Dépannage de l'utilisation CPU/mémoire élevée sur les modules Kubernetes

Table des matières

Introduction

Ce document décrit comment dépanner des problèmes de CPU ou de mémoire sur la plate-forme de déploiement natif en cloud (CNDP) qui est utilisée comme fonction de gestion de session (SMF) ou fonction de contrôle de stratégie (PCF).

1. Alerte de problème - CPU/mémoire élevée sur le pod

Il est important de bien comprendre l'alerte pour commencer à résoudre ce problème. Une explication de toutes les alertes par défaut qui sont préconfigurées se trouve sur cette liaison.

1.1. Alerte pour le processeur

Ici, il y a une alerte active par défaut qui est déclenchée nommée `k8s-pod-cpu-usage-high` .

Vous voyez qu'il est lié à un pod nommé : `smf-udp-proxy-0` et c'est un conteneur : `k8s_smf-udp-proxy_smf-udp-proxy-0_smf`

Vous voyez que ce conteneur est dans l'espace de noms : `smf`

```
alerts active detail k8s-pod-cpu-usage-high 36fbd5e0bbce
severity major
type "Processing Error Alarm"
startsAt 2024-02-23T12:45:44.558Z
source smf-udp-proxy-0
summary "Container: k8s_smf-udp-proxy_smf-udp-proxy-0_smf of pod: smf-udp-proxy-0 in namespace: smf has
labels [ "name: k8s_smf-udp-proxy_smf-udp-proxy-0_smf" "namespace: smf" "pod: smf-udp-proxy-0" ]
```

Sur Kubernetes master, recherchez le pod concerné en entrant cette commande :

```
master $ kubectl get pods smf-udp-proxy-0 -n smf
```

1.2. Alerte de mémoire

Ici, il y a une alerte active par défaut qui est déclenchée nommée container-memory-usage-high .

Vous pouvez voir qu'il est lié à un pod nommé : grafana-dashboard-sgw-765664b864-zwxct et c'est un conteneur : k8s_istio-proxy_grafana-dashboard-sgw-765664b864-zwxct_smf_389290ee-77d1-4ff3-981d-58ea1c8eabdb_0

Ce conteneur se trouve dans l'espace de noms :smf

```
alerts active detail container-memory-usage-high 9065cb8256ba
severity critical
type "Processing Error Alarm"
startsAt 2024-04-25T10:17:38.196Z
source grafana-dashboard-sgw-765664b864-zwxct
summary "Pod grafana-dashboard-sgw-765664b864-zwxct/k8s_istio-proxy_grafana-dashboard-sgw-765664b864-zw
labels [ "alertname: container-memory-usage-high" "beta_kubernetes_io_arch: amd64" "beta_kubernetes_io_
annotations [ "summary: Pod grafana-dashboard-sgw-765664b864-zwxct/k8s_istio-proxy_grafana-dashboard-sg
```

Sur Kubernetes master, recherchez le pod concerné en entrant cette commande :

```
master $ kubectl get pods grafana-dashboard-sgw-765664b864-zwxct -n smf
```

2. Profilage par processus Kubernetes

2.1. Profilage du processeur (/debug/prof/profile)

Le profilage du processeur sert de technique pour capturer et analyser l'utilisation du processeur d'un programme Go en cours d'exécution.

Il échantillonne régulièrement la pile d'appels et enregistre les informations, ce qui vous permet d'analyser l'endroit où le programme passe le plus de temps.

2.2. Profil de mémoire (/debug/prof/heap)

Le profilage de la mémoire fournit des informations sur l'allocation de mémoire et les modèles d'utilisation dans votre application Go.
Il peut vous aider à identifier les fuites de mémoire et à optimiser l'utilisation de la mémoire.

2.3. Profilage de routine (/debug/prof/goroutine)

Le profilage de routage fournit des informations sur le comportement de tous les routages actuels en affichant leurs traces de pile. Cette analyse permet d'identifier les groupes bloqués ou qui fuient et qui peuvent affecter les performances du programme.

