

La surveillance Prometheus est en passe de devenir l'un des outils de surveillance Docker et Kubernetes à utiliser. Ce guide explique comment mettre en œuvre la surveillance Kubernetes avec Prometheus. Vous apprendrez à déployer le serveur Prometheus, les exportateurs de métriques, la configuration de métriques kube-state-metric, l'extraction, la récupération et la collecte de métriques, la configuration des alertes avec Alertmanager et des tableaux de bord avec Grafana. Nous expliquerons comment le faire manuellement et en exploitant certaines des méthodes de déploiement / installation automatisées telles que les opérateurs Prometheus.

Ce guide est composé de quatre parties:

1 – Celle ci;)

- Introduction à Prometheus et ses concepts de base
- Comment Prometheus se compare-t-il aux autres solutions de surveillance
- Comment installer Prométhée
- Surveillance d'un service Kubernetes
 - Prometheus exporters
- Surveillance d'un cluster Kubernetes
 - Kubernetes node
 - Kube State Metrics
- Composants internes de Kubernetes

2 - Comment configurer et exécuter des composants supplémentaires de la pile Prometheus dans Kubernetes:

Alertmanager, Push gateway, Grafana, stockage externe, alertes.

3 - L'opérateur Prometheus, Définitions de ressources personnalisées, déploiement entièrement automatisé de Kubernetes pour Prometheus, AlertManager et Grafana.

4 - Considérations relatives aux performances Prometheus, haute disponibilité, stockage externe, limites de dimensionnalité.

Pourquoi utiliser Prometheus pour la surveillance de Kubernetes

Deux évolutions technologiques ont nécessité un nouveau cadre de surveillance:

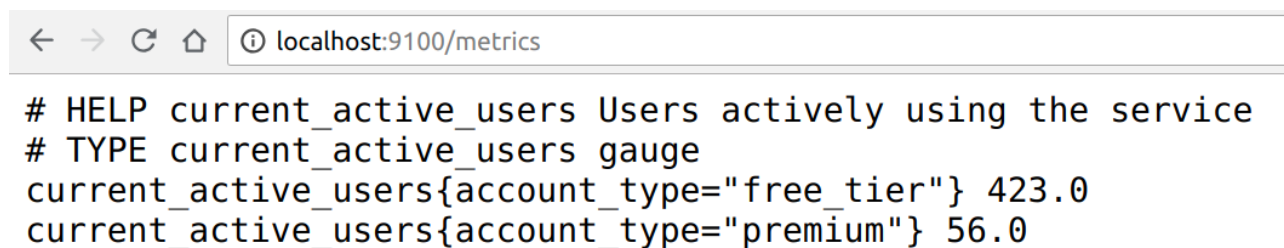
La Culture DevOps: Avant l'émergence de DevOps, on surveillait des hôtes, des réseaux et des services. Désormais, les développeurs doivent avoir la possibilité d'intégrer facilement des métriques relatives aux applications dans le cadre de l'infrastructure, car ils participent davantage au pipeline CI / CD et peuvent effectuer eux-mêmes de nombreuses opérations de débogage. Le suivi devait être démocratisé, rendu plus accessible et couvrir d'autres couches de la pile classique.

Containers et Kubernetes: les infrastructures basées sur des conteneurs modifient radicalement nos méthodes de journalisation, de débogage, de haute disponibilité... et la surveillance ne fait pas exception. Vous avez maintenant un très grand nombre d'entités logicielles volatiles, de services, d'adresses de réseau virtuel, de métriques exposées qui apparaissent ou disparaissent soudainement. Les outils de surveillance traditionnels ne sont pas conçus pour gérer cela.

Pourquoi Prométhée est-il le bon outil pour les environnements conteneurisés?

Un modèle de données multidimensionnel: le modèle repose sur des paires clé-valeur, similaires à la façon dont Kubernetes organise ses métadonnées d'infrastructure à l'aide d'étiquettes (Label). Il permet des séries de données temporelles flexibles et précises, alimentant son langage de requête Prometheus PromQL.

Format et protocoles accessibles: Exposer les métriques Prometheus est une tâche assez simple. Les métriques sont facilement compréhensibles, dans un format qui se passe d'explications et sont publiées en HTTP standard. Vous pouvez vérifier que les métriques sont correctement exposées en utilisant simplement votre navigateur Web:



```
# HELP current_active_users Users actively using the service
# TYPE current_active_users gauge
current_active_users{account_type="free_tier"} 423.0
current_active_users{account_type="premium"} 56.0
```

La Découverte de services: le serveur Prometheus est chargé de supprimer périodiquement les cibles afin que les applications et les services n'aient pas à s'inquiéter de l'émission de données (les métriques sont extraites, et non poussées). Ces serveurs Prometheus disposent de plusieurs méthodes pour détecter automatiquement les cibles de « scrap », certaines pouvant être configurées pour filtrer et faire correspondre les métadonnées de conteneur, ce qui en fait un excellent choix pour les charges de travail éphémères Kubernetes.

Composants modulaires et hautement disponibles: La collecte de métriques, les alertes, la visualisation graphique, etc., sont effectuées par différents services composables. Tous ces services sont conçus pour prendre en charge la redondance et le « sharding »

Comment Prometheus se compare-t-il aux autres outils de surveillance de Kubernetes

Prometheus a publié la version 1.0 en 2016, donc il s'agit d'une technologie relativement récente. Lorsque Prometheus est apparu, une multitude d'outils de surveillance éprouvés étaient disponibles. Comment Prometheus se compare-t-il aux autres projets de surveillance d'anciens combattants?

Clé / valeur vs dot-separated dimensions: plusieurs moteurs tels que StatsD / Graphite utilisent un format explicite séparé par des points pour exprimer les dimensions, générant ainsi une nouvelle métrique par étiquette:

```
current_active_users.free_tier = 423
current_active_users.premium = 56
```

Cette méthode peut devenir fastidieuse lorsque vous essayez d'exposer des données très dimensionnelles (contenant beaucoup d'étiquettes différentes par métrique). L'agrégation flexible basée sur des requêtes devient également plus difficile.

Imaginez que vous avez 10 serveurs et que vous souhaitez regrouper par code d'erreur. À l'aide de la valeur-clé, vous pouvez simplement grouper la métrique par {http_code = "500"}. En utilisant des dimensions séparées par des points, vous aurez un grand nombre de métriques indépendantes que vous devez agréger à l'aide d'expressions .

Journalisation d'événements et enregistrement de métriques: InfluxDB / Kapacitor sont plus similaires à la pile Prometheus. Ils utilisent une dimensionnalité basée sur les étiquettes et les mêmes algorithmes de compression de données. Influx est toutefois plus adapté à la journalisation d'événements en raison de sa résolution temporelle en nanosecondes et de sa capacité à fusionner différents journaux d'événements. Prometheus convient mieux à la collecte de mesures et dispose d'un langage de requête plus puissant pour les inspecter.

Surveillance Blackbox vs whitebox: Comme nous l'avons déjà mentionné, des outils tels que Nagios / Icinga / Sensu conviennent à la surveillance hôte / réseau / service, tâches classiques d'administrateur système. Nagios, par exemple, est basé sur un hôte. Si vous souhaitez obtenir des détails internes sur l'état de vos micro-services (surveillance des WhiteBox), Prometheus est un outil plus approprié.

Les défis de la surveillance par microservices et Kubernetes avec Prometheus

Il existe des défis uniques propres à la surveillance d'un ou de plusieurs clusters Kubernetes qui doivent être résolus avant de déployer une architecture fiable de surveillance / alerte / graphique.

Surveillance des conteneurs: visibilité

Les conteneurs sont des BlackBox légères, la plupart du temps immuables, ce qui peut poser des problèmes de surveillance... L'API de Kubernetes et les métriques de kube-state-metrics (qui utilisent en natif des métriques prometheus) résolvent une partie de ce problème en exposant des données internes à Kubernetes telles que le nombre de réplicas en cours d'exécution / souhaitées. dans un déploiement, unschedulable nodes,, etc.

