Solución de problemas y revisión de recursos NDO

Contenido

Introducción

Este documento describe cómo revisar y resolver problemas de NDO con la CLI de tiempo de ejecución de contenedor y kubectl.

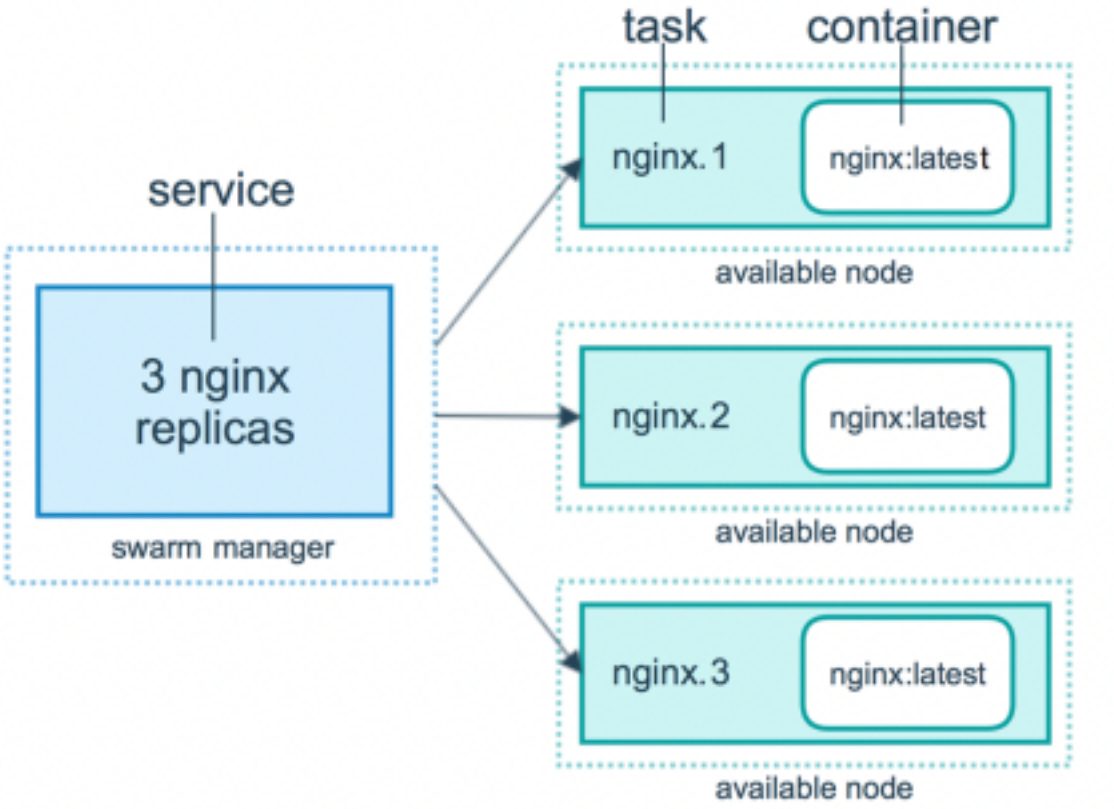
QuickStart de NDO

Cisco Nexus Dashboard Orchestrator (NDO) es una herramienta administrativa de fabric que permite a los usuarios gestionar distintos tipos de fabrics, entre los que se incluyen sitios de Cisco® Application Centric Infrastructure (Cisco ACI®), sitios de Cisco Cloud ACI y sitios de Cisco Nexus Dashboard Fabric Controller (NDFC). Cada uno de ellos se gestiona mediante su propio controlador (clúster APIC, clúster NDFC o instancias de Cloud APIC en una nube pública).

NDO proporciona una orquestación de políticas y redes, escalabilidad y recuperación ante desastres uniformes en varios Data Centers a través de un único panel.

En los primeros días, el MSC (Multi-Site Controller) se implementaba como un clúster de tres nodos con VMWare Open Virtual Appliances (OVA) que permitía a los clientes inicializar un clúster Docker Swarm y los servicios MSC. Este clúster de Swarm administra los microservicios de MSC como contenedores y servicios de Docker.

Esta imagen muestra una vista simplificada de cómo Docker Swarm administra los microservicios como réplicas del mismo contenedor para lograr una alta disponibilidad.



El Docker Swarm fue responsable de mantener el número esperado de réplicas para cada uno de los microservicios en la arquitectura MSC. Desde el punto de vista de Docker Swarm, el controlador multisitio fue la única implementación de contenedor que se pudo organizar.

Nexus Dashboard (ND) es una consola de gestión central para varios sitios de Data Centers y una plataforma común que aloja los servicios operativos de Cisco Data Center, que incluyen Nexus Insight y MSC versión 3.3 en adelante, y cuyo nombre ha cambiado a Nexus Dashboard Orchestrator (NDO).

Aunque la mayoría de los microservicios que componen la arquitectura MSC siguen siendo los mismos, NDO se implementa en un clúster de Kubernetes (K8s) en lugar de en uno de Docker Swarm. Esto permite a ND organizar varias aplicaciones o implementaciones en lugar de solo una.

Curso intensivo de Kubernetes con NDO

Kubernetes es un sistema de código abierto para automatizar la implementación, la escalabilidad y la gestión de aplicaciones en contenedores. Como Docker, Kubernetes funciona con la tecnología de contenedores, pero no está vinculado con Docker. Esto significa que Kubernetes soporta otras plataformas de contenedores (Rkt, PodMan).

Una diferencia clave entre Swarm y Kubernetes es que este último no funciona con contenedores directamente, sino que funciona con un concepto de grupos de contenedores co-ubicados, llamados Pods, en su lugar.

Los contenedores de un grupo de dispositivos deben ejecutarse en el mismo nodo. Un grupo de grupos de dispositivos se denomina Implementación. Una implementación de Kubernetes puede describir una aplicación completa.

