Fehlerbehebung und Überprüfung von NDO-Ressourcen

Inhalt

[Einleitung](#)
[NDO-Schnellstart](#)
[Kubernetes mit NDO-Crash-Kurs](#)
[NDO-Übersicht mit Kubernetes-Befehlen](#)
[CLI-Zugriffs-Anmeldung](#)
[NDO Namespaces-Überprüfung](#)
[Prüfung der NDO-Bereitstellung](#)
[NDO-Replikatsatz (RS) - Überprüfung](#)
[NDO Pod-Prüfung](#)
[Anwendungsfall: POD ist nicht fehlerfrei](#)
[CLI-Fehlerbehebung bei ungesunden PODs](#)
[Ausführen von Netzwerkdebugbefehlen innerhalb eines Containers](#)
[Prüfen Sie die Pod Kubernetes (K8s) ID.](#)
[Überprüfen der PID aus der Container-Laufzeit](#)
[Verwenden von nsenter zum Ausführen von Netzwerk-Debugbefehlen in einem Container](#)

Einleitung

In diesem Dokument wird beschrieben, wie NDO mit der Kommandozeile kubectl und container überprüft und Fehler behoben werden.

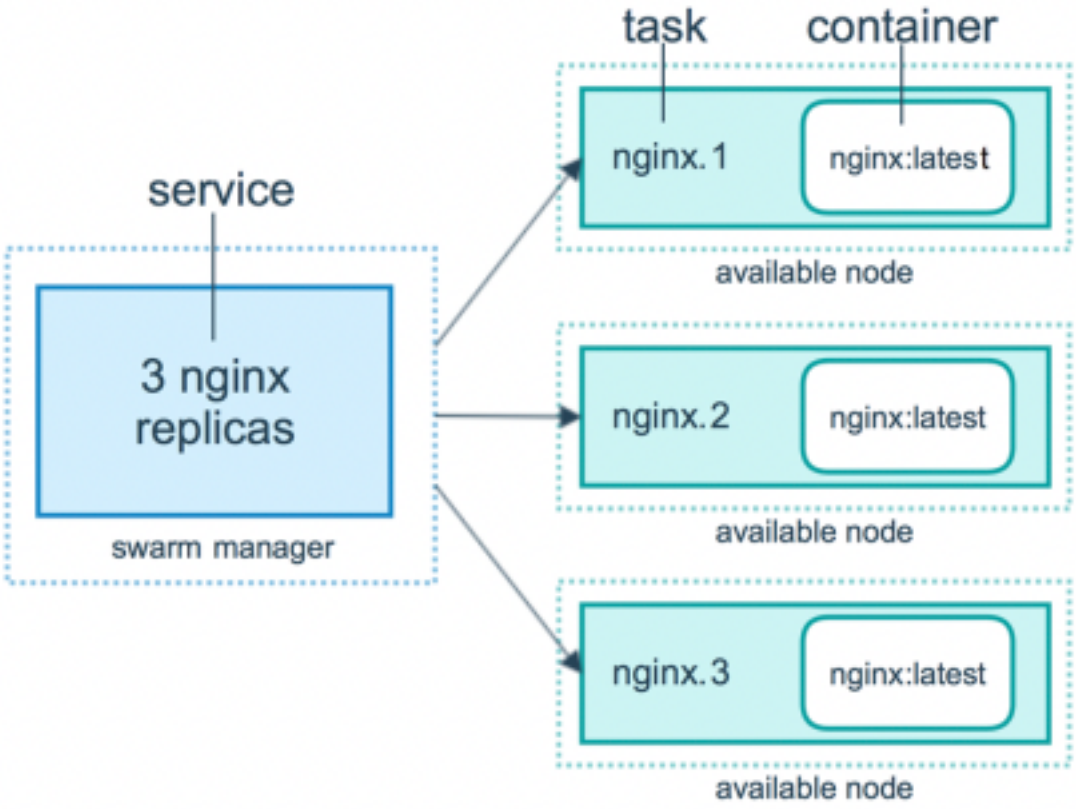
NDO-Schnellstart

Der Cisco Nexus Dashboard Orchestrator (NDO) ist ein Verwaltungstool für Fabrics. Mit diesem Tool können Benutzer verschiedene Arten von Fabrics verwalten, z. B. Cisco® Application Centric Infrastructure (Cisco ACI®)-Standorte, Cisco Cloud ACI-Standorte und Cisco Nexus Dashboard Fabric Controller (NDFC)-Standorte, von denen jeder über einen eigenen Controller verwaltet wird (APIC-Cluster, NDFC-Cluster oder Cloud APIC-Instanzen in einer Public Cloud).

NDO bietet konsistente Orchestrierung, Skalierbarkeit und Notfallwiederherstellung von Netzwerken und Richtlinien über mehrere Rechenzentren hinweg über eine zentrale Oberfläche.

Früher wurde der MSC (Multi-Site Controller) als Cluster mit drei Knoten und offenen virtuellen VMWare Appliances (OVAs) bereitgestellt, mit denen Kunden einen Docker Swarm-Cluster und die MSC-Services initialisieren konnten. Dieser Schwarm-Cluster verwaltet die MSC-Mikroservices als Docker-Container und -Services.

Dieses Bild zeigt eine vereinfachte Ansicht, wie der Docker-Schwarm die Mikroservices als Replikate desselben Containers verwaltet, um eine hohe Verfügbarkeit zu erreichen.



Der Docker-Schwarm war für die Verwaltung der erwarteten Anzahl von Replikaten für jeden der Mikroservices in der MSC-Architektur verantwortlich. Aus Sicht des Docker-Schwarms war der Controller für mehrere Standorte die einzige Container-Bereitstellung, die orchestriert werden konnte.

