Solução de problemas e revisão de recursos NDO

Contents

Introduction

Este documento descreve como revisar e solucionar problemas de NDO com a CLI de tempo de execução de contêiner e kubectl.

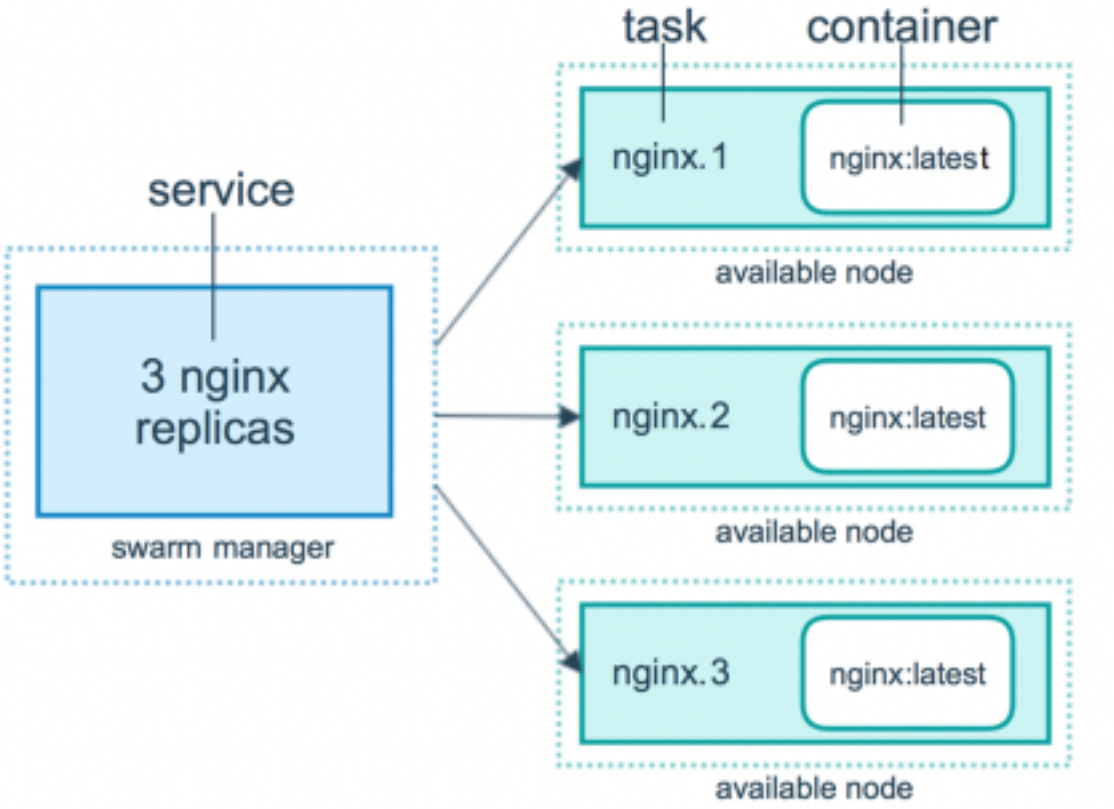
Início rápido do NDO

O Cisco Nexus Dashboard Orchestrator (NDO) é uma ferramenta administrativa de estrutura, que permite que os usuários gerenciem diferentes tipos de estrutura que incluem sites da Cisco® Application Centric Infrastructure (Cisco ACI®), sites da Cisco Cloud ACI e sites do Cisco Nexus Dashboard Fabric Controller (NDFC), cada um gerenciado por seu próprio controlador (cluster APIC, cluster NDFC ou instâncias do APIC de nuvem em uma nuvem pública).

O NDO oferece orquestração consistente de rede e política, escalabilidade e recuperação de desastres em vários data centers através de um único painel de controle.

Nos dias anteriores, o MSC (Multi-Site Controller) foi implantado como um cluster de três nós com os VMWare Open Virtual Appliances (OVAs) que permitiam que os clientes inicializassem um cluster Docker Swarm e os serviços MSC. Este cluster Swarm gerencia os microsserviços MSC como contêineres e serviços Docker.

Esta figura mostra uma visão simplificada de como o Docker Swarm gerencia os microsserviços como réplicas do mesmo contêiner para obter alta disponibilidade.



O Docker Swarm foi responsável por manter o número esperado de réplicas para cada um dos microsserviços da arquitetura MSC. Do ponto de vista do Docker Swarm, o Multi-Site Controller foi a única implantação de contêiner para orquestrar.

O Nexus Dashboard (ND) é um console de gerenciamento central para vários locais de data center e uma plataforma comum que hospeda os serviços de operação de data center da Cisco, que incluem o Nexus Insight e o MSC versão 3.3 em diante, e alterou o nome para Nexus Dashboard Orchestrator (NDO).

Embora a maioria dos microsserviços que compõem a arquitetura MSC permaneçam os mesmos, o NDO é implantado em um cluster Kubernetes (K8s) em vez de em um Docker Swarm. Isso permite que a ND orquestre vários aplicativos ou implantações em vez de apenas um.

Kubernetes com curso de travamento NDO

O Kubernetes é um sistema de código aberto para implantação automatizada, escalabilidade e gerenciamento de aplicativos em contêineres. Como Docker, o Kubernetes trabalha com a tecnologia de contêiner, mas não está ligado ao Docker. Isso significa que o Kubernetes suporta outras plataformas de contêiner (Rkt, PodMan).

Uma diferença chave entre Swarm e Kubernetes é que o último não funciona com contêineres diretamente, ele funciona com um conceito de grupos co-localizados de contêineres, chamados Pods, em vez disso.

Os contêineres em um Pod devem ser executados no mesmo nó. Um grupo de Pods é chamado de Implantação. Uma implantação do Kubernetes pode descrever um aplicativo inteiro.

O Kubernetes também permite que os usuários garantam que uma certa quantidade de recursos esteja disponível para qualquer aplicação. Isso é feito com o uso de controladores de replicação, para garantir que o número de pods seja consistente com os manifestos de aplicativos.

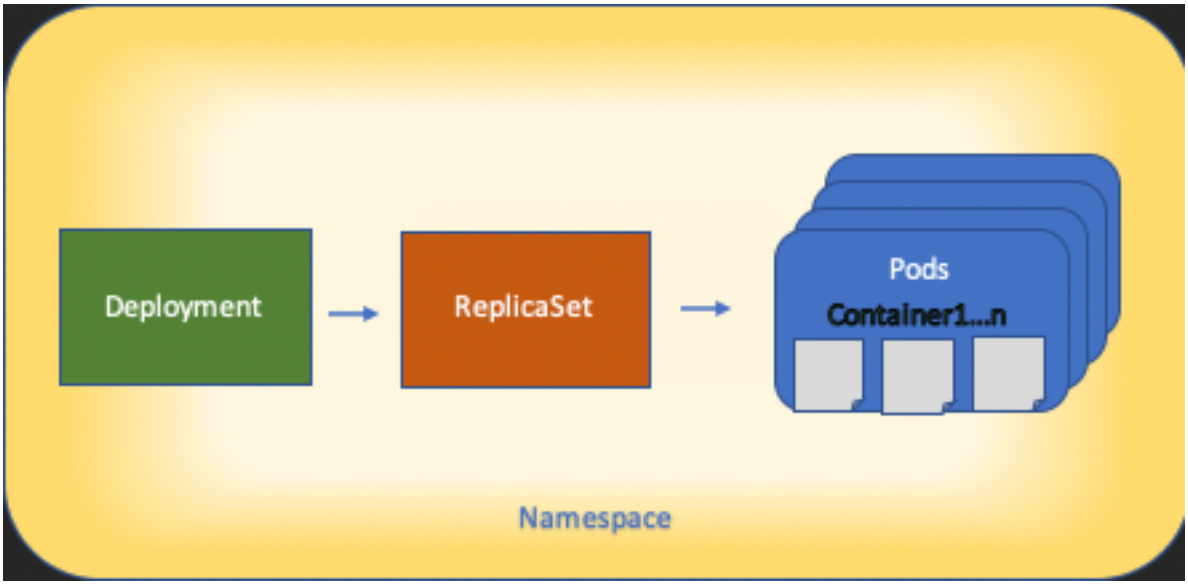
Um Manifesto é um arquivo formatado por YAML que descreve um recurso a ser implantado pelo Cluster. O recurso pode ser qualquer um dos descritos anteriormente ou outros disponíveis para os usuários.

O aplicativo pode ser acessado externamente com um ou mais serviços. O Kubernetes inclui uma opção de Balanceador de Carga para fazer isso.

