eu deux sous-répertoires qui sont `ent-monitoring` et `instance-monitoring` qui sont respectivement propres à la configuration de l'ENT (`ent-monitoring`) et le serveur de monitoring (`instance-monitoring`).

- Dans le répertoire de configuration du serveur de monitoring, nous avons :
  - Le `docker-compose.yml` qui est le point d'entrée de toute la configuration.

```yaml
prometheus:
  image: prom/prometheus:v2.51.1
  container_name: prometheus
  volumes:
    - ./prometheus:/etc/prometheus
    - prometheus_data:/prometheus
  command:
    - '--config.file=/etc/prometheus/prometheus.yml'
    - '--storage.tsdb.path=/prometheus'
    - '--web.console.libraries=/etc/prometheus/console_libraries'
    - '--web.console.templates=/etc/prometheus/consoles'
    - '--storage.tsdb.retention.time=200h'
    - '--web.enable-lifecycle'
  restart: unless-stopped
  expose:
    - 9090
  networks:
    - monitor-net
  labels:
    org.label-schema.group: "monitoring"

alertmanager:
  image: prom/alertmanager:v0.26.0
  container_name: alertmanager
  volumes:
    - ./alertmanager:/etc/alertmanager
```

FIGURE 2.5 : Docker-compose de Prometheus et Grafana

Ce fichier de configuration contient les services essentiels pour la surveillance de l'ENT comprenant Prometheus, Alertmanager, Grafana et Caddy. Il crée un réseau interne, "monitor-net", pour la communication entre les services. Chaque service est configuré avec des volumes pour le stockage des données et des paramètres spécifiques.

– Le fichier `prometheus.yml` du répertoire `prometheus`.

```yaml
  scrape_interval: 5s
  static_configs:
    - targets: ['18.117.179.158:7770']

- job_name: 'aplus_cntcadvisor'
  scrape_interval: 5s
  static_configs:
    - targets: ['18.117.179.158:7771']

- job_name: 'aplus_mooc_grader'
  metrics_path: /metrics
  static_configs:
    - targets: ['18.117.179.158:8080']

- job_name: 'aplus_frontend'
  metrics_path: /metrics
  static_configs:
    - targets: ['18.117.179.158:8000']

- job_name: 'aplus_mooc_jutut'
  metrics_path: /metrics
  static_configs:
    - targets: ['18.117.179.158:8080']
```

FIGURE 2.6 : Configuration des cibles à scrapper avec Prometheus

Ce fichier `prometheus.yml` contient la configuration de Prometheus pour la surveillance des différents microservices de l'ENT.

- La section `alerting` configure l'envoi d'alertes à un gestionnaire d'alertes [11].
- Afin que Prometheus puisse reconnaître les alertes et les envoyer, des règles prédéfinies doivent être configurées pour la détection de ces alertes. C'est en ce sens que le fichier `alerts.rules` du répertoire `prometheus` intervient.

- Le fichier `config.yml` contient les paramètres d'émetteur et du récepteur des mails ou des notifications par rapport aux alertes.
- Le répertoire `grafana` contient des fichiers nécessaires à la configuration de Grafana, des fichiers tels que la configuration de la source de données Prometheus, ainsi que la configuration des tableaux de bord.

Pour démarrer le système de monitoring, nous allons nous mettre dans le répertoire `instance monitoring` et lancer la commande : `docker-compose up -d`. Cette commande va nous permettre de démarrer la système de monitoring, ce qui va nous permettre de démarrer des instances des conteneurs en prenant en compte toute nos configurations.

Le système de monitoring a bien été mis en place, nous pouvons vérifier cela en allant sur le navigateur en tapant l'URL et le port :

- Prometheus est accessible par le port 9090 et le nom d'utilisateur et le mot de passe par défaut est `admin`.

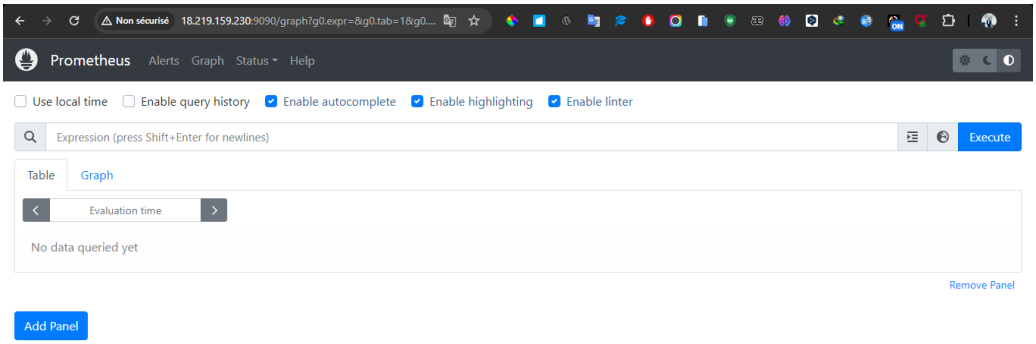


FIGURE 2.7 : Connexion à Prometheus

- Alertmanager est disponible sur le port 9093.
- Grafana est accessible sur le port 3000.

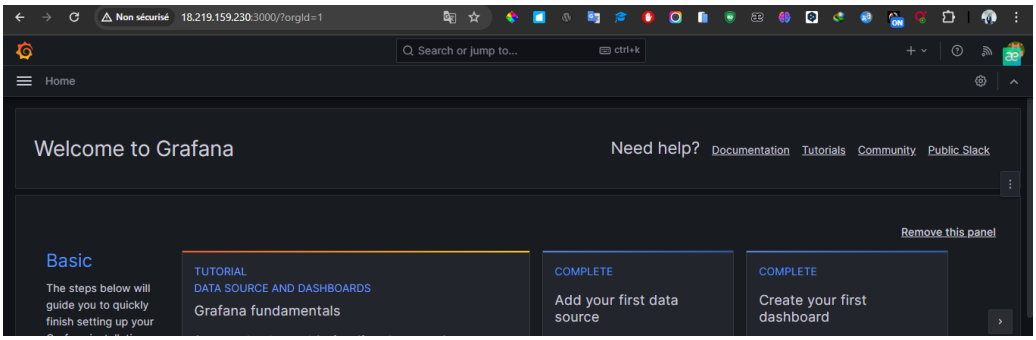


FIGURE 2.8 : Connexion à Grafana

2.3.2 Exposition des Métriques sur l'ENT

2.3.2.1 Configuration des Exporteurs

Pour que Prometheus puisse contacter nos points d'exposition des métriques (que nous avons définis dans les configurations plus haut) afin de récupérer les métriques, nous devons configurer ces points d'exposition sur le serveur de l'ENT. Pour cela, nous allons procéder en deux étapes :

- La première étape consiste à utiliser les exporteurs pour exposer les métriques concernant l'utilisation des ressources du serveur, des conteneurs Docker, des microservices qui sont sur le VPS, et enfin la base de données PostgreSQL.

- La deuxième étape consiste à modifier le code source des microservices afin d'y inclure les bibliothèques de Prometheus, ce qui nous permettra d'exposer les métriques sur une route définie et de créer par la suite nos propres métriques qui seront récupérées par Prometheus.