Kubernetes también permite a los usuarios asegurarse de que una cierta cantidad de recursos

están disponibles para cualquier aplicación dada. Esto se hace con el uso de los controladores de replicación, para garantizar que el número de grupos de dispositivos sea coherente con los manifiestos de aplicación.

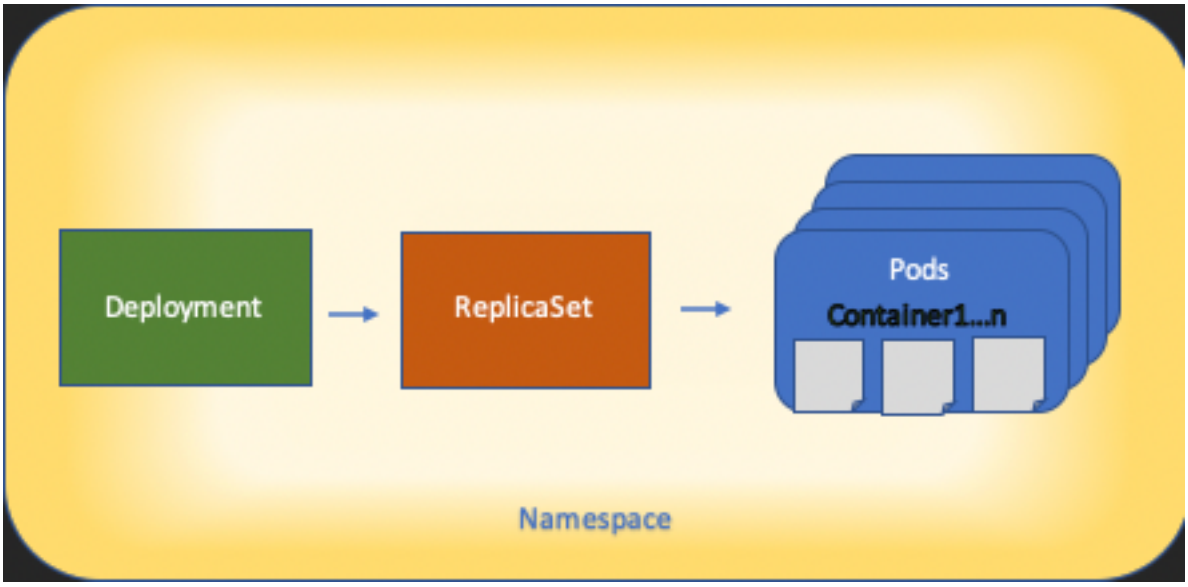
Un manifiesto es un archivo con formato YAML que describe un recurso que va a implementar el clúster. El recurso puede ser cualquiera de los descritos anteriormente u otros disponibles para los usuarios.

Se puede acceder a la aplicación externamente con uno o más servicios. Kubernetes incluye una opción de equilibrador de carga para lograr esto.

Kubernetes también ofrece una manera de aislar diferentes recursos con el concepto de espacios de nombres. ND utiliza espacios de nombres para identificar de forma única diferentes aplicaciones y servicios de clúster. Cuando se ejecutan comandos CLI, especifique siempre el espacio de nombres.

Aunque no se requiere un conocimiento profundo de Kubernetes para resolver problemas de ND o NDO, se requiere una comprensión básica de la arquitectura de Kubernetes para identificar adecuadamente los recursos con problemas o que necesitan atención.

Los fundamentos de la arquitectura de recursos de Kubernetes se muestran en este diagrama:



Es importante recordar cómo cada tipo de recurso interactúa con los demás y desempeña un papel importante en el proceso de revisión y solución de problemas.

Descripción general de NDO con comandos de Kubernetes

Inicio de sesión de acceso CLI

Para el acceso CLI mediante SSH a NDO, el **admin-user** se necesita una contraseña. Sin embargo, en su lugar utilizamos la **rescue-user** contraseña. Como en:

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

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

Este es el modo y el usuario predeterminados para el acceso CLI y la mayor parte de la información está disponible para ver.

Revisión de espacios de nombres NDO

Este concepto de K8 permite el aislamiento de diferentes recursos a través del clúster. El siguiente comando se puede utilizar para revisar los diferentes espacios de nombres implementados:

```
[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
```

Las entradas en negrita pertenecen a Aplicaciones en el NDO, mientras que las entidades que comienzan con el prefijo **kube** pertenecen al clúster Kubernetes. Cada espacio de nombres tiene sus propias implementaciones y grupos de dispositivos independientes

La CLI kubectl permite especificar un espacio de nombres con el **--namespace** , si un comando se ejecuta sin él, la CLI asume que el espacio de nombres es **default** (Espacio de nombres para k8s):

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

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

La CLI de kubectl permite diferentes tipos de formatos para la salida, como yaml, JSON o una

tabla hecha a medida. Esto se consigue con el **-o** opción [format]. Por ejemplo:

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

{

   "apiVersion": "v1",

   "items": [

       {

           "apiVersion": "v1",

           "kind": "Namespace",

           "metadata": {

               "annotations": {

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

               },

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

               "labels": {

                   "serviceType": "infra"

               },

               "name": "authy",

               "resourceVersion": "826",

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

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

           },

       }

   ],

   "kind": "List",

   "metadata": {

       "resourceVersion": "",

       "selfLink": ""

