

MODULE 7

Gestion de déploiement et monitoring

Stratégies de déploiement et surveillance d'applications



Métriques



Déploiement



Health Checks



Alertes



Dashboard

 Durée estimée : 3 heures

Introduction : Pourquoi surveiller et bien déployer ?

Les enjeux du monitoring

Le monitoring est essentiel pour garantir la fiabilité, la performance et la disponibilité des applications déployées sur OpenShift. Une surveillance efficace permet d'anticiper les problèmes et d'optimiser les ressources.

Bénéfices d'un déploiement maîtrisé

- ✓ Minimisation des interruptions de service
- ✓ Réduction des risques lors des mises à jour
- ✓ Capacité à revenir rapidement en arrière (rollback)
- ✓ Conformité aux SLAs et meilleures performances

Point clé



Un déploiement sans monitoring est comme piloter un avion sans tableau de bord. Le succès d'une application en production dépend autant de la qualité de son code que de sa stratégie de déploiement et de surveillance.

Piliers d'une surveillance efficace



Health Checks

Vérification régulière de l'état des applications et des services pour détecter les problèmes avant qu'ils n'affectent les utilisateurs.



Métriques et KPIs

Collecte et analyse des indicateurs de performance clés pour optimiser le fonctionnement des applications.



Alertes proactives

Détection précoce des anomalies et notification automatique aux équipes concernées.



Stratégies de déploiement

Méthodes contrôlées pour déployer de nouvelles versions avec risque minimal et possibilité de rollback.

Stratégies de déploiement : Blue/Green, Rolling Update, Canary

Blue/Green



Bascule immédiate du trafic

✓ Bascule instantanée, rollback immédiat

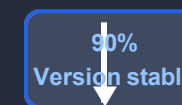
Rolling Update

Pods remplacés progressivement



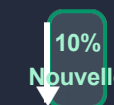
✓ Mise à jour progressive sans interruption

Canary




Exposition progressive

✓ Test sur un sous-ensemble d'utilisateurs



Comparaison des stratégies

Stratégie	Avantages	Inconvénients	Cas d'utilisation
Blue/Green	Bascule instantanée, rollback immédiat, simple	Double les ressources, coût plus élevé	Applications critiques, zéro downtime, rollback rapide
Rolling Update	Pas de ressources supplémentaires, contrôlé	Versions mixtes en production, rollback complexe	Applications à haute disponibilité, ressources limitées
Canary	Risque minimal, détection précoce des problèmes	Configuration complexe, gestion du routage	Fonctionnalités risquées, tests A/B, validation utilisateurs

 **Bon à savoir :** OpenShift implémente nativement ces stratégies via sa configuration de DeploymentConfig. La stratégie par défaut est Rolling Update.

Concepts clés : Health Checks et Readiness/Liveness Probes

Pourquoi utiliser des probes ?

Les probes permettent à Kubernetes de déterminer l'état de santé des conteneurs et de prendre des décisions automatiques pour maintenir la disponibilité des applications. Elles sont essentielles pour l'auto-guérison sur OpenShift.

Les 3 types de probes



Readiness Probe (Pod prêt à recevoir du trafic)

Si échec → Pod retiré des endpoints de service → trafic interrompu



Liveness Probe (Conteneur fonctionne correctement)

Si échec → Pod redémarré selon la politique définie



Startup Probe (Application démarrée)

Si échec → Désactive autres probes → Utile pour démarrage lent

Méthodes de vérification



HTTP GET
Endpoint REST



TCP Socket
Port ouvert



Exec
Commande shell

Exemples de configuration

HTTP Readiness Probe

```
readinessProbe:
  httpGet:
    path: /health/ready
    port: 8080
  initialDelaySeconds: 10
  periodSeconds: 5
```

TCP Liveness Probe

```
livenessProbe:
  tcpSocket:
    port: 8080
  initialDelaySeconds: 15
  periodSeconds: 10
```

Paramètres essentiels

initialDelaySeconds

Délai avant 1ère vérification

periodSeconds

Fréquence des vérifications

timeoutSeconds

Délai avant échec

failureThreshold

Nb d'échecs avant action

Bonnes pratiques



- Endpoints de readiness peu coûteux (< 1s)
- Vérifier dépendances critiques (DB, cache, services)
- Configurer seuils d'échec adaptés à l'application



Readiness → Gestion de trafic



Liveness → Auto-correction



Startup → Protection au démarrage

Métriques et alertes : observer et réagir

Indicateurs clés à surveiller

La surveillance d'une application OpenShift repose sur quatre piliers essentiels de métriques.



