Troubleshoot and Review of NDO Resources

Contents

[Introduction](#)
[NDO QuickStart](#)
[Kubernetes with NDO Crash-Course](#)
[NDO Overview with Kubernetes Commands](#)
[CLI Access Login](#)
[NDO Namespaces Review](#)
[NDO Deployment Review](#)
[NDO Replica Set (RS) Review](#)
[NDO Pod Review](#)
[Use-case Pod is not Healthy](#)
[CLI Troubleshoot for Unhealthy Pods](#)
[How to Run Network Debug Commands from Inside a Container](#)
[Inspect the Pod Kubernetes (K8s) ID](#)
[How toInspect the PID from the Container Runtime](#)
[How to Use nsenter to Run Network Debug Commands Inside a Container](#)

Introduction

This document describes how to review and troubleshoot NDO with the kubectl and container runtime CLI.

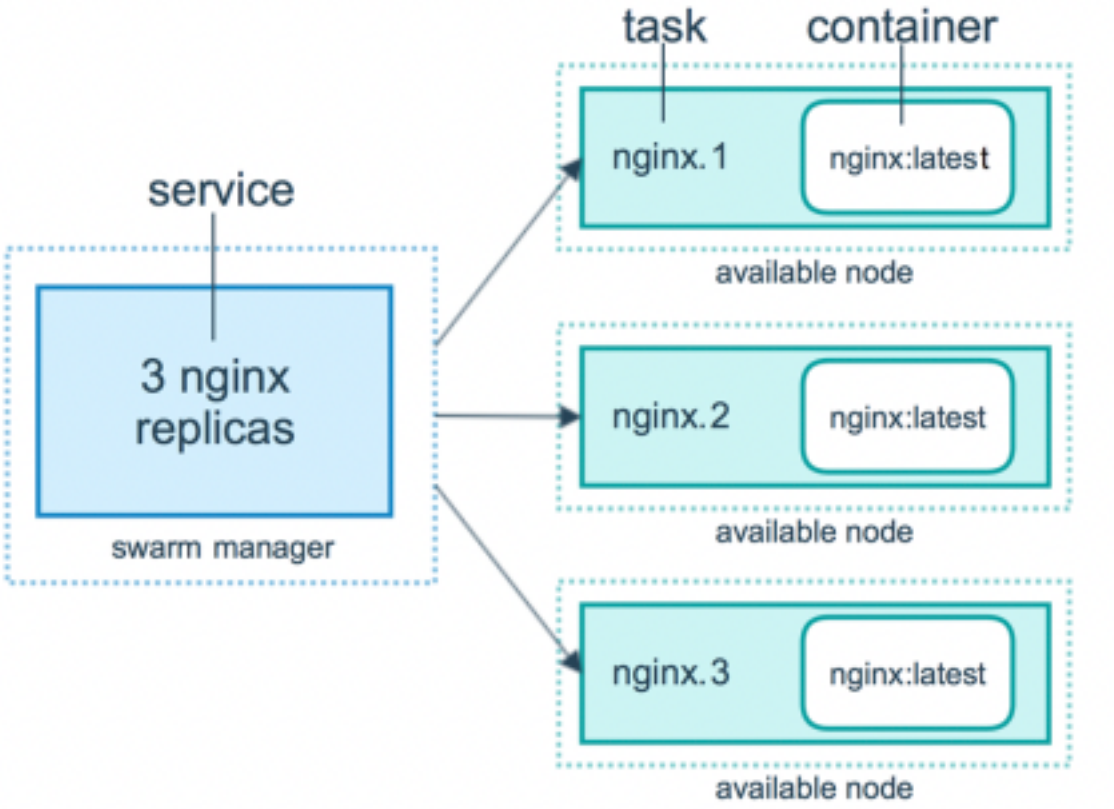
NDO QuickStart

The Cisco Nexus Dashboard Orchestrator (NDO) is a fabric administrative tool, which allows users to manage different kinds of fabrics that include Cisco® Application Centric Infrastructure (Cisco ACI®) sites, Cisco Cloud ACI sites, and Cisco Nexus Dashboard Fabric Controller (NDFC) sites, with each managed by its own controller (APIC cluster, NDFC cluster, or Cloud APIC instances in a public cloud).

