Cours Complet : Prometheus et Grafana

De la théorie à la pratique

Introduction : Pourquoi surveiller nos applications ?

Imaginez que vous êtes propriétaire d'un restaurant. Vous voulez savoir combien de clients vous avez chaque jour, quels plats sont les plus populaires, combien de temps les clients attendent, et si votre cuisine fonctionne bien. Dans le monde informatique, c'est exactement ce que fait le **monitoring** : surveiller la santé et les performances de nos applications.

Prometheus et Grafana forment un duo puissant dans cet univers. Prometheus joue le rôle du collecteur de données (comme un compteur qui note tout ce qui se passe), tandis que Grafana est le tableau de bord élégant qui transforme ces données en graphiques compréhensibles.

Chapitre 1 : Comprendre Prometheus

Qu'est-ce que Prometheus ?

Prometheus est un système de surveillance et d'alerte open-source développé par SoundCloud en 2012. Pensez à lui comme à un bibliothécaire méticuleux qui collecte, stocke et organise toutes les informations importantes sur vos applications.

Les concepts fondamentaux

1. Les métriques : le langage de Prometheus

Une métrique, c'est simplement une mesure chiffrée d'un aspect de votre système. Par exemple :

- Le nombre de requêtes HTTP reçues
- Le temps de réponse d'une base de données
- L'utilisation de la mémoire d'un serveur

2. Les quatre types de métriques

Counter (Compteur) : Imaginez un compteur de voiture qui ne peut qu'augmenter. C'est parfait pour mesurer des événements qui s'accumulent, comme le nombre total de requêtes.

```python
```

```
# Exemple : compter les visites sur une page web
visits_total = Counter('website_visits_total', 'Nombre total de visites')
visits_total.inc()  # Incrémente de 1 à chaque visite
```

Gauge (Jauge) : Comme le thermomètre de votre maison, cette valeur peut monter et descendre. Idéal pour mesurer des niveaux actuels comme l'utilisation de la mémoire.

```
python

# Exemple : surveiller l'utilisation de la mémoire
memory_usage = Gauge('memory_usage_bytes', 'Utilisation mémoire en bytes')
memory_usage.set(1024000)  # Peut augmenter ou diminuer
```

Histogram (Histogramme) : C'est comme un seau qui trie automatiquement les valeurs par catégories. Parfait pour mesurer la distribution des temps de réponse.

```
python

# Exemple : mesurer les temps de réponse
response_time = Histogram('http_request_duration_seconds', 'Durée des requêtes HTTP')
response_time.observe(0.25)  # Enregistre qu'une requête a pris 0.25 secondes
```

Summary (Résumé) : Similaire à l'histogramme mais calcule directement des percentiles. Moins utilisé que l'histogramme.

3. Le modèle Pull de Prometheus

Contrairement à d'autres systèmes qui attendent que les applications leur envoient des données (modèle Push), Prometheus vient chercher les données lui-même (modèle Pull). C'est comme un facteur qui passe à intervalles réguliers pour collecter le courrier.

Avantages du modèle Pull :

- Prometheus contrôle la fréquence de collecte
- Plus facile de détecter si une application ne répond plus
- Moins de surcharge réseau

Architecture de Prometheus

Prometheus fonctionne comme une machine bien huilée avec plusieurs composants :

Le serveur principal collecte et stocke les métriques dans une base de données de série temporelle. Chaque donnée est horodatée, permettant de voir l'évolution dans le temps.

Les exporters sont des programmes spécialisés qui exposent les métriques de systèmes qui ne les fournissent pas nativement. Par exemple, un exporter peut surveiller un serveur Apache et transformer ses logs en métriques Prometheus.

Service Discovery permet à Prometheus de découvrir automatiquement les services à surveiller, particulièrement utile dans des environnements dynamiques comme Kubernetes.

Configuration de Prometheus

Le fichier `prometheus.yml` est le cerveau de Prometheus. Voici un exemple simple mais détaillé :

yaml

```yaml
# Configuration globale
global:
  scrape_interval: 15s     # Fréquence de collecte par défaut
  evaluation_interval: 15s # Fréquence d'évaluation des règles

# Configuration des collectes
scrape_configs:
  - job_name: 'mon-application'
    static_configs:
      - targets: ['localhost:8000']  # Où trouver les métriques
    scrape_interval: 5s              # Collecte toutes les 5 secondes
    metrics_path: '/metrics'         # Chemin des métriques (par défaut)
```

Chapitre 2 : Intégrer Prometheus dans une application

L'instrumentation : rendre votre application "parlante"

L'instrumentation, c'est l'art d'ajouter des capteurs à votre application pour qu'elle puisse raconter son histoire à Prometheus. C'est comme installer des compteurs électriques dans votre maison pour savoir où va votre consommation.

