Compte-rendu d'amélioration de l'architecture — Projet Fil Rouge

Grille de notation n°3 — BC01 : Administrer et optimiser les systèmes d'exploitation et la virtualisation pour la sécurité et la performance

Objectif : Analyser l'architecture existante et son PCA pour proposer des améliorations et des outils de monitoring adaptés.

Table des matières

Contexte de l'architecture

Infrastructure analysée

Le projet **RP Construction System** repose sur une infrastructure conteneurisée déployée sur un VPS Hostinger (16 Go RAM, Ubuntu 22.04) :

Service	Technologie	Rôle	Ressources
Serveur de jeu	Garry's Mod via Docker (`ceifa/garrysmod`)	Héberge le serveur DarkRP + addon	3 Go RAM, 2 CPUs
Base de données	MySQL 8.0 (Docker)	Logs, futur partage de blueprints	512 Mo RAM, 0.5 CPU
Orchestration	Docker Compose v2	Gestion des deux services	—
Addon	RP Construction System v2.2	Code métier (Lua)	Bind mount
Versioning	Git + GitHub	Code source, config, documentation	—

Compétences validées

Compétence	Domaine	Document
C5	Redondance, réplication, clustering	redondance.md
C6	Surveillance et optimisation des performances	monitoring.md
C7	Gestion des incidents informatiques	incidents.md
C8	Automatisation des tâches d'administration	automatisation.md

Méthodologie

L'approche suivie s'inspire du cycle **PDCA** (Plan-Do-Check-Act) et de la norme **ISO 27001** pour la gestion de la sécurité :

1. **Plan** — Identifier les risques via une matrice, définir le PCA et les SLA
2. **Do** — Mettre en place les solutions (scripts, configurations, procédures)
3. **Check** — Vérifier l'efficacité via monitoring, métriques et tests
4. **Act** — Ajuster et documenter les retours d'expérience (post-mortem)

🔁 Redondance, réplication et clustering — C5

> **C5.1** — Pertinence des améliorations proposées (matrice de risques, solutions) **C5.2** — Qualité du PCA modifié (continuité opérationnelle)

1. Matrice de risques (C5.1)

Identification des actifs critiques

Actif	Criticité	Propriétaire	Données
Serveur GMod	Critique	Admin serveur	État du jeu, joueurs connectés
Base MySQL	Haute	Admin serveur	Logs, historique des actions
Code addon (Lua)	Haute	Développeur	Code source, configuration
Images Docker	Haute	Admin serveur	~8 Go Workshop + modules
Configuration DarkRP	Moyenne	Admin serveur	Jobs, entities, shipments
Blueprints joueurs	Basse (côté client)	Joueurs	Fichiers `.dat` locaux

Matrice de risques

#	Risque	Probabilité	Impact	Gravité	Stratégie de mitigation
R1	**Crash du serveur GMod**	Moyenne	Critique	Élevée	Restart auto Docker (`restart: unless-stopped`), snapshots Docker
R2	**Panne MySQL**	Faible	Haute	Moyenne	Healthcheck Docker, mode dégradé (addon fonctionne sans DB)
R3	**Corruption Workshop (~8 Go)**	Faible	Critique	Élevée	Workshop commitée dans l'image Docker, rollback instantané
R4	**Perte complète du VPS**	Très faible	Critique	Critique	Git distant (GitHub), export images Docker, dumps SQL
R5	**Exploitation de vulnérabilité net message**	Faible	Haute	Moyenne	Rate limiting, validation serveur, logging
R6	**Bug Lua bloquant (addon cassé)**	Moyenne	Haute	Moyenne	Rollback Git, snapshots Docker, tests pré-déploiement
R7	**Saturation mémoire VPS**	Faible	Haute	Moyenne	Limites Docker (`mem_limit`), monitoring `docker stats`
R8	**Perte de données MySQL**	Faible	Moyenne	Faible	Volume Docker persistant, dumps réguliers

Solutions de redondance proposées

Risque	Solution en place	Amélioration proposée
R1	`docker compose up -d` manuel	`restart: unless-stopped` + script watchdog
R2	Healthcheck MySQL 10s	Réplication master-slave MySQL
R3	Image Docker commitée (8 Go inclus)	Push vers Docker Hub / registry privé
R4	Git + GitHub	+ Export images Docker + dump SQL vers S3/ Backblaze
R5	Rate limiting `sv_security.lua`	WAF niveau réseau (iptables)
R6	Git revert	Pipeline CI/CD avec tests avant déploiement
R7	`mem_limit: 3g` / `mem_limit: 512m`	Alerting Prometheus quand RAM > 80%
R8	Volume Docker local	Dump MySQL automatisé (cron) + réplication

2. Plan de Continuité d'Activité — PCA (C5.2)

Objectifs du PCA

Métrique	Objectif actuel	Objectif cible
RTO (Recovery Time Objective)	< 5 min (restart Docker)	< 2 min (restart auto)
RPO (Recovery Point Objective)	Dernier commit Git	< 1h (dumps MySQL horaires)
Disponibilité cible	95%	99%
MTTR (Mean Time To Repair)	~10 min (diagnostic + restart)	< 5 min (procédure documentée)

Scénarios de continuité

Scénario 1 : Crash du container GMod

```
Détection : Docker healthcheck ou absence de joueurs
Temps de détection : < 30 secondes (avec restart: unless-stopped)
Action automatique : Docker redémarre le container
Action manuelle (si échec) :
  $ docker compose down && docker compose up -d
RTO : < 2 minutes
Impact : Déconnexion temporaire des joueurs, reconnexion automatique
```

