

Troubleshooting de Alta Utilização de CPU/Memória em Kubernetes Pods

Contents

[Introdução](#)

[1. Alerta de Problema CPU/Memória Alta no Pod](#)

[1.1. Alerta para CPU](#)

[1.2. Alerta para memória](#)

[2. Análise por processo do Kubernetes](#)

[2.1. Criação de perfil da CPU \(/debug/pprof/profile\)](#)

[2.2. Criação de perfis de memória \(/debug/pprof/heap\)](#)

[2.3. Criação de perfil do Goroutine \(/debug/pprof/goroutine\)](#)

[2.4. Localizar porta prof em um pod Kubernetes](#)

[3. Dados a Serem Coletados no Sistema](#)

[4. Entendendo as Saídas de Log de Prof Coletadas](#)

[4.1. Leitura de saída da criação de perfil de memória \(/debug/pprof/heap\)](#)

[5. Grafana](#)

[5.1. Consulta da CPU](#)

[5.2. Consulta de memória](#)

Introdução

Este documento descreve como solucionar problemas de CPU ou memória na plataforma de Plataforma de Implantação Nativa na Nuvem (CNDP) que é utilizada como Função de Gerenciamento de Sessão (SMF) ou Função de Controle de Política (PCF).

1. Alerta de Problema CPU/Memória Alta no Pod

Entender o alerta é importante para começar bem a solução desse problema. Uma explicação de todos os alertas padrão pré-configurados está [neste link](#).

1.1. Alerta para CPU

Aqui, há um alerta padrão ativo que é acionado com o nome `k8s-pod-cpu-usage-high`.

Você vê que está relacionado a um pod chamado: `smf-udp-proxy-0` e é um contêiner: `k8s_smf-udp-proxy_smf-udp-proxy-0_smf`

Você verá que este contêiner está no namespace: `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" ]
```

No Kubernetes master, localize o pod afetado inserindo este comando:

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

1.2. Alerta para memória

Aqui, há um alerta padrão ativo que é acionado com o nome `container-memory-usage-high`.

Você pode ver que está relacionado a um pod chamado: `grafana-dashboard-sgw-765664b864-zwxct` e é um contêiner: `k8s_istio-proxy_grafana-dashboard-sgw-765664b864-zwxct_smf_389290ee-77d1-4ff3-981d-58ea1c8eabdb_0`

Este contêiner está no namespace: `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-zwxct"
labels [ "alertname: container-memory-usage-high" "beta_kubernetes_io_arch: amd64" "beta_kubernetes_io_c" ]
annotations [ "summary: Pod grafana-dashboard-sgw-765664b864-zwxct/k8s_istio-proxy_grafana-dashboard-sgw-765664b864-zwxct" ]
```

No Kubernetes master, localize o pod afetado inserindo este comando:

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

2. Perfis por processo do Kubernetes

2.1. Criação de perfil da CPU (`/debug/pprof/profile`)

O perfil da CPU serve como uma técnica para capturar e analisar o uso da CPU de um programa Go em execução.

Ele faz uma amostragem periódica da pilha de chamadas e registra as informações, permitindo que você analise onde o programa passa a maior parte do tempo.

2.2. Criação de perfis de memória (`/debug/pprof/heap`)

A criação de perfis de memória fornece informações sobre alocação de memória e padrões de uso em seu aplicativo Go.

Ele pode ajudá-lo a identificar vazamentos de memória e otimizar a utilização da memória.

2.3. Criação de perfil do Goroutine (`/debug/pprof/goroutine`)

O perfil de Goroutine fornece insights sobre o comportamento de todos os Goroutines atuais, exibindo seus rastreamentos de pilha. Essa análise ajuda a identificar os Goroutines presos ou vazando que podem afetar o desempenho do programa.

2.4. Localizar porta prof em um pod Kubernetes

Comando:

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

Saída de exemplo:

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

3. Dados a Serem Coletados no Sistema

Durante o tempo da ocorrência e do alerta ativo sobre o Common Execution Environment (CEE), colete os dados que cobrem o tempo antes e durante/após a ocorrência:

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
```

Nó principal 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 enableed 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. Entendendo as Saídas de Log de Prof Coletadas

4.1. Leitura de saída da criação de perfil de memória (/debug/pprof/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`.

Cada seção que começa com um número (por exemplo, 200) representa um rastreamento de pilha de um 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/m
```

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/g
# 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/g