   }

}
```

A partir del texto anterior, el resultado es un diccionario donde una de sus claves se llama **items** y el valor es una **lista** de diccionarios donde cada **diccionario** representa una entrada de espacio de

nombres y sus atributos son valores de pareja clave-valor en el diccionario o diccionarios anidados.

Esto es relevante porque K8s proporciona a los usuarios la opción de seleccionar jsonpath como la salida, esto permite operaciones complejas para una matriz de datos JSON. Por ejemplo, desde la salida anterior, si accedemos al valor de **name** para los espacios de nombres, necesitamos acceder al valor de la lista de elementos y, a continuación, el metadata diccionario, y obtener el valor de la clave **name**. Esto se puede hacer con este comando:

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

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

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

La jerarquía descrita se utiliza para obtener la información específica necesaria. Básicamente, se accede a todos los elementos en el **items** con **items[*]**, luego la tecla metadata y **name** con **metadata.name**, la consulta puede incluir otros valores para mostrar.

Lo mismo se aplica a la opción de columnas personalizadas, que utilizan una forma similar de obtener la información de la matriz de datos. Por ejemplo, si creamos una tabla con la información sobre el **name** y el UID , podemos aplicar el comando:

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

NAME                 UID

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

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

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

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

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

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

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

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

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

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

El resultado requiere un nombre para cada columna que se va a mostrar y, a continuación, asignar el valor del resultado. En este ejemplo, hay dos columnas: **NAME** y **UID**. Estos valores pertenecen a .metada.name y .metadata.uid respectivamente. Puede encontrar más información y

ejemplos en:

Compatibilidad con JSONPath

Columnas personalizadas

Revisión de implementación de NDO

Una implementación es un objeto K8s que proporciona un espacio combinado para administrar ReplicaSet y Pods. Las implementaciones se ocupan de la implementación de todos los grupos de dispositivos que pertenecen a una aplicación y del número esperado de copias de cada uno.

La CLI de kubectl incluye un comando para verificar las implementaciones para cualquier espacio de nombres dado:

```
[rescue-user@MxNDsh01 ~]$ kubectl get deployment -n cisco-mso
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
auditservice         1/1     1            1           3d22h
backupservice        1/1     1            1           3d22h
cloudsecservice      1/1     1            1           3d22h
consistencyservice   1/1     1            1           3d22h
dcnmworker           1/1     1            1           3d22h
eeworker             1/1     1            1           3d22h
endpointservice      1/1     1            1           3d22h
executionservice     1/1     1            1           3d22h
fluentd              1/1     1            1           3d22h
importservice        1/1     1            1           3d22h
jobschedulerservice  1/1     1            1           3d22h
notifyservice        1/1     1            1           3d22h
pctagvnidservice     1/1     1            1           3d22h
platformservice      1/1     1            1           3d22h
platformservice2     1/1     1            1           3d22h
policyservice        1/1     1            1           3d22h
schemaservice        1/1     1            1           3d22h
sdaservice           1/1     1            1           3d22h
sdwanservice         1/1     1            1           3d22h
siteservice          1/1     1            1           3d22h
```

```
siteupgrade           1/1     1            1           3d22h

syncengine            1/1     1            1           3d22h

templateeng           1/1     1            1           3d22h

ui                    1/1     1            1           3d22h

userservice           1/1     1            1           3d22h
```

Podemos utilizar la misma tabla personalizada con el uso de deployment en lugar de **namespace** y el **-n** para ver la misma información que antes. Esto se debe a que la salida está estructurada de manera similar.

```
[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
```

Tenga en cuenta que el número de copias mostradas es para la implementación, no el número de vainas para cada microservicio.

Podemos usar la palabra clave **describe** en lugar de **get** para mostrar información más detallada sobre un recurso, en este caso, despliegue de schemaservice:

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

Name:               schemaservice

Namespace:          cisco-mso

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

Labels:             k8s-app=schemaservice

                   scaling.case.cncf.io=scale-service

Annotations:        deployment.kubernetes.io/revision: 1

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

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

Selector:           k8s-app=schemaservice

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

StrategyType:       Recreate

MinReadySeconds:    0

Pod Template:

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

                   k8s-app=schemaservice

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

 Service Account:  cisco-mso-sa

 Init Containers:

  init-msc:

   Image:      cisco-mso/tools:3.7.1j

   Port:       <none>

   Host Port:  <none>

   Command:

     /check_mongo.sh
```

```
    Environment:  <none>
    Mounts:
      /secrets from infracerts (rw)
  Containers:
   schemaservice:
    Image:       cisco-mso/schemaservice:3.7.1j
    Ports:       8080/TCP, 8080/UDP
    Host Ports:  0/TCP, 0/UDP
    Command:
      /launchscala.sh
      schemaservice
    Liveness:  http-get http://:8080/api/v1/schemas/health delay=300s timeout=20s period=30s
#success=1 #failure=3
    Environment:
      JAVA_OPTS:  -XX:+IdleTuningGcOnIdle
    Mounts:
      /jwtsecrets from jwtsecrets (rw)
      /logs from logs (rw)
      /secrets from infracerts (rw)
   msc-schemaservice-ssl:
    Image:       cisco-mso/sslcontainer:3.7.1j
    Ports:       443/UDP, 443/TCP
    Host Ports:  0/UDP, 0/TCP
    Command:
      /wrapper.sh
    Environment:
      SERVICE_PORT:  8080
    Mounts:
      /logs from logs (rw)
      /secrets from infracerts (rw)
   schemaservice-leader-election:
    Image:      cisco-mso/tools:3.7.1j
```

```
    Port:       <none>
    Host Port:  <none>
    Command:
      /start_election.sh
    Environment:
      SERVICENAME:  schemaservice
    Mounts:
      /logs from logs (rw)
 Volumes:
  logs:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same
namespace)
    ClaimName:  mso-logging
    ReadOnly:   false
  infracerts:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  cisco-mso-secret-infra
    Optional:    false
  jwtsecrets:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  cisco-mso-secret-jwt
    Optional:    false
Conditions:
 Type          Status  Reason
 ----          ------  ------
 Available     True    MinimumReplicasAvailable
 Progressing   True    NewReplicaSetAvailable
Events:         <none>
[rescue-user@MxNDsh01 ~]$
```

describe también permite la inclusión de la **--show-events=true** para mostrar cualquier evento relevante para la implementación.