Scénario 2 : Panne MySQL

```
Détection : Healthcheck mysqladmin ping (toutes les 10s)
Impact immédiat : L'addon passe en mode dégradé (pas de logs DB)
  → Les fonctionnalités core (blueprints, ghosts, caisses) continuent
  → Seul le logging en DB est interrompu
Action : docker restart gmod-mysql
RTO : < 1 minute
RPO : Aucune perte (logs console toujours actifs)
```

Scénario 3 : Perte complète du VPS

```
Détection : Monitoring externe (ping port 27015)
Plan de reprise :
  1. Provisionner un nouveau VPS (Hostinger, ~10 min)
  2. Installer Docker + Docker Compose (~5 min)
  3. git clone https://github.com/yguerch212-creator/Projet_fil_rouge.git
  4. Restaurer l'image Docker depuis le backup exporté
     $ docker load < backup-gmod-v2.2-vehicles.tar
  5. Restaurer le dump MySQL
     $ docker exec -i gmod-mysql mysql -u root -p < backup.sql
  6. docker compose up -d
RTO : < 30 minutes
RPO : Dernier backup (objectif : < 1 heure)
```

Scénario 4 : Corruption de l'addon (bug bloquant)

```
Détection : Erreurs Lua dans les logs serveur, signalements joueurs
Action immédiate :
  $ cd /root/ProjetFilRouge
  $ git log --oneline -5       # Identifier le commit fautif
  $ git revert HEAD            # Annuler le dernier commit
  $ docker restart gmod-server
RTO : < 5 minutes
RPO : Aucune perte (Git conserve tout l'historique)
Alternative : Changer le tag Docker dans docker-compose.yml
  image: projetfilrouge/gmod-server:v2.1-stable
```

Redondance des données — État actuel

Donnée	Stockage primaire	Redondance	Niveau
Code source addon	Bind mount VPS	Git + GitHub (distant)	Élevé
Configuration DarkRP	Bind mount VPS	Git + GitHub	Élevé
docker-compose.yml	VPS	Git + GitHub	Élevé
Images Docker (8 Go)	Stockage local VPS	`docker commit` (local)	⚠ Moyen
Base MySQL	Volume Docker local	Aucune réplication	⚠ Moyen
Workshop Collection	Dans l'image Docker	Commitée = persistante	Élevé
Blueprints joueurs	PC client (`data/`)	Hors périmètre serveur	ℹ N/A

Améliorations réalisées pour la continuité

1. Stratégie de snapshots Docker (rollback instantané)

Convention de nommage sémantique :

```
projetfilrouge/gmod-server:v{major}.{minor}-{description}
```

Image	Contenu	Taille
`v1.0-base`	GMod + DarkRP + 101 addons Workshop	~8 Go
`v1.1-mysql`	+ MySQLOO 64-bit	~8.1 Go
`v2-stable`	+ Addon v2.0 (ghosts + caisses)	~8.1 Go
`v2.1-stable`	+ Import AD2, UI refonte	~8.1 Go
`v2.2-vehicles`	+ Véhicules simfphys	~8.2 Go

Rollback : changer le tag dans `docker-compose.yml` → `docker compose up -d` → serveur restauré en ~30 secondes.

2. Mode dégradé MySQL

L'addon a été conçu pour fonctionner **sans base de données**. Si MySQL est indisponible : - Les blueprints continuent de fonctionner (stockage client) - Les ghosts et caisses fonctionnent normalement - Seul le logging en base est désactivé (les logs console restent actifs)

Ce design élimine MySQL comme **SPOF** (Single Point of Failure) pour les fonctionnalités critiques.

3. Healthcheck et démarrage ordonné

```
# docker-compose.yml
services:
  mysql:
    healthcheck:
      test: mysqladmin ping -h localhost
      interval: 10s
      timeout: 5s
      retries: 3
  gmod:
    depends_on:
      mysql:
        condition: service_healthy
```

Le serveur GMod ne démarre que lorsque MySQL est `healthy`, évitant les erreurs de connexion au démarrage.

Perspectives de clustering

Horizon	Solution	Bénéfice
Court terme	`restart: unless-stopped` dans Docker Compose	Restart automatique après crash
Moyen terme	Réplication MySQL master-slave (Docker Compose)	Lecture sur slave, failover
Moyen terme	Push images vers Docker Hub	Redondance géographique des snapshots
Long terme	Docker Swarm / Kubernetes	Orchestration multi-nœuds, HA
Long terme	Multi-VPS avec load balancer	Haute disponibilité géographique

📊 Monitoring et performances — C6

> **C6.1** — Pertinence du choix de l'outil de monitoring (SLA, PRA) **C6.2** — Sélection des données à monitorer

1. Définition des SLA et PRA (C6.1)

SLA (Service Level Agreement)

Engagements de niveau de service définis pour l'infrastructure du serveur de jeu :

Métrique SLA	Objectif	Mesure	Outil
Disponibilité	≥ 95% (hors maintenance planifiée)	Uptime mensuel	UptimeRobot / ping externe
Temps de réponse	Latence < 100ms (réseau)	Ping serveur	Monitoring réseau
Temps de démarrage	< 2 min après arrêt	Logs Docker timestamps	`docker logs`
Capacité joueurs	32 slots simultanés	Compteur Source Engine	Query port 27015
Perte de données	RPO < 1h pour MySQL	Dernière sauvegarde	Script cron dump SQL

PRA (Plan de Reprise d'Activité)