Das Nexus Dashboard (ND) ist eine zentrale Managementkonsole für mehrere Rechenzentrumsstandorte und eine gemeinsame Plattform, auf der Cisco Services für den Rechenzentrumsbetrieb gehostet werden. Zu diesen Services gehören Nexus Insight und MSC Version 3.3 sowie die Namensänderung in Nexus Dashboard Orchestrator (NDO).

Die meisten Mikroservices der MSC-Architektur bleiben zwar unverändert, NDO wird jedoch in einem Kubernetes-Cluster (K8s) und nicht in einem Docker Swarm-Cluster bereitgestellt. So kann ND mehrere Anwendungen oder Bereitstellungen orchestrieren, anstatt nur eine einzige zu verwenden.

Kubernetes mit NDO-Crash-Kurs

Kubernetes ist ein Open-Source-System für die automatisierte Bereitstellung, Skalierbarkeit und Verwaltung von containerisierten Anwendungen. Als Docker arbeitet Kubernetes mit der Containertechnologie, ist aber nicht an Docker gebunden. Das bedeutet, dass Kubernetes andere Container-Plattformen (Rkt, PodMan) unterstützt.

Ein wesentlicher Unterschied zwischen Swarm und Kubernetes besteht darin, dass letzteres nicht direkt mit Containern funktioniert, sondern mit einem Konzept von am gleichen Standort befindlichen Containergruppen, den so genannten Pods.

Die Container in einem Pod müssen auf demselben Knoten ausgeführt werden. Eine Gruppe von PODs wird als Bereitstellung bezeichnet. Eine Kubernetes-Bereitstellung kann eine ganze Anwendung beschreiben.

Kubernetes ermöglicht es den Benutzern auch, eine bestimmte Menge an Ressourcen für eine bestimmte Anwendung zur Verfügung zu stellen. Dies geschieht mithilfe von Replikations-Controllern, um sicherzustellen, dass die Anzahl der PODs mit den Anwendungsmanifesten konsistent ist.

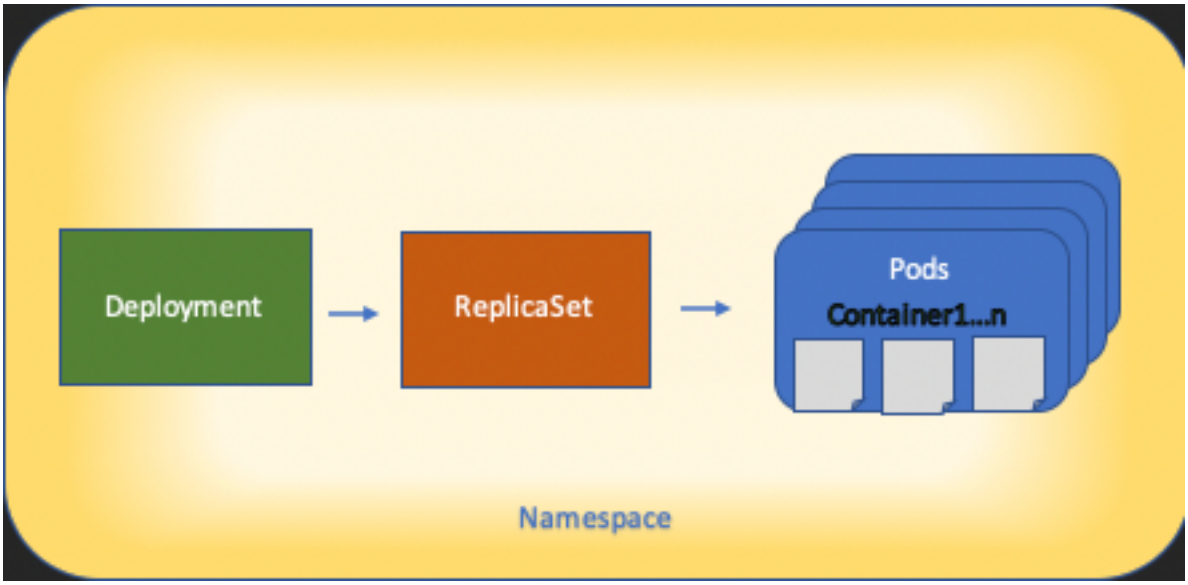
Ein Manifest ist eine YAML-formatierte Datei, die eine vom Cluster bereitzustellende Ressource beschreibt. Bei der Ressource kann es sich um eine der zuvor beschriebenen Ressourcen oder um andere für Benutzer verfügbare Ressourcen handeln.

Der Zugriff auf die Anwendung ist extern mit einem oder mehreren Diensten möglich. Kubernetes bietet hierfür eine Load Balancer-Option.

Kubernetes bietet auch eine Möglichkeit, verschiedene Ressourcen mit dem Konzept der Namespaces zu isolieren. Das ND verwendet Namespaces, um verschiedene Anwendungen und Clusterdienste eindeutig zu identifizieren. Wenn CLI-Befehle ausgeführt werden, geben Sie immer den Namespace an.

Obwohl keine tiefen Kenntnisse über Kubernetes erforderlich sind, um Probleme mit ND oder NDO zu beheben, ist ein grundlegendes Verständnis der Kubernetes-Architektur erforderlich, um die Ressourcen richtig zu identifizieren, die Probleme haben oder die Aufmerksamkeit benötigen.

Die Grundlagen der Kubernetes-Ressourcenarchitektur sind in diesem Diagramm dargestellt:



Es ist wichtig, sich zu erinnern, wie jede Art von Ressource mit anderen interagiert, und es spielt eine wichtige Rolle bei der Überprüfung und Fehlerbehebung.

