

PLAN DE DÉPLOIEMENT

PLAN DE MAINTENANCE

PLAN DE SÉCURISATION



L'APPLICATION DE VOTRE
SANTÉ MENTALE

David LACLEF

Table des matières

Note explicative	4
Contexte :	4
Objectif du document :	4
Destinataires :	4
Périmètre :	4
Plan de déploiement.....	5
Description de l'architecture	5
Architecture technique	5
Environnements de déploiement.....	6
Configuration infrastructure :	6
Services déployés :	6
Outils de versioning configurés	6
Outils d'automatisation configurés	7
Plan de maintenance	9
Outil de gestion des évolutions - GitHub Issues.....	9
Processus de gestion des évolutions	10
Méthodologie de gestion des incidents	10
Plan de sécurisation.....	11
Analyse des risques de sécurité identifiés	11
Actions préventives mises en œuvre	12
Structuration des développements et bonnes pratiques	14
Traitement des incidents de sécurité.....	15
Plan de communication et escalade	16
Templates de communication.....	20
Outils de communication.....	21

Métriques de communication.....	21
Conclusion	22

Note explicative

Ce document est le rapport écrit de l'infrastructure CI/CD mise en place pour l'application CESIZen dans le cadre du déploiement et de la sécurisation d'une solution informatique. Il présente les choix techniques, les processus opérationnels et les procédures de sécurité adoptées.

Contexte :

Application web de bien-être développée en .NET 8 avec Blazor Server, déployée sur infrastructure Azure.

Objectif du document :

Formaliser l'architecture de déploiement, définir les processus de maintenance et établir les procédures de sécurisation.

Destinataires :

Équipe(s) DevOps.

Périmètre :

Infrastructure de production, processus DevOps, gestion des incidents et évolutions, sécurité applicative et infrastructure.

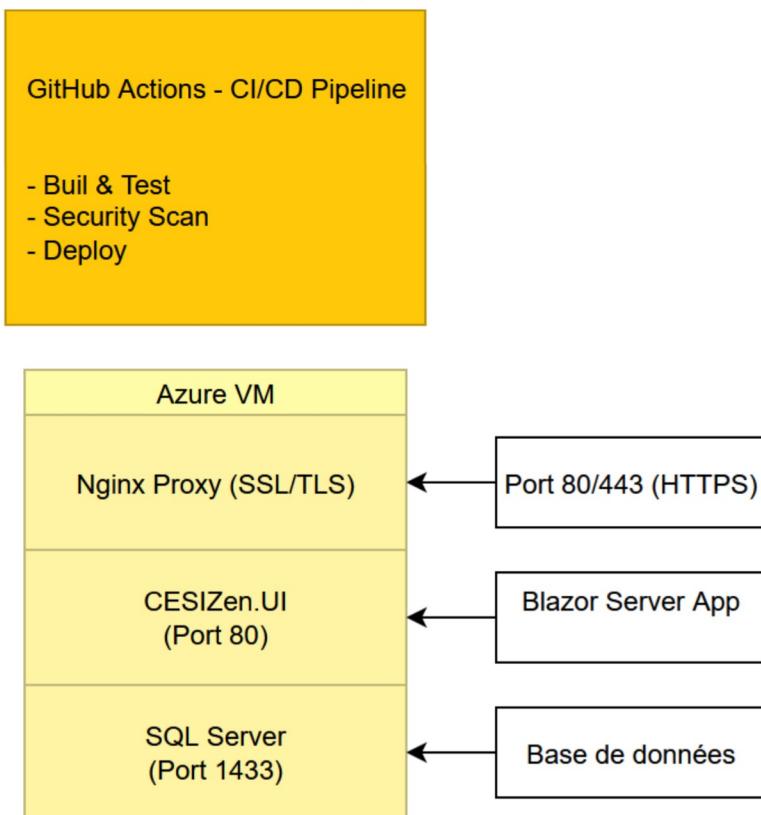
Plan de déploiement

Le plan de déploiement définit l'architecture technique et les processus automatisés mis en place pour assurer le déploiement de l'application CESIZen en environnement de production. Il décrit l'infrastructure Azure containerisée, les outils de versioning Git avec système de tags, ainsi que les pipelines CI/CD GitHub Actions qui orchestrent les phases de build, test et déploiement automatique.

Description de l'architecture

L'architecture de déploiement CESIZen repose sur une approche conteneurisée avec Docker, orchestrée par Docker Compose, et déployée sur une infrastructure cloud Azure.

