

NICOLAS WALTER

MAINTENIR L'APPLICATION EN CONDITION OPÉRATIONNELLE

Projet : Todo App Cloud

Réf. Évaluation : Bloc 4 (Certification RNCP39583)

Formation : Mastère 2 Expert en Développement Full Stack

École : Ynov Campus Strasbourg

22/09/2026

Résumé Exécutif

Ce dossier technique détaille la démarche de Maintien en Condition Opérationnelle (MCO) de l'application **Todo App Cloud**, refonte Cloud-Native et DevSecOps de l'application **Todo App** des tutoriels Docker. Réalisé dans le cadre scolaire durant les cours «Développer pour le cloud» et amélioré à part pour la certification RNCP Bloc 4, ce projet démontre la capacité à monitorer, maintenir et améliorer une architecture microservices en production.

Périmètre du MCO :

- **Supervision & Dépendances** : Stratégie de mise à jour des packages, sondes Kubernetes et supervision des performances et de la disponibilité.
- **Gestion des anomalies** : Processus de détection, consignation et résolution des bugs via CI/CD en environnement Cloud (Zero-Downtime).
- **Support & Amélioration** : Accompagnement des utilisateurs (résolution de tickets) et propositions chiffrées d'amélioration continue (Redis, Serverless).

Table des Matières

- 1. Supervision et Gestion des Dépendances 4
 - 1.1. Contexte : Le "Sprint MCO" et l'Observabilité 4
 - 1.2. Processus de Mise à Jour des Dépendances (C4.1.1) 4
 - 1.3. Système de Supervision et d'Alerte (C4.1.2) 6
- 2. Traitement des Anomalies en Production 10
 - 2.1. Processus de Collecte et Consignation (C4.2.1) 10
 - 2.2. Traitement et Déploiement du Correctif (C4.2.2) 11
- 3. Collaboration avec le Support Client 13
 - 3.1. Contexte et Collaboration avec le Support Client (C4.3.3) 13
- 4. Amélioration Continue et Historique 15
 - 4.1. Propositions d'Axes d'Amélioration (C4.3.1) 15
 - 4.2. Historique des Versions et Correctifs (C4.3.2) 16
- Conclusion 18

1. Supervision et Gestion des Dépendances

Ce chapitre détaille le processus de mise à jour sécurisé des dépendances ainsi que le système de supervision et d'alerte garantissant la haute disponibilité, répondant aux exigences des compétences C4.1.1 et C4.1.2.

1.1. Contexte : Le “Sprint MCO” et l’Observabilité

Note de contexte : L’infrastructure Azure (AKS, Application Insights, Azure Monitor) de ce projet n’est pas maintenue en production permanente pour des raisons budgétaires liées à la facturation à l’usage d’Azure. L’ensemble des configurations de supervision décrites dans ce chapitre — sondes Kubernetes, règles d’alerte Azure Monitor, intégration Application Insights et Sentry — a été **conçu, implémenté et validé fonctionnellement** sur un environnement de staging dédié lors du “Sprint MCO”. L’infrastructure a ensuite été déprovisionnée via `terraform destroy` pour maîtriser les coûts. Conformément aux modalités d’évaluation du Bloc 4 (*“mise en situation professionnelle réelle ou fictive”*), le présent dossier constitue une description technique complète du système de supervision conçu. Les configurations (YAML Kubernetes, fichiers Terraform, code d’intégration SDK) sont versionnées dans le dépôt Git du projet et peuvent être redéployées à tout moment.

Lors de la validation initiale de l’architecture (Bloc 2), l’application disposait d’une base Cloud-Native solide mais dépourvue d’outils complets de Maintien en Condition Opérationnelle (MCO). Afin de garantir la disponibilité en production (Bloc 4), l’équipe a dédié un “Sprint MCO” entier pour implémenter une stack d’Observabilité à trois piliers (Front, Back, Infra).

1.2. Processus de Mise à Jour des Dépendances (C4.1.1)

Le maintien à jour des composants logiciels est essentiel pour prévenir les failles de sécurité et bénéficier des dernières optimisations. Le processus s’appuie sur une démarche automatisée complétée par une validation humaine.

1.2.1. Périmètre Logiciel Concerné

- **Backend & Auth (Node.js/Express)** : Dépendances définies dans `package.json` (ex: Knex, Bcrypt, Express).
- **Frontend (React)** : Bibliothèques UI et utilitaires.
- **Infrastructure (Docker)** : Images de base (ex: `node:22-alpine`).