NDO-Übersicht mit Kubernetes-Befehlen

CLI-Zugriffs-Anmeldung

Für den CLI-Zugriff von SSH auf NDO muss der **admin-user** Kennwort erforderlich. Stattdessen verwenden wir jedoch die **rescue-user** Kennwort. Beispiel:

```
ssh rescue-user@ND-mgmt-IP
rescue-user@XX.XX.XX.XX's password:
```

```
[rescue-user@MxNDsh01 ~]$ pwd
/home/rescue-user
[rescue-user@MxNDsh01 ~]$
```

Dies ist der Standardmodus und -benutzer für den CLI-Zugriff, und die meisten Informationen sind verfügbar.

NDO Namespaces-Überprüfung

Dieses K8s-Konzept ermöglicht die Isolierung verschiedener Ressourcen im Cluster. Mit dem nächsten Befehl können die verschiedenen bereitgestellten Namespaces überprüft werden:

```
[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
```

Die fett markierten Einträge gehören zu Anwendungen im NDO, während die Entitäten, die mit dem Präfix **kube** beginnen, zum Kubernetes-Cluster gehören. Jeder Namespace verfügt über eigene unabhängige Bereitstellungen und Pods.

Die kubectl-CLI ermöglicht die Angabe eines Namespaces mit dem **--namespace** Wenn ein Befehl ohne diesen ausgeführt wird, geht die CLI davon aus, dass der Namespace **default** (Namespace für 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.
```

Die kubectl-CLI ermöglicht verschiedene Formate für die Ausgabe, z. B. yaml, JSON oder eine benutzerdefinierte Tabelle. Dies wird durch die **-o** [format]. Beispiele:

```
[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": ""
   }
}
```

Aus dem vorherigen Text wird ein Wörterbuch ausgegeben, in dem einer seiner Schlüssel **Items**

genannt wird, und der Wert ist eine **Liste** von Wörterbüchern, in denen jedes **Wörterbuch** einen **Namespace-**Eintrag berücksichtigt und seine Attribute ein Schlüssel-Wert-Paar im Wörterbuch oder geschachtelten Wörterbüchern sind.

Dies ist relevant, da K8s Benutzern die Möglichkeit bietet, jsonpath als Ausgabe auszuwählen. Dies ermöglicht komplexe Operationen für ein JSON-Datenarray. Wenn wir z. B. aus der vorherigen Ausgabe den Wert **name** für Namespaces, müssen wir auf den Wert der Elementliste zugreifen, dann die metadata und den Wert des Schlüssels **name**. Dies kann mit dem folgenden Befehl durchgeführt werden:

```
[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 ~]$
```

Die beschriebene Hierarchie wird zum Abrufen der spezifischen erforderlichen Informationen verwendet. Der Zugriff auf alle Elemente erfolgt im Wesentlichen über die **items** Liste mit **Elementen[*]**, dann der Schlüssel metadata und **name** Mit **metadaten.name** kann die Abfrage andere anzuzeigende Werte enthalten.

Dasselbe gilt für die Option benutzerdefinierter Spalten, die eine ähnliche Methode zum Abrufen der Informationen aus dem Datenarray verwenden. Wenn wir z. B. eine Tabelle mit den Informationen über die **name** und UID -Werte können wir den folgenden Befehl anwenden:

```
[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
```

Die Ausgabe erfordert einen Namen für jede anzuzeigende Spalte und dann die Zuweisung des

Werts für die Ausgabe. In diesem Beispiel gibt es zwei Spalten: **NAME** und **UID**. Diese Werte gehören zu .metada.name und .metadata.uid jeweils. Weitere Informationen und Beispiele finden Sie unter:

JSONPath-Unterstützung

Benutzerdefinierte Spalten

Prüfung der NDO-Bereitstellung

Eine Bereitstellung ist ein K8s-Objekt, das einen verknüpften Speicherplatz zum Verwalten von ReplicaSet und Pods bereitstellt. Bereitstellungen betreffen die Bereitstellung aller PODs, die zu einer Anwendung gehören, sowie die erwartete Anzahl von Kopien jeder Anwendung.

Die kubectl-CLI enthält einen Befehl zum Überprüfen der Bereitstellungen für einen beliebigen Namespace:

```
[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
```

Dieselbe benutzerdefinierte Tabelle kann mit der deployment statt **namespace** und **-n** um die gleichen Informationen wie zuvor anzuzeigen. Dies liegt daran, dass die Ausgabe ähnlich strukturiert ist.

```
[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
```

Beachten Sie, dass die Anzahl der angezeigten Kopien für die Bereitstellung und nicht die Anzahl der Pods für jeden Mikroservice gilt.

Wir können das Schlüsselwort **describe** statt **get** um detailliertere Informationen über eine Ressource anzuzeigen, in diesem Fall die Schemaservice-Bereitstellung:

```
[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 ~]$
```

Die Fehlermeldung **describe** ermöglicht auch die Einbindung des **--show-events=true** Option, um alle relevanten Ereignisse für die Bereitstellung anzuzeigen.

Spoiler

NDO-Replikatsatz (RS) - Überprüfung

Spoiler

NUR FÜR ROOT-BENUTZER VERFÜGBAR

Ein Replikatsatz (RS) ist ein K8s-Objekt mit dem Ziel, eine stabile Anzahl von Replikat-Pods zu erhalten. Dieses Objekt erkennt auch, wenn eine ungesunde Anzahl von Replikaten mit einer periodischen Sonde auf die PODs gesehen wird.

Die RS sind auch in Namespaces organisiert.

```
[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
```

Die Fehlermeldung describe enthält Informationen über die URL, den Port, den der Test verwendet, sowie die Häufigkeit der Tests und den Schwellenwert für Fehler.

