Dépannage et examen des ressources NDO

Table des matières

Introduction

Ce document décrit comment examiner et dépanner NDO avec l'interface de ligne de commande kubectl et container runtime.

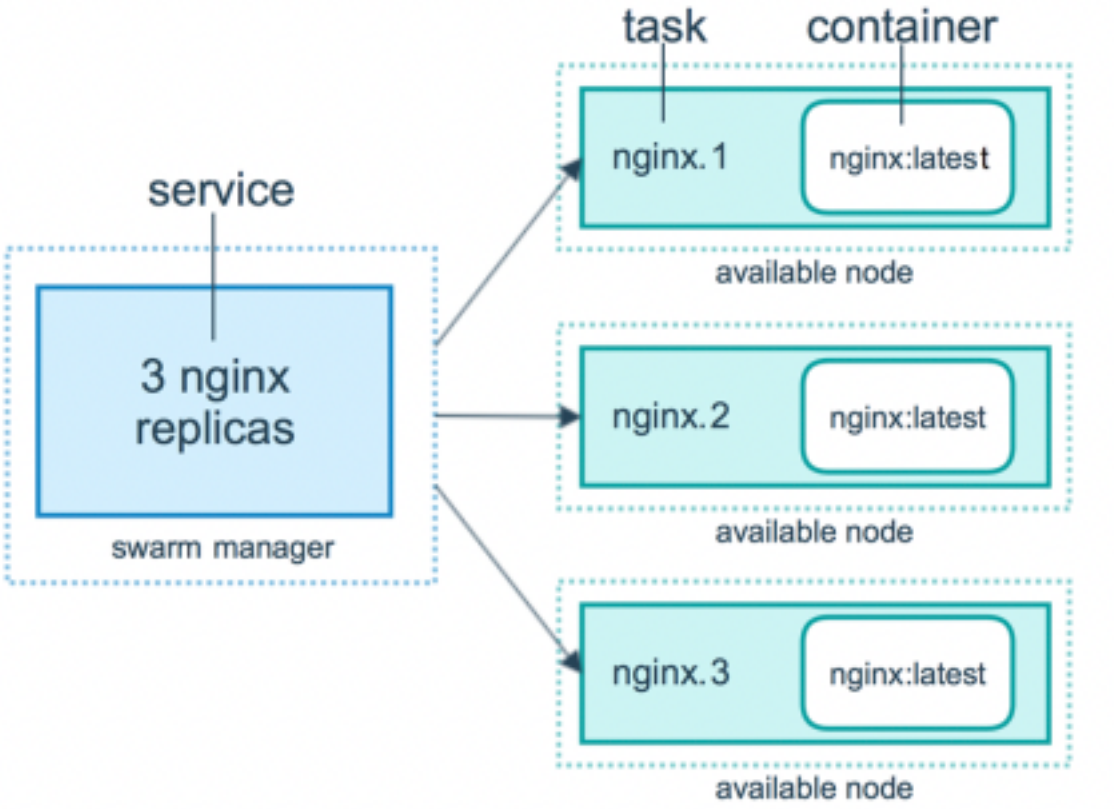
QuickStart NDO

Cisco Nexus Dashboard Orchestrator (NDO) est un outil d'administration de fabric qui permet aux utilisateurs de gérer différents types de fabrics, notamment les sites Cisco® ACI® (Application Centric Infrastructure), les sites Cisco Cloud ACI et les sites Cisco Nexus Dashboard Fabric Controller (NDFC), chacun étant géré par son propre contrôleur (cluster APIC, cluster NDFC ou instances APIC cloud dans un cloud public).

NDO offre une orchestration, une évolutivité et une reprise après sinistre cohérentes du réseau et des politiques sur plusieurs data centers via une seule interface.

Auparavant, le MSC (contrôleur multisite) était déployé en tant que cluster à trois noeuds avec des appliances virtuelles ouvertes (OVA) VMWare qui permettaient aux clients d'initialiser un cluster Docker Swarm et les services MSC. Ce cluster Swarm gère les microservices MSC comme conteneurs et services Docker.

Cette image montre une vue simplifiée de la façon dont la corbeille Docker gère les microservices comme des répliques du même conteneur pour obtenir une haute disponibilité.



Le Docker Swarm était chargé de maintenir le nombre de répliques attendu pour chacun des microservices dans l'architecture MSC. Du point de vue de Docker Swarm, le contrôleur multisite était le seul déploiement de conteneur à orchestrer.

Nexus Dashboard (ND) est une console de gestion centrale pour plusieurs sites de data center et une plate-forme commune qui héberge les services d'exploitation de data center Cisco, qui incluent Nexus Insight et MSC à partir de la version 3.3, et a changé le nom en Nexus Dashboard Orchestrator (NDO).

Alors que la plupart des microservices qui composent l'architecture MSC restent les mêmes, NDO est déployé dans un cluster Kubernetes (K8) plutôt que dans un cluster Docker Swarm. Cela permet à ND d'orchestrer plusieurs applications ou déploiements au lieu d'un seul.