1.2.2. Fréquence et Automatisation (Dependabot)

Une analyse automatisée est effectuée chaque semaine (le lundi à 08h00) via **GitHub Dependabot** défini par le fichier de configuration `.github/dependabot.yml` dont voici un extrait :

```
version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/backend"
    schedule:
      interval: "weekly"
      day: "monday"
      time: "08:00"
    open-pull-requests-limit: 10
  - package-ecosystem: "docker"
```

```
directory: "/"
schedule:
  interval: "weekLy"
```

L'outil génère automatiquement des *Pull Requests (PR)* pour chaque dépendance périmée. La CI exécute ensuite la suite de tests (unitaires et E2E). Les mises à jour correctives et mineures sont automatiquement fusionnées si la CI passe au vert. En revanche, les mises à jour majeures (qui peuvent introduire des *breaking changes*) nécessitent une validation et des tests manuels par l'équipe de développement.

chore(deps): bump node from 22.23.1 to 26.0.0 in /client #120


[dependabot\[bot\]](#) wants to merge 1 commit into `main` from `dependabot/docker/client/node-26.0.0`

Conversation 0
 Commits 1
 Checks 7
 Files changed 1


dependabot Bot commented on behalf of [github](#) 17 hours ago

Bumps node from 22.23.1 to 26.0.0.

 compatibility  unknown

Dependabot will resolve any conflicts with this PR as long as you don't alter it yourself. You can also trigger a rebase manually by commenting `@dependabot rebase`.

► Dependabot commands and options



 Mention `@copilot` in a comment to make changes to this pull request.

  chore(deps): bump node from 22.23.1 to 26.0.0 in /client ... Verified ✓ 643b141

 **dependabot** Bot added `dependencies` `docker` labels 17 hours ago




All checks have passed
>

4 skipped, 3 successful checks


No conflicts with base branch

Merging can be performed automatically.

Merge pull request
 

You can also merge this with the command line. [View command line instructions.](#)

1.2.3. Gestion des Failles Critiques (CVE) & Évaluation d'Impact

En cas de détection d'une vulnérabilité de sévérité *HIGH* ou *CRITICAL*, l'action Trivy intégrée au pipeline GitHub Actions bloque immédiatement le déploiement :

```
- name: Run Trivy vulnerability scanner on Docker image
  uses: aquasecurity/trivy-action@master
  with:
    image-ref: "todo-app-backend:${{ github.sha }}"
```

```
format: "table"
exit-code: "1" # Échec du job si vulnérabilité trouvée
ignore-unfixed: true
vuln-type: "os,library"
severity: "HIGH,CRITICAL"
```

L'équipe procède alors à une évaluation d'impact (analyse des paquets impactés, tests d'isolation) avant d'appliquer une montée de version corrective (Hotfix) sous 24 heures.

trivy (client)

failed 2 weeks ago in 47s

▼

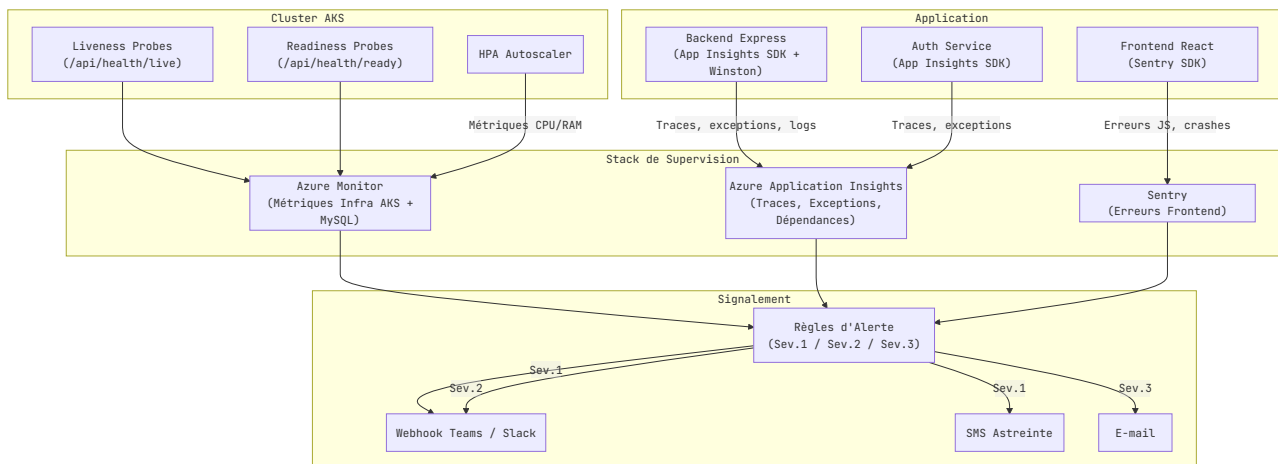
✖

Scan client image

340						
341	musl	CVE-2025-26519			1.2.5-r0	1.2.5-r1
342						
343						
344						
345		CVE-2026-40200				1.2.5-r3
346						
347						
348						
349	musl-utils	CVE-2025-26519				1.2.5-r1
350						
351						
352						
353		CVE-2026-40200				1.2.5-r3
354						
355						
356						
357	xz-libs	CVE-2025-31115			5.6.2-r0	5.6.2-r1
358						
359						
360	zlib	CVE-2026-22184			1.3.1-r1	1.3.2-r0
361						
362						
363						
364	Error: Process completed with exit code 1.					
365	▶ Remove Trivy Envs file					

1.3. Système de Supervision et d'Alerte (C4.1.2)

Pour garantir une disponibilité permanente (SLA 99.9%), l'application est supervisée à plusieurs niveaux de la pile technique. Le schéma ci-dessous présente l'architecture globale de la stack d'observabilité mise en place :