Le PRA définit les procédures de reprise après sinistre, complémentaire au PCA (voir redondance.md) :

Scénario	RTO cible	RPO cible	Procédure
Crash container	< 2 min	0 (état en mémoire perdu)	Restart auto Docker
Panne MySQL	< 1 min	0 (mode dégradé immédiat)	Healthcheck + restart
Perte VPS	< 30 min	< 1h (dernier dump)	Rebuild depuis Git + image + dump
Bug bloquant	< 5 min	0	Git revert + restart
Corruption image Docker	< 5 min	Dernier snapshot	Rollback tag image

Choix de l'outil de monitoring — Argumentation

Comparatif des solutions envisagées

Critère	Docker natif	Prometheus + Grafana	Portainer	Zabbix
Coût	Gratuit (inclus)	Gratuit (open source)	Gratuit (CE)	Gratuit (open source)
Complexité	Très faible	Moyenne	Faible	Élevée
Ressources	Aucune	~500 Mo RAM	~200 Mo RAM	~1 Go RAM
Métriques Docker	□ Basique	□ Complet	□ Bon	□ Complet
Métriques applicatives	□	□ Custom	□	□ Custom
Alerting	□	□ AlertManager	□ Webhooks	□ Intégré
Dashboard	CLI uniquement	□ Grafana	□ Web UI	□ Web UI
Adapté au projet	Phase dev	Phase production	Phase intermédiaire	Surdimensionné

Solution retenue : monitoring multi-couches

Phase actuelle (développement) : Docker natif + logging applicatif intégré - Justification : pas de surcharge mémoire sur un VPS partagé, suffisant pour le développement - Le serveur de jeu utilise déjà 3 Go RAM, MySQL 512 Mo → peu de marge pour des agents monitoring lourds

Phase cible (production) : Prometheus + Grafana + AlertManager - Justification : solution standard de l'industrie, open source, écosystème riche - `cAdvisor` pour les métriques Docker (CPU, RAM, réseau par container) - Exporteur custom Lua pour les métriques applicatives GMod - AlertManager pour les notifications Discord/Telegram - Grafana pour les dashboards visuels

Cette approche progressive permet d'adapter le monitoring à la maturité de l'infrastructure sans gaspiller des ressources.

2. Données à monitorer (C6.2)

Couche infrastructure

Métrique	Source	Seuil d'alerte	Criticité
CPU par container	`docker stats` / cAdvisor	> 80% pendant 5 min	Haute
RAM par container	`docker stats` / cAdvisor	> 90% du `mem_limit`	Critique
RAM GMod	`docker stats`	> 2.7 Go (sur 3 Go limit)	Critique
RAM MySQL	`docker stats`	> 450 Mo (sur 512 Mo limit)	Haute
Espace disque	`df -h` / node_exporter	> 85%	Critique
État container	`docker ps` / healthcheck	Container `unhealthy` ou `exited`	Critique
Réseau	`docker stats` / iptables	Trafic anormal (> 100 Mbps)	Haute
Latence MySQL	Healthcheck interval	Ping > 5s	Haute

Couche applicative (serveur GMod)

Métrique	Source	Seuil d'alerte	Criticité
Joueurs connectés	Query port 27015	0 pendant heures de pointe	Info
Actions/minute	`sv_logging.lua` (MySQL)	> 100/min (possible abus)	Haute
Rate limit hits	`sv_security.lua`	> 10 rejets/min pour 1 joueur	Haute
Erreurs Lua	Console serveur (`ERROR`)	Toute erreur	Moyenne
Ghosts actifs	`sv_ghosts.lua`	> 200 simultanés (perf)	Haute
Blueprints chargés	`sv_blueprints.lua`	> 500 props en 1 blueprint	Info
Net messages/s	Monitoring réseau GMod	> 50/s pour 1 joueur	Critique

Couche base de données

Métrique	Source	Seuil d'alerte	Criticité
Connexions actives	`SHOW PROCESSLIST`	> 10 connexions	Haute
Slow queries	`slow_query_log`	> 5 requêtes > 1s/heure	Haute
Taille base	`information_schema`	> 1 Go	Info
Requêtes/seconde	`SHOW STATUS`	> 100 QPS	Info
Uptime	`mysqladmin status`	Restart inattendu	Critique

3. Monitoring en place

Monitoring applicatif : sv_logging.lua

Le module de logging intégré à l'addon trace toutes les actions significatives :

```
[Construction] [SAVE] Player "Thomas" saved blueprint "base_militaire" (45 props)
[Construction] [LOAD] Player "Thomas" loaded blueprint "base_militaire"
[Construction] [MATERIALIZE] Player "Alex" materialized ghost (crate: 49 remaining)
[Construction] [VEHICLE] Player "Thomas" loaded crate onto sim_fphy_codww2opel
[Construction] [SECURITY] Rate limit hit for STEAM_0:0:12345 on "save" (cooldown: 10s)
```

Double destination (version dev) : - **Console serveur** : visibilité immédiate pour l'admin - **Base MySQL** (`blueprint_logs`) : historique persistant, requêtable via SQL

Données loguées :

Champ	Type	Contenu
`steamid`	VARCHAR	Identifiant unique du joueur
`player_name`	VARCHAR	Nom affiché
`action`	ENUM	save, load, delete, share, materialize
`blueprint_name`	VARCHAR	Nom du blueprint concerné
`details`	JSON	Contexte additionnel
`created_at`	TIMESTAMP	Horodatage précis

