Mode Opératoire

Gérer les règles d'alerte Prometheus

Code :	MO-PLT-023
Version :	1.1
Date :	26 juin 2026
Auteur :	Cédric LEGRAND
Classification :	USAGE INTERNE — Équipe BTS SIO

Historique des révisions

Version	Date	Modifications
1.0	13/06/2026	Création initiale — documentation des 7 règles en place
1.1	26/06/2026	Harmonisation de la présentation des étapes et des encadrés.

1 Objet

Ce mode opératoire décrit la gestion des **règles d'alerte** de la supervision Prometheus : lire les règles en place, en ajouter ou en modifier, valider la syntaxe avant application, recharger la configuration sans coupure et suivre le cycle de vie d'une alerte. Les règles sont évaluées en continu côté serveur : elles détectent les situations anormales (cible injoignable, disque presque plein, mémoire saturée, processeur durablement chargé) indépendamment de toute consultation des dashboards.

Le document précise également la **limite actuelle assumée** du dispositif : les alertes sont visibles dans les interfaces, mais aucune notification n'est expédiée (pas de relais SMTP joignable depuis le VLAN serveurs).

2 Champ d'application

Public concerné	Administrateurs de l'infrastructure BTS SIO
Application	Prometheus v3.3.1 — CT 200 `docker-srv` (`10.0.112.20`)
Fichier de règles	`/opt/docker/monitoring/alerts.yml`, monté dans le conteneur en `/etc/prometheus/alerts.yml`
Outils	éditeur de texte, `promtool` (embarqué dans le conteneur), `curl`
Durée	10 minutes pour l'ajout d'une règle

3 Concepts

3.1 Anatomie d'une règle

Une règle d'alerte associe une expression PromQL à un seuil de durée et à des métadonnées. Exemple de la règle la plus importante du fichier :

```
- alert: InstanceDown
  expr: up == 0
  for: 5m
  labels:
    severity: critical
  annotations:
    summary: "Cible {{ $labels.instance }} injoignable"
    description: "La cible {{ $labels.instance }} (job
      {{ $labels.job }}) ne répond plus depuis 5 minutes."
```

— `expr` : la condition, évaluée toutes les 15 secondes sur chaque série concernée ;
— `for` : la durée pendant laquelle la condition doit rester vraie avant le déclenchement — c'est l'amortisseur anti-faux-positifs ;
— `labels` : métadonnées de classement, ici la sévérité ;
— `annotations` : textes destinés aux humains ; les gabarits `{{ $labels.xxx }}` sont remplacés par les labels de la série fautive — d'où l'intérêt des libellés lisibles posés à la déclaration des cibles (cf. MO-PLT-022).

3.2 Cycle de vie d'une alerte

État	Signification
`inactive`	La condition est fausse : situation nominale.
`pending`	La condition est vraie, mais depuis moins longtemps que le `for` : l'alerte est en observation.
`firing`	La condition est vraie depuis plus longtemps que le `for` : l'alerte est déclenchée et visible partout (interface Prometheus, tuile « Alertes actives » de la vue d'ensemble Grafana, métrique `ALERTS`).

3.3 Les sept règles en place

Règle	Condition	Seuil	for	Sévérité
InstanceDown	cible injoignable (`up == 0`)	—	5 min	critical
LinuxDiskAlmostFull	espace libre d'un système de fichiers	<15 %	10 min	warning
WindowsDiskAlmostFull	espace libre d'un volume	<15 %	10 min	warning
LinuxMemoryPressure	mémoire utilisée	>90 %	10 min	warning
WindowsMemoryPressure	mémoire disponible	<10 %	10 min	warning
LinuxHighCpu	CPU moyen	>85 %	15 min	warning
WindowsHighCpu	CPU moyen	>85 %	15 min	warning

Les règles sont volontairement **génériques** : elles portent sur des familles de métriques (`node_*`, `windows_*`) et non sur des serveurs nommés. Toute nouvelle cible ajoutée à la collecte est donc couverte immédiatement, sans modification du fichier.

Détection sans notification

En l'état, une alerte `firing` ne déclenche aucun envoi (ni courriel, ni messagerie) : le VLAN serveurs n'a pas d'accès Internet et aucun relais SMTP interne n'est joignable. La détection est donc fiable, mais sa consultation reste à l'initiative de l'équipe (vue d'ensemble Grafana en page d'accueil, page *Alerts* de Prometheus). Le chaînon manquant — un Alertmanager relayé par un hôte disposant d'une sortie — est identifié comme évolution possible et devra faire l'objet d'une décision d'équipe.

