

Manifests are under `classfiles/os-nw`. Do a `git pull` first.

Cluster Network Operator

```
$ oc get clusteroperator
```

List of cluster operators incl network

This shows a healthy cluster network operator:

```
$ oc get clusteroperator/network
```

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
network	4.8.12	True	False	False	20d

The structure: It's a deployment with one pod.

```
$ oc get -n openshift-network-operator deployment/network-operator
```

```
$ oc get all -n openshift-network-operator
```

Warnings or errors in the log

```
$ oc logs deploy/network-operator -n openshift-network-operator |grep '^[WE]'
```

Cluster network config

Configuration is contained in a customresource:

```
$ oc get networks/cluster -o yaml
```

```
apiVersion: config.openshift.io/v1
```

```
kind: Network
```

```
metadata:
```

```
  creationTimestamp: "2021-09-23T06:59:03Z"
```

```
  generation: 2
```

```
  name: cluster
```

```
  resourceVersion: "2988"
```

```
  uid: ae1c3f58-bd30-4fc1-9d8c-ca07b2be07ae
```

```
spec:      ← As configured
```

```
  clusterNetwork:      ← Pod addresses
```

```
    - cidr: 10.217.0.0/22
```

```
      hostPrefix: 23      ← prefix for each node
```

```
  externalIP:
```

```
    policy: {}
```

```
  networkType: OpenShiftSDN
```

```
  serviceNetwork:      ← Services
```

```
    - 10.217.4.0/23
```

```
status:      ← Actual state
```

```
  clusterNetwork:
```

```
    - cidr: 10.217.0.0/22
```

```
      hostPrefix: 23
```

```
  clusterNetworkMTU: 1400
```

```
  networkType: OpenShiftSDN
```

```
  serviceNetwork:
```

```
    - 10.217.4.0/23
```

Configure the NodePort range

This is an example of configuring the CNO. A NodePort service can use ports between 30000 and 32767 by default. You can increase that range.

First confirm that the highest nodeport is 32767.

Log on to the console as *developer* and switch to the **administrator** perspective. Go to **Networking**→**Services** and create a service. You will be given a YAML file; add the *NodePort* type and a NodePort value greater than 32767:

```
spec:
  selector:
    app: MyApp
  type: NodePort
  ports:
    - protocol: TCP
      port: 80
      nodeport: 35000
      targetPort: 9376
```

When you create this service, you will see that the actual nodeport value is set to less than 32767.

To increase the nodeport range, apply the following manifest `cno-nodeport.yaml` as kubeadmin.

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  serviceNodePortRange: "30000-35000"
```

It will take a minute or two before this change is effective.

Go back to the web console. Delete the previous service and create a new one. This time, 35000 should be accepted.

In `cno-nodeport.yaml`, reset the range to 30000-32767 and apply the manifest again.

DNS Operator

```
$ oc project openshift-dns-operator
$ oc get all
```

There is a service.

```
$ oc describe dns.operator/default
```

status.conditions tells why the DNS operator is degraded, progressing or available

To see the DNS operator's daemon set:

```
$ oc get ds -n openshift-dns
```

Ingress Operator - access logs

The operator is configured by the `ingresscontroller/default` resource in the `openshift-ingress-operator` namespace.

Configuration options: Certificates, replicas, access logs, endpointPublishingStrategy

Ingress is implemented by a deployment in namespace `openshift-ingress`.

Explore the resources:

```
$ oc describe ingresscontroller/default -n openshift-ingress-operator
$ oc get all -n openshift-ingress
```

Feel free to [describe](#) the resources you see.

Configure access logging

As an example how you can configure the OpenShift ingress operator, enable access logging. This will enable you to see accesses to Ingresses and Routes.

Use this manifest named `ingress-log.yaml`.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
```

```

name: default
namespace: openshift-ingress-operator
spec:
  logging:
    access:
      destination:
        type: Container

```

Apply this manifest. This creates a new container named `logs` in the controller's pod. Access log messages are stored there.

```

$ oc project openshift-ingress
$ oc get pod
$ oc describe pod/router-default-6f855cccdb-8vz92 | more

```

Under `Containers`, there is a `router`, which is the main ingress container. View the environment. Interesting details: Certificate information, canonical hostname, load balance algorithm, several volume mounts. What are the mounts for?

The second container is named `logs`. Which command does it run? What are the mounts for?

View the access log

```
$ oc logs -f -c logs router-default-6f855cccdb-8vz92
```

There is considerable traffic, seemingly from internal OpenShift components. Leave this running.

Now generate traffic to the *parksmap* application. Open a separate terminal window and get the URL from the command line:

```
$ oc get route -n devproject
```

NAME	HOST/PORT	PATH	SERVICES	PORT
route-to-parksmap	route-to-parksmap-devproject.apps-crc.testing	/	parksmap-katacoda	8080-tcp

Use curl to access the URL repeatedly:

```

$ while true
do curl -sL route-to-parksmap-devproject.apps-crc.testing >/dev/null
sleep 1
done

```

Watch the ingress log. You should see parksmap entries appear in the log.

Check what happens in the logs container

Launch a shell and look around.

```

$ oc exec -it router-default-6f855cccdb-8vz92 -c logs -- bash
bash-4.4$ ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
1000610+      1     0   0 04:05 ?        00:00:00 /sbin/rsyslogd -n -i /tmp/rsyslog.pid -f /etc/rsyslog/rsyslog.conf

```

Check the syslog configuration.

```

bash-4.4$ more /etc/rsyslog/rsyslog.conf
$ModLoad imuxsock
$SystemLogSocketName /var/lib/rsyslog/rsyslog.sock
$ModLoad omstdout.so
*. * :omstdout:

```

This means that the log output goes to a socket, not a file. No storage space is used for the logs.

Ingress operator - route admission

The domain name of a route must normally be unique. However, identical domain names in different projects can be permitted by enabling *route admission*.

Confirm that identical domain names are not permitted

First, try to create two routes with the same hostname. This will fail, as expected.

Log on to the web console as *developer*. Create a second project named `project2`. Switch to the Administrator role.

In *devproject*, click **Networking**→**Services** and create a dummy service. You will be presented with a YAML file; just change the name under *metadata* to **dummyservice**.

Then create a route. Name it **dummyroute**, and set the Hostname to **dummy.apps-crc.testing**.

Now switch to `project2` and do the same. You will be able to create the service, but the route will have a status of *rejected*. On the end of the route page, you will find the reason for the rejection: "a route in another namespace holds dummy.apps-crc.testing and is older than dummyroute".

Configure route admission

As kubeadmin, apply the manifest named `ingress-route-admission.yaml`:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

Confirm that identical domain names are possible

Go back to the web console. Remove `dummyroute` and recreate it with the same parameters as before. This will now succeed.

Reset route admission

As kubeadmin, apply the manifest named `ingress-route-admission-off.yaml`:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: Strict
```