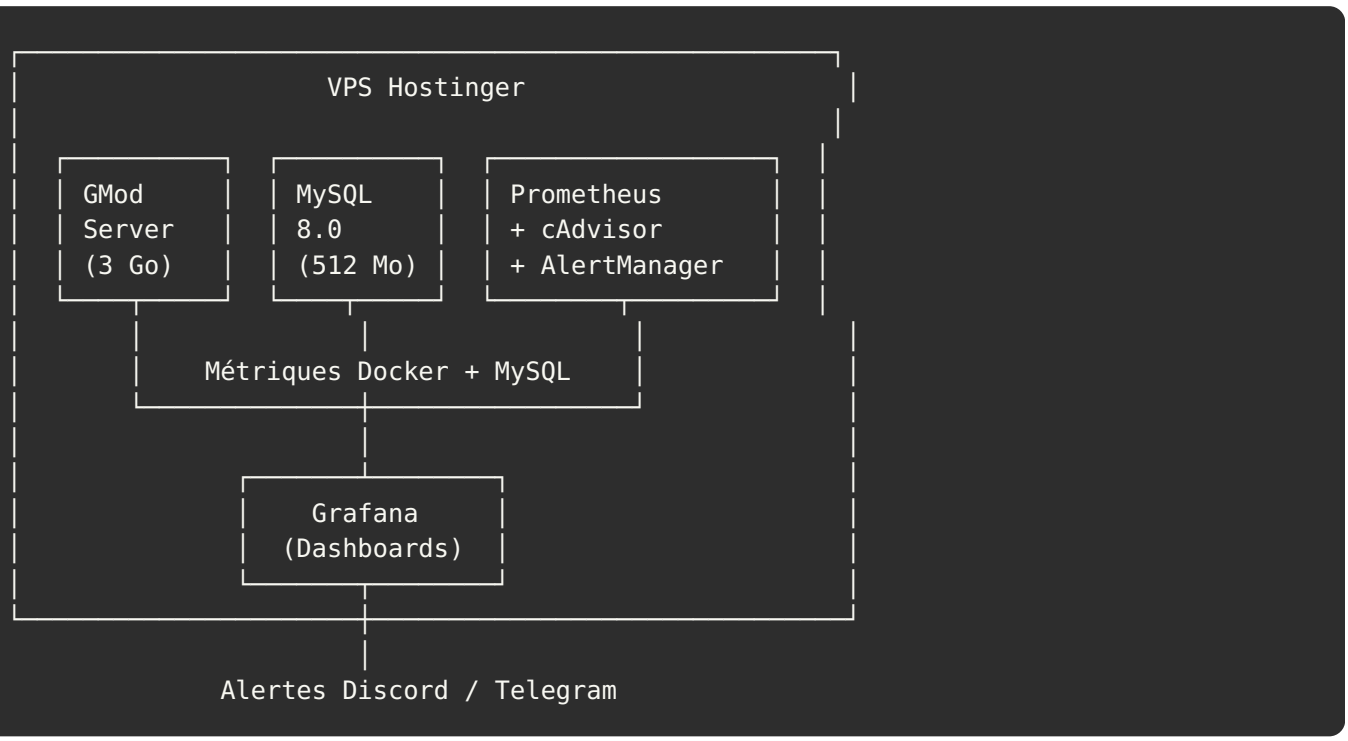
Monitoring infrastructure : Docker natif

```
# État des containers et healthcheck
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"

# Consommation temps réel (CPU, RAM, réseau, I/O)
docker stats --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.NetIO}}"

# Vérifier le healthcheck MySQL
docker inspect gmod-mysql --format='{{.State.Health.Status}}'

# Logs temps réel du serveur
docker logs -f --tail 100 gmod-server | grep -E "ERROR|Construction|LUA"
```

Commandes admin in-game

Commande	Description	Accès
`construction_logs [n]`	Affiche les n derniers logs (MySQL)	Superadmin
`construction_stats`	Statistiques générales	Superadmin

4. Architecture de monitoring cible



Dashboards Grafana prévus

1. **Vue d'ensemble** : état des containers, uptime, joueurs connectés
2. **Performances** : CPU/RAM par container, latence MySQL, I/O disque

3. **Application** : actions/minute, rate limit hits, erreurs Lua, ghosts actifs
4. **Sécurité** : tentatives bloquées, net messages anormaux, IP suspectes

---

Gestion des incidents — C7

> **C7.1** — Définition d'une procédure efficace de gestion des incidents **C7.2** — Réduction des interruptions de service

