PARTIE 4 : Observabilité avec Prometheus et Grafana

Durée : 45 minutes

Objectifs

Dans cette quatrième partie, vous allez déployer une stack complète d'observabilité avec Prometheus et Grafana pour monitorer la plateforme CloudShop.

Compétences évaluées : - Installation de Prometheus Operator - Configuration des ServiceMonitors pour scraping - Création de dashboards Grafana - Définition d'alerting rules - Calcul et monitoring des SLO/SLI

Travail à Réaliser

Tâche 4.1 : Installation de Prometheus et Grafana (4 points)

Installation via Kube-Prometheus-Stack

```
# 1. Ajouter le repo Helm
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update

# 2. Créer le namespace monitoring
kubectl create namespace monitoring

# 3. Installer la stack complète (Prometheus + Grafana + Alertmanager)
helm install prometheus prometheus-community/kube-prometheus-stack \
  --namespace monitoring \
  --set prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=false \
  --set prometheus.prometheusSpec.podMonitorSelectorNilUsesHelmValues=false \
  --set grafana.adminPassword=admin123

# 4. Attendre que tous les pods soient prêts
kubectl wait --for=condition=ready pod --all -n monitoring --timeout=300s

# 5. Vérifier l'installation
```

```
kubectl get pods -n monitoring
kubectl get svc -n monitoring
```

Accès aux Interfaces

```
# Prometheus
kubectl port-forward svc/prometheus-operated 9090:9090 -n monitoring
# URL : http://localhost:9090

# Grafana
kubectl port-forward svc/prometheus-grafana 3000:80 -n monitoring
# URL : http://localhost:3000
# User : admin
# Password : admin123

# Alertmanager
kubectl port-forward svc/prometheus-kube-prometheus-alertmanager 9093:9093 -n
monitoring
# URL : http://localhost:9093
```

Tâche 4.2 : ServiceMonitors pour Scraping (4 points)

Les ServiceMonitors configurent Prometheus pour scraper les métriques des microservices.

ServiceMonitor pour API Gateway

Fichier : monitoring/servicemonitors/api-gateway.yaml

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: api-gateway
  namespace: monitoring
  labels:
    app: api-gateway
    release: prometheus
spec:
  selector:
    matchLabels:
      app: api-gateway
  namespaceSelector:
    matchNames:
```

```
    - cloudshop-prod
  endpoints:
  - port: http
    interval: 30s
    path: /metrics
    scheme: http
```

ServiceMonitor pour Auth Service

Fichier : monitoring/servicemonitors/auth-service.yaml

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: auth-service
  namespace: monitoring
  labels:
    app: auth-service
    release: prometheus
spec:
  selector:
    matchLabels:
      app: auth-service
  namespaceSelector:
    matchNames:
    - cloudshop-prod
  endpoints:
  - port: http
    interval: 30s
    path: /metrics
    scheme: http
```

ServiceMonitor pour Products API

Fichier : monitoring/servicemonitors/products-api.yaml

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: products-api
  namespace: monitoring
  labels:
    app: products-api
    release: prometheus
spec:
```

```
  selector:
    matchLabels:
      app: products-api
  namespaceSelector:
    matchNames:
    - cloudshop-prod
  endpoints:
  - port: http
    interval: 30s
    path: /metrics
    scheme: http
```

ServiceMonitor pour Orders API

Fichier : monitoring/servicemonitors/orders-api.yaml

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: orders-api
  namespace: monitoring
  labels:
    app: orders-api
    release: prometheus
spec:
  selector:
    matchLabels:
      app: orders-api
  namespaceSelector:
    matchNames:
    - cloudshop-prod
  endpoints:
  - port: http
    interval: 30s
    path: /metrics
    scheme: http
```

Déploiement des ServiceMonitors

```
# Appliquer tous les ServiceMonitors
kubectl apply -f monitoring/servicemonitors/

# Vérifier
kubectl get servicemonitor -n monitoring
```

```
# Vérifier que Prometheus les détecte
# Aller dans Prometheus UI > Status > Targets
# Les targets cloudshop-prod/* doivent apparaître
```

Tâche 4.3 : Alerting Rules (3 points)

Définissez des règles d'alerte pour détecter les anomalies.

Fichier : monitoring/alerts/prometheus-rules.yaml

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: cloudshop-alerts
  namespace: monitoring
  labels:
    release: prometheus
spec:
  groups:
  - name: cloudshop.rules
    interval: 30s
    rules:

    # Alerte : Taux d'erreur élevé (>5%)
    - alert: HighErrorRate
      expr: |
        (
          sum(rate(http_requests_total{status=~"5..",namespace="cloudshop-prod"}[5m]))
          /
          sum(rate(http_requests_total{namespace="cloudshop-prod"}[5m]))
        ) > 0.05
      for: 5m
      labels:
        severity: warning
        component: backend
      annotations:
        summary: "Taux d'erreur élevé détecté"
        description: "Le service {{ $labels.service }} a un taux d'erreur de {{ $value | humanizePercentage }} sur les 5 dernières minutes."