Kubernetes também oferece uma maneira de isolar diferentes recursos com o conceito de Namespaces. O ND usa Namespaces para identificar exclusivamente diferentes Aplicativos e Serviços de Cluster. Quando os comandos CLI são executados, sempre especifique o Namespace.

Embora um profundo conhecimento de Kubernetes não seja necessário para resolver problemas de ND ou NDO, um entendimento básico da arquitetura Kubernetes é necessário para identificar corretamente os recursos com problemas ou que precisam de atenção.

Os fundamentos da arquitetura de recursos do Kubernetes são mostrados neste diagrama:



Éimportante lembrar como cada tipo de recurso interage com os outros e desempenha um papel importante no processo de revisão e solução de problemas.

Visão geral do NDO com comandos do Kubernetes

Login de acesso CLI

Para o acesso CLI por SSH a NDO, o comando **admin-user** senha é necessária. No entanto, em vez disso, usamos o **rescue-user** senha. Curtir em:

```
ssh rescue-user@ND-mgmt-IP
rescue-user@XX.XX.XX.XX's password:
[rescue-user@MxNDsh01 ~]$ pwd
/home/rescue-user
[rescue-user@MxNDsh01 ~]$
```

Esse é o modo padrão e o usuário para acesso CLI, e a maioria das informações está disponível para visualização.

Revisão de Namespaces NDO

Esse conceito K8s permite o isolamento de diferentes recursos no cluster. O próximo comando pode ser usado para revisar os diferentes Namespaces implantados:

```
[rescue-user@MxNDsh01 ~]$ kubectl get namespace
NAME                STATUS   AGE
authy               Active   177d
authy-oidc          Active   177d
cisco-appcenter     Active   177d
cisco-intersightdc  Active   177d
cisco-mso           Active   176d
cisco-nir           Active   22d
clicks              Active   177d
confd               Active   177d
default             Active   177d
elasticsearch       Active   22d
eventmgr            Active   177d
firmwared           Active   177d
installer           Active   177d
kafka               Active   177d
kube-node-lease     Active   177d
kube-public         Active   177d
kube-system         Active   177d
kubese              Active   177d
maw                 Active   177d
mond                Active   177d
mongodb             Active   177d
nodemgr             Active   177d
ns                  Active   177d
rescue-user         Active   177d
securitymgr         Active   177d
sm                  Active   177d
statscollect        Active   177d
ts                  Active   177d
zk                  Active   177d
```

As entradas em negrito pertencem a Applications no NDO, enquanto as entidades que começam com o prefixo **kube** pertencem ao cluster Kubernetes. Cada namespace tem suas próprias implantações e pods independentes

A CLI do kubectl permite especificar um namespace com o comando **--namespace** , se um comando for executado sem ele, a CLI assumirá que o Namespace é **default** (Namespace para k8s):

```
[rescue-user@MxNDsh01 ~]$ kubectl get pod --namespace cisco-mso
NAME                                READY   STATUS    RESTARTS   AGE
auditservice-648cd4c6f8-b29hh       2/2     Running   0          44h
…

[rescue-user@MxNDsh01 ~]$ kubectl get pod
No resources found in default namespace.
```

A CLI do kubectl permite diferentes tipos de formatos de saída, como yaml, JSON ou uma tabela personalizada. Isso é obtido com o **-o** opção [format]. Por exemplo:

```
[rescue-user@MxNDsh01 ~]$ kubectl get namespace -o JSON

{
```

```
    "apiVersion": "v1",

    "items": [

        {

            "apiVersion": "v1",

            "kind": "Namespace",

            "metadata": {

                "annotations": {

                    "kubectl.kubernetes.io/last-applied-configuration":
"{\"apiVersion\":\"v1\",\"kind\":\"Namespace\",\"metadata\":{\"annotations\":{},\"labels\":{\"se
rviceType\":\"infra\"},\"name\":\"authy\"}}\n"

                },

                "creationTimestamp": "2022-03-28T21:52:07Z",

                "labels": {

                    "serviceType": "infra"

                },

                "name": "authy",

                "resourceVersion": "826",

                "selfLink": "/api/v1/namespaces/authy",

                "uid": "373e9d43-42b3-40b2-a981-973bdddccd8d"

            },

        }

    ],

    "kind": "List",

    "metadata": {

        "resourceVersion": "",

        "selfLink": ""

    }

}
```


A partir do texto anterior, a saída é um dicionário onde uma de suas chaves é chamada de **itens** e o valor é uma **lista** de dicionários onde cada **dicionário** é responsável por uma entrada de Namespace e seus atributos são valor de par de chave-valor no dicionário ou dicionários aninhados.

Isso é relevante porque o K8s fornece aos usuários a opção de selecionar jsonpath como a saída, isso permite operações complexas para uma matriz de dados JSON. Por exemplo, na saída anterior, se acessarmos o valor de **name** para Namespaces, precisamos acessar o valor da lista de

itens, depois o metadata dicionário e obter o valor da chave **name**. Isso pode ser feito com este comando:

```
[rescue-user@MxNDsh01 ~]$ kubectl get namespace -o=jsonpath='{.items[*].metadata.name}'

authy authy-oidc cisco-appcenter cisco-intersightdc cisco-mso cisco-nir clicks confd default
elasticsearch eventmgr firmwared installer kafka kube-node-lease kube-public kube-system kubese
maw mond mongodb nodemgr ns rescue-user securitymgr sm statscollect ts zk

[rescue-user@MxNDsh01 ~]$
```

A hierarquia descrita é usada para buscar as informações específicas necessárias. Basicamente, todos os itens são acessados no **items** com **itens[*]**, a chave metadata e **name** com **metadata.name**, a consulta pode incluir outros valores a serem exibidos.

O mesmo se aplica à opção de colunas personalizadas, que usam uma maneira semelhante para buscar as informações da matriz de dados. Por exemplo, se criarmos uma tabela com as informações sobre o **name** e o UID , podemos aplicar o comando:

```
[rescue-user@MxNDsh01 ~]$ kubectl get namespace -o custom-
columns=NAME:.metadata.name,UID:.metadata.uid

NAME                 UID

authy                373e9d43-42b3-40b2-a981-973bdddccd8d

authy-oidc           ba54f83d-e4cc-4dc3-9435-a877df02b51e

cisco-appcenter      46c4534e-96bc-4139-8a5d-1d9a3b6aefdc

cisco-intersightdc   bd91588b-2cf8-443d-935e-7bd0f93d7256

cisco-mso            d21d4d24-9cde-4169-91f3-8c303171a5fc

cisco-nir            1c4dba1e-f21b-4ef1-abcf-026dbe418928

clicks               e7f45f6c-965b-4bd0-bf35-cbbb38548362

confd                302aebac-602b-4a89-ac1d-1503464544f7

default              2a3c7efa-bba4-4216-bb1e-9e5b9f231de2

elasticsearch        fa0f18f6-95d9-4cdf-89db-2175a685a761
```

A saída requer um nome para cada coluna a ser exibida e, em seguida, atribui o valor para a saída. Neste exemplo, há duas colunas: **NAME** e **UID**. Estes valores pertencem a .metada.name e .metadata.uid respectivamente. Mais informações e exemplos estão disponíveis em:

Suporte a JSONPath

Colunas personalizadas

Revisão da implantação de NDO

Uma Implantação é um objeto K8s que fornece um espaço unido para gerenciar ReplicaSet e Pods. As implantações lidam com a distribuição de todos os pods que pertencem a um aplicativo e o número esperado de cópias de cada um.