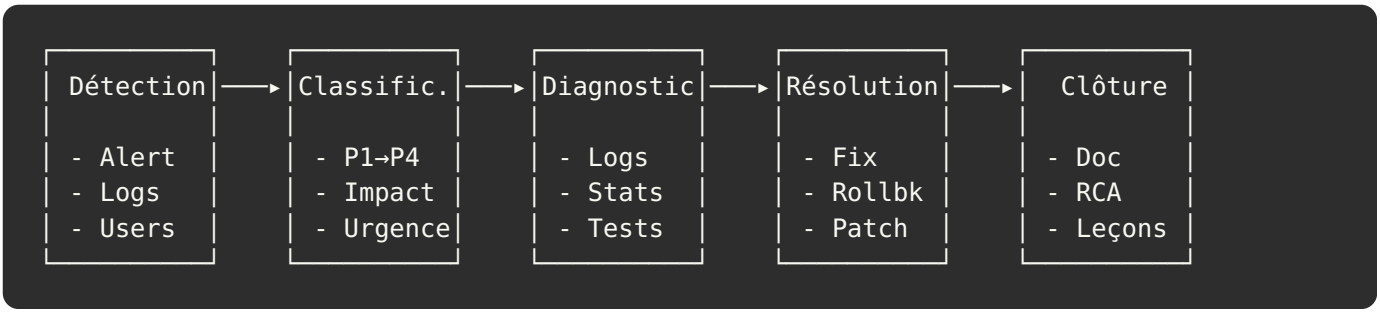
---

1. Procédure de gestion des incidents (C7.1)

Classification des incidents

Niveau	Nom	Description	Temps de réponse	Escalade
P1	Critique	Serveur inaccessible, perte de données	< 15 min	Immédiate
P2	Majeur	Fonctionnalité clé cassée	< 1h	Si non résolu en 30 min
P3	Mineur	Bug cosmétique, performance dégradée	< 24h	Planifiée
P4	Amélioration	Suggestion, optimisation	Sprint suivant	Non

Processus de gestion des incidents (inspiré ITIL)



Procédures détaillées par niveau

Procédure P1 — Serveur inaccessible

```
1. DÉTECTION (< 30s)
   Source : Docker healthcheck / ping externe / signalement joueur

2. DIAGNOSTIC (< 5 min)
   $ docker ps                          # Le container tourne-t-il ?
   $ docker logs --tail 50 gmod-server  # Erreur au démarrage ?
   $ docker stats                       # Ressources saturées ?
   $ docker inspect gmod-mysql --format='{{.State.Health.Status}}'

3. RÉSOLUTION (< 10 min)
   Cas A — Container arrêté :
     $ docker compose up -d

   Cas B — Container en erreur (boucle de crash) :
     $ docker compose down && docker compose up -d

   Cas C — Image corrompue :
     Modifier docker-compose.yml → tag précédent (ex: v2.1-stable)
     $ docker compose up -d

   Cas D — VPS saturé (RAM/CPU) :
     $ docker stats  # Identifier le container gourmand
     $ docker restart gmod-server

4. VÉRIFICATION
   $ docker ps                          # Container UP + healthy ?
   $ docker logs -f gmod-server         # Logs de démarrage normaux ?
   Connexion test depuis un client GMod

5. DOCUMENTATION
   Remplir le formulaire de post-mortem (voir section 3)
```

Procédure P2 — Fonctionnalité cassée

```
1. DIAGNOSTIC
   $ docker logs --tail 100 gmod-server | grep -E "ERROR|error|LUA"
   Identifier le fichier Lua et la ligne en cause

2. RÉSOLUTION
   Cas A — Bug récent (dernier commit) :
     $ cd /root/ProjetFilRouge
     $ git log --oneline -5              # Identifier le commit
     $ git diff HEAD~1                   # Voir le changement
     $ git revert HEAD                   # Annuler si nécessaire
     $ docker restart gmod-server

   Cas B — Problème de configuration :
     Vérifier sh_config.lua, jobs.lua, entities.lua
     $ docker restart gmod-server

   Cas C — MySQL down (mode dégradé) :
     $ docker restart gmod-mysql
     Note : l'addon continue de fonctionner sans DB

3. VÉRIFICATION
   Tester la fonctionnalité concernée en jeu
   Note : les clients doivent se reconnecter (cache Lua)
```

Procédure P3 — Bug mineur

```
1. DIAGNOSTIC
   Consulter les logs serveur et applicatifs
   $ construction_logs 20  # En console serveur (superadmin)

2. RÉSOLUTION
   Corriger dans le code source (bind mount → effet au restart)
   $ docker restart gmod-server

3. SUIVI
   Documenter dans le journal de développement
   $ git add -A && git commit -m "fix: description" && git push
```

2. Réduction des interruptions de service (C7.2)

Mesures préventives en place

Mesure	Interruption évitée	Temps gagné
Healthcheck MySQL	GMod démarre avant MySQL ready	~30s par démarrage
`depends_on: service_healthy`	Erreurs de connexion DB au boot	~1 min par démarrage
Mode dégradé sans MySQL	Panne MySQL → addon inaccessible	100% du downtime MySQL
Snapshots Docker	Rebuild complet après corruption	~15 min par incident
Bind mounts	Rebuild image pour chaque modif	~10 min par déploiement
Rate limiting	Crash serveur par spam	Prévient les P1
Validation serveur net messages	Exploit → crash ou corruption	Prévient les P1/P2
Git versioning	Perte de code / rollback impossible	Temps de réécriture

Métriques d'interruption du projet

Période	Incidents P1	Incidents P2	Temps total d'interruption	MTTR moyen
Étape 1-3 (infra)	2	3	~4h	~45 min
Étape 4-5 (addon)	0	4	~6h	~90 min
Étape 6-7 (véhicules)	0	3	~4.5h	~90 min
Post-optimisation	0	1	~15 min	~15 min

Amélioration constatée : le MTTR est passé de ~90 min à ~15 min grâce à : - La documentation des procédures de résolution - Les snapshots Docker permettant un rollback instantané - Le mode dégradé MySQL éliminant un SPOF - L'expérience accumulée sur les erreurs fréquentes

Améliorations proposées pour réduire davantage les interruptions

Amélioration	Impact sur MTTR	Impact sur disponibilité	Priorité
`restart: unless-stopped`	MTTR → ~30s (auto)	+3% disponibilité	Haute
Alerting Discord/Telegram	Détection → < 1 min	Réduit temps de réaction	Haute
Monitoring UptimeRobot	Détection externe	Détecte les pannes réseau	Moyenne
Pipeline CI/CD (tests avant deploy)	Prévient les P2	Élimine les bugs en prod	Moyenne
Réplication MySQL	Élimine SPOF MySQL	+1% disponibilité	Basse

