Formation: Grafana
Formateur: Mokhtar Sellami
Mail: mokhtar.sellami@data2-ai.com

Atelier 5 Observabilité Pods & Application avec Grafana et Prometheus OpenShift

Durée estimée : 90–120 minutes
Niveau : Intermédiaire / Avancé

🧩 Objectifs pédagogiques

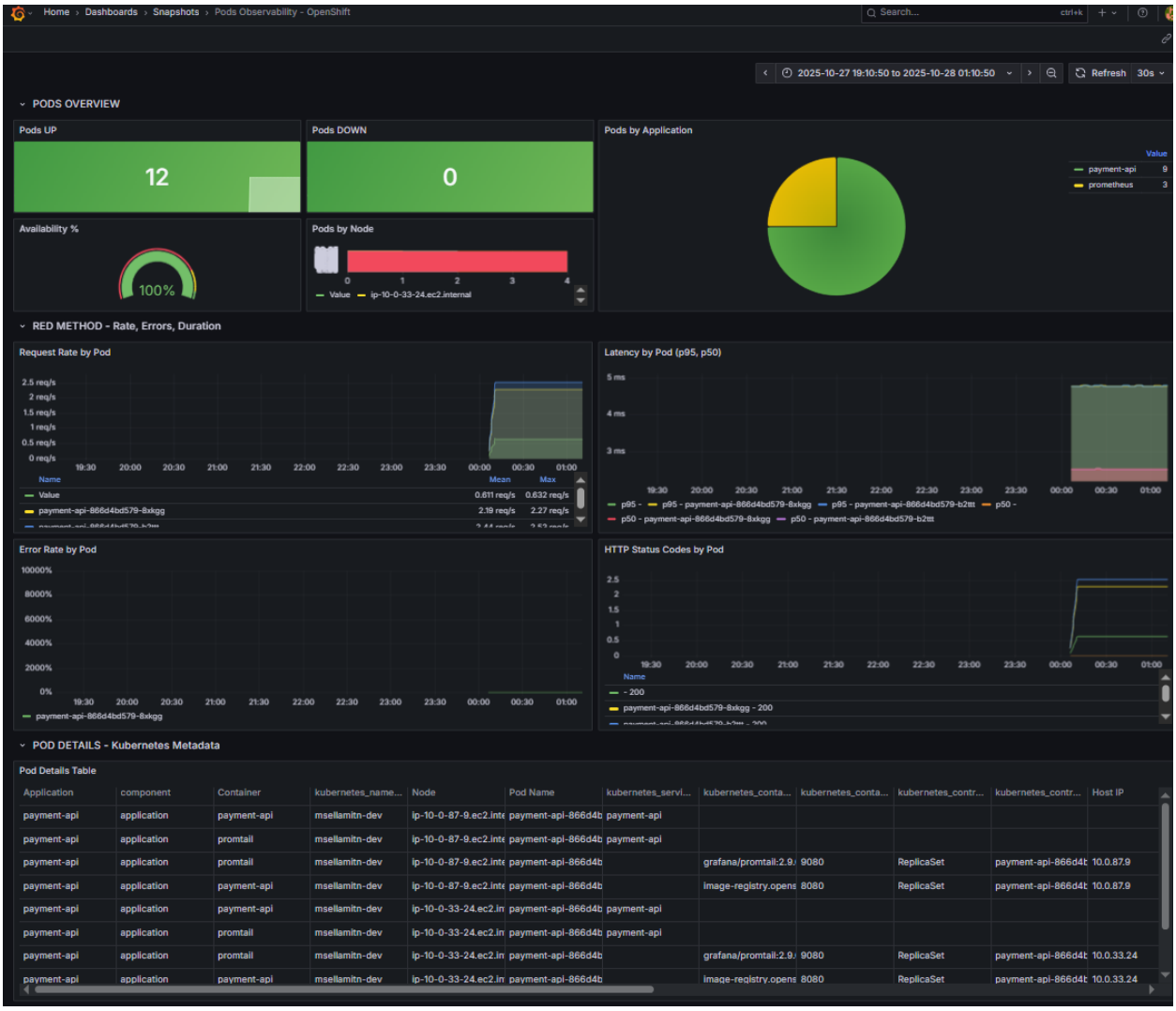
À la fin de cet atelier, le participant sera capable de :

1. Configurer une **datasource Prometheus OpenShift** et variables dynamiques.
2. Comprendre et adapter les **requêtes PromQL** pour monitoring des pods et applications.
3. Créer et personnaliser un **dashboard Grafana complet** : Gauges, TimeSeries, PieChart, BarChart, Table.
4. Mettre en place des **alertes basiques et thresholds** pour le monitoring des pods et des applications.
5. Sauvegarder et exporter le dashboard pour reproduction.

🔷 0.2. Pré-requis

- Grafana installé et accessible avec un compte admin.
- Prometheus exposant les métriques Kubernetes et applications (simulateur ou OpenShift).
- Accès au cluster OpenShift si on utilise la datasource live.
- Simulateur pour générer des métriques (optionnel pour atelier offline).

