

CONSTANT AIDAM

Projet Morning News

Dossier Projet - Administrateur Système DevOps



Sommaire

[1. Compétences Couvertes Par Le Projet](#)

[2. Cahier Des Charges](#)

[3. Spécifications Techniques](#)

[4. Démarche Et Outils](#)

[5. Réalisations Personnelles](#)

[6. Situation De Travail Ayant Nécessité Une Recherche](#)



1. Compétences Couvertes Par Le Projet

- a. Automatiser le déploiement d'une infrastructure dans le cloud
 - i. Automatiser la création de serveurs à l'aide de scripts
 - ii. Automatiser le déploiement d'une infrastructure
 - iii. Sécuriser l'infrastructure
 - iv. Mettre l'infrastructure en production dans le cloud
- b. Déployer en continu une application
 - i. Préparer un environnement de test
 - ii. Gérer le stockage des données
 - iii. Gérer des containers
 - iv. Automatiser la mise en production d'une application avec une plateforme
- c. Superviser les services déployés
 - i. Définir et mettre en place des statistiques de services
 - ii. Exploiter une solution de supervision
 - iii. Échanger sur des réseaux professionnels éventuellement en anglais




2. Cahier Des Charges

Dans le cadre de la mise en place d'un environnement de développement robuste et fiable, ce cahier des charges est conçu pour répondre aux besoins spécifiques de notre projet. Il englobe la gestion du versioning, la configuration des infrastructures, l'automatisation des pipelines CI/CD, la sécurité et le monitoring. Ces éléments sont cruciaux pour assurer la stabilité, la sécurité et l'efficacité des déploiements en production et en pré-production.

a. Versioning

- i. Utilisation assidue d'un outil de ticketing
- ii. Hébergement du code source et configuration du repository
- iii. Git Flow cohérent avec les pipelines de CI/CD

b. Infrastructures

- 
- i. Schéma complet de l'infrastructure
 - ii. Déploiement en production
 - iii. Création d'une image de conteneur Docker
 - iv. Hébergement de l'image dans un registre d'images Docker privé
 - v. Déploiement en production
 - vi. Environnement de production
 - vii. Environnement de pré-production
 - viii. Sauvegarde automatique et périodique
 - ix. Nom de domaine personnalisé

c. CI / CD

- i. Build et mise en production automatisée
- ii. Automatisation de l'exécution des tests unitaires du projet
- iii. Analyse de la qualité et de la sécurité du code
- iv. Automatisation de l'exécution des tests end-to-end (E2E)
- v. Mise à jour automatique du registre d'images
- vi. Création et exécution de tests de montée en charge

d. Sécurité

- i. Respect des bonnes pratiques de configuration SSH
- ii. Mise en place des règles de pare-feu réseau (AWS / GCP ou Linode)
- iii. Mise en place des règles de pare-feu logiciel (Iptables ou UFW)

e. Monitoring


- i. Dashboard de monitoring frontend
- ii. Dashboard de monitoring backend
- iii. Dashboard de monitoring base de données

3. Spécifications Techniques

a. Livrables

En se basant sur le cahier des charges nous avons pu identifier les livrables suivants :

- i. Méthode de gestion de projet
 - Utilisation d'outils adaptés pour une gestion efficace des tickets et du versioning.
- ii. Infrastructure Cloud
 - Hébergement des environnements de production et de pré production, incluant les composants applicatifs Frontend, Backend et Base de données (BDD).
 - Déploiement d'une infrastructure robuste et sécurisée, avec des schémas détaillés de l'architecture.
- iii. Gestion du versioning



- Mise en place d'une stratégie Git Flow cohérente avec les pipelines CI/CD pour un contrôle optimal des versions du code source.

iv. Automatisation des tests et des déploiements

- Exécution automatique des tests unitaires et end-to-end (E2E).
- Automatisation de la mise en production et de la mise à jour du registre d'images Docker.
- Création et exécution de tests de montée en charge pour garantir la performance sous contrainte.

v. Sécurité

- Mise en place des bonnes pratiques de configuration SSH et des règles de pare-feu réseau et logiciel (AWS, Linode, Iptables, UFW).

vi. Monitoring

- Développement de tableaux de bord pour le monitoring frontend, backend, et base de données.
- Surveillance proactive des éléments de l'infrastructure, incluant les machines, services, bases de données, et transactions.

b. Philosophie De La Solution

Pour concevoir cette solution, nous avons pris soin d'optimiser les coûts et les performances en utilisant des services open source, gratuits ou à faible coût tels que Linode et AWS. Nous avons évité le vendor lock-in en intégrant des outils comme Cloudflare pour la gestion des certificats SSL et le CDN, ce qui permet une flexibilité dans le choix des fournisseurs. MongoDB Atlas a été choisi pour sa scalabilité et facilité

de gestion. La gestion des secrets a été centralisée avec Infisical pour une sécurité renforcée. Enfin, pour le déploiement backend, Amazon ECS avec des instances EC2 a été privilégié pour sa simplicité et ses capacités d'auto-scaling, tout en évitant des solutions plus complexes comme Kubernetes.

L'adoption d'une approche itérative et collaborative a permis à chaque membre de l'équipe de contribuer à chaque aspect du projet, d'évaluer les solutions potentielles et de se spécialiser sur des sujets précis. Cette méthode, soutenue par des pratiques DevOps rigoureuses et des outils de monitoring et de tests, garantit une solution fiable, évolutive et adaptable aux besoins futurs.

c. Schémas De L'Infrastructure Cible

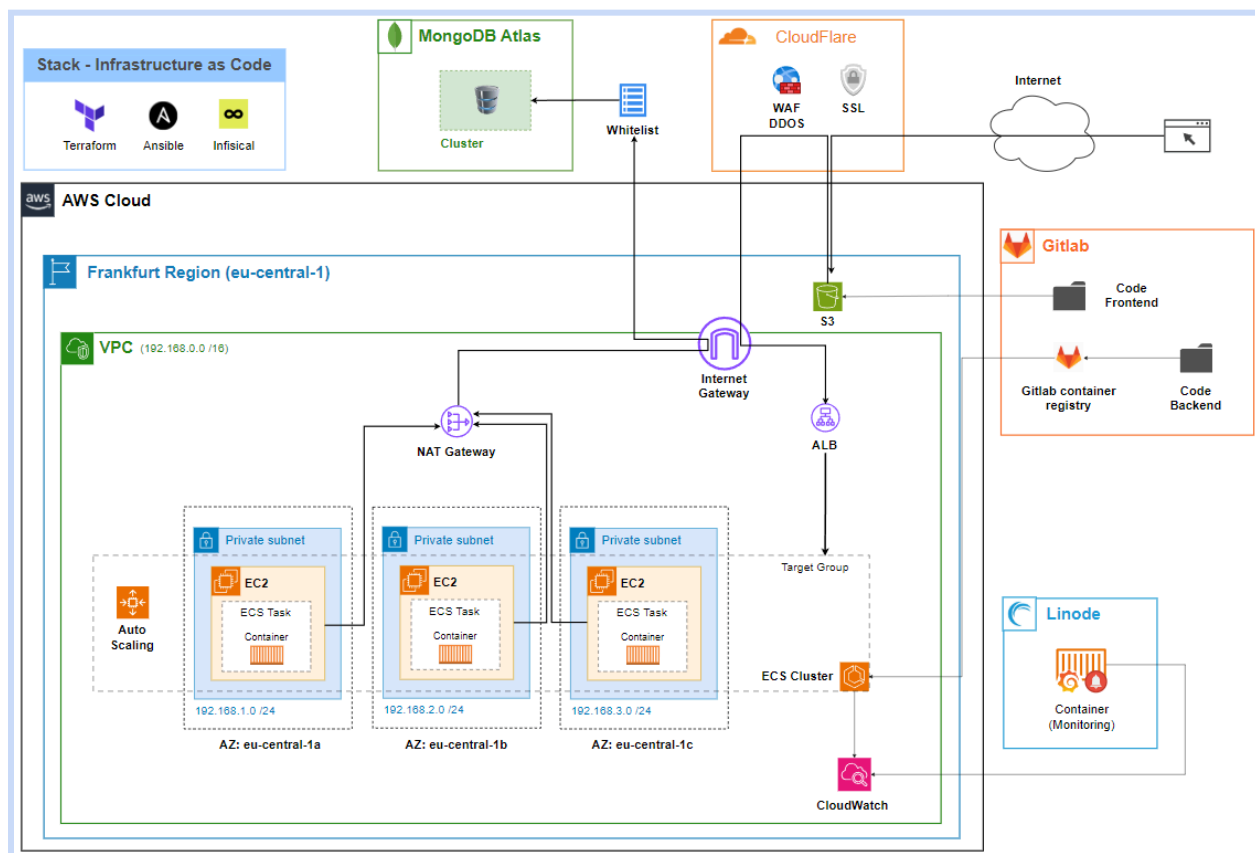


Fig. 1. Infrastructure Cible du Projet - Production | Schéma réalisé via draw.io.

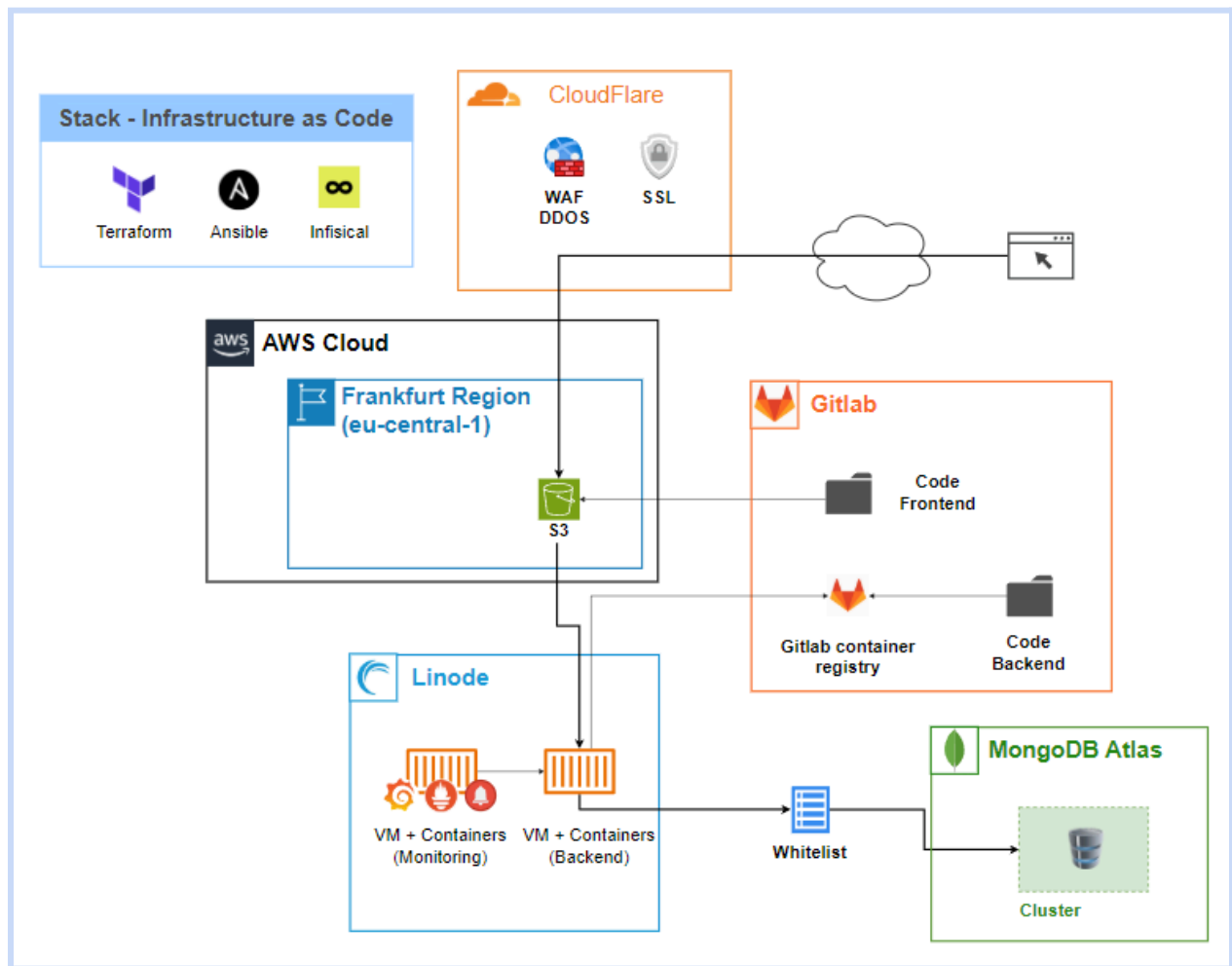


Fig. 2. Infrastructure Cible du Projet - Pré Production | Schéma réalisé via draw.io.



4. Démarche Et Outils

a. Démarche

Ce projet est le fruit d'un travail collaboratif au sein d'un groupe de trois profils DevOps, allant de débutant à junior, avec des parcours et niveaux d'expertise variés. Tous les membres de l'équipe ont suivi un bootcamp intensif de 10 semaines, se terminant par ce projet final de 2 semaines. Le but de cette initiative est de mettre en pratique les compétences acquises et d'appliquer les méthodologies DevOps dans un environnement professionnel simulé. Le projet doit être mené à bien en deux semaines, en suivant un rythme agile avec des sprints de 2 jours. Le code pour une application backend et une application frontend a été fourni sous forme de fichiers .zip, avec peu d'informations supplémentaires, ce qui renforce le défi et la nécessité d'une collaboration et d'une organisation efficace.