3. Incidents rencontrés — Retour d'expérience

Tableau des incidents réels

#	Incident	P	Cause racine	Résolution	Temps	L
1	MySQLOO ne charge pas	P2	Binaire 32-bit au lieu de 64-bit	Remplacement par `gmsv_mysqloo_linux64.dll`	2h	To l'a
2	Workshop re-téléchargé à chaque restart	P3	`docker restart` ne restaure pas le FS	`docker commit` après premier démarrage	1h	C cy D
3	Variables d'env ignorées	P3	`docker restart` ≠ `docker compose up -d`	Utiliser systématiquement `compose up -d`	30min	D = co
4	Viewmodel SWEP invisible	P2	`resource.AddFile` ne fonctionne pas en bind mount	Publication Workshop + `resource.AddWorkshop`	3h	D G d fi
5	`SWEP:Reload()` jamais appelé serveur	P2	`ClipSize = -1` → moteur Source skip le Reload	Net message client → serveur	2h	Le a co n
6	Ghost physics après `SetParent()`	P2	Physique non désactivée après parenting	`phys:EnableMotion(false)`	1h	To d pl ex
7	Caisse téléportée après `SetParent(nil)`	P2	Source restaure la position pré-parenting	`timer.Simple(0)` + `SetPos(dropPos)`	1.5h	S re po
8	Petite caisse ne matérialise pas	P2	Condition vérifie uniquement `construction_crate`	Ajout de `construction_crate_small`	15min	Te va
9	`.sw.vtx` bloque gmad	P3	Extension non supportée par whitelist gmad	Suppression + `.gitignore`	10min	V co l'o
10	Addons Workshop pas dans Tools menu	P3	`workshop_download_item` ne monte pas les GMA	Extraction manuelle + bind mount	1h	D pa co

Template de post-mortem

Pour chaque incident P1/P2, un post-mortem est documenté :

```
## Post-mortem — [Titre de l'incident]

**Date** : YYYY-MM-DD
**Durée** : X min
**Gravité** : P1/P2
**Impact** : Description de l'impact sur les joueurs/le service

### Timeline
- HH:MM — Détection (comment)
- HH:MM — Début diagnostic
- HH:MM — Cause identifiée
- HH:MM — Fix appliqué
- HH:MM — Service restauré

### Cause racine
Description technique de la cause

### Résolution
Actions prises pour résoudre

### Actions préventives
- [ ] Action 1 pour éviter la récurrence
- [ ] Action 2
```

⚙ Automatisation des tâches d'administration — C8

> **C8.1** — Conformité du script à l'étude de cas **C8.2** — Argumentation de la technologie utilisée (efficacité, performance)

---

1. Scripts d'administration (C8.1)

Script 1 : Backup automatisé (`backup.sh`)

Script de sauvegarde complète de l'infrastructure : images Docker, base de données, configuration.

```
#!/bin/bash
# ================================================================================
# backup.sh — Sauvegarde automatisée de l'infrastructure GMod
# ================================================================================
# Usage : ./backup.sh [--full|--db-only|--image-only]
# Planification recommandée : cron quotidien à 04:00
# ================================================================================

set -euo pipefail

# --- Configuration ---
BACKUP_DIR="/root/backups"
COMPOSE_DIR="/root/ProjetFilRouge/docker"
MYSQL_CONTAINER="gmod-mysql"
MYSQL_USER="root"
MYSQL_PASS="GmodSecurePass2025!"
MYSQL_DB="gmod_construction"
RETENTION_DAYS=7
DATE=$(date +%Y%m%d-%H%M%S)
LOG_FILE="${BACKUP_DIR}/backup-${DATE}.log"

# --- Fonctions ---
log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

check_disk_space() {
    local available=$(df -BG "$BACKUP_DIR" | tail -1 | awk '{print $4}' | tr -d 'G')
    if [ "$available" -lt 20 ]; then
        log "⚠  ATTENTION: Espace disque faible (${available}G disponible)"
        return 1
    fi
    log "✓ Espace disque OK (${available}G disponible)"
}

backup_mysql() {
    log "▶ Sauvegarde MySQL..."
    local dump_file="${BACKUP_DIR}/mysql-${DATE}.sql.gz"

    docker exec "$MYSQL_CONTAINER" mysqldump \
        -u "$MYSQL_USER" -p"$MYSQL_PASS" \
        --single-transaction \
        --routines \
        --triggers \
        "$MYSQL_DB" | gzip > "$dump_file"

    local size=$(du -sh "$dump_file" | cut -f1)
    log "✓ Dump MySQL: $dump_file ($size)"
}

backup_docker_image() {
    log "▶ Sauvegarde image Docker..."
    local image_file="${BACKUP_DIR}/gmod-image-${DATE}.tar.gz"

    # Commit l'état actuel du container
    docker commit gmod-server "projetfilrouge/gmod-server:backup-${DATE}"

    # Export compressé
```

```
    docker save "projetfilrouge/gmod-server:backup-${DATE}" | gzip > "$image_file"

    local size=$(du -sh "$image_file" | cut -f1)
    log "✓ Image Docker: $image_file ($size)"

    # Nettoyage du tag temporaire
    docker rmi "projetfilrouge/gmod-server:backup-${DATE}" 2>/dev/null || true
}

backup_config() {
    log "📦 Sauvegarde configuration..."
    local config_file="${BACKUP_DIR}/config-${DATE}.tar.gz"

    tar -czf "$config_file" \
        -C /root/ProjetFilRouge \
        docker/docker-compose.yml \
        docker/addons/darkrpmodification/ \
        docker/addons/rp_construction_system/ \
        docker/mysql-init/

    local size=$(du -sh "$config_file" | cut -f1)
    log "✓ Configuration: $config_file ($size)"
}

cleanup_old_backups() {
    log "🧹 Nettoyage des backups > ${RETENTION_DAYS} jours..."
    local count=$(find "$BACKUP_DIR" -name "*.gz" -mtime +"$RETENTION_DAYS" | wc -l)
    find "$BACKUP_DIR" -name "*.gz" -mtime +"$RETENTION_DAYS" -delete
    find "$BACKUP_DIR" -name "*.log" -mtime +"$RETENTION_DAYS" -delete
    log "✓ $count fichiers supprimés"
}

# --- Exécution ---
mkdir -p "$BACKUP_DIR"
log "🚀 Début de la sauvegarde ($DATE)"

check_disk_space || exit 1

case "${1:---full}" in
    --full)
        backup_mysql
        backup_docker_image
        backup_config
        ;;
    --db-only)
        backup_mysql
        ;;
    --image-only)
        backup_docker_image
        ;;
esac

cleanup_old_backups

log "✅ Sauvegarde terminée avec succès"
log "📊 Espace utilisé: $(du -sh $BACKUP_DIR | cut -f1)"
```