Deflector

Revisión del conjunto de réplicas NDO (RS)

Deflector

ESTO SOLO ESTÁ DISPONIBLE PARA EL USUARIO RAÍZ

Un conjunto de réplicas (RS) es un objeto K8s con el objetivo de mantener un número estable de grupos de réplicas. Este objeto también detecta cuando se ve un número inestable de réplicas con una sonda periódica a las vainas.

Los RR también se organizan en espacios de nombres.

```
[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
```

describe incluye la información sobre la URL, el puerto que utiliza la sonda y la periodicidad de las pruebas y el umbral de falla.

```
[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 #### THIS IS ONLY AVAILABLE FOR ROOT USER ##### Un conjunto de réplicas (RS) es un objeto K8s con el objetivo de mantener un número estable de grupos de réplicas. Este objeto también detecta cuando se ve un número inestable de réplicas con una sonda periódica a las vainas. Los RR también se organizan en espacios de nombres. [root@MxNDsh01 ~]# kubectl get rs -n cisco-msoNAME DESIRED CURRENT READY AGEauditservice-648cd4c6f8 1 1 3d22hbackupservice-64b755b44c 1 1 1 3d22hcloudsecservice-7df465576 1 1 1 3d22hconsistencyservice-c98955599 1 1 1 3d22hdcnmworker-5d4d5cbb6 4 1 1 1 3d22heeworker-56f9fb9ddb 1 1 1 3d22hendpointservice-7df9d599c 1 1 1 3d22hexecute service-58ff89595f 1 1 1 3d22hfluentd-86785f89bd 1 1 1 3d22himportservice-88bcc8547 1 1 1 3d22hobschedulerservice 5d4fdfd696 1 1 3d22hnotifyservice-75c988cfd4 1 1 3d22hpctagvnidservice-644b755596 1 1 1 3d22hplatformservice-65cddb946f 1 1 1 3d22hplatformservice2-6796576659 1 1 1 3d22hpolicyservice-545b9c7d9c 1 1 1 d22hschemaservice-7597ff4c5 1 1 3d22hsdaservice-5f477dd8c7 1 1 3d22hsdwanservice-6f87cd999d 1 1 1 3d22hsiteservice-86bb756585 1 1 1 3d22hsiteupgrade-7d578f9b6d 1 1 1 3d22hsyncsyncsync engine-5b8bdd6b45 1 1 1 3d22htemplateeng-5cbf9fdc48 1 1 1 3d22hui-

84588b7c96 1 1 1 3d22huserservice-87846f7c6 1 1 1 3d22h La opción describe incluye la información sobre la URL, el puerto que utiliza la sonda y la periodicidad de las pruebas y el umbral de error. [root@MxNDsh01 ~]# kubectl describe rs -n cisco-mso schemaservice-7597ff4c5Nombre: schemaservice-7597ff4c5Espacio de nombres: cisco-msoSelector: k8s-app=schemaservice,pod-template-hash=7597ff4c5Etiquetas: cpu.resource.case.cncf.io/schemaservice=cpu-lg-service k8s-app=schemaservice memory.resource.case.cncf.io/schemaservice=mem-xlg-service pod-template-hash=7597ff4c5Anotaciones: deployment.kubernetes.io/desired-replicas: 1 deployment.kubernetes.io/max-replicas: 1Controlado por Deployment/schemaserviceReplicas: 1 current / 1 desiredPods Status: 1 Running / 0 Waiting / 0 Succeeded / 0 FailedPod Template: Labels: deployment.kubernetes.io/revision k8s-app=schemaservice cpu.resource.case.cncf.io/schemaservice=cpu-lg-service pod-template-hash=7597ff4c5 Service Account: cisco-mso-sa Init Containers: init-msc: Image: cisco-mso/tools:3.7.1j Port: <none> Host Port: <none> Command: memory.resource.case.cncf.io/schemaservice=mem-xlg-service Environment: <none> Mounts: /secret from infracerts (rw) Contenedores: schemaservice: Image: cisco-mso/schemaservice:3.7.1j Puertos: 8080/TCP, 8080/UDP Puertos de host: 0/TCP, 0/UDP Comando: /check_mongo.sh schemaservice Vida: http-get /launchscala.sh delay=300s timeout=20s period=30s #success=1 #failure=3 Entorno: JAVA_OPTS: -XX:+IdleTuningGcOnIdle Mounts: /jwtsecretos de jwtsecretos (rw)) /logs de logs (rw) /secret from infracerts (rw) msc-schemaservice-ssl: Imagen: cisco-mso/sslcontainer:3.7.1j Puertos: 443/UDP, 443/TCP Puertos de host: 0/UDP, 0/TCP Comando: http://:8080/api/v1/schemas/health /wrapper.sh

Revisión de POD NDO

Un Pod es un grupo de contenedores estrechamente relacionados que se ejecutan en el mismo espacio de nombres de Linux (diferente del espacio de nombres K8s) y en el mismo nodo K8s. Este es el objeto atómico más manejado por K8, ya que no interactúa con contenedores. La aplicación puede constar de un solo contenedor o ser más compleja con muchos contenedores. Con el siguiente comando, podemos verificar los Pods de cualquier espacio de nombres dado:

```
[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 ~]$
```

El número que aparece en la segunda columna se refiere al número de contenedores de cada grupo de dispositivos.

describe también está disponible, que incluye información detallada sobre los contenedores de cada POD.

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

Name:         schemaservice-7597ff4c5-w4x5d

Namespace:    cisco-mso

Priority:     0

Node:         mxndsh01/172.31.0.0

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

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

              k8s-app=schemaservice

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

              pod-template-hash=7597ff4c5

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

                [{
```

```
                    "name": "default",
                    "interface": "eth0",
                    "ips": [
                        "172.17.248.16"
                    ],
                    "mac": "3e:a2:bd:ba:1c:38",
                    "dns": {}
                }]
              kubernetes.io/psp: infra-privilege
Status:       Running
IP:           172.17.248.16
IPs:
 IP:           172.17.248.16
Controlled By:  ReplicaSet/schemaservice-7597ff4c5
Init Containers:
 init-msc:
   Container ID:  cri-o://0c700f4e56a6c414510edcb62b779c7118fab9c1406fdac49e742136db4efbb8
   Image:         cisco-mso/tools:3.7.1j
   Image ID:      172.31.0.0:30012/cisco-
mso/tools@sha256:3ee91e069b9bda027d53425e0f1261a5b992dbe2e85290dfca67b6f366410425
   Port:          <none>
   Host Port:     <none>
   Command:
     /check_mongo.sh
   State:          Terminated
     Reason:       Completed
     Exit Code:    0
     Started:      Tue, 20 Sep 2022 02:05:39 +0000
     Finished:     Tue, 20 Sep 2022 02:06:24 +0000
   Ready:          True
   Restart Count:  0
   Environment:    <none>
   Mounts:
```

```
     /secrets from infracerts (rw)
     /var/run/secrets/kubernetes.io/serviceaccount from cisco-mso-sa-token-tn45l (ro)
Containers:
 schemaservice:
   Container ID:  cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac
   Image:         cisco-mso/schemaservice:3.7.1j
   Image ID:      172.31.0.0:30012/cisco-
mso/schemaservice@sha256:6d9fae07731cd2dcaf17c04742d2d4a7f9c82f1fc743fd836fe59801a21d985c
   Ports:         8080/TCP, 8080/UDP
   Host Ports:    0/TCP, 0/UDP
   Command:
     /launchscala.sh
     schemaservice
   State:          Running
     Started:      Tue, 20 Sep 2022 02:06:27 +0000
   Ready:          True
   Restart Count:  0
   Limits:
     cpu:     8
     memory:  30Gi
   Requests:
     cpu:     500m
     memory:  2Gi
```

La información mostrada incluye la imagen del contenedor de cada contenedor y muestra el tiempo de ejecución del contenedor utilizado. En este caso, CRI-O (**cri-o**), versiones anteriores de ND utilizadas para trabajar con Docker, esto influye en cómo adjuntar a un contenedor.

[Deflector]

Por ejemplo, cuando **cri-o** se utiliza, y queremos conectar mediante una sesión interactiva a un contenedor (a través de la exec -it) al contenedor de la salida anterior; pero en lugar de la opción docker , utilizamos el comando **crictl**:

```
schemaservice:
   Container ID:  cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac
   Image:         cisco-mso/schemaservice:3.7.1j
```

Utilizamos este comando:

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

root@schemaservice-7597ff4c5-w4x5d:/#

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

root
```

Para versiones posteriores de ND, el ID de contenedor que se utilizará es diferente. En primer lugar, tenemos que utilizar el comando **crictl ps** para mostrar todos los contenedores que se ejecutan en cada nodo. Podemos filtrar el resultado según sea necesario.

