🚀 Atelier 5 : Observabilité Pods & Application avec Grafana et Prometheus OpenShift

ODDO BHF TUNIS

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

Durée estimée : 90–120 minutes
Niveau : Intermédiaire / Avancé
Grafana Version : 12.0+ (GUI mise à jour)
Format : Self-service Dashboard Design (Étapes détaillées + Configuration complète des propriétés)

🏵️ Objectifs pédagogiques

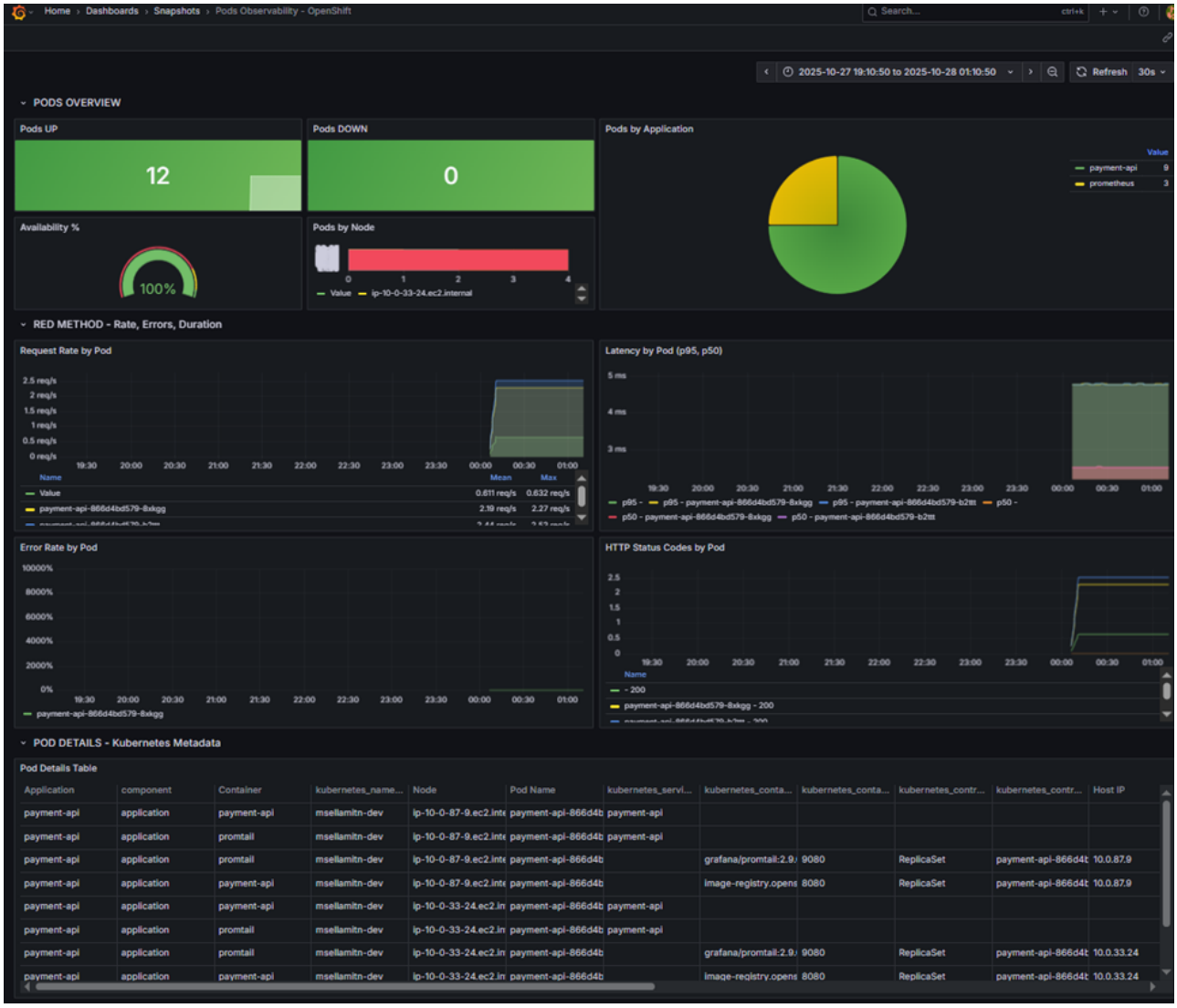
À la fin de cet atelier, vous serez capable de :

1. ✅ Configurer une datasource Prometheus OpenShift (Grafana 12.0)
2. ✅ Maîtriser les requêtes PromQL pour le monitoring des pods/applications
3. ✅ Créer un dashboard complet avec 10 panels (Stat, Gauge, PieChart, BarChart, TimeSeries, Table)
4. ✅ Configurer les seuils (thresholds) et options visuelles en détail
5. ✅ Exporter et reproduire le dashboard sur d'autres clusters
6. ✅ **Concevoir vos propres dashboards en self-service**