```
[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 Replica Set (RS) Review #### DIES IST NUR FÜR ROOT-BENUTZER VERFÜGBAR #### Ein Replikatset (RS) ist ein K8s-Objekt mit dem Ziel, eine stabile Anzahl von Replikat-Pods zu erhalten. Dieses Objekt erkennt auch, wenn eine ungesunde Anzahl von Replikaten mit einer periodischen Sonde auf die PODs gesehen wird. Die RS sind auch in Namespaces organisiert. [root@MxNDsh01 ~]# kubectl get rs -n cisco-msoNAME ERWÜNSCHT AKTUELL BEREIT AGEauditservice-648cd4c6f8 1 1 3d22hbackupservice-64b755b44c 1 1 1 3d22hcloudsecservice-7df 465576 1 1 1 3d22hKonsistenzdienst-c98955599 1 1 1 3d22hdcnmworker-5d4d5cbb64 1 1 1 3d22heeworker-56f9fb9ddb 1 1 3d22hendpointService-7df9d5599c 1 1 1 3d22hexecutionservice-58ff89595f 1 1 3d22hfluentd-86785f89bd 1 1 3d22himportservice-88bcc8547 1 1 1 3d22hjobscheduleService-5d4fdfd69 6 1 1 1 3d22hnInfoService-75c988cfd4 1 1 3d22hpctagvnidservice-644b755596 1 1 3d22hplatformservice-65cddb946f 1 1 3d22hplatformservice2-6796576659 1 1 1 3d22hpolicyservice-545b9c7d9c 1 1 1 3d22hschemaservice-7597ff4c5 1 1 3d22hsdaservice-5f477dd8c7 1 1 3d22hsdwanservice-6f87cc

d999d 1 1 1 3d22hSwitchService-86bb756585 1 1 1 3d22hStandortUpgrade-7d578f9b6d 1 1 1 3d22hsyncengine-5b8bdd6b45 1 1 1 3d22htemplateeng-5cbb f9fdc48 1 1 3d22hui-84588b7c96 1 1 1 3d22huserservice-87846f7c6 1 1 3d22h Die Option description umfasst die Informationen über die URL, den Port, den der Prüfpunkt verwendet, und die Periodizität von Tests und die Ausfallschwelle. [root@MxNDsh01 ~]# kubectl description 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 8s-app=schemaservice memory.resource.case.cncf.io/schemaservice=mem-xlg-service pod-template-hash=7597ff4c5Anmerkungen: deployment.kubernetes.io/desired-replicas: 1 deployment.kubernetes.io/max-replicas: 1 deployment.kubernetes.io/revision: 1Gesteuert von: Bereitstellung/schemaserviceReplika: 1 current / 1 wantedPods Status: 1 Running / 0 Waiting / 0 Succeeded / 0 FailedPod Vorlage: Labels: cpu.resource.case.cncf.io/schemaservice=cpu-lg-service k8s-app=schemaservice vice memory.resource.case.cncf.io/schemaservice=mem-xlg-service pod-template-hash=7597ff4c5 Servicekonto: cisco-mso-sa Init Container: init-msc: Image: cisco-mso/tools:3.7.1j Port: <none> Host Port: <none> Befehl: /check_mongo.sh Umgebung: <none> Mounts: /secrets from infracerts (rw) Container: schemaservice: Image: cisco-mso/schemaservice:3.7.1j Ports: 8080/TCP, 8080/UDP Host Ports: 0/TCP, 0/UDP Befehl: /launchscala.sh schemaservice Lebensdauer: http-get http://:8080/api/v1/schemas/health delay=300s timeout=20s period=30s #success=1 #failure=3 Umgebung: 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, 444 3/TCP-Host-Ports: 0/UDP, 0/TCP-Befehl: /wrapper.sh

NDO Pod-Prüfung

Ein Pod ist eine Gruppe eng verwandter Container, die im gleichen Linux-Namespace (anders als im K8s-Namespace) und im gleichen K8s-Knoten ausgeführt werden. Dies ist das atomarste Objekt, das K8s behandelt, da es nicht mit Containern interagiert. Die Anwendung kann aus einem einzigen Behälter bestehen oder komplexer mit vielen Behältern sein. Mit dem nächsten Befehl können wir die Pods eines beliebigen Namespaces überprüfen:

```
[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-5f477dd8c7-15jn7            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-87846f7c6-1zctd            2/2     Running   0          2d1h

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

Die in der zweiten Spalte angezeigte Zahl bezieht sich auf die Anzahl der Container für jeden Pod.

Die Fehlermeldung **describe** ist ebenfalls verfügbar, was detaillierte Informationen über die Container auf jedem Pod enthält.

```
[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
```

Die angezeigten Informationen enthalten das Containerbild für jeden Container und die verwendete Container-Laufzeit. In diesem Fall ist CRI-O (**cri-o**), frühere Versionen von ND für die Arbeit mit Docker, dies beeinflusst, wie man an einem Container zu befestigen.

Spoiler

Wenn z. B. **cri-o** verwendet wird, und wir möchten eine Verbindung über eine interaktive Sitzung zu einem Container herstellen (über die exec -it Option) aus der vorherigen Ausgabe in den Container zu übertragen. docker -Befehls verwenden wir den **crictl**-Befehl:

```
schemaservice:
   Container ID:  cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac
```

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

Wir verwenden diesen Befehl:

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

root@schemaservice-7597ff4c5-w4x5d:/#

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

root
```

Für spätere ND-Versionen wird eine andere Container-ID verwendet. Zuerst müssen wir den Befehl **crictl ps** um alle Container aufzulisten, die auf den einzelnen Knoten ausgeführt werden. Wir können das Ergebnis nach Bedarf filtern.

```
[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 ~]#
```