Dans cet ordre d'idée nous avons adopté une approche itérative basée sur des phases de *labs* et de tests. Cette approche a permis à chacun de s'intéresser à chaque partie du projet pour évaluer des solutions potentielles puis à se spécialiser sur un sujet précis retenu pour la solution de production.

Par exemple, avant d'adopter une solution de production pour le déploiement de l'application frontend, chaque membre du groupe aura créé un *proof of concept* qui aura été testé et comparé avec les autres dans des environnements de test puis de pré production.

En respectant nos ressources limitées en termes de temps, de compétences et de budget, nous avons eu pour objectif de créer une solution fiable et robuste qui suit les bonnes pratiques et que nous pouvons évaluer (et faire évoluer) en continu grâce à notre approche itérative soutenue par des solutions de monitoring et de tests.

b. Outils

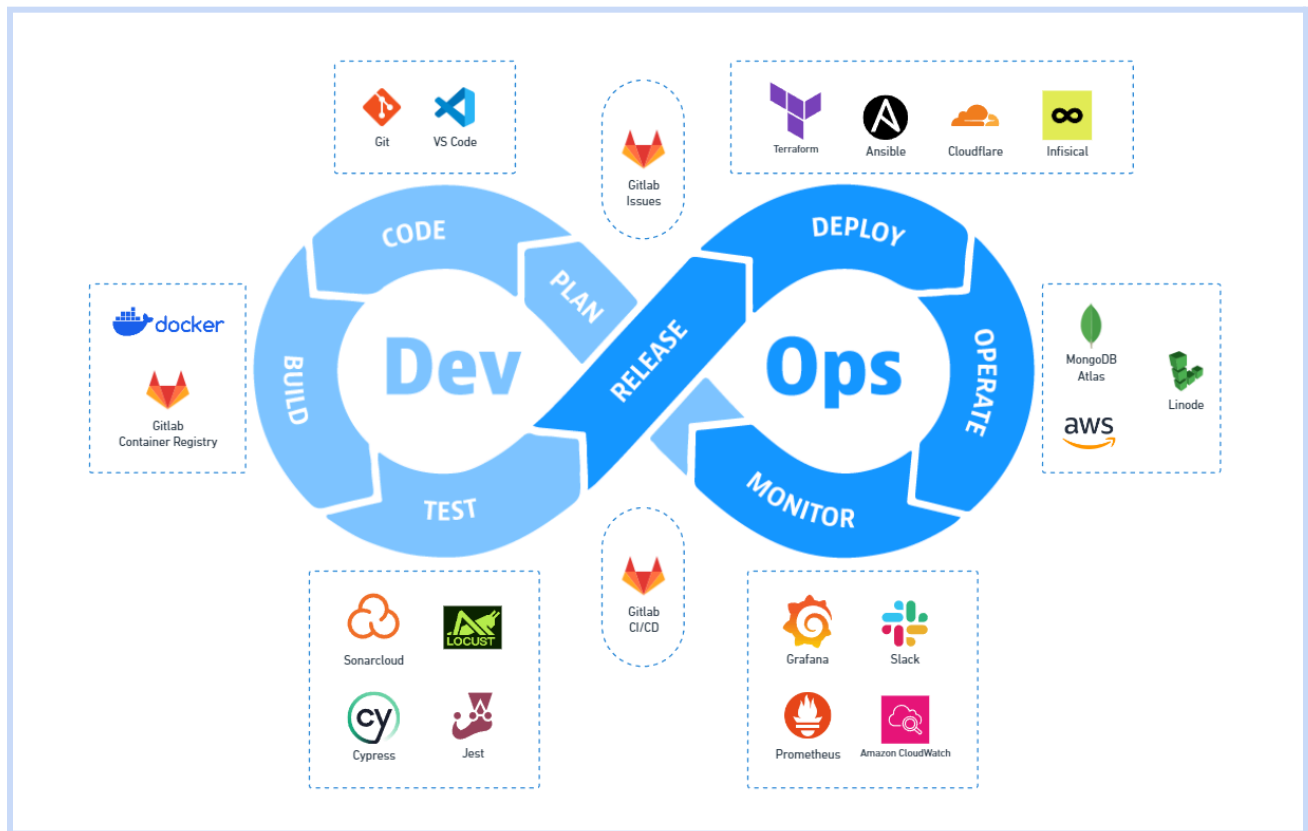



Fig. 3. Cycle DevOps et Tooling du Projet | Schéma réalisé via Whimsical.

i. Planification et Gestion

- **Gitlab Issues :**

Cet outil est utilisé pour gérer les tâches, suivre les bugs et organiser le travail collaboratif. Dans le cadre du cycle DevOps, il facilite la planification en permettant une centralisation des discussions, des descriptions des tâches et de leur avancement.

Gitlab Issues a été préféré à Jira en raison de son intégration native avec le pipeline CI/CD de GitLab, ce qui permet une transition transparente entre la gestion des tâches et le déploiement. Jira est puissant et flexible, mais



l'intégration complète de GitLab offre une expérience plus simple, cohérente et unifiée dans le contexte de notre équipe.

ii. Développement (Code & Build)

- **Git & Gitlab :**

Git est utilisé pour la gestion de version et le suivi des modifications du code source. Il permet une collaboration efficace entre les développeurs en offrant des fonctionnalités telles que les branches, les merges, et les commits.

GitLab est une plateforme intégrée qui combine la gestion du code source et l'automatisation CI/CD, ce qui réduit la complexité et augmente l'efficacité par rapport à des solutions séparées comme GitHub et Jenkins.


- **VS Code :**

Cet éditeur de code est utilisé pour écrire, modifier et tester le code. Grâce à ses nombreuses extensions, il s'intègre parfaitement aux workflows DevOps.

VS Code offre une extensibilité et une intégration supérieures avec des outils DevOps par rapport à d'autres éditeurs comme Sublime Text ou Atom. Son adoption massive et ses mises à jour fréquentes le rendent particulièrement adapté aux projets collaboratifs.

- **Docker :**

Docker est utilisé pour créer des conteneurs reproductibles qui assurent la cohérence entre les environnements de développement, de test et de production. Il joue un rôle crucial dans la phase de Build en facilitant l'emballage des applications et leurs dépendances.



Docker est préféré à d'autres solutions de conteneurisation comme Podman ou LXC en raison de son écosystème mature, de sa documentation exhaustive et de son intégration avec les outils CI/CD comme GitLab.

- **Gitlab Container Registry**

Ce registre permet de stocker et gérer les images Docker de manière centralisée. Cela simplifie le déploiement des applications conteneurisées dans le pipeline DevOps.

L'utilisation du registre de conteneurs intégré de GitLab réduit les besoins de configuration et de gestion comparé à des solutions tierces comme Docker Hub, tout en offrant des fonctionnalités de sécurité et d'automatisation intégrées.

- iii. Tests


- **SonarCloud :**

SonarCloud est utilisé pour l'analyse de la qualité du code, détectant les bugs, les vulnérabilités et les mauvaises pratiques. Cela garantit un code maintenable et sécurisé.

Par rapport à d'autres outils comme Codacy ou CodeClimate, SonarCloud offre une intégration plus robuste avec GitLab CI/CD et une couverture étendue de langages, facilitant ainsi une analyse de code complète et intégrée.

- **Cypress :**

Un outil de test de bout en bout qui permet de s'assurer que les applications web fonctionnent comme prévu. Il est intégré dans les pipelines CI/CD pour automatiser les tests.



Cypress a été choisi pour ses capacités de test en temps réel et son intégration directe avec le navigateur, surpassant des alternatives comme Selenium en termes de facilité d'utilisation et de rapidité des tests.

- **Jest :**

Jest est un framework de tests pour les applications JavaScript. Il permet d'effectuer des tests unitaires et d'intégration, s'assurant que les différentes parties de l'application fonctionnent correctement.

Jest offre une expérience de test plus rapide et plus simple par rapport à Mocha ou Chai, notamment grâce à son exécution parallèle des tests et à ses rapports de couverture de code intégrés.

- **Locust :**


Cet outil de test de charge est utilisé pour simuler un grand nombre d'utilisateurs et évaluer les performances de l'application sous contrainte.

Locust permet une grande flexibilité dans la définition des scénarios de test et une scalabilité facile par rapport à JMeter ou K6, offrant une alternative plus simple pour les tests de charge.

iv. Déploiement, Infrastructure et Exploitation

- **Terraform :**

Un outil d'infrastructure-as-code (IaC) utilisé pour définir, provisionner et gérer l'infrastructure de manière déclarative. Il s'intègre dans la phase Deploy pour automatiser le déploiement des ressources.



Terraform a été choisi pour sa compatibilité multi-cloud et son écosystème de modules riches, surpassant des alternatives comme AWS CloudFormation en termes de flexibilité et d'évolutivité.

- **Ansible :**

Ansible est utilisé pour la gestion de la configuration et l'automatisation des tâches complexes, garantissant une configuration cohérente de l'infrastructure.

Par rapport à Chef ou Puppet, Ansible offre une courbe d'apprentissage plus douce et une syntaxe YAML plus simple, ce qui facilite son adoption par une équipe avec des niveaux variés d'expertise.

- **Cloudflare :**

Cloudflare est utilisé pour la gestion DNS, la protection contre les attaques DDoS et l'amélioration des performances des applications en ligne.


Cloudflare a été préféré à AWS CloudFront ou Akamai en raison de son interface conviviale, de ses fonctionnalités de sécurité avancées et de ses options de tarification plus accessibles.

- **Infisical :**

Cet outil est utilisé pour gérer les secrets et les variables d'environnement en toute sécurité, une étape cruciale dans la gestion des configurations sensibles.

Infisical a été choisi pour sa simplicité d'utilisation et son intégration native avec les pipelines CI/CD, surpassant des solutions comme HashiCorp Vault en termes de simplicité pour des équipes avec des niveaux variés d'expertise.

- **MongoDB Atlas :**



MongoDB Atlas est une base de données NoSQL gérée dans le cloud. Elle est utilisée pour stocker les données de manière fiable et évolutive.

MongoDB Atlas a été préféré à Amazon DynamoDB ou Google Firestore en raison de sa flexibilité et de sa tarification avantageuse dans notre contexte.

- **AWS :**

Amazon Web Services est utilisé pour héberger l'infrastructure et les applications, fournissant une plateforme scalable et hautement disponible.

AWS a été choisi pour sa robustesse, sa maturité et sa large gamme de services, surpassant d'autres fournisseurs comme Google Cloud Platform ou Microsoft Azure en termes de diversité de services et de support communautaire.

- **Linode :**


Linode est une alternative d'hébergement utilisée pour des solutions cloud flexibles et économiques.

Linode a été choisi pour ses options de tarification plus abordables et sa simplicité par rapport à DigitalOcean ou Vultr, offrant une solution efficace pour des besoins de test et de développement.

v. Monitoring et Supervision

- **Grafana :**

Grafana est utilisé pour la visualisation des métriques et la surveillance des performances de l'application. Il aide à détecter les anomalies et à prendre des décisions basées sur les données.



Grafana a été préféré à Kibana en raison de sa capacité à intégrer et visualiser des données provenant de multiples sources, offrant une flexibilité et une personnalisation accrues.

- **Prometheus :**

Cet outil est utilisé pour collecter et stocker les métriques de l'infrastructure et des applications. Il s'intègre avec Grafana pour fournir une supervision complète.

Prometheus a été choisi pour sa forte intégration avec une multitude de services et sa capacité à gérer une grande quantité de métriques en temps réel.

- **Amazon CloudWatch :**

CloudWatch offre des capacités de surveillance pour les ressources AWS, en collectant des logs, des métriques et des événements pour garantir la stabilité de l'application.

CloudWatch a été choisi pour sa compatibilité native avec les services AWS, simplifiant l'intégration et la surveillance complète à un prix avantageux par rapport à d'autres outils comme Datadog.


- **Slack :**

Slack est utilisé pour les notifications en temps réel. Il s'intègre aux outils de monitoring pour alerter rapidement les équipes en cas d'incident.

Slack a été choisi pour son adoption généralisée et ses capacités d'intégration avec divers outils de monitoring et DevOps, proposant une alternative moins intrusive aux traditionnelles alertes et notifications par mail.

vi. Automatisation du Pipeline CI/CD

- **GitLab CI/CD :**



GitLab CI/CD permet d'automatiser le pipeline complet, du Build au Test, au Release et au Deploy. Chaque étape du cycle DevOps est orchestrée avec GitLab CI/CD pour garantir une livraison rapide et de haute qualité.

GitLab CI/CD est préféré à Jenkins pour son intégration directe avec GitLab, réduisant la complexité de configuration et offrant une interface utilisateur plus cohérente et intuitive pour gérer l'ensemble du pipeline DevOps.

5. Réalisations Personnelles

a. Collaboration & Versioning

Bien qu'elle peut sembler triviale, cette partie a été la plus critique pour moi, car notre groupe de trois a été constitué de manière aléatoire, et nous n'avions jamais collaboré auparavant. Nous avons des niveaux de compétences, des préférences et des sensibilités potentiellement différentes, ce qui ajoute une couche de complexité. Afin de garantir une collaboration efficace et un versioning ordonné, j'ai mis en place plusieurs initiatives clés.

i. Création de Comptes Dédiés

Pour faciliter la collaboration sans centraliser les responsabilités autour d'un seul membre, j'ai créé des comptes dédiés au nom de l'équipe pour GitLab, Gmail, Slack, Terraform Cloud et Infisical Cloud. Cela permet un partage équitable et réduit les frictions potentielles. Les comptes pour les fournisseurs cloud n'ont pas été créés spécifiquement pour des raisons de sécurité liées aux informations bancaires sensibles. L'utilisation de comptes d'équipe assure que chaque membre peut accéder aux ressources nécessaires et contribue à la transparence et à la continuité, même si un membre devait s'absenter.

ii. Organisation de l'Environnement GitLab

Pour structurer notre travail, j'ai organisé l'environnement GitLab de la manière suivante :

- **Création d'un Groupe au Nom du Projet** : Permet une gestion centralisée de toutes les ressources liées au projet, facilitant la coordination et l'administration.
- **Création des Repos** : Backend, frontend, infrastructure et project_management. Le dernier repo centralise la documentation et les informations de gestion du projet, comme les templates d'issues, ce qui standardise les pratiques et améliore l'efficacité.