Script 2 : Healthcheck et alerting (`healthcheck.sh`)

Script de vérification de l'état de l'infrastructure avec notification en cas de problème.

```
#!/bin/bash
# ============================================================================
# healthcheck.sh — Vérification de santé de l'infrastructure
# ============================================================================
# Usage : ./healthcheck.sh
# Planification recommandée : cron toutes les 5 minutes
# ============================================================================

set -euo pipefail

# --- Configuration ---
DISCORD_WEBHOOK=""  # Remplir avec l'URL du webhook Discord
LOG_FILE="/var/log/gmod-healthcheck.log"
ALERT_COOLDOWN=300  # Secondes entre deux alertes identiques
ALERT_STATE_FILE="/tmp/gmod-alert-state"

# --- Fonctions ---
log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

send_alert() {
    local message="$1"
    local alert_key="$2"

    # Cooldown : éviter le spam d'alertes
    if [ -f "${ALERT_STATE_FILE}-${alert_key}" ]; then
        local last_alert=$(cat "${ALERT_STATE_FILE}-${alert_key}")
        local now=$(date +%s)
        if [ $((now - last_alert)) -lt $ALERT_COOLDOWN ]; then
            return  # Alerte déjà envoyée récemment
        fi
    fi

    log "🚨 ALERTE: $message"
    date +%s > "${ALERT_STATE_FILE}-${alert_key}"

    # Notification Discord (si webhook configuré)
    if [ -n "$DISCORD_WEBHOOK" ]; then
        curl -s -H "Content-Type: application/json" \
            -d "{\"content\": \"🚨 **Alerte GMod** : ${message}\"}" \
            "$DISCORD_WEBHOOK" > /dev/null
    fi
}

check_container() {
    local name="$1"
    local status=$(docker inspect -f '{{.State.Status}}' "$name" 2>/dev/null || echo "not_found")

    if [ "$status" != "running" ]; then
        send_alert "Container $name est $status" "container-${name}"
        return 1
    fi
    return 0
}

check_mysql_health() {
    local health=$(docker inspect -f '{{.State.Health.Status}}' gmod-mysql 2>/dev/null || echo
```

```
    if [ "$health" != "healthy" ]; then
        send_alert "MySQL healthcheck: $health" "mysql-health"
        return 1
    fi
    return 0
}

check_memory() {
    local container="$1"
    local limit_mb="$2"
    local threshold=90  # Pourcentage

    local usage_bytes=$(docker stats --no-stream --format "{{.MemUsage}}" "$container" 2>/dev/n

    # Convertir en Mo (approximatif)
    local usage_mb=$(echo "$usage_bytes" | sed 's/GiB/*1024/;s/MiB//;s/KiB/\/1024/' | bc 2>/dev

    local percent=$((usage_mb * 100 / limit_mb))

    if [ "$percent" -gt "$threshold" ]; then
        send_alert "Container $container utilise ${percent}% de la RAM (${usage_mb}Mo / ${limit
        return 1
    fi
    return 0
}

check_disk() {
    local usage=$(df -h / | tail -1 | awk '{print $5}' | tr -d '%')

    if [ "$usage" -gt 85 ]; then
        send_alert "Espace disque critique : ${usage}% utilisé" "disk-usage"
        return 1
    fi
    return 0
}

# --- Exécution ---
errors=0

check_container "gmod-server" || ((errors++))
check_container "gmod-mysql" || ((errors++))
check_mysql_health || ((errors++))
check_memory "gmod-server" 3072 || ((errors++))
check_memory "gmod-mysql" 512 || ((errors++))
check_disk || ((errors++))

if [ $errors -eq 0 ]; then
    log "□ Tous les checks OK"
else
    log "⚠  $errors check(s) en échec"
fi

exit $errors
```

