# Sonde Livness et Readness

Il existe plusieurs types de sonde afin de faire un bilan de santé de nos services sur kubernete, ici nous avons utilisé une sonde liveness afin de détecter quand redémarrer un conteneur. Elles permettent de rattraper les deadlocks dans les cas où l’application est en cours d'exécution, mais qu’il est incapable de traiter les requêtes. Mais nous avons aussi utilisé une sonde readiness pour savoir quand un conteneur est prêt à accepter le trafic.

Ci-dessous un exemple de sonde readness/livness mis en place pour l’api web.

Une image contenant texte

Description générée automatiquement

Pour le mettre en place il vous suffira d’exécuter la commande suivante : kubectl appli -f livness-exec.yml

# Mise en place de Prometheus

Qu’est-ce que Prometheus ?

Prometheus est un logiciel de monitoring open-source utilisée afin de surveiller des systemes informatique et de générer des alertes. Pour cela il enregistre des métriques en temps réel dans une base de données de séries temporelles en se basant sur le contenu de point d'entrée exposé à l'aide du protocole HTTP.

Comment mettre en place Prometheus ?

Option 1 : via l’IDE Lens

Pour mettre en place Prometheus nous pouvons directement passé par l’IDE lens qui propose via les paramètres de le mettre en place facilement pour un cluser en activant l’option.

Une image contenant texte

Description générée automatiquement

Option 2 : Via les fichiers de config yaml

Etape 1 : Mise en place d’un Name space

Pour commencer nous créons un Name space Kubernetes nos futurs composants afin d’effectuer la surveillance. Si cet espace dédié n’est pas ils sont déployés au Name space par défaut.

Une image contenant texte

Description générée automatiquement

Commande : kubectl create -f orch\_monitoring\_ns.yml

Etape 2 : Mise en place du cluster Rôle

Dans le fichier yml cluster rôle va être ajouté les autorisations get, list et watch aux nœuds, points de terminaison de services, pods et entrées. Le cluser rôle est lié au name space monitoring crée préalablement. Dans notre cas il va être utile afin de récupérer des métriques à partir de n'importe quel objet stipulé eu sein de celui-ci.

Une image contenant texte

Description générée automatiquement

Commande : kubectl create -f orch\_CluserRole.yml

Etape 3 : Mise en place de la config map

La Configmap est un objet Kubernetes qui va nous permettre le maintien d’une valeur clé qui facilement être utilisée par d'autres objets Kubernetes tels que pod, déploiements et services. Ci-dessous la config map réalisée pour prometheus.



Commande : kubectl create -f orch\_ConfigMap.yml

Etape 4 : Mise en place du fichier de déploiement + service

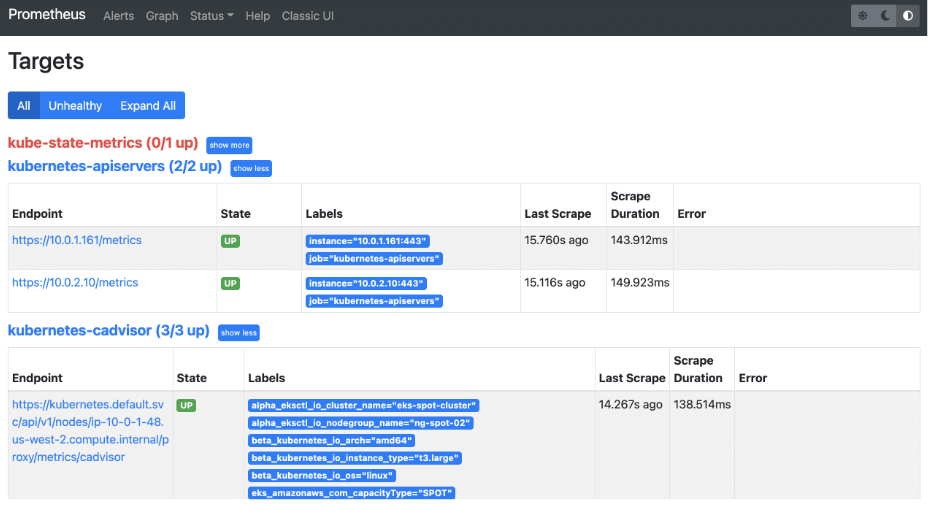
Créez un fichier nommé orch\_prometheus.yml afin de mettre en place l’outil ainsi que son service. Ci-dessous notre fichier yml.

# 

Commande : kubectl create -f orch\_prometheus.yml

Conclusion :

Une fois ces étapes effectuées et le port de prometheus forwardé via Lens. Vous aurez directement accès via le localhost à l’outil et pourrez désormais surveiller les composantes de votre application.



# Déploiement automatisé

Pour le déploiement automatisé, nous allons utiliser une infrastructure générer avec helm create permettant de déployer un notre livrable sur un environnement cible. Pour cela il faut suivre les étapes suivantes.

Etape 1 :

Via votre terminal exécuter la commande « helm create autoDeploy » , cela va permettre de crée un Template de déploiement helm. Ci-dessous l’arborescence du fichier généré.

Une image contenant texte

Description générée automatiquement

Une fois le fichier obtenu il nous suffit d’y effectuer les remplacements nécessaires dans le fichier Template, en mettant à jour les yml qui nous intéresse.

Une image contenant texte

Description générée automatiquement

Ci-dessous un exemple pour l’api job uniquement.



Pour lancer le déploiement il vous suffira d’exécuter la commande suivante :

helm install --dry-run --debug ./autoDeploy --set service.internalPort=8286