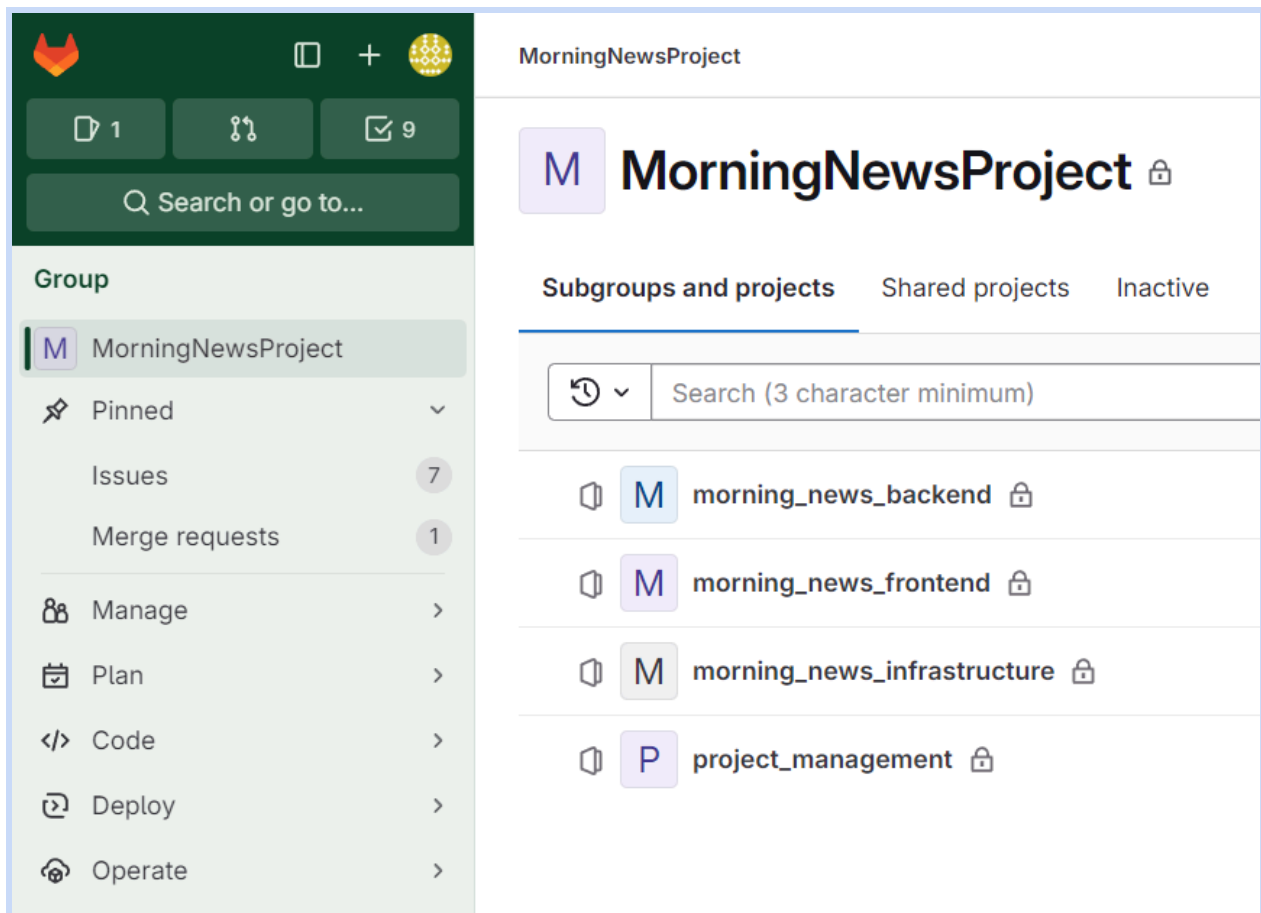


Fig. 4. Groupe Gitlab et Repos du Projet | Capture d'Écran via GitLab.

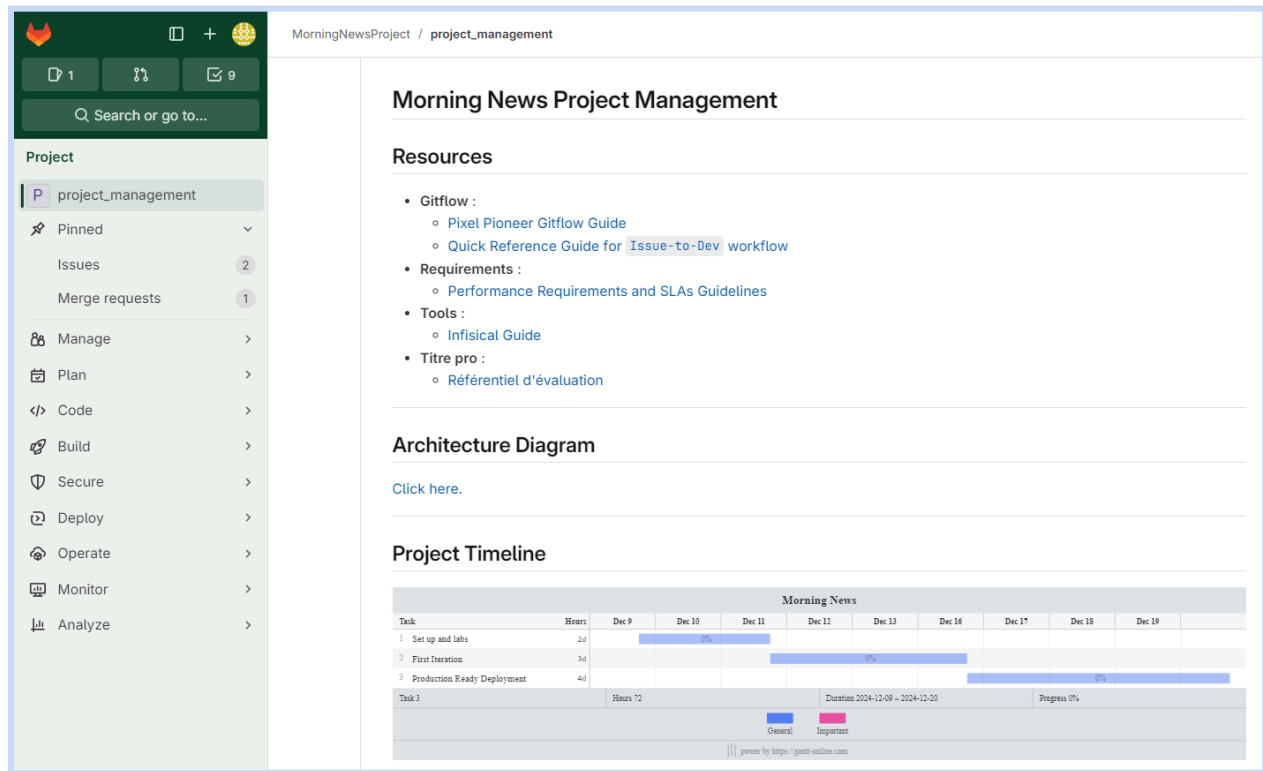


Fig. 5. Repo dédié à la Gestion de Projet et la Doc | Capture d'Écran via GitLab.

- **Labels et Issues Boards** : Création des labels pour les issues et des boards de gestion au niveau du groupe et des repos pour un suivi optimal. Cela nous aide à prioriser les tâches, suivre les progrès et assurer que rien n'est négligé.

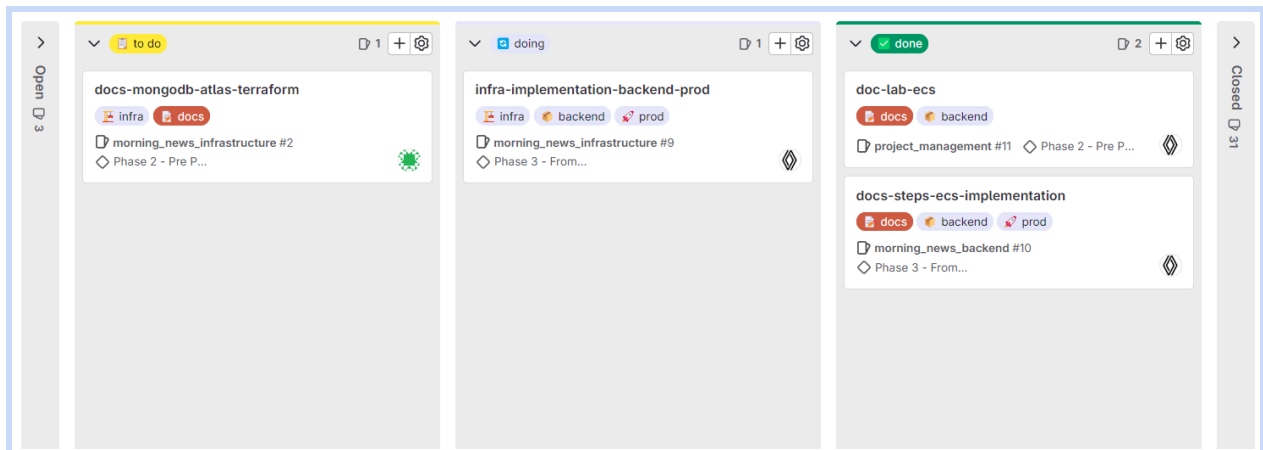


Fig. 6. Issue Board du Projet I Capture d'Écran via GitLab.

- **Git Flow** : Adoption et documentation d'un git flow basé sur les meilleures pratiques, avec un tutoriel d'onboarding pour les nouveaux membres. Cela assure une contribution cohérente et structurée, réduisant les risques de conflits et d'erreurs.

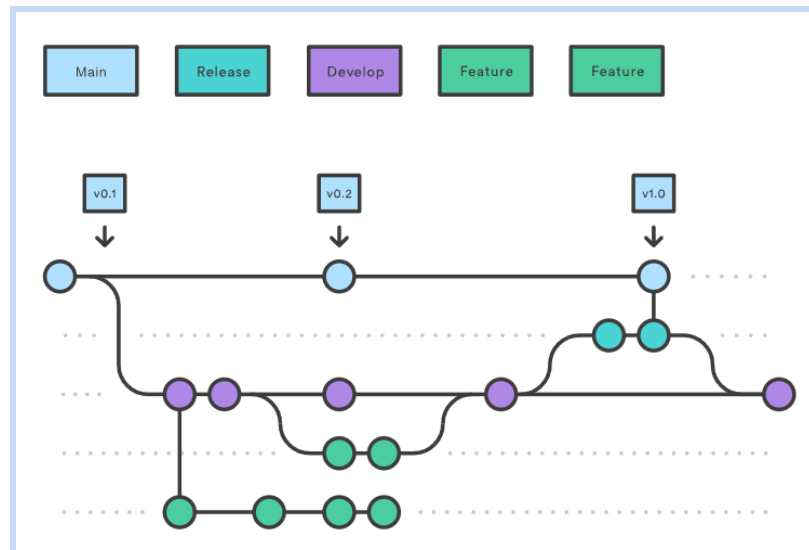


Fig. 7. Illustration du Git Flow I Capture d'Écran via Atlassian.

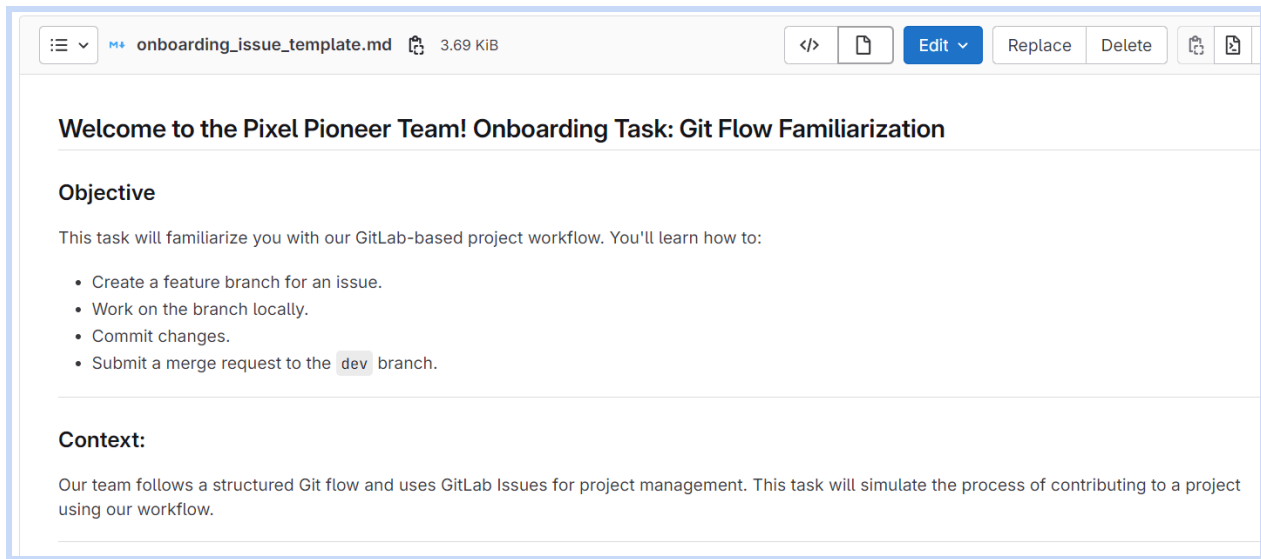


Fig. 8. Template de Tâche d'Onboarding | Capture d'Écran via GitLab.