```
[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 ~]#
```

Con el valor de la primera columna, podemos acceder al Contenedor en tiempo de ejecución con el mismo comando que antes:

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

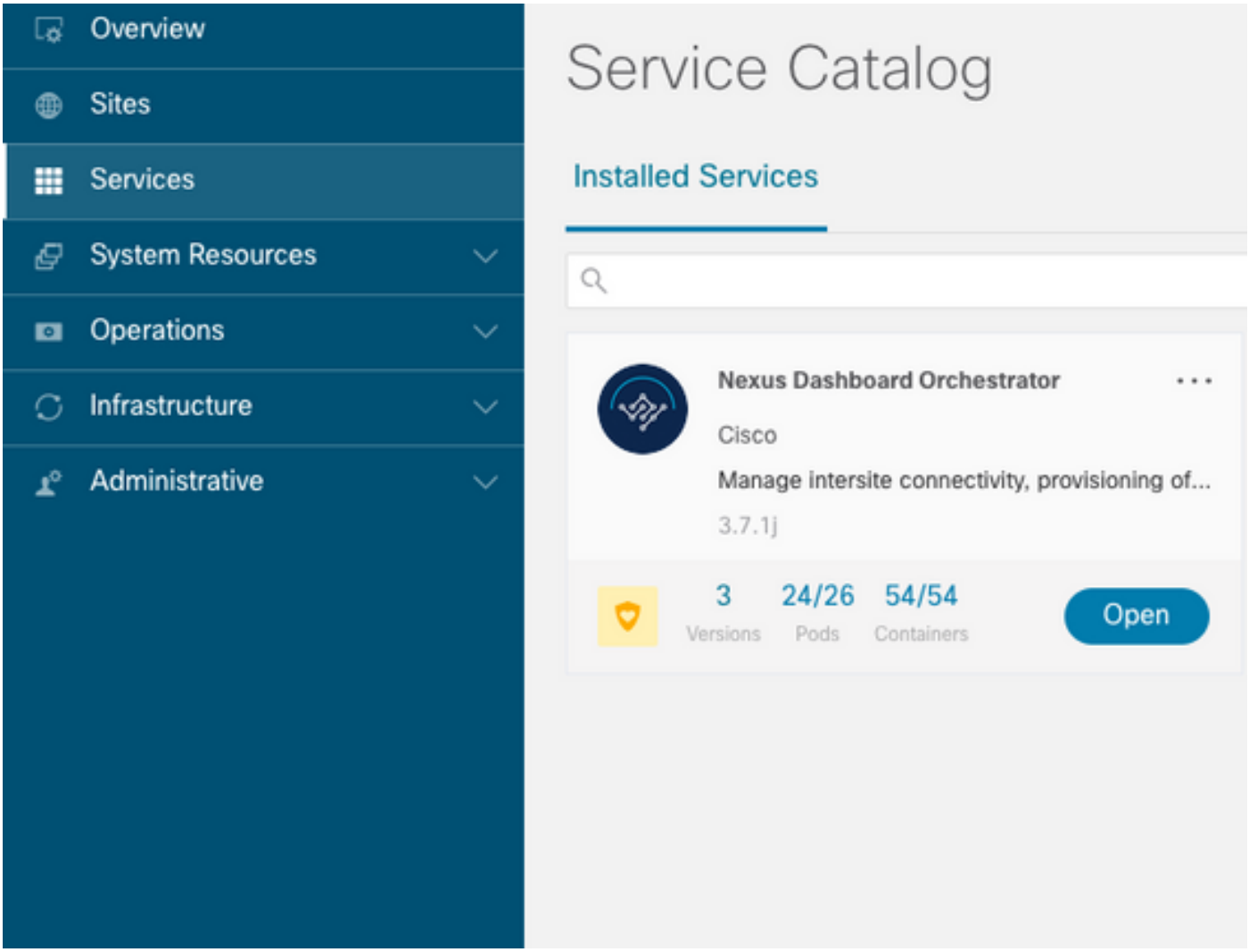
Por ejemplo, cuando se utiliza cri-o, y queremos conectar mediante una sesión interactiva a un contenedor (mediante la opción exec -it) al contenedor de la salida anterior; pero en lugar del comando docker, utilizamos el comando criclt: schemaservice: Container ID: cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac Image: cisco-mso/schemaservice:3.7.1j Usamos este comando: [root@MxNDsh01 ~]# crictl exec -it d2287f8659it6848c0100j b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac bashroot@schemaservice-7597ff4c5-w4x5d:/#root@schemaservice-7597ff4c5-w4x5d:/# whoamiroot Para versiones posteriores de ND, el ID de contenedor que se utilizará es diferente. En primer lugar, necesitamos utilizar el comando crictl ps para enumerar todos los contenedores que se ejecutan en cada nodo. Podemos filtrar el resultado según sea necesario. [root@singleNode ~]# crictl ps| grep backupa9bb161d67295 10.31.125.241:30012/cisco-mso/sslcontainer@sha256:26581eebd0bd6f4378a5fe4a98973dbda417c1905689f71f229765621f0cee75 Hace 2 días que ejecuta msc-backupservice-ssl 0 84b3c691cfc2b4b26f67fc10cf 10.31.125.241:30012/cisco-mso/backupservice@sha256:c21f4cdde696a5f2dfa7bb910b7278fc3fb4d46b02f42c3554f872ca8c87c061 Hace 2 días Ejecutando backupservice 0 84b3c691cfc2b[root@singleNode ~]# Con el valor de la primera columna, podemos acceder al contenedor en tiempo de ejecución con el mismo comando que antes: [root@singleNode ~]# crictl exec -it 4b26f67fc10cf bashroot@backupservice-8c699779f-j9jtr:/# pwd/

La POD del caso práctico no es saludable

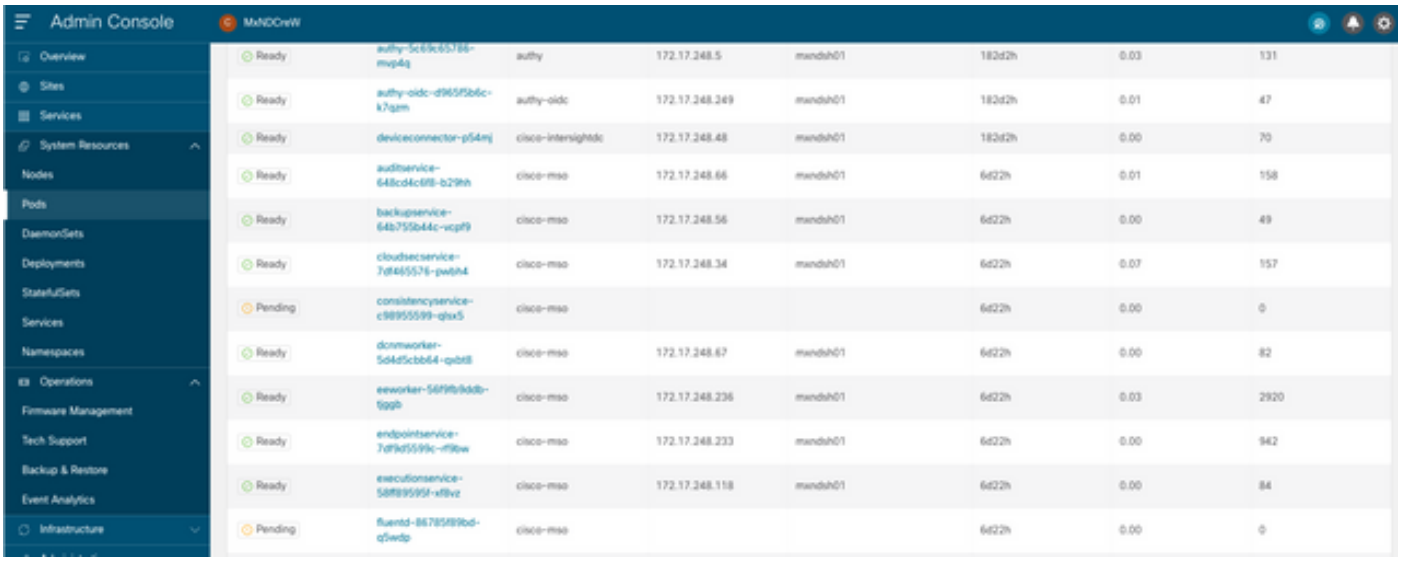
Podemos utilizar esta información para resolver los problemas por los que los dispositivos de una

implementación no están en buen estado. En este ejemplo, la versión del panel de Nexus es 2.2-1d y la aplicación afectada es Nexus Dashboard Orchestrator (NDO).

La GUI de NDO muestra un conjunto incompleto de grupos de dispositivos desde la vista de servicio. En este caso, 24 de 26 grupos.



Otra vista disponible en el **System Resources -> Pods** Ver dónde los Pods muestran un estado diferente de **Ready**.



Solución de problemas de CLI para grupos de dispositivos inadecuados

Con el hecho conocido de que el espacio de nombres es cisco-mso (aunque cuando se solucionan problemas, es lo mismo para otras aplicaciones/espacios de nombres), la vista de POD muestra si hay alguna que no funcione correctamente:

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

Para este ejemplo, nos centramos en los Pods de servicio de consistencia. A partir de la salida de JSON, podemos obtener la información específica de los campos de estado, con el uso de jsonpath:

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

Vemos el diccionario de **estado** y dentro de una lista llamada **conditions** con diccionarios como elementos con el **mensaje** keys y el **valor**, la parte {"\n"} es crear una nueva línea al final:

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

Este comando muestra cómo comprobar desde el **get Pod** para el espacio de nombres:

```
[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
```

Con el get pods , podemos obtener el ID de POD con problemas que deben coincidir con el del resultado anterior. En este ejemplo **consistencyservice-c98955599-qlsx5**.

El formato de salida JSON también proporciona cómo comprobar información específica, a partir de la salida dada.

```
[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 ~]$
```

La salida JSON debe incluir información sobre el estado en el atributo con el mismo nombre. El mensaje incluye información sobre el motivo.

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

Podemos acceder a la información sobre el estado y los requisitos de los dispositivos:

```
[rescue-user@MxNDsh01 ~]$ kubectl get pods -n cisco-mso consistencyservice-c98955599-qlsx5 -
```

```
o=jsonpath='{.spec.containers[*].resources.requests}{"\n"}'
map[cpu:500m memory:1Gi]
```

Aquí es importante mencionar cómo se calcula el valor. En este ejemplo, la cpu **500m** se refiere a **500 milicores**, y el **1G** en la memoria es para GB.

Describe para el nodo muestra el recurso disponible para cada trabajador K8 del clúster (host o VM):

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

La sección **Asignable** muestra el total de Recursos en CPU , Memoria y Almacenamiento disponibles para cada nodo. La sección **Asignado** muestra los recursos que ya están en uso. El valor **13** para CPU se refiere a **13 núcleos** o **13 000 (13 000) milicores**.

Para este ejemplo, el nodo está **sobresuscrito**, lo que explica por qué el Pod no puede iniciarse. Después de borrar el ND con la eliminación de las aplicaciones ND o la adición de los recursos de VM.

El clúster intenta constantemente implementar las políticas pendientes, por lo que si los recursos están libres, se pueden implementar los grupos de dispositivos.

```
[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
```

Con el comando utilizado para la comprobación de recursos, se confirma que el clúster tiene recursos disponibles para la CPU:

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

Los detalles de la implementación incluyen un mensaje con información sobre las condiciones actuales de los grupos de dispositivos:

```
[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 ~]$
```

Deflector

Cómo ejecutar comandos de depuración de red desde dentro de un contenedor

Dado que los contenedores sólo incluyen las bibliotecas y dependencias mínimas específicas para el Pod, la mayoría de las herramientas de depuración de red (ping, ruta ip y dirección ip) no están disponibles dentro del propio contenedor.

Estos comandos son muy útiles cuando hay una necesidad de resolver problemas de red para un servicio (entre nodos ND) o conexión hacia los Apic porque varios microservicios necesitan comunicarse con los controladores con la interfaz de datos (**bond0** o **bond0br**).

nsenter (sólo usuario raíz) nos permite ejecutar comandos de red desde el nodo ND tal como está dentro del contenedor. Para ello, busque el ID de proceso (PID) del contenedor que desea depurar. Esto se logra con el ID de Pod K8s contra la información local de Container Runtime, como Docker para versiones antiguas, y **cri-o** para los más nuevos como opción predeterminada.

Inspeccionar la ID de Pod Kubernetes (K8s)

De la lista de Pods dentro del espacio de nombres cisco-mso, podemos seleccionar el contenedor para resolver problemas:

```
[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
```

Los Pods deben ejecutarse en el mismo nodo K8s. Para los entornos de producción, podemos añadir la **-o wide** al final para averiguar el nodo que ejecuta cada POD. Con el ID de Pod K8s (en negrita en el ejemplo de salida anterior) podemos verificar el Proceso (PID) asignado por Container Runtime.

Cómo inspeccionar el PID desde el tiempo de ejecución del contenedor

El nuevo Container Runtime predeterminado es CRI-O para Kubernetes. Así que el documento viene después de esa regla para los comandos. El ID de proceso (PID) asignado por CRI-O puede ser único en el nodo K8s, que se puede detectar con el crictl utilidad.

ps revela el ID dado por CRI-O a cada contenedor que construye el POD, dos para el ejemplo de sitesevice:

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

Con esta información, podemos utilizar el inspect CRIO-ID opción para ver el PID real dado a cada contenedor. Esta información es necesaria para el nsenter comando:

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

Cómo utilizar Insenter para ejecutar comandos de depuración de red dentro de un contenedor

Con el PID del resultado anterior, podemos utilizar como destino en la siguiente sintaxis de comandos:

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

--net permite ejecutar comandos en los espacios de nombres de red, por lo que el número de comandos disponibles es limitado.

Por ejemplo:

```
[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
```

El ping también está disponible y prueba la conectividad desde el contenedor hacia el exterior, en lugar de solamente el nodo K8.