2.4. Trouver un port prof sur un pod Kubernetes

commande :

```
master:~$ kubectl describe pod <POD NAME> -n <NAMESPACE> | grep -i pprof
```

Exemple de rapport :

```
master:~$ kubectl describe pod udp-proxy-0 -n smf-rcdn | grep -i pprof
PPROF_EP_PORT: 8851
master:~$
```

3. Données à collecter auprès du système

Pendant la durée du problème et de l'alerte active sur Common Execution Environment (CEE), veuillez collecter les données qui couvrent la période avant, pendant et après le problème :

CEE :

```
cee# show alerts active detail
cee# show alerts history detail
cee# tac-debug-pkg create from yyyy-mm-dd_hh:mm:ss to yyyy-mm-dd_hh:mm:ss
```

Noeud maître CNDP :

```
General information:
master-1:~$ kubectl get pods <POD> -n <NAMESPACE>
master-1:~$ kubectl pods describe <POD> -n <NAMESPACE>
master-1:~$ kubectl logs <POD> -n <NAMESPACE> -c <CONTAINER>

Login to impacted pod and check top tool:
master-1:~$ kubectl exec -it <POD> -n <NAMESPACE> bash
root@protocol-n0-0:/opt/workspace# top

If pprof socket is enabeled on pod:
master-1:~$ kubectl describe pod <POD NAME> -n <NAMESPACE> | grep -i pprof
master-1:~$ curl http://<POD IP>:<PPROF PORT>/debug/pprof/goroutine?debug=1
master-1:~$ curl http://<POD IP>:<PPROF PORT>/debug/pprof/heap
master-1:~$ curl http://<POD IP>:<PPROF PORT>/debug/pprof/profile?seconds=30
```

4. Présentation des résultats du journal d'épreuves collectés

4.1. Lecture du résultat du profilage de mémoire (/debug/prof/heap)

This line indicates that a total of 1549 goroutines were captured in the profile. The top frame (0x9207a9) shows that the function google.golang.org/grpc.(*addrConn).resetTransport is being executed, and the line number in the source code is clientconn.go:1164 .

Chaque section commençant par un nombre (par exemple, 200) représente une trace de pile d'un Goroutine.

```
goroutine profile: total 1549
200 @ 0x4416c0 0x415d68 0x415d3e 0x415a2b 0x9207aa 0x46f5e1
#    0x9207a9    google.golang.org/grpc.(*addrConn).resetTransport+0x6e9    /opt/workspace/gtpc-ep/pkg/
```

The first line in each section shows the number of goroutines with the same stack trace. For example, there are 200 goroutines with the same stack trace represented by memory addresses (0x4416c0 , 0x415d68, and more.). The lines that start with # represent the individual frames of the stack trace. Each frame shows the memory address, function name, and the source code location (file path and line number) where the function is defined.