CPU

Usage > 80% indique un besoin de scaling.



Mémoire

Les fuites mémoire peuvent causer des OOMKilled.



Réseau

Latence, taux d'erreur et bande passante.



Disponibilité

Uptime, temps de réponse et codes d'erreur.

Configuration des seuils d'alerte

Niveaux d'alertes

Un système d'alertes bien configuré permet d'intervenir au bon moment.



Avertissement

CPU > 75% (5 min), Mémoire > 80%, Latence > 500ms



Critique

CPU > 90% (10 min), Mémoire > 95%, Erreurs HTTP > 5%

Bonnes pratiques d'alertes

- ✓ Configurer une fenêtre de temps
- ✓ Déduplication d'alertes
- ✓ Adapter les seuils par service
- ✓ Actions automatisées



Configuration Prometheus

```
- alert: HighCpuUsage
  expr: cpu_usage_percent > 80 for: 5m
  labels: {severity: warning} annotations:
    summary: High CPU detected
```

Automatisation des déploiements et monitoring

GitOps & CI/CD sur OpenShift

L'automatisation des déploiements avec GitOps permet de synchroniser l'infrastructure et les applications avec un référentiel Git. Principes : déclarativité, versionnement et automatisation.

Pipeline CI/CD avec monitoring intégré

- 1. Intégration continue
Tests et construction des images conteneurs.
Tests **Build**
- 2. Déploiement en test
Déploiement avec vérification des métriques santé.
Deploy **Monitor**
- 3. Promotion en production
Déploiement progressif avec monitoring et rollback.
Release **Alert**

Intégration du monitoring dans l'automatisation



Scaling automatique

Adaptation dynamique des ressources selon la charge. Mise à l'échelle basée sur les métriques.



Rollback automatisé

Retour automatique à la version précédente en cas d'anomalies, basé sur des seuils prédéfinis.



A/B Testing automatisé

Comparaison des performances entre versions et routage intelligent du trafic.

Configuration de Rollback automatique

```
strategy:
  type: Rolling
  rollingParams:
    timeoutSeconds: 120
    maxSurge: "20%"
    post:
      failurePolicy: Abort
```



Outils d'intégration GitOps + Monitoring





ArgoCD, Tekton et OpenShift Pipelines intègrent les métriques dans le processus CI/CD. L'approche GitOps garantit que tout changement est tracé, validé et surveillé en continu.

Surveillance avancée avec Prometheus & Grafana

Prometheus : Collecte de métriques

Système de surveillance intégré à OpenShift qui collecte et stocke des métriques en temps réel dans une base de données temporelle.

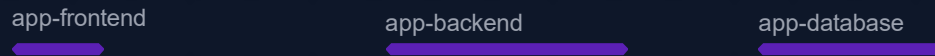
Fonctionnalités clés

-  Base de données temporelle
-  Alertes configurables
-  Langage PromQL
-  Découverte de services

Exemple de requête PromQL :

```
sum(rate(container_cpu_usage_seconds_total{pod=~"app-.*"}[5m])) by (pod)
```





CPU par pod :

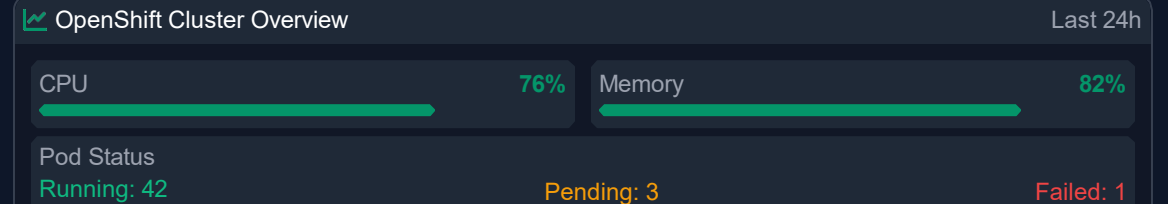


Grafana : Visualisation de données

Plateforme d'analyse qui se connecte à Prometheus pour créer des tableaux de bord interactifs et des alertes visuelles.

Dashboards et alertes

-  Dashboards personnalisés
Tableaux de bord avec visualisations variées
-  Alertes visuelles
Notifications multi-canaux (email, Slack)
-  Multi-sources
Compatible avec différentes sources de données
-  Contrôle d'accès
Gestion basée sur les rôles



-  **Intégration OpenShift**
OpenShift intègre nativement Prometheus et Grafana dans sa Console Web, avec des templates de dashboards disponibles pour un démarrage rapide.