2.3.2.2 Exporteurs

Pour installer les exportateurs, plutôt que de les installer manuellement, nous allons utiliser la version conteneurisée de ces exportateurs. Ainsi, nous allons créer un fichier docker-compose pour y placer les configurations.

Nous configurons un conteneur Node Exporter pour exporter les métriques système sur la route /metrics. Nous utilisons des volumes montés et des paramètres pour spécifier le chemin et exclure les points de montage du système de fichiers.

```
    image: prometheuscommunity/postgres-exporter:v0.15.0
    container_name: postgres-exporter
    restart: unless-stopped
    ports:
      - "7772:9187"
    environment:
      - DATA_SOURCE_NAME=postgresql://rubyric:rubyric@18.117.179.158:5432/rubyric?sslmode=disable
#     depends_on:
#       - postgresql
    labels:
      org.label-schema.group: "monitoring"
    networks:
      - monitoring_network

  cadvisor:
    image: gcr.io/cadvisor/cadvisor:v0.49.1
    container_name: cadvisor
    privileged: true
    devices:
      - /dev/kmsg:/dev/kmsg
    volumes:
      - /:/rootfs:ro
```

FIGURE 2.9 : Configuration de Postgre-exporteur et de cadvisor

Nous avons ici deux services : le premier déploie un conteneur exportateur de métriques PostgreSQL, en spécifiant l'adresse de la base de données, les ports exposés et les étiquettes de groupe de surveillance. Le second lance un conteneur cAdvisor pour surveiller les conteneurs Docker, avec des volumes montés et des ports exposés pour l'accès aux métriques. Pour démarrer ces configurations, nous allons exécuter la commande `docker-compose up -d`.

2.3.2.3 Modification du Code Source des Microservices

Étant donné que les microservices sont écrits dans différents langages, nous allons modifier le code source de ces microservices conteneurisés. Deux solutions possibles s'offrent à nous : la première solution consiste à concevoir le code source de l'application et à le monter dans le conteneur dans le répertoire précis qui sera utilisé. La deuxième solution est de modifier le code source après avoir apporté les modifications, puis de construire nous-mêmes l'image du microservice.

- **Aplus front, grader, jutu et radar :**

Pour monitorer Aplus front, nous avons cloné le code source de Aplus front sur le dépôt GitHub de Aplus dans notre serveur au niveau de notre répertoire mémoire avec la commande suivante :

```
mkdir codesource-aplus && cd codesource-aplus &&
git clone https://github.com/aplusLms/a-plus.git && cd a-plus
```

Le microservice Aplus front est écrit en Python, donc nous allons modifier le code source pour y inclure les modules de Prometheus. Nous avons ajouté les dépendances dans le fichier requirements.txt et ensuite, dans le répertoire Aplus, nous allons ajouter les réglages de Prometheus dans le fichier setting.py.

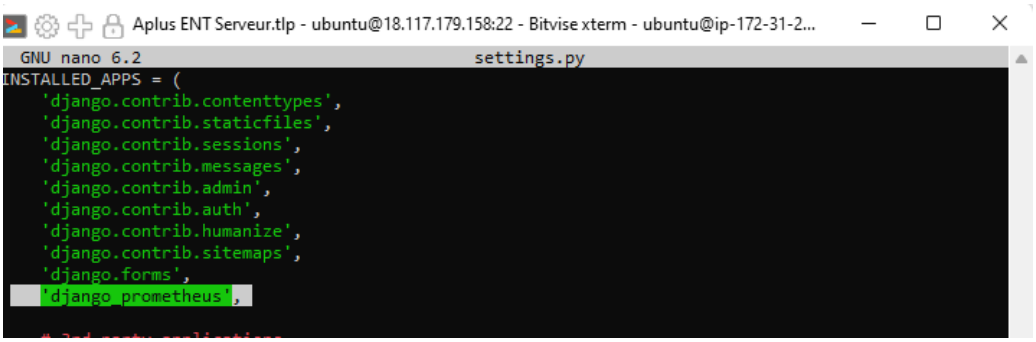


FIGURE 2.10 : Ajout du module dans les paramètres de Aplus front

Ces configurations ajoutent le module `django_prometheus` au microservice Django, qui permet de collecter des métriques pour la surveillance et la visualisation[6].

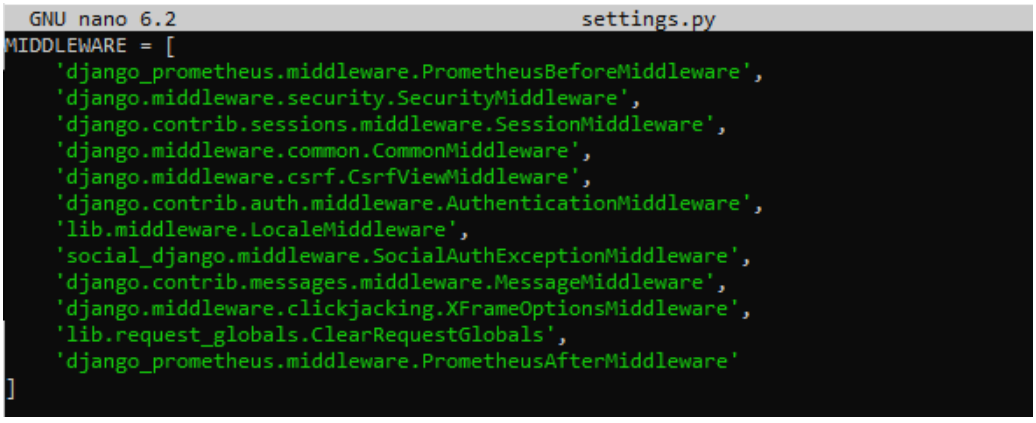


FIGURE 2.11 : Ajout du middleware de prometheus

Le middleware associé est intégré avant et après le traitement des requêtes pour capturer des données telles que les temps de réponse et les erreurs puis ensuite, nous ajoutons les URLs `django_prometheus` aux points de terminaison du microservice dans le fichier urls.py : Cela ajoutera un nouveau point de terminaison qui est `18.117.179.158:8000/metrics`.

Nous ferons les mêmes ajouts pour tous les autres microservices écrits en Python avec Django : grader, jutu et radar.
Après avoir modifié le code source en y intégrant la configuration pour exposer les métriques

des microservices, nous allons monter les répertoires contenant les codes sources des microservices dans les volumes des conteneurs des microservices.

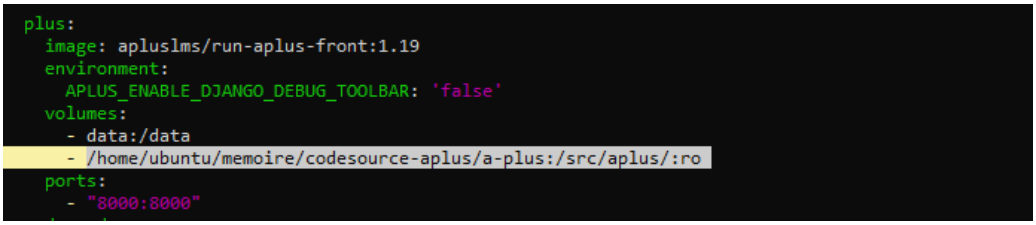


FIGURE 2.12 : Montage du code source de Aplus front

- **Acos :**

  Pour le microservice ACOS, nous allons dans un premier temps récupérer le code source depuis le dépôt du projet. Ensuite, nous allons modifier le code source pour ajouter les dépendances et les configurations de Prometheus, puis nous allons pusher ce code vers notre propre dépôt afin de concevoir l'image du conteneur ACOS intégrant directement les métriques dans l'image du microservice ACOS.