- **Protection des Branches** : Configuration des protections de branches suivant le git flow pour garantir la qualité du code. En exigeant des merge requests et des approbations pour les branches principales, nous assurons une revue rigoureuse du code avant l'intégration.

- **Onboarding des Membres** : Invitation des membres avec les privilèges appropriés et création des premières milestones et issues. Cela permet à chaque membre de démarrer rapidement et de comprendre ses responsabilités et ses tâches.

Phase 1 - Set up and initial Lab phase 🔗 MorningNewsProject	Dec 6, 2024–Dec 11, 2024
Phase 2 - Pre Prod Env Prep 🔗 MorningNewsProject	Dec 11, 2024–Dec 13, 2024
Phase 3 - From Pre Prod to Prod 🔗 MorningNewsProject	Dec 16, 2024–Dec 18, 2024
Phase 4 - Prod deployment and restitution 🔗 MorningNewsProject	Dec 19, 2024–Dec 20, 2024
Project Checklist 🔗 MorningNewsProject	Dec 9, 2024–Dec 20, 2024

Fig. 9. “Milestones” (Grandes Étapes) du Projet | Capture d’Écran via GitLab.

iii. Intégration Slack GitLab

Pour utiliser Slack comme outil de notification et d'alerte, j'ai mis en place un webhook pour recevoir les notifications GitLab dans Slack. Cela nous permet de rester informés en temps réel des contributions et des changements dans le projet. Cette intégration initiale de Slack nous a également préparés à des utilisations plus complexes, telles que les alertes de Prometheus Alertmanager et Grafana, ainsi que les notifications de MongoDB Atlas. En introduisant Slack avec une utilisation simple, j'ai permis à l'équipe de se familiariser avec l'outil avant de passer à des intégrations plus techniques et critiques.

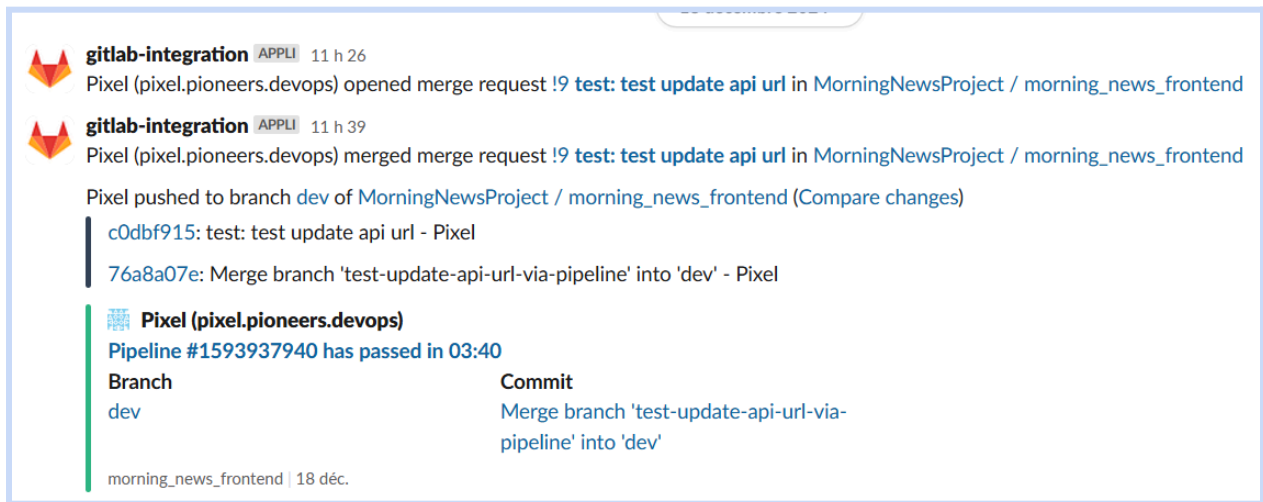


Fig. 10. Canal Slack dédié aux notifications GitLab | Capture d'Écran via Slack.

iv. Prise en Main et Intégration d'Infisical

J'ai pris en main et intégré Infisical pour la gestion et le partage des secrets entre environnements (dev, staging, prod). En introduisant cet outil dès le début, j'ai permis aux membres de se familiariser rapidement avec son utilisation, intégrant ainsi les pratiques de sécurité dès le départ. Cela a été crucial pour s'assurer que les secrets étaient gérés de manière sécurisée et standardisée, évitant les fuites potentielles et les erreurs de configuration.

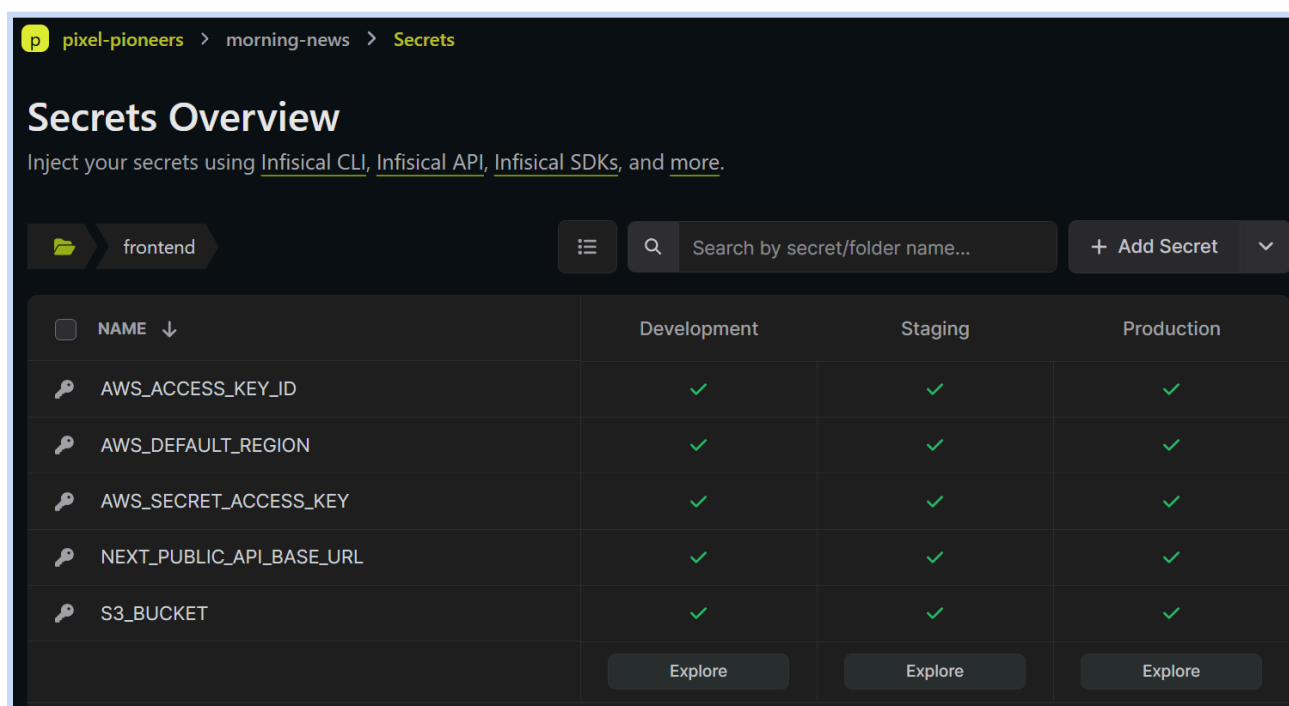


Fig. 11. Exemple de l'Interface d'Infisical Cloud | Capture d'Écran via Infisical Cloud.

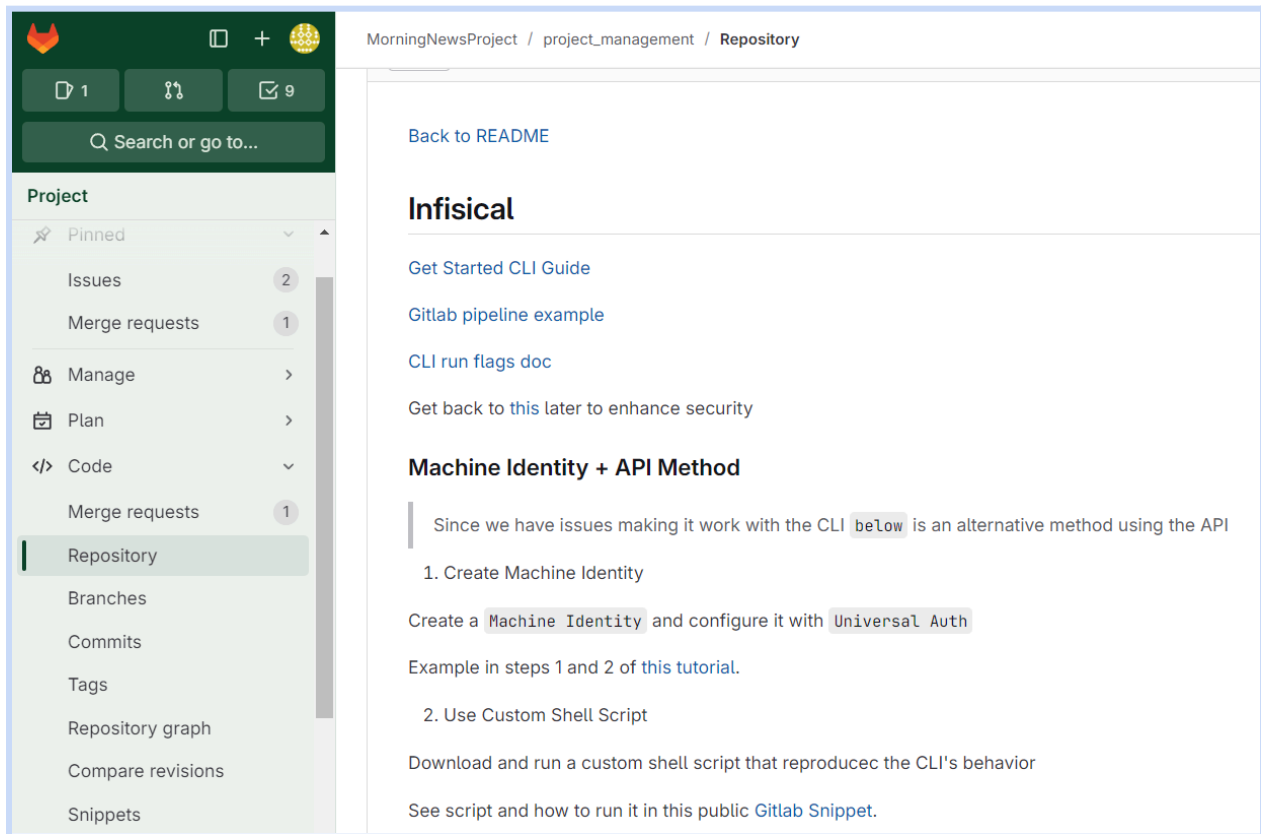


Fig. 12. Documentation dédiée à l'intégration d'Infisical dans le cadre du projet | Capture d'Écran via GitLab.

v. Définition de Requirements Techniques et SLAs

Pour guider nos choix de solutions et servir de référence tout au long du projet, j'ai tenté de définir des requirements techniques et des SLAs basés sur les standards du marché pour une application du même type que la nôtre. Ces guidelines devront nous permettre d'évaluer les performances et d'orienter nos stratégies de test, de monitoring et d>alerting.

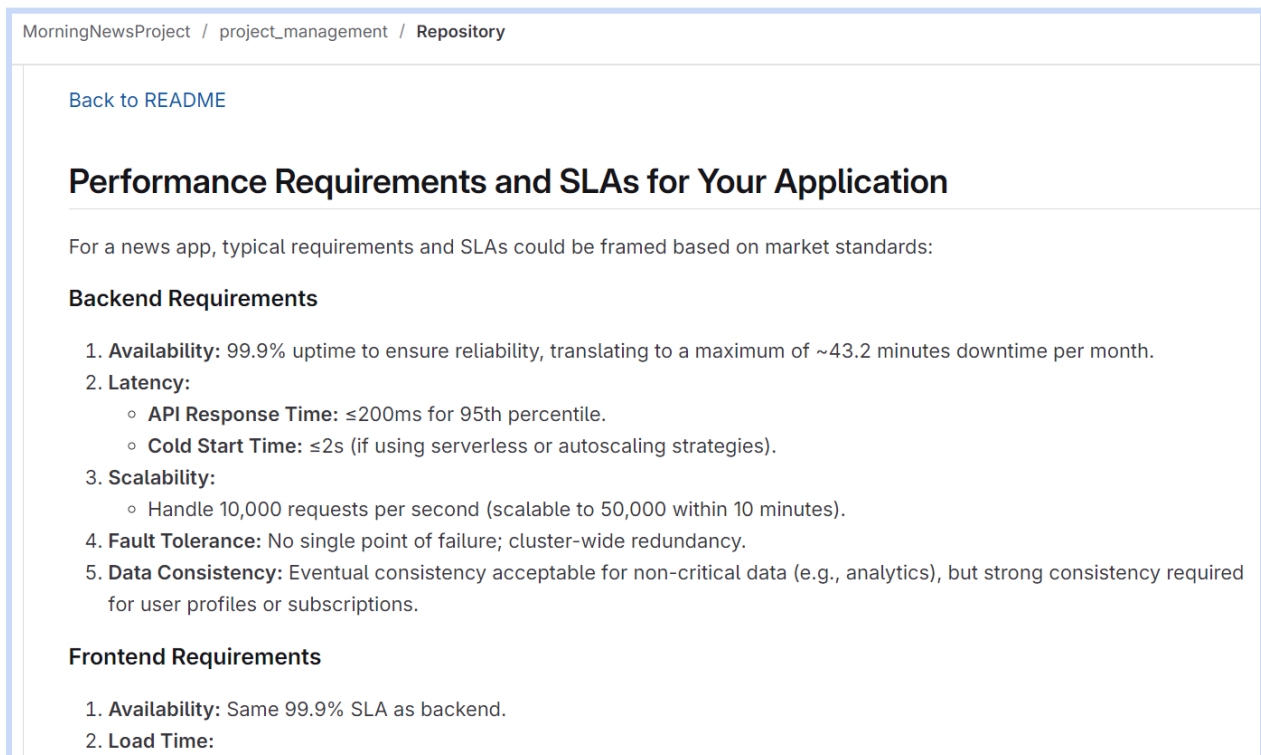


Fig. 13. Documentation dédiée aux Requirements et SLAs de Référence | Capture d'Écran via GitLab.