4 Prérequis

Prérequis

- **Accès SSH `root` au CT 200 (`10.0.112.20`), via le VPN WireGuard**
- **Notions de base de PromQL pour écrire ou relire une expression**
- **Connaître le nom des métriques visées — vérifiable dans l'interface Prometheus (auto-complétion de la barre de requête)**

5 Procédure

1 Consulter l'état des règles

Dans l'interface Prometheus (`https://prometheus.docker.bts.sio`, cf. MO-PLT-015), ouvrir le menu Alerts : chaque règle apparaît avec son état courant et le détail de sa définition.

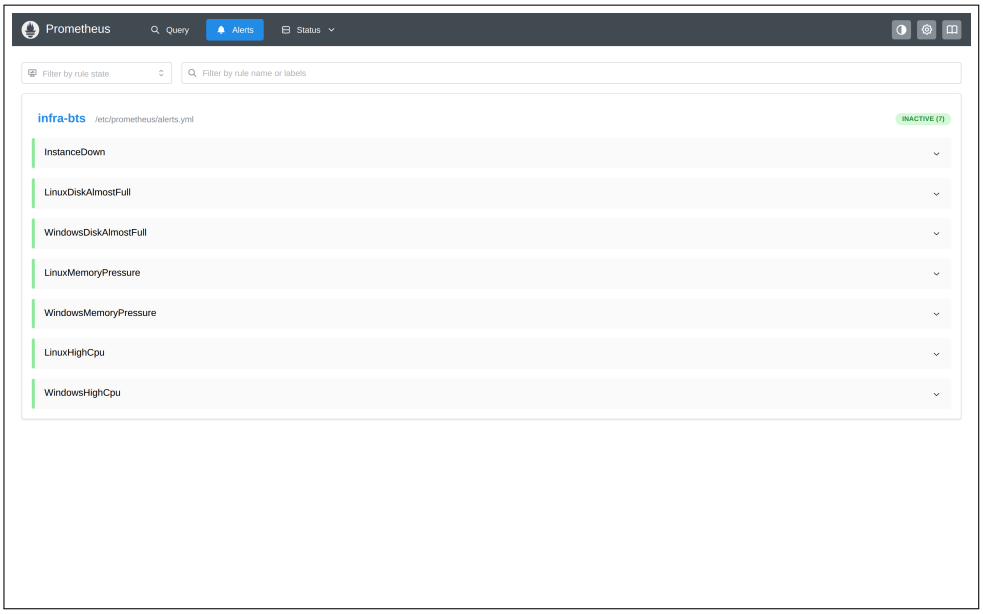

Figure 1 : Page *Alerts* de Prometheus : les sept règles du groupe `infra-bts`, toutes à l'état `inactive` en situation nominale.

2 Ajouter ou modifier une règle

Sur le CT 200, sauvegarder puis éditer le fichier de règles :

```
cd /opt/docker/monitoring
cp alerts.yml alerts.yml.bak
nano alerts.yml
```

Exemple d'ajout : un second seuil disque, critique celui-ci, qui se déclenche plus vite quand la situation devient urgente :

```
- alert: LinuxDiskFullCritical
  expr: (node_filesystem_avail_bytes{fstype!~"tmpfs|overlay|squashfs"}
        / node_filesystem_size_bytes{fstype!~"tmpfs|overlay|squashfs"})
        < 0.05
  for: 5m
  labels:
    severity: critical
  annotations:
    summary: "Disque < 5 % libre sur {{ $labels.instance }}
      ({{ $labels.mountpoint }})"
```

Tester l'expression avant d'en faire une règle

Coller l'expression seule dans la barre de requête de l'interface Prometheus : si elle ne retourne aucune série alors que la situation visée existe, la règle ne se déclenchera jamais (nom de métrique erroné, filtre trop strict). C'est le moyen le plus rapide de mettre au point le `expr`.

3 Valider la syntaxe avec promtool

`promtool` est embarqué dans le conteneur ; il refuse tout fichier mal formé avant que Prometheus ne l'applique :

```
docker exec prometheus promtool check rules \
  /etc/prometheus/alerts.yml
```

Figure 2 : Validation du fichier de règles par `promtool` : la sortie liste le nombre de règles reconnues, ou l'erreur exacte (ligne et cause) en cas de refus.

4 Recharger la configuration à chaud

```
curl -X POST http://127.0.0.1:9090/-/reload
```

Le rechargement est immédiat et sans interruption de la collecte. La nouvelle règle apparaît dans le menu Alerts à l'état `inactive`.