Architecture technique



Environnements de déploiement

L'application CESIZen est déployée uniquement en environnement de production sur Azure, les phases de développement et de test s'effectuant en local sur les postes des développeurs.

Configuration infrastructure :

Hébergement : Azure Virtual Machine (Ubuntu 20.04 LTS)

Ressources : 4 GB RAM, 2 vCPU, 50 GB SSD

Réseau : IP publique avec domaine personnalisé (optionnel)

Sécurité : Firewall UFW, Fail2ban, certificats SSL

Services déployés :

Base de données : SQL Server 2022 Express (conteneur Docker)

Interface utilisateur : CESIZen.UI (Blazor Server) via conteneur Docker

Reverse proxy : Nginx avec SSL/TLS

Outils de versioning configurés

Repository principal : GitHub avec branches main et develop

Workflow de branches : GitFlow

main : branche de production, déploiement automatique

develop : branche de développement, tests d'intégration

feature/* : branches de fonctionnalités

fix/* : corrections

Tags Git pour le versioning :

Format des tags : v1.0.0 (Semantic Versioning)

Création automatique : Tags créés automatiquement lors des déploiements

Utilisation :

v1.0.0 -> Version majeure (changements incompatibles)

v1.1.0 -> Version mineure (nouvelles fonctionnalités)

v1.1.1 -> Version patch (corrections de bugs)

Exemple de création de tag :

```
> git tag -a v1.0.0 -m "Version 1.0.0 - Release initiale"
```

```
> git push origin v1.0.0
```

Outils d'automatisation configurés

GitHub Actions - Pipeline CI/CD

Workflow d'intégration (01_Integration.yaml) :

- **Déclenchement :** Push sur develop/main, Pull Request vers main
- **Étapes :**
 1. Tests unitaires et d'intégration
 2. Build des applications (.NET 8)
 3. Analyse de qualité SonarCloud
 4. Scan de sécurité Trivy
 5. Build des images Docker

Workflow de déploiement (02_Deployment.yaml) :

- **Déclenchement** : Push sur main
- **Étapes** :
 1. Build et push des images Docker vers GHCR
 2. Déploiement sur Azure VM
 3. Création automatique de tag Git

Plan de maintenance

Le plan de maintenance établit les processus et outils nécessaires pour assurer la continuité de service et l'évolution de l'application CESIZen. Il définit l'utilisation de GitHub Issues comme système de ticketing centralisé pour la gestion des incidents et des demandes d'évolution, ainsi que les procédures de maintenance préventive.

Outil de gestion des évolutions - GitHub Issues

GitHub Issues est configuré comme système de ticketing principal pour gérer les évolutions de l'application et le traitement des incidents.

Organisation des tickets :

Labels de classification :

- bug : Dysfonctionnement à corriger
- enhancement : Amélioration de fonctionnalité existante
- feature : Nouvelle fonctionnalité
- security : Problème de sécurité
- urgent : Priorité élevée

Milestones :

- Version releases (v1.0.0, v1.1.0, etc.)
- Sprints de développement
- Corrections de sécurité

Processus de gestion des évolutions

Workflow de traitement des tickets :

1. **Création** : Ticket créé avec template approprié
2. **Triage** : Assignation des labels et priorités
3. **Planification** : Ajout au milestone approprié
4. **Développement** : Création de branche feature/ ou fix/
5. **Tests** : Validation sur branche develop
6. **Déploiement** : Merge vers main et déploiement automatique
7. **Vérification** : Tests post-déploiement
8. **Clôture** : Fermeture du ticket avec résolution

Méthodologie de gestion des incidents

Niveaux de criticité :

- **Critique** : Application inaccessible, perte de données
- **Élevé** : Fonctionnalité majeure non disponible
- **Moyen** : Fonctionnalité mineure impactée
- **Faible** : Problème cosmétique ou d'ergonomie

Temps de réponse :

- **Critique** : 1 heure
- **Élevé** : 4 heures
- **Moyen** : 1 jour ouvré
- **Faible** : 3 jours ouvrés

Plan de sécurisation

Le plan de sécurisation présente une approche multicouche visant à protéger l'application CESIZen contre les menaces internes et externes. Il détaille l'analyse des risques identifiés, les mesures préventives implémentées au niveau applicatif et infrastructure, ainsi que les procédures de réponse aux incidents de sécurité. Cette stratégie s'appuie sur des outils automatisés de détection des vulnérabilités, des bonnes pratiques de développement et un processus structuré de gestion de crise pour minimiser l'impact des incidents sur les utilisateurs et les données.