Cas pratique : Mise en place du monitoring dans OpenShift

1. Activation de Prometheus & Grafana

Vérifiez l'Operator Monitoring et installez Grafana :

```
$ oc get pods -n openshift-monitoring
$ oc apply -f grafana-operator.yaml
$ oc new-project monitoring-demo
```

2. Configuration des ServiceMonitors

Créez un ServiceMonitor pour votre application :

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: app-monitor
spec:
  selector:
    matchLabels: {app: myapp}
  endpoints:
    - port: web
      interval: 30s
```

3. Création d'un dashboard Grafana

Créez un dashboard avec ces métriques clés :

- CPU par pod
- Mémoire par pod
- Requêtes HTTP/s
- Temps de réponse

Exemple de requête PromQL :

```
sum(container_memory_usage_bytes{namespace="monitoring-demo"}) by (pod)
```

4. Configuration des alertes mémoire

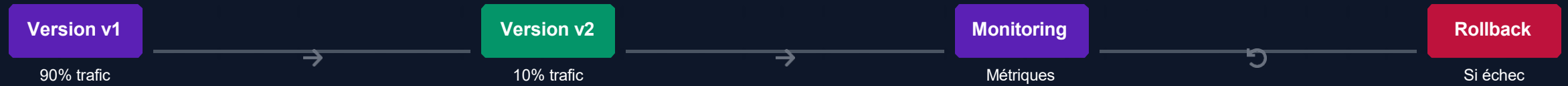
Règle d'alerte pour la surconsommation mémoire :

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: memory-alerts
spec:
  groups:
    - name:
      memory
      rules:
        - alert: HighMemoryUsage
          expr: container_memory_usage_bytes > 800000000
          for: 5m
```

Astuce : Testez vos alertes en générant une charge artificielle

```
$ oc exec -it [pod-name] -- stress --vm 1 --vm-bytes 900M --timeout 300s
```


Cas pratique : Déploiement canary automatisé



Étapes du déploiement canary

- 1 Préparation**
Configurez des métriques (CPU, mémoire, temps de réponse) et des health checks pour votre application.
- 2 Configuration canary**
Déployez v2 et répartissez le trafic 90/10 entre v1 et v2.

```
oc set route-backends myapp myapp-v1=90 myapp-v2=10
```
- 3 Monitoring & rollback**
Configurez Prometheus pour surveiller les métriques et déclencher le rollback automatique si nécessaire.

Indicateurs à surveiller

- Taux d'erreur HTTP
- Mémoire
- Latence (P95)
- CPU

Rollback automatisé

Alerte Prometheus :

```
- alert: CanaryHighErrorRate
  expr: sum(rate(http_requests_total{code=~"5.."}[2m]))
        / sum(rate(http_requests_total[2m])) > 0.05 for: 1m
```

Script de rollback :

```
#!/bin/bash
oc set route-backends myapp myapp-v1=100 myapp-v2=0
curl -X POST $SLACK_WEBHOOK -d '{"text": "ALERTE: Rollback canary"}
```

- Critères de succès**
Surveillance 30 min : taux d'erreur < 1%, latence stable. Augmentez progressivement (20%, 50%, 100%) avec validation à chaque étape.

Synthèse & perspectives

Concepts clés du module

 Blue/Green

 Rolling

 Canary

 Health

 Probes

 Métriques




 Alertes

 GitOps

Ces concepts vous permettent de mettre en œuvre des déploiements fiables et une surveillance efficace de vos applications sur OpenShift.

Questions & Réponses

C'est le moment de clarifier vos points d'interrogation :

-  Quelle stratégie pour des applications critiques ?
-  Comment dimensionner les alertes pour éviter le bruit ?
-  Intégration avec les outils existants ?

Vers l'observabilité avancée

Au-delà du simple monitoring, l'observabilité combine :



Métriques



Logs



Traces

Une vision à 360° pour diagnostiquer rapidement les problèmes complexes.

L'avenir avec AIOps

L'intelligence artificielle au service des opérations :



Détection d'anomalies



Analyse prédictive



Auto-remédiation



Optimisation auto

Ressources pour aller plus loin



Documentation



Red Hat Academy



OpenShift Samples

“La surveillance s'intègre dans une culture DevOps où le feedback et l'amélioration continue sont au cœur des pratiques pour garantir fiabilité et performance.”