A CLI do kubectl inclui um comando para verificar as implantações para qualquer Namespace fornecido:

```
[rescue-user@MxNDsh01 ~]$ kubectl get deployment -n cisco-mso

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE

auditservice          1/1     1            1           3d22h

backupservice         1/1     1            1           3d22h

cloudsecservice       1/1     1            1           3d22h

consistencyservice    1/1     1            1           3d22h

dcnmworker            1/1     1            1           3d22h

eeworker              1/1     1            1           3d22h

endpointservice       1/1     1            1           3d22h

executionservice      1/1     1            1           3d22h

fluentd               1/1     1            1           3d22h

importservice         1/1     1            1           3d22h

jobschedulerservice   1/1     1            1           3d22h

notifyservice         1/1     1            1           3d22h

pctagvnidservice      1/1     1            1           3d22h

platformservice       1/1     1            1           3d22h

platformservice2      1/1     1            1           3d22h

policyservice         1/1     1            1           3d22h

schemaservice         1/1     1            1           3d22h

sdaservice            1/1     1            1           3d22h

sdwanservice          1/1     1            1           3d22h

siteservice           1/1     1            1           3d22h

siteupgrade           1/1     1            1           3d22h

syncengine            1/1     1            1           3d22h

templateeng           1/1     1            1           3d22h

ui                    1/1     1            1           3d22h
```

```
userservice           1/1     1            1           3d22h
```

Podemos usar a mesma tabela personalizada com o uso de deployment em vez de **namespace** e o **-n** para ver as mesmas informações de antes. Isso ocorre porque a saída é estruturada de maneira semelhante.

```
[rescue-user@MxNDsh01 ~]$ kubectl get deployment -n cisco-mso -o custom-
columns=NAME:.metadata.name,UID:.metadata.uid

NAME                  UID

auditservice          6e38f646-7f62-45bc-add6-6e0f64fb14d4

backupservice         8da3edfc-7411-4599-8746-09feae75afee

cloudsecservice       80c91355-177e-4262-9763-0a881eb79382

consistencyservice    ae3e2d81-6f33-4f93-8ece-7959a3333168

dcnmworker            f56b8252-9153-46bf-af7b-18aa18a0bb97

eeworker              c53b644e-3d8e-4e74-a4f5-945882ed098f

endpointservice       5a7aa5a1-911d-4f31-9d38-e4451937d3b0

executionservice      3565e911-9f49-4c0c-b8b4-7c5a85bb0299

fluentd               c97ea063-f6d2-45d6-99e3-1255a12e7026

importservice         735d1440-11ac-41c2-afeb-9337c9e8e359

jobschedulerservice   e7b80ec5-cc28-40a6-a234-c43b399edbe3

notifyservice         75ddb357-00fb-4cd8-80a8-14931493cfb4

pctagvnidservice      ebf7f9cf-964e-46e5-a90a-6f3e1b762979

platformservice       579eaae0-792f-49a0-accc-d01cab8b2891

platformservice2      4af222c9-7267-423d-8f2d-a02e8a7a3c04

policyservice         d1e2fff0-251a-447f-bd0b-9e5752e9ff3e

schemaservice         a3fca8a3-842b-4c02-a7de-612f87102f5c

sdaservice            d895ae97-2324-400b-bf05-b3c5291f5d14

sdwanservice          a39b5c56-8650-4a4b-be28-5e2d67cae1a9

siteservice           dff5aae3-d78b-4467-9ee8-a6272ee9ca62

siteupgrade           70a206cc-4305-4dfe-b572-f55e0ef606cb

syncengine            e0f590bf-4265-4c33-b414-7710fe2f776b

templateeng           9719434c-2b46-41dd-b567-bdf14f048720

ui                    4f0b3e32-3e82-469b-9469-27e259c64970

userservice           73760e68-4be6-4201-959e-07e92cf9fbb3
```

Tenha em mente que o número de cópias exibidas é para a implantação, não o número de pods

para cada microsserviço.

Podemos usar a palavra-chave **describe** em vez de **get** para exibir informações mais detalhadas sobre um recurso, neste caso, a implantação do schemaservice:

```
[rescue-user@MxNDsh01 ~]$ kubectl describe deployment -n cisco-mso schemaservice

Name:               schemaservice

Namespace:          cisco-mso

CreationTimestamp:  Tue, 20 Sep 2022 02:04:58 +0000

Labels:             k8s-app=schemaservice

                    scaling.case.cncf.io=scale-service

Annotations:        deployment.kubernetes.io/revision: 1

                    kubectl.kubernetes.io/last-applied-configuration:

                      {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"c
reationTimestamp":null,"labels":{"k8s-app":"schemaservice","sca...

Selector:           k8s-app=schemaservice

Replicas:           1 desired | 1 updated | 1 total | 1 available | 0 unavailable

StrategyType:       Recreate

MinReadySeconds:    0

Pod Template:

 Labels:           cpu.resource.case.cncf.io/schemaservice=cpu-lg-service

                   k8s-app=schemaservice

                   memory.resource.case.cncf.io/schemaservice=mem-xlg-service

 Service Account:  cisco-mso-sa

 Init Containers:

  init-msc:

   Image:      cisco-mso/tools:3.7.1j

   Port:       <none>

   Host Port:  <none>

   Command:

     /check_mongo.sh

   Environment:  <none>

   Mounts:

     /secrets from infracerts (rw)
```

```
 Containers:

  schemaservice:

   Image:       cisco-mso/schemaservice:3.7.1j

   Ports:       8080/TCP, 8080/UDP

   Host Ports:  0/TCP, 0/UDP

   Command:

     /launchscala.sh

     schemaservice

   Liveness:  http-get http://:8080/api/v1/schemas/health delay=300s timeout=20s period=30s
#success=1 #failure=3

   Environment:

     JAVA_OPTS:  -XX:+IdleTuningGcOnIdle

   Mounts:

     /jwtsecrets from jwtsecrets (rw)

     /logs from logs (rw)

     /secrets from infracerts (rw)

  msc-schemaservice-ssl:

   Image:       cisco-mso/sslcontainer:3.7.1j

   Ports:       443/UDP, 443/TCP

   Host Ports:  0/UDP, 0/TCP

   Command:

     /wrapper.sh

   Environment:

     SERVICE_PORT:  8080

   Mounts:

     /logs from logs (rw)

     /secrets from infracerts (rw)

  schemaservice-leader-election:

   Image:      cisco-mso/tools:3.7.1j

   Port:       <none>

   Host Port:  <none>

   Command:
```

```
     /start_election.sh

   Environment:

     SERVICENAME:  schemaservice

   Mounts:

     /logs from logs (rw)

 Volumes:

  logs:

   Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same
namespace)

   ClaimName:  mso-logging

   ReadOnly:   false

  infracerts:

   Type:        Secret (a volume populated by a Secret)

   SecretName:  cisco-mso-secret-infra

   Optional:    false

  jwtsecrets:

   Type:        Secret (a volume populated by a Secret)

   SecretName:  cisco-mso-secret-jwt

   Optional:    false

Conditions:

 Type          Status  Reason

 ----          ------  ------

 Available     True    MinimumReplicasAvailable

 Progressing   True    NewReplicaSetAvailable

Events:         <none>

[rescue-user@MxNDsh01 ~]$
```

O **describe** também permite a inclusão do comando **--show-events=true** para mostrar qualquer evento relevante para a implantação.

Spoiler

Revisão do Conjunto de Réplicas (RS) de NDO

Spoiler

DISPONÍVEL SOMENTE PARA O USUÁRIO RAIZ

Um conjunto de réplicas (RS) é um objeto K8s com o objetivo de manter um número estável de pods de réplica. Este objeto também detecta quando um número não íntegro de réplicas é visto com uma sonda periódica para os Pods.

Os RS também são organizados em espaços para nomes.

```
[root@MxNDsh01 ~]#  kubectl get rs  -n cisco-mso
NAME                              DESIRED   CURRENT   READY   AGE
auditservice-648cd4c6f8           1         1         1       3d22h
backupservice-64b755b44c          1         1         1       3d22h
cloudsecservice-7df465576         1         1         1       3d22h
consistencyservice-c98955599      1         1         1       3d22h
dcnmworker-5d4d5cbb64             1         1         1       3d22h
eeworker-56f9fb9ddb               1         1         1       3d22h
endpointservice-7df9d5599c        1         1         1       3d22h
executionservice-58ff89595f       1         1         1       3d22h
fluentd-86785f89bd                1         1         1       3d22h
importservice-88bcc8547           1         1         1       3d22h
jobschedulerservice-5d4fdfd696    1         1         1       3d22h
notifyservice-75c988cfd4          1         1         1       3d22h
pctagvnidservice-644b755596       1         1         1       3d22h
platformservice-65cddb946f        1         1         1       3d22h
platformservice2-6796576659       1         1         1       3d22h
policyservice-545b9c7d9c          1         1         1       3d22h
schemaservice-7597ff4c5           1         1         1       3d22h
sdaservice-5f477dd8c7             1         1         1       3d22h
sdwanservice-6f87cd999d           1         1         1       3d22h
siteservice-86bb756585            1         1         1       3d22h
siteupgrade-7d578f9b6d            1         1         1       3d22h
syncengine-5b8bdd6b45             1         1         1       3d22h
templateeng-5cbf9fdc48            1         1         1       3d22h
ui-84588b7c96                     1         1         1       3d22h
userservice-87846f7c6             1         1         1       3d22h
```