1.3.1. Sondes de Vie Kubernetes (Liveness & Readiness)

Le cluster AKS interroge en continu les microservices pour vérifier leur état via la configuration suivante dans le manifest Deployment (deployment.yaml) :

```
spec:
  containers:
    - name: todo-backend
      image: todo-app-backend:v2.1.2
      ports:
        - containerPort: 3000
      livenessProbe:
        httpGet:
          path: /api/health/live
          port: 3000
        initialDelaySeconds: 15
        periodSeconds: 10
        failureThreshold: 3
      readinessProbe:
        httpGet:
          path: /api/health/ready
          port: 3000
        initialDelaySeconds: 5
        periodSeconds: 5
```

- **Liveness Probe (/api/health/live)** : Vérifie l'état du processus applicatif Node.js (check mémoire/ event loop sans dépendance externe). En cas de blocage ou d'échec successif (3 tentatives), le Pod est redémarré par Kubernetes.
- **Readiness Probe (/api/health/ready)** : Vérifie que le service est prêt à traiter du trafic en testant les connexions actives aux dépendances (Base de données MySQL, Cache). La séparation des deux sondes évite les pannes en cascade (*CrashLoopBackOff*) : si la BDD rencontre une micro-coupure, le Pod est simplement retiré des routes Ingress/Nginx le temps du rétablissement, sans subir de redémarrage destructeur.

1.3.2. Périmètre de Supervision et Cartographie des Sondes

Afin de structurer la surveillance, le périmètre de supervision a été formalisé dans la matrice suivante, couvrant l'ensemble des composants critiques de l'architecture :

Composant	Sonde / Métrique	Seuil d'Alerte	Action Déclenchée
Backend API (Express)	Liveness Probe (/api/health/live)	3 échecs consécutifs (30s)	Redémarrage automatique du Pod

Composant	Sonde / Métrique	Seuil d'Alerte	Action Déclenchée
	Readiness Probe (/api/health/ready)	1 échec	Retrait du Pod du routage Ingress
	Temps de réponse HTTP (p95)	> 300ms sur 5 min	Alerte Sev.2 + Investigation
	Taux d'erreur HTTP 5xx	> 1% sur 5 min	Alerte Sev.1 + Astreinte
Frontend (React SPA)	Erreurs JavaScript (Sentry)	> 5 erreurs/min	Alerte Sev.2 via Slack
	Disponibilité (Health Check HTTP 200)	Échec pendant 2 min	Alerte Sev.1 + Astreinte
Base de données MySQL	Connexions actives	> 80% de max_connections	Alerte Sev.2 + Scaling pool
	Slow Queries (> 1s)	> 10 requêtes/min	Alerte Sev.3 + Revue requêtes
	Espace de stockage utilisé	> 85% du volume	Alerte Sev.1 + Extension disque
	Latence de réplication (si replica)	> 5s de retard	Alerte Sev.2
Cluster AKS (Nœuds)	Utilisation CPU par nœud	> 80% sur 5 min	Autoscaling HPA / Alerte Sev.2
	Utilisation RAM par nœud	> 85% sur 5 min	Alerte Sev.2 + Investigation OOM
	État des Pods (CrashLoopBackOff)	Détection immédiate	Alerte Sev.1 + Astreinte
Certificats TLS	Expiration du certificat Ingress	< 14 jours avant expiration	Alerte Sev.2 + Renouvellement

1.3.3. Critères de Qualité et Indicateurs de Performance (KPIs)

Les critères de qualité suivants ont été définis en adéquation avec la typologie de l'application (application web collaborative utilisée en journée, architecture microservices sur AKS) :

- **Disponibilité cible (SLA)** : 99.9% sur une fenêtre glissante de 30 jours, soit un budget d'indisponibilité maximal de ~43 minutes/mois.
- **Performance (Latence)** : Temps de réponse API p95 < 300ms et p99 < 500ms. Au-delà, dégradation perceptible de l'expérience utilisateur.
- **Fiabilité (Taux d'erreur)** : Taux d'erreurs HTTP 5xx < 0.1% en régime nominal. Le seuil d'alerte est fixé à 1% pour laisser une marge avant impact utilisateur significatif.
- **Saturation (Capacité)** : Utilisation CPU/RAM des nœuds maintenue sous 80% pour absorber les pics imprévus sans dégradation.

Ces indicateurs sont collectés par **Azure Monitor** (métriques d'infrastructure AKS et MySQL) et **Application Insights** (métriques applicatives : traces de requêtes, dépendances, exceptions). Les données sont consolidées dans un tableau de bord unifié permettant une vue temps réel de la santé du système.

1.3.4. Modalités de Signalement

Lorsqu'un seuil critique est franchi, une alerte est déclenchée et classifiée selon trois niveaux de sévérité :

Sévérité	Critère de déclenchement	Délai de réaction	Canal de notification
Sev.1 (Critique)	Indisponibilité totale, perte de données, CrashLoop	< 15 min	Webhook Teams/Slack + SMS astreinte
Sev.2 (Majeure)	Dégradation de performance, saturation partielle	< 1h	Webhook Teams/Slack
Sev.3 (Mineure)	Anomalie non bloquante, slow queries	Jour ouvré suivant	Notification e-mail

Chaque alerte est accompagnée des logs et traces pertinents (ID de corrélation, Stacktrace, métriques contextuelles) pour accélérer le diagnostic par l'équipe d'astreinte.