  Ensuite, dans le fichier app.js, nous avons pris soin d'importer la librairie prometheus ainsi que le middleware de prometheus, nous configurons notre microservice pour qu'il utilise ce middleware [12].



FIGURE 2.13 : Ajout du middleware de prometheus

Aussi, nous créons un nouveau point de terminaison qui sera appelé par l'application Prometheus lorsqu'elle analysera les métriques de notre microservice [4].

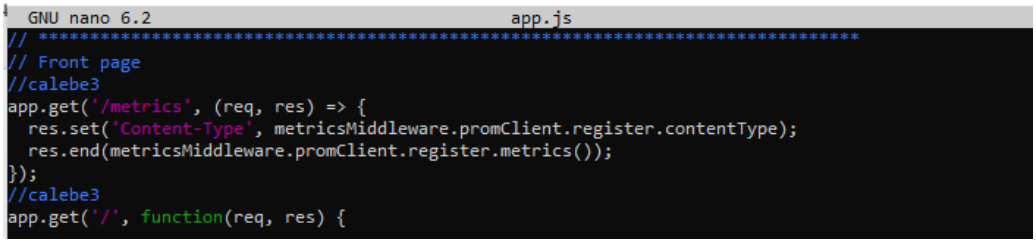


FIGURE 2.14 : Création d'un point d'exposition des métriques

Nous créons une image Docker basée sur la version "node :18-bookworm". Ensuite, nous créons un répertoire pour les journaux d'ACOS, installons ACOS à partir d'un dépôt Git spécifique, copions les fichiers de configuration et de dépendances, exposons le port 3000, et définissons la commande à exécuter lorsque le conteneur démarre, qui est `npm start`.

Nous avons utilisé un script pour builder notre image et l'envoyer dans le registre docker.

Donc, dans notre docker-compose qui démarre Aplus, nous allons utiliser notre image, puis nous allons démarrer notre ENT avec la commande docker-compose up -d pour que tous nos points d'exposition soient pris en compte par prometheus.

Sur notre serveur de supervision sur lequel nous avons installé prometheus, nous remarquons que effectivement prometheus arrive à récupérer toutes les métriques sur tous les points de récupération que nous avons définis dans nos fichiers de configuration.

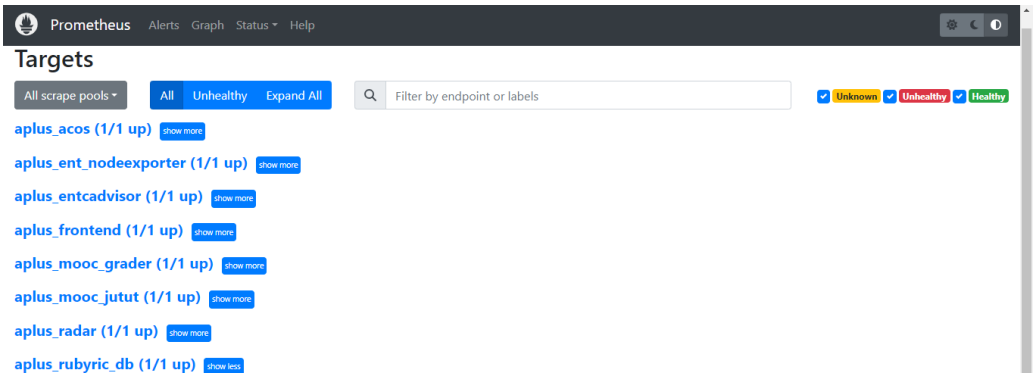


FIGURE 2.15 : Visualisation de l'état des cibles de prometheus

2.4 Intégration de la sécurité dans le processus de surveillance

2.4.1 Authentification pour cadvisor et node exporter

Pour sécuriser l'accès aux services et aux métriques exposés par nos conteneurs, nous avons configuré un serveur proxy Nginx avec l'authentification, redirigeant les requêtes vers les ports réels des autres conteneurs de manière sécurisée. En parallèle, nous avons généré un fichier .htpasswd pour stocker les informations d'identification des utilisateurs autorisés.

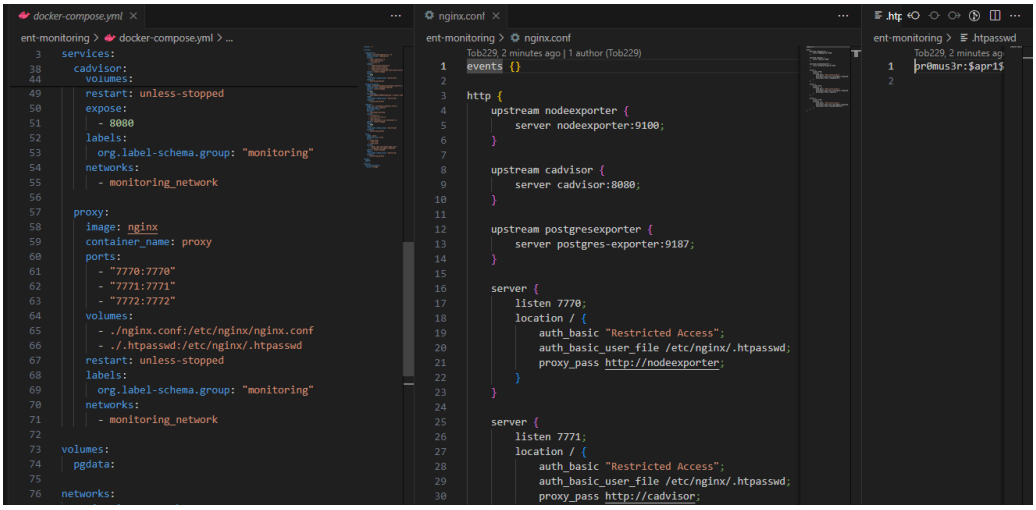


FIGURE 2.16 : Ajout d'un proxy et authentification

Dans le fichier prometheus.yml de notre configuration Prometheus, nous avons ajouté une section basic_auth pour spécifier les informations d'authentification à utiliser lors de la récupération des métriques.