Analyse des risques de sécurité identifiés

Niveau de probabilité :

Élevée	
Moyenne	
Faible	

Type	Risque	Impact	Probabilité
Vulnérabilités des dépendances	Exploitation de failles dans les packages NuGet	Compromission de l'application, injection de code	
Injection SQL	Injection de code SQL malveillant	Accès non autorisé aux données, corruption de la base	
Authentification et autorisation	Contournement des mécanismes de sécurité	Accès non autorisé aux fonctionnalités	
Accès SSH non sécurisé	Attaque par force brute sur SSH	Compromission du serveur	
Certificats SSL expirés	Communication non chiffrée	Interception des données	
Déni de service (DDoS)	Surcharge du serveur	Indisponibilité de l'application	

Actions préventives mises en œuvre

Sécurité du code

<i>Dependabot activé</i>	<ul style="list-style-type: none"> - Surveillance automatique des dépendances - Création de Pull Requests pour les mises à jour de sécurité - Notifications en cas de vulnérabilité critique
<i>Code Scanning (CodeQL)</i>	<ul style="list-style-type: none"> - Analyse statique du code source - Détection automatique des vulnérabilités - Intégration dans le pipeline CI/CD
<i>Secret Scanning</i>	<ul style="list-style-type: none"> - Détection des secrets écrits dans le code - Prévention des commits contenant des clés - Rotation automatique des secrets compromis

Sécurité infrastructure

Firewall UFW configuré

Ports ouverts uniquement nécessaires

22/tcp # SSH (accès admin)

80/tcp # HTTP (redirection HTTPS)

443/tcp # HTTPS (application)

Fail2ban actif

- Protection contre les attaques par force brute
- Bannissement automatique des IP malveillantes
- Surveillance des tentatives de connexion SSH

Certificats SSL/TLS

- Chiffrement des communications
- Renouvellement automatique via Let's Encrypt
- Configuration sécurisée (TLS 1.2+)

Structuration des développements et bonnes pratiques

Bonnes pratiques de développement

<i>Principe de moindre privilège</i>	<ul style="list-style-type: none"> -Rôles utilisateur distincts (Utilisateur, -Administrateur) -Autorisation granulaire par fonctionnalité -Validation des permissions côté serveur
<i>Validation des entrées</i>	<ul style="list-style-type: none"> -Sanitisation des données utilisateur -Utilisation de paramètres SQL (Entity Framework) -Validation côté client ET serveur
<i>Gestion des erreurs</i>	<ul style="list-style-type: none"> -Logs détaillés sans exposition d'informations sensibles -Messages d'erreur génériques pour l'utilisateur -Monitoring des exceptions

Sécurité des données

<i>Chiffrement</i>	<ul style="list-style-type: none"> -Mots de passe hashés (ASP.NET Core Identity) -Données sensibles chiffrées en base -Communication HTTPS obligatoire
<i>Sauvegarde sécurisée</i>	<ul style="list-style-type: none"> -Sauvegardes chiffrées -Stockage sur support séparé -Tests de restauration réguliers

Traitement des incidents de sécurité

Méthodologie de réponse aux incidents

<i>Phase 1 : Détection et analyse (0-1h)</i>	<ol style="list-style-type: none"> 1. Identification de l'incident via monitoring 2. Évaluation de la criticité 3. Isolation du système affecté 4. Notification de l'équipe de sécurité
<i>Phase 2 : Confinement (1-2h)</i>	<ol style="list-style-type: none"> 1. Limitation de la propagation 2. Préservation des preuves 3. Mise en place de mesures temporaires 4. Communication aux parties prenante
<i>Phase 3 : Éradication (2-4h)</i>	<ol style="list-style-type: none"> 1. Identification de la cause racine 2. Suppression de la menace 3. Correction des vulnérabilités 4. Mise à jour des systèmes
<i>Phase 4 : Récupération (4-8h)</i>	<ol style="list-style-type: none"> 1. Restauration des services 2. Surveillance renforcée 3. Tests de validation 4. Retour à la normale
<i>Phase 5 : Retour d'expérience (J+1)</i>	<ol style="list-style-type: none"> 1. Analyse post-incident 2. Mise à jour des procédures 3. Formation de l'équipe 4. Amélioration continue

Plan de communication et escalade

Le plan de communication et escalade définit les procédures pour assurer une réponse coordonnée lors d'incidents affectant l'application CESIZen. Il établit une matrice de sévérité des incidents avec des délais de réponse adaptés, des circuits d'escalade hiérarchique selon la criticité, et des canaux de communication différenciés pour l'équipe technique et les utilisateurs finaux. Cette organisation garantit une mobilisation rapide des ressources appropriées et une information transparente de toutes les parties prenantes durant la résolution des incidents.