Kubernetes avec NDO Crash-Course

Kubernetes est un système open source permettant d'automatiser le déploiement, l'évolutivité et la gestion des applications conteneurisées. En tant que Docker, Kubernetes fonctionne avec la technologie de conteneur, mais n'est pas lié à Docker. Cela signifie que Kubernetes prend en charge d'autres plates-formes de conteneurs (Rkt, PodMan).

Une différence clé entre Swarm et Kubernetes est que ce dernier ne fonctionne pas directement avec les conteneurs, il fonctionne avec un concept de groupes de conteneurs co-localisés, appelés Pods, à la place.

Les conteneurs d'un POD doivent s'exécuter dans le même noeud. Un groupe de pods est appelé un déploiement. Un déploiement Kubernetes peut décrire une application complète.

Kubernetes permet également aux utilisateurs de s'assurer qu'une certaine quantité de ressources sont disponibles pour une application donnée. Pour ce faire, des contrôleurs de réplication sont utilisés afin de garantir que le nombre de pods est cohérent avec les manifestes d'application.

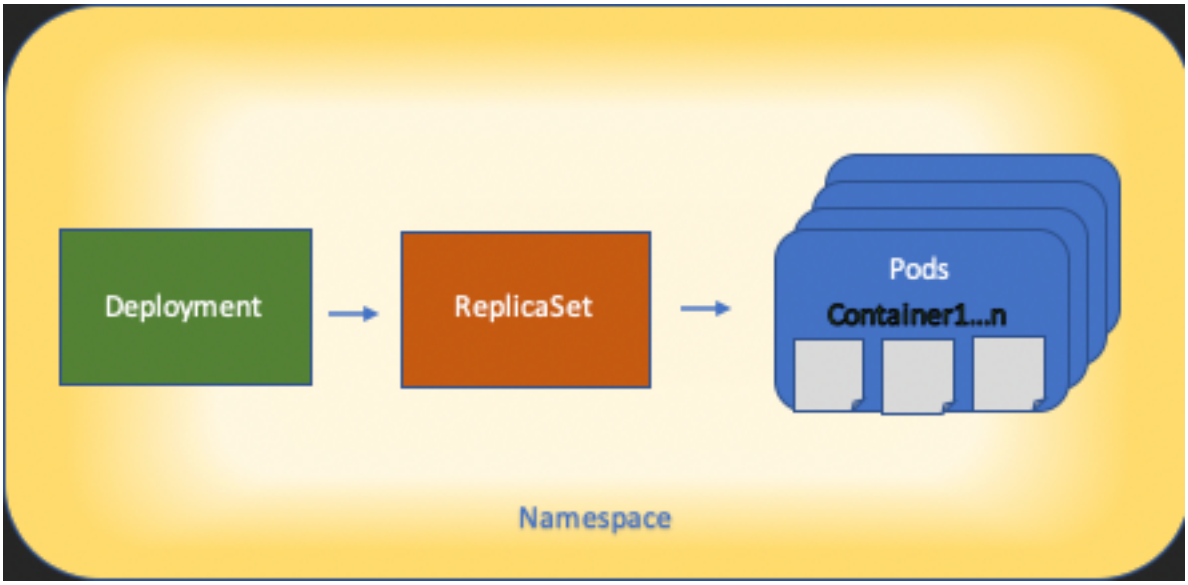
Un manifeste est un fichier au format YAML qui décrit une ressource à déployer par le cluster. La ressource peut être l'une de celles décrites précédemment ou d'autres disponibles pour les utilisateurs.

L'application est accessible en externe avec un ou plusieurs services. Kubernetes inclut une option Load Balancer pour y parvenir.

Kubernetes offre également un moyen d'isoler différentes ressources avec le concept d'espaces de noms. Le ND utilise des espaces de noms pour identifier de manière unique différentes applications et différents services de cluster. Lorsque vous exécutez des commandes CLI, spécifiez toujours l'espace de noms.

Bien qu'une connaissance approfondie de Kubernetes ne soit pas nécessaire pour dépanner ND ou NDO, une compréhension de base de l'architecture Kubernetes est nécessaire pour identifier correctement les ressources présentant des problèmes ou nécessitant une attention particulière.

Les bases de l'architecture des ressources Kubernetes sont présentées dans ce schéma :



Il est important de se rappeler comment chaque type de ressource interagit avec les autres, et il joue un rôle majeur dans le processus de révision et de dépannage.

Présentation de NDO avec les commandes Kubernetes

Connexion à CLI Access

Pour l'accès CLI par SSH à NDO, la commande **admin-user** mot de passe requis. Cependant, nous utilisons plutôt le **rescue-user** mot de passe. Comme dans :

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

Il s'agit du mode et de l'utilisateur par défaut pour l'accès CLI et la plupart des informations sont disponibles.