📋 Pré-requis

- ✅ Grafana 12.0+ installé et accessible (compte admin)
- ✅ Prometheus exposant les métriques Kubernetes
- ✅ Accès au cluster OpenShift
- ✅ Namespace cible : `msellamitn-dev`
- ✅ Application instrumentée : `payment-api`

- ☑ Browser moderne (Chrome, Firefox, Safari, Edge)



PHASE 0 : Configuration Datasource

◈ Étape 0.1 : Accéder aux Data Sources

1. Ouvrir Grafana → Menu latéral → **Connections** ✎
2. Sélectionner **"Data Sources"**
3. Cliquer **"+ Create new data source"**
4. Chercher et sélectionner **"Prometheus"**

◈ Étape 0.2 : Remplir la configuration Prometheus

Champ	Valeur	Notes
Name	`OpenShift Prometheus`	Nom unique
Description	`Prometheus pour OpenShift`	Optionnel
Prometheus server URL	`https://prometheus-msellamitn-dev.apps.rm3.7wse.p1.openshiftapps.com/`	Avec https:// et / final

Champ	Valeur	Notes
HTTP method	`GET`	Défaut
Timeout	`30s`	Défaut
Skip TLS verification	❌ Non	Garder sécurisé

Si authentification Bearer Token :

- Scroller → **"Authentication"** → Cocher **"Bearer token"**
- Coller le token : `oc whoami -t`

Sauvegarder : Cliquer **"Save & test"** → Vérifier ✅ **"Data source is working"**

PHASE 1 : Créer le Dashboard

🔷 Étape 1.1 : Créer un nouveau Dashboard

1. Menu latéral → **"Dashboards"** 📊
2. Cliquer **"+ Create"** → **"New dashboard"**
3. Cliquer **"Edit"** (barre violette en haut)

🔷 Étape 1.2 : Configurer les paramètres du Dashboard

1. Cliquer l'icône ⚙️ **(Settings)** en haut à droite
2. Panneau Settings s'ouvre à droite

Remplissez :

Champ	Valeur
Title	`Pods Observability - OpenShift`
Description	`Dashboard complet pour monitoring des pods et applications Kubernetes`
Tags	`kubernetes`, `pods`, `observability`, `openshift`
Timezone	`Browser Time`
Auto-refresh	`30s`
Editable	✅ Enabled

3. Cliquer **"Save"** pour appliquer

PHASE 2 : Variables Dynamiques

◈ Étape 2.1 : Créer une variable Datasource

1. **Settings** → Onglet **"Variables"**
2. Cliquer **"+ Create variable"**

Configuration :

Champ	Valeur
Variable name	`datasource`
Variable type	`Datasource`
Datasource type	`Prometheus`
Multi-value	☑ Enabled
Include all option	☑ Enabled
Default value	`OpenShift Prometheus`

3. Cliquer **"Save variable"**

PHASE 3 : PODS OVERVIEW (5 Panels)

◈ Étape 3.1 : Préparer le Dashboard

1. Fermer Settings panel
2. Cliquer **"Edit"** (barre violette)
3. Cliquer **"+ Add"** → **"Row"**
4. Nommer la Row : `PODS OVERVIEW`

◈ Étape 3.2 : PANEL 1 - PODS UP (Stat)

ÉTAPE 1 : Créer et configurer la Query

1. Cliquer **"+ Add"** → **"Visualization"**
2. **À gauche (Query Editor) :**

Champ	Valeur
Datasource	`OpenShift Prometheus`
Query	`count(up{kubernetes_namespace="msellamitn-dev"} == 1)`

3. Appuyer **Ctrl+Entrée** pour exécuter

ÉTAPE 2 : Configurer le Panel

À droite, Section "Panel options" :

Champ	Valeur
Title	`Pods UP`
Description	`Nombre de pods UP dans le namespace`

Cliquer sur Visualization type (en haut à droite) → Sélectionner **"Stat"**

ÉTAPE 3 : Configurer Field Config (Defaults)

Cliquer l'onglet **"Field config"** à droite

Sous-section "Defaults" :

Champ	Valeur	Étapes
Unit	(vide)	Garder par défaut
Color mode	`Background`	Cliquer dropdown
Graph mode	`Area`	Cliquer dropdown
Text mode	`Auto`	Cliquer dropdown

Scroller → "Thresholds" :

- Mode : Vérifier **"Absolute"** ☑
- **Cliquer "Add threshold"** pour ajouter les steps :

```
Step 1: Value = 0  → Color = Red    (pas de pods = rouge)
Step 2: Value = 1  → Color = Green  (pods OK = vert)
```

Ajouter un Step :

- Cliquer **"+ Add threshold"**
- Entrer la valeur (ex: 0)
- Cliquer le carré de couleur → Sélectionner **Red**
- Répéter pour Step 2

