Openshift Tips

Contents

[Labels & Annotations 5](#_Toc168645837)

[Add label to a Node 5](#_Toc168645838)

[Add node selector at namespace level 5](#_Toc168645839)

[Add tolerations at namespace level 6](#_Toc168645840)

[API 7](#_Toc168645841)

[API resources 7](#_Toc168645842)

[API resources per API group 7](#_Toc168645843)

[Explain resources 7](#_Toc168645844)

[Explain resources per api group 7](#_Toc168645845)

[Certificates 8](#_Toc168645846)

[Sign all the pending csr 8](#_Toc168645847)

[Authenticate users using TLS certificates 8](#_Toc168645848)

[Generate a private key and a CSR for the new user. 8](#_Toc168645849)

[Authenticate to Openshift API with an user with permissions to create CertificateSigningRequest objects (e.g. kube-admin). 8](#_Toc168645850)

[Create a CertificateSigningRequest to sign the CSR by the kube-apiserver CA. 8](#_Toc168645851)

[Approve the pending CSR. 8](#_Toc168645852)

[Get the user certificate from the signed CSR. 9](#_Toc168645853)

[Get the CA chain for the API server. 9](#_Toc168645854)

[Create a kubeconfig to authenticate the new user using the TLS certificate. 9](#_Toc168645855)

[Authenticate using the new kubeconfig. 9](#_Toc168645856)

[Verify the new user can make operations against the API server. 9](#_Toc168645857)

[Verify the API certificates 9](#_Toc168645858)

[Extract etcd CA 9](#_Toc168645859)

[Clean up 10](#_Toc168645860)

[Delete ‘Completed’ pods 10](#_Toc168645861)

[Change the image GC thresholds 10](#_Toc168645862)

[Run a full /var/lib/containers/storage clean-up using podman 10](#_Toc168645863)

[Cluster 12](#_Toc168645864)

[Set control-plane nodes as NoSchedulable 12](#_Toc168645865)

[Routers 12](#_Toc168645866)

[Image-registry 12](#_Toc168645867)

[Monitoring stack 12](#_Toc168645868)

[List all container images running in a cluster 13](#_Toc168645869)

[List all container images stored in a cluster 13](#_Toc168645870)

[Cluster version 14](#_Toc168645871)

[Switch clusterversion channel 14](#_Toc168645872)

[Unmanage operators 14](#_Toc168645873)

[Get the operator deployment information 14](#_Toc168645874)

[Create the patch yaml file 15](#_Toc168645875)

[Patch the clusterversion object 15](#_Toc168645876)

[Disabling the cluster-version operator 16](#_Toc168645877)

[Configuration 17](#_Toc168645878)

[Get CRI-O settings 17](#_Toc168645879)

[OCP Master configuration 17](#_Toc168645880)

[Observe the SDN configuration 17](#_Toc168645881)

[Making Master Unscheduleable/Scheduleable 17](#_Toc168645882)

[Get Settings used in install config 17](#_Toc168645883)

[Deployments 18](#_Toc168645884)

[Import environment variables from a file 18](#_Toc168645885)

[Set some of the local shell environment into a deployment config 18](#_Toc168645886)

[Import environment variables from a configmap with a prefix 18](#_Toc168645887)

[Import specific keys from a configmap 18](#_Toc168645888)

[Remove environment variable in a deployment 18](#_Toc168645889)

[Remove environment variable from container ‘c1’ in a deployment 18](#_Toc168645890)

[Remove environment variable from container ‘c1’ in all deployments 18](#_Toc168645891)

[Show logs for all containers within a pod 18](#_Toc168645892)

[Show logs for all containers within a pod with pod name and container name 18](#_Toc168645893)

[Identity Providers 19](#_Toc168645894)

[Add HTPasswd authentication (OpenShift 4 only) 19](#_Toc168645895)

[Create htpasswd file (with admin username) 19](#_Toc168645896)

[Create secret in openshift-config project 19](#_Toc168645897)

[Edit cluster OAuth resource 19](#_Toc168645898)

[Optional: grant cluster-admin role 19](#_Toc168645899)

[Remove kubeadmin user 19](#_Toc168645900)

[Images 20](#_Toc168645901)

[Patch image pull policy 20](#_Toc168645902)

[Get tags from a particular image in a particular container image registry 20](#_Toc168645903)

[Get tags from a particular image in quay.io registry 20](#_Toc168645904)

[Get raw disk usage of all images per namespace 21](#_Toc168645905)

[Machine config 22](#_Toc168645906)

[NTP configuration 22](#_Toc168645907)

[Disable auto rebooting after a change with the machine-config-operator 23](#_Toc168645908)

[Wait for a machine-config to be applied 23](#_Toc168645909)

[Apply sysctl tweaks to nodes 23](#_Toc168645910)

[Modify MTU in a second interface in workers 24](#_Toc168645911)

[Monitoring 26](#_Toc168645912)

[Limit prometheus storage usage 26](#_Toc168645913)

[Networking 27](#_Toc168645914)

[IPv4 IPv6 DualStack 27](#_Toc168645915)

[oc 30](#_Toc168645916)

[Download and extract oc, kubectl and openshift-install one liner 30](#_Toc168645917)

[Show console URL 30](#_Toc168645918)

[Show API url 30](#_Toc168645919)

[Cluster info 30](#_Toc168645920)

[Cluster info DUMP 30](#_Toc168645921)

[Create objects using bash here documents 30](#_Toc168645922)

[Apply all manifests inside a folder 31](#_Toc168645923)

[Extract the OpenShift payloads (aka files, assets, etc.) 31](#_Toc168645924)

[Dump OpenShift release information 31](#_Toc168645925)

[Get the repositories and commits used for the OpenShift release images 31](#_Toc168645926)

[Get pull specs for all the release images 32](#_Toc168645927)

[Get the pull spec for one component’s image 32](#_Toc168645928)

[Get commit URLs for all the release components 32](#_Toc168645929)

[Get the list of images included in a release 32](#_Toc168645930)