NDO provides consistent network and policy orchestration, scalability, and disaster recovery across multiple data centers through a single pane of glass.

In the earlier days, the MSC (Multi-Site Controller) was deployed as a three-node cluster with VMWare Open Virtual Appliances (OVAs) that allowed customers to initialize a Docker Swarm cluster and the MSC services. This Swarm cluster manages the MSC microservices as Docker containers and services.

This picture shows a simplified view on how the Docker Swarm manages the microservices as replicas of the same container to achieve high availability.



The Docker Swarm was responsible to maintain the expected number of replicas for each one of the microservices in the MSC Architecture. From the Docker Swarm point of view, the Multi-Site Controller was the only container deployment to orchestrate.

Nexus Dashboard (ND) is a central management console for multiple data center sites and a common platform that hosts Cisco data center operation services, which include Nexus Insight and MSC version 3.3 onwards, and changed the name to Nexus Dashboard Orchestrator (NDO).

While most of the microservices that comprise the MSC architecture remain the same, NDO is deployed in a Kubernetes (K8s) cluster rather than in a Docker Swarm one. This allows ND to orchestrate multiple applications or deployments instead of just one.

Kubernetes with NDO Crash-Course

Kubernetes is an open-source system for automate deployment, scalability, and management of containerized applications. As Docker, Kubernetes works with the container technology, but is not tied with Docker. This means Kubernetes supports other container platforms (Rkt, PodMan).

A key difference between Swarm and Kubernetes is that the latter does not work with containers directly, it works with a concept of co-located groups of containers, called Pods, instead.

The containers in a Pod must run in the same node. A group of Pods is called a Deployment. A Kubernetes deployment can describe a whole application.

Kubernetes also allows the users to ensure a certain amount of resources are available for any given application. This is done with the use of Replication Controllers, to ensure the number of Pods are consistent with the Application Manifests.

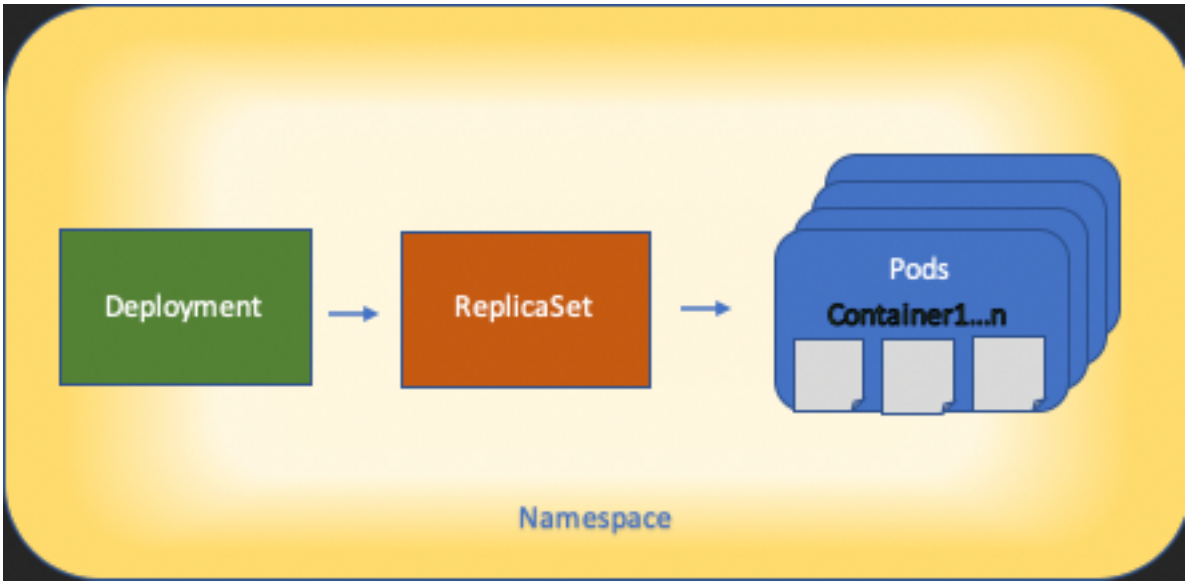
A Manifest is a YAML-formatted file that describes a resource to be deployed by the Cluster. The resource can be any of those described before or others available for users.