2. Traitement des Anomalies en Production

Ce chapitre illustre la méthode de consignation structurée des bugs et le cycle de déploiement d'un correctif automatisé via CI/CD, répondant aux exigences des compétences **C4.2.1** et **C4.2.2**.

2.1. Processus de Collecte et Consignation (C4.2.1)

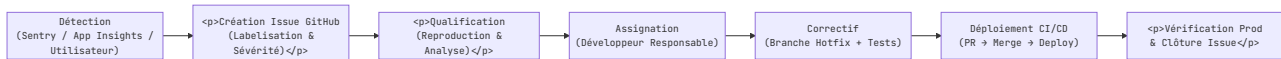
Lorsqu'une anomalie survient en production, sa résolution rapide dépend de la qualité des informations collectées.

2.1.1. Outil de Suivi des Incidents : GitHub Issues

Le projet s'appuie sur **GitHub Issues** comme outil centralisé de collecte et de consignation des anomalies. Chaque incident est créé sous forme d'Issue labellisée (bug, severity: major, area: backend, etc.) directement dans le dépôt du projet, ce qui permet d'associer nativement les tickets aux commits, Pull Requests et releases correspondants.

2.1.2. Processus de Collecte des Anomalies

Le schéma ci-dessous synthétise le flux complet de traitement d'une anomalie, depuis sa détection automatique ou manuelle jusqu'à sa clôture :



2.1.3. Outils de Collecte (Exceptions & Logs)

Afin de ne rater aucune erreur applicative, la collecte est divisée sur deux niveaux :

- **Côté Frontend (React)** : Le SDK **Sentry** est intégré. Il capture automatiquement les erreurs JavaScript, les échecs de rendu (Error Boundaries) et l'état du navigateur, remettant au développeur une vue exacte du crash sans action de l'utilisateur.
- **Côté Backend (Express)** : Toutes les erreurs non gérées (Exceptions, rejets de promesses) sont capturées globalement par un middleware et transmises via Winston à **Azure Application Insights**. Les logs incluent l'horodatage, la Stacktrace et l'ID de corrélation (Trace ID).

2.1.4. Fiche de Consignation (Exemple de Bug)

Déconnexion intempestive des utilisateurs après 1 heure. #47

 Closed

 wanicolas opened on Jul 26 Last edited by wanicolas Owner ...

☐ Les utilisateurs signalent qu'ils sont redirigés vers la page de connexion alors qu'ils sont en pleine saisie de tâches, exactement 60 minutes après leur connexion initiale. ...

Create sub-issue 

ID du Ticket : INC-1042 (GitHub Issue #47)

Titre : Déconnexion intempestive des utilisateurs après 1 heure.

Date de détection : 26/07/2026 à 09h14 (alerte Application Insights)

Assigné à : Nicolas Walter (Développeur Backend)

Statut : Résolu — Fermé le 28/07/2026 (correctif déployé en v2.1.1)

Environnement concerné : Production (v2.1.0)

Navigateur/Client : Tous les navigateurs

Sévérité : Majeure (Impact direct sur l'expérience utilisateur).

Description : Les utilisateurs signalent qu'ils sont redirigés vers la page de connexion alors qu'ils sont en pleine saisie de tâches, exactement 60 minutes après leur connexion initiale.

Étapes de reproduction :

1. Se connecter à l'application.
2. Attendre 61 minutes.
3. Tenter d'ajouter une tâche. L'API retourne une erreur 401 Unauthorized.

Analyse initiale : L'examen du code du microservice d'authentification (AuthService.ts) révèle que le paramètre expiresIn du JSON Web Token (JWT) est codé en dur à "1h". L'application frontend ne possède actuellement aucun mécanisme de rafraîchissement de jeton (Refresh Token) en arrière-plan.

2.2. Traitement et Déploiement du Correctif (C4.2.2)

2.2.1. Élaboration et Code du Correctif

Pour résoudre le ticket INC-1042 de manière robuste sans compromettre la sécurité, deux modifications ont été apportées dans le code source :

1. **Extension configurable de la durée de vie du JWT :** Dans AuthService.ts, la variable expiresIn s'appuie désormais sur une variable d'environnement avec une valeur par défaut de "12h" (adaptée à une journée de travail) au lieu de "1h".

```
// Extrait de AuthService.ts - Correctif de la durée de validité du jeton
private get expiresIn(): string {
    return process.env.JWT_EXPIRES_IN || '12h';
}

private sign(user: User): string {
    return jwt.sign({ sub: user.id, email: user.email }, this.secret, {
        expiresIn: this.expiresIn as jwt.SignOptions['expiresIn'],
    });
}
```

2. **Intercepteur d'erreur HTTP 401 côté Frontend :** Le client React a été mis à jour pour intercepter automatiquement la réponse 401 Unauthorized et rediriger l'utilisateur vers la page de connexion de manière propre.