```
[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]
```

Cómo ejecutar comandos de depuración de red desde dentro de un contenedor Dado que los contenedores sólo incluyen las bibliotecas y dependencias mínimas específicas para el grupo de dispositivos, la mayoría de las herramientas de depuración de red (ping, ruta ip y dirección ip) no están disponibles dentro del propio contenedor. Estos comandos son muy útiles cuando hay una necesidad de resolver problemas de red para un servicio (entre nodos ND) o conexión hacia los Apic porque varios microservicios necesitan comunicarse con los controladores con la interfaz de datos (bond0 o bond0br). La utilidad nsenter (sólo usuario raíz) nos permite ejecutar comandos de red desde el nodo ND, ya que se encuentra dentro del contenedor. Para ello, busque el ID de proceso (PID) del contenedor que desea depurar. Esto se logra con el ID de Pod K8s contra la información local de Container Runtime, como Docker para las versiones heredadas y cri-o para las más recientes como valor predeterminado. Inspeccionar el ID de Pod Kubernetes (K8s) De la lista de Pods dentro del espacio de nombres cisco-mso, podemos seleccionar el contenedor para resolver problemas: [root@MxNDsh01 ~]# kubectl get pod -n cisco-msoNAME READY STATUS RESTARTS AGEconsistencyservice-569bdf5969-xkwpg 3/3 Running 0 9heeworker-65dc5dd849-485tq 2/2 Running 0 163mendpointes service-5db6f57884-hkf5g 2/2 Running 0 9hexecute tionservice-6c4894d4f7-p8fzk 2/2 Running 0 9hsiteservice-64dfcdf658-lvbr4 3/3 Running 0 9hsiteupgrade-68bcf987cc-ttn7h 2/2 Running 0 9h Los Pods deben ejecutarse en el mismo nodo

K8s. Para entornos de producción, podemos agregar la opción -o wide al final para averiguar el nodo que ejecuta cada POD. Con el ID de Pod K8s (en negrita en el ejemplo de salida anterior) podemos verificar el Proceso (PID) asignado por Container Runtime. Cómo inspeccionar el PID desde el tiempo de ejecución del contenedor El nuevo tiempo de ejecución del contenedor predeterminado es CRI-O para Kubernetes. Así que el documento viene después de esa regla para los comandos. El ID de proceso (PID) asignado por CRI-O puede ser único en el nodo K8s, que se puede descubrir con la utilidad crictl. La opción ps revela el ID dado por CRI-O a cada contenedor que construye el Pod, dos para el ejemplo del dispositivo sitesevice: [root@MxNDsh01 ~]# crictl ps |grep siteservicefb560763b06f2 172.31.0.0:30012/cisco-mso/sslcontainer@sha256:2d788fa493c885ba8c9e5944596b864d090d9051b0eab82123ee4d19596279c9 hace 10 horas Running msc-siteservice2-ssl 0 074727b4e9f51ad2d42aae1ad91d0195292f7fcc62f38529e135a1315c358067004a086cfed7e059986ce615b0 10 horas Running siteservice-leader election 0 074727b4e9f5129b0b6d41d1e3 172.31.0.0:30012/cisco-mso/siteservice@sha256:80a2335bcd5366952b4d60a275b20c70de0bb65a47bf8ae6d988f07b1e0bf494 hace 10 horas Running siteservice 0 074727b4e9f51[root@MxNDsh01 ~]# Con esta información, podemos usar la opción inspect CRIO-ID para ver el PID real dado a cada contenedor. Esta información es necesaria para el comando nsenter: [root@MxNDsh01 ~]# crictl inspect fb560763b06f2| grep -i pid"pid": 239563,"pids": {"type": "pid" How to Use nsenter to Run Network Debug Commands Inside a Container Con el PID del resultado anterior, podemos utilizar como destino en la siguiente sintaxis del comando: nsenter —target <PID> —net <NETWORK COMMAND> La opción —net nos permite ejecutar comandos en los espacios de nombres de red, por lo que el número de comandos disponibles es limitado. Por ejemplo: [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 0.0.0inet6 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 0TX packets 828016 bytes 307255950 (293.0 MiB)TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536inet 127.0 0.0.1 máscara de red 255.0.0.0inet6 ::1 prefixlen 128 scopeid 0x10<host>cola de texto de bucle 1000 (Loopback local)paquetes RX 42289 bytes 14186082 (13.5 MiB)errores RX 0 descartados 0 desbordamientos 0 trama 0TX paquetes 42289 bytes 14186082 (13.5 MiB)Errores TX 0 descartados 0 desbordamientos 0 colisiones de portadora 0 El ping también está disponible, y prueba la conectividad desde el contenedor hacia el exterior, en lugar de solo K8nodo s. [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.ADVERTENCIA: no se puede verificar el certificado de 1xx.2xx.3xx.4xx, emitido por ‘/C=US/ST=CA/O=Cisco System/CN=APIC’:No se puede verificar localmente la autoridad del emisor.ADVERTENCIA: el nombre común del certificado ‘APIC’ no coincide con el nombre de host solicitado ‘1xx.2xx.2xx.3xx 4xx".Solicitud HTTP enviada, a la espera de respuesta... 200 OKLongitud: 3251 (3.2K) [text/html]Guardar en: "index.html"100%[==========>] 3.251 —.-K/s en 0s2023-01-24 23:46:04 (548 MB/s) - "index.html" guardado [3251/3251]

Acerca de esta traducción
Cisco ha traducido este documento combinando la traducción automática y los recursos humanos a fin de ofrecer a nuestros usuarios en todo el mundo contenido en su propio idioma.

Tenga en cuenta que incluso la mejor traducción automática podría no ser tan precisa como la proporcionada por un traductor profesional.

Cisco Systems, Inc. no asume ninguna responsabilidad por la precisión de estas traducciones y recomienda remitirse siempre al documento original escrito en inglés (insertar vínculo URL).