O describe inclui as informações sobre o URL, a porta que a sonda usa e a periodicidade dos testes e do limite de falha.

```
[root@MxNDsh01 ~]#  kubectl describe rs  -n cisco-mso schemaservice-7597ff4c5
Name:           schemaservice-7597ff4c5
Namespace:      cisco-mso
Selector:       k8s-app=schemaservice,pod-template-hash=7597ff4c5
Labels:         cpu.resource.case.cncf.io/schemaservice=cpu-lg-service
                k8s-app=schemaservice
                memory.resource.case.cncf.io/schemaservice=mem-xlg-service
                pod-template-hash=7597ff4c5
Annotations:    deployment.kubernetes.io/desired-replicas: 1
                deployment.kubernetes.io/max-replicas: 1
                deployment.kubernetes.io/revision: 1
Controlled By:  Deployment/schemaservice
Replicas:       1 current / 1 desired
Pods Status:    1 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
 Labels:           cpu.resource.case.cncf.io/schemaservice=cpu-lg-service
                   k8s-app=schemaservice
                   memory.resource.case.cncf.io/schemaservice=mem-xlg-service
                   pod-template-hash=7597ff4c5
 Service Account:  cisco-mso-sa
 Init Containers:
  init-msc:
   Image:      cisco-mso/tools:3.7.1j
   Port:       <none>
   Host Port:  <none>
   Command:
     /check_mongo.sh
   Environment:  <none>
   Mounts:
     /secrets from infracerts (rw)
 Containers:
```

```
  schemaservice:

   Image:       cisco-mso/schemaservice:3.7.1j

   Ports:       8080/TCP, 8080/UDP

   Host Ports:  0/TCP, 0/UDP

   Command:

     /launchscala.sh

     schemaservice

    Liveness:  http-get http://:8080/api/v1/schemas/health delay=300s timeout=20s period=30s
#success=1 #failure=3

   Environment:

     JAVA_OPTS:  -XX:+IdleTuningGcOnIdle

   Mounts:

     /jwtsecrets from jwtsecrets (rw)

     /logs from logs (rw)

     /secrets from infracerts (rw)

  msc-schemaservice-ssl:

   Image:       cisco-mso/sslcontainer:3.7.1j

   Ports:       443/UDP, 443/TCP

   Host Ports:  0/UDP, 0/TCP

   Command:

     /wrapper.sh
```

Revisão de NDO Replica Set (RS) #### ISSO ESTÁ DISPONÍVEL SOMENTE PARA O USUÁRIO ROOT #### Um Conjunto de Réplicas (RS) é um objeto K8s com o objetivo de manter um número estável de Pods de réplica. Este objeto também detecta quando um número não íntegro de réplicas é visto com uma sonda periódica para os Pods. Os RS também são organizados em espaços para nomes. [root@MxNDsh01 ~]# kubectl get rs -n cisco-msoNAME DESIRED CURRENT READY AGEauditservice-648cd4c6f8 1 1 3d22hbackupservice-64b755b44c 1 1 1 3d22hcloudsecservice-7df465576 1 1 1 3d22hconsistencyservice-c98955599 1 1 1 3d22hdcnmworker-5d4ccd bb64 1 1 1 3d22heeworker-56f9fb9ddb 1 1 1 3d22hendpointservice-7df9d5599c 1 1 3d22hexecutionservice-58ff89595f 1 1 1 3d22hfluentd-86785f89bd 1 1 3d22himportservice-88bcc8547 1 1 1 3d22hd obschedulerservice-5d4fdfd696 1 1 1 3d22hnotifyservice-75c988cfd4 1 1 3d22hpctagvnidservice-644b755596 1 1 1 3d22hplatformservice-65cddb946f 1 1 3d22hplatformservice2-6796576659 1 1 1 3d22hpolicyservice-545b9c7c9c 1 1 1 3d22hschemaservice-7597ff4c5 1 1 1 3d22hsdaservice-5f477dd8c7 1 1 3d22hsdwanservice-6f87cd999d 1 1 1 3d22hsiteservice-86bb756585 1 1 3d22hsiteupgrade-7d578f9b6d 1 1 1 3d 22hsyncengine-5b8bdd6b45 1 1 1 3d22htemplateeng-5cbf9fdc48 1 1 1 3d22hui-84588b7c96 1 1 1 3d22huserservice-87846f7c6 1 1 1 3d22h A opção descrever inclui as informações sobre o URL, a porta que a sonda usa e a periodicidade dos testes e o limite de falhas. [root@MxNDsh01 ~]# kubectl describe rs -n cisco-mso schemaservice-7597ff4c5Name: schemaservice-7597ff4c5Namespace: cisco-msoSelector: k8s-app=schemaservice,pod-template-hash=7597ff4c5Labels:

cpu.resource.case.cncf.io/schemaservice=cpu-lg-service k8s-app=schemaservice memory.resource.case.cncf.io/schemaservice=mem-xlg-service pod-template-hash=7597ff4c5Anotações: deployment.kubernetes.io/desired-replicas: 1 deployment.kubernetes.io/max-replicas: 1: 1Controlado por: Deployment/schemaserviceReplicas: 1 atual / 1 desirablePods Status: 1 Running / 0 Waiting / 0 Succeeded / 0 FailedPod Template: Labels: deployment.kubernetes.io/revision k8s-app=schemaservice cpu.resource.case.cncf.io/schemaservice=cpu-lg-service pod-template-hash=7597ff4c5 Service Account: cisco-mso-sa Init Containers: init-msc: Image: cisco-mso/tools:3.7.1j Port: <none> Host Port: <none> Comando: memory.resource.case.cncf.io/schemaservice=mem-xlg-service Environment: <none> Mounts /segredos de infracerts (rw) Contêineres: schemaservice: Imagem: cisco-mso/schemaservice:3.7.1j Portas: 8080/TCP, 8080/UDP Portas de Host: 0/TCP, 0/UDP Comando: /check_mongo.sh schemaservice Litiveness: http-get /launchscala.sh delay=300s timeout=20s period=30s #success=1 #failure=3 Ambiente: JAVA_OPTS: -XX:+IdleTuningGcOn3 Montagens: /jwtsecrets de jwtsecrets (rw) /logs de logs (rw) /secrets de infracerts (rw) msc-schemaservice-ssl: Imagem: cisco-mso/sslcontainer:3.7.1j Portas: 443/UDP, 443/TCP Portas do Host: 0/UDP, 0/TCP Comando: http://:8080/api/v1/schemas/health /wrapper.sh

Revisão de NDO Pod

Um Pod é um grupo de contêineres intimamente relacionados que são executados no mesmo Namespace Linux (diferente do Namespace K8s) e no mesmo nó K8s. Este é o objeto mais atômico que K8s manipula, como ele não interage com contêineres. O aplicativo pode consistir em um único contêiner ou ser mais complexo com muitos contêineres. Com o próximo comando, podemos verificar os Pods de qualquer namespace:

```
[rescue-user@MxNDsh01 ~]$ kubectl get pod --namespace cisco-mso

NAME                                  READY   STATUS    RESTARTS   AGE

auditservice-648cd4c6f8-b29hh         2/2     Running   0          2d1h

backupservice-64b755b44c-vcpf9        2/2     Running   0          2d1h

cloudsecservice-7df465576-pwbh4       3/3     Running   0          2d1h

consistencyservice-c98955599-qlsx5    3/3     Running   0          2d1h

dcnmworker-5d4d5cbb64-qxbt8           2/2     Running   0          2d1h

eeworker-56f9fb9ddb-tjggb             2/2     Running   0          2d1h

endpointservice-7df9d5599c-rf9bw      2/2     Running   0          2d1h

executionservice-58ff89595f-xf8vz     2/2     Running   0          2d1h

fluentd-86785f89bd-q5wdp              1/1     Running   0          2d1h

importservice-88bcc8547-q4kr5         2/2     Running   0          2d1h

jobschedulerservice-5d4fdfd696-tbvqj  2/2     Running   0          2d1h

mongodb-0                             2/2     Running   0          2d1h

notifyservice-75c988cfd4-pkkfw        2/2     Running   0          2d1h

pctagvnidservice-644b755596-s4zjh     2/2     Running   0          2d1h
```

```
platformservice-65cddb946f-7mkzm       3/3     Running   0          2d1h

platformservice2-6796576659-x2t8f      4/4     Running   0          2d1h

policyservice-545b9c7d9c-m5pbf         2/2     Running   0          2d1h

schemaservice-7597ff4c5-w4x5d          3/3     Running   0          2d1h

sdaservice-5f477dd8c7-l5jn7            2/2     Running   0          2d1h

sdwanservice-6f87cd999d-6fjb8          3/3     Running   0          2d1h

siteservice-86bb756585-5n5vb           3/3     Running   0          2d1h

siteupgrade-7d578f9b6d-7kqkf           2/2     Running   0          2d1h

syncengine-5b8bdd6b45-2sr9w            2/2     Running   0          2d1h

templateeng-5cbf9fdc48-fqwd7           2/2     Running   0          2d1h

ui-84588b7c96-7rfvf                    1/1     Running   0          2d1h

userservice-87846f7c6-lzctd            2/2     Running   0          2d1h

[rescue-user@MxNDsh01 ~]$
```

O número visto na segunda coluna refere-se ao número de contêineres para cada Pod.