Révision des espaces de noms NDO

Ce concept K8 permet d'isoler différentes ressources dans le cluster. La commande suivante peut être utilisée pour examiner les différents espaces de noms déployés :

```
[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
```

Les entrées en gras appartiennent aux Applications dans le NDO, tandis que les entités qui commencent par le préfixe **kube** appartiennent au cluster Kubernetes. Chaque espace de noms possède ses propres déploiements et pods indépendants

L'interface de ligne de commande kubectl permet de spécifier un espace de noms **--namespace** , si une commande est exécutée sans elle, l'interface de ligne de commande suppose que l'espace de noms est **default** (Espace de noms pour 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.
```

L'interface de ligne de commande kubectl permet différents types de formats pour la sortie, tels que yaml, JSON ou une table personnalisée. Cela est possible grâce à la **-o** option [format]. Exemple :

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

Àpartir du texte précédent, la sortie est un dictionnaire où l'une de ses clés est appelée **éléments** et la valeur est une **liste** de dictionnaires où chaque **dictionnaire** compte pour une entrée d'**espace de noms** et ses attributs sont une valeur de paire clé-valeur dans le dictionnaire ou des dictionnaires imbriqués.

Ceci est pertinent car K8s permet aux utilisateurs de sélectionner jsonpath comme sortie, ce qui

permet des opérations complexes pour un tableau de données JSON. Par exemple, à partir de la sortie précédente, si nous accédons à la valeur de **name** pour les espaces de noms, nous devons accéder à la valeur de la liste des éléments, puis metadata et d'obtenir la valeur de la clé **name**. Pour ce faire, utilisez la commande suivante :

```
[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 hiérarchie décrite est utilisée pour extraire les informations spécifiques requises. En fait, tous les éléments sont accessibles dans le **items** avec **éléments[*]**, puis la touche metadata et **name** avec **metadata.name**, la requête peut inclure d'autres valeurs à afficher.

Il en va de même pour l'option des colonnes personnalisées, qui utilisent une méthode similaire pour extraire les informations du tableau de données. Par exemple, si nous créons une table avec les informations relatives à la **name** et la UID valeurs, nous pouvons appliquer la commande :

```
[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
```

Le résultat nécessite un nom pour chaque colonne à afficher, puis affecte la valeur au résultat. Dans cet exemple, il y a deux colonnes : **NAME** et **UID**. Ces valeurs appartiennent à .metada.name et .metadata.uid respectivement. Pour plus d'informations et d'exemples, consultez :

[Prise en charge JSONPath]

[Colonnes personnalisées]

Examen du déploiement NDO

Un déploiement est un objet K8s qui fournit un espace joint pour gérer le jeu de réplicas et les pods. Les déploiements prennent en charge le déploiement de tous les pods appartenant à une application et le nombre de copies attendu de chacun d'eux.

L'interface de ligne de commande kubectl inclut une commande permettant de vérifier les déploiements pour un espace de noms donné :

```
[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
```

Nous pouvons utiliser la même table personnalisée avec l'utilisation de deployment au lieu de **namespace** et la **-n** pour afficher les mêmes informations que précédemment. En effet, le résultat est structuré de la même manière.

```
[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
```

Gardez à l'esprit que le nombre de copies affichées correspond au déploiement et non au nombre

de pods de chaque microservice.

Nous pouvons utiliser le mot clé **describe** au lieu de **get** pour afficher des informations plus détaillées sur une ressource, dans ce cas, le déploiement 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 ~]$
```

Les **describe** permet également d'inclure la commande **--show-events=true** pour afficher tout événement pertinent pour le déploiement.

Déflecteur

Révision du jeu de réplicas NDO (RS)

Déflecteur

UNIQUEMENT DISPONIBLE POUR L'UTILISATEUR RACINE

Un jeu de répliques (RS) est un objet K8 dont l'objectif est de maintenir un nombre stable de pods de répliques. Cet objet détecte également lorsqu'un nombre incorrect de réplicas est détecté avec une sonde périodique vers les modules.

Les RS sont également organisés en espaces de noms.

```
[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
```

Les describe inclut les informations relatives à l'URL, au port utilisé par la sonde et à la périodicité des tests et du seuil d'échec.