ÉTAPE 4 : Configurer les Options du Panel

Cliquer l'onglet **"Options"** à droite

Champ	Valeur
Reduce options → Calc	`Last not null`
Orientation	`Auto`
Text justify	`Auto`

ÉTAPE 5 : Appliquer et redimensionner

1. Cliquer **"Apply"** ou **"Save & close"** (bouton bleu)
2. Drag le panel pour redimensionner → **Largeur : 6, Hauteur : 4**

✅ Résultat attendu :

Un bloc Stat avec :

- Nombre affiché gros (ex: "3")
- Titre "Pods UP"
- Couleur de fond : **Vert si ≥ 1, Rouge si = 0**

🔷 Étape 3.3 : PANEL 2 - PODS DOWN (Stat)

ÉTAPE 1 : Créer le Panel

1. Cliquer **"+ Add"** → **"Visualization"**
2. Positionner **à droite du panel Pods UP**

ÉTAPE 2 : Query

```
count(up{kubernetes_namespace="msellamitn-dev"} == 0) or vector(0)
```

ÉTAPE 3 : Panel Config

Champ	Valeur
Title	`Pods DOWN`
Visualization type	`Stat`

ÉTAPE 4 : Field Config

Defaults → Thresholds :

- Mode : `Absolute`
- Step 1: Value = 0 → Color = **Green** ✅ (bon = 0 down)
- Step 2: Value = 1 → Color = **Red** ❌ (mauvais = pods down)

⚠️ **Important :** Logique inversée (0 = bon, >0 = mauvais)

ÉTAPE 5 : Redimensionner

Largeur : 6, Hauteur : 4 (à côté Pods UP)

✅ Résultat attendu :

Deux panels Stat côte à côte avec couleurs inversées

◈ Étape 3.4 : PANEL 3 - PODS BY APPLICATION (Pie Chart)

ÉTAPE 1 : Créer le Panel

1. Cliquer **"+ Add"** → **"Visualization"**
2. Position : **Nouvelle ligne, largeur complète**

ÉTAPE 2 : Query

```
count(up{kubernetes_namespace="msellamitn-dev"}) by (app)
```

ÉTAPE 3 : Panel Config

Champ	Valeur
Title	`Pods by Application`
Visualization type	`Pie chart`
Largeur	12 colonnes
Hauteur	8 lignes

ÉTAPE 4 : Field Config (Defaults)

Champ	Valeur
Color mode	`Palette Classic`
Custom → Pie type	`pie` (dropdown)

ÉTAPE 5 : Configurer les Options

Onglet **"Options"** à droite :

Champ	Valeur
Pie type	`Pie`
Sort	`Desc` (décroissant)
Tooltip → Mode	`Single`
Tooltip → Sort	`None`

Legend :

- **Display mode** : `Table` (dropdown)
- **Placement** : `Right` (dropdown)
- **Show legend** : ☑ Enabled
- **Legend values** : Cocher `value`

ÉTAPE 6 : Appliquer

Cliquer **"Apply"**

✅ Résultat attendu :

Graphique camembert avec :

- Sections colorées par app (payment-api, user-api, etc.)
- Légende en table à droite
- Pourcentages visibles

◈ Étape 3.5 : PANEL 4 - AVAILABILITY % (Gauge)

ÉTAPE 1 : Créer le Panel

1. Cliquer **"+ Add"** → **"Visualization"**
2. Position : **Nouvelle ligne, 6 colonnes**

ÉTAPE 2 : Query

```
count(up{kubernetes_namespace="msellamitn-dev"} == 1) /
count(up{kubernetes_namespace="msellamitn-dev"})
```

ÉTAPE 3 : Panel Config

Champ	Valeur
Title	`Availability %`
Visualization type	`Gauge`

ÉTAPE 4 : Field Config (Defaults)

Champ	Valeur	Action
Unit	`percentunit`	Dropdown → Chercher "percent"
Min	`0`	Taper 0
Max	`1`	Taper 1
Thresholds → Mode	`Percentage`	Dropdown (changer depuis Absolute)

Configurer les Steps (Thresholds) :

```
Step 1: Value = 0%   → Color = Red
Step 2: Value = 80%  → Color = Yellow
Step 3: Value = 95%  → Color = Green
```

Ajouter les steps :

- Cliquer **"+ Add threshold"**
- Entrer valeur (0, 80, 95)
- Sélectionner couleur

ÉTAPE 5 : Configurer les Options

Onglet **"Options"** :

Champ	Valeur
Orientation	`Auto`
Show threshold labels	☑ Enabled
Show threshold markers	☑ Enabled
Min viz height	`75`
Min viz width	`75`
Sizing	`Auto`

ÉTAPE 6 : Redimensionner

Largeur : 6, Hauteur : 4

☑ Résultat attendu :

Gauge style "speedometer" :

- Aiguille qui monte vers 95%
- Couleur rouge → jaune → verte selon valeur
- Marqueurs de seuil visibles

◈ Étape 3.6 : PANEL 5 - PODS BY NODE (Bar Chart)

