对NDO资源进行故障排除和检查

目录

简介

本文档介绍如何使用kubectl和容器运行时CLI对NDO进行审核和故障排除。

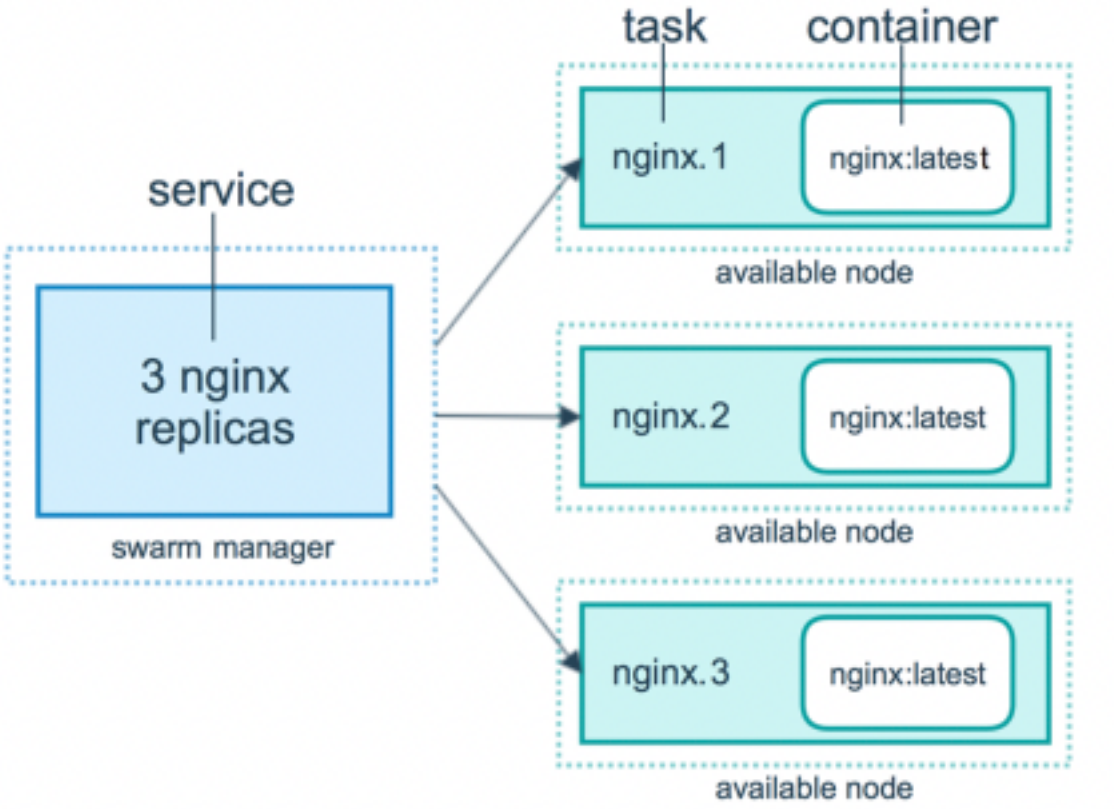
NDO快速入门

Cisco Nexus Dashboard Orchestrator(NDO)是一个交换矩阵管理工具，允许用户管理各种交换矩阵，包括思科®以应用为中心的基础设施(Cisco ACI®)站点、思科云ACI站点和思科Nexus控制面板交换矩阵控制器(NDFC)站点，每个站点都由其自己的控制器（APIC集群、NDFC集群或公共云中的云APIC实例）进行管理。

NDO通过单一管理平台跨多个数据中心提供一致的网络和策略协调、可扩展性和灾难恢复。

在早期，MSC（多站点控制器）部署为带有VMWare开放式虚拟设备(OVA)的三节点集群，允许客户初始化Docker Swarm集群和MSC服务。此Swarm群集将MSC微服务作为Docker容器和服务进行管理。

此图片显示了有关Docker Swarm如何管理作为同一容器副本的微服务以实现高可用性的简化视图。



Docker群负责维护MSC架构中每个微服务的预期副本数量。从Docker Swarm的角度来看，多站点控制器是唯一需要协调的容器部署。

Nexus Dashboard(ND)是多个数据中心站点的中央管理控制台和托管思科数据中心运营服务（包括Nexus Insight和MSC 3.3版及更高版本）的通用平台，并将名称更改为Nexus Dashboard Orchestrator(NDO)。