The Application can be accessed externally with one or more services. Kubernetes includes a Load Balancer option to accomplish this.

Kubernetes also offers a way to isolate different resources with the concept of Namespaces. The ND uses Namespaces to uniquely identify different Applications and Cluster Services. When CLI commands are run, always specify the Namespace.

Although a deep knowledge of Kubernetes is not required to troubleshoot ND or NDO, a basic understanding of the Kubernetes architecture is required to properly identify the resources with issues or that need attention.

The basics of Kubernetes resource architecture is shown in this diagram:



It is important to remember how each kind of resource interacts with the others, and it plays a major role in the review and troubleshoot process.

NDO Overview with Kubernetes Commands

CLI Access Login

For the CLI access by SSH to NDO, the **admin-user** password is needed. However, instead we use the **rescue-user** password. Like in:

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

This is the default mode and user for CLI access and most of the information is available to see.

NDO Namespaces Review

This K8s concept allows for isolation of different resources across the cluster. The next command can be used to review the different Namespaces deployed:

```
[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
```

The entries in bold belong to Applications in the NDO, while the entities that begin with the prefix **kube** belong to the Kubernetes cluster. Each Namespace has its own independent deployments and Pods

The kubectl CLI allows to specify a namespace with the **--namespace** option, if a command is run without it, the CLI assumes the Namespace is **default** (Namespace for 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.
```

The kubectl CLI allows different kinds of formats for the output, such as yaml, JSON, or a custom-made table. This is achieved with the **-o** [format] option. For example:

```
[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": ""

    }

}
```


From the previous text, the output is a **dictionary** where one of its keys is called **items** and the value is a **list** of dictionaries where each **dictionary** accounts for a **Namespace** entry and its attributes are key-value pair value in the dictionary or nested dictionaries.

This is relevant because K8s provides users with the option to select jsonpath as the output, this allows for complex operations for a JSON data array. For example, from the previous output, if we access the value of **name** for Namespaces, we need to access the value of items list, then the metadata dictionary, and get the value of the key **name**. This can be done with this command:

```
[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 ~]$
```

The hierarchy described is used to fetch the specific information required. Basically, all items are accessed in the **items** list with **items[*]**, then the key metadata and **name** with **metadata.name**, the query can include other values to display.

The same applies for the option of custom columns, which use a similar way to fetch the information from the data array. For example, if we create a table with the information about the **name** and the UID values, we can apply the command:

```
[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
```

The output requires a name for each column to display and then assign the value for the output. In this example, there are two columns: **NAME** and **UID**. These values belong to .metada.name and .metadata.uid respectively. More information and examples are available at:

JSONPath Support

Custom columns

NDO Deployment Review

A Deployment is a K8s object that provides a joined space to manage ReplicaSet and Pods. Deployments deal with the roll out of all Pods that belong to an Application and the expected number of copies of each one.

The kubectl CLI includes a command to check the deployments for any given Namespace:

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

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE

auditservice          1/1     1            1           3d22h

backupservice         1/1     1            1           3d22h

cloudsecservice       1/1     1            1           3d22h

consistencyservice    1/1     1            1           3d22h

dcnmworker            1/1     1            1           3d22h

eeworker              1/1     1            1           3d22h

endpointservice       1/1     1            1           3d22h

executionservice      1/1     1            1           3d22h

fluentd               1/1     1            1           3d22h

importservice         1/1     1            1           3d22h

jobschedulerservice   1/1     1            1           3d22h

notifyservice         1/1     1            1           3d22h

pctagvnidservice      1/1     1            1           3d22h

platformservice       1/1     1            1           3d22h

platformservice2      1/1     1            1           3d22h

policyservice         1/1     1            1           3d22h

schemaservice         1/1     1            1           3d22h

sdaservice            1/1     1            1           3d22h

sdwanservice          1/1     1            1           3d22h

siteservice           1/1     1            1           3d22h

siteupgrade           1/1     1            1           3d22h

syncengine            1/1     1            1           3d22h

templateeng           1/1     1            1           3d22h

ui                    1/1     1            1           3d22h

userservice           1/1     1            1           3d22h
```

We can use the same custom table with the use of deployment instead of **namespace** and the **-n** option to see the same information as before. This is because the output is structured in a similar way.

```
[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
```

Keep in mind the number of copies displayed is for the deployment, not the number of Pods for each microservice.