ÉTAPE 1 : Créer le Panel

1. Cliquer **"+ Add"** → **"Visualization"**
2. Position : **À côté du Gauge, 6 colonnes**

ÉTAPE 2 : Query

```
count(up{kubernetes_namespace="msellamitn-dev"}) by (kubernetes_node_name)
```

Legend : `{{kubernetes_node_name}}`

ÉTAPE 3 : Panel Config

Champ	Valeur
Title	`Pods by Node`
Visualization type	`Bar chart`

ÉTAPE 4 : Field Config (Defaults)

Champ	Valeur
Color mode	`Palette Classic`

ÉTAPE 5 : Configurer les Options

Onglet **"Options"** :

Champ	Valeur
Orientation	`Horizontal` (dropdown)
Bar width	`0.97`
Group width	`0.7`
Full highlight	✕ Unchecked
Show value	`Auto`
Stacking	`None`
Tooltip → Mode	`Single`
Tooltip → Sort	`None`

Legend :

- **Display mode** | `List` (dropdown)
- **Placement** | `Bottom` (dropdown)
- **Show legend** | ☑ Enabled

ÉTAPE 6 : Redimensionner

Largeur : 6, Hauteur : 4

☑ Résultat attendu :

Graphique bar horizontal :

- Barres colorées par node
- Noms des nodes sur l'axe Y
- Nombre de pods sur l'axe X

PHASE 4 : RED METHOD (4 Panels)

◈ Étape 4.1 : Ajouter une Row "RED METHOD"

1. Cliquer **"+ Add"** → **"Row"**
2. Nommer : `RED METHOD - Rate, Errors, Duration`

◈ Étape 4.2 : PANEL 6 - REQUEST RATE BY POD

ÉTAPE 1 : Créer le Panel

1. Cliquer **"+ Add"** → **"Visualization"**

ÉTAPE 2 : Query

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

Legend : `{{kubernetes_pod_name}}`

ÉTAPE 3 : Panel Config

Champ	Valeur
Title	`Request Rate by Pod`
Visualization type	`Time series`
Largeur	12
Hauteur	8

ÉTAPE 4 : Field Config (Defaults)

Champ	Valeur
Unit	`reqps` (requests per second)
Color mode	`Palette Classic`

Custom settings (scroller) :

Champ	Valeur
Draw style	`Line`
Line interpolation	`Smooth`
Line width	`1`

Champ	Valeur
Fill opacity	`20`
Point size	`5`
Show points	`Auto`

ÉTAPE 5 : Configurer les Options

Onglet **"Options"** :

Champ	Valeur
Tooltip → Mode	`Single`
Tooltip → Sort	`None`

Legend :

- **Display mode** | `Table`
- **Placement** | `Bottom`
- **Show legend** | ✅ Enabled
- **Legend calcs** | Cocher `mean`, `max`

✅ Résultat attendu :

Graphique TimeSeries :

- Ligne(s) lisse(s) montrant les requêtes/seconde
- Tableau legend avec Mean et Max en bas
- Couleurs différentes par pod

🔷 Étape 4.3 : PANEL 7 - LATENCY (p95, p50)

ÉTAPE 1 : Créer le Panel

1. Cliquer **"+ Add"** → **"Visualization"**
2. Position : **À côté du Request Rate, 12 colonnes**

ÉTAPE 2 : Ajouter les 2 Queries

Query A (p95) :

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

Legend A : `p95 - {{kubernetes_pod_name}}`

À gauche, cliquer "+ Query" pour ajouter Query B

Query B (p50) :

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

Legend B : `p50 - {{kubernetes_pod_name}}`

ÉTAPE 3 : Panel Config

Champ	Valeur
Title	`Latency by Pod (p95, p50)`
Visualization type	`Time series`

ÉTAPE 4 : Field Config

Champ	Valeur
Unit	`s` (secondes)
Color mode	`Palette Classic`

Custom settings :

Champ	Valeur
Draw style	`Line`
Line interpolation	`Smooth`
Fill opacity	`20`

ÉTAPE 5 : Options

Legend :

- **Display mode** | `List`
- **Placement** | `Bottom`
- **Show legend** | ✅ Enabled

✅ Résultat attendu :

Deux courbes (p50 et p95) :

- p50 (médiane) en bas, plus stable
- p95 (haute latence) en haut, plus volatile
- Légende listant les deux quantiles

◈ Étape 4.4 : PANEL 8 - ERROR RATE BY POD

ÉTAPE 1 : Créer le Panel

1. Cliquer **"+ Add"** → **"Visualization"**

ÉTAPE 2 : Query

```
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)
```

ÉTAPE 3 : Panel Config

Champ	Valeur
Title	`Error Rate by Pod`
Visualization type	`Time series`
Largeur	12
Hauteur	8

ÉTAPE 4 : Field Config

Champ	Valeur
Unit	`percentunit`
Color mode	`Palette Classic`

Thresholds :