```
[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 #### UNIQUEMENT DISPONIBLE POUR L'UTILISATEUR RACINE #### Un jeu de réplicas (RS) est un objet K8s dont l'objectif est de maintenir un nombre stable de pods de réplicas. Cet objet détecte également lorsqu'un nombre incorrect de réplicas est détecté avec une sonde périodique vers les modules. Les RS sont également organisés en espaces de noms. [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 3d22hconsistcyservice-c98955599 1 1 1 3d22hdcnmworker-5d4d5cbb64 1 1 1 3d 1 1 3d22hjobschedulerservice-5d4fdfd696 1 1 3d22hendpointservice-7df9d5599c 1 1 1 3d22hexecutionservice-58ff89595f 1 1 1 3d22hfluentd-86785f89bd 1 1 1 3d22himportservice-88bcc8547 1 1 1 3d22hjobschedulerservice-5d4fdfd696 1 1 1 3d 2hnotifyservice-75c988cfd4 1 1 3d22hpctagvnidservice-644b755596 1 1 1 3d22hplatformservice-65cddb946f 1 1 1 3d22hplatformservice2-6796576659 1 1 1 3d22hpolicy service-545b9c7d9c 1 1 3d22hschemasservice-7597ff4c5 1 1 3d22 2hsdaservice-5f477dd8c7 1 1 3d22hsdwanservice-6f87cd999d 1 1 3d22hsiteservice-86bb756585 1 1 1 1 3d22hsiteupgrade-7d578f9b6d 1 1 3d22hsyncengine-5b8bdd6b45 1 1 1 3d22htemplateng-5cbf9fdc 48 1 1 1 3d22hui-84588b7c96 1 1 1 3d22huserservice-87846f7c6 1 1 1 1 3d22h L'option de description inclut des informations sur l'URL, le port utilisé par la sonde, ainsi que la périodicité des tests et le seuil d'échec. [root@MxNDsh01 ~]# kubectl description rs -n cisco-mso schemaservice-7597ff4c5Nom : schemaservice-7597ff4c5Espace de noms : cisco-msoSelector : k8s-app=schemaservice, pod-template-hash=7597ff4c5Étiquettes :

cpu.resource.case.cncf.io/schemaservice=cpu-lg-service k8s-app=schemaservice memory.resource.case.cncf.io/schemaservice=mem-xlg-service pod-template-hash=7597ff4c5Annotations : deployment.kubernetes.io/desired-replicas: 1 deployment.kubernetes.io/max-replicas: deployment.kubernetes.io/revision: 1Contrôlé par : Déploiement/Schéma ServiceReplicas : 1 current / 1 desirablePods Status : 1 Running / 0 Waiting / 0 Succeeded / 0 FailedPod 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 Compte de service : cisco-mso-sa Init Containers : init-msc : Image : cisco-mso/tools : 3.7.1j Port : <none> Port hôte : <none> Commande : /check_mongo.sh Environnement : <none> Montage : /secrets from infracerts (rw) Containers : schemaservice : Image : cisco-mso/schemaservice : 3.7.1j Ports : 8080/TCP, 8080/UDP Ports hôtes : 0/TCP, 0/UDP Commande : /launchscala.sh schemaservice Liveness : http-get http://:8080/api/v1/schemas/health delay=300s timeout=20s period=30s #success=1 #failure=3 Environnement : JAVA_OPTS : -XX:+IdleTuningGcOnIdle Monts : /jwtsecrets from jwtsecrets (rw) /logs from logs (rw) /logs infracerts (rw) msc-schemaservice-ssl : Image : cisco-mso/sslcontainer : 3.7.1j Ports : 443/UDP, 443/TCP Ports d'hôte : 0/UDP, 0/TCP Commande : /wrapper.sh

Examen de pods NDO

Un pod est un groupe de conteneurs étroitement liés qui s'exécutent dans le même espace de noms Linux (différent de l'espace de noms K8s) et dans le même noeud K8s. Il s'agit de l'objet K8s le plus atomique, car il n'interagit pas avec les conteneurs. L'application peut consister en un seul conteneur ou être plus complexe avec de nombreux conteneurs. Avec la commande suivante, nous pouvons vérifier les Pods de n'importe quel espace de noms donné :

```
[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 ~]$
```

Le nombre indiqué dans la deuxième colonne correspond au nombre de conteneurs pour chaque module.

Les **describe** est également disponible, qui inclut des informations détaillées sur les conteneurs de chaque module.

```
[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
```

Les informations affichées incluent l'image du conteneur pour chaque conteneur et indiquent le Container Runtime utilisé. Dans ce cas, CRI-O (**cri-o**), les versions précédentes de ND utilisées pour travailler avec Docker, cela influence la façon de se fixer à un conteneur.

[Déflecteur]

Par exemple, lorsque **cri-o** est utilisé, et nous voulons nous connecter par une session interactive à un conteneur (via le exec -it) au conteneur à partir de la sortie précédente ; mais au lieu de la docker , nous utilisons la commande **crictl** :

```
schemaservice:

   Container ID:  cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac

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

Nous utilisons cette commande :

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

root@schemaservice-7597ff4c5-w4x5d:/#

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

root
```

Pour les versions ND ultérieures, l'ID de conteneur à utiliser est différent. Tout d'abord, nous devons utiliser la commande **crictl ps** pour répertorier tous les conteneurs qui s'exécutent sur chaque noeud. Nous pouvons filtrer le résultat selon les besoins.

```
[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 ~]#
```

Avec la valeur de la première colonne, nous pouvons alors accéder au runtime Container avec la même commande que précédemment :