Prometheus convient parfaitement aux microservices, car il vous suffit d'exposer un port de métriques et n'aurez donc pas besoin d'ajouter trop de complexité ni d'exécuter des services supplémentaires. Souvent, le service lui-même présente déjà une interface HTTP et le développeur doit simplement ajouter un chemin supplémentaire, tel que / metrics.

Dans certains cas, le service n'est pas prêt à servir les métriques Prometheus et vous ne pouvez pas modifier le code pour le prendre en charge. Dans ce cas, vous devez déployer un exportateur Prometheus [Prometheus exporter](#) fourni avec le service, souvent sous la forme d'un conteneur side-car dans le même pod.

Surveillance dynamique: infrastructure changeante et volatile

Comme nous l'avons déjà mentionné, les entités éphémères qui peuvent démarrer ou arrêter la génération de rapports à tout moment sont un problème pour les systèmes de surveillance classiques et plus statiques.

Prometheus dispose de plusieurs mécanismes de découverte automatique pour remédier à ce problème. Les plus pertinents sont:

Consul: Un outil pour la découverte et la configuration de services. Consul est distribué, hautement disponible et extrêmement évolutif.

Kubernetes: les configurations Kubernetes SD permettent de récupérer des cibles à partir de l'API REST de Kubernetes tout en restant synchronisées avec l'état du cluster.

Prometheus Operator: Pour générer automatiquement des configurations de cible de surveillance basées sur des requêtes d'étiquettes Kubernetes bien connues. Nous nous concentrerons sur cette option de déploiement ultérieurement.

La surveillance de nouvelles couches d'infrastructure:

composants Kubernetes

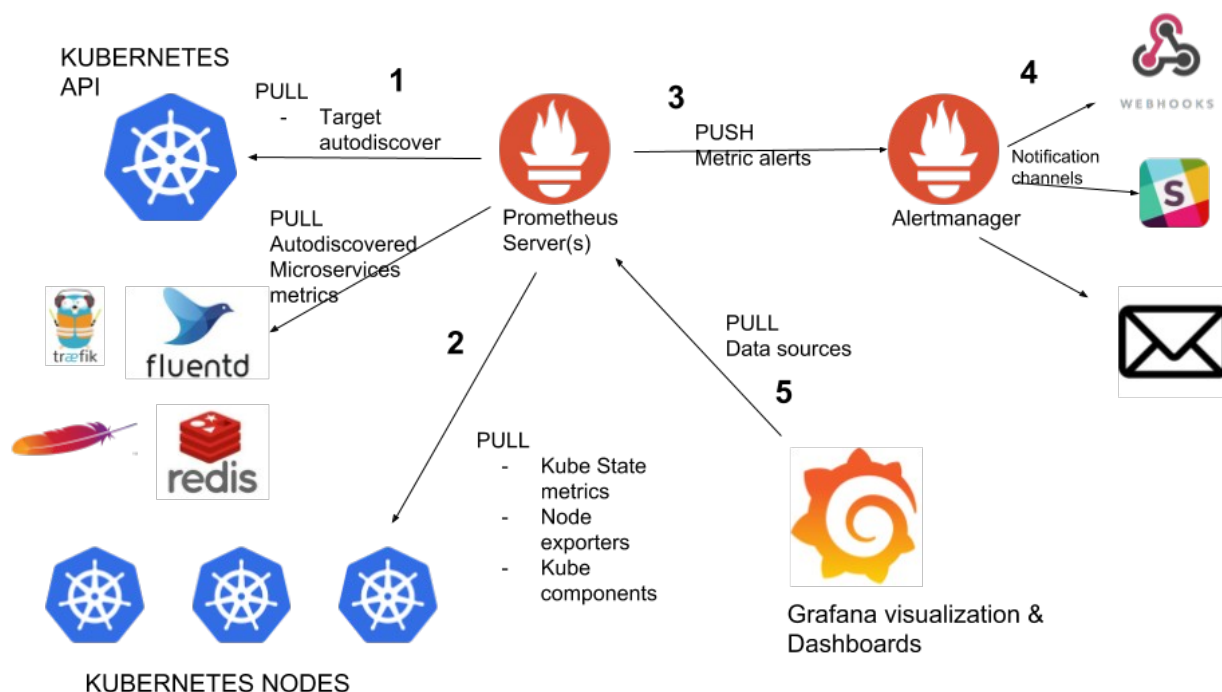
L'utilisation de concepts Kubernetes tels que l'hôte physique ou le port de service devient moins pertinente. Vous devez organiser la surveillance autour de différents groupes, tels que les performances de microservices (avec différents modules répartis sur plusieurs nœuds), les espaces de noms, les versions de déploiement, etc.

En utilisant le modèle de données à base d'étiquettes de Prometheus avec [PromQL](#), vous pouvez facilement vous adapter à ces nouvelles étendues.

Surveillance Kubernetes avec Prometheus:

vue d'ensemble de l'architecture

Nous détaillerons plus tard, ce diagramme couvre les entités de base que nous souhaitons déployer dans notre cluster Kubernetes:



1 - Les serveurs Prometheus ont besoin d'un maximum de reconnaissance automatique de la cible. Il y a plusieurs options pour y parvenir:

- Consul

- Prometheus Kubernetes SD plugin

- L'opérateur Prometheus et ses définitions de ressources personnalisées

2 - Outre les statistiques de l'application, nous souhaitons que Prometheus collecte les statistiques relatives aux services Kubernetes, aux nœuds et au statut d'orchestration.

- Node exporter**, pour les métriques classiques relatives à l'hôte: cpu, mem, réseau, etc.

- Kube-state-metrics**, pour les métriques d'orchestration et de niveau cluster: déploiements, métriques de pod, réservation de ressources, etc.

- Kube-system metrics** à partir de composants internes: kubelet, etcd, dns, ordonnanceur, etc.

3 - Prometheus peut configurer des règles pour déclencher des alertes à l'aide de PromQL.

- AlertManager sera chargé de la gestion des notifications d'alerte, du regroupement, de l'inhibition, etc.

4 - Le composant alertmanager configure les récepteurs, les passerelles pour envoyer des notifications d'alerte.

5 - Grafana peut extraire des métriques de n'importe quel nombre de serveurs Prometheus, de panneaux d'affichage et de tableaux de bord.

Comment installer Prometheus

Il existe différentes manières d'installer Prometheus sur votre hôte ou dans votre cluster Kubernetes:

Directement en tant que fichier binaire s'exécutant sur vos hôtes, ce qui convient parfaitement à des fins d'apprentissage, de test et de développement, mais non approprié pour un déploiement en conteneur.

En tant que conteneur Docker doté de plusieurs options d'orchestration:

Conteneurs Raw Docker, Kubernetes Deployments / StatefulSets, le gestionnaire de packages Helm Kubernetes, les opérateurs Kubernetes, etc.

Passons de déploiements plus manuels à plus automatisés:

Single binary -> Docker container -> Kubernetes Deployment -> Opérateur Prometheus (Chapitre 3)

Vous pouvez directement télécharger et exécuter le binaire Prometheus sur votre hôte:

```
prometheus-2.3.1.linux-amd64$ ./prometheus
level=info ts=2018-06-21T11:26:21.472233744Z caller=main.go:222
msg="Starting Prometheus"
```

Ce qui peut être intéressant d'obtenir une première impression de l'interface Web Prometheus (port 9090 par défaut).