2.4.2 Développement d'un mécanisme d'authentification

Pour sécuriser l'accès à une route /metrics de nos microservices, nous avons développé un middleware d'authentification. Dans notre microservice, nous avons créé un fichier middleware.py dans le même répertoire que notre fichier settings.py.

```python
def __call__(self, request):
    # Vérifier si la demande est pour /metrics
    if request.path == '/metrics':
        # Vérifier l'authentification de base
        auth_header = request.headers.get('Authorization')
        if not auth_header or not self._check_basic_auth(auth_header):
            # Demander l'authentification
            response = HttpResponse('Unauthorized', status=401)
            response['WWW-Authenticate'] = 'Basic realm="Prometheus Exporter"'
            return response

    return self.get_response(request)

def _check_basic_auth(self, auth_header):
    try:
        method, b64_value = auth_header.split(' ')
        if method.lower() != 'basic':
            return False

        value = base64.b64decode(b64_value).decode('utf-8')
        username, password = value.split(':')
        return username == settings.BASIC_AUTH_USERNAME and password == settings.BASIC_AUTH_PASSWORD
    except:
        return False
```

FIGURE 2.17 : Création du middleware d'authentification

Nous avons importé et utilisé ce middleware dans notre fichier settings.py, en ajoutant son chemin complet à la liste MIDDLEWARE. De plus, dans le même fichier settings.py, nous avons défini deux variables : BASIC_AUTH_USERNAME et BASIC_AUTH_PASSWORD, contenant respectivement le nom d'utilisateur et le mot de passe utilisés pour l'authentification. Une fois configuré, Prometheus pourra récupérer les métriques en utilisant les informations d'authentification spécifiées.

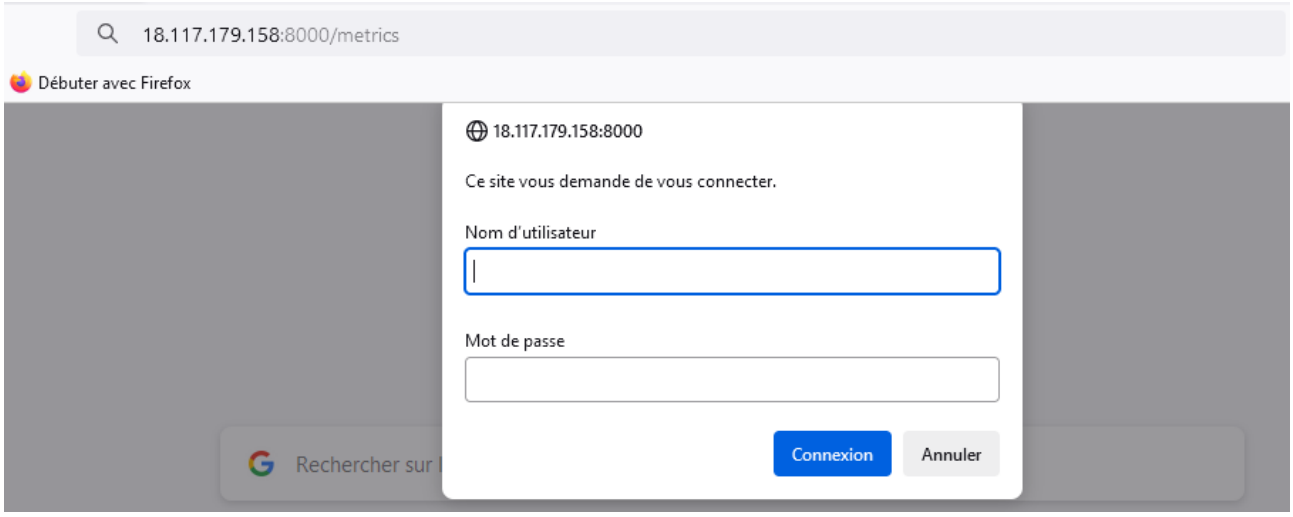


FIGURE 2.18 : Vérification du fonctionnement de l'authentification

En utilisant cette méthode, nous avons assuré que seuls Prometheus et les personnes disposant des identifiants corrects peuvent accéder à la page, que ce soit via un navigateur internet ou à travers les requêtes de Prometheus.

2.4.2.1 Utilisation d'un utilisateur non privilégié pour la récupération des métriques au niveau de la base de données

La récupération des métriques dans une base de données PostgreSQL peut être effectuée par un utilisateur non privilégié pour des raisons de sécurité. À cette fin, nous avons créé un utilisateur spécifique à l'aide d'un script pour lui attribuer des rôles intégrés comme `pg_monitor` ou `pg_read_all_stats`, lui permettant d'accéder aux vues de statistiques sans droits de superutilisateur.

Deux scripts sont utilisés pour ce processus :

- `script.sh` : Ce script se connecte à la base de données avec l'utilisateur "rubyric" et exécute le second script SQL.
- `script.sql` : Il crée l'utilisateur `aplusdb_exporter` avec un mot de passe sécurisé et lui accorde les rôles nécessaires pour accéder aux tables `pg_monitor` ou `pg_read_all_stats` [13].

```
ubuntu@ip-172-31-24-181:~/memoire/ent-aplus-monitoring/ent-monitoring/script$ ./script.sh
Password for user rubyric:
CREATE FUNCTION
 __tmp_create_user
-------------------

(1 row)

DROP FUNCTION
ALTER ROLE
ALTER ROLE
GRANT
ubuntu@ip-172-31-24-181:~/memoire/ent-aplus-monitoring/ent-monitoring/script$
```

FIGURE 2.19 : Création d'utilisateur pour les métriques

Après l'exécution de ce script, nous modifions notre fichier `docker-compose` pour utiliser l'utilisateur créé par le script, puis nous redémarrons notre configuration pour prendre les modifications.

```
    restart: unless-stopped
    expose:
      - 9187
    environment:
      - DATA_SOURCE_NAME=postgresql://aplusdb_exporter:aplusbS3curPA22w0rD@18.117.179.158:5432/postgres?sslmode=disable
    labels:
      org.label-schema.group: "monitoring"
    networks:
      - monitoring_network
```

FIGURE 2.20 : Mise à jour de la configuration de la stack des exporteurs

Ainsi, en utilisant l'utilisateur `aplusdb_exporter`, l'exporteur n'aura accès qu'aux tables spécifiées dans la base de données PostgreSQL et ne pourra pas accéder à d'autres bases de données.

2.5 Définition des métriques propres aux microservices et Création des tableaux de bord Grafana

2.5.1 Nombre total de tentatives de connexion

Pour visualiser le nombre total de connexions, nous avons utilisé une métrique qui additionne le nombre de réponses HTTP ayant le code de statut "200" et générées par la connexion (login) à partir de l'instance spécifiée de notre application (18.117.179.158 :8000) nommée "aplus_frontend".

```
sum(django_http_responses_total_by_status_view_method_total{
instance="18.117.179.158:8000", job="aplus_frontend",
method="POST",  status="200", view="login"})
```