We can use the keyword **describe** instead of **get** to display more detailed information about a resource, in this case the schemaservice deployment:

```
[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 ~]$
```

The **describe** command also allows inclusion of the **--show-events=true** option to show any relevant event for the deployment.

Spoiler

NDO Replica Set (RS) Review

Spoiler

THIS IS ONLY AVAILABLE FOR ROOT USER

A Replica Set (RS) is a K8s object with the objective to maintain a stable number of replica Pods. This object also detects when an unhealthy number of replicas are seen with a periodic probe to the Pods.

The RS are also organized in namespaces.

```
[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
```

The describe option includes the information about the URL, the port the probe uses, and the periodicity of tests and failure threshold.

```
[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 ##### A Replica Set (RS) is a K8s object with the objective to maintain a stable number of replica Pods. This object also detects when an unhealthy number of replicas are seen with a periodic probe to the Pods. The RS are also organized in namespaces. [root@MxNDsh01 ~]# kubectl get rs -n cisco-msoNAME DESIRED CURRENT READY AGEauditservice-648cd4c6f8 1 1 1 3d22hbackupservice-64b755b44c 1 1 1 3d22hcloudsecservice-7df465576 1 1 1 3d22hconsistencyservice-c98955599 1 1 1 3d22hdcnmworker-5d4d5cbb64 1 1 1 3d22heeworker-56f9fb9ddb 1 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 3d22hnotifyservice-75c988cfd4 1 1 1 3d22hpctagvnidservice-644b755596 1 1 1 3d22hplatformservice-65cddb946f 1 1 1 3d22hplatformservice2-6796576659 1 1 1 3d22hpolicyservice-545b9c7d9c 1 1 1 3d22hschemaservice-7597ff4c5 1 1 1 3d22hsdaservice-5f477dd8c7 1 1 1 3d22hsdwanservice-6f87cd999d 1 1 1 3d22hsiteservice-86bb756585 1 1 1 3d22hsiteupgrade-7d578f9b6d 1 1 1 3d22hsyncengine-5b8bdd6b45 1 1 1 3d22htemplateeng-5cbf9fdc48 1 1 1 3d22hui-84588b7c96 1 1 1 3d22huserservice-87846f7c6 1 1 1 3d22h The describe option includes the information about the URL, the port the probe uses, and the periodicity of tests and failure threshold. [root@MxNDsh01 ~]# kubectl describe rs -n cisco-mso schemaservice-7597ff4c5Name: schemaservice-7597ff4c5Namespace: cisco-msoSelector: k8s-app=schemaservice,pod-template-hash=7597ff4c5Labels: cpu.resource.case.cncf.io/schemaservice=cpu-lg-service k8s-app=schemaservice memory.resource.case.cncf.io/schemaservice=mem-xlg-service pod-template-hash=7597ff4c5Annotations: deployment.kubernetes.io/desired-replicas: 1 deployment.kubernetes.io/max-replicas: 1 deployment.kubernetes.io/revision: 1Controlled By: Deployment/schemaserviceReplicas: 1 current / 1 desiredPods 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 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 Pod Review

A Pod is a group of closely related containers that run in the same Linux Namespace (different from K8s Namespace) and in the same K8s node. This is the most atomic object K8s handles, as it does not interact with containers. The application can consist of a single container or be more complex with many containers. With the next command, we can check the Pods of any given namespace:

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

NAME                                READY   STATUS    RESTARTS   AGE

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

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

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

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

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

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

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

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

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

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

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

mongodb-0                           2/2     Running   0          2d1h

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The number seen in the second column refers to the number of containers for each Pod.

The **describe** option is also available, which includes detailed information about the containers on each 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
```

The information displayed includes the container image for each container and shows the Container Runtime used. In this case, CRI-O (**cri-o**), previous versions of ND used to work with Docker, this influences how to attach to a container.