Une meilleure option consiste à déployer le serveur Prometheus dans un conteneur:

```
docker run -p 9090:9090 -v \
/tmp/prometheus.yml:/etc/prometheus/prometheus.yml \
prom/prometheus
```

Notez que vous pouvez facilement adapter ce conteneur Docker à un déploiement Kubernetes qui montera la configuration à partir d'un ConfigMap, exposera un service, déploiera plusieurs réplicas, etc.

```
kubectl create configmap prometheus-example-cm --from-file
prometheus.yml
```

Et puis vous pouvez appliquer ce yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus-deployment
  labels:
    app: prometheus
    purpose: example
spec:
  replicas: 2
  selector:
    matchLabels:
      app: prometheus
      purpose: example
  template:
```

```

metadata:
  labels:
    app: prometheus
    purpose: example
spec:
  containers:
  - name: prometheus-example
    image: prom/prometheus
    volumeMounts:
      - name: config-volume
        mountPath: /etc/prometheus/prometheus.yml
        subPath: prometheus.yml
    ports:
      - containerPort: 9090
  volumes:
  - name: config-volume
    configMap:
      name: prometheus-example-cm
---
kind: Service
apiVersion: v1
metadata:
  name: prometheus-example-service
spec:
  selector:
    app: prometheus
    purpose: example
  ports:
  - name: promui
    protocol: TCP
    port: 9090
    targetPort: 9090

nodePort: 32321

type: NodePort

```

Si vous ne souhaitez pas configurer une adresse IP LoadBalancer / externe, vous pouvez toujours spécifier le type [NodePort](#) pour votre service.

Après quelques secondes, vous devriez voir les pods Prometheus s'exécuter dans votre cluster:

```

kubectl get pods

```

NAME	READY	STATUS	RESTARTS	AGE
prometheus-deployment-68c5f4d474-cn5cb	1/1	Running	0	3h
prometheus-deployment-68c5f4d474-ldk9p	1/1	Running	0	3h

Vous pouvez mettre en œuvre plusieurs modifications de configuration à ce stade, telles que la configuration de l'anti affinity du pod afin de forcer les pods du serveur Prometheus à être situés sur différents nœuds. Nous aborderons les aspects de performance et de haute disponibilité au quatrième chapitre de ce guide.

Une option plus avancée et automatisée consiste à utiliser [Prometheus operator](#), traité dans le troisième chapitre de ce guide. Vous pouvez le considérer comme un méta-déploiement: un déploiement qui gère d'autres déploiements, les configure et les met à jour en fonction de spécifications de service de haut niveau. Nous commencerons par configurer manuellement la pile Prometheus et, dans le chapitre suivant, nous utiliserons l'opérateur pour rendre le déploiement de Prometheus facilement portable, déclaratif et flexible.

Comment surveiller un service Kubernetes avec Prometheus

Les métriques Prometheus sont exposées par les services via HTTP (S). Cette approche présente plusieurs avantages par rapport à d'autres solutions de surveillance similaires:

Vous n'avez pas besoin d'installer un agent sur les noeuds, il vous suffit d'exposer un port HTTP. Les serveurs Prometheus scrap régulièrement (tirent, pull), vous n'avez donc pas à vous soucier de l'ajout de métriques ou de la configuration d'un point de terminaison distant. Plusieurs microservices utilisent déjà HTTP pour leurs fonctionnalités habituelles, et vous pouvez réutiliser ce serveur Web interne et simplement ajouter un dossier tel que **/metrics**.

Le format des mesures lui-même est lisible et facile à comprendre. Si vous êtes le responsable du code du microservice, vous pouvez commencer à publier des métriques sans trop de complexité ni d'apprentissage.

Certains services sont conçus pour exposer les métriques Prometheus (kubelet, le proxy Web Traefik, Istio microservice mesh, etc.). Certains autres services ne sont pas intégrés de manière native, mais peuvent être facilement adaptés via un exportateur. Un exportateur est un service qui collecte des statistiques de service et "traduit" les métriques Prometheus prêtes à être supprimées. Il y a des exemples des deux dans ce chapitre.

Commençons par le meilleur scénario: le microservice que vous déployez offre déjà un endpoint Prometheus.

Traefik est un proxy inverse conçu pour s'intégrer étroitement aux microservices et aux conteneurs. Un cas d'utilisation courant de Traefik doit être utilisé en tant qu'Ingress controller ou Entrypoint, c'est-à-dire le pont entre Internet et les microservices spécifiques de votre cluster.

Vous avez plusieurs options pour installer Traefik et un guide d'installation spécifique à Kubernetes. Si vous souhaitez simplement un déploiement simple de Traefik avec le support Prometheus, utilisez l'extrait de code suivant:

```
kubectl create -f https://raw.githubusercontent.com/mateobur/prometheus-monitoring-guide/master/traefik-prom.yaml
```

```
kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	238d
prometheus-example-service	ClusterIP	10.103.108.86	<none>	9090/TCP	5h
traefik	ClusterIP	10.108.71.155	<none>	80/TCP, 443/TCP, 8080/TCP	35s

Vous pouvez vérifier que les métriques Prometheus sont exposées uniquement à l'aide de curl:

```
curl 10.108.71.155:8080/metrics
# HELP go_gc_duration_seconds A summary of the GC invocation durations.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.4895e-05
go_gc_duration_seconds{quantile="0.25"} 4.4988e-05
...
```

Maintenant, vous devez ajouter la nouvelle cible au fichier de prometheus.yml. Vous remarquerez que Prométhée se « scrap » automatiquement:

```
- job_name: 'prometheus'

# metrics_path defaults to '/metrics'
# scheme defaults to 'http'.

static_configs:
- targets: ['localhost:9090']
```

Ajoutez le nouvel endpoints, traefik :

```
- job_name: 'traefik'
  static_configs:
  - targets: ['traefik:8080']
```

Corrigez le ConfigMap et le déploiement:

```
kubectl create configmap prometheus-example-cm --from-
file=prometheus.yml -o yaml --dry-run | kubectl apply -f -
```

```
kubectl patch deployment prometheus-deployment -p \
'{"spec":{"template":{"metadata":{"labels":{"date":"'`date +%s`'"}}}}}'
```

Si vous accédez à l'URL / target dans l'interface Web Prometheus, vous devriez voir le noeud final Traefik UP:

Prometheus

Alerts

Graph

Status ▾

Help

Targets

AllUnhealthy

prometheus (1/1 up)

show less

Endpoint	State	Labels	Last Scrape	Error
http://localhost:9090/metrics	UP	instance="localhost:9090"	2.692s ago	

traefik (1/1 up)

show less

Endpoint	State	Labels	Last Scrape	Error
http://traefik:8080/metrics	UP	instance="traefik:8080"	6.752s ago	

En utilisant l'interface Web principale, nous pouvons localiser certaines métriques de Traefik (très peu d'entre elles, car nous n'avons aucune interface Traefik ni arrière-plan configurés pour cet exemple) et récupérer ses valeurs:

Nous avons déjà un exemple concret de Prométhée sur Kubernetes!

Comment surveiller les services Kubernetes avec les exportateurs Prometheus

Surveillance des applications à l'aide des exportateurs Prometheus

Il est probable que de nombreuses applications que vous souhaitez déployer dans votre cluster Kubernetes n'exposent pas les métriques Prometheus prêtes à l'emploi. Dans ce cas, vous devez regrouper un exportateur Prometheus, un processus supplémentaire capable d'extraire les formats d'état / journaux / autres métriques du service principal et d'exposer ces informations en tant que métriques Prometheus. En d'autres termes, un adaptateur Prometheus.