O **describe** também está disponível, que inclui informações detalhadas sobre os contêineres em cada Pod.

```
[rescue-user@MxNDsh01 ~]$ kubectl describe pod  -n cisco-mso schemaservice-7597ff4c5-w4x5d

Name:         schemaservice-7597ff4c5-w4x5d

Namespace:    cisco-mso

Priority:     0

Node:         mxndsh01/172.31.0.0

Start Time:   Tue, 20 Sep 2022 02:04:59 +0000

Labels:       cpu.resource.case.cncf.io/schemaservice=cpu-lg-service

              k8s-app=schemaservice

              memory.resource.case.cncf.io/schemaservice=mem-xlg-service

              pod-template-hash=7597ff4c5

Annotations:  k8s.v1.cni.cncf.io/networks-status:

                [{

                    "name": "default",

                    "interface": "eth0",

                    "ips": [

                        "172.17.248.16"
```

```
                    ],
                    "mac": "3e:a2:bd:ba:1c:38",
                    "dns": {}
                }]
              kubernetes.io/psp: infra-privilege
Status:       Running
IP:           172.17.248.16
IPs:
 IP:           172.17.248.16
Controlled By:  ReplicaSet/schemaservice-7597ff4c5
Init Containers:
 init-msc:
   Container ID:  cri-o://0c700f4e56a6c414510edcb62b779c7118fab9c1406fdac49e742136db4efbb8
   Image:         cisco-mso/tools:3.7.1j
   Image ID:      172.31.0.0:30012/cisco-
mso/tools@sha256:3ee91e069b9bda027d53425e0f1261a5b992dbe2e85290dfca67b6f366410425
   Port:          <none>
   Host Port:     <none>
   Command:
     /check_mongo.sh
   State:          Terminated
     Reason:       Completed
     Exit Code:    0
     Started:      Tue, 20 Sep 2022 02:05:39 +0000
     Finished:     Tue, 20 Sep 2022 02:06:24 +0000
   Ready:          True
   Restart Count:  0
   Environment:    <none>
   Mounts:
     /secrets from infracerts (rw)
     /var/run/secrets/kubernetes.io/serviceaccount from cisco-mso-sa-token-tn45l (ro)
Containers:
```

```
 schemaservice:

   Container ID:  cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac

   Image:         cisco-mso/schemaservice:3.7.1j

   Image ID:      172.31.0.0:30012/cisco-
mso/schemaservice@sha256:6d9fae07731cd2dcaf17c04742d2d4a7f9c82f1fc743fd836fe59801a21d985c

   Ports:         8080/TCP, 8080/UDP

   Host Ports:    0/TCP, 0/UDP

   Command:

     /launchscala.sh

     schemaservice

   State:          Running

     Started:      Tue, 20 Sep 2022 02:06:27 +0000

   Ready:          True

   Restart Count:  0

   Limits:

     cpu:     8

     memory:  30Gi

   Requests:

     cpu:     500m

     memory:  2Gi
```

As informações exibidas incluem a imagem do contêiner para cada contêiner e mostram o Tempo de execução do contêiner usado. Neste caso, o CRI-O (**cri-o**), versões anteriores do ND usadas para trabalhar com Docker, isso influencia como anexar a um recipiente.

Spoiler

Por exemplo, quando **cri-o** é usado e queremos nos conectar por meio de uma sessão interativa a um contêiner (por meio do exec -it opção) para o contêiner da saída anterior, mas em vez da saída docker , usamos o comando **crictl**:

```
schemaservice:

   Container ID:  cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac

   Image:         cisco-mso/schemaservice:3.7.1j
```

Usamos este comando:

```
[root@MxNDsh01 ~]# crictl exec -it
```

```
d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac bash

root@schemaservice-7597ff4c5-w4x5d:/#

root@schemaservice-7597ff4c5-w4x5d:/# whoami

root
```

Em versões posteriores do ND, a ID do contêiner a ser usada é diferente. Primeiro, precisamos usar o comando **crictl ps** para listar todos os contêineres executados em cada nó. Podemos filtrar o resultado conforme necessário.

```
[root@singleNode ~]# crictl ps| grep backup
a9bb161d67295 10.31.125.241:30012/cisco-
mso/sslcontainer@sha256:26581eebd0bd6f4378a5fe4a98973dbda417c1905689f71f229765621f0cee75 2 days
ago that run msc-backupservice-ssl 0 84b3c691cfc2b
4b26f67fc10cf 10.31.125.241:30012/cisco-
mso/backupservice@sha256:c21f4cdde696a5f2dfa7bb910b7278fc3fb4d46b02f42c3554f872ca8c87c061 2 days
ago Running backupservice 0 84b3c691cfc2b
[root@singleNode ~]#
```

Com o valor da primeira coluna, podemos acessar o tempo de execução do contêiner com o mesmo comando de antes:

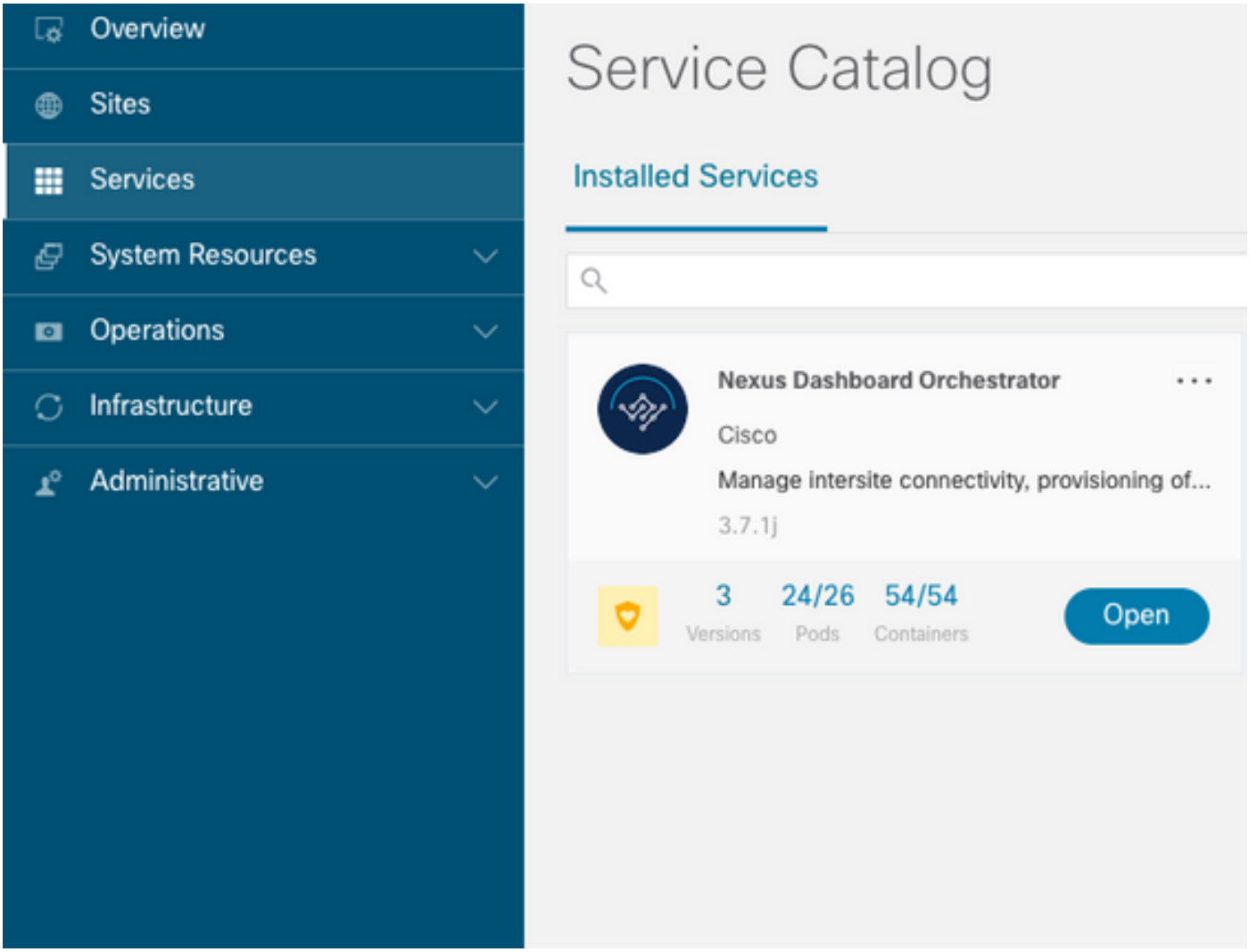
```
[root@singleNode ~]# crictl exec -it 4b26f67fc10cf bash
root@backupservice-8c699779f-j9jtr:/# pwd
/
```