    # Alerte : Latence P95 élevée (>1s)
    - alert: HighLatencyP95
```

```
    expr: |
      histogram_quantile(0.95,
        sum(rate(http_request_duration_seconds_bucket{namespace="cloudshop-prod"}
[5m])) by (le, service)
      ) > 1
    for: 10m
    labels:
      severity: warning
      component: performance
    annotations:
      summary: "Latence P95 élevée"
      description: "Le service {{ $labels.service }} a une latence P95 de {{ $value }}s."

  # Alerte : Pod CrashLooping
  - alert: PodCrashLooping
    expr: |
      rate(kube_pod_container_status_restarts_total{namespace="cloudshop-prod"}[15m])
> 0
    for: 5m
    labels:
      severity: critical
      component: infrastructure
    annotations:
      summary: "Pod en CrashLoop détecté"
      description: "Le pod {{ $labels.pod }} redémarre fréquemment."

  # Alerte : Pod non disponible
  - alert: PodNotReady
    expr: |
      kube_pod_status_phase{namespace="cloudshop-prod",phase!="Running"} > 0
    for: 5m
    labels:
      severity: warning
      component: infrastructure
    annotations:
      summary: "Pod non disponible"
      description: "Le pod {{ $labels.pod }} n'est pas en état Running."

  # Alerte : Utilisation mémoire élevée (>80%)
  - alert: HighMemoryUsage
    expr: |
      (
        container_memory_working_set_bytes{namespace="cloudshop-prod"}
```

```
        /
        container_spec_memory_limit_bytes{namespace="cloudshop-prod"}
      ) > 0.8
    for: 10m
    labels:
      severity: warning
      component: resources
    annotations:
      summary: "Utilisation mémoire élevée"
      description: "Le conteneur {{ $labels.container }} utilise {{ $value |
humanizePercentage }} de sa limite mémoire."

  # Alerte : SLO Breach (Availability < 99.9%)
  - alert: SLOBreach
    expr: |
      (
        sum(rate(http_requests_total{status!~"5..",namespace="cloudshop-prod"}[30m]))
        /
        sum(rate(http_requests_total{namespace="cloudshop-prod"}[30m]))
      ) < 0.999
    for: 5m
    labels:
      severity: critical
      component: slo
    annotations:
      summary: "SLO de disponibilité non respecté"
      description: "La disponibilité est de {{ $value | humanizePercentage }}, en dessous
du SLO de 99.9%."

# Appliquer les règles
kubectl apply -f monitoring/alerts/prometheus-rules.yaml

# Vérifier dans Prometheus UI
# Status > Rules
# Alerts
```

Tâche 4.4 : Dashboards Grafana (4 points)

Dashboard 1 : CloudShop Overview

Fichier : monitoring/dashboards/cloudshop-overview.json

```
{
  "dashboard": {
    "title": "CloudShop - Overview",
    "panels": [
      {
        "title": "Request Rate (RPS)",
        "targets": [
          {
            "expr": "sum(rate(http_requests_total{namespace=\"cloudshop-prod\"}[5m]))",
            "legendFormat": "Total RPS"
          }
        ],
        "type": "graph"
      },
      {
        "title": "Error Rate (%)",
        "targets": [
          {
            "expr": "sum(rate(http_requests_total{status=~\"5..\",namespace=\"cloudshop-
prod\"}[5m])) / sum(rate(http_requests_total{namespace=\"cloudshop-prod\"}[5m])) *
100",
            "legendFormat": "Error Rate"
          }
        ],
        "type": "graph"
      },
      {
        "title": "P95 Latency",
        "targets": [
          {
            "expr": "histogram_quantile(0.95,
sum(rate(http_request_duration_seconds_bucket{namespace=\"cloudshop-prod\"}[5m]))
by (le))",
            "legendFormat": "P95 Latency"
          }
        ],
        "type": "graph"
      },
      {
        "title": "Pods Status",
        "targets": [
          {
            "expr": "count(kube_pod_status_phase{namespace=\"cloudshop-
```

```
prod\",phase=\"Running\"})",
          "legendFormat": "Running"
        },
        {
          "expr": "count(kube_pod_status_phase{namespace=\"cloudshop-prod\",phase!
=\"Running\"})",
          "legendFormat": "Not Running"
        }
      ],
      "type": "stat"
    }
  ]
 }
}
```

Dashboard 2 : SLO Monitoring

Queries PromQL pour le dashboard SLO :

```
# Availability SLO (99.9%)
(
  sum(rate(http_requests_total{status!~"5..",namespace="cloudshop-prod"}[30d]))
  /
  sum(rate(http_requests_total{namespace="cloudshop-prod"}[30d]))
) * 100

# Error Budget Remaining (30 days)
(
  1 - (
    sum(rate(http_requests_total{status=~"5..",namespace="cloudshop-prod"}[30d]))
    /
    sum(rate(http_requests_total{namespace="cloudshop-prod"}[30d]))
  )
  - 0.999
) / (1 - 0.999) * 100