虽然构成MSC架构的大多数微服务仍然相同，但NDO部署在Kubernetes(K8s)集群中，而不是Docker Swarm集群中。这允许ND协调多个应用或部署，而不是仅协调一个应用或部署。

具有NDO崩溃课程的Kubernetes

Kubernetes是一个开源系统，用于自动化容器化应用的部署、可扩展性和管理。作为Docker，Kubernetes与容器技术合作，但不与Docker合作。这意味着Kubernetes支持其他容器平台(Rkt、PodMan)。

Swarm和Kubernetes之间的一个关键区别是，后者并不直接与容器配合使用，而是与容器共置组的概念配合使用，称为Pod。

Pod中的容器必须在同一节点中运行。一组Pod称为部署。Kubernetes部署可以描述整个应用程序。

Kubernetes还允许用户确保一定数量的资源可用于任何给定的应用程序。这可以通过使用复制控制器来完成，以确保Pod的数量与应用程序清单一致。

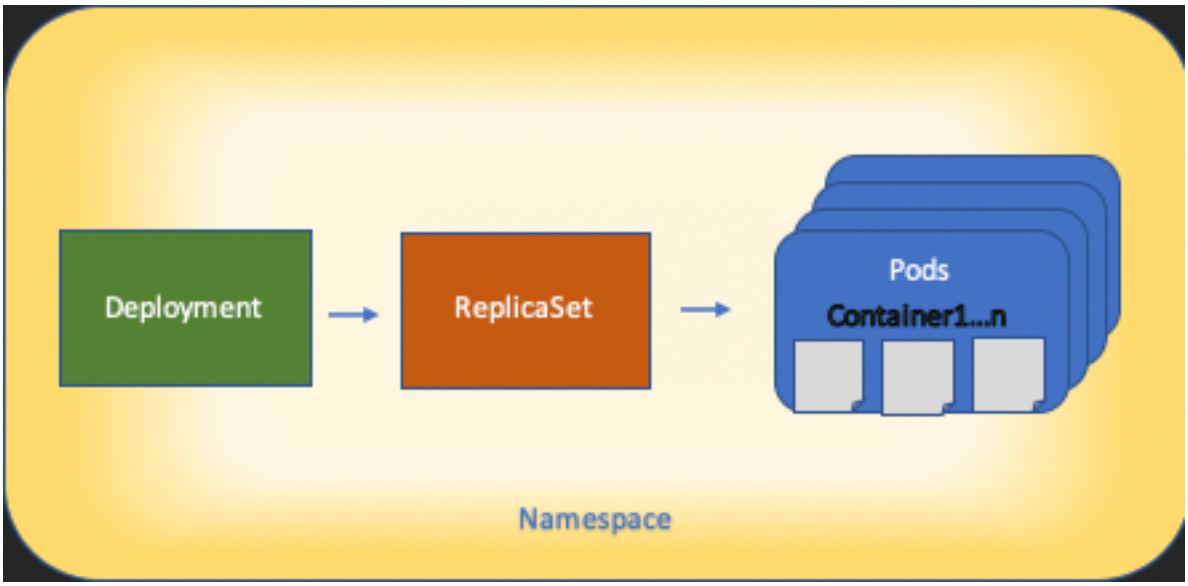
清单是一个YAML格式的文件，用于描述将由集群部署的资源。资源可以是之前描述的任何资源或可供用户使用的其他资源。

可通过一项或多项服务在外部访问应用。Kubernetes包含负载均衡器选项以实现此目的。

Kubernetes还提供了一种通过名称空间的概念隔离不同资源的方法。ND使用命名空间来唯一地标识不同的应用程序和群集服务。运行CLI命令时，请始终指定Namespace。

虽然对ND或NDO进行故障排除时不需要深入了解Kubernetes，但需要对Kubernetes架构有基本了解，才能正确识别有问题或需要注意的资源。

Kubernetes资源体系结构的基本信息如下图所示：



记住各种资源与其他资源的交互方式非常重要，它在审核和故障排除过程中起着重要作用。

使用Kubernetes命令的NDO概述

CLI访问登录

对于通过SSH访问NDO的CLI， **admin-user** 需要密码。但是，我们使用 **rescue-user** 密码.例如：

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

这是CLI访问的默认模式和用户，大部分信息都可供查看。

NDO命名空间评审

此K8概念允许跨集群隔离不同的资源。下一个命令可用于查看部署的不同命名空间：

```
[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
```