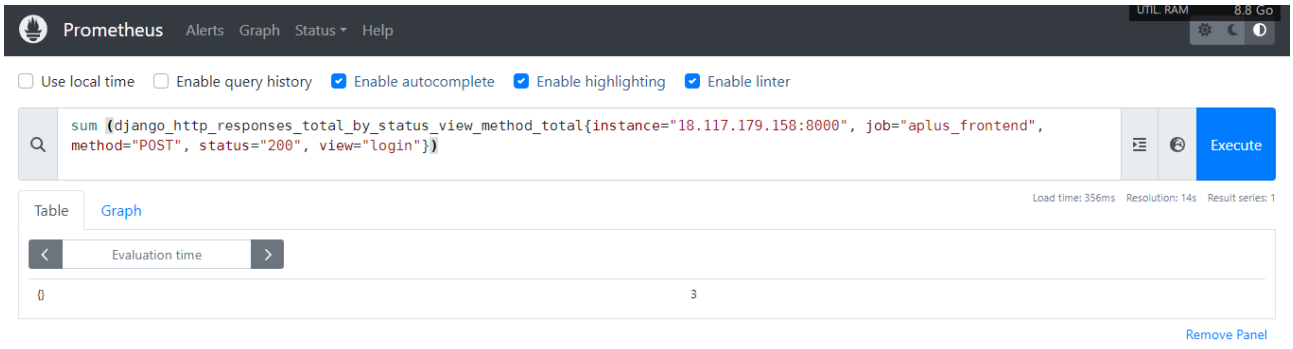


FIGURE 2.21 : Visualisation du nombre de connexions

2.5.2 Les microservices en cours de fonctionnement

Pour obtenir les microservices en cours de fonctionnement, nous avons utilisé une métrique qui sélectionne les instances de microservices (`up`) tout en excluant spécifiquement les jobs associés aux services "nodeexporter", "cadvisor", et "rubyric_db". Cela nous permet de filtrer les résultats pour ne récupérer que les instances actives des microservices spécifiques que nous souhaitons surveiller, en ignorant les instances associées à d'autres tâches ou services.

```
up{job!="aplus_ent_nodeexporter", job!="aplus_entcadvisor",
job!="aplus_rubyric_db"}
```

2.5.3 Le nombre de requêtes vers la base de données

Pour trouver le nombre de transactions commitées dans la base de données PostgreSQL, nous avons utilisé la métrique `pg_stat_database_xact_commit`.

2.5.4 Mise en place des tableaux de bord Grafana

Bien que Prometheus permette de faire des requêtes avec PromQL, la visualisation des données est facilitée par Grafana. Pour cela, nous avons d'abord créé un dossier dans Grafana à partir de la section "Tableau de bord". Ensuite, nous avons ajouté un tableau de bord et commencé à y construire des visualisations en ajoutant des panneaux. Les visualisations sont sélectionnées selon le type de données, par exemple, un graphique pour afficher les métriques sous forme visuelle.

Pour monitorer les microservices actifs, nous avons utilisé la métrique suivante :

```
up{job!="aplus_ent_nodeexporter",
job!="aplus_entcadvisor", job!="aplus_rubyric_db"}
```

2.5.5 Configuration des règles d'alertes

Dans le cadre de notre système basé sur Prometheus, nous avons consacré une attention particulière à la configuration des règles d'alertes. Ces règles sont essentielles pour surveiller efficacement les microservices et garantir leur bon fonctionnement.

Prometheus récupère les métriques et déclenche des alertes en fonction des règles définies dans le fichier alert.rules. Ces règles fixent des seuils et une fenêtre temporelle pour activer les alertes en fonction des métriques reçues. Par exemple, une alerte peut être déclenchée si l'utilisation d'un microservice dépasse 80

Ensuite, l'Alert Manager reçoit ces alertes, les dédouble, les regroupe et les envoie par les canaux de communication. Son fichier de configuration, config.yml se trouve dans le répertoire alertmanager. Pour mettre en place les règles d'alertes, nous avons configuré le fichier alert.rules.

```
name: Microservice

rules:

  - alert: MicroserviceKilled
    expr: 'time() - container_last_seen > 60'
    for: 0m
    labels:
      severity: warning
    annotations:
      summary: Microservice killed (instance {{ $labels.instance }})
      description: "A Microservice has disappeared\n  VALUE = {{ $value }}\n  LABELS = {{ $labels }}"

  - alert: MicroserviceAbsent
    expr: 'absent(container_last_seen)'
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: Microservice absent (instance {{ $labels.instance }})
      description: "A Microservice is absent for 5 min\n  VALUE = {{ $value }}\n  LABELS = {{ $labels }}"

  - alert: MicroserviceHighCpuUtilization
    expr: '(sum(rate(container_cpu_usage_seconds_total{container!=""}[5m])) by (pod, container) / sum(container_spec_cpu_
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: Microservice High CPU utilization (instance {{ $labels.instance }})
      description: "Microservice CPU utilization is above 80%\n  VALUE = {{ $value }}\n  LABELS = {{ $labels }}"
```

FIGURE 2.22 : Mise en place des règles d'alertes au microservices

Après avoir chargé les alertes, nous avons redémarré la stack Prometheus afin que Prometheus puisse prendre en compte les alertes . Ensuite, au niveau de Prometheus, dans l'onglet Alert, nous observons les alertes et leurs états. [14] [15]

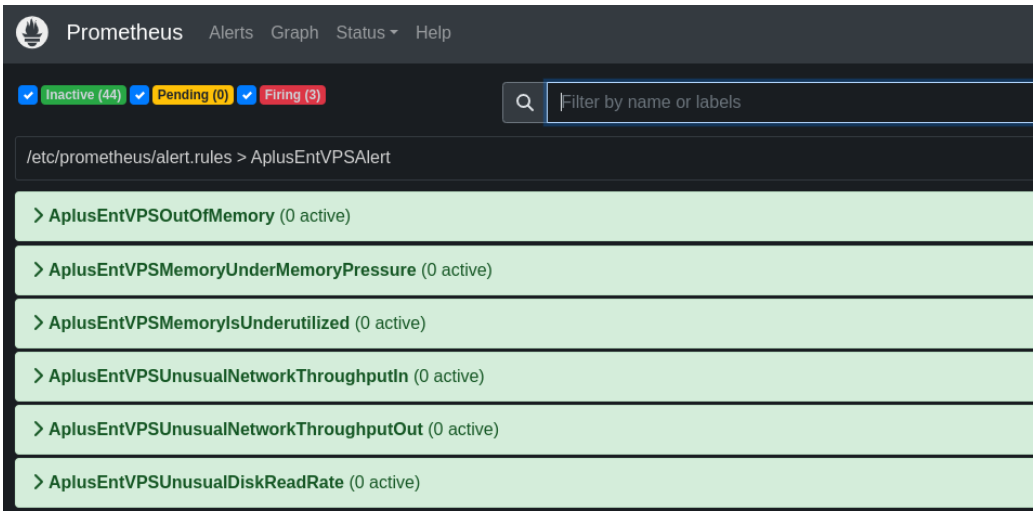


FIGURE 2.23 : Présentation des alertes et de leurs états

La couleur rouge est pour les alertes dont les règles ont été déjà déclenchées, la couleur jaune est pour les règles qui sont en train de tendre vers l'activation, et la couleur verte est pour les règles qui ne sont pas encore activées.

De même, au niveau d'Alert Manager, nous pouvons vérifier les alertes dont les mails ont été envoyés.