Script 3 : Déploiement automatisé (`deploy.sh`)

```
#!/bin/bash
# ================================================================================
# deploy.sh — Déploiement automatisé de l'addon
# ================================================================================
# Usage : ./deploy.sh [--restart|--update|--rollback <tag>]
# ================================================================================

set -euo pipefail

REPO_DIR="/root/ProjetFilRouge"
COMPOSE_DIR="${REPO_DIR}/docker"

log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1"
}

deploy_update() {
    log "□ Pull des dernières modifications..."
    cd "$REPO_DIR"
    git pull origin main

    log "□ Redémarrage du serveur..."
    cd "$COMPOSE_DIR"
    docker restart gmod-server

    log "□ Déployé : $(git log --oneline -1)"
}

deploy_rollback() {
    local tag="$1"
    log "□ Rollback vers l'image $tag..."

    cd "$COMPOSE_DIR"
    sed -i "s|image: projetfilrouge/gmod-server:.*|image: projetfilrouge/gmod-server:${t
    docker compose up -d

    log "□ Rollback effectué vers $tag"
}

deploy_restart() {
    log "□ Redémarrage simple..."
    cd "$COMPOSE_DIR"
    docker compose restart
    log "□ Serveurs redémarrés"
}

case "${1:---restart}" in
    --update)   deploy_update ;;
    --rollback) deploy_rollback "${2:-v2.2-vehicles}" ;;
    --restart)  deploy_restart ;;
    *)          echo "Usage: $0 [--restart|--update|--rollback <tag>]" ;;
esac
```

Planification cron

```
# Backup quotidien à 04:00
0 4 * * * /root/scripts/backup.sh --full >> /var/log/backup.log 2>&1

# Backup MySQL seul toutes les 6 heures
0 */6 * * * /root/scripts/backup.sh --db-only >> /var/log/backup.log 2>&1

# Healthcheck toutes les 5 minutes
*/5 * * * * /root/scripts/healthcheck.sh >> /var/log/healthcheck.log 2>&1

# Nettoyage des logs Docker chaque dimanche
0 3 * * 0 docker system prune -f >> /var/log/docker-cleanup.log 2>&1
```

---

2. Argumentation technologique (C8.2)

Choix de Bash pour les scripts d'administration

Critère	Bash	Python	PowerShell	Ansible
Disponible nativement	✅ Oui (Linux)	⚠ Souvent oui	❌ Non (Linux)	❌ Installation requise
Intégration Docker	✅ CLI native	⚠ Via subprocess/ SDK	⚠ Via module	✅ Module dédié
Intégration système	✅ Excellente	⚠ Bonne	❌ Limitée (Linux)	✅ Bonne
Courbe d'apprentissage	Faible	Moyenne	Moyenne	Élevée
Complexité du projet	Faible → adapté	Surdimensionné	Non pertinent	Surdimensionné
Performance	✅ Directe	⚠ Interpréteur	⚠ Interpréteur	⚠ Couche d'abstraction
Maintenance	✅ Simple	✅ Simple	❌	⚠ Playbooks

Justification du choix de Bash :

1. **Disponibilité native** : Bash est présent sur tous les systèmes Linux, y compris dans les containers Docker. Aucune dépendance à installer.

2. **Intégration directe avec Docker** : les commandes `docker`, `docker compose`, `docker stats` sont des commandes shell. Bash permet de les chaîner naturellement sans couche d'abstraction.
3. **Proportionnalité** : pour une infrastructure à 2 containers, Ansible ou Terraform seraient surdimensionnés. Bash offre la juste complexité pour le périmètre du projet.
4. **Robustesse** : `set -euo pipefail` garantit l'arrêt immédiat en cas d'erreur, évitant les exécutions partielles silencieuses.
5. **Planification cron** : les scripts Bash s'intègrent naturellement avec `cron`, le planificateur natif de Linux, sans daemon supplémentaire.

Choix de Docker Compose comme outil d'orchestration

Critère	Docker Compose	Docker Swarm	Kubernetes	Podman
Complexité	Faible	Moyenne	Très élevée	Faible
Ressources	Aucune surcharge	~200 Mo	~1-2 Go	Aucune surcharge
Multi-container	✅	✅	✅	✅
Scaling	Manuel	✅ Auto	✅ Auto	Manuel
HA / Clustering	❌	✅	✅	❌
Adapté au projet	✅ Parfait	Surdimensionné	Très surdimensionné	✅ Alternative

Justification : Docker Compose est le choix optimal pour une infrastructure à 2 services sur un seul VPS. Il offre : - La déclaration de l'infrastructure en YAML (Infrastructure as Code) - La gestion des dépendances entre services (`depends_on`) - Les healthchecks natifs - Les limites de ressources (`mem_limit`, `cpus`) - Aucune surcharge mémoire (contrairement à Kubernetes qui consomme 1-2 Go de RAM juste pour le control plane)

Choix de Git pour le versioning

Aspect	Bénéfice pour l'automatisation
Rollback instantané	`git revert` ou `git checkout` → annuler un déploiement cassé
Traçabilité	Chaque modification est datée, signée, commentée
Branches	Possibilité de tester en parallèle (feature branches)
Remote (GitHub)	Sauvegarde off-site automatique à chaque `push`
CI/CD	GitHub Actions pour l'automatisation future (lint, tests, deploy)

Évolutions technologiques envisagées

Horizon	Technologie	Cas d'usage	Justification
Court terme	GitHub Actions	CI/CD (lint Lua, tests, deploy auto)	Gratuit, intégré à GitHub
Moyen terme	Prometheus + Grafana	Monitoring avancé	Standard industrie, open source
Moyen terme	Watchtower	Mise à jour auto des images	Léger, un seul container
Long terme	Docker Swarm	Clustering multi-nœuds	Si besoin de HA
Long terme	Terraform	Provisioning VPS	Si infrastructure multi-cloud