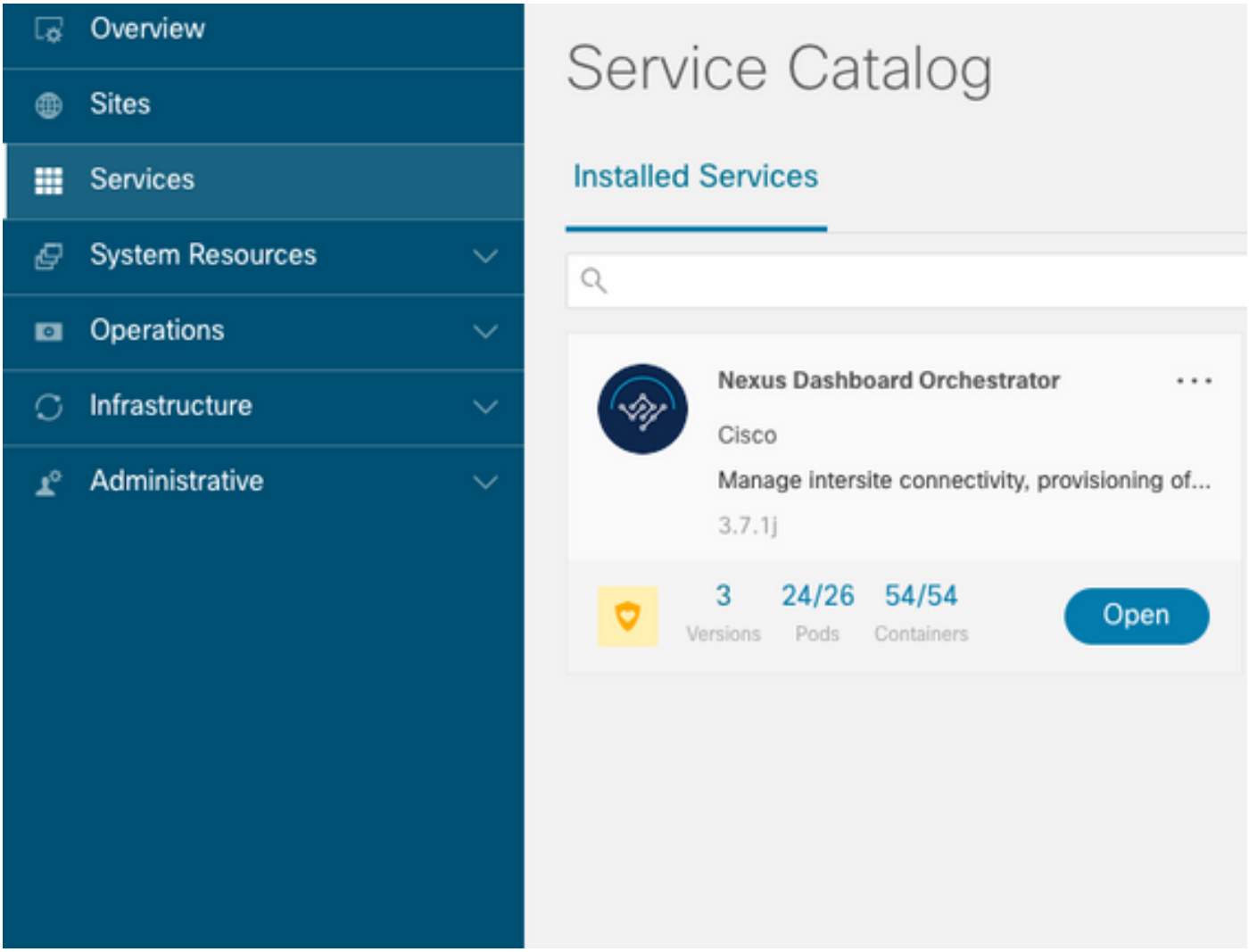
```
[root@singleNode ~]# crictl exec -it 4b26f67fc10cf bash
root@backupservice-8c699779f-j9jtr:/# pwd
/
```

Par exemple, lorsque cri-o est utilisé et que nous voulons nous connecter par une session interactive à un conteneur (via l'option exec -it) à partir du résultat précédent ; mais au lieu de la commande docker, nous utilisons la commande criclt : schemaservice : ID du conteneur : cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac Image : cisco-mso/schemaservice : 3.7.1j Nous utilisons cette commande : [root@MxNDsh01 ~]# crictl exec -it d2287f8659dec6848c0100b7d22 aeebd506f3f77af660238ca0c9c7e8946f4ac bashroot@schemaservice-7597ff4c5-w4x5d:/#root@schemaservice-7597ff4c5-w4x5d:/# whoamiroot Pour les versions ND ultérieures, l'ID de conteneur à utiliser est différent. Tout d'abord, nous devons utiliser la commande crictl ps pour répertorier tous les conteneurs qui s'exécutent sur chaque noeud. Nous pouvons filtrer le résultat selon les besoins. [root@singleNode ~]# crictl ps| grep backupa9bb161d67295 il y a 10.31.125.241:30012/cisco-mso/sslcontainer@sha256:26581eebd0bd6f4378a5fe4a98973dbda417c1905689f71f229765621f0cee75 2 jours qui ont exécuté msc-backupservice-ssl 0 84b3c691cfc2b4b26f67fc10cf il y a 10.31.125.241:30012/cisco-mso/backupservice@sha256:c21f4cdde696a5f2dfa7bb910b7278fc3fb4d46b02f42c3554f872ca8c87c061 2 jours Exécution de backupservice 0 84b3c691cfc2b[root@singleNode ~]# Avec la valeur de la première colonne, nous pouvons ensuite accéder à l'exécution du conteneur avec la même commande que précédemment : [root@singleNode ~]# crictl l exec -it 4b26f67fc10cf bashroot@backupservice-8c699779f-j9jtr:/# pwd/