Ces initiatives ont non seulement permis une gestion efficace de la collaboration, mais ont également établi des bases solides pour le développement et le déploiement, tout en assurant une sécurité et une qualité constante du code. En prenant en charge cette partie critique du projet, j'ai contribué à créer un environnement où chaque membre pouvait travailler de manière productive et coordonnée, malgré les différences de niveaux et d'expertise.

b. Infrastructure & Monitoring

Dans la continuité de l'organisation de l'environnement de collaboration et de versioning, j'ai mis en place une gestion de l'infrastructure et du monitoring qui a permis de mener à bien notre approche itérative. Cela a nécessité l'intégration d'outils permettant de créer et de détruire rapidement des stacks et des environnements, dans la logique de notre méthode basée sur des labs, afin de tester et d'évaluer rapidement des solutions avant d'adopter les plus performantes selon nos guidelines.

i. Gestion de l'Infrastructure

Pour la gestion de l'infrastructure, j'ai utilisé Terraform et Ansible via le repository infrastructure. L'état de Terraform est géré via des *workspaces* dédiés avec Terraform Cloud et les variables sensibles via Infisical.



```
1 terraform {
2   cloud {
3
4     organization = "morning_news"
5
6     workspaces {
7       name = "terraform_preprod_backend"
8     }
9   }
10 }
11
```

Fig. 14. Intégration de Terraform Cloud via un fichier *backend.tf* et le choix d'un *workspace* dédié | Capture d'Écran via GitLab.

ii. Structure du Repository

Répertoire Terraform : Organisé de manière modulaire pour faciliter la gestion et la scalabilité de l'infrastructure.

- **Dossiers par Composant d'Infrastructure** :
 - **frontend** : Contient les configurations Terraform pour déployer l'infrastructure frontend.
 - **backend_pre_prod** : Contient les configurations pour le backend en environnement de pré-production.

```
34 # Cloudflare DNS Config
35
36 variable "subdomains" {
37   description = "List of subdomains to create Cloudflare DNS records for"
38   type        = list(string)
39   default     = ["mn-api-preprod", "cadvisor-backend", "node-exporter-backend"]
40 }
41
42 module "cloudflare_dns" {
43   for_each     = toset(var.subdomains) # Iterate over the list of subdomains
44   source       = "../modules/cloudflare_module"
45   cloudflare_zone = var.cloudflare_zone
46   subdomain    = each.value           # This refers to the current subdomain in the iteration
47   ip_address   = module.linode_instance.linode_instance_ip # Use the instance IP for each subdomain
48
49   depends_on = [module.linode_instance]
50 }
51
```

Fig. 15. Utilisation du module Cloudflare dans la Configuration de l'infrastructure de pré production | Capture d'Écran via GitLab.

- **backend_prod** : Contient les configurations pour le backend en environnement de production.
- **monitoring** : Contient les configurations pour les services de monitoring.

```

3  module "ssh_key" {
4      source      = "../modules/ssh_key_module"
5      private_key_path = "../../${var.private_key_path}"
6      public_key_path  = "../../${var.public_key_path}"
7  }
8
9  data "local_file" "public_key" {
10     filename      = module.ssh_key.public_key_path
11     depends_on    = [module.ssh_key]
12 }
13
14 # Linode Instance
15
16 module "linode_instance" {
17     source = "../modules/linode_module"
18
19     label                = "terraform_monitoring_instance"
20     type                 = "g6-dedicated-2"
21     public_key_content   = replace(module.ssh_key.public_key_content, "\n", "")
22     linode_root_password = var.linode_root_password
23     ansible_hosts_file_path = "../..../ansible/inventories/monitoring/hosts"
24     ansible_hosts_group_name = "monitoring_server"
25     ansible_host_name      = "monitoring_server"
26     ansible_ssh_private_key_path = "../${var.private_key_path}"
27
28     depends_on = [data.local_file.public_key]
29 }
30
31 output "linode_instance_ip" {
32     value = module.linode_instance.linode_instance_ip
33 }

```

Fig. 16. Utilisation des modules SSH et Linode dans la Configuration de l'infrastructure de monitoring, création et utilisation des clés pour créer l'instance Linode puis output de l'IP de l'instance | Capture d'Écran via GitLab.

- **database** : Contient les configurations pour la base de données.
- **Dossier Modules** : Comprend des modules réutilisables pour diverses configurations :

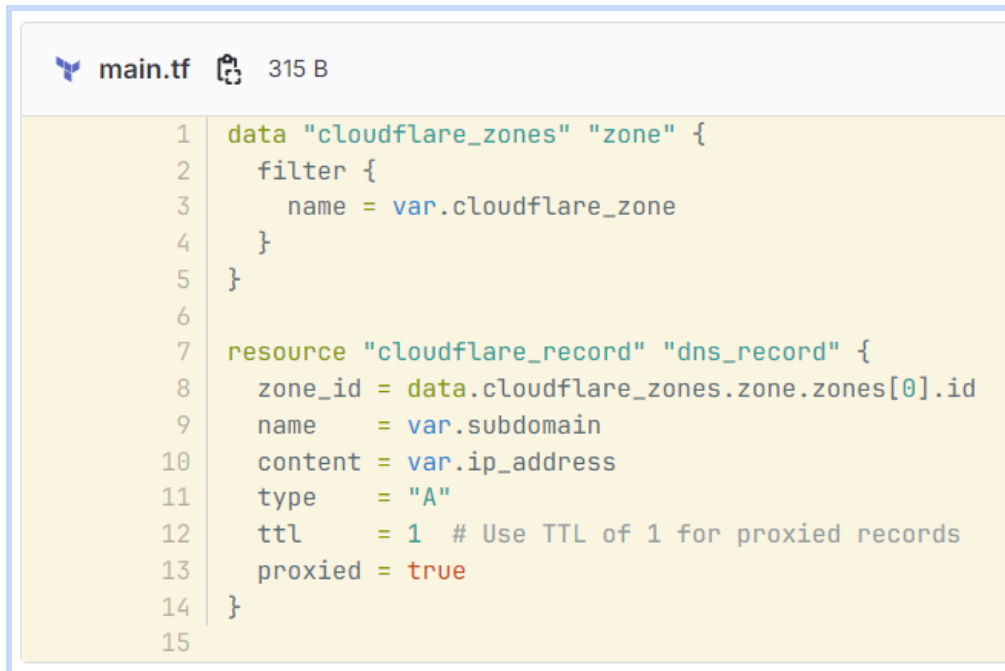
- **linode_module** : Module pour la gestion des instances Linode, crée dynamiquement un fichier “hosts” dédié aux instances créées et utilisable par Ansible.



```
main.tf 591 B
1 resource "linode_instance" "terraform_linode_instance" {
2   image           = var.image
3   label           = var.label
4   region          = var.region
5   type            = var.type
6   authorized_keys = [var.public_key_content]
7   root_pass       = var.linode_root_password
8 }
9
10 resource "local_file" "ansible_inventory" {
11   content = <<EOF
12   [${var.ansible_hosts_group_name}]
13   ${var.ansible_host_name} ansible_host=${linode_instance.ter
14 EOF
15   filename = var.ansible_hosts_file_path
16 }
17
```

Fig. 17. Fichier principal de du module Linode | Capture d’Écran via GitLab.

- **cloudflare_module** : Module pour la gestion des enregistrements DNS et des certificats SSL via Cloudflare.



```
1 data "cloudflare_zones" "zone" {
2   filter {
3     name = var.cloudflare_zone
4   }
5 }
6
7 resource "cloudflare_record" "dns_record" {
8   zone_id = data.cloudflare_zones.zone.zones[0].id
9   name     = var.subdomain
10  content  = var.ip_address
11  type     = "A"
12  ttl      = 1 # Use TTL of 1 for proxied records
13  proxied  = true
14 }
15
```

Fig. 18. Fichier principal de du module Cloudflare | Capture d'Écran via GitLab.

- **ssh_key_module** : Module pour la création automatique de clés SSH utilisables par Ansible.



```
1 resource "tls_private_key" "example" {
2   algorithm = "RSA"
3   rsa_bits  = 4096
4 }
5
6 resource "local_file" "private_key" {
7   content  = tls_private_key.example.private_key_pem
8   filename = var.private_key_path
9   file_permission = "0600"
10 }
11
12 resource "local_file" "public_key" {
13   content  = tls_private_key.example.public_key_openssh
14   filename = var.public_key_path
15   file_permission = "0644"
16 }
17
18 output "private_key_path" {
19   value = var.private_key_path
20 }
```

Fig. 19. Fichier principal de du module SSH Keys | Capture d'Écran via GitLab.

iii. Création Automatisée de Ressources

Terraform : Automatisation de la création de clés SSH, des enregistrements DNS via Cloudflare et autres ressources nécessaires pour les différents environnements. Cela assure une intégration transparente entre les configurations Terraform et Ansible. Par exemple, les clés SSH sont créées automatiquement et mises à disposition d'Ansible, et les informations d'hosts sont écrites automatiquement dans les configurations Ansible.

Ansible : Utilisation des informations générées par Terraform pour configurer les hôtes et appliquer les rôles de manière dynamique.

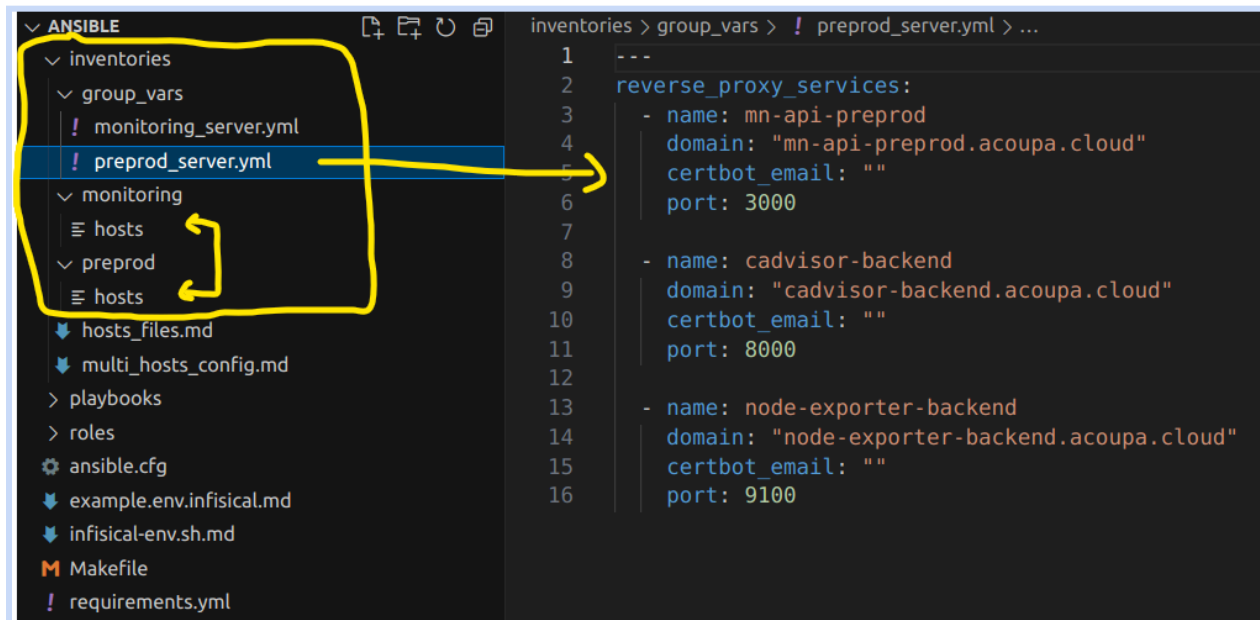


Fig. 20. Création dynamique par Terraform de fichiers *hosts* liés aux différentes configuration Ansible | Capture d'Écran via VS Code.

iv. Intégration des Secrets

Infisical : Gestion des secrets et des variables d'environnement de manière sécurisée, accessibles à la fois par Terraform et Ansible pour assurer la cohérence et la sécurité des configurations. Cette intégration permet de partager facilement les secrets entre les différents environnements (dev, staging, prod).

