Prometheus & Grafana

Introduction à Prometheus et Grafana

- Prometheus et Grafana sont deux outils puissants de monitoring qui travaillent en tandem pour offrir une surveillance efficace des applications et des infrastructures.
- Prometheus se concentre sur la collecte des métriques, tandis que Grafana est une plateforme de visualisation qui permet de créer des tableaux de bord dynamiques et personnalisés.
- Dans le monde informatique moderne, le monitoring est essentiel pour assurer la disponibilité, les performances et la fiabilité des systèmes.
- Prometheus et Grafana sont des solutions open source qui répondent aux défis complexes liés à la surveillance des environnements conteneurisés et des architectures orientées microservices.

Pourquoi Prometheus et Grafana?

- Flexibilité : Adaptation aux architectures modernes, notamment les conteneurs et les microservices.
- Extensibilité : Possibilité d'intégration avec d'autres outils et sources de données.
- Facilité d'utilisation : Interface utilisateur intuitive de Grafana et configuration aisée de Prometheus.

Vue d'ensemble de l'architecture

- Prometheus collecte les métriques via le mécanisme de scrapping, les stocke localement et expose des endpoints pour l'accès.
- Grafana se connecte à Prometheus en tant que source de données pour visualiser ces métriques et créer des tableaux de bord interactifs.
- Le mécanisme de scrapping

Mécanisme de scrapping dans Prometheus

Définition des Cibles (Target) :

L'administrateur configure les cibles que Prometheus doit surveiller. Ces cibles peuvent être des endpoints HTTP exposant des métriques au format Prometheus.

Configuration de la Fréquence de Scrapping :

Une fréquence de scrapping est définie, indiquant à quelle fréquence Prometheus doit interroger les cibles pour collecter les métriques. Cette fréquence peut être configurée en fonction des besoins spécifiques de surveillance.

Interrogation des Endpoints HTTP :

À intervalles réguliers, Prometheus envoie des requêtes HTTP GET aux endpoints spécifiés pour récupérer les métriques. Ces endpoints sont souvent exposés par les applications, les serveurs, ou d'autres composants de l'infrastructure.

Collecte des Métriques :

Lors de chaque requête, les métriques exposées par l'endpoint sont récupérées sous forme de texte au format Prometheus. Ces métriques peuvent inclure des informations telles que l'utilisation du CPU, la mémoire, le nombre de requêtes HTTP, etc.

Stockage Local des Métriques :

Les métriques collectées sont stockées localement dans la base de données de séries temporelles de Prometheus. Chaque métrique est associée à des labels qui fournissent des informations supplémentaires pour permettre une analyse dimensionnelle.

Exposition des Métriques pour l'Évaluation :