Spoiler

For example, when **cri-o** is used, and we want to connect by an interactive session to a container (via the exec -it option) to the container from the previous output; but instead of the docker command, we use the **crictl** command:

```
schemaservice:

   Container ID:  cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac

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

We use this command:

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

root@schemaservice-7597ff4c5-w4x5d:/#

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

root

For later ND releases, the Container ID to be used is different. First, we need to use the command **crictl ps** to list all the containers that run on each node. We can filter the result as required.

```
[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 ~]#
```

With the value from the first column, we can then access the Container run-time with the same command as before:

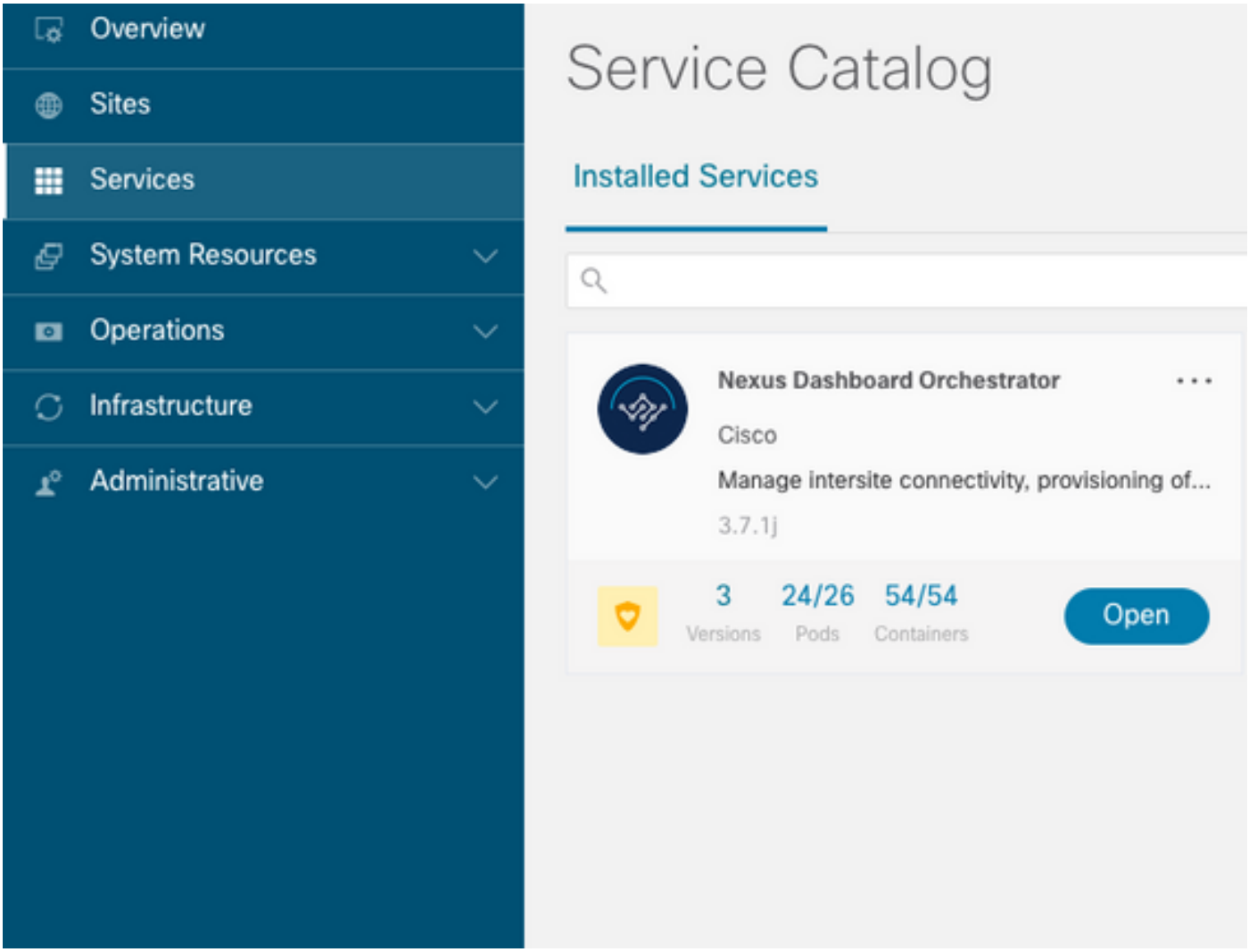
```
[root@singleNode ~]# crictl exec -it 4b26f67fc10cf bash
root@backupservice-8c699779f-j9jtr:/# pwd
/
```

For example, when cri-o is used, and we want to connect by an interactive session to a container (via the exec -it option) to the container from the previous output; but instead of the docker command, we use the criclt command: schemaservice: Container ID: cri-o://d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac Image: cisco-mso/schemaservice:3.7.1j We use this command: [root@MxNDsh01 ~]# crictl exec -it d2287f8659dec6848c0100b7d24aeebd506f3f77af660238ca0c9c7e8946f4ac