Mit dem Wert aus der ersten Spalte können wir dann mit dem gleichen Befehl wie zuvor auf die Container-Laufzeit zugreifen:

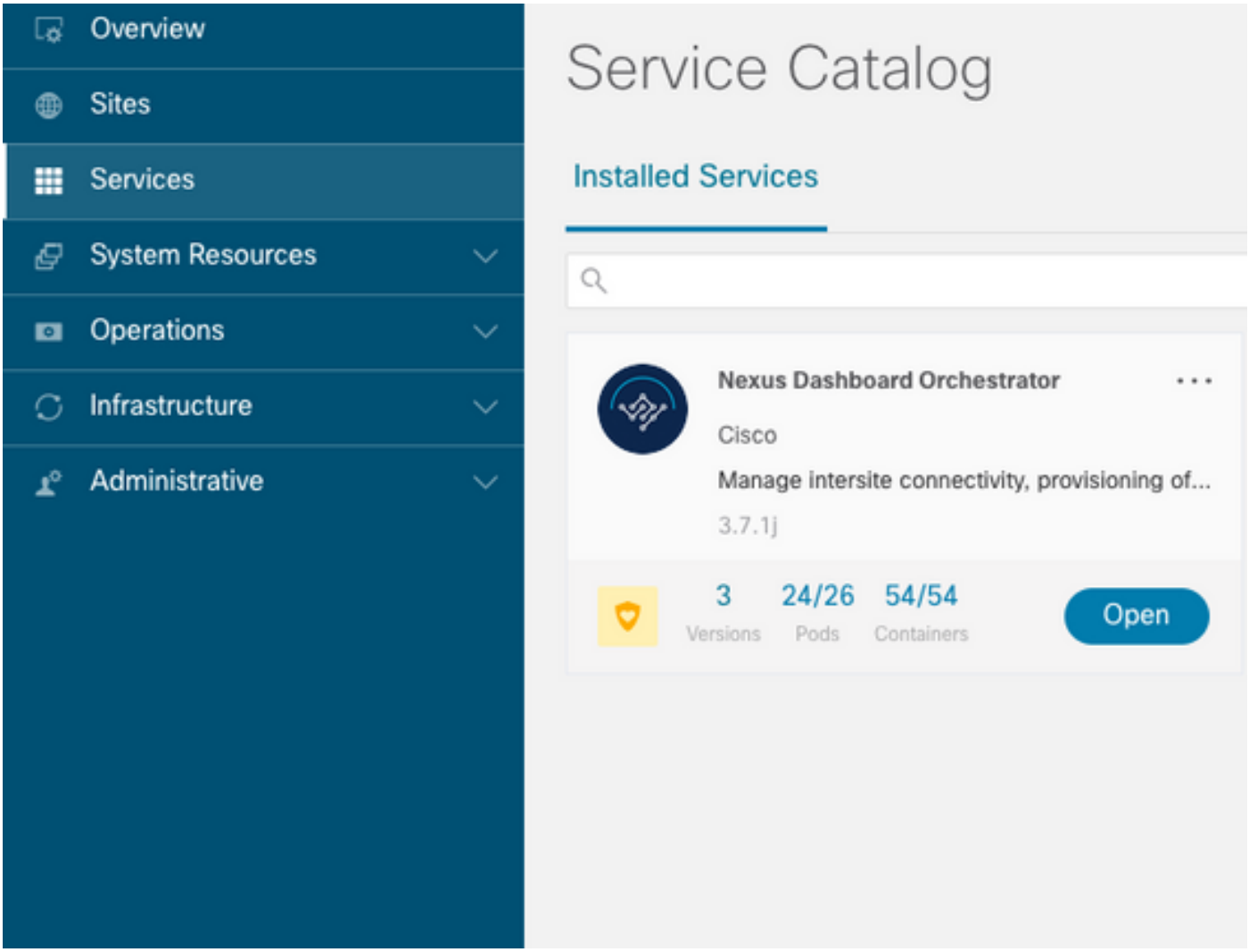
```
[root@singleNode ~]# crictl exec -it 4b26f67fc10cf bash
root@backupservice-8c699779f-j9jtr:/# pwd
/
```

Zum Beispiel, wenn cri-o verwendet wird, und wir wollen durch eine interaktive Sitzung mit einem Container verbinden (über die exec -it Option), um den Container aus der vorherigen Ausgabe; aber anstelle des docker-Befehl, verwenden wir den criclt-Befehl: schemaservice: Container-ID: cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac Bild: cisco-mso/schemaservice:3.7.1j Wir verwenden diesen Befehl: [root@MxNDsh01 ~]# cricttc l exec -it d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac bashroot@schemaservice-7597ff4c5-w4x5d:/#root@schemaservice-7597ff4c5-w4x5d:/# whoamiroot Für spätere ND-Versionen wird eine andere Container-ID verwendet. Zuerst müssen wir den Befehl crictl ps verwenden, um alle Container aufzulisten, die auf jedem Knoten ausgeführt werden. Wir können das Ergebnis nach Bedarf filtern. [root@singleNode ~]# crictl ps| grep backupa9bb161d67295 10.31.125.241:30012/cisco-mso/sslcontainer@sha256:26581eebd0bd6f4378a5fe4a98973dbda417c1905689f71f229765621f0cee75 vor 2 Tagen, die msc-backupservice-ssl 0 84b3c691cfc2b4b26f67fc10cf 10.31.125.241:30012/cisco-mso/backupservice@sha256:c21f4cdde696a5f2dfa7bb910b7278fc3fb4d46b02f42c3554f872ca8c87c061 vor 2 Tagen ausführen backupservice 0 84b3c691cfc2b[root@singleNode ~]# Mit dem Wert aus der ersten Spalte können wir dann mit dem gleichen Befehl wie zuvor auf die Container-Laufzeit zugreifen: [root@singleNode ~]# crictl exec -it 4b26f67fc10cf bashroot@backupservice-8c699779f-j9jtr:/# pwd/