[Get a diff from an image between different OpenShift versions 32](#_Toc168645931)

[View different channels and releases information 32](#_Toc168645932)

[Show logs for all containers running in the same pod 33](#_Toc168645933)

[Create Kubeconfig out of credentials 33](#_Toc168645934)

[Merge multiple Kubeconfigs 34](#_Toc168645935)

[Show events ordered by timestamp 35](#_Toc168645936)

[Avoid the managed fields output 35](#_Toc168645937)

[Avoid the managed fields and other metadata output 35](#_Toc168645938)

[Get nodes ready without grep or jq 35](#_Toc168645939)

[Wait for a CRD to be created 35](#_Toc168645940)

[Wait for a StorageClass to be created 36](#_Toc168645941)

[Operator-Lifecycle-Manager (OLM) 37](#_Toc168645942)

[Disable all default sources 37](#_Toc168645943)

[Use older version catalogs 37](#_Toc168645944)

[Pull Secrets 38](#_Toc168645945)

[Update pull secret without reinstalling 38](#_Toc168645946)

[Opt out telemetry 38](#_Toc168645947)

[Registries 39](#_Toc168645948)

[Configure insecure registry 39](#_Toc168645949)

[Configure custom/insecure registry to search path in OCP 4.x 39](#_Toc168645950)

[Routers 41](#_Toc168645951)

[Scale routers 41](#_Toc168645952)

[Moving routers to a specific (infra) node 41](#_Toc168645953)

[Storage 42](#_Toc168645954)

[Get default StorageClass name 42](#_Toc168645955)

[Unbound an existing pvc from one pod to be used by another pod and retaining data 42](#_Toc168645956)

[Troubleshooting 44](#_Toc168645957)

[Get the status of all the operators in your cluster 44](#_Toc168645958)

[Get pods not running nor completed 44](#_Toc168645959)

[Get cluster and operators status 44](#_Toc168645960)

[Get node logs 44](#_Toc168645961)

[Debug node issues 45](#_Toc168645962)

[Run debugging tools in the RHCOS hosts 45](#_Toc168645963)

[Copy a file to a node 45](#_Toc168645964)

[Modify kubelet log level 45](#_Toc168645965)

[Get MCP rendered ignition 46](#_Toc168645966)

[Using netcat for file transfer from emergency shell 46](#_Toc168645967)

[Upgrade cluster to latest 47](#_Toc168645968)

[Force the update to a specific version/hash 47](#_Toc168645969)

[Verify the available upgrade versions 47](#_Toc168645970)

[Switch OpenShift Channel and trigger a refresh 48](#_Toc168645971)

Labels & Annotations

## Add label to a Node

oc label node NODE\_NAME node-role.kubernetes.io/storage=

Above command is used to mark a node as a storage node using the label “node-role.kubernetes.io/storage=”

## Add node selector at namespace level

In order to run the pods of a namespace on a specific node, we add can a node selector at namespace level

For example, If a node has a label “node-role.kubernetes.io/worker=”

oc annotate namespace NAMESPACE 'openshift.io/node-selector=node-role.kubernetes.io/worker='

Add taints on a node

To control the scheduling of pods on a node, we can add taints to a node.

There are different types of default Taints available. For example, the effect “NoSchedule” prevents the scheduling of pods on a node which has this taint

oc adm taint nodes NODE\_NAME KEY=VALUE:NoSchedule

oc adm taint nodes NODE\_NAME node-role.kubernetes.io/storage:NoSchedule

Above command prevents the scheduling of pods on the nodes using the key name “node-role.kubernetes.io/storage” and the Operator “Exists” (as we have not defined any value here). To allow pods to run on the tainted node, one must add tolerations in the pod.

tolerations:

- key: node-role.kubernetes.io/storage

operator: Exists

effect: NoSchedule

## Add tolerations at namespace level

oc annotate namespace NAMESPACE 'scheduler.alpha.kubernetes.io/defaultTolerations=[{"operator": "Exists", "effect": "NoSchedule", "key": "node-role.kubernetes.io/storage"}]'

In the above command, we added the toleration for the taint “NoSchedule” having key “node-role.kubernetes.io/storage” and the operator “Exists”

API

## API resources

oc api-resources

## API resources per API group

oc api-resources --api-group config.openshift.io -o name

oc api-resources --api-group machineconfiguration.openshift.io -o name

## Explain resources

oc explain pods.spec.containers

## Explain resources per api group

oc explain --api-version=config.openshift.io/v1 scheduler

oc explain --api-version=config.openshift.io/v1 scheduler.spec

oc explain --api-version=config.openshift.io/v1 scheduler.spec.policy

oc explain --api-version=machineconfiguration.openshift.io/v1 containerruntimeconfigs

Certificates

## Sign all the pending csr

oc get csr -o name | xargs oc adm certificate approve

## Authenticate users using TLS certificates

Create a new user OCP\_USERNAME to perform operations against the API server OCP\_API\_SERVER.

export OCP\_USERNAME="alice"

export OCP\_API\_SERVER="https://api.example.com:6443"

## Generate a private key and a CSR for the new user.

mkdir ${OCP\_USERNAME}

openssl req -new -nodes -subj "/CN=${OCP\_USERNAME}" \

-keyout ${OCP\_USERNAME}/private.key -out ${OCP\_USERNAME}/request.csr

## Authenticate to Openshift API with an user with permissions to create CertificateSigningRequest objects (e.g. kube-admin).

oc login --server=${OCP\_API\_SERVER}

## Create a CertificateSigningRequest to sign the CSR by the kube-apiserver CA.

cat <<EOF | oc apply -f -

apiVersion: certificates.k8s.io/v1beta1

kind: CertificateSigningRequest

metadata:

name: tls-auth-${OCP\_USERNAME}

spec:

signerName: "kubernetes.io/kube-apiserver-client"

request: $(cat ${OCP\_USERNAME}/request.csr | base64 | tr -d '\n')

usages:

- digital signature

- key encipherment

- client auth

extra:

scopes.authorization.openshift.io:

- user:full

EOF

## Approve the pending CSR.

oc adm certificate approve tls-auth-${OCP\_USERNAME}

## Get the user certificate from the signed CSR.

oc get csr tls-auth-${OCP\_USERNAME} -o jsonpath="{.status.certificate}" |\

base64 -d > ${OCP\_USERNAME}/certificate.pem

## Get the CA chain for the API server.

oc get cm kube-apiserver-server-ca \

-o jsonpath="{.data.ca-bundle\.crt}" -n openshift-kube-apiserver > api-ca.pem

## Create a kubeconfig to authenticate the new user using the TLS certificate.

oc adm create-kubeconfig \

--kubeconfig=${OCP\_USERNAME}/kubeconfig \

--user=${OCP\_USERNAME} \

--client-certificate=${OCP\_USERNAME}/certificate.pem \

--client-key=${OCP\_USERNAME}/private.key \

--certificate-authority=api-ca.pem \

--public-master=${OCP\_API\_SERVER} \

--master=${OCP\_API\_SERVER}

## Authenticate using the new kubeconfig.

export KUBECONFIG="${OCP\_USERNAME}/kubeconfig"

## Verify the new user can make operations against the API server.

oc whoami

## Verify the API certificates

echo | openssl s\_client -connect api.ocp4.example.com:6443 | openssl x509 -noout -text

## Extract etcd CA

oc get secrets -n openshift-config etcd-signer -o "jsonpath={.data['tls\.crt']}" | base64 -d | openssl x509 -text

Clean up

## Delete ‘Completed’ pods

During the installation process, a few temporary pods are created. Keeping those pods as ‘Completed’ doesn’t harm nor waste resources but if you want to delete them to have only ‘running’ pods in your environment you can use the following command:

oc delete pod --field-selector=status.phase==Succeeded --all-namespaces

## Change the image GC thresholds

Kubernetes triggers the image garbage collector by default when the 85% (image-gc-high-threshold) of the disk has been used and the image garbage collector will try to free up to the 80% (image-gc-low-threshold). To modify those parameters a kubelet config can be created and applied to a certain labeled nodes, for example:

oc label machineconfigpool worker custom-kubelet=enabled

cat <<EOF | oc apply -f -

apiVersion: machineconfiguration.openshift.io/v1

kind: KubeletConfig

metadata:

name: custom-config

spec:

machineConfigPoolSelector:

matchLabels:

custom-kubelet: enabled

kubeletConfig:

ImageGCHighThresholdPercent: 70

ImageGCLowThresholdPercent: 60

EOF

WARNING: modifying the kubelet config will trigger an inmediate reboot of the affected nodes.

## Run a full /var/lib/containers/storage clean-up using podman

When running out of space under /var/lib/containers/storage you can run a full system prune using podman:

sudo podman system prune -a

WARNING! This will remove:

- all stopped containers

- all stopped pods

- all dangling images

- all build cache

Are you sure you want to continue? [y/N]

You can also skip the interactivity confirmation using the -f parameter:

sudo podman system prune -a -f

Deleted Pods

Deleted Containers

Deleted Images

...

Cluster

## Set control-plane nodes as NoSchedulable

oc patch schedulers.config.openshift.io/cluster --type merge --patch '{"spec":{"mastersSchedulable": false}}'

This will remove the worker label from the masters. The OCP components will be eventually moved to the workers as instructed by their node selectors but that process will only happen when the pods are rescheduled. This operation can be performed by deleting the pods and letting OpenShift reconciliation to reschedule them.

## Routers

Rollout the latest deployment to force rescheduling the router pods without losing availability:

oc rollout -n openshift-ingress restart deployment/router-default

Or delete the router pods to force the reconciliation:

oc delete pod -n openshift-ingress -l ingresscontroller.operator.openshift.io/deployment-ingresscontroller=default

## Image-registry

Rollout the latest deployment to force rescheduling the image-registry pod:

oc rollout -n openshift-image-registry restart deploy/image-registry

Or delete the image-registry pod to force the reconciliation:

oc delete pod -n openshift-image-registry -l docker-registry=default

## Monitoring stack

Rollout the latest deployments and statefulsets to force rescheduling the monitoring stack pods:

oc rollout -n openshift-monitoring restart statefulset/alertmanager-main

oc rollout -n openshift-monitoring restart statefulset/prometheus-k8s

oc rollout -n openshift-monitoring restart deployment/grafana

oc rollout -n openshift-monitoring restart deployment/kube-state-metrics

oc rollout -n openshift-monitoring restart deployment/openshift-state-metrics

oc rollout -n openshift-monitoring restart deployment/prometheus-adapter

oc rollout -n openshift-monitoring restart deployment/telemeter-client

oc rollout -n openshift-monitoring restart deployment/thanos-querier

Or delete the pods to force the reconciliation:

oc delete pod -n openshift-monitoring -l app=alertmanager

oc delete pod -n openshift-monitoring -l app=prometheus

oc delete pod -n openshift-monitoring -l app=grafana

oc delete pod -n openshift-monitoring -l app.kubernetes.io/name=kube-state-metrics

oc delete pod -n openshift-monitoring -l k8s-app=openshift-state-metrics

oc delete pod -n openshift-monitoring -l name=prometheus-adapter

oc delete pod -n openshift-monitoring -l k8s-app=telemeter-client

oc delete pod -n openshift-monitoring -l app.kubernetes.io/component=query-layer

## List all container images running in a cluster

<https://kubernetes.io/docs/tasks/access-application-cluster/list-all-running-container-images/>

oc get pods -A -o go-template --template='{{range .items}}{{range .spec.containers}}{{printf "%s\n" .image -}} {{end}}{{end}}' | sort -u | uniq

## List all container images stored in a cluster

for node in $(oc get nodes -o name);do oc debug ${node} -- chroot /host sh -c 'crictl images -o json' 2>/dev/null | jq -r .images[].repoTags[]; done | sort -u

Cluster version

## Switch clusterversion channel

oc patch \

--patch='{"spec": {"channel": "prerelease-4.1"}}' \

--type=merge \

clusterversion/version

## Unmanage operators

(via <https://github.com/openshift/cluster-version-operator/blob/master/docs/dev/clusterversion.md#setting-objects-unmanaged>)

For testing purposes the CVO can unmanage some operators, so you can alter objects without the CVO stomping on your changes by overriding the specific operator spec in the clusterversion object.