v. Déploiement et Configuration

Les configurations et déploiements sont actuellement effectués en local (apply Terraform et exécution des playbooks Ansible) en attendant la mise en place d'un bastion sécurisé dédié. Voici les principaux composants de cette infrastructure :

- Rôles Ansible :

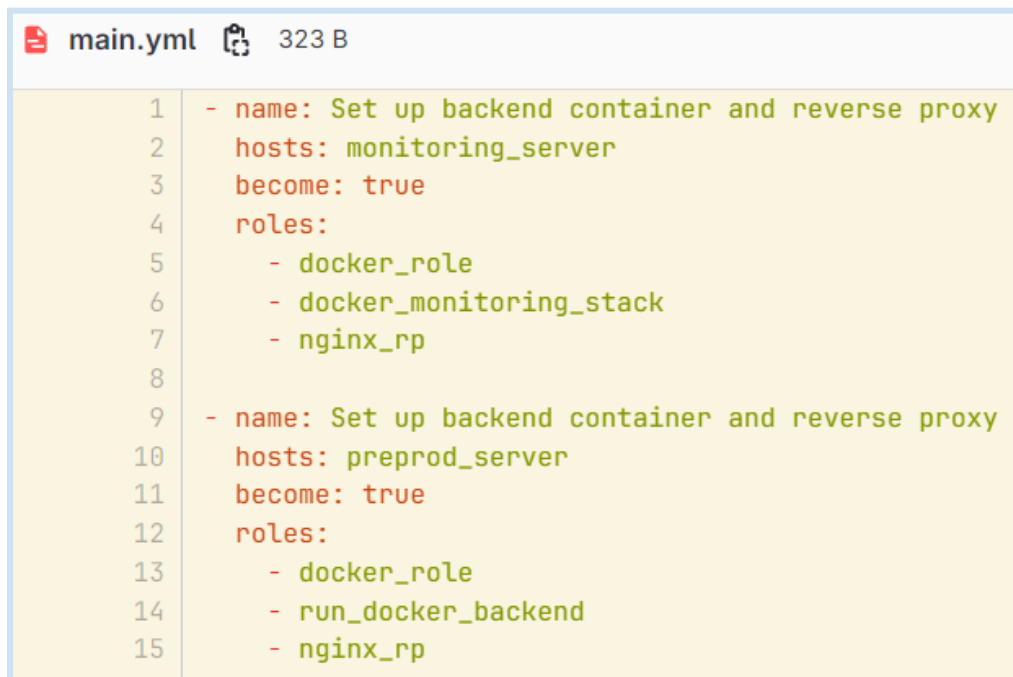
docker_monitoring_stack : Installe et configure la stack de monitoring via Docker Compose.

docker_role : Installe Docker sur les instances, assurant une base pour les déploiements containerisés.

nginx_rp : Configure des reverse proxies pour les différents services, les rendant accessibles via SSH et DNS (sous-domaines gérés par Cloudflare).

run_docker_backend : Configure et lance les Docker Compose pour l'application backend en pré-production.

security_role : Applique les bonnes pratiques de sécurité (configuration SSH, firewall, Fail2ban, etc.) pour assurer la sécurité des instances.



```
1 - name: Set up backend container and reverse proxy
2   hosts: monitoring_server
3   become: true
4   roles:
5     - docker_role
6     - docker_monitoring_stack
7     - nginx_rp
8
9 - name: Set up backend container and reverse proxy
10  hosts: preprod_server
11  become: true
12  roles:
13    - docker_role
14    - run_docker_backend
15    - nginx_rp
```

Fig. 21. Exemple d'utilisation des Roles dans un Playbook | Capture d'Écran via GitLab.

- **Stack de Monitoring :**

Services de Monitoring : Une fois l'infrastructure déployée, les services de monitoring tels que Prometheus, Grafana, et Alertmanager sont configurés et accessibles via les

sous-domaines définis. Les dashboards Grafana sont pré-configurables et rapidement mis en place via des templates.

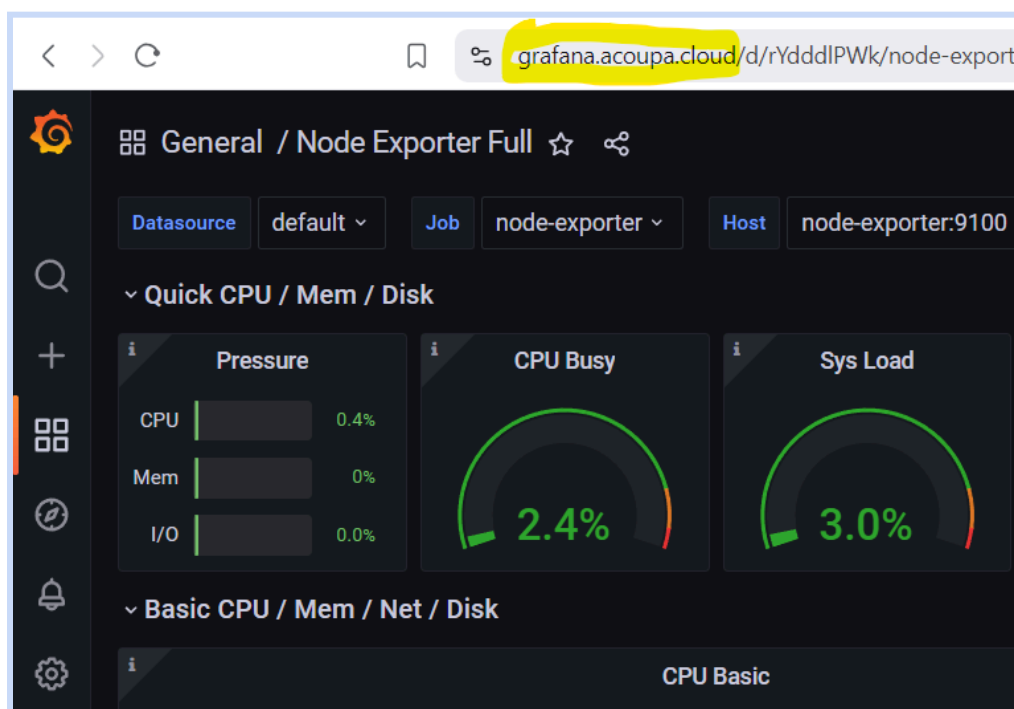


Fig. 22. Le service et dashboard Grafana sont disponibles via le sous-domaine configuré avec Terraform | Capture d'Écran via l'UI de Grafana.

Alertes Pré-configurées : Les alertes sont configurées selon les guidelines et SLAs définis, assurant une surveillance proactive des performances et de la disponibilité des services. Les alertes sont liées à Slack via un webhook, dirigeant les notifications vers un canal dédié, ce qui permet à l'équipe de rester informée en temps réel des incidents et anomalies.

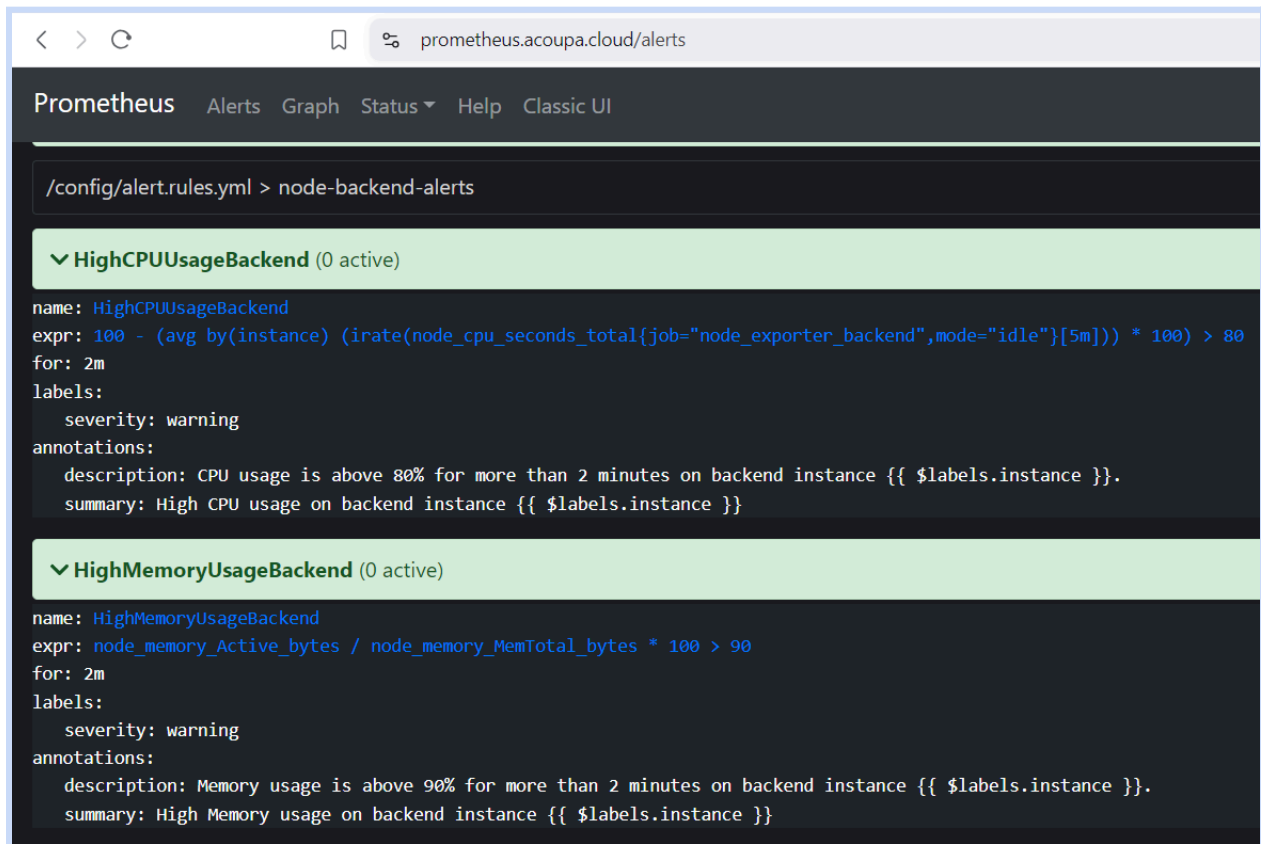


Fig. 23. Exemple d'alertes via l'interface du service Prometheus pré-configurée avec Terraform et Ansible | Capture d'Écran via l'UI de Grafana.

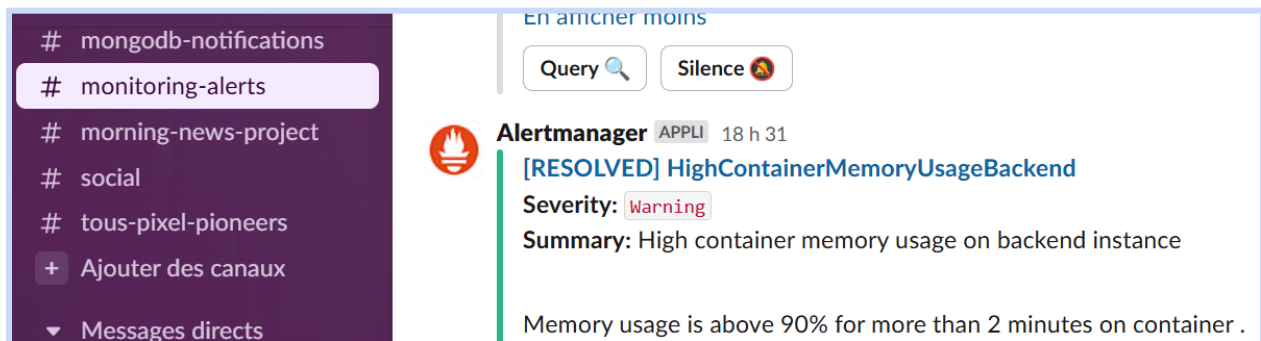


Fig. 24. Exemple de notifications d'alerte via Slack | Capture d'Écran via Slack.

- Gestion Dynamique des Environnements :

Création et Destruction Rapide : La capacité à créer et détruire rapidement des stacks et des environnements permet de tester différentes configurations et de choisir les solutions les plus adaptées. Cette flexibilité est cruciale pour notre approche itérative basée sur des labs, permettant de valider les choix techniques avant de les adopter définitivement.

Intégration Complète : Toute la stack s'intègre avec nos choix techniques et environnements, assurant un monitoring des instances et containers via Node Exporter et cAdvisor. L'intégration de Grafana avec AWS CloudWatch permet de surveiller la partie ECS.