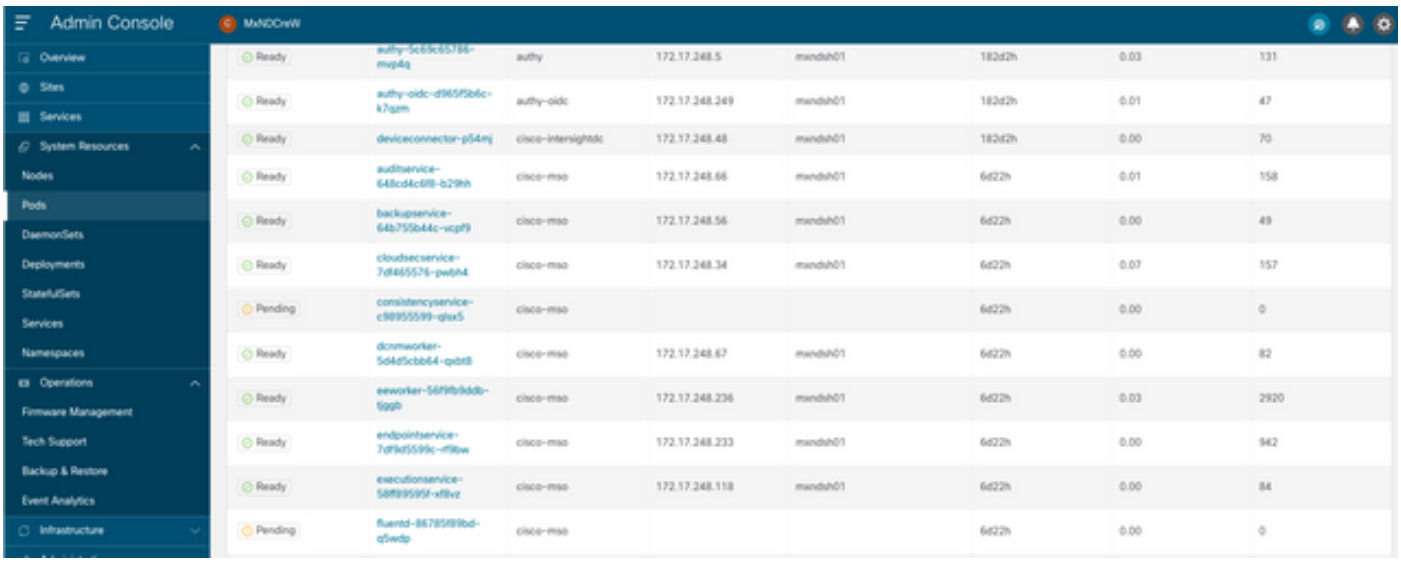
Anwendungsfall: POD ist nicht fehlerfrei

Wir können diese Informationen verwenden, um Probleme mit Pods aus einer Bereitstellung zu beheben. In diesem Beispiel ist die Nexus Dashboard-Version 2.2-1d, und die betroffene Anwendung ist Nexus Dashboard Orchestrator (NDO).

In der NDO-GUI wird ein unvollständiger Satz von PODs aus der Serviceansicht angezeigt. In diesem Fall 24 von 26 Pods.



Eine andere Ansicht, die unter dem **System Resources -> Pods** Anzeige, bei der die PODs einen anderen Status anzeigen als **Ready**.



CLI-Fehlerbehebung bei ungesunden PODs

Der Namespace ist bekanntlich cisco-mso (obwohl er bei der Fehlerbehebung für andere Anwendungen/Namespaces identisch ist). Die Pod-Ansicht wird angezeigt, wenn eine fehlerhafte Anwendung vorhanden ist:

```
[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
```

Für dieses Beispiel konzentrieren wir uns auf die Konsistenzservice-Pods. Aus der JSON-Ausgabe können wir die spezifischen Informationen aus den Statusfeldern abrufen. Verwenden Sie hierzu 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 ~]$
```

Wir sehen das **Status-**Dictionary und in einer Liste, die **Bedingungen** genannt wird, mit Dictionarys als Elementen mit der Schlüsselmeldung und dem **Wert** besteht der Teil {"\n"} darin, am Ende eine neue Zeile zu erstellen:

```
[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 ~]$
```

Dieser Befehl zeigt, wie Sie mithilfe des **get Pod** für den 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
```

Mit dem get pods können wir die Pod-ID mit Problemen abrufen, die mit denen in der vorherigen Ausgabe übereinstimmen müssen. In diesem Beispiel **consistencyservice-c98955599-qlsx5**.

Das JSON-Ausgabeformat bietet auch die Möglichkeit, bestimmte Informationen aus der jeweiligen Ausgabe zu überprüfen.

```
[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 ~]$
```

Die JSON-Ausgabe muss Informationen über den Status des Attributs mit dem gleichen Namen enthalten. Die Nachricht enthält Informationen zur Ursache.

```
[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 ~]$
```

Wir können auf Informationen über den Status und die Anforderungen für die PODs zugreifen:

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

An dieser Stelle ist es wichtig zu erwähnen, wie der Wert berechnet wird. In diesem Beispiel bezieht sich die CPU **500m** auf **500 Millicores**, und der **1G-**Speicher gilt für GB.