粗体条目属于NDO中的应用程序，而以前缀**kube**开头的实体属于Kubernetes群集。每个名称空间都有自己的独立部署和Pod

kubectl CLI允许使用 **--namespace** 选项，如果运行命令时没有该选项，则CLI会假定命名空间为 **default** （k8的命名空间）：

```
[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.
```

kubectl CLI允许输出采用不同类型的格式，例如yaml、JSON或自定义表。这是通过 **-o** [format]选项。例如：

```
[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": ""

    }

}
```


从上一文本中，输出是一个词典，其中它的键之一被称作项，值是一个词典列表，其中每个词典都包含Namespace条目，其属性是词典或嵌套词典中的键 — 值对值。

这是相关的，因为K8为用户提供了选择jsonpath作为输出的选项，这允许对JSON数据数组执行复杂的操作。例如，从上一个输出中，如果我们访问 **name** 对于命名空间，我们需要访问项目列表的值，然后 metadata 并获取密钥的值 **name**.这可以通过以下命令完成：

```
[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 ~]$
```

所述的层次结构用于获取所需的特定信息。基本上，所有项目都可在 **items** 列出项**目[*]**，然后按键 metadata 和 **name** 对于**metadata.name**，查询可以包含要显示的其他值。

这同样适用于自定义列的选项，该选项使用类似的方法从数据数组获取信息。例如，如果我们创建一个表，其中包含关于 **name** 和 UID 值，我们可以应用以下命令：

```
[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
```

输出要求显示每列的名称，然后为输出分配值。在本示例中，有两列： **NAME** 和 **UID**.这些值属于 .metada.name 和 .metadata.uid 的报价。有关更多信息和示例，请访问：

JSONPath支持

自定义列

NDO部署审核

部署是一个K8s对象，提供用于管理ReplicaSet和Pod的连接空间。部署处理属于某个应用的所有Pod的展开以及每个应用的预期副本数量。

kubectl CLI包含用于检查任何给定命名空间的部署的命令：

```
[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
```

我们可以使用相同的自定义表， deployment 而非 **namespace** 和 **-n** 选项来查看与之前相同的信息。 这是因为输出采用类似结构。

```
[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
```

请记住，显示的副本数量用于部署，而不是每个微服务的Pod数量。

我们可以使用关键字 **describe** 而非 **get** 要显示有关资源的更多详细信息，本例中为schemaservice部署：

```
[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 ~]$
```

此 **describe** 命令还允许包含 **--show-events=true** 选项以显示部署的任何相关事件。

Spoiler

NDO副本集(RS)检查

Spoiler

####仅对根用户#####可用

副本集(RS)是一个K8s对象，目标是维护稳定数量的副本Pod。此对象还检测通过定期探测到Pod发现不正常数量的复制副本的时间。

RS也以命名空间组织。

```
[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
```

此 describe 选项包括有关URL、探测使用的端口以及测试和故障阈值的周期性信息。

```
[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
```