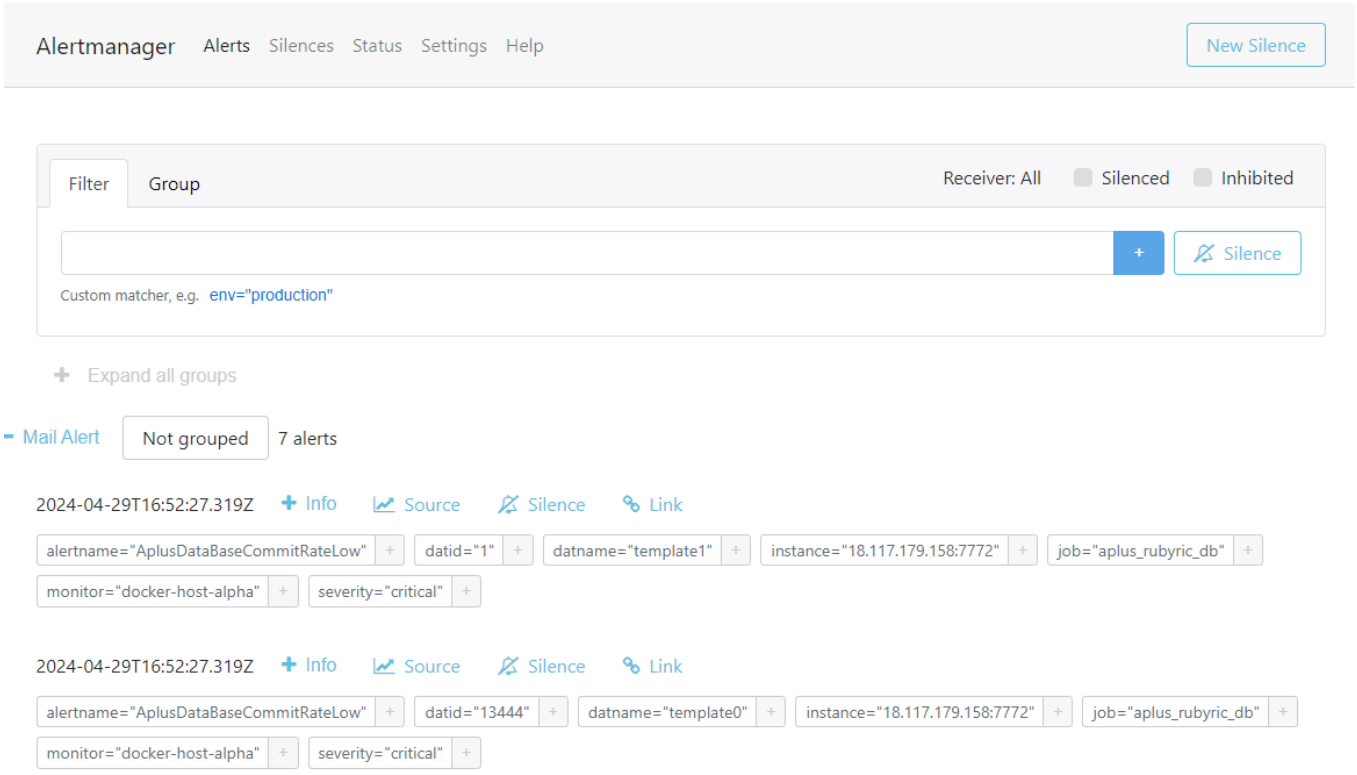


FIGURE 2.24 : Alert Manager pour l'envoi des alertes

Mais avant de recevoir les notifications par mail, nous avons configuré Alert Manager pour qu'il puisse envoyer les alertes par dans le fichier config.yml du répertoire alertmanager.



FIGURE 2.25 : Configuration de l'envoi des alertes par mail

Conclusion

Ce chapitre a permis de déployer notre solution de surveillance en installant et en configurant les outils sélectionnés précédemment avec Docker, ainsi qu'en modifiant le code source des microservices. La configuration de ces outils nous permet désormais de suivre de manière proactive et sécurisée l'état de nos microservices, tout en respectant les bonnes pratiques DevSecOps.

Chapitre 3

Résultats et discussions

Introduction

Dans ce chapitre, nous détaillons les différents tests réalisés pour vérifier la performance et la fiabilité de notre solution de surveillance. Chaque fonctionnalité a été testée conformément aux pratiques DevSecOps afin de garantir un fonctionnement optimal des outils configurés. Enfin, nous analysons en profondeur les résultats obtenus suite à ces tests, permettant ainsi de valider la robustesse de la solution mise en place.

Résultats

3.0.1 Tableau de bord

Sur Grafana, nous avons ajouté des tableaux de bord et des visualisations avec des panneaux.

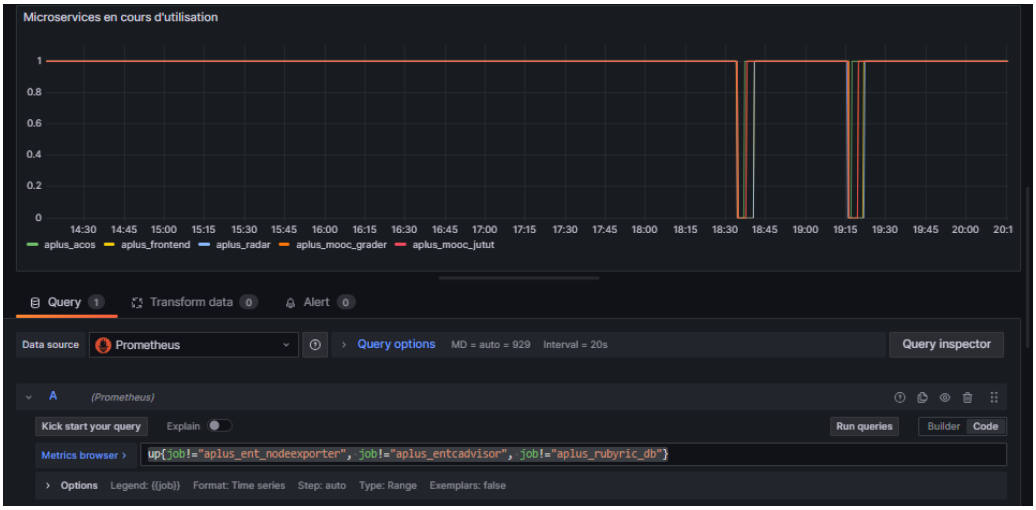


FIGURE 3.1 : Tableau de bord pour les microservices en cours d'utilisation

Le graphique montre deux périodes d'indisponibilité des microservices, de 18h30 à 18h40 et de 19h15 à 19h20. Pour dynamiser les tableaux de bord, nous avons utilisé des variables dans les para-

mètres de settings, permettant ainsi une gestion plus flexible des visualisations.

Un second tableau de bord pour surveiller les requêtes effectuées par les microservices :



FIGURE 3.2 : État des requêtes et de leur code d'erreur

Nous avons effectué un test de tentative de brute force sur la page de connexion et observé que, dans Grafana, le nombre de connexions augmente significativement. Cette activité anormale est détectée et représentée visuellement dans le tableau de bord. le nombre de connection augmente de façon anormale sur le tableaud de bord