bashroot@schemaservice-7597ff4c5-w4x5d:/#root@schemaservice-7597ff4c5-w4x5d:/# whoamiroot For later ND releases, the Container ID to be used is different. First, we need to use the command crictl ps to list all the containers that run on each node. We can filter the result as required. [root@singleNode ~]# crictl ps| grep backupa9bb161d67295 10.31.125.241:30012/cisco-mso/sslcontainer@sha256:26581eebd0bd6f4378a5fe4a98973dbda417c1905689f71f229765621f0cee75 2 days ago that run msc-backupservice-ssl 0 84b3c691cfc2b4b26f67fc10cf 10.31.125.241:30012/cisco-mso/backupservice@sha256:c21f4cdde696a5f2dfa7bb910b7278fc3fb4d46b02f42c3554f872ca8c87c061 2 days ago Running backupservice 0 84b3c691cfc2b[root@singleNode ~]# With the value from the first column, we can then access the Container run-time with the same command as before: [root@singleNode ~]# crictl exec -it 4b26f67fc10cf bashroot@backupservice-8c699779f-j9jtr:/# pwd/

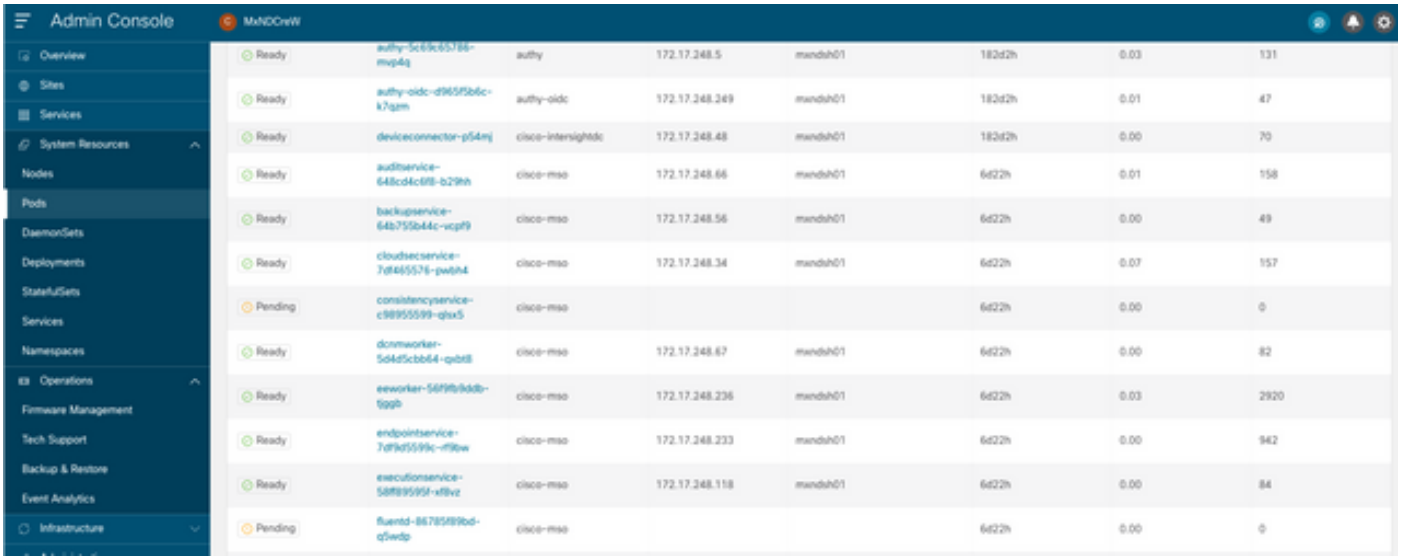
Use-case Pod is not Healthy

We can use this information to troubleshoot why Pods from a deployment are not healthy. For this example, the Nexus Dashboard version is 2.2-1d and the affected Application is Nexus Dashboard Orchestrator (NDO).

The NDO GUI displays an incomplete set of Pods from the Service view. In this case 24 out of 26 Pods.



Another view available under the **System Resources -> Pods** view where the Pods show a status different from **Ready**.



CLI Troubleshoot for Unhealthy Pods

With the known fact the Namespace is cisco-mso (although when troubleshot, it is the same for other apps/namespaces) the Pod view displays if there is any unhealthy ones:

```
[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
```

For this example, we focus in the consistencyservice Pods. From the JSON output, we can get the specific information from the status fields, with the use of 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 ~]$
```

We see the **status** dictionary and inside a list called **conditions** with dictionaries as items with the keys **message** and **value**, the {"\n"} part is to create a new line at the end:

```
[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 ~]$
```

This command shows how to check from the **get Pod** for the Namespace:

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

With the get pods command, we can get the Pod ID with issues that must match with the one from the previous output. In this example **consistencyservice-c98955599-qlsx5.**

The JSON output format also provides how to check specific information, from the given output.

```
[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 ~]$
```

The JSON output must include information about the status in the attribute with same name. The message includes information about reason.

```
[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 ~]$
```

We can access Information about the Status and the requirements for the 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]
```

Here it is important to mention how the value is calculated. In this example, the cpu **500m** refers to **500 milicores**, and the **1G** in memory is for GB.

The **Describe** option for the node shows the resource available for each K8s worker in the cluster (host or 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 ~]$
```

The **Allocatable** section shows the total Resources in CPU , Memory, and Storage available for each node. The **Allocated** section shows the Resources already in use. The value **13** for CPU refers to **13 Cores** or **13,000 (13K) millicores**.

For this example, the node is **oversubscribed**, which explains why the Pod cannot initiate. After we clear out the ND with the deletion of ND APPs or addition of VM Resources.

The Cluster constantly tries to deploy any pending policies, so if the resources are free, the Pods can be deployed.

```
[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
```

With the command used for resource check, we confirm the Cluster has available Resource for 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 ~]$
```

The deployment details include a message with information about the current conditions for Pods:

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

Spoiler

How to Run Network Debug Commands from Inside a Container

Because the containers only include the minimal libraries and dependencies specific for the Pod, most of network debug tools (ping, ip route, and ip addr) are not available inside the container itself.

These commands are very useful when there is a need to troubleshoot network issues for a service (between ND nodes) or connection toward the Apics because several microservices need to communicate with the controllers with the Data interface (**bond0** or **bond0br**).

The **nsenter** utility (root user only) allows us to run network commands from the ND node as it is inside the container. For this, find the process ID (PID) from the container we want to debug. This is accomplished with the Pod K8s ID against the local information from the Container Runtime, like Docker for legacy versions, and **cri-o** for newer ones as default.

Inspect the Pod Kubernetes (K8s) ID

From the list of Pods inside the cisco-mso Namespace, we can select the container to troubleshoot:

```
[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
```

The Pods must run in the same K8s node. For production environments, we can add the **-o wide** option at the end to find out the node each Pod runs. With the Pod K8s ID (bolded in the previous output example) we can check the Process (PID) assigned by the Container Runtime.

How to Inspect the PID from the Container Runtime

The new default Container Runtime is CRI-O for Kubernetes. So the document comes after that rule for the commands. The Process ID (PID) assigned by CRI-O can be unique in the K8s Node, which can be discovered with the crictl utility.

The **ps** option reveals the ID given by CRI-O to each container that builds the Pod, two for the sitesevice example:

```
[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 ~]#
```

With this information, we can then use the inspect CRIO-ID option to see the actual PID given to each container. This information is needed for the nsenter command:

```
[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

With the PID from the output above, we can use as the target in the next command syntax:

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

The **--net** option allows us to run commands in the network Namespaces, so the number of commands available is limited.