- Mode : `Absolute`

Configurer les Steps :

```
Step 1: Value = 0     → Color = Green
Step 2: Value = 0.02  → Color = Yellow  (2%)
Step 3: Value = 0.05  → Color = Red     (5%)
```

Custom settings :

Champ	Valeur

Champ	Valeur
Draw style	`Line`
Line interpolation	`Smooth`
Fill opacity	`0`

ÉTAPE 5 : Options

Champ	Valeur
Legend display mode	`List`
Legend placement	`Bottom`

✅ Résultat attendu :

Graphique montrant % d'erreurs 5xx :

- Zone verte si < 2%
- Zone jaune si 2-5%
- Zone rouge si > 5%

🔷 Étape 4.5 : PANEL 9 - HTTP STATUS CODES

ÉTAPE 1 : Créer le Panel

1. Cliquer **"+ Add"** → **"Visualization"**
2. Position : **À côté du Error Rate**

ÉTAPE 2 : Query

```
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}}`

ÉTAPE 3 : Panel Config

Champ	Valeur
Title	`HTTP Status Codes by Pod`
Visualization type	`Time series`

ÉTAPE 4 : Field Config

Champ	Valeur

Champ	Valeur
Color mode	`Palette Classic`

Custom settings :

Champ	Valeur
Draw style	`Line`
Line interpolation	`Linear` (pas smooth)
Fill opacity	`0`

ÉTAPE 5 : Options

Legend :

- **Display mode** | `Table`
- **Placement** | `Bottom`
- **Show legend** | ✅ Enabled

✅ Résultat attendu :

Plusieurs courbes (200, 201, 400, 404, 500, etc.) :

- Chaque code HTTP une couleur différente
- Tableau legend en bas listant tous les codes

PHASE 5 : POD DETAILS TABLE

🔷 Étape 5.1 : Ajouter une Row "POD DETAILS"

1. Cliquer **"+ Add"** → **"Row"**
2. Nommer : `POD DETAILS - Kubernetes Metadata`

🔷 Étape 5.2 : PANEL 10 - POD DETAILS TABLE

ÉTAPE 1 : Créer le Panel

1. Cliquer **"+ Add"** → **"Visualization"**

ÉTAPE 2 : Query

```
up{kubernetes_namespace="msellamitn-dev", kubernetes_pod_name!=""}
```

ÉTAPE 3 : Query Settings (À gauche)

Champ	Valeur
Format	`Table` (dropdown, pas Time series)
Instant	☑ Cocher (snapshot du moment)

ÉTAPE 4 : Panel Config

Champ	Valeur
Title	`Pod Details Table`
Visualization type	`Table`
Largeur	24 (complète)
Hauteur	10

ÉTAPE 5 : Ajouter les Transformations

Onglet **"Transformations"** (à droite, après Options)

1. Cliquer **"+ Add transformation"**
2. Sélectionner **"Organize fields"**

Exclure les colonnes :

- Cliquer le **"X"** à côté de : `Time`, `__name__`, `instance`, `job`

Renommer les colonnes (cliquer chaque colonne pour la renommer) :

```
Value                     → Status
app                       → Application
kubernetes_container_name → Container
kubernetes_host_ip        → Host IP
kubernetes_node_name      → Node
kubernetes_pod_ip         → Pod IP
kubernetes_pod_name       → Pod Name
```

3. Cliquer **"Apply"**

ÉTAPE 6 : Configurer les Options de Table

Onglet **"Options"** :

Champ	Valeur
Show header	☑ Enabled
Column width	`Auto`
Text alignment	`Left`

✅ Résultat attendu :

Tableau avec colonnes :

- Pod Name
- Node
- Application
- Container
- Pod IP
- Host IP
- Status (0 ou 1)

PHASE 6 : FINALISATION & EXPORT

🔹 Étape 6.1 : Sauvegarder le Dashboard

1. Cliquer **"Save"** (barre violette en haut)
2. Vérifier les infos :

Champ	Valeur
Dashboard name	`Pods Observability - OpenShift`
Folder	Créer ou sélectionner `Kubernetes`
Description	`Monitoring complet des pods et applications`

3. Cliquer **"Save"** (bouton bleu)

🔹 Étape 6.2 : Configurer l'URL Slug

1. **Settings** (⚙️) → **"General"**
2. Scroller → **"URL options"**

Champ	Valeur
URL Slug	`pods-observability-openshift`

3. Cliquer **"Save"**

🔹 Étape 6.3 : Activer Preload (Performance)

1. **Settings** → Onglet **"General"**
2. Scroller → **"Preload dashboard"** : ✅ Cocher
3. Cliquer **"Save"**

🔹 Étape 6.4 : Exporter le JSON

1. Menu ⋮ (haut droit) → **"More"** → **"Export"**
2. Cliquer **"Save JSON to file"**

Fichier téléchargé : `pods-observability-openshift.json`

Utilité :

- 📄 Backup du dashboard
- 🔄 Répliquer sur autre Grafana
- 📦 Versionner dans Git

🎯 GUIDE SELF-SERVICE : Concevoir Vos Propres Dashboards

Méthode : Les 5 étapes de design

📊 ÉTAPE 1 : Définir vos KPIs

Avant de créer des panels, clarifiez vos objectifs :

Pour une application Kubernetes :