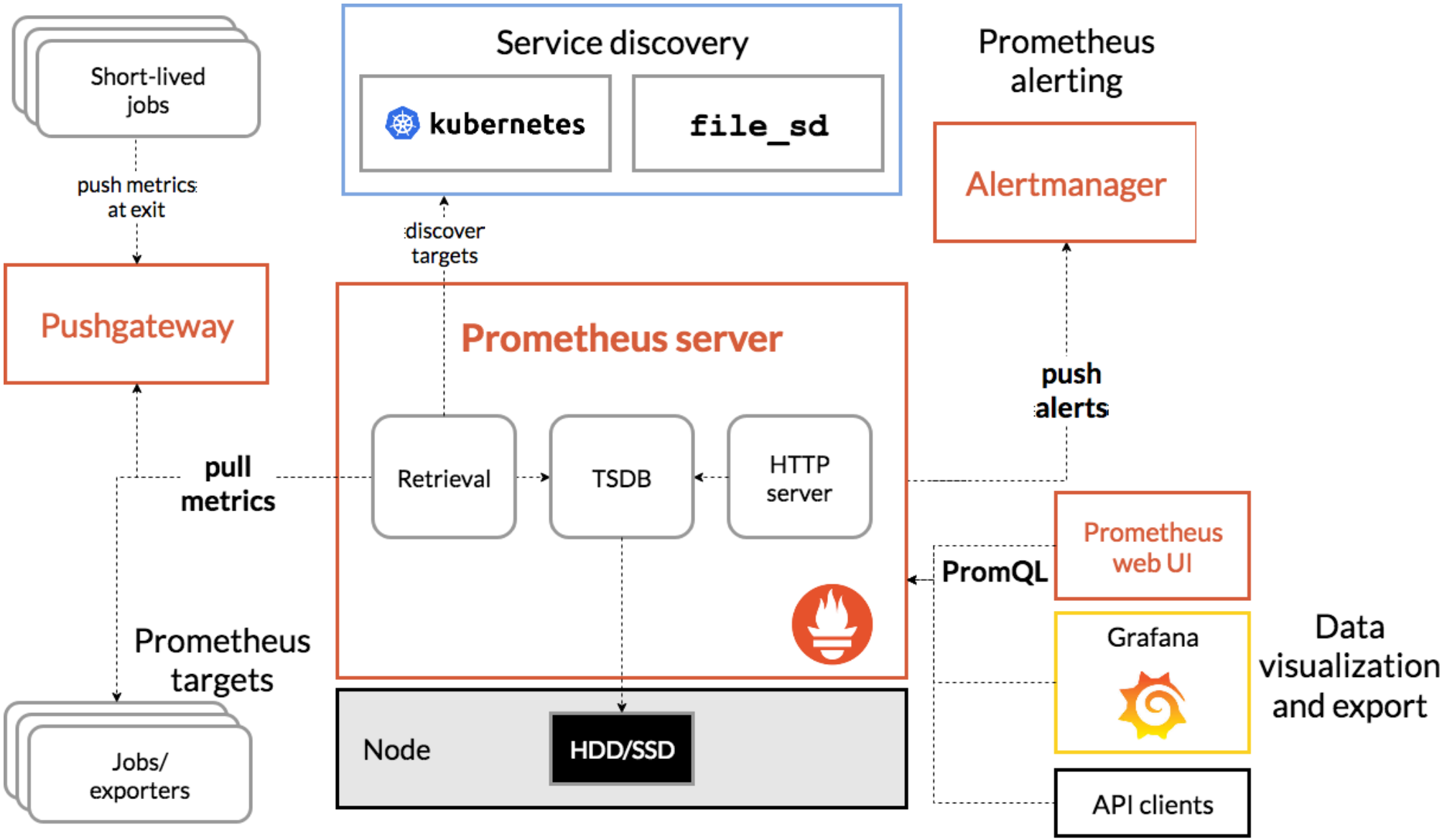
Les métriques collectées sont exposées par Prometheus via un endpoint spécifique (/metrics par défaut). Cela permet à d'autres outils, tels que Grafana, de se connecter à Prometheus et de récupérer ces métriques pour la visualisation et l'analyse.

Évaluation des Règles d'Alerte :

Si des règles d'alerte ont été configurées, Prometheus les évalue régulièrement en fonction des métriques collectées. Si une règle est déclenchée (c'est-à-dire qu'une condition d'alerte est satisfaite), Prometheus déclenche une alerte.

Répétition du Processus :

Ce processus de scrapping est répété à intervalles réguliers, assurant une surveillance continue des cibles configurées.


Short-lived jobs
push metrics at exit
Pushgateway
Service discovery
kubernetes
file_sd
discover targets
Prometheus server
Retrieval
TSDB
HTTP server
pull metrics
Prometheus targets
Jobs/ exporters
Node
HDD/SSD
Prometheus alerting
Alertmanager
push alerts
PromQL
Prometheus web UI
Grafana
API clients
Data visualization and export

Modèle de données de Prometheus

- Métriques : Mesures quantifiables telles que le temps d'exécution, l'utilisation du CPU, etc.
- Labels : Attributs ajoutant des dimensions aux métriques, permettant une plus grande flexibilité dans l'analyse.
- Time Series : Ensemble ordonné de points de données associés à un timestamp, essentiel pour suivre l'évolution des métriques.

Métriques

Les métriques sont des mesures quantifiables, telles que des compteurs, des jauges ou des histogrammes. Voici quelques exemples :

http_requests_total : Compteur du nombre total de requêtes HTTP reçues.

cpu_usage_seconds_total : Compteur du temps total d'utilisation du CPU.

memory_usage_bytes : Jauge représentant la quantité actuelle de mémoire utilisée.