Vous pouvez déployer un pod contenant le serveur Redis et un conteneur Prometheus sidecar à l'aide de la commande suivante:

```
# Clone the repo if you don't have it already
git clone git@github.com:mateobur/prometheus-monitoring-guide.git
kubectl create -f prometheus-monitoring-guide/redis_prometheus_exporter.yaml
```

Si vous affichez le redis pod, vous remarquerez qu'il contient deux conteneurs:

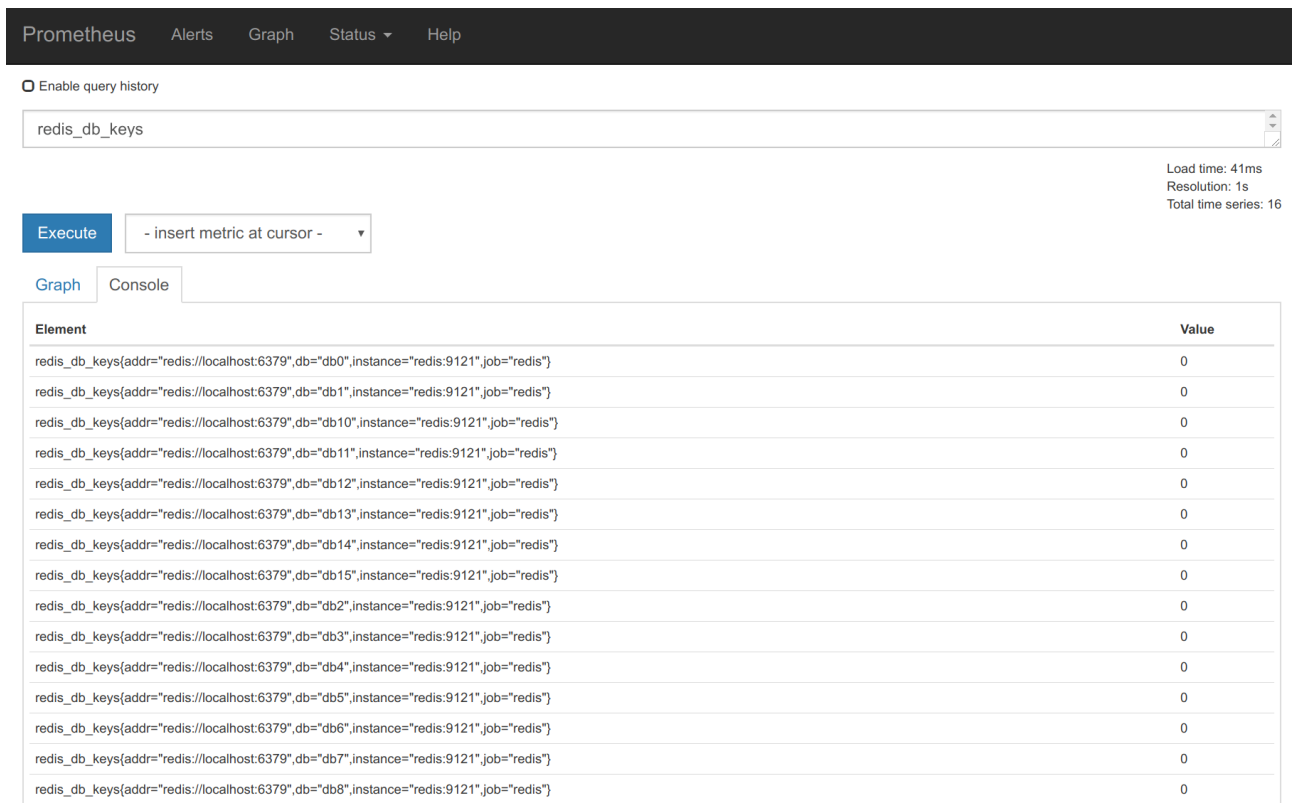
```
kubectl get pod redis-546f6c4c9c-lmf6z
```

NAME	READY	STATUS	RESTARTS	AGE
redis-546f6c4c9c-lmf6z	2/2	Running	0	2m

Maintenant, il vous suffit de mettre à jour la configuration de Prometheus et de recharger comme nous l'avons fait dans la dernière section:

```
- job_name: 'redis'
  static_configs:
    - targets: ['redis:9121']
```

To obtain all the redis service metrics:



Comment surveiller les applications Kubernetes avec Prometheus !

Surveillance du cluster Kubernetes avec Prometheus et les métriques kube-state-state

Outre la surveillance des services déployés dans le cluster, vous souhaitez également surveiller le cluster Kubernetes lui-même. Les trois aspects de la surveillance des clusters à prendre en compte sont les suivants:

Les hôtes (nœuds) Kubernetes - métriques sysadmin classiques telles que cpu, charge, disque, mémoire, etc.

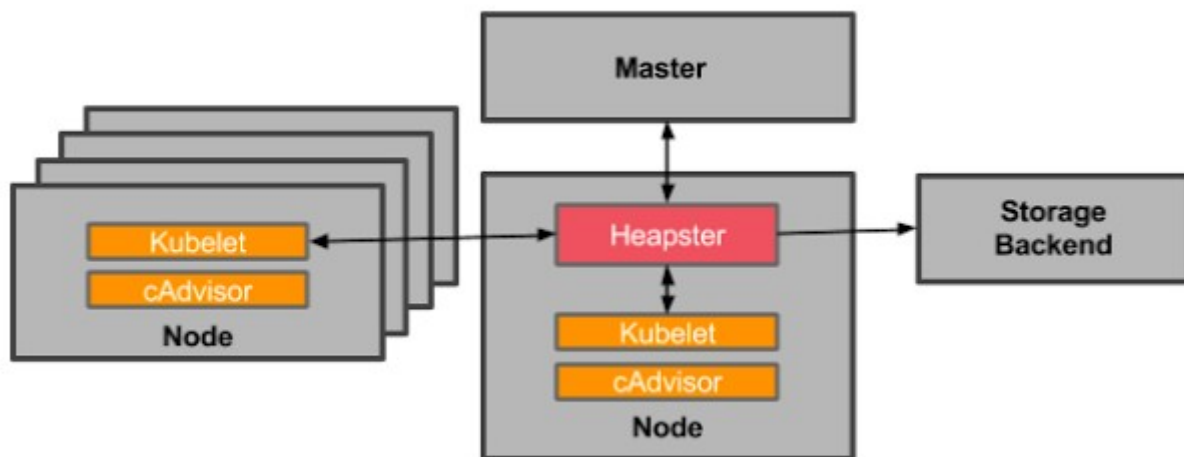
les métriques d'orchestration: état de déploiement, demandes de ressources, planification et latence du serveur api, etc.

Les Composants internes du système kube - planificateur, le gestionnaire de contrôleur, le service DNS, etc.

L'architecture de surveillance interne de Kubernetes a récemment subi quelques modifications que nous allons essayer de résumer ici. Pour plus d'informations, vous pouvez lire [design proposal](#).

Surveillance des composants Kubernetes avec une stack Prometheus

Heapster: Heapster est un agrégateur de données de surveillance et d'événement à l'échelle du cluster qui s'exécute en tant que pod dans le cluster.



Outre les points de terminaison Kubelets / **cAdvisor**, vous pouvez ajouter d'autres sources de métriques à Heapster, telles que les métriques kube-state-metrics (voir ci-dessous).

Heapster est maintenant DEPRECATED, son remplacement est le serveur de métriques.

cAdvisor: cAdvisor est un agent d'analyse des performances et de l'utilisation des ressources open source. Il est spécialement conçu pour les conteneurs et prend en charge les conteneurs Docker de manière native. Dans Kubernetes, cAdvisor s'exécute avec Kubelet. Les agrégateurs récupérant les métriques du nœud local et les métriques Docker extraient directement vers endpoints Prometheus.

Kube-state-metrics: kube-state-metrics est un service simple qui écoute le serveur d'API Kubernetes et génère des métriques sur l'état des objets tels que les déploiements, les nœuds et les pods. Il est important de noter que la métrique kube-state-metric est simplement un point de

terminaison de métriques, une autre entité doit la parcourir et fournir un stockage à long terme (à savoir le serveur Prometheus).

Metrics-server: Metrics Server est un agrégateur de données d'utilisation des ressources à l'échelle du cluster. Il est destiné à être le remplacement par défaut de Heapster. Encore une fois, le serveur de mesures ne présentera que les derniers points de données et n'est pas responsable du stockage à long terme.

Ainsi:

Kube-state metrics est centré sur les métadonnées d'orchestration: déploiement, pod, statut de réplique, etc.

Metrics-server se concentre sur l'implémentation de l'API des métriques de ressources: CPU, descripteurs de fichier, mémoire, latences des requêtes, etc.