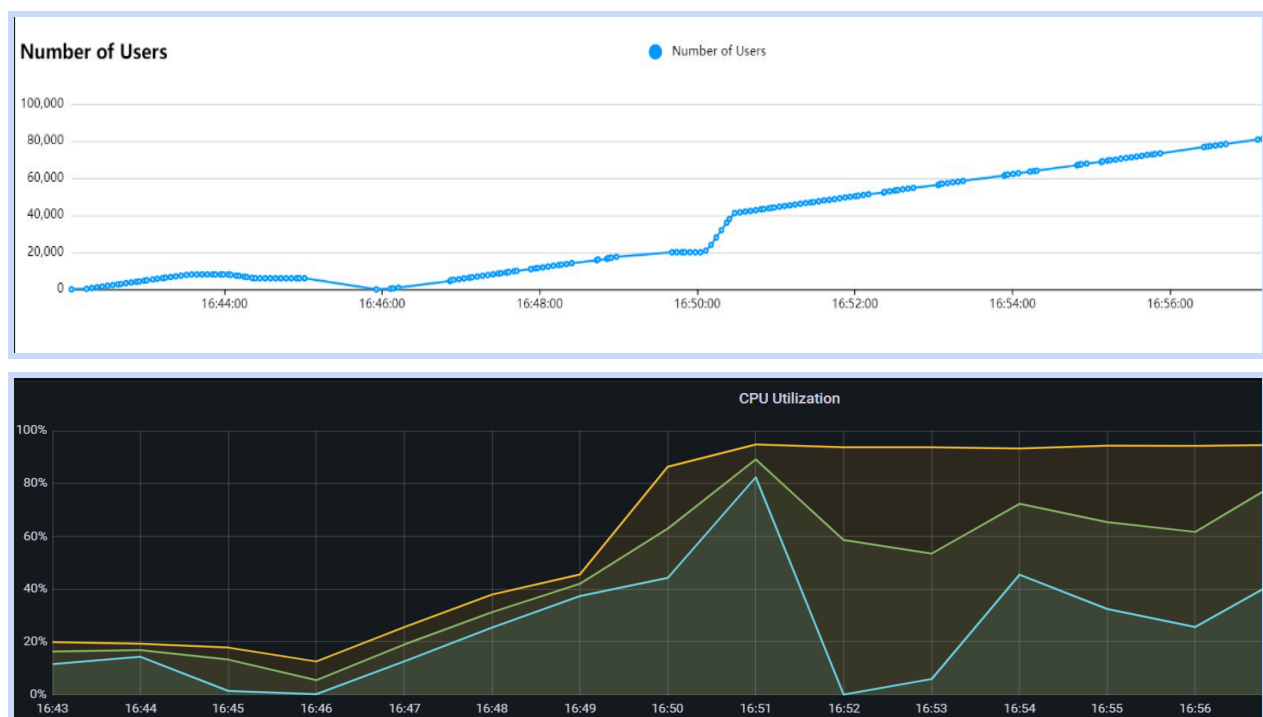



Fig. 25. Mise en parallèle d'un test de montée en charge via Locust et du monitoring du comportement du cluster ECS via Grafana et AWS Cloudwatch | Captures d'Écran via les interfaces Locust et Grafana.



Cette configuration et cette capacité à itérer et tester rapidement nous ont permis d'expérimenter nos différents choix techniques, notamment en comparant les performances dans un contexte de load testing avec Locust. En combinant Terraform et Ansible avec une gestion centralisée des secrets via Infisical, nous avons pu automatiser et sécuriser la configuration et le déploiement de l'infrastructure. Cette approche nous a permis de rester agiles et réactifs, tout en garantissant la qualité et la sécurité de nos environnements.



c. Solution Frontend

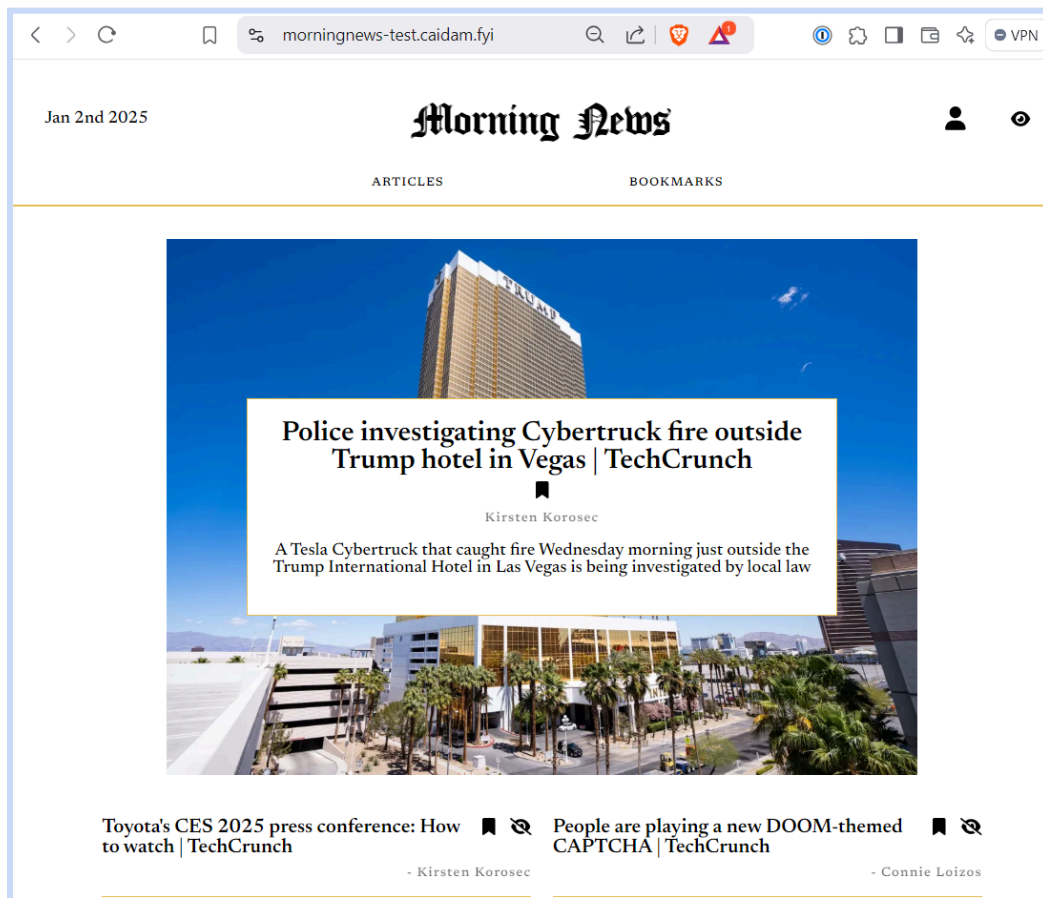


Fig. 26. Page d'Accueil du Frontend déployé en environnement de pré-production | Capture d'Écran dans le navigateur.

Pour la mise en œuvre de la solution frontend, j'ai choisi d'utiliser l'hébergement statique sur S3 combiné à Cloudflare pour héberger l'application Next.js. Cette approche offre une performance optimale, une grande scalabilité et une gestion simplifiée des coûts. Voici un aperçu des étapes et des choix techniques effectués.

i. Préparations Initiales

- **Variabilisation de l'URL de l'API :**

Création de la variable NEXT_PUBLIC_API_BASE_URL conformément aux directives de Next.js pour les variables d'environnement.

```
18 fetch(`${process.env.NEXT_PUBLIC_API_BASE_URL}/users/canBookmark/${user.token}`)  
19 .then(response => response.json())  
20 .then(data => {  
21   if (data.result && data.canBookmark) {  
22     if (props.isBookmarked) {  
23       dispatch(removeBookmark(props));  
24     } else {  
25       dispatch(addBookmark(props));  
26     }  
27   }  
28 });  
29 }
```

Fig. 27. Exemple de la variabilisation de l'URL d'API dans un composant du frontend | Capture d'Écran via GitLab.

Refactorisation du code pour intégrer cette variable, simplifiant ainsi l'intégration d'Infisical et le changement d'URL en fonction des environnements (dev, staging, prod).

NEXT_PUBLIC_API_BASE_URL		✓	✓
Key	NEXT_PUBLIC_API_BASE_URL		
Environment	Value	Hide Values	
Development	http://localhost:3000		
Staging	https://mn-api-preprod.acoupa.cloud		
Production	https://alb.ashlay-moudivendar.cloud		

Fig. 28. Définition des variables par Environnement dans Infisical | Capture d'Écran via Infisical Cloud.

- Résolution du Problème de Loader d'Images :

Problème : En raison de la manière dont Next.js gère les images, il a été nécessaire de trouver une solution pour le loader afin de permettre un chargement correct des images lorsqu'elles sont hébergées sur un serveur statique comme S3.

Solution : Utilisation d'un loader custom. Sur les conseils de camarades ayant rencontré le même problème, nous avons choisi le loader Akamai pour résoudre ce problème.

Voici la configuration dans next.config.js :

A screenshot of a code editor showing the configuration for next.config.js. The file is named 'next.config.js' and is 278 B in size. The code is as follows:

```
1  /** @type {import('next').NextConfig} */
2  const nextConfig = {
3    reactStrictMode: true,
4    images: {
5      domains: ['cdn.vox-cdn.com', 'techcrunch.com', 's.yimg.com'],
6      loader: 'akamai', // custom loader for static hosting
7      path: '', //
8    },
9  };
10
11 module.exports = nextConfig;
```

Fig. 29. Ajout d'un Loader Custom dans la configuration du frontend | Capture d'Écran via GitLab.

ii. Process de Build & Export

Pour permettre l'hébergement statique avec un projet Next.js, il est obligatoire de d'abord construire et exporter (*build* et *export*) le projet pour générer les fichiers statiques dans un dossier distinct. Une fois cette étape réalisée, nous avons tout en place pour implémenter la solution S3 + Cloudflare.

iii. Choix S3 + Cloudflare

Hébergement Statique sur S3 : Amazon S3 offre un stockage d'objets simple, peu coûteux (gratuit si éligible au Free Tier) et permet des uploads faciles (par exemple

depuis un CI). Il s'intègre parfaitement dans l'écosystème AWS et fournit un serveur web simple.

Utilisation de Cloudflare : Cloudflare est utilisé comme fournisseur DNS et CDN, offrant un plan gratuit, une configuration facile, un certificat TLS, etc. Comparé à CloudFront, Cloudflare est souvent plus économique et offre des fonctionnalités similaires voire supérieures pour ceux qui sont prêts à explorer des options en dehors d'AWS.

iv. Mise en Oeuvre

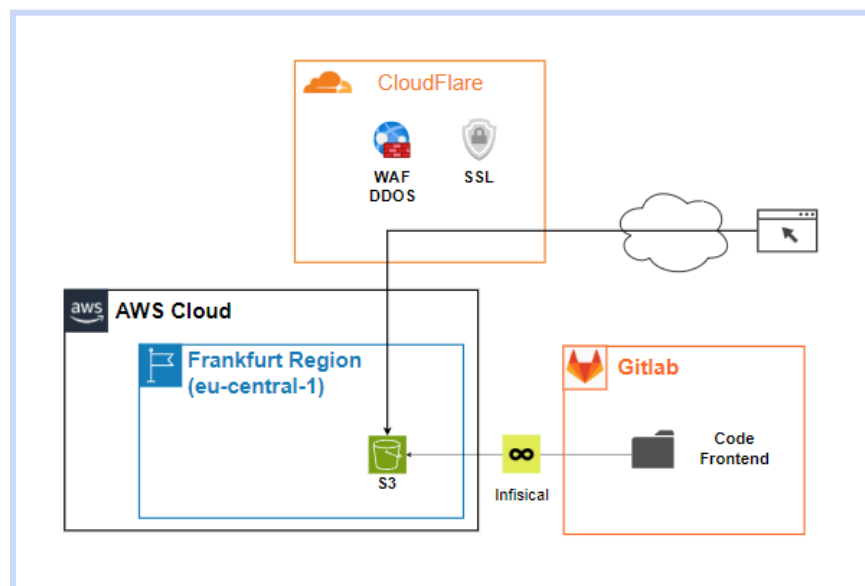


Fig. 30. Schéma d'architecture de la solution Frontend | Créé via draw.io.

Création du Bucket S3 :

- Créer un bucket S3 avec le même nom que le domaine (incluant tous les niveaux de domaine).
- Configurer le bucket pour l'hébergement de site web statique et uploader les fichiers construits.

Configuration du Bucket pour l'Accès Public :

- Configurer les permissions du bucket pour permettre l'accès public aux fichiers, en modifiant la *policy* du bucket.

Configuration DNS avec Cloudflare :

- Créer un enregistrement CNAME dans Cloudflare pointant vers l'URL d'hébergement de site web statique de S3.
- S'assurer que le trafic passe par Cloudflare pour bénéficier du CDN et des fonctionnalités de sécurité.

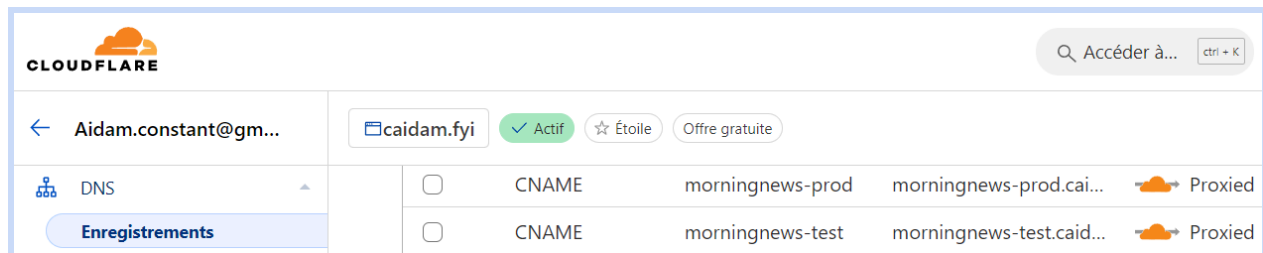


Fig. 31. Enregistrements DNS via l'interface Cloudflare | Capture d'écran via Cloudflare.


V. Avantages de Sécurité avec Cloudflare

Cloudflare offre de nombreux avantages en matière de sécurité qui renforcent notre solution d'hébergement :

Protection DDoS : Cloudflare fournit une protection avancée contre les attaques DDoS, capable de mitiger des attaques de grande ampleur et de maintenir la disponibilité du site.

Certificats TLS Automatiques : Cloudflare gère automatiquement les certificats TLS, garantissant une communication sécurisée via HTTPS sans intervention manuelle.