5 Suivre une alerte déclenchée

Trois points d'observation, du plus synthétique au plus détaillé :

- **la tuile « Alertes actives » de la vue d'ensemble Grafana (page d'accueil) passe de 0 (vert) au nombre d'alertes `firing` (rouge) ;**
- **la page Alerts de Prometheus détaille la ou les séries fautives et l'horodatage du déclenchement ;**
- **en PromQL, la métrique synthétique `ALERTS{alertstate="firing"}` permet de filtrer, compter ou croiser les alertes comme n'importe quelle série.**

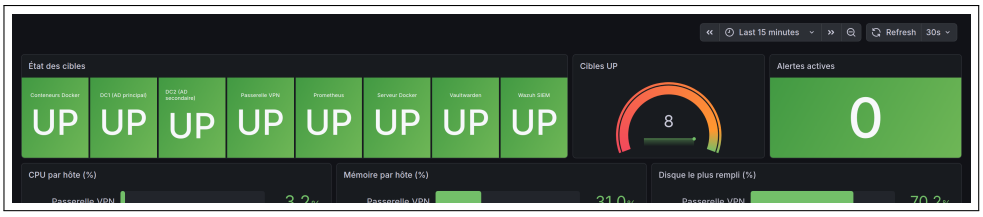

Figure 3 : La tuile « Alertes actives » de la vue d'ensemble Grafana : 0 en vert en situation nominale, elle bascule en rouge avec le nombre d'alertes `firing`.

6 Vérification

Vérification

Après toute modification du fichier de règles :

- ☐ `promtool check rules` **retourne** `SUCCESS` **avec le nombre de règles attendu**
- ☐ **Le rechargement répond sans erreur (HTTP 200, corps vide)**
- ☐ **La règle apparaît dans Alerts avec l'état** `inactive` **(situation nominale)**
- ☐ **L'expression testée seule dans la barre de requête retourne des séries cohérentes**
- ☐ **Le fichier de sauvegarde** `alerts.yml.bak` **existe en cas de retour arrière**

7 Rollback

6 Restaurer la version précédente du fichier

```
cd /opt/docker/monitoring
cp alerts.yml.bak alerts.yml
docker exec prometheus promtool check rules \
  /etc/prometheus/alerts.yml
curl -X POST http://127.0.0.1:9090/-/reload
```

Les règles reviennent à l'état antérieur ; l'historique des alertes passées n'est pas affecté.

8 Dépannage

Problème	Solution
`promtool` refuse le fichier	L'erreur indique la ligne fautive : le plus souvent une indentation YAML incorrecte (les listes de règles sont indentées sous `rules:`) ou une expression PromQL invalide. Corriger puis revalider ; Prometheus continue de fonctionner avec l'ancienne version tant que le rechargement n'a pas eu lieu.
La règle ne passe jamais à `pending`	L'expression ne matche aucune série. La tester seule dans la barre de requête ; vérifier le nom exact de la métrique (auto-complétion) et les filtres de labels.
L'alerte bascule sans arrêt (`firing` ↔ `inactive`)	La métrique oscille autour du seuil. Allonger le `for`, ou écarter le seuil de la valeur de croisière constatée sur le graphique.
Les annotations affichent des champs vides	Le label référencé dans le gabarit n'existe pas sur la série (par exemple `{{ $labels.mountpoint }}` sur une métrique Windows). N'utiliser que des labels portés par les séries de l'expression.
Personne n'a vu une alerte pourtant déclenchée	C'est la limite documentée en section 3 : aucune notification n'est expédiée. Prendre le réflexe de la vue d'ensemble Grafana en début de journée, et inscrire l'évolution Alertmanager à l'ordre du jour de l'équipe.

9 Voir aussi

— **MO-PLT-022** : Ajouter un serveur à la supervision Prometheus — les règles génériques couvrent automatiquement les nouvelles cibles
— **MO-PLT-015** : Se connecter à Prometheus — accès aux pages *Alerts* et *Targets*
— **MO-PLT-024** : Importer et maintenir les dashboards Grafana — la tuile « Alertes actives » de la vue d'ensemble
— **MO-SEC-001** : Exploitation de Wazuh — alertes de sécurité (événements), complémentaires des alertes de métriques

Références

— Prometheus — *Alerting rules* (syntaxe, états, gabarits d'annotations)
— Prometheus — *promtool* (validation de configuration et de règles)
— Google — *Site Reliability Engineering*, chapitre *Practical Alerting* : alerter sur les symptômes durables plutôt que sur les causes ponctuelles
— ANSSI — *Recommandations de sécurité pour la journalisation et la supervision*