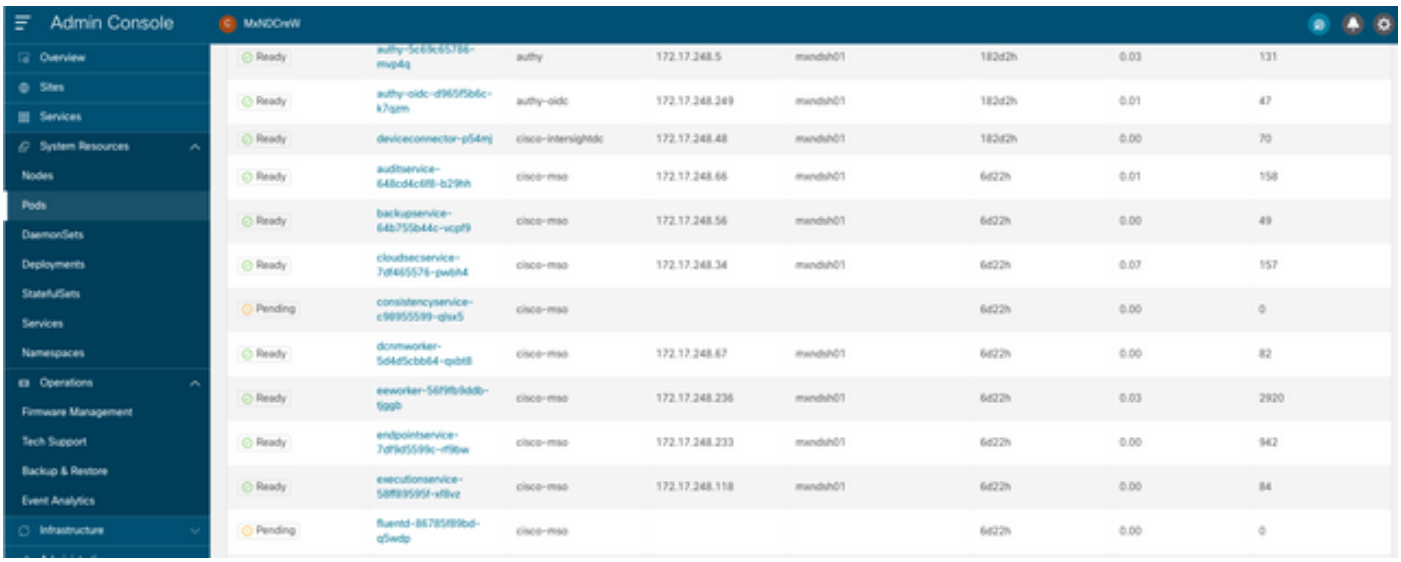
Le pod du cas d'utilisation n'est pas sain

Nous pouvons utiliser ces informations pour résoudre les problèmes d'intégrité des pods d'un déploiement. Pour cet exemple, la version du tableau de bord Nexus est 2.2-1d et l'application affectée est Nexus Dashboard Orchestrator (NDO).

L'interface utilisateur graphique de NDO affiche un ensemble incomplet de pods depuis la vue Service. Dans ce cas, 24 des 26 modules.



Une autre vue disponible sous la **System Resources -> Pods** afficher un état différent de celui des modules **Ready**.



Dépannage CLI pour les pods défectueux

Avec le fait connu que l'espace de noms est cisco-mso (bien que lors du dépannage, il soit le même pour d'autres applications/espaces de noms), la vue Pod s'affiche s'il y a des applications malsaines :

```
[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
```

Dans cet exemple, nous nous concentrons sur les modules de service de cohérence. À partir de la sortie JSON, nous pouvons obtenir les informations spécifiques des champs d'état, avec l'utilisation 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 ~]$
```

Nous voyons le dictionnaire d'**état** et dans une liste appelée **conditions** avec des dictionnaires comme éléments avec les clés **message** et **valeur**, la partie {"\n"} est de créer une nouvelle ligne à la fin :

```
[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 ~]$
```

Cette commande montre comment vérifier à partir de **get Pod** pour l'espace de noms :

```
[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
```

Avec la get pods , nous pouvons obtenir l'ID de pod avec les problèmes qui doivent correspondre à celui du résultat précédent. Dans cet exemple **consistencyservice-c98955599-qlsx5**.

Le format de sortie JSON indique également comment vérifier des informations spécifiques, à partir de la sortie donnée.

```
[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 sortie JSON doit inclure des informations sur l'état de l'attribut portant le même nom. Le message inclut des informations sur la raison.

```
[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 ~]$
```

Nous pouvons accéder aux Informations sur l'état et les exigences pour les pods :

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

Il est important de mentionner ici comment la valeur est calculée. Dans cet exemple, le processeur **500m** se réfère à **500 milicores**, et la **1G** en mémoire est pour Go.