For example:

```
[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
```

The ping is also available, and it tests connectivity from the container to the outside, rather than only the K8s node.

```
[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]
```

How to Run Network Debug Commands from Inside a Container Because the containers only include the minimal libraries and dependencies specific for the Pod, most of network debug tools (ping, ip route, and ip addr) are not available inside the container itself. These commands are very useful when there is a need to troubleshoot network issues for a service (between ND nodes) or connection toward the Apics because several microservices need to communicate with the controllers with the Data interface (bond0 or bond0br). The nsenter utility (root user only) allows us to run network commands from the ND node as it is inside the container. For this, find the process ID (PID) from the container we want to debug. This is accomplished with the Pod K8s ID against the local information from the Container Runtime, like Docker for legacy versions, and cri-o for newer ones as default. Inspect the Pod Kubernetes (K8s) ID From the list of Pods inside the cisco-mso Namespace, we can select the container to troubleshoot: [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 163mendpointservice-5db6f57884-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 The Pods must run in the same K8s node. For production environments, we can add the -o wide option at the end to find out the node each Pod runs. With the Pod K8s ID (bolded in the previous output example) we can check the Process (PID) assigned by the Container Runtime. How to Inspect the PID from the Container Runtime The new default Container Runtime is CRI-O for Kubernetes. So the document comes after that rule for the commands. The Process ID (PID) assigned by CRI-O can be unique in the K8s Node, which can be discovered with the crictl utility. The ps option reveals the ID given by CRI-O to each container that builds the Pod, two for the sitesevice example: [root@MxNDsh01 ~]# crictl ps |grep siteservicefb560763b06f2 172.31.0.0:30012/cisco-mso/sslcontainer@sha256:2d788fa493c885ba8c9e5944596b864d090d9051b0eab82123ee4d19596279c9 10 hours ago Running msc-siteservice2-ssl 0 074727b4e9f51ad2d42aae1ad91d0195292f7fcc62f38529e135a1315c358067004a086cfed7e059986ce615b0 10 hours ago Running siteservice-leader-election 0 074727b4e9f5129b0b6d41d1e3 172.31.0.0:30012/cisco-mso/siteservice@sha256:80a2335bcd5366952b4d60a275b20c70de0bb65a47bf8ae6d988f07b1e0bf494 10 hours ago Running siteservice 0 074727b4e9f51[root@MxNDsh01 ~]# With this

information, we can then use the inspect CRIO-ID option to see the actual PID given to each container. This information is needed for the nsenter command: [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 With the PID from the output above, we can use as the target in the next command syntax: nsenter --target <PID> --net <NETWORK COMMAND> The --net option allows us to run commands in the network Namespaces, so the number of commands available is limited. For example: [root@MxNDsh01 ~]# nsenter --target 239563 --net ifconfigeth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450inet 172.17.248.146 netmask 255.255.0.0 broadcast 0.0.0.0inet6 fe80::984f:32ff:fe72:7bfb prefixlen 64 scopeid 0x20<link>ether 9a:4f:32:72:7b:fb txqueuelen 0 (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 127.0.0.1 netmask 255.0.0.0inet6 ::1 prefixlen 128 scopeid 0x10<host>loop txqueuelen 1000 (Local Loopback)RX packets 42289 bytes 14186082 (13.5 MiB)RX errors 0 dropped 0 overruns 0 frame 0TX packets 42289 bytes 14186082 (13.5 MiB)TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 The ping is also available, and it tests connectivity from the container to the outside, rather than only the K8s node. [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 OKLength: 3251 (3.2K) [text/html]Saving to: ‘index.html’100%[==========>] 3,251 --.-K/s in 0s2023-01-24 23:46:04 (548 MB/s) - ‘index.html’ saved [3251/3251]