To get a list of current overrides:

oc get -o json clusterversion version | jq .spec.overrides

To add an entry to that list, you can use a JSON Patch to add a [ComponentOverride](https://godoc.org/github.com/openshift/api/config/v1#ComponentOverride). For example, to set the network operator’s deployment unmanaged:

## Get the operator deployment information

* [Extract the OpenShift payloads](https://openshift.tips/clusterversion/oc/#extract-the-openshift-payloads-aka-files-assets-etc).
* Observe the operator definition (api group, kind, name, namespace):
* head -n5 /tmp/mystuff/0000\_07\_cluster-network-operator\_03\_daemonset.yaml

In this case:

apiVersion: apps/v1

kind: Deployment

metadata:

name: network-operator

namespace: openshift-network-operator

**Note:** In this case, even if the file uses daemonset, it is in a deployment instead.

## Create the patch yaml file

* If there are currently no other overrides configured:

cat <<EOF >version-patch.yaml

- op: add

path: /spec/overrides

value:

- kind: Deployment

group: apps

name: network-operator

namespace: openshift-network-operator

unmanaged: true

EOF

* To add to list of already existing overrides:

cat <<EOF >version-patch.yaml

- op: add

path: /spec/overrides/-

value:

- kind: Deployment

group: apps

name: network-operator

namespace: openshift-network-operator

unmanaged: true

EOF

Observe the path differences if there are overrides already.

## Patch the clusterversion object

oc patch clusterversion version --type json -p "$(cat version-patch.yaml)"

You can verify the update with:

oc get -o json clusterversion version | jq .spec.overrides

Output:

[

{

"group": "apps",

"kind": "Deployment",

"name": "cluster-network-operator",

"namespace": "openshift-network-operator",

"unmanaged": true

}

]

After updating the ClusterVersion, you can make your desired edits to the unmanaged object.

## Disabling the cluster-version operator

(via <https://github.com/openshift/cluster-version-operator/blob/master/docs/dev/clusterversion.md#disabling-the-cluster-version-operator>)

When you just want to turn off the cluster-version operator instead of fiddling with per-object overrides, you can:

oc scale --replicas 0 -n openshift-cluster-version deployments/cluster-version-operator

Configuration

## Get CRI-O settings

oc get containerruntimeconfig

## OCP Master configuration

The master configuration is now stored in a configMap. During the installation process, a few configMaps are created, so in order to get the latest:

oc get cm -n openshift-kube-apiserver | grep config

Observe the latest id and then:

oc get cm -n openshift-kube-apiserver config-ID

To get the output in a human-readable form, use:

oc get cm -n openshift-kube-apiserver config-ID \

-o jsonpath='{.data.config\.yaml}' | jq

For the OpenShift api configuration:

oc get cm -n openshift-apiserver config -o jsonpath='{.data.config\.yaml}' | jq

## Observe the SDN configuration

oc get cm sdn-config -o yaml -n openshift-sdn

Or:

oc exec -n openshift-sdn $(oc get pods -n openshift-sdn -l app=sdn --no-headers=true -o custom-columns=:metadata.name|head -n1) cat /config/{kube-proxy-config,sdn-config}.yaml

## Making Master Unscheduleable/Scheduleable

To configure master as unscheduleable (when UPI installation without setting this prior to the install):

oc patch --type=merge --patch='{"spec":{"mastersSchedulable": false}}' schedulers.config.openshift.io cluster

scheduler.config.openshift.io/cluster patched

## Get Settings used in install config

oc get cm -n kube-system cluster-config-v1 -o yaml

Deployments

## Import environment variables from a file

(via <https://twitter.com/kamesh_sampath/status/1179984908690739201>)

Instead modifying the deployment, you can create an environment variables file such as myvars.env:

MYSQL\_DB="mysql"

MYSQL\_DBPORT="3306"

OTHERVAR="foo"

Then:

cat myvars.env | oc set env -e - deployment/mydeployment -n mynamespace

## Set some of the local shell environment into a deployment config

env | grep MYSQL\_ | oc set env -e - deployment/mydeployment -n mynamespace

## Import environment variables from a configmap with a prefix

oc set env --from=configmap/myconfigmap --prefix=MYSQL\_ deployment/mydeployment -n mynamespace

## Import specific keys from a configmap

oc set env --from=configmap/myconfigmap --keys=OTHERVAR deployment/mydeployment -n mynamespace

## Remove environment variable in a deployment

oc set env -e OTHERVAR- deployment/mydeployment -n mynamespace

## Remove environment variable from container ‘c1’ in a deployment

oc set env deployment/mydeployment --containers="c1" OTHERVAR-

## Remove environment variable from container ‘c1’ in all deployments

oc set env deployments --all --containers="c1" OTHERVAR-

## Show logs for all containers within a pod

oc logs <pod> --all-containers

## Show logs for all containers within a pod with pod name and container name

oc logs <pod> --all-containers --prefix

Identity Providers

## Add HTPasswd authentication (OpenShift 4 only)

## Create htpasswd file (with admin username)

htpasswd -c htpasswd admin

## Create secret in openshift-config project

oc create secret generic htpasswd-secret --from-file htpasswd -n openshift-config

## Edit cluster OAuth resource

cat << EOF | oc apply -f -

apiVersion: config.openshift.io/v1

kind: OAuth

metadata:

name: cluster

spec:

identityProviders:

- name: htpasswd

challenge: true

login: true

mappingMethod: claim

type: HTPasswd

htpasswd:

fileData:

name: htpasswd-secret

EOF

## Optional: grant cluster-admin role

oc adm policy add-cluster-role-to-user cluster-admin admin

## Remove kubeadmin user

oc delete secret kubeadmin -n kube-system

Images

## Patch image pull policy

oc patch dc mydeployment -p '{"spec":{"template":{"spec":{"containers":[{"imagePullPolicy":"IfNotPresent","name":"mydeployment"}]}}}}'