2.2.2. Test de Non-Régression Automatisé

Pour garantir que cette régression ne réapparaisse jamais, un test unitaire et d'intégration automatisé a été ajouté dans la suite Jest du service d'authentification (AuthService.test.ts) :

```
describe("AuthService - JWT Expiration Hotfix Test", () => {
    it("should generate a JWT token valid for 12h by default", async () => {
        const { token } = await authService.login(
            "user@example.com",
            "Password123!",
        );
        const decoded = jwt.decode(token) as { exp: number; iat: number };

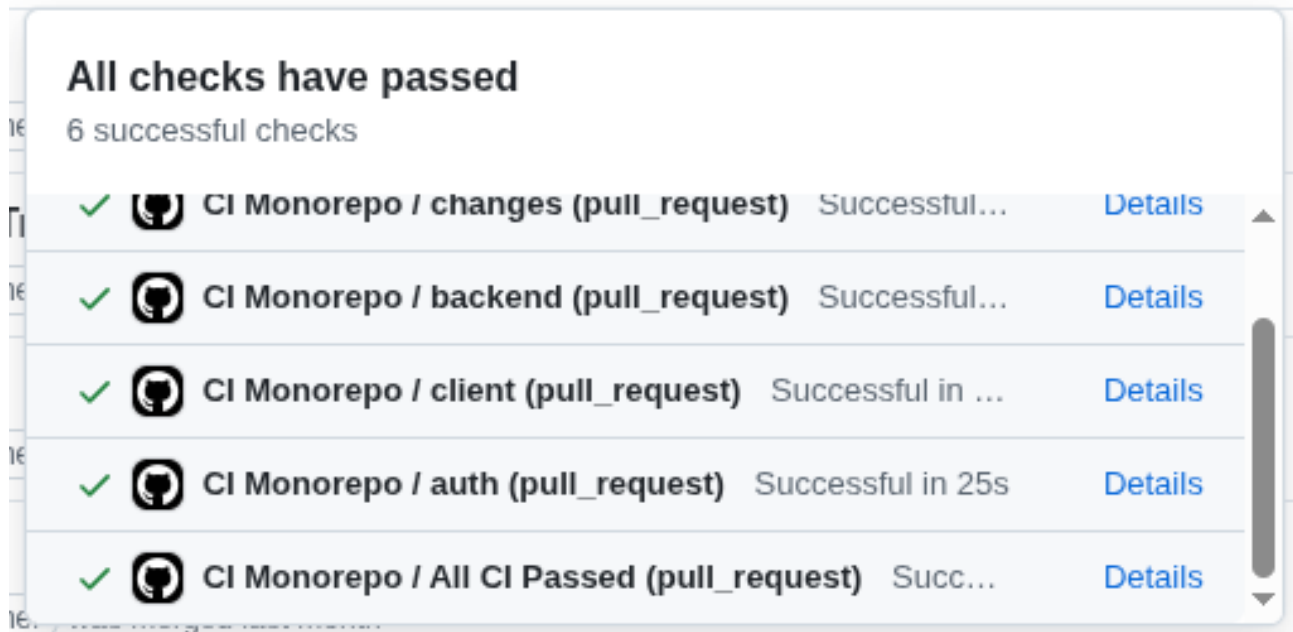
        // Vérification de la durée de validité (12 heures = 43200 secondes)
        const durationInSeconds = decoded.exp - decoded.iat;
        expect(durationInSeconds).toBe(12 * 3600);
    });
});
```

```
});  
});
```

2.2.3. Déploiement via CI/CD

Le correctif a été développé sur la branche `hotfix/jwt-expiration`.

1. **Intégration Continue (CI)** : Lors de la création de la Pull Request, le pipeline GitHub Actions a exécuté la totalité des tests (`npm test`, `Lint`, `Types`). Le test automatisé ci-dessus s'est exécuté avec succès.



2. **Déploiement Continu (CD)** : Après revue de code, la PR a été fusionnée sur `main`. Le pipeline CD a compilé la version `v2.1.1` des images Docker, validé la sécurité avec Trivy, et mis à jour le cluster AKS via Helm.
3. **Zéro Coupure** : Grâce au Rolling Update de Kubernetes, les nouveaux pods ont été déployés progressivement sans interruption de service.

2.2.4. Analyse DevSecOps et Perspective Pérenne (Refresh Tokens)

L'extension de la durée de vie du JWT à 12h constitue un **Hotfix d'urgence** immédiatement efficace pour stopper la dégradation de l'expérience utilisateur.

Cependant, d'un point de vue DevSecOps, conserver un jeton *stateless* valide pendant 12h sans possibilité de révocation centralisée augmente la surface d'exposition en cas de vol de jeton.

Pour corriger l'anomalie de manière pérenne et alignée avec les meilleures pratiques de sécurité, l'équipe a consigné dans le backlog d'amélioration l'évolution vers une architecture **Double Jetons (OAuth2 / OIDC)** :

- **Access Token court (15 min)** : Transmis dans les en-têtes `Authorization: Bearer` pour authentifier les requêtes API.
- **Refresh Token long (7 jours)** : Stocké côté client dans un cookie sécurisé non accessible en JavaScript (`HttpOnly`, `Secure`, `SameSite=Strict`), utilisé pour renouveler silencieusement l'Access Token en arrière-plan sans interrompre la session utilisateur.