```
🎯 KPI Critical (Rouge si breach) :
   ├─ Pods UP/DOWN count
   ├─ Availability % (target: > 95%)
   ├─ Error rate (target: < 2%)
   └─ CPU/Memory usage

📊 KPI Warning (Jaune si alerte) :
   ├─ Response latency p95 (target: < 500ms)
   ├─ Request rate (baseline)
   └─ Pod restart count

ℹ️ KPI Info (Informationnel) :
   ├─ Nombre de nodes
   ├─ Répartition des pods
   └─ Versions des services
```

📋 ÉTAPE 2 : Choisir les bonnes Visualizations

Tableau de correspondance KPI → Visualization :

KPI	Visualization	Raison
Pods UP/DOWN	**Stat**	Affiche nombre simple + couleur

KPI	Visualization	Raison
Availability %	**Gauge**	Speedometer intuitif
Répartition pods	**Pie Chart**	Montre proportions
Pods par node	**Bar Chart**	Comparaison horizontale
Trend rate/latency	**Time Series**	Évolution dans le temps
Inventaire pods	**Table**	Détail et drill-down
Comparaison multi-pods	**Heatmap**	Patterns spatiaux

🔧 ÉTAPE 3 : Construire vos Queries PromQL

Modèle générique :

```
# Synthétique (pour Stat/Gauge)
count(métrique{label="valeur"} == condition)

# Par dimension (pour charts)
count(métrique{label="valeur"}) by (dimension1, dimension2)

# Rate (pour trends)
rate(métrique_count{app="xxx"}[5m])

# Ratio (pour %)
count(bons_résultats) / count(total)

# Histogram quantile (latence)
histogram_quantile(0.95, rate(métrique_bucket[5m]))
```

Exemples pratiques :

Cas d'usage	Query
CPU par pod	`sum(rate(container_cpu_usage_seconds_total[5m])) by (pod_name)`
Memory usage	`sum(container_memory_usage_bytes) by (pod_name)`
Disk I/O	`rate(container_fs_io_time_seconds_total[5m])`
Network traffic	`sum(rate(container_network_transmit_bytes_total[5m])) by (pod_name)`
Pod restarts	`increase(kube_pod_container_status_restarts_total[1h])`
Deployment replicas	`kube_deployment_status_replicas{deployment="xxx"}`

🎨 ÉTAPE 4 : Configurer les Propriétés Visuelles

A. Choix des couleurs et thresholds

Règle d'or : Adapter les seuils à votre SLO (Service Level Objective)

Métrique	Green	Yellow	Red
Availability	> 99%	95-99%	< 95%
Error Rate	< 1%	1-5%	> 5%
Latency p95	< 200ms	200-500ms	> 500ms
CPU usage	< 50%	50-80%	> 80%
Memory usage	< 60%	60-85%	> 85%

Pour chaque Panel, configurer :

1. **Thresholds (dans Field Config)**
   - Mode : `Absolute` ou `Percentage` selon métrique
   - Ajouter des steps aux limites
2. **Unit (dans Field Config)**
   - `percentunit` pour les %
   - `s` pour les secondes
   - `bytes` pour la mémoire
   - `reqps` pour les requêtes/sec
3. **Color Mode (dans Options)**
   - `Background` pour Stat (impact visuel fort)
   - `Palette Classic` pour TimeSeries (multi-couleurs)

B. Configurations par Panel Type

Pour Stat Panel :

```
Field Config:
├─ Unit: (adapter)
├─ Thresholds:
│  ├─ Mode: Absolute
│  ├─ Step 0: value=0, color=red
│  └─ Step 1: value=1, color=green
└─ Color mode: Background

Options:
├─ Reduce calc: Last not null
├─ Graph mode: Area
└─ Text mode: Auto
```

Pour Gauge Panel :

```
Field Config:
├─ Unit: percentunit
├─ Min: 0
├─ Max: 1
├─ Thresholds:
│  ├─ Mode: Percentage
│  ├─ Step 0: value=0%, color=red
│  ├─ Step 1: value=80%, color=yellow
│  └─ Step 2: value=95%, color=green

Options:
├─ Orientation: Auto
├─ Show threshold labels: ✅
└─ Show threshold markers: ✅
```

Pour Time Series Panel :

```
Field Config:
├─ Unit: (adapter - reqps, s, ms, %)
├─ Color mode: Palette Classic
├─ Custom:
│  ├─ Draw style: Line
│  ├─ Line interpolation: Smooth (pour trend) ou Linear (pour données discrètes)
│  ├─ Fill opacity: 20
│  └─ Point size: 5