## Get tags from a particular image in a particular container image registry

In order to get images from Red Hat’s registries, it is required to have a pull secret that contains base64 encoded tokens to reach those registries, such as:

'{

"auths":{

"quay.io":{

"auth":"xxx",

"email":"xxx"

},

"registry.redhat.io":{

"auth":"xxx",

"email":"xxx"

},

"registry.example.com":{

"auth":"xxx",

"email":"xxx"

},

}

}'

First step is to get the token. We do this with this handy one liner:

REGISTRY=registry.example.com

echo $PULL\_SECRET | jq -r ".auths.\"${REGISTRY}\".auth" | base64 -d | cut -d: -f2

Or, store it in an environment variable:

TOKEN=$(echo $PULL\_SECRET | jq -r ".auths.\"${REGISTRY}\".auth" | base64 -d | cut -d: -f2)

Then we can use regular container image registry API queries:

curl -s -H "Authorization: Bearer ${TOKEN}" https://${REGISTRY}/v2/\_catalog

So, one liner to get the list of available tags for a particular image:

curl -s -H "Authorization: Bearer $(echo $PULL\_SECRET | jq -r '.auths."registry.example.com".auth' | base64 -d | cut -d: -f2)" https://registry.example.com/v2/eminguez/myawesomecontainer/tags/list | jq -r '.tags | .[]' | sort

## Get tags from a particular image in quay.io registry

If the images are public (like openshift/origin-installer), it is as simple as:

curl -X GET "https://quay.io/api/v1/repository/openshift/origin-installer/tag/" | jq -r .tags[].name | sort | uniq

Please note that [quay.io API](https://docs.quay.io/api/swagger/) is slightly different

## Get raw disk usage of all images per namespace

This script provides the sum of the size of all images per namespace algthough it’s not considering the disk saving provided by the reuse of the layers.

#/bin/bash

# use a temp file

\_tmpfile=$( mktemp )

# extract the data for each image: name and size (in Bytes)

for \_line in $( oc adm top images | head | awk '!/^NAME/ { print $2":"$NF }' | grep -v "none" | sort ); do

\_image=$( echo $\_line | cut -d ":" -f 1 )

\_size=$( echo $\_line | cut -d ":" -f 2 | sed 's/B$//' | numfmt --from=iec-i )

echo "$\_image:$\_size"

done > $\_tmpfile

# process the previous list for each namespace and show the total size in human-readable values

echo "Raw disk usage per namespace (not considering layers re-use):"

for \_namespace in $( cat $\_tmpfile | cut -d \/ -f 1 | sort -u ); do

\_size=$( grep $\_namespace $\_tmpfile | cut -d\: -f 2 | paste -s -d+ - | bc | numfmt --to=iec-i --suffix=B --padding=7 )

echo -e "- $\_namespace\t$\_size"

done

# cleanup temp dir

rm -fr $\_tmpfile

Machine config

## NTP configuration

RHCOS uses chronyd to synchronize the system time. The default configuration uses the \*.rhel.pool.ntp.org servers:

$ grep -v -E '^#|^$' /etc/chrony.conf

server 0.rhel.pool.ntp.org iburst

server 1.rhel.pool.ntp.org iburst

server 2.rhel.pool.ntp.org iburst

server 3.rhel.pool.ntp.org iburst

driftfile /var/lib/chrony/drift

makestep 1.0 3

rtcsync

logdir /var/log/chrony

As the hosts configuration shouldn’t be managed manually, in order to configure chronyd to use custom servers or a custom setting, it is required to use the machine-config-operator to modify the files used by the masters and workers by the following procedure:

Create the proper file with your custom tweaks and encode it as base64:

cat << EOF | base64

server clock.redhat.com iburst

driftfile /var/lib/chrony/drift

makestep 1.0 3

rtcsync

logdir /var/log/chrony

EOF

Create the MachineConfig file with the base64 string from the previous command as:

cat << EOF > ./masters-chrony-configuration.yaml

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfig

metadata:

labels:

machineconfiguration.openshift.io/role: master

name: masters-chrony-configuration

spec:

config:

ignition:

config: {}

security:

tls: {}

timeouts: {}

version: 2.2.0

networkd: {}

passwd: {}

storage:

files:

- contents:

source: data:text/plain;charset=utf-8;base64,c2VydmVyIGNsb2NrLnJlZGhhdC5jb20gaWJ1cnN0CmRyaWZ0ZmlsZSAvdmFyL2xpYi9jaHJvbnkvZHJpZnQKbWFrZXN0ZXAgMS4wIDMKcnRjc3luYwpsb2dkaXIgL3Zhci9sb2cvY2hyb255Cg==

verification: {}

filesystem: root

mode: 420

path: /etc/chrony.conf

osImageURL: ""

EOF

Substitute the base64 string with your own.

* oc apply -f ./masters-chrony-configuration.yaml

## Disable auto rebooting after a change with the machine-config-operator

Every change performed by the machine-config-operator triggers a reboot in the hosts where the change needs to be performed.

In the event of having a few changes to apply (such as modify NTP, registries, etc.) and specially for baremetal scenarios, the auto reboot feature can be paused by setting the spec.paused field in the machineconfigpool to true:

oc patch --type=merge --patch='{"spec":{"paused":true}}' machineconfigpool/master

## Wait for a machine-config to be applied

The machineconfigpool condition will be updated so we can wait for it as:

oc wait mcp/master --for condition=updated

## Apply sysctl tweaks to nodes

In order to modify sysctl parameters is recommended to create machine configs to add those parameters in /etc/sysctl.d/ directory.