Formation: Grafana
Formateur: Mokhtar Sellami
Mail: mokhtar.sellami@data2-ai.com



0.2. Ajouter la datasource OpenShift Prometheus

1. Grafana → **Configuration → Data Sources → Add data source → Prometheus**
2. Paramètres :

Champ	Valeur
Name	OpenShift Prometheus
URL	https://prometheus-msellamitn-dev.apps.rm3.7wse.p1.openshiftapps.com/
Access	Server
Auth	(Bearer token si nécessaire)

3. **Save & Test** → Vérifier "Data source is working"

0.3. Créer une variable dynamique pour datasource

1. Dashboard → ⚙ Settings → Variables → Add variable
2. Type : **Datasource → Prometheus**
3. Nom : datasource
4. Valeur par défaut : OpenShift Prometheus
5. Modifier toutes les requêtes PromQL pour utiliser [$__rate_interval] et la variable datasource.

Tip : Cela permet de basculer facilement entre Prometheus local ou OpenShift.

🔷 Partie 1 : Création du Dashboard

1. Grafana → + → **Dashboard → Add new panel**
2. Paramètres généraux :

Champ	Valeur
Title	Pods Observability – OpenShift

Champ	Valeur
Tags	kubernetes, pods, observability, openshift
Time zone	Browser Time
Refresh	30s

🔷 Partie 2 : Panels Kubernetes Pods Overview

2.1. Pods UP / DOWN (Stat Panels)

- UP : count(up{kubernetes_namespace="msellamitn-dev"} == 1)
- DOWN : count(up{kubernetes_namespace="msellamitn-dev"} == 0) or vector(0)
- **Thresholds** : Vert = OK, Rouge = Down
- **Tips** : utiliser le **colorMode = background** pour une lecture rapide.

2.2. Pods by Application (Pie Chart)

- Query : count(up{kubernetes_namespace="msellamitn-dev"}) by (app)
- Legend : {{app}}
- **Tips** : permet de visualiser rapidement la répartition des pods par application.

2.3. Availability % (Gauge)

- Query : count(up{kubernetes_namespace="msellamitn-dev"} == 1) / count(up{kubernetes_namespace="msellamitn-dev"})
- Thresholds : Rouge < 80%, Jaune 80–95%, Vert > 95%
- **Astuce** : utiliser la **unit = percentunit** pour lisibilité.

2.4. Pods by Node (Bar Chart)

- Query : count(up{kubernetes_namespace="msellamitn-dev"}) by (kubernetes_node_name)
- Orientation : Horizontal
- **Tips** : permet de détecter les noeuds avec trop ou peu de pods.

🔷 Partie 3 : RED Method – Request, Errors, Duration

3.1. Request Rate by Pod (Timeseries)

sum(rate(http_server_request_duration_seconds_count{app="payment-api"}[5m])) by (kubernetes_pod_name)

- **Legend** : {{kubernetes_pod_name}}
- Unit : req/s
- **Tip** : utiliser **lineInterpolation = smooth** pour une lecture claire des tendances.

3.2. Latency by Pod (p95, p50)

histogram_quantile(0.95, sum(rate(http_server_request_duration_seconds_bucket{app="payment-api"}[5m])) by (kubernetes_pod_name, le))

histogram_quantile(0.50, sum(rate(http_server_request_duration_seconds_bucket{app="payment-api"}[5m])) by (kubernetes_pod_name, le))

- Panel Timeseries
- Unit : seconds

- **Astuce** : ajouter les deux courbes p50 et p95 dans le même panel pour comparer latence moyenne vs haute latence.

3.3. Error Rate by Pod (Timeseries, %)

sum(rate(http_server_request_duration_seconds_count{app="payment-api", http_response_status_code=~"5.."}[5m])) by (kubernetes_pod_name)

/

sum(rate(http_server_request_duration_seconds_count{app="payment-api"}[5m])) by (kubernetes_pod_name)

- Unit : percentunit
- Thresholds : Rouge > 5%, Jaune 2–5%, Vert < 2%

3.4. HTTP Status Codes by Pod (Timeseries)

sum(rate(http_server_request_duration_seconds_count{app="payment-api"}[5m])) by (kubernetes_pod_name, http_response_status_code)

- Legend : {{kubernetes_pod_name}} - {{http_response_status_code}}
- Affichage : Table / Timeseries selon préférence

🔷 Partie 4 : Pod Details Table

- Query : up{kubernetes_namespace="msellamitn-dev", kubernetes_pod_name!=""}
- Transformations : renommer colonnes (Pod Name, Node, Application, Host IP, Pod IP, Container)
- **Astuce** : permet un inventaire rapide de tous les pods et statuts.

🔷 Partie 5 : Tips & Best Practices

- Toujours **utiliser des variables** pour datasources et namespaces → facile à répliquer.
- **Color coding et thresholds** améliorent la lecture rapide des panels critiques.
- **Preload panels** activé → meilleure performance dashboard.
- **Sauvegarder et exporter le JSON** pour partager l'atelier ou répliquer dans d'autres clusters.
- **Auto-refresh** à 30s ou 1m selon criticité.
- **Annotations** : ajouter des événements Kubernetes importants (deployments, scaling) pour corréler avec pics métriques.