Por exemplo, quando o cri-o é usado e queremos nos conectar por uma sessão interativa a um contêiner (através da opção exec -it) ao contêiner da saída anterior; mas, em vez do comando docker, usamos o comando criclt: schemaservice: Container ID: cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac Imagem: cisco-mso/schemaservice:3.7.1j Usamos este comando: [root@MxNDsh01 ~]# crictl exec -it d2287f8659dec6848c010 0b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac bashroot@schemaservice-7597ff4c5-w4x5d:/#root@schemaservice-7597ff4c5-w4x5d:/# whoamiroot Para versões posteriores do ND, a ID de contêiner a ser usada é diferente. Primeiro, precisamos usar o comando crictl ps para listar todos os contêineres que são executados em cada nó. Podemos filtrar o resultado conforme necessário. [root@singleNode ~]# crictl ps| grep backupa9bb161d67295 10.31.125.241:30012/cisco-mso/sslcontainer@sha256:26581eebd0bd6f4378a5fe4a98973dbda417c1905689f71f229765621f0cee75 Há 2 dias que executam msc-backupservice-ssl 0 84b3c691cfc2b4b26f67fc10cf 10.31.125.241:30012/cisco-mso/backupservice@sha256:c21f4cdde696a5f2dfa7bb910b7278fc3fb4d46b02f42c3554f872ca8c87c061 Há 2 dias executando o serviço de backup 0 84b3c691cfc2b[root@singleNode ~]# Com o valor da primeira coluna, podemos acessar o tempo de execução do contêiner com o mesmo comando de antes: [root@singleNode ~]# crictl exec -it 4b26f67fc10cf bashroot@backupservice-8c699779f-j9jtr:/# pwd/

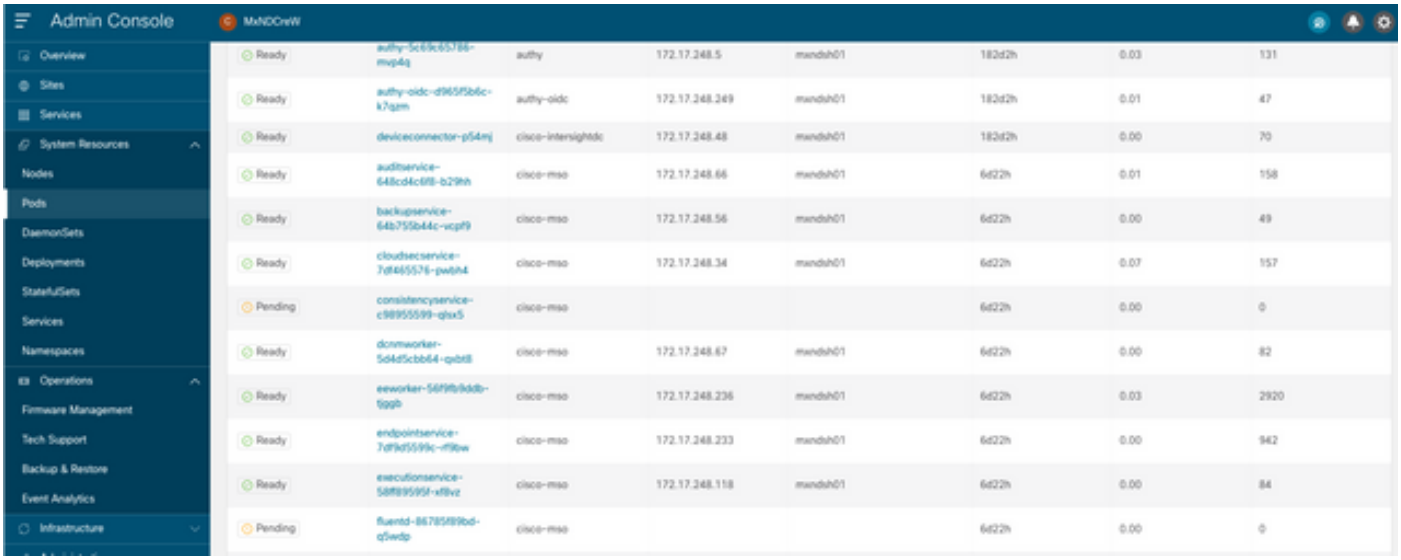
O Pod do caso de uso não está íntegro

Podemos usar essas informações para solucionar o motivo pelo qual os Pods de uma implantação não estão íntegros. Para este exemplo, a versão do Nexus Dashboard é 2.2-1d e o aplicativo afetado é o Nexus Dashboard Orchestrator (NDO).

A GUI do NDO exibe um conjunto incompleto de pods da visualização Serviço. Neste caso, 24 de 26 pods.



Outra exibição disponível no **System Resources -> Pods** ver onde os pods mostram um status diferente de **Ready**.



Solução de problemas de CLI para pods não íntegros

Com o fato conhecido de que o Namespace é cisco-mso (embora, quando for feita a solução de problemas, seja o mesmo para outros aplicativos/namespaces), a visualização Pod exibe se há algum não íntegro:

```
[rescue-user@MxNDsh01 ~]$ kubectl get deployment -n cisco-mso
NAME READY UP-TO-DATE AVAILABLE AGE
auditservice 1/1 1 1 6d18h
backupservice 1/1 1 1 6d18h
cloudsecservice 1/1 1 1 6d18h
consistencyservice 0/1 1 0 6d18h <---
fluentd 0/1 1 0 6d18h <---
syncengine 1/1 1 1 6d18h
templateeng 1/1 1 1 6d18h
ui 1/1 1 1 6d18h
userservice 1/1 1 1 6d18h
```

Para este exemplo, nos concentramos nos pods de serviço de consistência. Na saída JSON, podemos obter as informações específicas dos campos de status, com o uso de jsonpath:

```
[rescue-user@MxNDsh01 ~]$ kubectl get deployment -n cisco-mso consistencyservice -o json
{
<--- OUTPUT OMITTED ---->
"status": {
"conditions": [
{
"message": "Deployment does not have minimum availability.",
"reason": "MinimumReplicasUnavailable",
},
{
"message": "ReplicaSet \"consistencyservice-c98955599\" has timed out progressing.",
"reason": "ProgressDeadlineExceeded",
}
],
}
}
[rescue-user@MxNDsh01 ~]$
```

Vemos o dicionário de **status** e dentro de uma lista chamada **condições** com dicionários como itens com as chaves **mensagem** e **valor**, a parte {"\n"} é criar uma nova linha no final:

```
[rescue-user@MxNDsh01 ~]$ kubectl get deployment -n cisco-mso consistencyservice -
o=jsonpath='{.status.conditions[*].message}{"\n"}'
Deployment does not have minimum availability. ReplicaSet "consistencyservice-c98955599" has
timed out progressing.
[rescue-user@MxNDsh01 ~]$
```

Esse comando mostra como verificar a partir do **get Pod** para o Namespace:

```
[rescue-user@MxNDsh01 ~]$ kubectl get pods -n cisco-mso
NAME READY STATUS RESTARTS AGE
consistencyservice-c98955599-qlsx5 0/3 Pending 0 6d19h
executionservice-58ff89595f-xf8vz 2/2 Running 0 6d19h
fluentd-86785f89bd-q5wdp 0/1 Pending 0 6d19h
importservice-88bcc8547-q4kr5 2/2 Running 0 6d19h
jobschedulerservice-5d4fdfd696-tbvqj 2/2 Running 0 6d19h
mongodb-0 2/2 Running 0 6d19h
```

Com a get pods , podemos obter o ID do Pod com problemas que devem corresponder ao da saída anterior. Neste exemplo **consistencyservice-c98955599-qlsx5**.