In this example, the vm.max\_map\_count parameter will be increased to 262144 in the masters hosts:

cat << EOF | oc create -f -

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfig

metadata:

labels:

machineconfiguration.openshift.io/role: master

name: 99-sysctl-elastic

spec:

config:

ignition:

version: 2.2.0

storage:

files:

- contents:

# vm.max\_map\_count=262144

source: data:text/plain;charset=utf-8;base64,dm0ubWF4X21hcF9jb3VudD0yNjIxNDQ=

filesystem: root

mode: 0644

path: /etc/sysctl.d/99-elasticsearch.conf

EOF

## Modify MTU in a second interface in workers

Sometimes a storage network interface is attached to nodes in order to reach an external storage. In order to improve the performance, you could need to modify the MTU in those interfaces to 9000 (aka. jumboframes)

You can do that adding a script for the NetworkManager service in the /etc/NetworkManager/dispatcher.d/ path. But if SELinux is enabled in your installation you could have errors when NetworkManager runs that script. To fix it, you should add a new one-shot systemd service to modify the context.

In this example the MTU of the ens4 interface will change to 9000 to enable jumboframes:

This is the script (/etc/NetworkManager/dispatcher.d/30-mtu) for the NetworkManager.

#!/bin/sh

MTU=9000

INTERFACE=ens4

IFACE=$1

STATUS=$2

if [ "$IFACE" = "$INTERFACE" -a "$STATUS" = "up" ]; then

ip link set "$IFACE" mtu $MTU

fi

We need to encode in base64 and paste the result in the machine-config

$ cat 30-mtu | base64 -w0

IyEvYmluL3NoCk1UVT05MDAwCklOVEVSRkFDRT1lbnM0CgpJRkFDRT0kMQpTVEFUVVM9JDIKaWYgWyAiJElGQUNFIiA9ICIkSU5URVJGQUNFIiAtYSAiJFNUQVRVUyIgPSAidXAiIF07IHRoZW4KICAgIGlwIGxpbmsgc2V0ICIkSUZBQ0UiIG10dSAkTVRVCmZpCg==

cat << EOF | oc create -f -

kind: MachineConfig

apiVersion: machineconfiguration.openshift.io/v1

metadata:

name: 99-worker-mtu

creationTimestamp:

labels:

machineconfiguration.openshift.io/role: worker

spec:

osImageURL: ''

config:

ignition:

version: 2.2.0

storage:

files:

- filesystem: root

path: "/etc/NetworkManager/dispatcher.d/30-mtu"

contents:

source: data:text/plain;charset=utf-8;base64,IyEvYmluL3NoCk1UVT05MDAwCklOVEVSRkFDRT1lbnM0CgpJRkFDRT0kMQpTVEFUVVM9JDIKaWYgWyAiJElGQUNFIiA9ICIkSU5URVJGQUNFIiAtYSAiJFNUQVRVUyIgPSAidXAiIF07IHRoZW4KICAgIGlwIGxpbmsgc2V0ICIkSUZBQ0UiIG10dSAkTVRVCmZpCg==

verification: {}

mode: 0755

systemd:

units:

- contents: |

[Unit]

Requires=systemd-udevd.target

After=systemd-udevd.target

Before=NetworkManager.service

DefaultDependencies=no

[Service]

Type=oneshot

ExecStart=/usr/sbin/restorecon /etc/NetworkManager/dispatcher.d/30-mtu

[Install]

WantedBy=multi-user.target

name: one-shot-mtu.service

enabled: true

EOF

Monitoring

## Limit prometheus storage usage

If not configured properly, prometheus can take up all the storage on the nodes, leading to a blocked cluster. This is very common in environments where nodes are running on low storage VMs.

To limit the usage apply the following configuration:

apiVersion: v1

kind: ConfigMap

metadata:

name: cluster-monitoring-config

namespace: openshift-monitoring

data:

config.yaml: |

prometheusK8s:

retention: 24h

Networking

## IPv4 IPv6 DualStack

At the time of writting this document the priority IP protocol inside and OpenShift cluster is IPv4, that means that by default the first IP that are getting the nodes, pods and services is IPv4.

**Installation**

The install-config.yaml file needs to include both network configurations, IPv4 and IPv6, an example:

networking:

networkType: OVNKubernetes

machineNetwork:

- cidr: 192.168.111.0/24

- cidr: fd2e:6f44:5dd8:c956::/120

clusterNetwork:

- cidr: 10.128.0.0/14

hostPrefix: 23

- cidr: fd01::/48

hostPrefix: 64

serviceNetwork:

- 172.30.0.0/16

- fd02::/112

...

platform:

baremetal:

provisioningBridge: ostestpr

provisioningNetworkCIDR: fd00:1101::0/64

provisioningNetworkInterface: enp1s0

externalBridge: ostestbm

bootstrapOSImage: http://192.168.111.1/images/rhcos-47.83.202101161239-0-qemu.x86\_64.qcow2.gz?sha256=3a14ff77b4b7a5c89d145226759c71e852bd54eb8eea50866e760c801c7b623a

clusterOSImage: http://192.168.111.1/images/rhcos-47.83.202101161239-0-openstack.x86\_64.qcow2.gz?sha256=ccc2c776ce3d4bbb7585fdf497286c3694633f609bf7aeee42c5f8c274560bd2

apiVIP: 192.168.111.5

ingressVIP: 192.168.111.4

Pay special attention to the order of the networks, first IPv4, second IPv6, at the moment that is important and mandatory.

The IPv6DualStackNoUpgrade *FeatureGate* also needs to be enabled by adding the following to the list of manifests:

apiVersion: config.openshift.io/v1

kind: FeatureGate

metadata:

name: cluster

spec:

featureSet: IPv6DualStackNoUpgrade

**Setting up IPv4, IPv6 or DualStack in a service**

After the cluster is up and running you can test the IPv6 configuration with a Service. By default the Services are created IPv4 only, if you want to create a Service with IPv6 you have to modify the spec.ipFamilyPolicy setting.