Die Fehlermeldung **Describe** -Option für den Knoten zeigt die für jeden K8s-Worker im Cluster (Host oder VM) verfügbare Ressource an:

```
[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 ~]$
```

Im Abschnitt **Zuweisbar** werden die insgesamt für die einzelnen Knoten verfügbaren Ressourcen in CPU, Arbeitsspeicher und Speicher angezeigt. Im Abschnitt **Zugewiesen** werden die bereits verwendeten Ressourcen angezeigt. Der Wert **13** für die CPU bezieht sich auf **13 Kerne** oder **13.000 (13.000) Millikerne**.

In diesem Beispiel ist der Knoten **überbelegt**, was erklärt, warum der Pod nicht initiieren kann. Nach dem Löschen des ND mit dem Löschen der ND-APPs oder Hinzufügen von VM-Ressourcen.

Der Cluster versucht ständig, ausstehende Richtlinien bereitzustellen. Wenn die Ressourcen also verfügbar sind, können die PODs bereitgestellt werden.

```
[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
```

Mit dem für die Ressourcenüberprüfung verwendeten Befehl wird bestätigt, dass der Cluster über eine verfügbare CPU-Ressource verfügt:

```
[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 ~]$
```

Die Bereitstellungsdetails enthalten eine Meldung mit Informationen zu den aktuellen Bedingungen für 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

Ausführen von Netzwerkdebugbefehlen innerhalb eines Containers

Da die Container nur die minimalen Bibliotheken und Abhängigkeiten enthalten, die für den Pod spezifisch sind, sind die meisten Netzwerk-Debug-Tools (ping, ip route und ip addr) nicht innerhalb des Containers selbst verfügbar.

Diese Befehle sind sehr nützlich, wenn es notwendig ist, Netzwerkprobleme für einen Dienst (zwischen ND-Knoten) oder eine Verbindung zu den Apics zu beheben, da mehrere Mikrodienste mit den Controllern mit der Datenschnittstelle (**bond0** oder **bond0br)** kommunizieren müssen.

Die Fehlermeldung **nsenter** Utility (nur Root-Benutzer) ermöglicht die Ausführung von Netzwerkbefehlen vom ND-Knoten aus, der sich im Container befindet. Suchen Sie dazu die Prozess-ID (PID) aus dem Container, den Sie debuggen möchten. Dies wird mit der Pod K8s ID anhand der lokalen Informationen aus der Container Runtime, wie Docker für ältere Versionen,

und **cri-o** für neuere als Standard.

Prüfen Sie die Pod Kubernetes (K8s) ID.

Aus der Liste der PODs im cisco-mso-Namespace können wir den Container auswählen, in dem die Fehlerbehebung durchgeführt werden soll:

```
[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
```

Die PODs müssen im selben K8s-Knoten ausgeführt werden. Für Produktionsumgebungen können wir die **-o wide** um den Knoten zu ermitteln, der auf jedem Pod ausgeführt wird. Mit der Pod K8s-ID (im vorherigen Ausgabebeispiel fett dargestellt) können wir den Prozess (PID) überprüfen, der von der Container Runtime zugewiesen wurde.

So überprüfen Sie die PID aus der Container-Laufzeit

Die neue Standard-Container-Laufzeit ist CRI-O für Kubernetes. Das Dokument folgt also der Regel für die Befehle. Die von CRI-O zugewiesene Prozess-ID (PID) kann im K8s-Knoten eindeutig sein, was mit dem crictl verwenden.

Die Fehlermeldung **ps** -Option zeigt die ID an, die CRI-O jedem Container zuweist, der den Pod erstellt. Zwei Container für das Beispiel mit Standortdiensten:

```
[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 ~]#
```

Anhand dieser Informationen können wir dann die inspect CRIO-ID um die tatsächliche PID anzuzeigen, die jedem Container zugewiesen wurde. Diese Informationen werden für die nsenter command:

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

Verwenden von nsenter zum Ausführen von Netzwerk-Debugbefehlen in einem Container

Mit der PID aus der obigen Ausgabe können wir als Ziel in der nächsten Befehlssyntax verwenden:

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

Die Fehlermeldung **--net** ermöglicht die Ausführung von Befehlen in den Netzwerk-Namespaces, sodass die Anzahl der verfügbaren Befehle begrenzt ist.

Beispiele:

```
[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
```

Der Ping ist ebenfalls verfügbar, und er testet die Verbindung vom Container nach außen und nicht nur vom K8s-Knoten.

```
[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]
```

So führen Sie Netzwerk-Debugbefehle aus einem Container aus Da die Container nur die minimalen Bibliotheken und Abhängigkeiten enthalten, die für den Pod spezifisch sind, sind die meisten Netzwerk-Debugtools (ping, ip route und ip addr) nicht im Container selbst verfügbar. Diese Befehle sind sehr nützlich, wenn es notwendig ist, Netzwerkprobleme für einen Dienst (zwischen ND-Knoten) oder eine Verbindung zu den Apics zu beheben, da mehrere Mikrodienste mit den Controllern mit der Datenschnittstelle (bond0 oder bond0br) kommunizieren müssen. Mit dem Dienstprogramm nsenter (nur root-Benutzer) können Sie Netzwerkbefehle vom ND-Knoten ausführen, der sich im Container befindet. Suchen Sie dazu die Prozess-ID (PID) aus dem Container, den Sie debuggen möchten. Dies geschieht mit der Pod K8s ID gegen die lokalen Informationen aus der Container Runtime, wie Docker für ältere Versionen und cri-o für neuere als Standard. Inspizieren Sie die Pod Kubernetes (K8s) ID Aus der Liste der PoDs im cisco-mso Namespace können wir den Container auswählen, für den eine Fehlerbehebung durchgeführt