NDO副本集(RS)审阅#####这仅适用于根用户#####副本集(RS)是一个K8s对象，目标是维护稳定数量的副本Pod。此对象还检测通过定期探测到Pod发现不正常数量的复制副本的时间。 RS也以命名空间组织。[root@MxNDsh01 ~]# kubectl get rs -n cisco-msoNAME DESIRED CURRENT READY AGEauditservice-648cd4c6f8 1 1 1 3d22hbackupservice-64b755b44c 1 1 1 3d22hcloudsecservice-7df465576 1 1 1 1 3d22hconsistencyservice-c98955599 1 1 1 3d22hdcnmworker-5d5cbb64 1 1 1 3d22heeworker-56f fb9ddb 1 1 1 3d22hendpointservice-7df9d5599c 1 1 1 3d22hexecutionservice-58ff89595f 1 1 3d22hfluentd-86785f89bd 1 1 1 3d22himportservice-88bcc8547 1 1 1 3d22hjobschedulerservice-5d4fdfd696 1 1 1 3d22hnotifyservice-75c98 cfd 1 1 3d22hpctagvnidservice-644b755596 1 1 1 3d22platformservice-65cddb946f 1 1 3d22platformservice2-6796576659 1 1 1 3d22hpolicyservice-545b9c7d9c 1 1 1 3d22hschemaservice-7597ff4c5 1 1 3d22hsdaservice-5f477dd8 c7 1 1 1 3d22hsdwanservice f87cd999d 1 1 3d22hsiteservice-86bb756585 1 1 3d22hsiteupgrade-7d578f9b6d 1 1 3d22hsyncengine-5b8bdd6b45 1 1 1 3d22htemplateeng-5cbf9fdc48 1 1 1 3d22hui-84588b7c96 1 1 3d22huserservice-87846f7c6 1 1 3d22h describe选项包括有关URL、探测使用的端口以及测试和故障阈值的周期性信息。[root@MxNDsh01 ~]# kubectl description rs -n cisco-mso schemaservice-7597ff4c5名称： schemaservice-7597ff4c5命名空间： cisco-mso选择器： k8s-app=schemaservice，pod-template-hash=7597ff4c5标签： 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注释： deployment.kubernetes.io/desired-replicas: deployment.kubernetes.io/max-replicas: deployment.kubernetes.io/revision: 1控制者：部署/schemaservice副本：当前/1 Pods状态：1正在运行/0等待/0成功/0失败Pod模板：标签：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服务帐户：cisco-mso-sa Init容器：init-msc：映像：cisco-mso/tools:3.7.1j端口：<none>主机端口：<none>命令：/check_mongo.sh/launchscala.sh http://:8080/api/v1/schemas/health /wrapper.sh环境：<none>安装： /infracerts(rw)的秘密：容器：schemaservice：映像： cisco-mso/schemaservice 7.1j端口：8080/TCP、8080/UDP主机端口：0/TCP、0/UDP命令：.....架构活动性：http-get的delay=300s timeout=20s period=30s #success=1 #failure=3环境：JAVA_OPTS: -XX:+IdleTuningGcOnIdle安装： /jwtsecrets(rw)/logs from logs(logs)/secrets(rw)msc-schemts service-ssl：映像：cisco-mso/sslcontainer:3.7.1j端口：443/UDP、443/TCP主机端口：0/UDP、0/TCP命令：.....

NDO Pod审核

Pod是在同一Linux命名空间（不同于K8s命名空间）和同一K8s节点中运行的一组密切相关的容器。这是最原子化的物体K8s把手，因为它不与容器相互作用。该应用程序可由单个容器组成，或者对于许多容器而言较为复杂。使用下一命令，我们可以检查任何给定名称空间的Pod:

```
[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 ~]$
```

第二列中显示的数字表示每个Pod的容器数量。

此 **describe** 选项也可用，其中包括每个Pod上容器的详细信息。

```
[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
```

显示的信息包括每个容器的容器图像并显示使用的容器运行时。在本例中，CRI-O(**cri-o**)，早期版本的ND用于与Docker配合使用，这会影响如何连接到容器。

Spoiler

例如，当 **cri-o** ，我们希望通过交互式会话连接到容器(通过 exec -it 选项)；但不是 docker 命令，我们使用**crictl**命令：

```
schemaservice:

   Container ID:  cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac

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

我们使用以下命令：

```
[root@MxNDsh01 ~]# crictl exec -it
d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac bash

root@schemaservice-7597ff4c5-w4x5d:/#

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

root
```

对于以后的ND版本，要使用的容器ID不同。首先，我们需要使用命令 **crictl ps** 列出在每个节点上运行的所有容器。我们可以根据需要过滤结果。