O formato de saída JSON também fornece como verificar informações específicas, a partir da saída fornecida.

```
[rescue-user@MxNDsh01 ~]$ kubectl get pods -n cisco-mso consistencyservice-c98955599-qlsx5 -o
json
{
<--- OUTPUT OMITTED ---->
"spec": {
<--- OUTPUT OMITTED ---->
"containers": [
{
<--- OUTPUT OMITTED ---->
"resources": {
"limits": {
"cpu": "8",
"memory": "8Gi"
},
"requests": {
"cpu": "500m",
"memory": "1Gi"
}
},
<--- OUTPUT OMITTED ---->
"status": {
"conditions": [
{
"lastProbeTime": null,
"lastTransitionTime": "2022-09-20T02:05:01Z",
"message": "0/1 nodes are available: 1 Insufficient cpu.",
"reason": "Unschedulable",
"status": "False",
"type": "PodScheduled"
}
],
"phase": "Pending",
"qosClass": "Burstable"
}
}
[rescue-user@MxNDsh01 ~]$
```

A saída JSON deve incluir informações sobre o status no atributo com o mesmo nome. A mensagem inclui informações sobre o motivo.

```
[rescue-user@MxNDsh01 ~]$ kubectl get pods -n cisco-mso consistencyservice-c98955599-qlsx5 -
o=jsonpath='{.status}{"\n"}'
map[conditions:[map[lastProbeTime:<nil> lastTransitionTime:2022-09-20T02:05:01Z message:0/1
nodes are available: 1 Insufficient cpu. reason:Unschedulable status:False type:PodScheduled]]
phase:Pending qosClass:Burstable]
[rescue-user@MxNDsh01 ~]$
```

Podemos acessar informações sobre o status e os requisitos para os pods:

```
[rescue-user@MxNDsh01 ~]$ kubectl get pods -n cisco-mso consistencyservice-c98955599-qlsx5 -
o=jsonpath='{.spec.containers[*].resources.requests}{"\n"}'
map[cpu:500m memory:1Gi]
```

Neste ponto, é importante mencionar como o valor é calculado. Neste exemplo, a cpu **500m** se refere a **500 milicores**, e a **1G** na memória é para GB.

O **Describe** opção para o nó mostra o recurso disponível para cada trabalhador do K8s no cluster (host ou VM):

```
[rescue-user@MxNDsh01 ~]$ kubectl describe nodes | egrep -A 6 "Allocat"
Allocatable:
cpu: 13
ephemeral-storage: 4060864Ki
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 57315716Ki
pods: 110
--
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource Requests Limits
-------- -------- ------
cpu 13 (100%) 174950m (1345%)
memory 28518Mi (50%) 354404Mi (633%)
ephemeral-storage 0 (0%) 0 (0%)
>[rescue-user@MxNDsh01 ~]$
```

A seção **Alocável** mostra o total de recursos em CPU, memória e armazenamento disponíveis para cada nó. A seção **Alocado** mostra os recursos que já estão em uso. O valor **13** para CPU refere-se a **13** núcleos **ou** 13.000 (13K) **milinúcleos**.

Para este exemplo, o nó está **com excesso de assinaturas**, o que explica por que o Pod não pode iniciar. Depois de eliminarmos o ND com a exclusão de APLICATIVOS ND ou a adição de recursos de VM.

O Cluster tenta constantemente implantar quaisquer políticas pendentes, de modo que se os recursos estiverem livres, os Pods podem ser implantados.

```
[rescue-user@MxNDsh01 ~]$ kubectl get deployment -n cisco-mso
NAME READY UP-TO-DATE AVAILABLE AGE
auditservice 1/1 1 1 8d
backupservice 1/1 1 1 8d
cloudsecservice 1/1 1 1 8d
consistencyservice 1/1 1 1 8d
dcnmworker 1/1 1 1 8d
eeworker 1/1 1 1 8d
endpointservice 1/1 1 1 8d
executionservice 1/1 1 1 8d
fluentd 1/1 1 1 8d
importservice 1/1 1 1 8d
jobschedulerservice 1/1 1 1 8d
notifyservice 1/1 1 1 8d
pctagvnidservice 1/1 1 1 8d
platformservice 1/1 1 1 8d
platformservice2 1/1 1 1 8d
policyservice 1/1 1 1 8d
schemaservice 1/1 1 1 8d
sdaservice 1/1 1 1 8d
sdwanservice 1/1 1 1 8d
siteservice 1/1 1 1 8d
siteupgrade 1/1 1 1 8d
syncengine 1/1 1 1 8d
templateeng 1/1 1 1 8d
ui 1/1 1 1 8d
userservice 1/1 1 1 8d
```

Com o comando usado para verificação de recursos, confirmamos que o cluster tem recursos disponíveis para a CPU:

```
[rescue-user@MxNDsh01 ~]$ kubectl describe nodes | egrep -A 6 "Allocat"
Allocatable:
cpu: 13
ephemeral-storage: 4060864Ki
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 57315716Ki
pods: 110
--
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource Requests Limits
-------- -------- ------
cpu 12500m (96%) 182950m (1407%)
memory 29386Mi (52%) 365668Mi (653%)
ephemeral-storage 0 (0%) 0 (0%)
[rescue-user@MxNDsh01 ~]$
```

Os detalhes da implantação incluem uma mensagem com informações sobre as condições atuais para Pods:

```
[rescue-user@MxNDsh01 ~]$ kubectl get deployment -n cisco-mso consistencyservice -
o=jsonpath='{.status.conditions[*]}{"\n"}'
map[lastTransitionTime:2022-09-27T19:07:13Z lastUpdateTime:2022-09-27T19:07:13Z
message:Deployment has minimum availability. reason:MinimumReplicasAvailable status:True
type:Available] map[lastTransitionTime:2022-09-27T19:07:13Z lastUpdateTime:2022-09-27T19:07:13Z
message:ReplicaSet "consistencyservice-c98955599" has successfully progressed.
reason:NewReplicaSetAvailable status:True type:Progressing]
[rescue-user@MxNDsh01 ~]$
```

Spoiler

Como executar comandos de depuração de rede de dentro de um contêiner

Como os contêineres incluem apenas as bibliotecas e dependências mínimas específicas para o Pod, a maioria das ferramentas de depuração de rede (ping, ip route e ip addr) não está disponível dentro do próprio contêiner.

Esses comandos são muito úteis quando há necessidade de solucionar problemas de rede para um serviço (entre nós ND) ou conexão em direção às Apics porque vários microsserviços precisam se comunicar com os controladores com a interface de dados (**bond0** ou **bond0br**).

O **nsenter** (somente usuário raiz) permite executar comandos de rede a partir do nó ND como está dentro do contêiner. Para isso, localize o ID do processo (PID) do contêiner que desejamos depurar. Isso é realizado com o ID do pod K8s em relação às informações locais do Container Runtime, como Docker para versões herdadas e **cri-o** para os mais novos como padrão.

Inspecione a ID do Pod Kubernetes (K8s)

Na lista de Pods dentro do Namespace cisco-mso, podemos selecionar o contêiner para solucionar o problema:

```
[root@MxNDsh01 ~]# kubectl get pod -n cisco-mso
NAME READY STATUS RESTARTS AGE
consistencyservice-569bdf5969-xkwpg 3/3 Running 0 9h
eeworker-65dc5dd849-485tq 2/2 Running 0 163m
```

```
endpointservice-5db6f57884-hkf5g 2/2 Running 0 9h
executionservice-6c4894d4f7-p8fzk 2/2 Running 0 9h
siteservice-64dfcdf658-lvbr4 3/3 Running 0 9h
siteupgrade-68bcf987cc-ttn7h 2/2 Running 0 9h
```

Os pods devem ser executados no mesmo nó K8s. Para ambientes de produção, podemos adicionar o **-o wide** no final para descobrir o nó que cada Pod executa. Com o ID do pod K8s (em negrito no exemplo de saída anterior) podemos verificar o processo (PID) atribuído pelo tempo de execução do contêiner.

Como inspecionar o PID a partir do tempo de execução do contêiner

O novo Container Runtime padrão é CRI-O para Kubernetes. Então o documento vem depois daquela regra para os comandos. O ID do processo (PID) atribuído pelo CRI-O pode ser exclusivo no nó K8s, que pode ser descoberto com o crictl utilitário.