Labels

Les labels ajoutent des dimensions aux métriques, permettant une plus grande flexibilité dans l'analyse. Chaque série temporelle associée à une métrique peut avoir des labels différents. Voici des exemples :

http_requests_total{method="GET", status="200"} : Nombre de requêtes HTTP GET réussies.

http_requests_total{method="POST", status="404"} : Nombre de requêtes HTTP POST avec erreur 404.

cpu_usage_seconds_total{cpu="cpu0"} : Temps total d'utilisation du CPU pour le cœur CPU0.

Séries Temporelles

Les séries temporelles représentent l'évolution d'une métrique au fil du temps. Elles sont identifiées par le nom de la métrique et un ensemble de labels. Voici des exemples :

http_requests_total{method="GET"} : Séries temporelle pour le nombre de requêtes HTTP GET.

http_requests_total{method="POST"} : Séries temporelle pour le nombre de requêtes HTTP POST.

cpu_usage_seconds_total{cpu="cpu0"} : Séries temporelle pour le temps d'utilisation du CPU sur le cœur CPU0.

Définir les métriques à superviser dans Prometheus

Dans Prometheus, les métriques que le système doit superviser sont définies au niveau des cibles de scrapping.

Les cibles sont les endroits (c'est-à-dire les applications, services ou serveurs) que Prometheus va interroger régulièrement pour collecter des métriques.

La définition des cibles se fait généralement dans un fichier de configuration appelé le fichier de configuration de découverte de cible (Target Discovery File)

Création du Fichier de Configuration de Découverte de Cible

Vous devez créer un fichier de configuration qui spécifie les cibles que Prometheus devrait scrapper. Cela peut être un fichier YAML ou JSON. Un exemple de fichier YAML pourrait ressembler à ceci :

yaml

```yaml
job_name: 'my_application'
static_configs:
  - targets: ['localhost:9090', 'app-server:8080']
```

Dans cet exemple, my_application est le nom du job, et les cibles sont localhost:9090 (où Prometheus lui-même expose des métriques par défaut) et app-server:8080 (où votre application expose des métriques).

Intégration avec le Serveur Prometheus

- L'application à superviser doit exposer des métriques au format Prometheus via un endpoint HTTP spécifique (par défaut /metrics)
- Configurer Prometheus pour inclure le fichier de configuration de découverte de cible en dans le fichier de configuration principal de Prometheus (généralement prometheus.yml).

```
scrape_configs:
  - job_name: 'my_application'
    static_configs:
      - targets: ['localhost:9090']
    metric_relabel_configs:
      - source_labels: [__name__]
        regex: 'http_requests_total'
        action: keep
      - source_labels: [method]
        regex: 'POST'
        action: drop
```