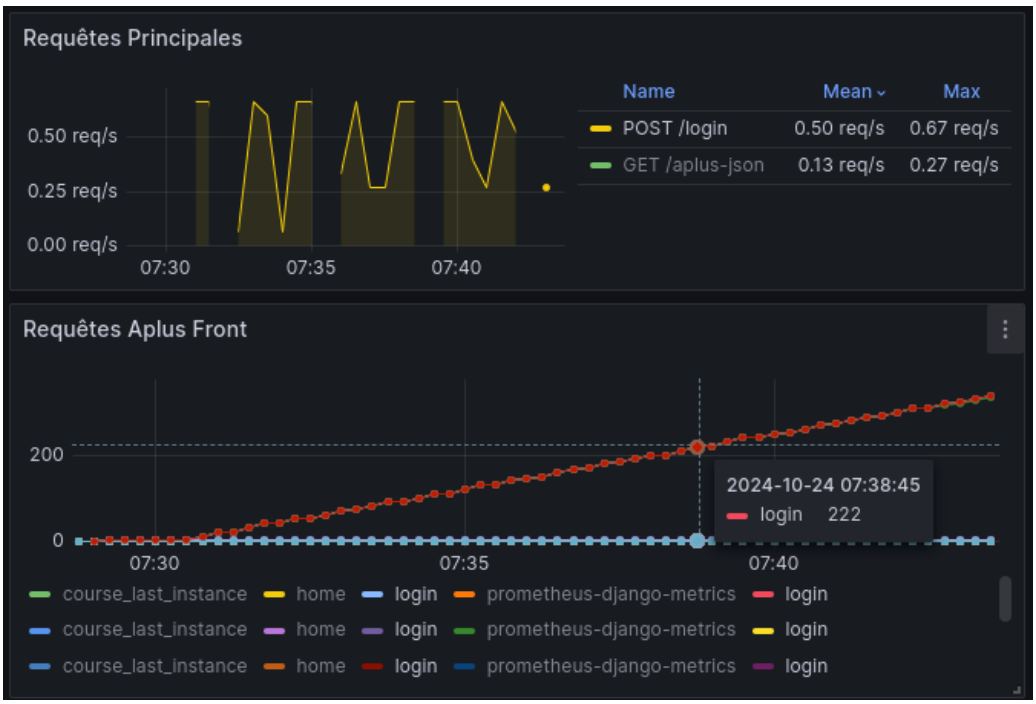


FIGURE 3.3 : Augmentation du nombre de connexions suite à une tentative de brute force

3.0.2 Réception des mails

Nous avons également testé la réception des alertes via e-mail. Les notifications sont envoyées en cas d'événements critiques, comme l'indisponibilité d'un microservice.

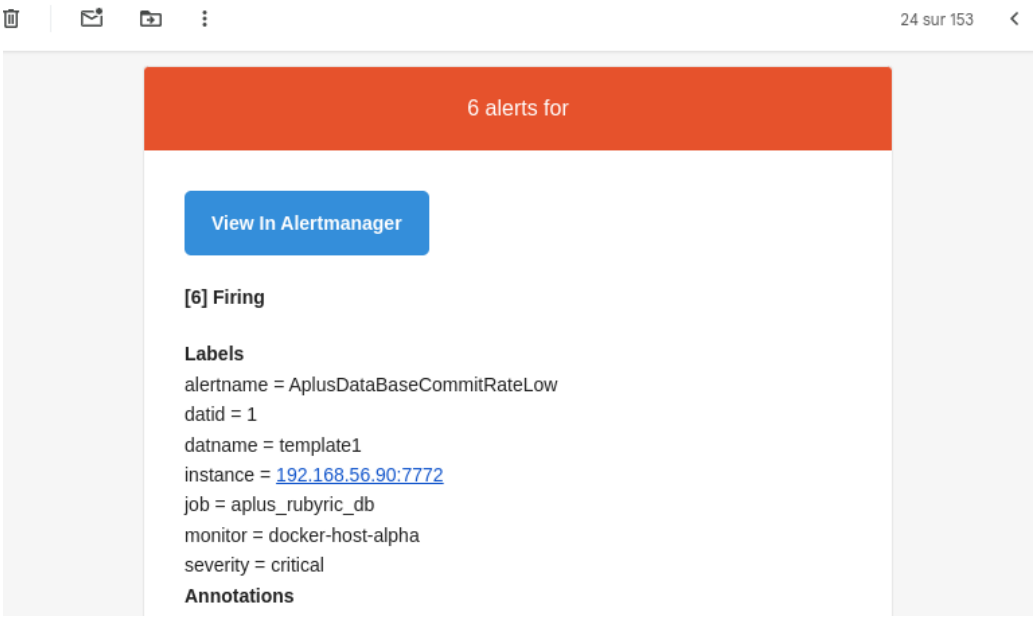


FIGURE 3.4 : Réception des alertes par mail

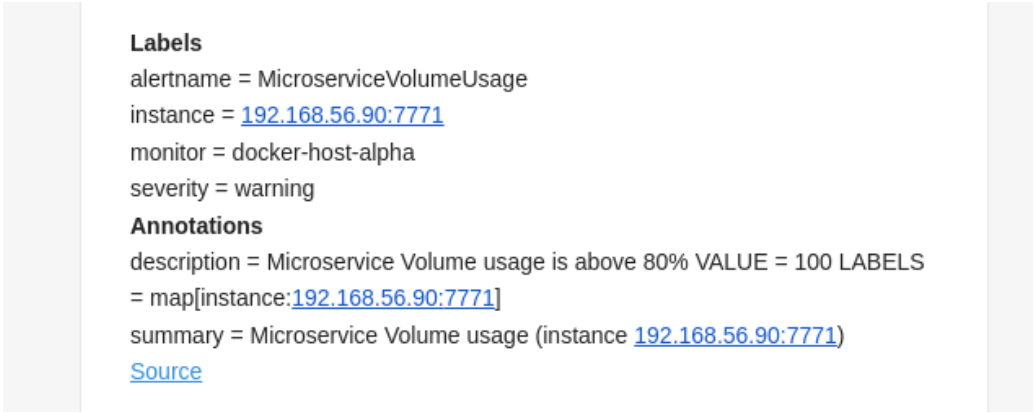


FIGURE 3.5 : Exemple d'alerte par mail

Ainsi, en cas de problème lié à l'indisponibilité d'un microservice, une alerte sera automatiquement envoyée aux administrateurs par e-mail.

Discussion

En appliquant les meilleures pratiques DevSecOps et en choisissant des outils adaptés tels que Docker et Docker Compose, nous avons conçu une solution robuste pour surveiller et gérer efficacement les microservices. Cette approche inclut l'intégration de tests rigoureux avec l'envoi successif de requêtes de connexion afin d'évaluer nos configurations, analyser les métriques et garantir le bon fonctionnement des alertes par e-mail. Les résultats obtenus ont été positifs, confirmant la fiabilité de notre démarche. Cependant, nous sommes conscients des limites actuelles du projet et travaillons activement à les surmonter. Nos efforts se concentrent sur l'amélioration continue de la sécurité, l'op-

timisation de la surveillance et le renforcement des mécanismes d'alerte. En adoptant des pratiques DevSecOps encore plus avancées, nous visons à faire évoluer notre solution pour répondre aux besoins croissants en matière de performance et de résilience.

Conclusion

Ce chapitre nous a permis de tester notre solution de surveillance. Nous avons mené des tests conformes aux pratiques DevSecOps sur les différentes fonctionnalités mises en œuvre afin de garantir leur bon fonctionnement. Les résultats obtenus démontrent la détection d'une montée en charge des requêtes, ainsi que l'augmentation de la charge des ressources, et l'efficacité du système de surveillance. Que ce soit en termes de détection des tentatives d'attaques, telles que les attaques par force brute, ou en termes de notification des administrateurs via des alertes par e-mail en cas d'indisponibilité des microservices, les tests valident la solution comme étant capable de répondre aux exigences de sécurité tout en maintenant une visibilité en temps réel sur l'état des services critiques.