Web Application Firewall (WAF) : Le WAF de Cloudflare protège contre les menaces courantes en filtrant les requêtes malveillantes avant qu'elles n'atteignent le serveur.



IP Whitelisting : En configurant des règles d'accès, nous pouvons restreindre l'accès aux ressources S3 uniquement aux IPs de Cloudflare, augmentant ainsi la sécurité et réduisant les risques d'accès non autorisé.

HSTS et Options de Sécurité HTTP : Cloudflare permet de configurer HTTP Strict Transport Security (HSTS) et d'autres options de sécurité HTTP pour renforcer la sécurité des communications web.

Gestion des Bots : Cloudflare inclut des outils pour identifier et gérer le trafic des bots, réduisant les risques d'abus et de surcharge du serveur.

vi. Vers une automatisation du Déploiement

J'ai également documenté le processus d'intégration et d'automatisation du build et du déploiement afin qu'il puisse être intégré dans un pipeline CI/CD par un autre membre de l'équipe. Cela inclut :

- **Scripts de Build** : Scripts pour construire et exporter l'application Next.js.
- **Scripts de Déploiement** : Scripts pour uploader les fichiers statiques dans le bucket S3 et invalider les caches Cloudflare si nécessaire.

En résumé, cette approche avec S3 et Cloudflare offre une solution d'hébergement frontend efficace et évolutive. Elle permet une gestion simplifiée des coûts et une haute disponibilité, tout en assurant des performances optimales grâce au CDN de Cloudflare. En automatisant le processus de build et de déploiement, nous avons également facilité la gestion continue et les mises à jour de l'application.

6. Situation De Travail Ayant Nécessité Une Recherche

a. Contexte et Problème Initial

Nous avons unanimement choisi d'utiliser Infisical pour la gestion des secrets et des environnements en raison de sa praticité et de la façon optimale et simple dont il permet de gérer ces aspects. Cependant, bien que tout ait fonctionné en utilisant la CLI pour une utilisation locale, nous avons rencontré des difficultés pour utiliser la CLI dans le contexte d'une Machine Identity (système d'authentification pour que Infisical puisse être utilisé par un service, dans un pipeline CI/CD par exemple).

b. Recherche et Tentatives de Résolution

Face à ce problème, j'ai entrepris une recherche approfondie pour trouver une solution :

- **Consultation de la Documentation Officielle :**

Je me suis d'abord tourné vers la documentation d'Infisical directement. Malheureusement, cela n'a pas permis de résoudre le problème. La documentation n'était pas forcément à jour en raison des évolutions continues de la gestion des identités machines et leur mode d'authentification.

- **Exploration des Issues GitHub :**

Je me suis ensuite référé aux discussions dans les diverses issues en cours ou fermées sur le repo GitHub d'Infisical. J'ai cherché à voir si d'autres utilisateurs avaient rencontré le même problème et quelles avaient été leurs solutions. Bien que cela m'ait

aidé à mieux comprendre le contexte du problème, je n'ai pas trouvé de solution définitive à ce stade.

- **Recherches sur les Blogs et Forums :**

Conscient des défis potentiels liés aux outils plus récents et moins matures comme Infisical mais jugeant qu'il vaut le coût de s'y confronter, j'ai décidé d'investir davantage de temps en explorant les blogs et les forums à la recherche d'une alternative viable. C'est ainsi que j'ai découvert un forum détaillant une alternative potentielle : utiliser l'API directement pour contourner le problème spécifique à la CLI.



Fig. 32. Article présentant une alternative viable à notre problème | Capture d'écran via un navigateur.



c. Mise en Oeuvre de la Solution

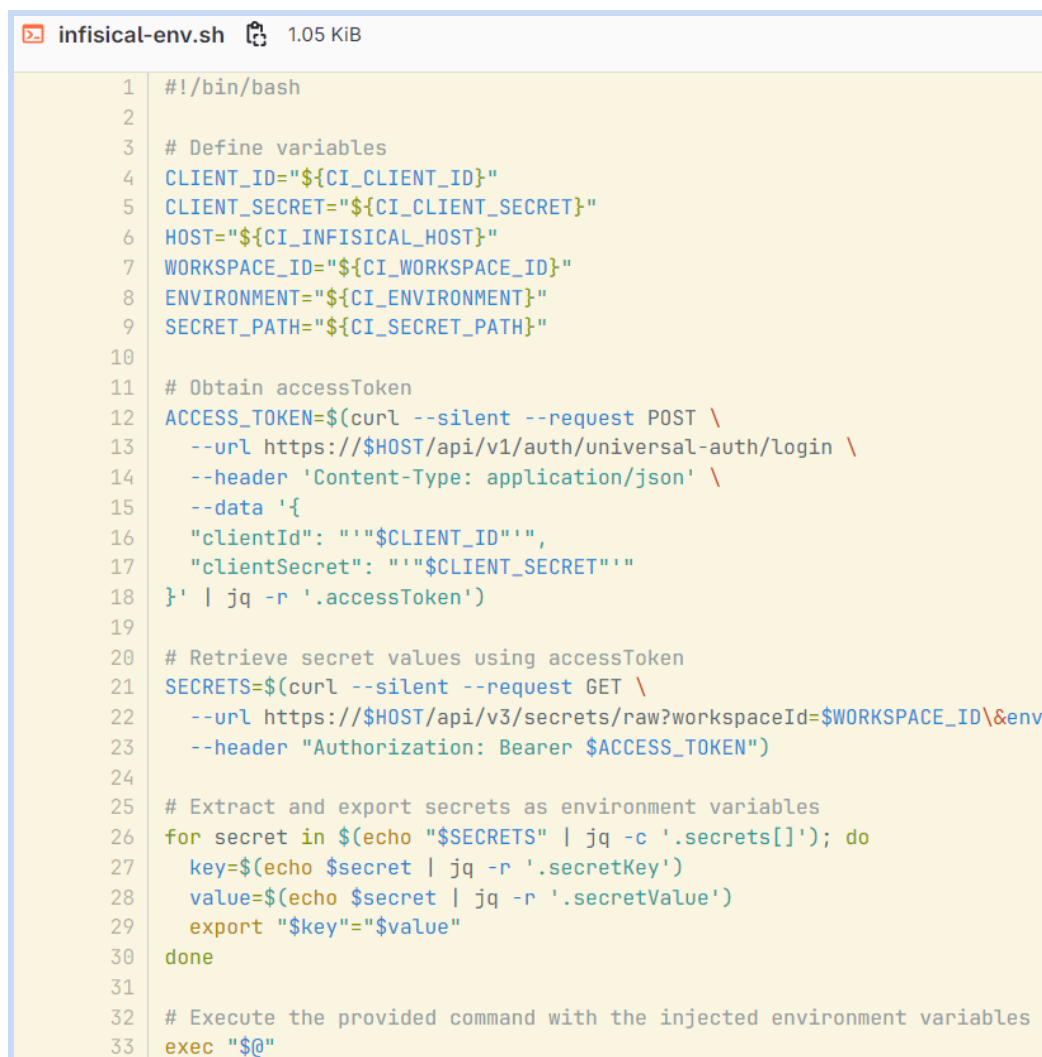
Après avoir identifié une solution potentiellement applicable, j'ai procédé comme suit :

- **Test de l'Implémentation :**

J'ai rapidement testé l'implémentation de l'API pour vérifier si elle pouvait résoudre notre problème. Les résultats ont confirmé que cette approche était effectivement viable dans notre contexte.

- **Création d'un Script Shell :**

J'ai ensuite créé un script shell automatisant le processus de récupération des secrets via une Machine Identity. Ce script permet d'obtenir un comportement similaire à celui de la CLI, mais en utilisant l'API pour l'authentification et la récupération des données.

The image is a screenshot of a code editor window from GitLab, displaying a shell script named 'infisical-env.sh'. The window title bar shows the file name and its size, '1.05 KiB'. The script content is as follows:

```
1  #!/bin/bash
2
3  # Define variables
4  CLIENT_ID="${CI_CLIENT_ID}"
5  CLIENT_SECRET="${CI_CLIENT_SECRET}"
6  HOST="${CI_INFISICAL_HOST}"
7  WORKSPACE_ID="${CI_WORKSPACE_ID}"
8  ENVIRONMENT="${CI_ENVIRONMENT}"
9  SECRET_PATH="${CI_SECRET_PATH}"
10
11 # Obtain accessToken
12 ACCESS_TOKEN=$(curl --silent --request POST \
13   --url https://$HOST/api/v1/auth/universal-auth/login \
14   --header 'Content-Type: application/json' \
15   --data '{
16     "clientId": "'"$CLIENT_ID"'",
17     "clientSecret": "'"$CLIENT_SECRET"'
18   }' | jq -r '.accessToken')
19
20 # Retrieve secret values using accessToken
21 SECRETS=$(curl --silent --request GET \
22   --url https://$HOST/api/v3/secrets/raw?workspaceId=$WORKSPACE_ID&env.
23   --header "Authorization: Bearer $ACCESS_TOKEN")
24
25 # Extract and export secrets as environment variables
26 for secret in $(echo "$SECRETS" | jq -c '.secrets[]'); do
27   key=$(echo $secret | jq -r '.secretKey')
28   value=$(echo $secret | jq -r '.secretValue')
29   export "$key"="$value"
30 done
31
32 # Execute the provided command with the injected environment variables
33 exec "$@"
```

Fig. 33. Script Shell qui reproduit le comportement de la CLI Infisical | Capture d'écran via GitLab.

- Documentation et Partage :

Pour faciliter l'accès et l'utilisation de cette solution par l'équipe, j'ai publié le script sous forme de Snippet GitLab. Cela permet une documentation claire et une intégration facile dans nos pipelines CI/CD et autres usages éventuels.

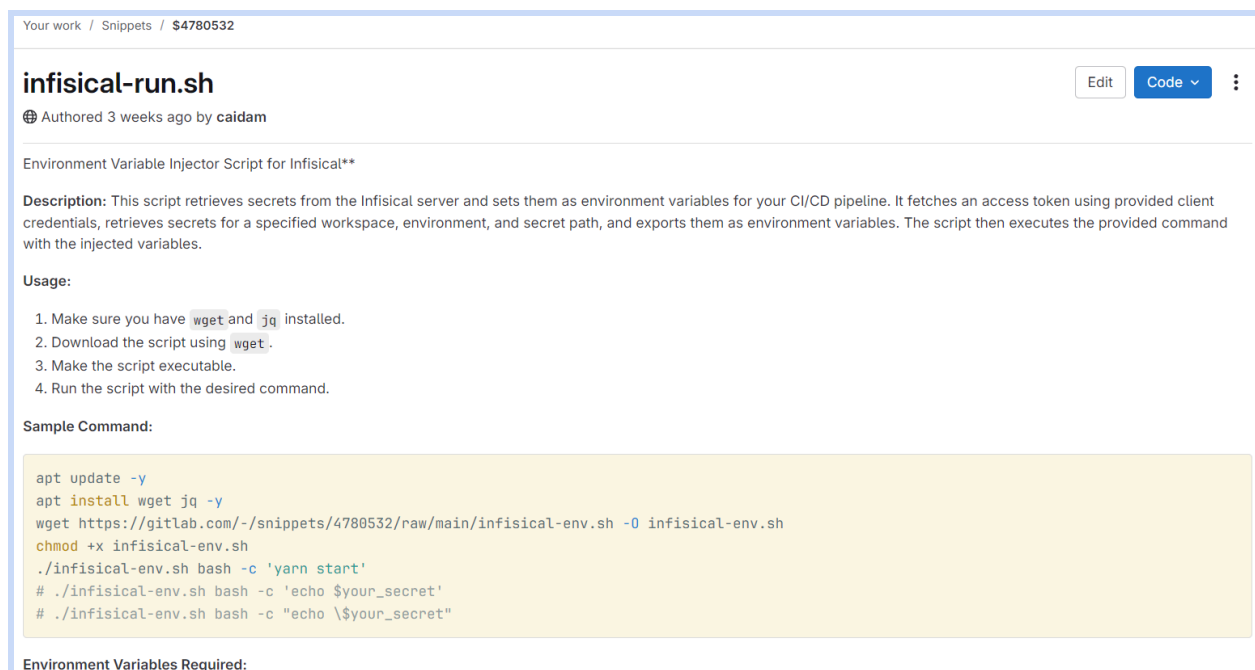


Fig. 34. Extrait du Snippet GitLab qui documente le script et facilite son intégration dans nos solutions | Capture d'écran via GitLab.

```
89  deploy-s3:
90    stage: deploy
91    dependencies:
92      - setup-infisical
93      - build-app
94    variables:
95      CI_ENVIRONMENT: "staging" # TODO - Dynamic
96    image:
97      name: amazon/aws-cli
98      entrypoint: [""] # By default, with this image, aws is defined as the
99    before_script:
100      - yum install -y jq
101    script:
102      - ./env-inject.sh bash -c "aws s3 rm s3://\${S3_BUCKET} --recursive"
103      - ./env-inject.sh bash -c "aws s3 cp out s3://\${S3_BUCKET} --recursive"
104    rules:
105      - if: $CI_COMMIT_BRANCH == "dev"
```

Fig. 35. Exemple d'utilisation du script par un membre du groupe dans un pipeline GitLab CI | Capture d'écran via GitLab.

d. Conclusion

Cette situation a démontré l'importance de la recherche approfondie et de la persévérance face à des outils en constante évolution. En explorant diverses ressources et en testant des solutions alternatives, nous avons pu surmonter les limitations initiales et tirer parti des avantages d'Infisical pour la gestion des secrets et des environnements, assurant ainsi une intégration fluide et sécurisée dans nos processus de développement et de déploiement.