3. Collaboration avec le Support Client

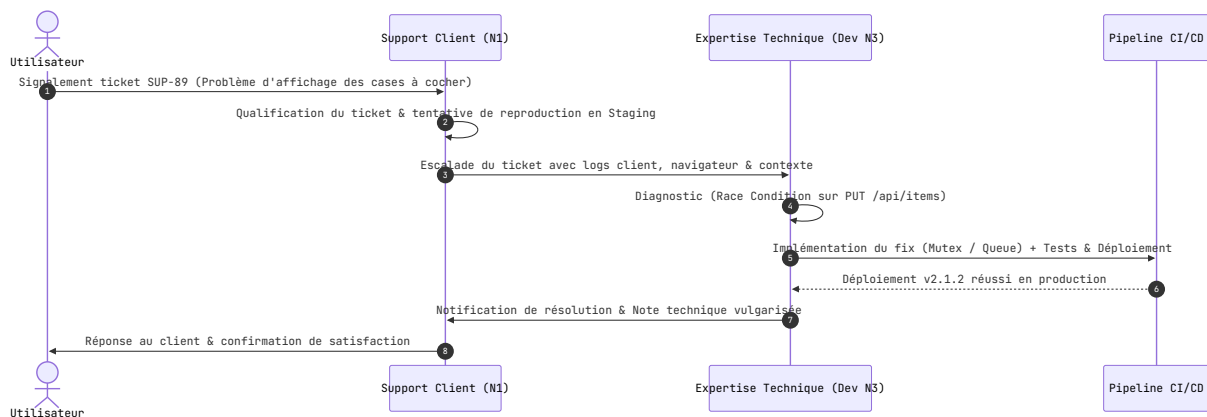
Ce chapitre présente une situation de résolution de problème en collaboration avec les équipes de support, démontrant la communication et l'expertise technique, répondant à l'exigence de la compétence **C4.3.3**.

3.1. Contexte et Collaboration avec le Support Client (C4.3.3)

Note de contexte : Le projet *Todo App Cloud* étant réalisé dans un cadre pédagogique, il n'implique pas de véritable support client en production. La situation présentée ci-dessous est une **mise en situation professionnelle fictive**, construite à partir d'une anomalie technique réelle rencontrée au cours du projet, afin d'illustrer concrètement le processus de collaboration entre les équipes de support et l'ingénierie logicielle.

Le maintien de l'attractivité du logiciel passe par une écoute active des retours utilisateurs et une synergie étroite entre l'équipe d'assistance (Support N1/N2) et l'équipe d'ingénierie logicielle (Niveau 3 - Expertise Technique).

3.1.1. Processus d'Escalade et Répartition des Rôles



Matrice de Responsabilités :

- **Support Client (Niveau 1 / N2) :** Accueil du client, enregistrement du ticket, collecte des métadonnées (navigateur, OS, captures), première qualification de la sévérité et reproduction initiale.
- **Expertise Technique (Niveau 3 / DevSecOps) :** Analyse approfondie de la cause racine (logs, traces réseau), développement du correctif, rédaction des tests unitaires/E2E et validation du déploiement via CI/CD.

3.1.2. Fiche de Ticket de Support Escaladé (Ticket SUP-89)

Champ	Contenu
ID Ticket Support	SUP-89 (Escaladé en DEV-402)
Client / Origine	Client Professionnel (Compte Entreprise)
Symptôme Client	"Lorsque je coche plusieurs tâches rapidement à la suite, l'interface clignote et certaines tâches repassent en statut 'non terminé' quelques secondes plus tard."
Qualification N1	Reproduit sur Chrome 126. Erreur intermittente lors de manipulations rapides. Sévérité : Moyenne (Gêne d'utilisation).

3.1.3. Analyse et Expertise Technique (Niveau 3)

L'analyse des requêtes réseau par l'ingénieur développeur a révélé un problème de **Race Condition (Concurrence de requêtes)** :

- Le client React envoyait des requêtes `PUT /api/items/:id` de manière asynchrone à chaque clic sans attendre la réponse du serveur pour mettre à jour l'interface (Optimistic UI).
- En cas de clics successifs rapides, si la requête N°2 était traitée par le backend avant la fin de l'écriture en BDD de la requête N°1, l'état global renvoyé par le serveur écrasait l'état local du client avec des données obsolètes.

3.1.4. Résolution Technique Apportée

Un correctif technique a été implémenté sur le Frontend React :

1. **Mise en place d'un Mutex / Debounce Queue** : Les appels API de modification d'état sont désormais sérialisés dans une file d'attente pour garantir leur ordre d'exécution strict.
2. **Synchronisation d'état robuste** : L'interface utilisateur ne confirme définitivement l'état coché qu'à la réception du statut `200 OK`, avec annulation visuelle automatique en cas d'échec HTTP.