Options:
├─ Legend display: Table ou List
├─ Legend placement: Bottom
└─ Legend calcs: Mean, Max (ajouter si pertinent)
```

Pour Bar Chart Panel :

```
Field Config:
├─ Color mode: Palette Classic
├─ Custom:
│  ├─ Fill opacity: 80

Options:
├─ Orientation: Horizontal (pour longues listes) ou Vertical
├─ Bar width: 0.97
├─ Group width: 0.7
├─ Show value: Auto
```

```
├─ Stacking: None (ou Stack pour totaux)
└─ Legend placement: Bottom
```

Pour Pie Chart Panel :

```
Field Config:
├─ Color mode: Palette Classic

Options:
├─ Pie type: Pie (ou donut)
├─ Sort: Desc (décroissant)
├─ Legend display: Table
├─ Legend placement: Right
└─ Show legend: ✅
```

Pour Table Panel :

```
Options:
├─ Show header: ✅
├─ Column width: Auto
├─ Text alignment: Left

Transformations:
├─ Organize fields (exclure/renommer colonnes)
├─ Sort by (trier par colonnes clés)
└─ Filter by value (si besoin)
```

🔗 ÉTAPE 5 : Intégrer et Documenter

Checklist avant production

- ✅ **Toutes les queries testées** → Affichent des données
- ✅ **Thresholds validés** → Alignés avec SLO
- ✅ **Titre clair** → Indique le KPI
- ✅ **Légendes explicites** → Contexte visible
- ✅ **Unités correctes** → Pas de confusion (ms vs s)
- ✅ **Refresh adapté** → 30s pour alertes, 5m pour rétrospective
- ✅ **Description ajoutée** → Pour futurs utilisateurs

Documentation à ajouter

Dans les Settings du dashboard :

Champ	Contenu

Champ	Contenu
Title	Nom clair (ex: "API Payment - Production")
Description	Objectif + SLOs (ex: "Monitoring de payment-api. Target: 99.5% availability")
Tags	`production`, `payment-api`, `kubernetes`, `sre`

Créer un fichier README.md :

```
# Dashboard: API Payment Monitoring

## Objectif
Surveiller la santé et performance de l'API Payment en production.

## Métriques clés
- **Availability**: Target > 99.5%
- **Latency p95**: Target < 200ms
- **Error Rate**: Target < 0.1%

## Alerts associées
- Pod down > 2min → Slack #payments-alerts
- Error rate > 5% → PagerDuty

## Runbook
- [Incident response payment-api](link)
- [Troubleshooting guide](link)
```

📚 EXEMPLES DE DASHBOARDS PRÊTS À L'EMPLOI

Template 1 : Monitoring Applicatif Simple

```
Layout:
├─ Row 1: Overview
│  ├─ Stat: Requests/sec
│  ├─ Stat: Error count
│  ├─ Gauge: Availability %
│  └─ Time Series: Request Rate

├─ Row 2: Performance
│  ├─ Time Series: Latency p50/p95/p99
│  ├─ Time Series: Error Rate
│  └─ Bar Chart: Top errors

└─ Row 3: Resources
   ├─ Time Series: CPU %
   ├─ Time Series: Memory %
   └─ Table: Pod Details
```

Template 2 : Monitoring Infrastructure K8s

```
Layout:
├─ Row 1: Cluster Health
│  ├─ Stat: Nodes up
│  ├─ Stat: Pods running
│  ├─ Gauge: Cluster health %
│  └─ Pie Chart: Pods par namespace

├─ Row 2: Node Status
│  ├─ Bar Chart: Pods par node
│  ├─ Table: Node resources
│  └─ Heatmap: CPU by node

└─ Row 3: Pod Activity
   ├─ Time Series: Pod restarts
   ├─ Time Series: OOMKills
   └─ Table: Recently restarted pods
```

🚀 GUIDE AVANCÉ : Optimiser les Performances

1. Optimiser les Queries PromQL

❌ Mauvais (lent) :

```
count(container_memory_usage_bytes)  # Récupère ALL pods
```

✅ Bon (rapide) :

```
count(container_memory_usage_bytes{namespace="prod"})  # Filtre early
```

Règles :

- Toujours ajouter des labels de filtre (`namespace`, `app`)
- Utiliser des regex minimales : `"api|payment"` vs `".*"`
- Augmenter `[5m]` pour réduire les points de données

2. Configurer les Intervalles de Rafraîchissement

Type de dashboard	Refresh
Alerting/On-call	10-15s

Type de dashboard	Refresh
SOC/Monitoring temps réel	30s
Manager/Executive	5-10 min
Rétrospective/Analyse	Manual

Dans Settings :

- `Auto-refresh` : Dropdown → `30s` (défaut)
- Utilisateur peut override avec `?refresh=5s` en URL

3. Utiliser les Variables pour Dynamisme

Variable `namespace` :