Surveillance des nœuds Kubernetes avec Prometheus

Les nœuds ou les hôtes Kubernetes doivent être surveillés. Nous disposons de nombreux outils pour surveiller un hôte Linux. Dans ce guide, nous allons utiliser le nœud exportateur Prometheus [node-exporter](#):

Ils sont hébergés par le projet Prométhée lui-même

Est-ce celui qui sera déployé automatiquement lorsque nous utiliserons l'opérateur Prometheus dans les prochains chapitres

Peut être déployé en tant que DaemonSet et, par conséquent, sera automatiquement mis à l'échelle si vous ajoutez ou supprimez des nœuds de votre cluster.

Vous avez plusieurs options pour déployer ce service, par exemple, en utilisant DaemonSet dans ce référentiel:

```
kubectl create ns monitoring
kubectl create -f https://raw.githubusercontent.com/bakins/minikube-prometheus-demo/master/node-exporter-daemonset.yml
```

Ou en utilisant Helm / Tiller:

Si vous souhaitez utiliser Helm, n'oubliez pas de créer les rôles RBAC et les comptes de service serviceaccount.

```
helm init --service-account tiller
helm install --name node-exporter stable/prometheus-node-exporter
```

Une fois le graphique installé et en cours d'exécution, vous pouvez afficher le service que vous devez « scrap »:

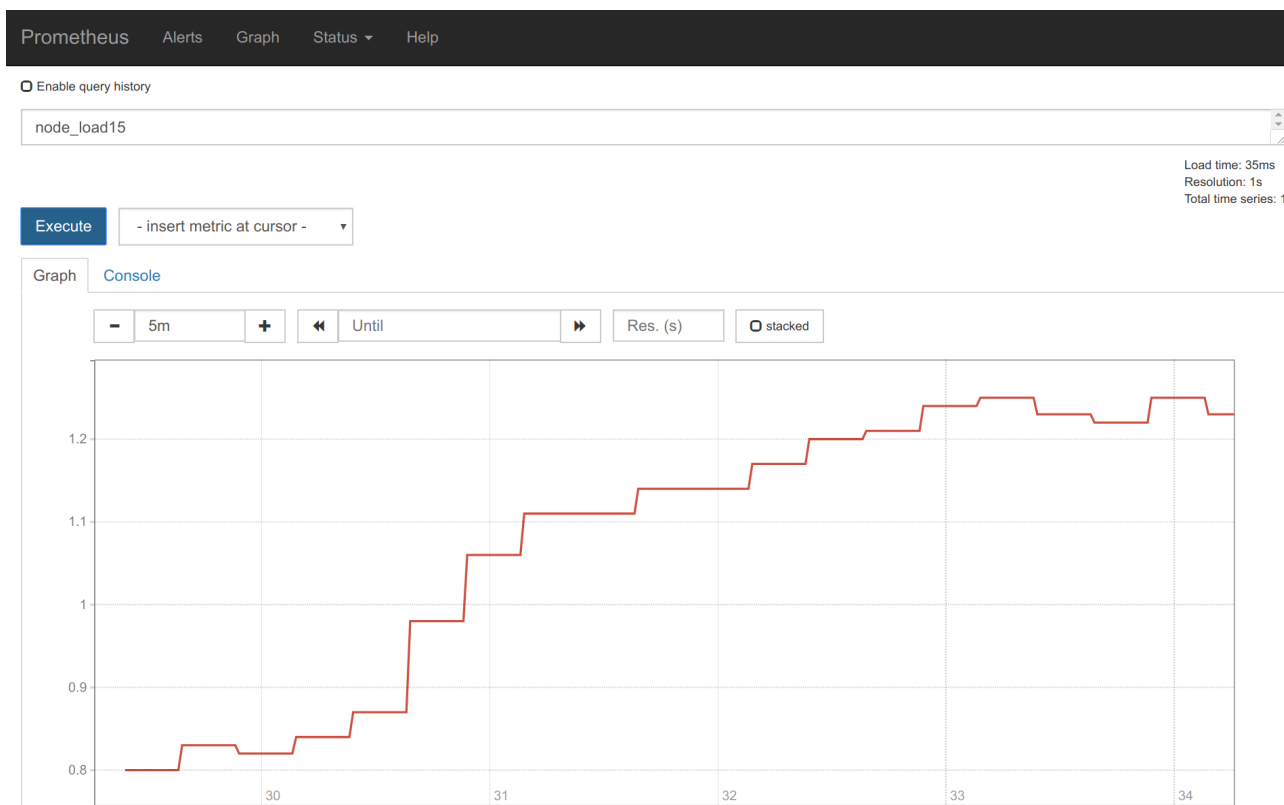
```
kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
node-exporter-prometheus-node-exporter	ClusterIP	10.101.57.207	<none>	9100/TCP

Une fois que vous avez ajouté la configuration de scrape comme nous l'avons fait dans les sections précédentes, vous pouvez commencer à collecter et afficher les métriques de nœud:

```
- job_name: 'node-exporter'

static_configs:
- targets: ['localhost:9100']
```



Surveillance des métriques d'état de kube avec Prometheus

Le déploiement et la surveillance des métriques kube-state-state est également une tâche assez simple. Encore une fois, vous pouvez déployer directement comme dans l'exemple ci-dessous ou utiliser un Chart Helm.

```
git clone https://github.com/kubernetes/kube-state-metrics.git
kubectl apply -f kube-state-metrics/kubernetes/
...
kubectl get svc -n kube-system
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kube-dns      ClusterIP   10.96.0.10     <none>         53/UDP,53/TCP    13h
kube-state-metrics ClusterIP   10.102.12.190 <none>         8080/TCP,8081/TCP 1h
```

Là encore, il vous suffit de définir ce service (port 8080) dans la configuration Prometheus. N'oubliez pas d'utiliser le nom de domaine complet cette fois-ci:

```
- job_name: 'kube-state-metrics'
  static_configs:
    - targets: ['kube-state-metrics.kube-system.svc.cluster.local:8080']
```

Et après?

Le chapitre suivant couvrira les composants supplémentaires qui sont généralement déployés avec le service Prometheus. Nous allons commencer à utiliser le langage PromQL pour agréger les métriques, déclencher des alertes et générer des tableaux de bord de visualisation.

Une Stack complète de «surveillance Kubernetes avec Prometheus» comprend beaucoup plus que des serveurs Prometheus qui collectent des métriques en grattant (scrap) des points de terminaison. Pour déployer une véritable solution de surveillance Kubernetes et microservices, vous avez besoin de nombreux autres composants, dont des règles et des alertes (AlertManager), une couche de visualisation graphique (Grafana), un stockage des mesures à long terme, ainsi que des adaptateurs de mesures supplémentaires pour le logiciel non compatible Out of the Box.

Dans cette seconde partie, nous allons couvrir brièvement tous ces composants de support, en supposant que vous compreniez déjà les bases du déploiement d'un serveur de surveillance Prometheus, décrites dans le chapitre précédent.

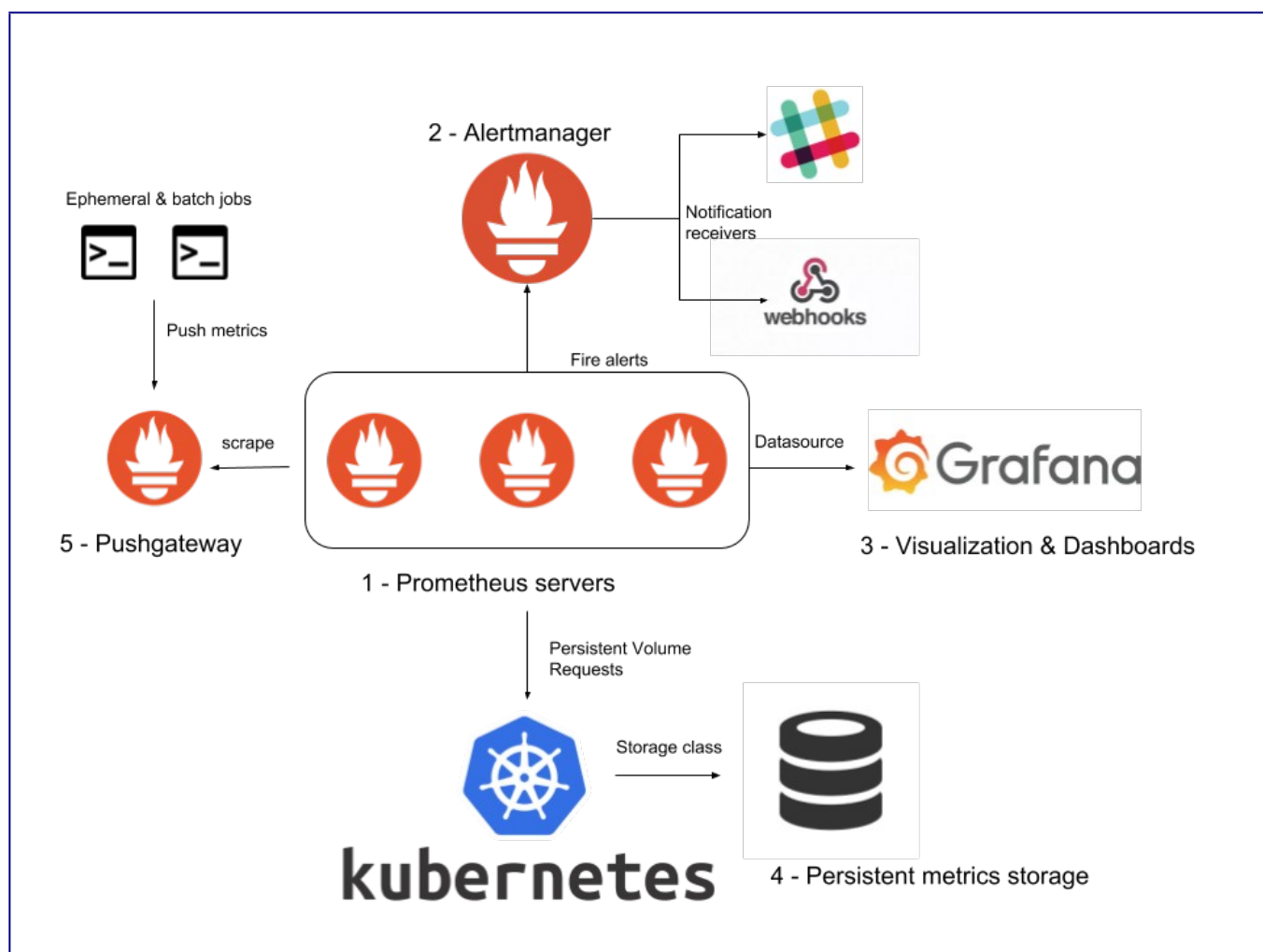
1 - Surveillance Kubernetes avec Prometheus, concepts de base et déploiement initial

2 - Surveillance Kubernetes avec Prometheus: AlertManager, Grafana, PushGateway (celui-ci):

- AlertManager, Prometheus alertant pour Kubernetes
- Tableaux de bord Grafana, Prometheus
- Métriques Prometheus pour les tâches éphémères - Push Gateway
- Stockage de métriques persistantes Prometheus

Prometheus monitoring stack – Architecture

Commençons par un aperçu de l'architecture de déploiement afin de placer tous les composants dont nous discuterons dans les sections suivantes.



Les serveurs Prometheus, expliqués dans la première partie, sont au cœur de ce déploiement. Les serveurs Prometheus transmettent des alertes au composant **AlertManager**, tandis qu'Alertmanager classifie, achemine et informe en utilisant les différents canaux ou récepteurs de notification.

Nous allons configurer une source de données Prometheus pour Grafana, présentant les visualisations de données et le tableau de bord via son interface Web.

En utilisant Kubernetes PersistentVolumes, nous allons configurer le stockage des métriques à long terme.

Nous couvrirons également les tâches de maintenance éphémères et les métriques associées. La Pushgateway sera chargée de les stocker suffisamment longtemps pour pouvoir être collectées par les serveurs Prometheus.

AlertManager, Prometheus Alert pour Kubernetes

Il y a deux parties à alerter avec Prometheus:

- Les conditions d'alerte réelles sont configurées à l'aide de PromQL sur les serveurs Prometheus.
- Le composant AlertManager reçoit les alertes actives:
 - AlertManager les classe et les groupe en fonction de leurs métadonnées (étiquettes), et les met éventuellement en sourdine ou les informe à l'aide d'un destinataire (Webhook, email, PagerDuty, etc.).
 - AlertManager est conçu pour être mis à l'échelle horizontalement. Une instance peut communiquer avec ses pairs en offrant une configuration minimale.

Nous allons commencer par quelques concepts de base. La section suivante contient un exemple pratique que vous pouvez exécuter immédiatement dans votre cluster.

Passons en revue la structure d'une règle d'alerte Prometheus:

```
groups:
- name: etcd
  rules:
  - alert: NoLeader
    expr: etcd_server_has_leader{job="kube-etcd"} == 0
    for: 1m
    labels:
      severity: critical
      k8s-component: etcd
    annotations:
      description: etcd member {{ $labels.instance }} has no leader
      summary: etcd member has no leader
```

La clé **expr** est une expression de Prométhée qui sera évaluée périodiquement et qui se déclenchera si elle est vraie.

Vous pouvez définir une durée d'évaluation minimale pour éviter toute alerte sur des problèmes temporaires d'auto-guérison.

Comme beaucoup d'autres architectures dans Kubernetes, les étiquettes que vous choisissez sont très pertinentes pour le regroupement, la classification et la hiérarchie.

Ultérieurement, AlertManager décidera quelles alertes ont une priorité plus élevée, comment les regrouper, etc., toutes basées sur ces étiquettes.

Vous pouvez maintenant afficher les alertes que le serveur Prometheus a chargées avec succès, directement sur l'interface Web (Status -> Rules):

Prometheus	Alerts	Graph	Status ▾	Help
------------	--------	-------	----------	------

Rules

example	592.8us
Rule	Evaluation Time
alert: <code>HighErrorRate</code> expr: <code>job:request_latency_seconds:mean5m{job="myjob"} > 0.5</code> for: 10m labels: severity: page annotations: summary: High request latency	237.4us
alert: <code>DeadMansSwitch</code> expr: <code>vector(1)</code> labels: severity: none annotations: description: This is a DeadMansSwitch meant to ensure that the entire Alerting pipeline is functional. summary: Alerting DeadMansSwitch	252.1us

Celle qui sont « Firing » également (Alerts):

Prometheus	Alerts	Graph	Status ▾	Help
------------	--------	-------	----------	------

Alerts

Show annotations

DeadMansSwitch (1 active)			
alert: <code>DeadMansSwitch</code> expr: <code>vector(1)</code> labels: severity: none annotations: description: This is a DeadMansSwitch meant to ensure that the entire Alerting pipeline is functional. summary: Alerting DeadMansSwitch			
Labels	State	Active Since	Value
alertname="DeadMansSwitch" severity="none"	FIRING	2018-07-24 18:38:25.582918892 +0000 UTC	1

HighErrorRate (0 active)			
--------------------------	--	--	--

Vous avez maintenant des conditions et des alertes que vous devez transférer à un AlertManager.

À l'instar des points de terminaison, les services AlertManager peuvent également être détectés automatiquement à l'aide de différentes méthodes: découverte DNS, Consul, etc.

Étant donné que nous parlons de la surveillance Prometheus avec Kubernetes, nous pouvons tirer parti d’une abstraction de base de Kubernetes:

le service.