werden soll: [root@MxNDsh01 ~]# kubectl get pod -n cisco-msoNAME READY STATUS RESTARTS AGEconsistencyservice-569bdf5969-xkwpg 3 /3 mit 0 9heeworker-65dc5dd849-485tq 2/2 mit 0 163mendpointservice-5db6f57884-hkf5g 2/2 mit 0 9hexecutionservice-6c4894d4f7-p8fzk 2/2 mit 0 9hsiteservice-64dfcdf658-lvbr4 3/3 Running 0 9hsiteupgrade-68bcf987cc-ttn7h 2/2 Running 0 9h Die PODs müssen im selben K8s Node laufen. Für Produktionsumgebungen können wir die Option -o wide am Ende hinzufügen, um den Knoten zu ermitteln, der auf jedem Pod ausgeführt wird. Mit der Pod K8s-ID (im vorherigen Ausgabebeispiel fett dargestellt) können wir den Prozess (PID) überprüfen, der von der Container Runtime zugewiesen wurde. Wie die PID aus der Container-Laufzeit überprüft wird Die neue Standard-Container-Laufzeit ist CRI-O für Kubernetes. Das Dokument folgt also der Regel für die Befehle. Die von CRI-O zugewiesene Prozess-ID (PID) kann im K8s-Knoten eindeutig sein, was mit dem Dienstprogramm "crictl" erkannt werden kann. Die Option ps zeigt die ID an, die CRI-O jedem Container zuweist, der den Pod erstellt. Zwei davon sind für das Beispiel des Standortdiensts: [root@MxNDsh01 ~]# crictl ps |grep siteservicefb560763b06f2 172.31.0.0:30012/cisco-mso/sslcontainer@sha256:2d788fa493c885ba8c9e5944596b864d090d9051b0eab82123ee4d19596279c9 Vor 10 Stunden Ausführen msc-siteservice2-ssl 0 074727b4e9f51ad2d42aae1ad91d0195292f7fcc62f38529e135a131 5c358067004a086cfed7e059986ce615b0 vor 10 Stunden Running siteservice-leader-election 0 074727b4e9f5129b0b6d41d1e3 vor 172.31.0.0:30012/cisco-mso/siteservice@sha256:80a2335bcd5366952b4d60a275b20c70de0bb65a47bf8ae6d988f07b1e0bf494 vor 10 Stunden Running siteservice 0 074727b4e9f 51[root@MxNDsh01 ~]# Mit diesen Informationen können wir dann die Option "inspect CRIO-ID" verwenden, um die tatsächliche PID anzuzeigen, die jedem Container zugewiesen wurde. Diese Informationen werden für den nsenter-Befehl benötigt: [root@MxNDsh01 ~]# crictl inspect fb560763b06f2| grep -i pid"pid": 239563,"pids": {"type": "pid" Wie man nsenter verwendet, um Netzwerk-Debug-Befehle in einem Container auszuführen Mit der PID aus der obigen Ausgabe können wir als Ziel in der nächsten Befehlssyntax verwenden: nsenter —target <PID> —net <NETZWERKBEFEHL> Die Option —net ermöglicht die Ausführung von Befehlen in den Netzwerk-Namespaces, sodass die Anzahl der verfügbaren Befehle begrenzt ist. Beispiel: [root@MxNDsh01 ~]# nsenter —target 239563 —net ifconfigure0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450inet 172.17.248.146 netmask 255.255.0.0 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 (Ethernet)RX Pakete 916346 Byte 271080553 (258.0) 5 MiB)RX-Fehler 0 Drop 183 Overruns 0 Frame 0TX-Pakete 828016 Bytes 307255950 (293,0 MiB)TX-Fehler 0 Drop 0 Overruns 0 Carrier 0 Collisions 0lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536inet 127.0.0.1 netmask 25 1.0.0.0inet6 ::1 prefixlen 128 scopeid 0x10<host>loop txqueuelen 1000 (Lokales Loopback)RX Pakete 42289 Bytes 14186082 (13,5 MiB)RX Fehler 0 fallen gelassen 0 Überläufe 0 Frame 0TX Pakete 42289 Bytes 14186082 (13,5 MiB)TX Fehler 0 fallen gelassen 0 überläuft 0 Carrier 0 Kollisionen 0 Der Ping ist ebenfalls verfügbar, und er testet die Verbindung vom Container nach außen, anstatt nur den K8s-Knoten. [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.WARNUNG: Zertifikat von 1xx.2xx.3xx.4xx kann nicht verifiziert werden, ausgestellt von ‘/C=US/ST=CA/O=Cisco System/CN=APIC":Die Autorität des Herausgebers kann nicht lokal überprüft werden.WARNUNG: Der allgemeine Zertifikatname "APIC" stimmt nicht mit dem angeforderten Hostnamen "1xx.2xx.3xx.4xx" überein.HTTP-Anfrage gesendet, wartet auf Antwort... 200 OKLänge: 3251 (3,2K) [text/html]Speichern unter: index.html’100%[==========>] 3.251 —.-K/s in 0s2023-01-24 23:46:04 (548 MB/s) - "index.html" gespeichert [3251/3251]

Informationen zu dieser Übersetzung

Cisco hat dieses Dokument maschinell übersetzen und von einem menschlichen Übersetzer editieren und korrigieren lassen, um unseren Benutzern auf der ganzen Welt Support-Inhalte in ihrer eigenen Sprache zu bieten. Bitte beachten Sie, dass selbst die beste maschinelle Übersetzung nicht so genau ist wie eine von einem professionellen Übersetzer angefertigte. Cisco Systems, Inc. übernimmt keine Haftung für die Richtigkeit dieser Übersetzungen und empfiehlt, immer das englische Originaldokument (siehe bereitgestellter Link) heranzuziehen.