The default setting is to expose only as a SingleStack cluster IP for the Service, using the first configured service cluster IP range (IPv4 by default):

apiVersion: v1

kind: Service

metadata:

labels:

app: hello-openshift

app.kubernetes.io/component: hello-openshift

app.kubernetes.io/instance: hello-openshift

name: hello-openshift-v6

namespace: hello-openshift

spec:

ipFamilies:

- IPv6

ipFamilyPolicy: SingleStack

ports:

- name: 8080-tcp

port: 8080

protocol: TCP

targetPort: 8080

- name: 8888-tcp

port: 8888

protocol: TCP

targetPort: 8888

selector:

deployment: hello-openshift

sessionAffinity: None

type: ClusterIP

In order to use DualStack, the spec.ipFamilyPolicy setting needs to be configured whether to PreferDualStack or RequireDualStack as:

apiVersion: v1

kind: Service

metadata:

labels:

app: hello-openshift

app.kubernetes.io/component: hello-openshift

app.kubernetes.io/instance: hello-openshift

name: hello-openshift-dual

namespace: hello-openshift

spec:

ipFamilies:

- IPv6

- IPv4

ipFamilyPolicy: PreferDualStack

ports:

- name: 8080-tcp

port: 8080

protocol: TCP

targetPort: 8080

- name: 8888-tcp

port: 8888

protocol: TCP

targetPort: 8888

selector:

deployment: hello-openshift

sessionAffinity: None

type: ClusterIP

For more information about IPv4/IPv6 dual stack you can check the [upstream documentation](https://kubernetes.io/docs/concepts/services-networking/dual-stack/)

oc

## Download and extract oc, kubectl and openshift-install one liner

curl -sL https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/openshift-client-linux-${OCPVERSION}.tar.gz | sudo tar -C /usr/local/bin -xzf - oc kubectl

curl -sL https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/openshift-install-linux-${OCPVERSION}.tar.gz | sudo tar -C /usr/local/bin -xzf - openshift-install

## Show console URL

oc whoami --show-console

## Show API url

oc whoami --show-server

## Cluster info

oc cluster-info

## Cluster info DUMP

oc cluster-info dump

## Create objects using bash here documents

This is just an example of a LoadBalancer service, but it can be anything yaml based!:

cat <<EOF | oc apply -f -

apiVersion: v1

kind: Service

metadata:

name: hello-openshift-lb

spec:

externalTrafficPolicy: Cluster

ports:

- name: http

port: 80

protocol: TCP

targetPort: 8080

selector:

app: hello-openshift

sessionAffinity: None

type: LoadBalancer

EOF

## Apply all manifests inside a folder

oc apply -R -f myfolder/

## Extract the OpenShift payloads (aka files, assets, etc.)

You just need your pull secret file and:

oc adm release extract --registry-config=./pull\_secret.txt --from=quay.io/openshift-release-dev/ocp-release:4.1.15 --to=/tmp/mystuff

In case you want the payloads from the current version running in the cluster:

oc adm release extract --registry-config=./pull-secret.txt --from=$(oc get clusterversion version -o jsonpath='{.status.desired.image}') --to=/tmp/mystuff

You can extract individual files such as the oc or the installer with the --command flag

## Dump OpenShift release information

Get the tag you are interested in by visiting [the openshift-release-dev](https://quay.io/repository/openshift-release-dev/ocp-release?tab=tags) repository.

Then:

oc adm release info quay.io/openshift-release-dev/ocp-release:<version>

For a super verbose and huge json file with all the details:

oc adm release info --contents quay.io/openshift-release-dev/ocp-release:<version>

## Get the repositories and commits used for the OpenShift release images

oc adm release info --commits quay.io/openshift-release-dev/ocp-release:<version>

For instance, the multus-cni one for 4.1.18:

oc adm release info --commits quay.io/openshift-release-dev/ocp-release:4.1.18 | grep multus-cni

multus-cni https://github.com/openshift/multus-cni 0ad77469f3dbe7fa0a9cf5df5cd2a7fd3f099d2a

If you prefer the commit URLs directly, use this command instead.

oc adm release info --commit-urls quay.io/openshift-release-dev/ocp-release:<version>

## Get pull specs for all the release images

oc adm release info --pullspecs quay.io/openshift-release-dev/ocp-release:<version>

## Get the pull spec for one component’s image

oc adm release info --image-for=<component> quay.io/openshift-release-dev/ocp-release:<version>

## Get commit URLs for all the release components

So, you can go to <https://github.com/openshift/multus-cni/commits/0ad77469f3dbe7fa0a9cf5df5cd2a7fd3f099d2a> to see the actual code.

## Get the list of images included in a release

RELEASE="4.6.16"

FROM=$(curl -s https://mirror.openshift.com/pub/openshift-v4/x86\_64/clients/ocp/${RELEASE}/release.txt | awk '/Pull From:/ { print $3 }')

oc adm release info ${RELEASE} | awk '/NAME/,0 { getline; print $1 }'

## Get a diff from an image between different OpenShift versions

IMAGE="cli"

FROMVER="4.6.1"

TOVER="4.6.16"

FROM=$(curl -s https://mirror.openshift.com/pub/openshift-v4/x86\_64/clients/ocp/${FROMVER}/release.txt | awk '/Pull From:/ { print $3 }')

TO=$(curl -s https://mirror.openshift.com/pub/openshift-v4/x86\_64/clients/ocp/${TOVER}/release.txt | awk '/Pull From:/ { print $3 }')

FROMCOMMIT=$(oc adm release info --commits ${FROM} | grep " ${IMAGE} " | awk "{ print \$3 }")

TOCOMMIT=$(oc adm release info --commits ${TO} | grep " ${IMAGE} " | awk "{ print \$3 }")

REPOSITORY=$(oc adm release info --commits ${TO} | grep " ${IMAGE} " | awk "{ print \$2 }")

echo "${REPOSITORY}/compare/${FROMCOMMIT:0:6}..${TOCOMMIT:0:6}#files\_bucket"

## View different channels and releases information

Kudos to [Ryan Howe](https://github.com/rjhowe)