Les **Describe** pour le noeud affiche la ressource disponible pour chaque travailleur K8 dans le

cluster (hôte ou machine virtuelle) :

```
[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 section **Allocatable** affiche le total des ressources en CPU , mémoire et stockage disponibles pour chaque noeud. La section **Allocated** affiche les ressources déjà utilisées. La valeur **13** pour CPU correspond à **13 coeurs** ou **13 000 (13 000) millicores**.

Dans cet exemple, le noeud est **surabonné**, ce qui explique pourquoi le Pod ne peut pas démarrer. Après avoir effacé le ND avec la suppression des applications ND ou l'ajout de ressources VM.

Le cluster essaie constamment de déployer des stratégies en attente. Ainsi, si les ressources sont libres, les pods peuvent être déployés.

```
[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
```

Avec la commande utilisée pour la vérification des ressources, nous confirmons que le cluster

dispose d'une ressource disponible pour le processeur :

```
[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 ~]$
```

Les détails du déploiement incluent un message contenant des informations sur les conditions actuelles des modules :

```
[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 ~]$
```

Déflecteur

Exécution des commandes de débogage réseau à partir d'un conteneur

Étant donné que les conteneurs incluent uniquement les bibliothèques minimales et les dépendances spécifiques au pod, la plupart des outils de débogage réseau (ping, ip route et ip addr) ne sont pas disponibles dans le conteneur lui-même.

Ces commandes sont très utiles lorsqu'il est nécessaire de dépanner des problèmes de réseau pour un service (entre des noeuds ND) ou une connexion vers les Apic car plusieurs microservices doivent communiquer avec les contrôleurs avec l'interface Data (**bond0** ou **bond0br**).

Les **nsenter** (utilisateur root uniquement) nous permet d'exécuter des commandes réseau à partir du noeud ND tel qu'il se trouve à l'intérieur du conteneur. Pour cela, recherchez l'ID de processus (PID) dans le conteneur que nous voulons déboguer. Ceci est accompli avec l'ID de Pod K8 par rapport aux informations locales du Container Runtime, comme Docker pour les versions héritées, et **cri-o** pour les plus récents par défaut.

Examinez l'ID de Pod Kubernetes (K8)

Dans la liste des pods de l'espace de noms cisco-mso, nous pouvons sélectionner le conteneur à dépanner :

```
[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
```

Les modules doivent être exécutés dans le même noeud K8. Pour les environnements de production, nous pouvons ajouter le **-o wide** à la fin pour connaître le noeud que chaque pod exécute. Avec l'ID de Pod K8 (en gras dans l'exemple de sortie précédent), nous pouvons vérifier le processus (PID) attribué par le Container Runtime.

Comment inspecter le PID à partir du Container Runtime

Le nouveau Container Runtime par défaut est CRI-O pour Kubernetes. Le document suit donc cette règle pour les commandes. L'ID de processus (PID) attribué par le CRI-O peut être unique dans le noeud K8s, qui peut être découvert avec le crictl utilitaire.

Les **ps** indique l'ID donné par CRI-O à chaque conteneur qui construit le Pod, deux pour l'exemple de service de site :

```
[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 ~]#
```

Avec ces informations, nous pouvons ensuite utiliser le inspect CRIO-ID pour afficher le PID réel attribué à chaque conteneur. Ces informations sont nécessaires pour le nsenter commande :

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

Utilisation de nsenter pour exécuter des commandes de débogage réseau dans un conteneur

Avec le PID du résultat ci-dessus, nous pouvons utiliser comme cible dans la syntaxe de commande suivante :

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

Les **--net** permet d'exécuter des commandes dans les espaces de noms du réseau, de sorte que le nombre de commandes disponibles est limité.

Exemple :

```
[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
```

La commande ping est également disponible et teste la connectivité du conteneur vers l'extérieur, plutôt que le noeud K8 uniquement.