```
[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 ~]#
```

使用第一列中的值，我们可以使用与之前相同的命令访问容器运行时：

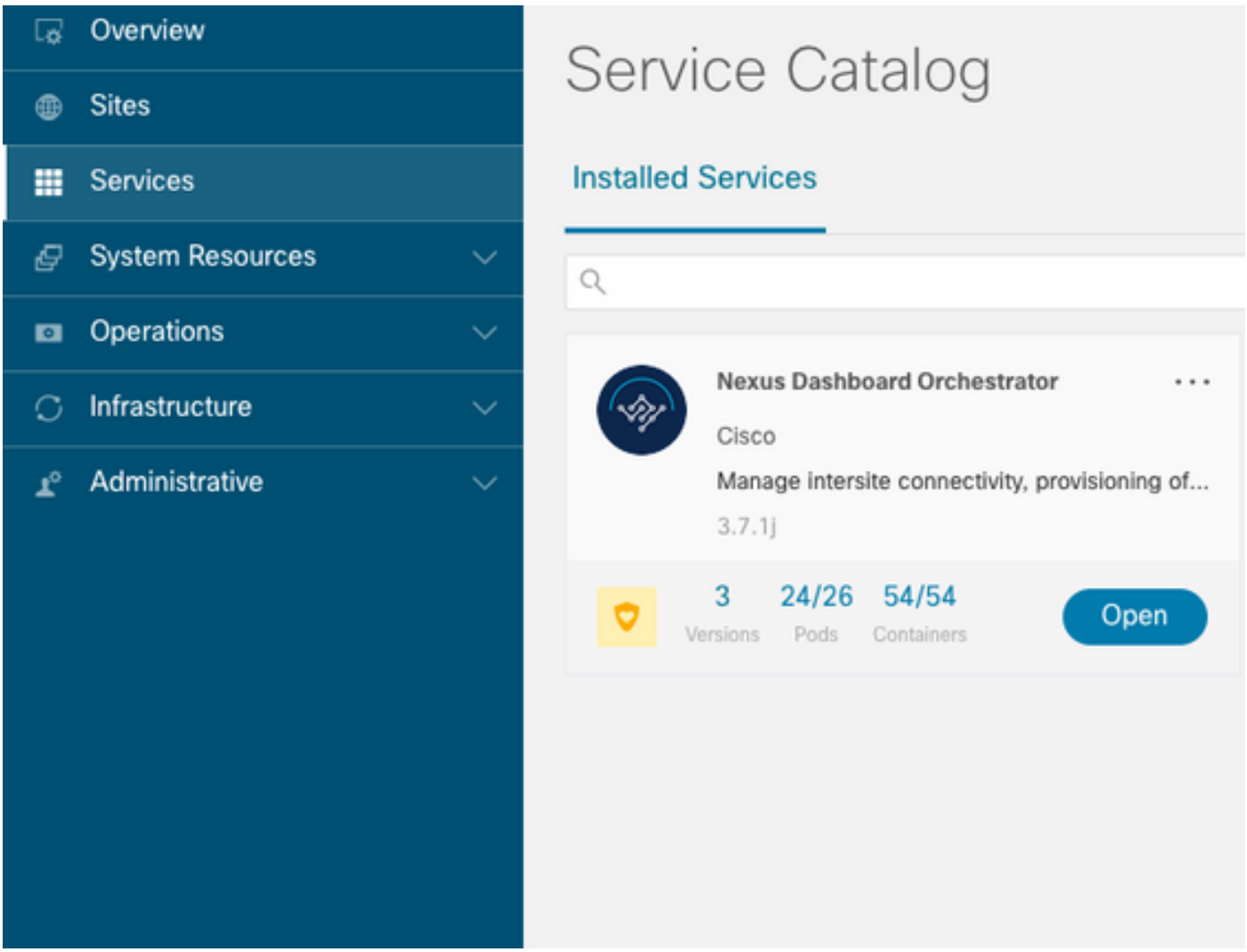
```
[root@singleNode ~]# crictl exec -it 4b26f67fc10cf bash
root@backupservice-8c699779f-j9jtr:/# pwd
/
```

例如，当使用cri-o时，我们希望通过交互式会话将容器（通过exec -it选项）连接到上一个输出的容器；但我们不使用docker命令，而是使用criclt命令：schemaservice：容器ID:cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac　映像：cisco-mso/schemaservice:3.7.1j我们使用此命令：[root@MxNDsh01 ~]# crictl exec -it d2287f8659dec6848c0100b7d24aeebd50 6f3f77af660238ca0c9c7e8946f4ac bashroot@schemaservice-7597ff4c5-w4x5d:/#root@schemaservice-7597ff4c5-w4x5d:/# whoamiroot对于以后的ND版本，要使用的容器ID不同。首先，我们需要使用命令crictl ps列出在每个节点上运行的所有容器。我们可以根据需要过滤结果。