Stockage des données

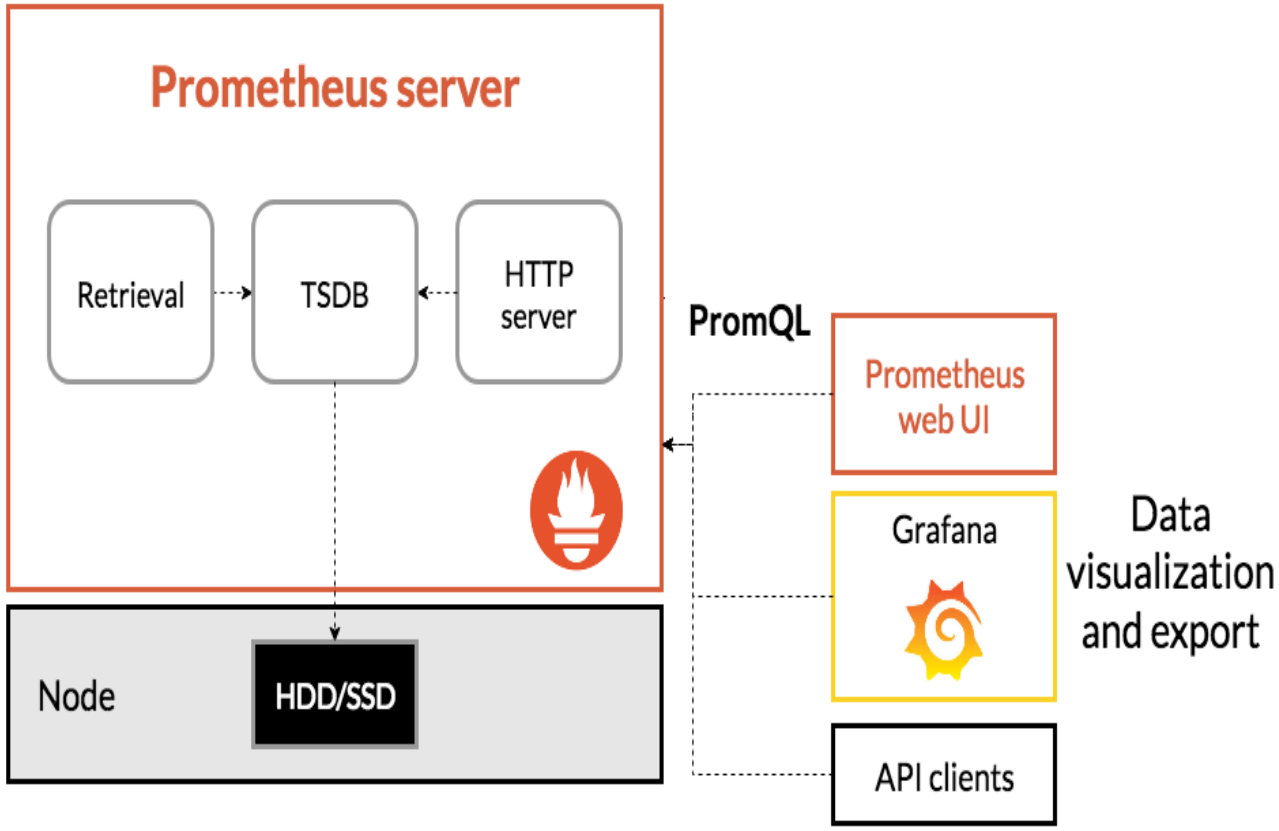
- Prometheus utilise un stockage local pour les données chronologiques, ce qui facilite l'accès et l'analyse.
- La rétention des données et la compaction garantissent une utilisation efficace de l'espace de stockage.

Langage de requête Prometheus

- PromQL est un langage de requête puissant permettant d'interroger les métriques de manière flexible.
- Exemples de requêtes, telles que l'agrégation, la filtration et la comparaison de métriques.

Nombre total de requêtes HTTP/méthode :

promql : *sum by (method) (http_requests_total)*



Alerting avec Prometheus

Configuration des règles d'alerte pour surveiller les seuils prédéfinis.

Intégration avec des canaux d'alerte externes, tels que Slack ou email, pour une notification rapide en cas de problème.

Définition d'une Règle d'Alerte Simple

Surveillez le taux d'erreur HTTP dépassant un seuil donné pendant les 5 dernières minutes

```yaml
groups:
- name: example
  rules:
  - alert: HighErrorRate
    expr: rate(http_requests_total{status="500"}[5m]) > 0.1
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "High error rate on {{ $labels.instance }}"
      description: "{{ $labels.instance }} has a high error rate."
```

Introduction à Grafana

- Grafana est une plateforme de visualisation qui offre une interface graphique conviviale pour explorer les données de Prometheus.
- La connectivité directe avec Prometheus en tant que source de données simplifie la création de tableaux de bord dynamiques.
- La création de tableaux de bord dans Grafana est intuitive, avec une interface glisser-déposer pour les graphiques, les jauges et les autres widgets.
- Les métriques de Prometheus peuvent être sélectionnées facilement pour personnaliser les visualisations.
- Possibilité d'ajouter des widgets personnalisés et des panneaux spécifiques aux besoins de surveillance.
- Grafana prend en charge l'ajout de plugins pour étendre les fonctionnalités et intégrer d'autres sources de données.