Niveaux de严重性 des incidents

Niveau	Critères	Exemples
CRITIQUE	Application inaccessible, perte de données, faille de sécurité majeure	Serveur down, corruption BDD, injection SQL réussie
ÉLEVÉ	Fonctionnalité majeure indisponible, dégradation performance importante	Connexion impossible, erreur 500 récurrente
MOYEN	Fonctionnalité mineure impactée, problème d'ergonomie	Formulaire défaillant, affichage incorrect
FAIBLE	Problème cosmétique, amélioration mineure	Alignement CSS, texte mal formaté

Matrice d'escalade par niveau

Niveau	Délai de réponse	Première ligne	Escalade Niveau 1	Escalade Niveau 2	Escalade Niveau 3
CRITIQUE	15 minutes	Développeur de garde	Lead Developer (30 min)	DevOps Manager (1h)	CTO (2h)
ÉLEVÉ	1 heure	Développeur assigné	Lead Developer (4h)	DevOps Manager (1h)	CTO (2 jours)
MOYEN	4 heures	Développeur assigné	Lead Developer (1 jour)	DevOps Manager (3 jours)	-
FAIBLE	24 heures	Développeur assigné	Lead Developer (3 jours)	-	-

Canaux de communication

Type d'incident	Canal principal	Canal secondaire	Notification externe
CRITIQUE	Teams #alerts + Appel	Email + SMS	Utilisateurs (status page)
ÉLEVÉ	Teams #alerts	Email	Utilisateurs (si impact visible)
MOYEN	Teams #dev-team	Email	-
FAIBLE	GitHub Issues	Teams #dev-team	-

Responsabilités par rôle

Rôle	Responsabilités	Niveau d'intervention
Développeur de garde	Première intervention, diagnostic initial	Critique, Élevé
Lead Developer	Coordination technique, décisions d'architecture	Critique, Élevé, Moyen
DevOps Manager	Supervision infrastructure, décisions de déploiement	Critique, Élevé
Product Owner	Priorisation fonctionnelle, communication business	Critique, Élevé
CTO	Décisions stratégiques, communication direction	Critique

Templates de communication

Template notification CRITIQUE

 INCIDENT CRITIQUE - CESIZen

Heure : [HH:MM]
Statut : [EN COURS/RÉSOLU]
Impact : [Description]
Actions en cours : [Description]
Responsable : [Nom]
Prochaine mise à jour : [HH:MM]

Template notification ÉLEVÉ

 INCIDENT ÉLEVÉ – CESIZen

Heure : [HH:MM]
Fonctionnalité : [Nom]
Impact : [Description]
ETA résolution : [Description]
Responsable : [Nom]

Outils de communication

Outil	Usage	Niveau d'accès
Teams	Communication temps réel	Équipe technique
Email	Communication formelle	Toute l'équipe
GitHub Issues	Suivi incidents	Équipe développement
Status Page	Communication externe	Public
Téléphone	Escalade critique	Management

Métriques de communication

Métrique	Cible	Mesure
Temps de première réponse	< 15 min (critique)	Temps entre détection et première intervention
Temps de résolution	< 2h (critique)	Temps entre détection et résolution
Satisfaction communication	> 90%	Enquête post-incident
Temps d'escalade	Selon matrice	Respect des délais d'escalade

Conclusion

Ce rapport présente une infrastructure CI/CD complète pour l'application CESIZen, intégrant un déploiement automatisé avec GitHub Actions et Docker sur Azure, une architecture sécurisée avec monitoring continu et sauvegardes automatiques, un processus de maintenance structuré avec gestion des tickets via GitHub Issues, un plan de sécurisation complet avec mesures préventives et réactives, ainsi qu'une communication d'incident organisée avec escalade claire.

L'ensemble de cette infrastructure garantit la sécurité grâce à de multiples couches de protection et surveillance, la maintenabilité par des processus clairs et un outillage adapté, et l'évolutivité via une architecture modulaire et un pipeline flexible. Cette approche DevOps permet un développement agile tout en maintenant un haut niveau de qualité et de sécurité pour l'application CESIZen.