[root@singleNode ~]# crictl ps| grep backupa9bb161d67295 10.31.125.241:30012/cisco-mso/sslcontainer@sha256:26581eebd0bd6f4378a5fe4a98973dbda417c1905689f71f229765621f0cee75 2天前运行msc-backupservice-ssl 0 84b3c691cfc2b4b26f67fc10cf 10.31.125.241:30012/cisco-mso/backupservice@sha256:c21f4cdde696a5f2dfa7bb910b7278fc3fb4d46b02f42c3554f872ca8c87c061 2天前运行backupservice 0 84b3c691cfc2b[root@singleNode ~]#使用第一列的值，我们可以使用与之前相同的命令访问容器运行时：[root@singleNode ~]# crictl exec -it 4b26f67fc 10cf bashroot@backupservice-8c699779f-j9jtr:/# pwd/

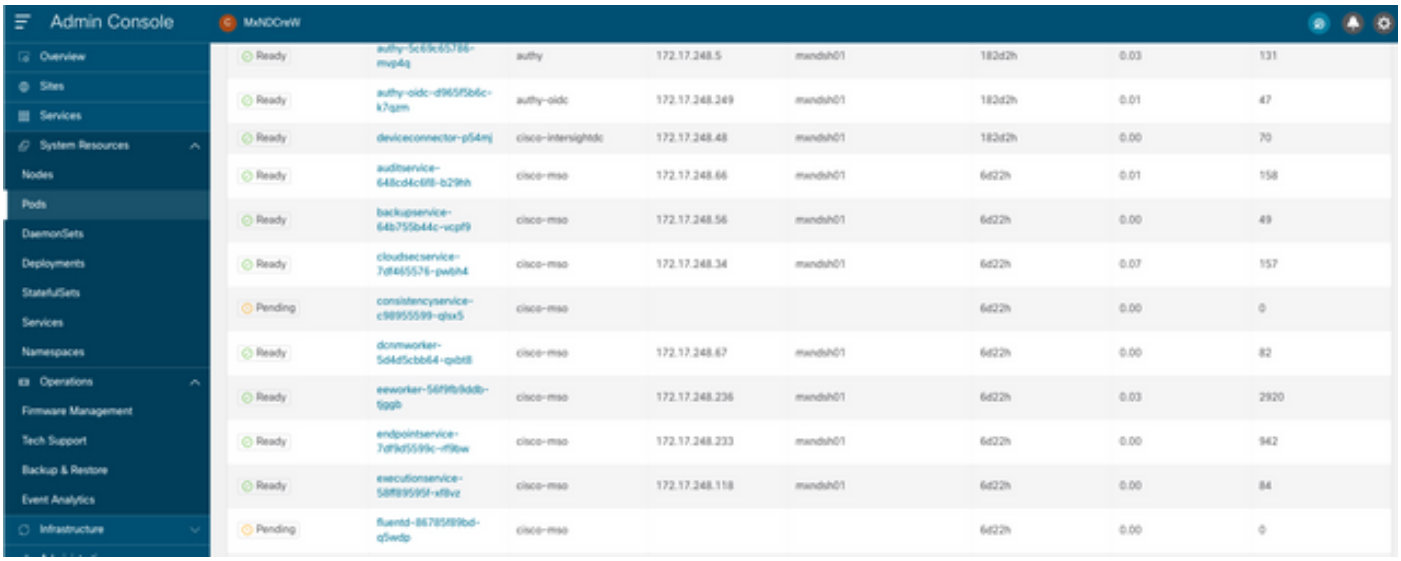
使用案例Pod不正常

我们可以使用此信息来排除部署中的Pod不正常的原因。在本示例中，Nexus控制面板版本为2.2-1d，受影响的应用为Nexus控制面板协调器(NDO)。

NDO GUI从“服务”(Service)视图显示一组不完整的Pod。在本例中，26个Pod中有24个。



另一个视图位于 **System Resources -> Pods** 查看Pod显示的状态与显示的 **Ready**.



不正常Pod的CLI故障排除

根据已知事实，命名空间为cisco-mso（虽然排除故障时，其他应用/命名空间也相同），如果存在任何不正常的Pod视图，则会显示：