Exemple pratique avec Python/Flask

Reprenons votre application Flask et analysons chaque ligne :

python

```python
from flask import Flask
from prometheus_client import start_http_server, Counter, Histogram
import threading
import time

app = Flask(__name__)

# Créer nos métriques
REQUEST_COUNT = Counter(
    'flask_requests_total',          # Nom de la métrique
    'Nombre total de requêtes HTTP',  # Description
    ['method', 'endpoint', 'status']  # Labels pour catégoriser
)

REQUEST_DURATION = Histogram(
    'flask_request_duration_seconds',
    'Durée des requêtes HTTP en secondes'
)

@app.route("/")
def hello():
    # Mesurer le temps de début
    start_time = time.time()

    try:
        # Votre logique métier ici
        result = "Hello from Flask with Prometheus!"
        status = "200"
        return result
    except Exception as e:
        status = "500"
        raise
    finally:
        # Enregistrer les métriques
        REQUEST_COUNT.labels(method='GET', endpoint='/', status=status).inc()
        REQUEST_DURATION.observe(time.time() - start_time)

def start_metrics_server():
    """Démarre le serveur de métriques sur un port séparé"""
    start_http_server(8000)

if __name__ == "__main__":
    # Démarrer le serveur de métriques en arrière-plan
```

```
threading.Thread(target=start_metrics_server, daemon=True).start()

# Lancer l'application Flask
app.run(host="0.0.0.0", port=5000)
```

Comprendre les labels

Les labels sont comme des étiquettes qui permettent de catégoriser les métriques. Dans l'exemple ci-dessus, `['method', 'endpoint', 'status']` nous permet de distinguer :

- Les requêtes GET des requêtes POST
- Les différents endpoints de notre API
- Les requêtes réussies des requêtes échouées

Chapitre 3 : Le langage PromQL

Introduction à PromQL

PromQL (Prometheus Query Language) est le langage de requête de Prometheus. C'est comme SQL pour les bases de données, mais spécialement conçu pour les données de séries temporelles.

Les requêtes de base

Sélection simple :

```promql
flask_requests_total
```

Cette requête retourne toutes les valeurs de la métrique `flask_requests_total`.

Filtrage avec labels :

```promql
flask_requests_total{method="GET"}
```

Ne retourne que les requêtes GET.

Combinaison de filtres :

```promql
```

```
flask_requests_total{method="GET", status="200"}
```

Seulement les requêtes GET qui ont réussi.

Les fonctions essentielles

rate() - Calculer un taux :

```promql
rate(flask_requests_total[5m])
```

Calcule le nombre de requêtes par seconde sur les 5 dernières minutes. Très utile pour voir la charge en temps réel.

increase() - Calculer une augmentation :

```promql
increase(flask_requests_total[1h])
```

Montre combien de requêtes ont été ajoutées dans la dernière heure.

avg() - Calculer une moyenne :

```promql
avg(flask_request_duration_seconds)
```

Donne le temps de réponse moyen.

Agrégation par labels :

```promql
sum by (endpoint) (rate(flask_requests_total[5m]))
```

Groupe les requêtes par endpoint et somme les taux.

Chapitre 4 : Comprendre Grafana

Qu'est-ce que Grafana ?

Si Prometheus est le comptable qui collecte tous les chiffres, Grafana est le designer qui transforme ces chiffres en graphiques magnifiques et compréhensibles. Grafana excelle dans la visualisation de données et la création de tableaux de bord interactifs.

Les concepts clés de Grafana

Data Sources (Sources de données)

Une data source est une connexion vers un système qui stocke des données. Grafana peut se connecter à Prometheus, mais aussi à des bases de données SQL, Elasticsearch, InfluxDB, et bien d'autres.

Dashboards (Tableaux de bord)

Un dashboard est une collection de panneaux (panels) arrangés sur une page. C'est votre vue d'ensemble sur vos systèmes.

Panels (Panneaux)

Un panel est un composant individuel qui affiche des données sous forme de graphique, table, jauge, etc. Chaque panel contient une ou plusieurs requêtes vers les sources de données.

Types de visualisations

Time Series (Séries temporelles) : Le graphique classique avec le temps en abscisse. Parfait pour voir l'évolution d'une métrique dans le temps.

Stat : Affiche une valeur unique, idéal pour des indicateurs clés comme "Nombre total d'utilisateurs connectés".

Gauge (Jauge) : Comme un compteur de vitesse, parfait pour montrer un pourcentage ou une valeur dans une plage.

Bar Gauge (Barre de jauge) : Plusieurs jauges en barres, utile pour comparer plusieurs valeurs.

Table : Affichage en tableau, pratique pour lister des informations détaillées.

Chapitre 5 : Créer votre premier dashboard

Étape par étape : de zéro au dashboard fonctionnel

1. Configuration de la source de données

Connectez-vous à Grafana (généralement http://localhost:3000, admin/admin par défaut) et suivez ces étapes :

Dans le menu latéral, cliquez sur "Configuration" puis "Data Sources". Ajoutez une nouvelle source de données Prometheus avec l'URL de votre serveur Prometheus. Dans un environnement Docker, ce sera quelque chose comme `http://prometheus:9090`.