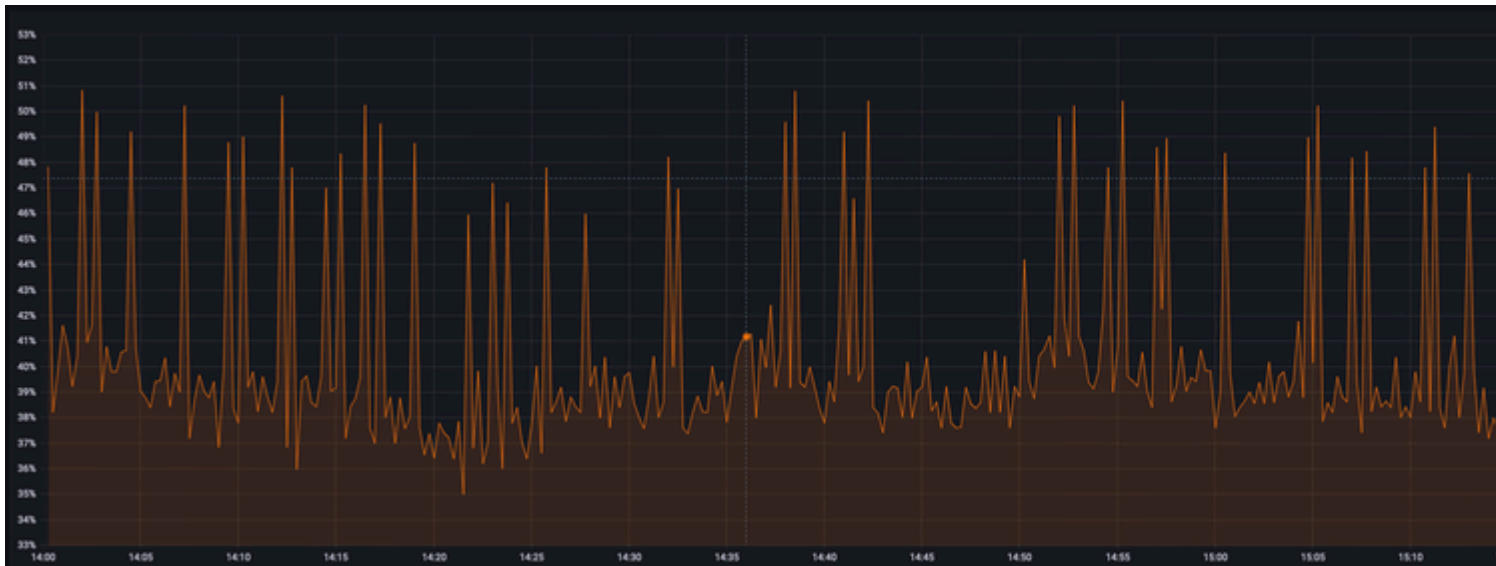
92 @ 0x4416c0 0x45121b 0x873ee2 0x874803 0x897b2b 0x46f5e1
# 0x873ee1 google.golang.org/grpc/internal/transport.(*controlBuffer).get+0x121 /opt/workspace/g
# 0x874802 google.golang.org/grpc/internal/transport.(*loopyWriter).run+0x1e2 /opt/workspace/g
# 0x897b2a google.golang.org/grpc/internal/transport.newHTTP2Server.func2+0xca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

5. Grafana

5.1. Consulta da CPU

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

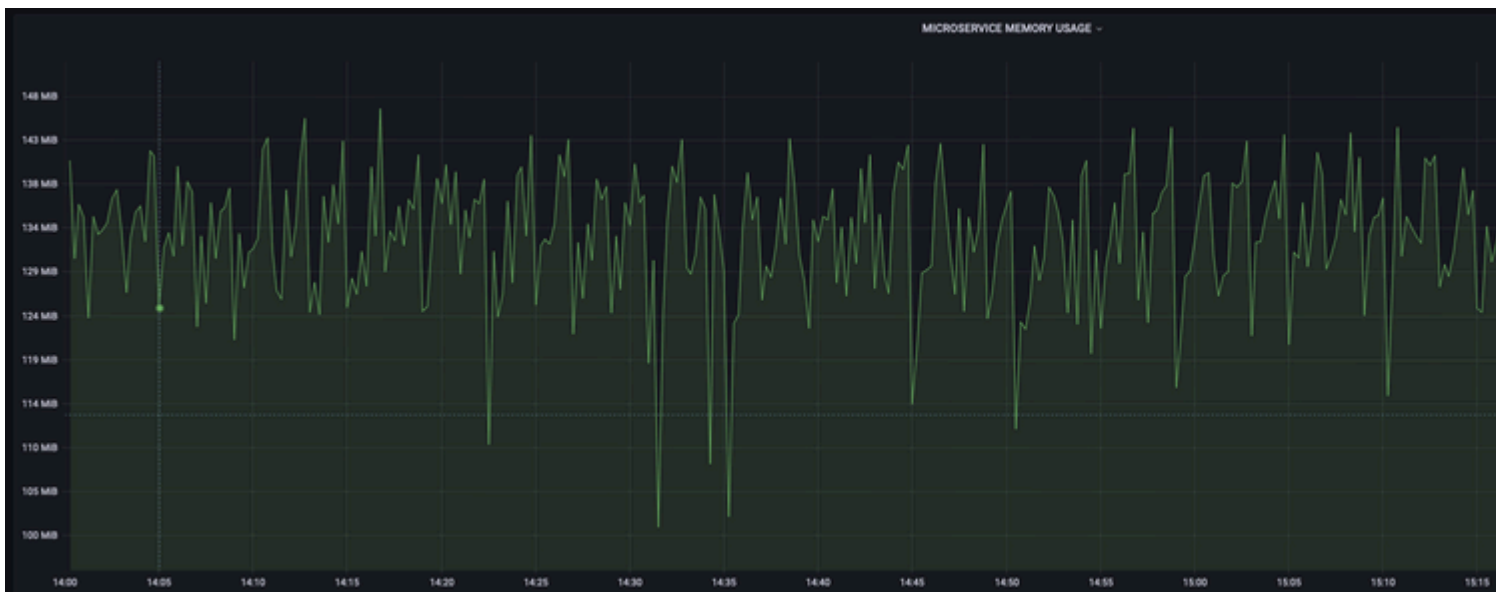
Exemplo:



5.2. Consulta de memória

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

Exemplo:



Sobre esta tradução

A Cisco traduziu este documento com a ajuda de tecnologias de tradução automática e humana para oferecer conteúdo de suporte aos seus usuários no seu próprio idioma, independentemente da localização.

Observe que mesmo a melhor tradução automática não será tão precisa quanto as realizadas por um tradutor profissional.

A Cisco Systems, Inc. não se responsabiliza pela precisão destas traduções e recomenda que o documento original em inglês ([link fornecido](#)) seja sempre consultado.