```
count(up{kubernetes_namespace="$namespace"})
```

Variable `app` :

```
rate(http_requests_total{app="$app"}[5m])
```

Avantage : Un dashboard = multi-tenants/multi-apps

4. Limiter les Données Affichées

Pour Tables :

- Ajouter un filtre : `Top 20 rows` dans Options
- Utiliser paging si > 100 rows

Pour Time Series :

- Limiter à 10-15 séries max (lisibilité)
- Utiliser `topk()` ou `bottomk()` dans PromQL

5. Annotations pour Contexte

Ajouter des événements Kubernetes :

```
Deployments, Pod restarts, Node maintenance
```

Configuration (Grafana 12.0) :

1. Settings → **Annotations**
2. Ajouter datasource Prometheus ou Loki

3. Query : `ALERTS` ou `kube_*_created`

TROUBLESHOOTING COURANT

Panel affiche "No data"

Causes possibles :

Symptôme	Solution
Query retourne vide	Vérifier le namespace/label dans query
Prometheus timeout	Augmenter timeout dans datasource (Settings → HTTP)
Label n'existe pas	Chercher label exact : `label_names()` dans Prometheus UI
Time range trop court	Augmenter à `Last 6 hours`

Colors ne changent pas avec les seuils

Vérifier :

- ☑ Thresholds mode = `Absolute` ou `Percentage`
- ☑ Values entre min/max du gauge
- ☑ Color mode = `Background` pour Stat
- ☑ Reduce calc = `Last not null` (pas Average)

Legend affiche trop d'entries

Solution :

- Limiter series dans PromQL : `topk(5, métrique)`
- Ou masquer dans Options : **Legend** → **Hide** pour séries secondaires

Dashboard très lent à charger

Actions :

- Réduire nombre de panels (par Row)
- Réduire time range par défaut (24h vs 6h)
- Activer Preload : Settings → `Preload dashboard`
- Optimiser queries (voir section "Optimiser les Queries")

CHECKLIST FINAL

Avant de partager le dashboard

- ☑ Dashboard sauvegardé avec titre descriptif

- ✅ Description complète (objectif, SLOs, contacts)
- ✅ Tous les panels avec titres et descriptions
- ✅ Tags appropriés pour discovery
- ✅ Refresh interval adapté au use case
- ✅ JSON exporté et versionné (Git)
- ✅ Permissions définies (Viewer, Editor, Admin)
- ✅ Runbook/documentation lié en description

Partager avec l'équipe

Option 1 : Lien direct

```
https://grafana.example.com/d/pods-observability-openshift?refresh=30s&orgId=1
```

Option 2 : JSON + Import

```
# Exporter le JSON
# Partager le fichier
# Autre cluster : Dashboards → Import → Upload JSON
```

Option 3 : Helm Chart / Provisioning

```
# ConfigMap dans Kubernetes
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-dashboard-payment
data:
  dashboard.json: |
    # Contenu du JSON exporté
```

📚 RESSOURCES OFFICIELLES

Ressource	Lien
Grafana Docs	https://grafana.com/docs/grafana/latest/
PromQL Tutorial	https://prometheus.io/docs/prometheus/latest/querying/basics/
Kubernetes Metrics	https://kubernetes.io/docs/tasks/debug-application-cluster/resource-metrics-pipeline/
RED Method	https://www.weave.works/blog/the-red-method-key-metrics-for-microservices-architecture/

Ressource	Lien
Four Golden Signals	https://sre.google/sre-book/monitoring-distributed-systems/

✨ RÉSUMÉ DU DASHBOARD CRÉÉ

10 Panels au total

#	Panel	Type	Métrique	Seuils
1	Pods UP	Stat	count(up==1)	0=Red, 1=Green
2	Pods DOWN	Stat	count(up==0)	0=Green, 1=Red
3	Pods by App	Pie	by(app)	-
4	Availability %	Gauge	up ratio	0%=Red, 80%=Yellow, 95%=Green
5	Pods by Node	Bar	by(node)	-
6	Request Rate	TimeSeries	rate[5m]	Mean, Max legend
7	Latency p95/p50	TimeSeries	histogram_quantile	-
8	Error Rate	TimeSeries	5xx ratio	0%=Green, 2%=Yellow, 5%=Red
9	HTTP Status	TimeSeries	by(status_code)	Table legend
10	Pod Details	Table	up metric	7 columns

3 Sections (Rows)

1. **PODS OVERVIEW** : État global des pods
2. **RED METHOD** : Métriques applicatives (Rate, Error, Duration)
3. **POD DETAILS** : Inventaire détaillé

Variables

- `datasource` : Switcher entre Prometheus instances

☆ **Atelier complété avec toutes les configurations détaillées pour Grafana 12.0+**

✅ **Production-ready** | 📦 **Exportable** | 🔄 **Réplicable** | 🎯 **Self-service ready**