2. Création du dashboard

Cliquez sur le "+" dans le menu latéral puis "Dashboard". Vous arrivez sur une page vide avec un bouton "Add new panel".

3. Premier panel : Nombre total de requêtes

Cliquez sur "Add new panel". Dans la section Query, entrez :

```promql
flask_requests_total
```

Changez la visualisation pour "Stat" dans le menu déroulant à droite. Donnez un titre à votre panel comme "Total Requests". Sauvegardez le panel.

4. Deuxième panel : Taux de requêtes

Ajoutez un nouveau panel avec la requête :

```promql
rate(flask_requests_total[5m])
```

Gardez la visualisation "Time series" pour voir l'évolution dans le temps. Titre : "Requests Rate (per second)".

5. Troisième panel : Temps de réponse

Nouveau panel avec :

```promql
histogram_quantile(0.95, flask_request_duration_seconds_bucket)
```

Cette requête montre le 95e percentile des temps de réponse, c'est-à-dire que 95% des requêtes sont plus rapides que cette valeur.

Organisation et esthétique

Grafana permet de redimensionner et déplacer les panels par glisser-déposer. Organisez votre dashboard de manière logique : les métriques les plus importantes en haut et bien visibles, les détails plus bas.

Chapitre 6 : Concepts avancés

Variables et templating

Les variables permettent de rendre vos dashboards dynamiques. Par exemple, créer une variable "environment" qui permet de basculer entre production, staging, et développement d'un simple clic.

```promql
flask_requests_total{environment="$environment"}
```

Alerting (Alertes)

Les alertes sont des notifications automatiques quand certaines conditions sont remplies. Par exemple, vous pouvez être alerté si :

- Le taux d'erreur dépasse 5%
- Le temps de réponse moyen dépasse 2 secondes
- Votre application ne répond plus

Bonnes pratiques de monitoring

La règle des quatre signaux d'or :

1. **Latence** : Combien de temps prennent vos requêtes ?
2. **Trafic** : Combien de requêtes recevez-vous ?
3. **Erreurs** : Combien de requêtes échouent ?
4. **Saturation** : À quel point vos ressources sont-elles utilisées ?

Métriques métier vs métriques techniques

Métriques techniques : CPU, mémoire, temps de réponse, taux d'erreur HTTP.

Métriques métier : Nombre de commandes passées, revenus générés, utilisateurs actifs. Ces métriques sont souvent plus importantes pour comprendre la santé réelle de votre application du point de vue utilisateur.

Chapitre 7 : Déploiement et architecture en production

Considérations de performance

Prometheus stocke toutes les données localement par défaut. En production, vous devez planifier :

- La rétention des données (combien de temps garder les métriques)
- L'espace disque nécessaire
- La fédération pour des déploiements multi-sites

Haute disponibilité

Pour des systèmes critiques, vous pouvez déployer plusieurs instances Prometheus en parallèle ou utiliser des solutions comme Thanos pour la haute disponibilité et le stockage à long terme.

Sécurité

En production, activez l'authentification dans Grafana, utilisez HTTPS, et limitez l'accès réseau à Prometheus. Les métriques peuvent parfois révéler des informations sensibles sur votre architecture.

Exercices pratiques

Exercice 1 : Enrichir votre application Flask

Ajoutez ces métriques à votre application :

- Un gauge pour le nombre d'utilisateurs connectés
- Un histogram pour la taille des réponses HTTP
- Un counter pour différents types d'erreurs

Exercice 2 : Dashboard avancé

Créez un dashboard avec :

- Un panel showing les top 5 endpoints les plus utilisés
- Une heatmap des temps de réponse
- Un panel d'alertes visuelles (rouge si erreur > 5%)

Exercice 3 : Monitoring système

Installez node_exporter pour monitorer votre serveur et créez des panels pour :

- L'utilisation CPU
- L'utilisation mémoire
- L'espace disque disponible

Conclusion : La surveillance comme culture

Prometheus et Grafana ne sont pas seulement des outils, ils incarnent une philosophie : l'observabilité. Dans un monde où les systèmes deviennent de plus en plus complexes, la capacité à comprendre ce qui se passe dans nos applications devient cruciale.

La surveillance efficace suit un cycle :

1. **Instrumenter** : Ajouter des métriques significatives
2. **Collecter** : Prometheus rassemble les données
3. **Visualiser** : Grafana rend les données compréhensibles
4. **Alerter** : Être notifié des problèmes
5. **Analyser** : Comprendre les causes et améliorer

En maîtrisant ces outils, vous développez une vision en profondeur de vos systèmes qui vous permettra de résoudre les problèmes plus rapidement, d'optimiser les performances, et de prendre des décisions basées sur des données réelles plutôt que sur des suppositions.

Le monitoring n'est pas une tâche à effectuer après coup, mais une discipline à intégrer dès la conception. Chaque nouvelle fonctionnalité devrait s'accompagner de la réflexion : "Comment vais-je savoir si cela fonctionne bien ?"