O **ps** revela a ID fornecida pelo CRI-O para cada contêiner que constrói o Pod, dois para o exemplo de sitesevice:

```
[root@MxNDsh01 ~]# crictl ps |grep siteservice
fb560763b06f2 172.31.0.0:30012/cisco-
mso/sslcontainer@sha256:2d788fa493c885ba8c9e5944596b864d090d9051b0eab82123ee4d19596279c9 10
hours ago Running msc-siteservice2-ssl 0 074727b4e9f51
ad2d42aae1ad9 1d0195292f7fcc62f38529e135a1315c358067004a086cfed7e059986ce615b0 10 hours ago
Running siteservice-leader-election 0 074727b4e9f51
29b0b6d41d1e3 172.31.0.0:30012/cisco-
mso/siteservice@sha256:80a2335bcd5366952b4d60a275b20c70de0bb65a47bf8ae6d988f07b1e0bf494 10 hours
ago Running siteservice 0 074727b4e9f51
[root@MxNDsh01 ~]#
```

Com essas informações, podemos usar o comando inspect CRIO-ID para ver a PID real fornecida a cada contêiner. Essas informações são necessárias para a nsenter comando:

```
[root@MxNDsh01 ~]# crictl inspect fb560763b06f2| grep -i pid
"pid": 239563,
"pids": {
"type": "pid"
```

Como usar o nsenter para executar comandos de depuração de rede dentro de um contêiner

Com o PID da saída acima, podemos usar como destino na próxima sintaxe de comando:

```
nsenter --target <PID> --net <NETWORK COMMAND>
```

O **--net** permite executar comandos nos Namespaces de rede, de modo que o número de comandos disponíveis é limitado.

Por exemplo:

```
[root@MxNDsh01 ~]# nsenter --target 239563 --net ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
inet 172.17.248.146 netmask 255.255.0.0 broadcast 0.0.0.0
inet6 fe80::984f:32ff:fe72:7bfb prefixlen 64 scopeid 0x20<link>
ether 9a:4f:32:72:7b:fb txqueuelen 0 (Ethernet)
RX packets 916346 bytes 271080553 (258.5 MiB)
```

```
RX errors 0 dropped 183 overruns 0 frame 0
TX packets 828016 bytes 307255950 (293.0 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 42289 bytes 14186082 (13.5 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 42289 bytes 14186082 (13.5 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

O ping também está disponível e testa a conectividade do contêiner para o exterior, em vez de apenas o nó K8s.

```
[root@MxNDsh01 ~]# nsenter --target 239563 --net wget --no-check-certificate
https://1xx.2xx.3xx.4xx
--2023-01-24 23:46:04-- https://1xx.2xx.3xx.4xx/
Connecting to 1xx.2xx.3xx.4xx:443... connected.
WARNING: cannot verify 1xx.2xx.3xx.4xx's certificate, issued by ‘/C=US/ST=CA/O=Cisco
System/CN=APIC’:
Unable to locally verify the issuer's authority.
WARNING: certificate common name ‘APIC’ doesn't match requested host name ‘1xx.2xx.3xx.4xx’.
HTTP request sent, awaiting response... 200 OK
Length: 3251 (3.2K) [text/html]
Saving to: ‘index.html’

100%[=============================================================================================
=====================================================>] 3,251 --.-K/s in 0s

2023-01-24 23:46:04 (548 MB/s) - ‘index.html’ saved [3251/3251]
```

Como executar comandos de depuração de rede de dentro de um contêiner Como os contêineres incluem apenas as bibliotecas e dependências mínimas específicas para o Pod, a maioria das ferramentas de depuração de rede (ping, rota ip e endereço ip) não está disponível dentro do próprio contêiner. Esses comandos são muito úteis quando há necessidade de solucionar problemas de rede para um serviço (entre nós ND) ou conexão em direção às Apics porque vários microsserviços precisam se comunicar com os controladores com a interface de dados (bond0 ou bond0br). O utilitário nsenter (somente usuário raiz) permite executar comandos de rede a partir do nó ND, pois ele está dentro do contêiner. Para isso, localize o ID do processo (PID) do contêiner que desejamos depurar. Isso é realizado com o ID do K8s do Pod em relação às informações locais do Container Runtime, como Docker para versões herdadas e cri-o para versões mais recentes como padrão. Inspecione o ID do Pod Kubernetes (K8s) Na lista de Pods dentro do Namespace cisco-mso, podemos selecionar o contêiner para solução de problemas: [root@MxNDsh01 ~]# kubectl get pod -n cisco-msoNAME READY STATUS RESTARTS AGEconsistencyservice-569bdf5969-xkwpg 3/3 Running 0 9heeworker-65dc5dd849-485tq 2/2 Running 0 1 63mendpointservice-5db6f57884-hkf5g 2/2 Executando 0 9hexecutionservice-6c4894d4f7-p8fzk 2/2 Executando 0 9hsiteservice-64dfcdf658-lvbr4 3/3 Executando 0 9hsiteupgrade-68bcf987cc-ttn7h 2/2 Executando 0 9h Os Pods devem ser executados no mesmo nó K8s. Para ambientes de produção, podemos adicionar a opção -o wide no final para descobrir o nó que cada Pod executa. Com o ID do pod K8s (em negrito no exemplo de saída anterior) podemos verificar o processo (PID) atribuído pelo tempo de execução do contêiner. Como inspecionar o PID do Container Runtime O novo Container Runtime padrão é CRI-O para Kubernetes. Então o documento vem depois daquela regra para os comandos. O ID do processo (PID) atribuído pelo CRI-O pode ser exclusivo no nó K8s, que pode ser descoberto com o utilitário crictl. A opção ps revela o ID fornecido pelo CRI-O para cada contêiner que constrói o Pod, dois para o exemplo de serviço de site: [root@MxNDsh01 ~]# crictl ps |grep

siteservicefb560763b06f2 172.31.0.0:30012/cisco-mso/sslcontainer@sha256:2d788fa493c885ba8c9e5944596b864d090d9051b0eab82123ee4d19596279c9 Há 10 horas Executando msc-siteservice2-ssl 0 074727b4e9f51ad2d42aae1ad91d0195292f7fcc62f38529e135a1315c358067004a086cfed7e059986ce615b0 Há 10 horas Executando siteservice-service -election 0 074727b4e9f5129b0b6d41d1e3 172.31.0.0:30012/cisco-mso/siteservice@sha256:80a2335bcd5366952b4d60a275b20c70de0bb65a47bf8ae6d988f07b1e0bf494 Há 10 horas Executando o siteservice 0 074727b4e9f51[root@MxNDsh01 ~]# Com essa informação, podemos usar a opção inspect CRIO-ID para ver o PID real fornecido a cada contêiner. Essas informações são necessárias para o comando nsenter: [root@MxNDsh01 ~]# crictl inspect fb560763b06f2| grep -i pid"pid": 239563,"pids": {"type": "pid" How to Use nsenter to Run Network Debug Commands Inside a Container Com o PID da saída acima, podemos usar como destino na próxima sintaxe de comando: nsenter —target <PID> —net <NETWORK COMMAND> A opção —net nos permite executar comandos nos Namespaces de rede, portanto, o número de comandos disponíveis é limitado. Por exemplo: [root@MxNDsh01 ~]# nsenter —target 239563 —net ifconfigeth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450inet 172.17.248.146 netmask 255.255.0.0 broadcast 0.0.0.0inet6 fe80::984f:32ff:fe72:7bfffb prefixlen 64 scopeid 0x20<link>ether 9a:4f:32:72:7b:fb txqueuelen 0 (Ethernet)Pacotes RX 916346 bytes 271080553 (258,5 MiB)Erros RX 0 drop 183 overruns 0 frame 0TX pacotes 828016 bytes 307255950 (293,0 MiB)Erros TX 0 drops 0 overruns 0 carrier 0 colisões 0lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536 inet 127.0.0.1 máscara de rede 255.0.0.0inet6 ::1 prefixlen 128 scopeid 0x10<host>loop txqueuelen 1000 (Loopback local)Pacotes RX 42289 bytes 14186082 (13.5 MiB)Erros RX 0 descartados 0 overruns 0 pacotes frame 0TX 42289 bytes 14186082 (13.5 MiB)Erros TX 0 descartados 0 overruns 0 colisões 0 portadora O ping também está disponível e testa a conectividade do contêiner para fora, em vez de apenas o nó K8s. [root@MxNDsh01 ~]# nsenter —target 239563 —net wget —no-check-certificate https://1xx.2xx.3xx.4xx--2023-01-24 23:46:04— https://1xx.2xx.3xx.4xx/Connecting to 1xx.2xx.3xx.4xx:443... connected.AVISO: não é possível verificar o certificado de 1xx.2xx.3xx.4xx, emitido por ‘/C=US/ST=CA/O=Cisco System/CN=APIC’:Não é possível verificar localmente a autoridade do emissor.AVISO: o nome comum do certificado ‘APIC’ não corresponde ao nome de host solicitado ‘1xx.2xx 3xx.4xx".Solicitação HTTP enviada, aguardando resposta... 200 OKLontendimento: 3251 (3.2K) [texto/html]Salvando em: ‘index.html’100%[==========>] 3,251 —.-K/s em 0s2023-01-24 23:46:04 (548 MB/s) - ‘index.html’ salvo [3251/325 1]

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.