```
[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]
```

Comment exécuter des commandes de débogage réseau à partir d'un conteneur Comme les conteneurs incluent uniquement les bibliothèques et dépendances minimales spécifiques au pod, la plupart des outils de débogage réseau (ping, ip route et ip addr) ne sont pas disponibles à l'intérieur du conteneur lui-même. Ces commandes sont très utiles lorsqu'il est nécessaire de dépanner des problèmes de réseau pour un service (entre des noeuds ND) ou une connexion vers les Apic car plusieurs microservices doivent communiquer avec les contrôleurs avec l'interface Data (bond0 ou bond0br). L'utilitaire nsenter (utilisateur racine uniquement) nous permet d'exécuter des commandes réseau à partir du noeud ND tel qu'il se trouve à l'intérieur du conteneur. Pour cela, recherchez l'ID de processus (PID) dans le conteneur que nous voulons déboguer. Pour ce faire, l'ID de Pod K8 est comparé aux informations locales du Container Runtime, comme Docker pour les versions héritées et cri-o pour les versions plus récentes par défaut. Inspection de l'ID de Kubernetes (K8s) du pod Dans la liste des pods de l'espace de noms cisco-mso, nous pouvons sélectionner le conteneur à dépanner : [root@MxNDsh01 ~]# kubectl get pod -n cisco-msoNAME READY STATUS RESTARTS AGEconsistcyservice-569bdf5969-xkwpg 3/3 Running 0 9heeworker-65dc5dd849-485tq 2/2 Running 0 165mendpointservice-2 db6f57884-hkf5g 2/2 Running 0 9hexecutionservice-6c4894d4f7-p8fzk 2/2 Running 0 9hsiteservice-64dfcdf658-lvbr4 3/3 Running 0 9hsiteupgrade-68bcf987cc-ttn7h 2/2 Running 0 9h Les pods doivent s'exécuter dans le même noeud K8. Pour les environnements de production, nous pouvons ajouter l'option -o wide à la fin pour connaître le noeud exécuté par

chaque pod. Avec l'ID de Pod K8 (en gras dans l'exemple de sortie précédent), nous pouvons vérifier le processus (PID) attribué par le Container Runtime. Comment inspecter le PID à partir du Container Runtime Le nouveau Container Runtime par défaut est CRI-O pour Kubernetes. Le document suit donc cette règle pour les commandes. L'ID de processus (PID) attribué par CRI-O peut être unique dans le noeud K8s, qui peut être découvert avec l'utilitaire crictl. L'option ps indique l'ID donné par CRI-O à chaque conteneur qui construit le Pod, deux pour l'exemple de service de site : [root@MxNDsh01 ~]# crictl ps |grep siteservicefb560763b06f2 172.31.0.0:30012/cisco-mso/sslcontainer@sha256:2d788fa493c885ba8c9e5944596b864d090d9051b0eab82123ee4d19596279c9 ll y a 10 heures Exécution de msc-siteservice2-ssl 0 074727b4e9f51ad2d42aae1ad91d0195292f7fcc62f38529e135a1315c358067004a086cfed7e059986ce615b0 ll y a 10 heures Exécution de siteservice-leader-election 0 074727b9f5129b0b6d41d1e3 il y a 172.31.0.0:30012/cisco-mso/siteservice@sha256:80a2335bcd5366952b4d60a275b20c70de0bb65a47bf8ae6d988f07b1e0bf494 il y a 10 heures Exécution de siteservice 0 074727b4e9f51[root@MxNDsh01 ~]# Avec ces informations, nous pouvons ensuite utiliser l'option inspect CRIO-ID pour voir le PID réel donné à chaque conteneur. Ces informations sont nécessaires pour la commande 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 Avec le PID de la sortie ci-dessus, nous pouvons utiliser comme cible dans la syntaxe de commande suivante : nsenter —target <PID> —net <COMMANDE RÉSEAU> L'option —net nous permet d'exécuter des commandes dans les espaces de noms du réseau, de sorte que le nombre de commandes disponibles est limité. Par exemple : [root@MxNDsh01 ~]# nsenter —target 239563 —net ifconfigeth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450inet 172.17.248.146 netmask 255.255.0.0 broadcast 0.0.0.0inet6 fe80::984f:32ff:fe72:7bfb prefixlen 0x4 scopeid 20<link>ether 9a:4f:32:72:7b:fb txqueuelen 0 (Ethernet)Paquets RX 916346 octets 271080553 (258,5 MiB)Erreurs RX 0 abandon 183 dépassements 0 trame 0TX paquets 828016 octets 307255950 (293,0 MiB)Erreurs TX 0 abandon 0 dépassements 0 porteuse 0 collisions 0lo : indicateurs=73<UP, LOOPBACK, RUNNING> mtu 65536inet 127.0.0.1 masque réseau 25 2.0.0.0inet6 ::1 prefixlen 128 scopeid 0x10<host>loop txqueuelen 1000 (Local Loopback)RX packets 42289 bytes 14186082 (13,5 MiB)RX errors 0 drop 0 overruns 0 frame 0TX packets 42289 bytes 14186082 (13,5 MiB)TX errors 0 drop 0 overruns 0 carrier 0 collisions 0 La commande ping est également disponible et teste la connectivité du conteneur vers l’extérieur, plutôt que le noeud K8s uniquement. [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 : impossible de vérifier le certificat de 1xx.2xx.3xx.4xx, émis par ‘/C=US/ST=CA/O=Cisco System/CN=APIC’ :impossible de vérifier localement l'autorité de l'émetteur.WARNING : le nom commun de certificat ‘APIC’ ne correspond pas au nom d'hôte demandé ‘1xx.2xx.3xx.4xx’.requête HTTP attente de réponse... 200 OKLength : 3251 (3.2K) [text/html]Enregistrement dans : « index.html » 100%[==========>] 3,251 —.-K/s dans 0s2023-01-24 23:46:04 (548 MB/s) - « index.html » enregistré [3251/3251]

À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.