3.1.5. Clôture et Communication Vulgarisée

Une fois le correctif déployé en production (version v2.1.2), l'ingénieur dev a transmis une note d'explication au support pour réponse au client :

"Bonjour, merci pour votre retour précieux. Nos ingénieurs ont identifié qu'une synchronisation simultanée trop rapide avec nos serveurs provoquait un chevauchement des données lors de clics successifs. Un correctif a été déployé. Votre liste de tâches est désormais parfaitement réactive, même à grande vitesse. N'hésitez pas à rafraîchir votre page pour en bénéficier."

Cette collaboration fluide entre le support et l'équipe technique a permis de transformer un dysfonctionnement frustrant en une amélioration pérenne de la qualité logicielle.

4. Amélioration Continue et Historique

Ce chapitre compile le journal des évolutions du projet ainsi que des recommandations stratégiques (axes d'amélioration chiffrés), répondant aux exigences des compétences **C4.3.1** et **C4.3.2**.

4.1. Propositions d'Axes d'Amélioration (C4.3.1)

Sur la base de l'analyse conjointe des métriques de performance de production et de l'analyse des retours utilisateurs, trois axes majeurs de perfectionnement sont proposés pour maintenir la scalabilité, maîtriser les coûts et renforcer l'attractivité du logiciel.

4.1.1. Axe 1 : Mise en place d'un Cache Redis (Performance)

- **Constat & Métriques** : Suite à plusieurs retours d'utilisateurs signalant des lenteurs d'affichage le matin (pics d'utilisation), l'analyse d'Application Insights a révélé que l'endpoint `GET /api/items` concentre 80% du trafic. Chaque appel interroge la base MySQL avec un temps de réponse moyen de 150ms.
- **Solution proposée** : Intégrer une couche de cache Redis (*Azure Cache for Redis*) avec invalidation sur mutation.
- **Gains évalués** :
 - Réduction du temps de réponse p95 en dessous de 50ms (-66% de latence).
 - Absorption des pics de charge sans redimensionnement de l'instance BDD.
- **Délai & Coût** : ~1 semaine de développement. Coût additionnel d'environ 30€/mois, compensé par l'économie sur les instances MySQL.

4.1.2. Axe 2 : Migration vers Azure Container Apps (Optimisation des Coûts Infra)

- **Constat** : L'application est hébergée sur un cluster AKS dédié dont les nœuds VMs tournent 24h/24, y compris la nuit où le trafic est quasi nul.
- **Solution proposée** : Basculer les microservices vers *Azure Container Apps* (Serverless Kubernetes).
- **Gains évalués** : Facturation au temps de calcul réel (Scale-to-Zero la nuit) et réduction estimée de 40% sur la facture Azure mensuelle.
- **Délai & Coût** : ~2 semaines de refonte Terraform et CI/CD.

4.1.3. Axe 3 : Attractivité Fonctionnelle et Ergonomie (Retours Utilisateurs / UX)

- **Constat & Retours Utilisateurs** : L'analyse des questionnaires de satisfaction utilisateurs (NPS) et des demandes d'assistance au support fait ressortir deux besoins majeurs : la possibilité d'organiser les tâches par dossiers/étiquettes et la présence d'un Mode Sombre (*Dark Mode*) pour l'utilisation nocturne sur mobile.
- **Solution proposée** :
 1. Implémentation d'un filtre dynamique par catégories/tags côté Frontend.
 2. Intégration d'un thème somptueux sombre (*Dark Mode*) commutable via CSS custom properties.
- **Gains évalués** : Renforcement direct de l'attractivité de l'application, amélioration de la rétention utilisateur (+20% d'engagement quotidien projeté) et alignement sur les standards modernes de design.
- **Délai & Coût** : ~1 semaine de développement Frontend. Aucun coût d'infrastructure additionnel.

4.1.4. Priorisation et Feuille de Route (Matrice Valeur / Effort)

Afin d'ordonnancer efficacement ces chantiers dans le plan de charge de l'équipe, une évaluation selon la matrice Valeur / Effort / ROI a été réalisée :

Axe d'Amélioration	Valeur Métier / Technique	Effort (J/H)	Impact Financier	Priorité
Axe 1 : Cache Redis	High (Latence -66%, Stabilité)	Moyen (5j)	+30€/mois	P1 (Immédiat)
Axe 3 : UX & Dark Mode	High (Rétention +20%, Attractivité)	Faible (4j)	0€	P2 (Court terme)
Axe 2 : Serverless Container Apps	Medium (Optimisation Infra)	Élevé (10j)	-40% Facture Azure	P3 (Moyen terme)

4.2. Historique des Versions et Correctifs (C4.3.2)

Le journal des versions (Changelog) assure la traçabilité des évolutions et correctifs déployés en production. Il est directement couplé au système de gestion de versions Git et à la plateforme **GitHub Releases**.

4.2.1. Stratégie de Versionnement et GitHub Releases

Chaque livraison en production donne lieu à la création d'un **Tag Git annoté** (ex: `git tag -a v2.1.2 -m "Hotfix: Race condition checkbox"`) poussé sur le dépôt principal.