```
[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
```

在本例中，我们重点介绍一致性服务Pod。从JSON输出中，我们可以使用jsonpath从状态字段获取特定信息：

```
[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 ~]$
```

我们看到**status**字典，并且在一个名为**conditions**的列表中，该列表包含词典，并且项为**message**和**value**，其中{"\n"}部分是在结尾创建新行：

```
[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 ~]$
```

此命令显示如何从 **get Pod** 对于命名空间：

```
[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
```

使用 get pods 命令，我们可以获得与上一个输出中的问题匹配的Pod ID。在本例中 **consistencyservice-c98955599-qlsx5**.

JSON输出格式还提供如何从给定输出检查特定信息。

```
[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 ~]$
```

JSON输出必须在具有相同名称的属性中包含有关状态的信息。消息中包含有关原因的信息。

```
[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 ~]$
```

我们可以访问有关Pod的状态和要求的信息：

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

这里必须说明如何计算该值。在本例中，cpu 500m指500 miligores，内存中的1G表示GB。

此 **Describe** 节点的选项显示可用于集群中每个K8工作者的资源（主机或虚拟机）：

```
[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 ~]$
```

Allocatable部分显示每个节点可用的CPU 、内存和存储中的总资源。Allocated部分显示已在使用的资源。CPU的**值13**指**13**个内核或13,000(13K)**毫内核**。

在本示例中，节点超订用**了**，这解释了为什么Pod无法启动。通过删除ND APP或添加VM资源清除ND后。

集群会不断尝试部署任何挂起策略，因此，如果资源是空闲的，则可以部署Pod。

```
[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
```

使用用于资源检查的命令，我们确认集群具有可用的CPU资源：

```
[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 ~]$
```

部署详细信息包括一条消息，其中包含有关Pod当前状况的信息：

```
[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

如何从容器内部运行网络调试命令

由于容器仅包含特定于Pod的最小库和依赖关系，因此大多数网络调试工具（ping、ip route和ip addr）在容器本身中不可用。

当需要排除服务（在ND节点之间）或与Apic的连接网络故障时，这些命令非常有用，因为多个微服务需要通过数据接口(bond0或bond0br)与控制器通信。

此 **nsenter** 实用程序（仅根用户）允许从ND节点运行网络命令，因为它在容器内。为此，请从要调试的容器中找到进程ID(PID)。这是通过Pod K8s ID根据容器运行库中的本地信息完成的，例如旧版的Docker，以及 **cri-o** 作为默认选项。

检查Pod Kubernetes(K8s)ID

从cisco-mso命名空间内的Pod列表中，我们可以选择要进行故障排除的容器：

```
[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
```

Pod必须在同一个K8s节点中运行。对于生产环境，我们可以添加 **-o wide** 选项，了解每个Pod运行的节点。使用Pod K8s ID（在上一输出示例中加粗），我们可以检查容器运行时分配的进程(PID)。

如何从容器运行时检查PID

新的默认容器运行时是CRI-O for Kubernetes。因此，文档在命令规则之后。CRI-O分配的进程ID(PID)在K8s节点中可以是唯一的，可以使用 crictl 实用程序。

此 **ps** 选项显示CRI-O为构建Pod的每个容器提供的ID，其中两个用于站点服务示例：

```
[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 ~]#
```

有了此信息，我们就可以使用 inspect CRIO-ID 选项可查看赋予每个容器的实际PID。此信息对于 nsenter 指令：

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

如何使用nsenter在容器内运行网络调试命令

使用上述输出中的PID，我们可以在下一命令语法中用作目标：

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

此 **--net** 选项允许我们在网络名称空间中运行命令，因此可用的命令数量是有限的。

例如：

```
[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
```

ping也可用，它测试从容器到外部的连接，而不只是测试K8节点。

```
[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]
```