# Burn Rate (1h)
(
  sum(rate(http_requests_total{status=~"5..",namespace="cloudshop-prod"}[1h]))
  /
  sum(rate(http_requests_total{namespace="cloudshop-prod"}[1h]))
) / (1 - 0.999)
```

```
# Latency SLI (P95 < 500ms)
histogram_quantile(0.95,
  sum(rate(http_request_duration_seconds_bucket{namespace="cloudshop-prod"}[5m]))
by (le)
) * 1000
```

Import des Dashboards dans Grafana

Méthode 1 : Via UI 1. Grafana UI > Dashboards > Import 2. Copier/coller le JSON 3. Sélectionner Prometheus datasource 4. Save

Méthode 2 : Via ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-dashboards
  namespace: monitoring
  labels:
    grafana_dashboard: "1"
data:
  cloudshop-overview.json: |
    # Contenu du JSON dashboard
```

```
kubectl apply -f monitoring/dashboards/configmap-dashboards.yaml
```

Définition des SLO/SLI

SLO (Service Level Objective)

Availability SLO : 99.9% uptime sur 30 jours glissants

Calcul : - Downtime autorisé = (1 - 0.999) * 30 jours * 24h * 60min = 43.2 minutes/mois

Error Budget : - Budget initial = 43.2 minutes - Budget consommé = downtime réel - Budget restant = 43.2 - downtime réel

SLI (Service Level Indicator)

Availability SLI : Ratio de requêtes réussies

```
sum(rate(http_requests_total{status!~"5.."}[5m]))
/
sum(rate(http_requests_total[5m]))
```

Latency SLI : P95 latency < 500ms

```
histogram_quantile(0.95,
  sum(rate(http_request_duration_seconds_bucket[5m])) by (le)
) < 0.5
```

Validation

Checklist

```
# 1. Stack Prometheus/Grafana installée
kubectl get pods -n monitoring
# prometheus-prometheus-kube-prometheus-prometheus-0   Running
# prometheus-grafana-xxx                               Running
# prometheus-kube-prometheus-operator-xxx              Running
# alertmanager-prometheus-kube-prometheus-alertmanager-0  Running

# 2. ServiceMonitors créés
kubectl get servicemonitor -n monitoring
# Doit lister : api-gateway, auth-service, products-api, orders-api

# 3. Prometheus scrape les targets
# Ouvrir http://localhost:9090 > Status > Targets
# Les targets cloudshop-prod/* doivent être UP

# 4. Alerting rules configurées
# Prometheus UI > Alerts
# Les alertes cloudshop.rules doivent apparaître

# 5. Grafana accessible avec dashboards
# Ouvrir http://localhost:3000
# Dashboards > CloudShop Overview et SLO Monitoring doivent exister

# 6. Métriques affichées
# Dans Grafana, les panels doivent afficher des données (pas vides)
```

Queries de Test dans Prometheus

```
# 1. Total requests per second
sum(rate(http_requests_total{namespace="cloudshop-prod"}[5m]))

# 2. Error rate
sum(rate(http_requests_total{status=~"5..",namespace="cloudshop-prod"}[5m])) /
sum(rate(http_requests_total{namespace="cloudshop-prod"}[5m]))

# 3. P95 latency
histogram_quantile(0.95,
sum(rate(http_request_duration_seconds_bucket{namespace="cloudshop-prod"}[5m]))
by (le))

# 4. Pods running
count(kube_pod_status_phase{namespace="cloudshop-prod",phase="Running"})

# 5. CPU usage by pod
sum(rate(container_cpu_usage_seconds_total{namespace="cloudshop-prod"}[5m])) by
(pod)

# 6. Memory usage by pod
sum(container_memory_working_set_bytes{namespace="cloudshop-prod"}) by (pod) /
1024 / 1024
```

ServiceMonitor ne scrape pas

```
# Vérifier les labels
kubectl get servicemonitor <name> -n monitoring -o yaml

# Vérifier le selector dans Prometheus
kubectl get prometheus -n monitoring -o yaml | grep serviceMonitorSelector

# Solution : Ajouter le label release: prometheus
metadata:
  labels:
    release: prometheus
```

Aucune métrique dans Grafana

```
# 1. Vérifier Prometheus datasource dans Grafana
# Configuration > Data sources > Prometheus
```

```
# URL : http://prometheus-operated:9090

# 2. Tester une query simple
# Explore > Prometheus > up

# 3. Vérifier que les targets sont UP
# Prometheus UI > Status > Targets
```

Alerting rules ne s'affichent pas

```
# Vérifier les labels de la PrometheusRule
kubectl get prometheusrule -n monitoring -o yaml

# Doit avoir le label : release: prometheus

# Vérifier que Prometheus les charge
# Prometheus UI > Status > Rules
```

Ressources

- Prometheus Operator
- PromQL Basics
- Grafana Dashboards
- SLO Best Practices

Une fois cette partie terminée, passez à PARTIE5_SECURITE_SRE.md