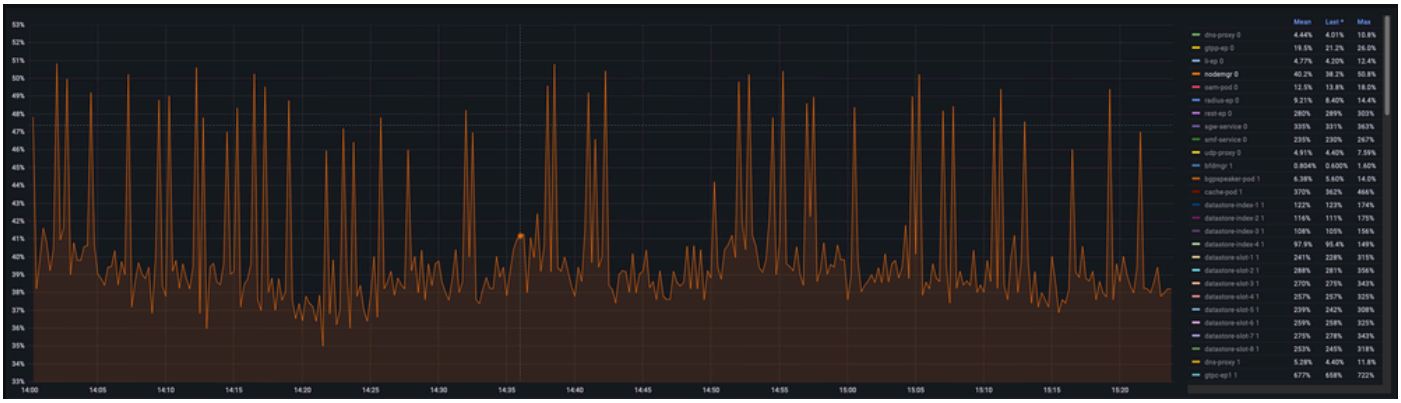
```
200 @ 0x4416c0 0x45121b 0x873ee2 0x874803 0x89674b 0x46f5e1
#    0x873ee1    google.golang.org/grpc/internal/transport.(*controlBuffer).get+0x121    /opt/workspace
#    0x874802    google.golang.org/grpc/internal/transport.(*loopyWriter).run+0x1e2    /opt/workspace/g
#    0x89674a    google.golang.org/grpc/internal/transport.newHTTP2Client.func3+0x7a    /opt/workspace/

92 @ 0x4416c0 0x45121b 0x873ee2 0x874803 0x897b2b 0x46f5e1
#    0x873ee1    google.golang.org/grpc/internal/transport.(*controlBuffer).get+0x121    /opt/workspace
#    0x874802    google.golang.org/grpc/internal/transport.(*loopyWriter).run+0x1e2    /opt/workspace/g
#    0x897b2a    google.golang.org/grpc/internal/transport.newHTTP2Server.func2+0xca    /opt/workspace/
```

5. Grafana

5.1. Requête CPU

```
sum(cpu_percent{service_name=~"[[microservice]]"}) by (service_name,instance_id)
```

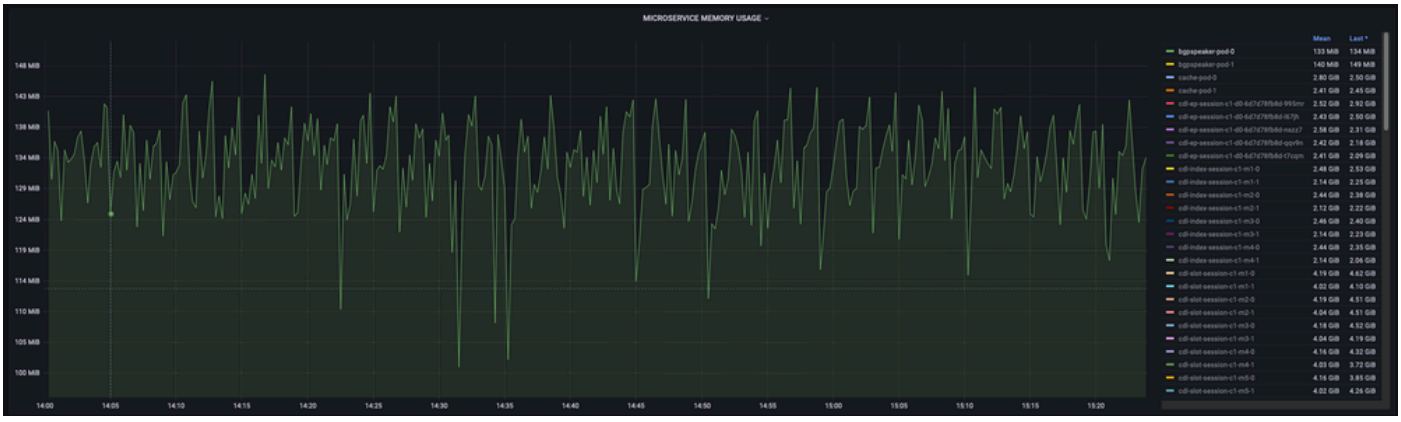
Exemple :



5.2. Requête de mémoire

```
sum(increase(mem_usage_kb{service_name=~"[[microservice]]"}[15m])) by (service_name,instance_id)
```

Exemple :



À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.