```
alerting:
  alertmanagers:
  - scheme: http
    static_configs:
    - targets:
      - "alertmanager-service:9093"
```

Du point de vue du serveur Prometheus, cette configuration n'est qu'un nom statique, mais le service Kubernetes peut effectuer différentes transmissions HA / LoadBalancing.

Le logiciel AlertManager lui-même est un logiciel sophistiqué, couvrant les bases:

- AlertManager regroupe les différentes alertes en fonction de leurs libellés et de leur origine.

Ce regroupement et cette hiérarchie forment «l'arbre de routage». Un arbre de décision qui détermine les actions à prendre.

Par exemple, vous pouvez configurer votre arbre de routage de sorte que chaque alerte portant le libellé k8s-cluster-composant soit envoyée à l'adresse de messagerie «cluster-admin».

À l'aide des règles d'inhibition, une alerte ou un groupe d'alertes peut être inhibé si une autre alerte est déclenchée. Par exemple, si un cluster est en panne et complètement inaccessible, il est inutile de notifier le statut des microservices individuels qu'il contient.

Les alertes peuvent être transférées aux 'destinataires', c'est-à-dire aux passerelles de notification telles que courrier électronique, PagerDuty, webhook, etc.

Un exemple simple AlertManager config:

```
global:
  resolve_timeout: 5m
route:
  group_by: ['alertname']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 1h
  receiver: 'sysdig-test'
receivers:
  - name: 'sysdig-test'
    webhook_configs:
      - url: 'https://webhook.site/8ce276c4-40b5-4531-b4cf-5490d6ed83ae'
```

Cet arbre de routage configure simplement un nœud racine.

Dans cet exemple, nous utilisons un Webhook JSON générique comme récepteur. Il n'est pas nécessaire de déployer votre propre serveur. <https://webhook.site> vous fournira un point de terminaison temporaire à des fins de test (évidemment, votre code URL spécifique exemple variera).

Surveillance Prométhée avec AlertManager, essayez-la!

Il vous suffit de cloner le référentiel et d'appliquer les fichiers yaml dans le bon ordre:

```
git clone git@github.com:mateobur/prometheus-monitoring-guide.git
cd prometheus-monitoring-guide/alertmanager-example/
kubectl create cm prometheus-example-cm --from-file prometheus.yaml
kubectl create cm prometheus-rules-general --from-file generalrules.yaml
```

Maintenant accédez [https://webhook.site/](https://webhook.site) et remplacer l'url radom dans `alertmanager.yaml`,
`url: 'your url here':`

```
kubectl create cm alertmanager-config --from-file alertmanager.yaml
kubectl create -f prometheus-example.yaml
kubectl create -f alertmanager-deployment.yaml
```

Au bout de quelques secondes, tous les pods devraient être à l'état de fonctionnement:

```
kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
alertmanager-deployment-78944d4bfc-rk2sg	1/1	Running	0	27s
prometheus-deployment-784594c89f-fjbd6	1/1	Running	0	32s
prometheus-deployment-784594c89f-gpvzf	1/1	Running	0	32s

Cette configuration inclut un «DeadManSwitch», c'est une alerte qui se déclenchera toujours, destinée à vérifier que tous les microservices Prometheus -> AlertManager -> Les canaux de notification fonctionnent comme prévu:

Vérifiez l'URL webhook à la recherche de la notification au format JSON:

FirstPreviousNextLast

Request Detailspermalink

URL<https://webhook.site/f8bafa7d-ef9b-4184-82cc-4907c67f486b>

Host35.193.137.98

Date2018-08-01 01:20:23

ID97bbfcac-3105-4959-aba2-2e5c44d282d4

☐ Auto redirectSettings...Redirect Now☒ Format JSON☐ Auto navigate

Headers

content-typeapplication/json

content-length941

user-agentAlertmanager/0.15.1

hostwebhook.site

```
{
  "receiver": "sysdig-test",
  "status": "firing",
  "alerts": [
    {
      "status": "firing",
      "labels": {
        "alertname": "DeadMansSwitch",
        "monitor": "sysdig-prometheus",
        "severity": "none"
      },
      "annotations": {
        "description": "This is a DeadMansSwitch meant to ensure that the entire Alerting pipeline is functional.",
        "summary": "Alerting DeadMansSwitch"
      },
      "startsAt": "2018-07-25T09:49:25.582918892Z",
      "endsAt": "0001-01-01T00:00:00Z",
      "generatorURL": "http://prometheus-deployment-784594c89f-nxj4x:9090/graph?g0.expr=vector%281%29&g0.tab=1"
    }
  ],
  "groupLabels": {
    "alertname": "DeadMansSwitch"
  },
  "commonLabels": {
    "alertname": "DeadMansSwitch",
    "monitor": "sysdig-prometheus",
    "severity": "none"
  },
  "commonAnnotations": {
    "description": "This is a DeadMansSwitch meant to ensure that the entire Alerting pipeline is functional.",
    "summary": "Alerting DeadMansSwitch"
  },
  "externalURL": "http://alertmanager-deployment-78944d4bfc-rvb85:9093",
  "version": "4",
  "groupKey": "{}:{alertname=\"DeadMansSwitch\"}"
}
```

Grafana, Surveillance Kubernetes avec Prometheus - Tableaux de bord

Le projet Grafana est une plate-forme d'analyse et de surveillance agnostique. Il n'est pas affilié à Prometheus, mais est devenu l'un des composants add-on les plus populaires pour la création d'une solution complète de Prometheus.

Nous allons utiliser le gestionnaire de paquets Helm Kubernetes pour ce déploiement, ce qui nous permettra de] configurer Grafana à l'aide d'un ensemble de ConfigMaps répétables, en évitant toute post-configuration manuelle:

```
helm install --name mygrafana stable/grafana --set
sidecar.datasources.enabled=true --set sidecar.dashboards.enabled=true --set
sidecar.datasources.label=grafana_datasource --set
sidecar.dashboards.label=grafana_dashboard
```

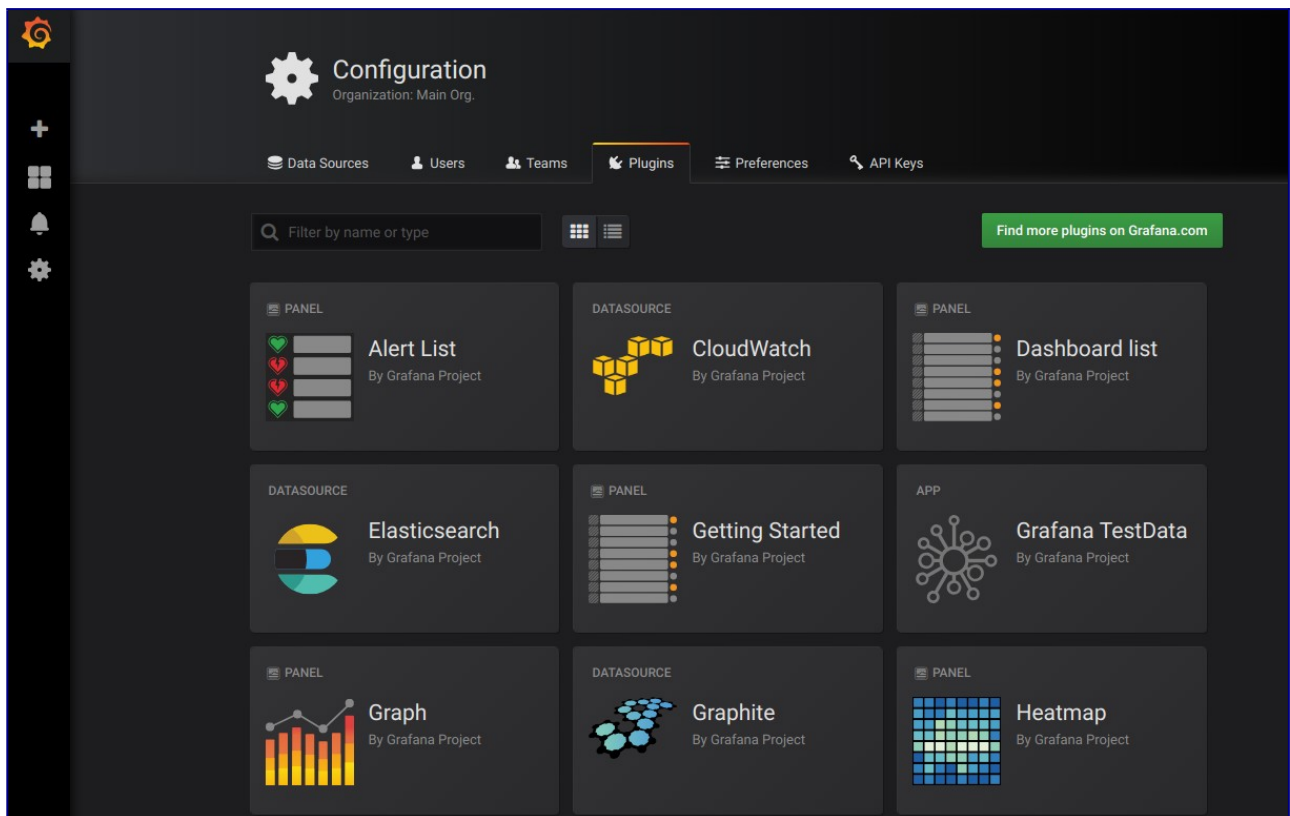
Faites attention aux instructions qui seront affichées par la console, en particulier celles concernant la récupération de votre mot de passe administrateur:

```
kubectl get secret --namespace default mygrafana -o jsonpath="{.data.admin-
password}" | base64 --decode ; echo
```