Conclusion Générale et Perspectives

Ce mémoire a conduit à la mise en place réussie d'une solution de surveillance des microservices de l'ENT Aplus LMS. Cette solution de surveillance assure efficacement la disponibilité et les performances des microservices, contribuant ainsi à la détection proactive des problèmes de sécurité, de disponibilité et de performance. Les principaux résultats obtenus comprennent la création d'un environnement de surveillance des microservices, la gestion proactive des problèmes de disponibilité grâce aux alertes, et l'amélioration de la sécurité des outils utilisés en conformité avec la méthodologie DevSecOps au sein d'une CI/CD. De plus, ce mémoire nous a permis d'approfondir nos connaissances sur les concepts fondamentaux de la surveillance et du DevSecOps, ainsi que sur les outils et technologies utilisés pour mettre en place la surveillance des microservices. Pour les perspectives futures, il est envisagé d'intégrer un protocole de communication sécurisé et de mettre en place une communication sécurisée basée sur la cryptographie pour permettre une communication sans risque d'interception des informations surveillées.

Bibliographie

[1] Yannig Perré, *Prometheus et Grafana - Surveillez vos applications et composants système*, https://www.amazon.fr/Prometheus-Grafana-Surveillez-applications-composants/dp/2409031749.

[2] Yannig Perré, *DevSecOps – Développez et administrez vos services en toute sécurité*, https://www.amazon.fr/DevSecOps-D%C3%A9veloppez-administrez-services-s%C3%A9curit%C3%A9/dp/2409039189/.

[3] Sam Newman, *Monolith to Microservices : Evolutionary Patterns to Transform Your Monolith*, https://www.amazon.fr/Monolith-Microservices-Evolutionary-Patterns-Transform/dp/1492047848.

[4] Abhijeet Dhane, Atharva Josh, Chinmay Borgaonkar, Manas Deshpande, Prof.Pravin Patil : *Application Monitoring using Prometheus and Grafana*, https://ijercse.com/article/6.8%20August%202024%20IJERCSE.pdf.

[5] Cyrielle Thevenin : *Les entreprises françaises touchées par la panne mondiale*, https://www.ledauphine.com/science-et-technologie/2024/07/19/tf1-fnac-quelles-sont-les-entreprises-francaises-touchees-par-la-panne-mondiale.

[6] Sipios : *Get Started to monitor your Django application with Prometheus and Grafana*, https://medium.com/sipios/get-started-to-monitor-your-django-application-with-prometheus-grafana-in-10-minutes-dac9c0fdcf58.

Webographie

[1] Webclues. Devops vs devsecops : https://www.webcluesinfotech.com/devops-vs-devsecops-how-they-differ-and-whats-better/.

[2] OpenLegacy. Monolithic vs microservices architecture : https://www.openlegacy.com/blog/monolithic-application.

[3] education.pf. La révolution numérique : https://www.education.pf/la-revolution-numerique-en-education-larrivee-de-lent-en-polynesie-francaise.

[4] BrowserStack. Devops : https://www.browserstack.com/guide/devops-lifecycle.

[5] Cegid. Un changement de mindset pour gagner en agilités : https://www.cegid.com/fr/blog/approche-cicd-definition-et-avantages/.

[6] AppMaster. Monolitique vs microservice : https://appmaster.io/fr/blog/architecture-monolithique-vs-microservices.

[7] AppMaster. Surveillance des microservices : https://appmaster.io/fr/glossary/surveillance-des-microservices.

[8] Aplus LMS. The extendable learning management system : https://apluslms.github.io.

[9] DevOpsCube. Learn prometheus architecture : https://devopscube.com/prometheus-architecture/.

[10] Quentin. Prometheus de a à y : https://une-tasse-de.cafe/blog/prometheus/.

[11] Prometheus. Documentation officielle de alert manager : https://prometheus.io/docs/alerting/latest/alertmanager/.

[12] Austin Cunningham. Get prometheus metrics from a express.js app : https://dev.to/austincunningham/get-prometheus-metrics-from-a-express-js-app-53ck.

[13] Prometheus Community. Un exportateur de métriques postgresql pour prometheus : https://github.com/prometheus-community/postgres_exporter.

[14] Stackhero. Comment fonctionnent les alertes prometheus et comment les configurer : https://www.stackhero.io/fr-fr/services/Prometheus/documentations/Alertes.

[15] Prometheus Community. Les règles d'enregistrement : https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/.

Table des matières

Dédicace ii

Remerciements iii

Résumé iv

. iv

Abstract v

. v

Liste des figures vi

Liste des tableaux vii

Liste des acronymes viii

Introduction 1

1 Etat de l'art 3

Introduction 3

1.1 Concepts fondamentaux 3

1.1.1 Du DevOps au DevSecOps 3

1.1.1.1 Le **DevOps** 3

1.1.1.2 L'Intégration de la Sécurité dans le DevOps 4

1.1.1.3 La CI/CD 4

1.1.1.4 Les architectures **Monolithiques** et les architectures **Microservices** . 4

1.1.2 Surveillance des **Microservices** 5

1.1.3 Docker et la Conteneurisation 5

1.1.4 Présentation de l'ENT Aplus LMS 5

1.1.4.1 Fonctionnalité de l'application Aplus LMS 6

1.1.4.2 Architecture et Composants Principaux 6

1.1.5 Outils de Surveillance des Microservices 7

1.1.6 Analyse des outils de surveillance des microservices 7

1.1.7 Motivation du Choix et Présentation 8

1.1.7.1 Présentation de Prometheus 8

1.1.7.2 Motivation du Choix 9

1.2 Choix des Métriques 10

2 Mise en œuvre 11

Introduction . . . 11
2.1 Environnement de surveillance . . . 11
2.2 Installation et Configuration de l'ENT Aplus LMS . . . 12
2.3 Installation et Configuration de la Solution de Monitoring . . . 14
2.3.1 Installation et Configuration de Prometheus et Grafana . . . 14
2.3.2 Exposition des Métriques sur l'ENT . . . 16
2.3.2.1 Configuration des Exporteurs . . . 16
2.3.2.2 Exporteurs . . . 17
2.3.2.3 Modification du Code Source des Microservices . . . 17
2.4 Intégration de la sécurité dans le processus de surveillance . . . 20
2.4.1 Authentification pour cadvisor et node exporter . . . 20
2.4.2 Développement d'un mécanisme d'authentification . . . 21
2.4.2.1 Utilisation d'un utilisateur non privilégié pour la récupération des métriques au niveau de la base de données . . . 22
2.5 Définition des métriques propres aux microservices et Création des tableaux de bord Grafana . . . 23
2.5.1 Nombre total de tentatives de connexion . . . 23
2.5.2 Les microservices en cours de fonctionnement . . . 23
2.5.3 Le nombre de requêtes vers la base de données . . . 23
2.5.4 Mise en place des tableaux de bord Grafana . . . 23
2.5.5 Configuration des règles d'alertes . . . 24

3 Résultats et discussions **27**
Introduction . . . 27
3.0.1 Tableau de bord . . . 27
3.0.2 Réception des mails . . . 29

Conclusion **31**

Bibliographie **32**

Webographie **33**

Table des matières **34**