如何从容器内部运行网络调试命令由于容器仅包含特定于Pod的最小库和依赖关系，因此大多数网络调试工具（ping、ip route和ip addr）在容器本身中不可用。 当需要排除服务（在ND节点之间）或与Apic的连接网络故障时，这些命令非常有用，因为多个微服务需要通过数据接口（bond0或bond0br）与控制器通信。 Nsenter实用程序（仅根用户）允许从ND节点运行网络命令，因为它位于容器内。为此，请从要调试的容器中找到进程ID(PID)。这是通过Pod K8s ID根据容器运行库中的本地信息实现的，例如旧版的Docker，以及新版的cri-o（默认值）。 检查Pod Kubernetes(K8s)ID从cisco-mso命名空间内的Pod列表中，我们可以选择要排除故障的容器：[root@MxNDsh01 ~]# kubectl get pod -n cisco-msoNAME READY STATUS RESTARTS AGEconsistencyservice-569bdf5969-xkwpg 3/3运行0 9heeworker-65dc5dd849-485tq 2/2运行0 163mendpointservice-5db6f57884-2 5g 2/2运行0 9hexecutionservice-6c4894d4f7-p8fzk 2/2运行0 9hsiteservice-64dfcdf658-lvbr4 3/3运行0 9hsiteupgrade-68bcf987cc-ttn7h 2/2运行0 9h Pod必须在同一K8s节点中运行。对于生产环境，我们可以在末尾添加 — o wide选项以找出每个Pod运行的节点。使用Pod K8s ID（在上一输出示例中加粗），我们可以检查容器运行时分配的进程(PID)。 How to Inspect the PID from the Container Runtime新的默认容器运行时是CRI-O for Kubernetes。因此，文档在命令规则之后。CRI-O分配的进程ID(PID)在K8s节点中可以是唯一的，可以使用crictl实用程序发现该节点。 ps选项显示CRI-O为构建Pod的每个容器提供的ID，其中两个用于站点服务示例：[root@MxNDsh01 ~]# crictl ps |grep siteservicefb560763b06f2 172.31.0.0:30012/cisco-mso/sslcontainer@sha256:2d788fa493c885ba8c9e5944596b864d090d9051b0eab82123ee4d19596279c9 10小时前，运行msc-siteservice2-ssl 0 074727b4e9f51ad2d42aae1ad9 1d0195292f7fcc62f38529e135a1315c358067004a086cfed7e059986ce615b0 10小时前，运行siteservice-leader-election 0 074727b4e9f5129b0d4 1d1e3 172.31.0.0:30012/cisco-mso/siteservice@sha256:80a2335bcd5366952b4d60a275b20c70de0bb65a47bf8ae6d988f07b1e0bf494 10小时前Running siteservice 0 074727b4e9f51[root@MxNDsh01 ~]#使用此信息，我们可以使用inspect CRIO-ID选项查看每个容器的实际PID。nsenter命令需要此信息：[root@MxNDsh01 ~]# crictl inspect fb560763b06f2| grep -i pid"pid": 239563,"pids": {"type": "pid"如何使用nsenter在容器内运行网络调试命令使用上述输出中的PID，我们可以在下一命令语法中用作目标：nsenter —target <PID> —net <NETWORK COMMAND> —net选项允许我们在网络命名空间内运行命令，因此可用的命令数量有限。例如：[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:fe72:7bfb prefixlen 64 scopeid 0x20<link>ether 9a:4f 32:72:7b:fb txqueuelen 0（以太网）RX数据包916346 bytes 271080553(258.5 MiB)RX错误0丢弃183超载0帧0TX数据包828016 bytes 307255950(293.0 MiB)TX错误0丢弃0超载0载波0冲突0lo：标志=73<UP，LOOPBACK，RUNNING> mtu 65536inet 127.0.0.1 netmask 25.0.0inet6 ::1 prefixlen 12 scopeid 0x10<host>loop txqueuelen 1000（本地环回）RX packets 42289 bytes 14186082(13.5 MiB)RX errors 0 dropped 0 overruns 0 frame 0TX packets 42289 bytes 14186082(13.5 MiB)TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0该ping也可用，它测试从容器到外部的连接，而不只是K8s节点。[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到1xx.2xx.3xx.4xx:443... connected.警告：无法验证1xx.2xx.3xx.4xx由“/C=US/ST=CA/O=Cisco System/CN=APIC”颁发的证书：无法本地验证颁发者的权限。警告：证书公用名“APIC”与请求的主机名“1xx.2xx.3xx”不匹配.HTTP请求已发送，等待响应…… 200 OKLength: 3251(3.2K)[text/html]保存到：'index.html'100%[===>] 3,251 —.-K/s in 0s2023-01-24 23:46:04(548 MB/s)- 'index.html'已保存

关于此翻译
思科采用人工翻译与机器翻译相结合的方式将此文档翻译成不同语言，希望全球的用户都能通过各自的语言得到支持性的内容。

请注意：即使是最好的机器翻译，其准确度也不及专业翻译人员的水平。

Cisco Systems, Inc. 对于翻译的准确性不承担任何责任，并建议您总是参考英文原始文档（已提供链接）。