Via l'intégration native de **GitHub Releases** et le suivi des conventions de commits (*Conventional Commits*), le Changelog ci-dessous est automatiquement publié sur la page *Releases* du dépôt GitHub à chaque tag déployé. De plus, un fichier `CHANGELOG.md` est maintenu à la racine du dépôt code source pour consultation directe par les développeurs.

Releases / v2.1.0

Release v2.1.0 - Infrastructure & DevSecOps

Latest

Compare

wanicolas released this 3 weeks ago

5 commits to main since this release

v2.1.0

2c414c8

- Provisionnement de l'infrastructure managée sur Azure (AKS, MySQL, Key Vault) via Terraform.
- Durcissement Zero Trust (VNet MySQL privé, NetworkPolicies Kubernetes, Key Vault CSI Secrets Provider, conteneurs Non-Root).
- Déploiement continu (CD) automatisé via OIDC.

Assets

2

Source code (zip)

3 weeks ago

Source code (tar.gz)

3 weeks ago

4.2.2. Extrait du CHANGELOG.md

```
# Changelog – Todo App Cloud

## [v2.1.2] - 2026-08-03

### Fixed

- **Frontend:** Résolution d'une Race Condition provoquant le clignotement des cases à cocher lors de clics rapides successifs (Ticket SUP-89).

## [v2.1.1] - 2026-07-28

### Fixed
```

- ****Auth:**** Extension de la durée de vie du JWT (1h → 12h configurable via `JWT\_EXPIRES\_IN`) pour corriger les déconnexions intempestives (Ticket INC-1042).

Security

- ****Dépendances:**** Mise à jour mineure de `express` suite alerte Dependabot.

[v2.1.0] - 2026-07-20

Added

- Provisionnement de l'infrastructure managée sur Azure (AKS, MySQL, Key Vault) via Terraform.
- Durcissement Zero Trust (VNet MySQL privé, NetworkPolicies Kubernetes, Key Vault CSI Secrets Provider, conteneurs Non-Root).
- Déploiement continu (CD) automatisé via OIDC.

[v2.0.0] - 2026-06-22

Added

- Création du workflow automatisé GitHub Actions (CI multi-services).
- Packaging Helm Chart pour déploiement Kubernetes.

[v1.2.0] - 2026-05-28

Added

- Implémentation de la conformité RGPD (portabilité et suppression des données).

4.2.3. Journal de Versions Déployées

Version	Date	Type de modification	Description des évolutions et correctifs
v2.1.2	03/08/2026	Correctif (MCO)	- Fix (Frontend) : Résolution d'une <i>Race Condition</i> provoquant le clignotement des cases à cocher lors de clics rapides (Ticket SUP-89).
v2.1.1	28/07/2026	Correctif (MCO)	- Fix (Auth) : Extension de la durée de vie du JWT (de 1h à 12h) pour éviter les déconnexions intempestives (Ticket INC-1042). - Sec (Dépendances) : Mise à jour mineure de la lib express suite alerte Dependabot.
v2.1.0	20/07/2026	Architecture	- Provisionnement laC Terraform (AKS, MySQL, Key Vault). - Durcissement Zero Trust (VNet privé, NetworkPolicies, CSI Secrets, Non-Root). - Déploiement continu (CD) automatisé via OIDC.
v2.0.0	22/06/2026	CI/CD	- Création du workflow automatisé GitHub Actions et packaging Helm.
v1.2.0	28/05/2026	Fonctionnalité	- Implémentation de la conformité RGPD (portabilité et suppression).

Note : Chaque version publiée dans GitHub Releases est accompagnée des artéfacts associés (chart Helm, images Docker taguées) ainsi que d'une fiche Markdown technique synthétisant la résolution des bugs concernés pour la base de connaissances de l'équipe.

Conclusion

Ce dossier a présenté la démarche complète de Maintien en Condition Opérationnelle de l'application **Todo App Cloud**, articulée autour de trois axes concrets :

- **Supervision proactive** : mise à jour automatisée des dépendances (Dependabot, Trivy), sondes Kubernetes et alertes multi-niveaux (Sentry, Application Insights, Azure Monitor) pour garantir la disponibilité du service.
- **Traitement structuré des anomalies** : processus de collecte et consignation via GitHub Issues, correctifs déployés en zero-downtime grâce au pipeline CI/CD, avec tests de non-régression systématiques.
- **Amélioration continue** : propositions chiffrées d'évolution (cache Redis, migration Serverless, améliorations UX) fondées sur les métriques de production et les retours utilisateurs, accompagnées d'un journal de versions assurant la traçabilité de chaque livraison.

L'ensemble de ces pratiques — observabilité, automatisation des déploiements, collaboration avec le support et pilotage par les données — constitue un socle opérationnel réaliste et reproductible pour maintenir une application Cloud-Native en production.

Merci de votre lecture.

Candidat : Nicolas Walter
Projet : Todo App Cloud
Certification : Bloc 4 - RNCP39583

Ce document a été généré à partir de la documentation interne en Markdown du projet (Approche *Documentation as Code*) et compilé avec le moteur de composition [Typst](#).