Vous pouvez transférer directement le port du service grafana à votre hôte local pour accéder à l'interface:

```
kubectl port-forward mygrafana-859c68577f-4h6zp 3000:3000
```

Allez sur <http://localhost:3000> en utilisant votre navigateur si vous voulez jeter un oeil à l'interface:



Deux configurations sont nécessaires pour que Grafana soit utile:

- Les sources de données, le serveur Prometheus dans notre cas
- Les tableaux de bord pour visualiser les métriques choisies

De manière pratique, vous pouvez (et devriez, si possible) autoconfigurer les deux à partir de fichiers de configuration:

```
kubectl create -f prometheus-monitoring-guide / grafana / configmap "dashboard-k8s-capacity"
créé configmap "échantillon-grafana-datasource" créé
```

Ici encore, nous utilisons l'abstraction Service pour pointer sur plusieurs nœuds de serveur Prometheus avec une seule URL.

url: <http://prometheus-example-service:9090>

Ensuite, Grafana fournira la visualisation et les tableaux de bord des métriques collectées par Prométhée:



Métriques Prometheus pour les tâches éphémères - Push Gateway

Tous les programmes ne constituent pas un service continu que vous pouvez vous attendre à supprimer à tout moment. Chaque déploiement informatique comporte une multitude de tâches ponctuelles ou récurrentes pour la sauvegarde, le nettoyage, la maintenance, les tests, etc.

Vous souhaiterez peut-être collecter certaines mesures à partir de ces tâches, mais le modèle cible / récupération de Prometheus ne fonctionnera certainement pas ici.

C'est la raison pour laquelle le projet Prometheus fournit le service Pushgateway: push acceptor pour les tâches éphémères et par lots. Vous pouvez envoyer les métriques à la passerelle et celle-ci conservera les informations afin de pouvoir les récupérer ultérieurement.

Le déploiement d'un pushgateway fonctionnel de base est assez simple:

```
kubectl create -f prometheus-monitoring-guide / pushgateway /
```

Transférez le port de service de votre pod Pushgateway:

```
kubectl port-forward pushgateway-deployment-66478477fb-gmsld 9091: 9091
```

Et essayez de publier une métrique à l'aide de curl. C'est ce que feront vos travaux par lots:

```
echo "some_metric 3.14" | curl --data-binary @-  
http://localhost:9091/metrics/job/some_job
```

Maintenant, vous pouvez gratter cette métrique normalement, même si la source d'origine n'est plus en cours d'exécution:

```
curl localhost: 9091 / metrics | grep une_métrique
```

```
# TYPE some_metric non typé
```

```
some_metric {instance = "", job = "some_job"} 3.14
```

Il vous suffit d'ajouter la passerelle en tant que cible normale dans votre configuration Prometheus pour récupérer ces métriques supplémentaires.

Stockage de métriques persistantes Prometheus

Le serveur Prometheus stockera les métriques dans un dossier local pendant 15 jours, par défaut. Notez également que le stockage de pod local par défaut est éphémère, ce qui signifie que si le pod est remplacé pour une raison quelconque, toutes les métriques disparaîtront.

Tout déploiement prêt à la production nécessite que vous configuriez une interface de stockage persistant capable de conserver les données de métriques historiques et de survivre au redémarrage des pods.

Encore une fois, Kubernetes fournit déjà une abstraction qui remplira ce rôle: le PersistentVolume

Cependant, comme nous venons de le dire, PersistentVolume n'est qu'une abstraction pour un volume de données. Vous avez donc besoin d'une pile matériel / logiciel fournissant les volumes réels.

Il y a plusieurs façons d'accomplir cela:

Les principaux fournisseurs de cloud exposent généralement un orchestrateur de volume au départ, vous n'avez donc pas besoin d'installer de logiciel supplémentaire.

Si vous utilisez vos propres hôtes et n'avez pas encore de fournisseur de stockage, il existe des solutions open source et approuvées par le CNCF telles que Rook.

Vous pouvez également utiliser Helm pour installer Rook si vous souhaitez être opérationnel rapidement.

Plusieurs solutions de stockage commerciales telles que NetApp offrent également une couche de compatibilité pour les volumes persistants de Kubernetes.

Si vous envisagez d'utiliser le stockage avec état, vous pouvez utiliser Kubernetes StatefulSets plutôt que des déploiements. Chaque pod sera lié sans ambiguïté à un PersistentVolume distinct. Vous pourrez donc supprimer et recréer les pods et ils joindront automatiquement le volume correct.

Vous pouvez essayer de détruire le déploiement et créer un service similaire à l'aide de StatefulSets:

```
kubectl delete -f prometheus-monitoring-guide/alertmanager-example/prometheus-example.yaml
kubectl create -f prometheus-monitoring-guide/storage/prometheus-example.yaml
```

Chaque pod créera son propre PersistentVolume:

```
kubectl get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS
CLAIM		STORAGECLASS	REASON	AGE
pvc-b5a4aaa1-9584-11e8-97c9-42010a800278	50Gi	RWO	Delete	Bound
default/prometheus-metrics-db-prometheus-deployment-0		standard		4h
pvc-f4ee61e4-9584-11e8-97c9-42010a800278	50Gi	RWO	Delete	Bound
default/prometheus-metrics-db-prometheus-deployment-1		standard		4h

Il y a trois différences principales entre le déploiement que nous utilisions auparavant et cet ensemble avec état:

Le type d'objet de l'API:

```
kind: StatefulSet
```

VolumeClaim définissant le stockage à créer pour chaque pod de l'ensemble:

```
volumeClaimTemplates:
- metadata:
  name: prometheus-metrics-db
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
```

Les paramètres du serveur Prometheus définissant le répertoire de données et la période de rétention:

```
containers:
- args:
  - --storage.tsdb.path=/data
  - --storage.tsdb.retention=400d
```

En moyenne, Prometheus utilise environ 1-2 octets par échantillon. Ainsi, pour planifier la capacité d'un serveur Prometheus, vous pouvez utiliser la formule brute:

```
needed_disk_space = retention_time_seconds * ingested_samples_per_second *  
bytes_per_sample
```

Conclusions

La première partie de ce guide expliquait les bases du service Prometheus, son intégration avec l'orchestrateur Kubernetes et différents microservices que vous pouvez trouver sur un déploiement typique à l'échelle du cloud. Avec cette discussion sur l'ajout d'alertes, de tableaux de bord et d'un stockage à long terme au-dessus du serveur principal Prometheus, ici dans la partie 2, nous nous rapprochons beaucoup d'une solution de surveillance viable.

Dans le chapitre suivant, nous expliquerons comment utiliser la version Kubernetes de prometheus-operator, qui permet de déployer plus rapidement une pile complète de manière plus automatisée, évolutive et déclarative.