OCP4 is released in different ‘channels’ (“prerelease-4.1”, “stable-4.1”, “candidate-4.2”, “fast-4.2”, “stable-4.2”,…) that contains different releases. In order to view the different releases and some information, the following snippet can be used (in this example the “stable-4.2” channel is used):

curl -sH 'Accept: application/json' "https://api.openshift.com/api/upgrades\_info/v1/graph?channel=stable-4.2&arch=amd64" | jq -S '.nodes | sort\_by(.version | sub ("-rc";"") | split(".") | map(tonumber)) | .[]'

Output:

{

"metadata": {

"description": "",

"io.openshift.upgrades.graph.release.channels": "stable-4.2",

"io.openshift.upgrades.graph.release.manifestref": "sha256:c5337afd85b94c93ec513f21c8545e3f9e36a227f55d41bc1dfb8fcc3f2be129",

"url": "https://access.redhat.com/errata/RHBA-2019:2922"

},

"payload": "quay.io/openshift-release-dev/ocp-release@sha256:c5337afd85b94c93ec513f21c8545e3f9e36a227f55d41bc1dfb8fcc3f2be129",

"version": "4.2.0"

}

This can be wrapped in a handy script such as:

#!/bin/bash

PS3='Please enter the channel: '

options=("prerelease-4.1" "stable-4.1" "candidate-4.2" "fast-4.2" "stable-4.2")

PS3='Please enter the arch: '

options2=("amd64" "s390x" "ppc64le")

\_Command () {

echo "Showing upgrade channel: ${channel} arch: ${arch}"

curl -sH 'Accept: application/json' "https://api.openshift.com/api/upgrades\_info/v1/graph?channel=${channel}&arch=${arch}" | jq -S '.nodes | sort\_by(.version | sub ("-rc";"") | split(".") | map(tonumber)) | .[]'

}

select opt2 in "${options2[@]}"

do

select opt in "${options[@]}"

do

channel="${opt}"

arch="${opt2}"

\_Command

break

done

break

done

## Show logs for all containers running in the same pod

oc logs <pod> --all-containers

## Create Kubeconfig out of credentials

For creating a Kubeconfig file from a given credentials we can run the following commands:

export KUBECONFIG=/path/to/new/kubeconfig/file

oc login -u kubeadmin -p <your\_password> https://api.<cluster\_name>.<base\_domain>:6443

oc config view --flatten > ${KUBECONFIG}

## Merge multiple Kubeconfigs

If you want to merge multiple kubeconfig files you can run the following commands:

Ensure there aren’t duplicated entries (Context Names or User Names)

grep -A3 -x '\- context:' kubeconfig1 kubeconfig2 kubeconfig3 | egrep "name|user"

kubeconfig1- user: admin

kubeconfig1- name: admin

kubeconfig2- user: admin

kubeconfig2- name: admin

kubeconfig3- user: admin

kubeconfig3- name: admin

There are duplicated users and names so we need to edit the kubeconfig files and assign a correct value for each kubeconfig file

sed -i 's/admin/kube1/' kubeconfig1

sed -i 's/admin/kube2/' kubeconfig2

sed -i 's/admin/kube3/' kubeconfig3

Now we’re ready to merge the three kubeconfigs into a single one

**Export all three kubeconfigs**

export KUBECONFIG=/path/to/kubeconfig1:/path/to/kubeconfig2:/path/to/kubeconfig3

**Explore the context created**

oc config get-contexts

CURRENT NAME CLUSTER AUTHINFO NAMESPACE

\* kube1 hub kube1

kube2 spoke kube2

kube3 spoke2 kube3

**Merge the configs**

oc config view --flatten > /path/to/merged/kubeconfig

**Check merged config file**

oc --config /path/to/merged/kubeconfig config view

Now you can export the new Kubeconfig and use --context [kube1|kube2|kube3] or oc config use [kube1|kube2|kube3] work to with the different clusters

oc --context kube1 get clusterversion

NAME VERSION AVAILABLE PROGRESSING SINCE STATUS

version 4.4.0-0.nightly-2020-03-23-010639 True False 53m Cluster version is 4.4.0-0.nightly-2020-03-23-010639

oc config use kube2

Switched to context "kube2".

oc get clusterversion

NAME VERSION AVAILABLE PROGRESSING SINCE STATUS

version 4.4.0-0.nightly-2020-03-26-010528 True False 52m Cluster version is 4.4.0-0.nightly-2020-03-26-010528

## Show events ordered by timestamp

oc get event --sort-by=.metadata.creationTimestamp

## Avoid the managed fields output

For instance, to ‘export’ the restricted SCC without the metadata.managedFields:

oc patch scc restricted --type=json -p '[{"op": "remove", "path": "/metadata/managedFields"}]' -o yaml --dry-run=client

This won’t affect the object in the cluster as it is done at client side with the --dry-run=client flag.

Via <https://github.com/kubernetes/kubernetes/issues/90066#issuecomment-716206402>

## Avoid the managed fields and other metadata output

For instance, to ‘export’ the restricted SCC without the metadata.managedFields, creationTimestamp, generation, etc.:

oc patch scc restricted --type=json -p '[{"op": "remove", "path": "/metadata/managedFields"},{"op": "remove", "path": "/metadata/creationTimestamp"},{"op": "remove", "path": "/metadata/generation"},{"op": "remove", "path": "/metadata/resourceVersion"},{"op": "remove", "path": "/metadata/selfLink"},{"op": "remove", "path": "/metadata/uid"}]' -o yaml --dry-run=client

This won’t affect the object in the cluster as it is done at client side with the --dry-run=client flag.

## Get nodes ready without grep or jq

oc get nodes -o go-template='{{range .items}}{{$node := .}}{{range .status.conditions}}{{if eq .type "Ready"}}{{if eq .status "True"}}node/{{$node.metadata.name}}{{"\n"}}{{end}}{{end}}{{end}}{{end}}'

## Wait for a CRD to be created

until oc wait crd/localvolumes.local.storage.openshift.io --for condition=established --timeout 10s >/dev/null 2>&1 ; do sleep 1 ; done

## Wait for a StorageClass to be created

StorageClass object doesn’t have a condition field, so instead, wait for the object to be created:

until oc get sc/local-sc >/dev/null 2>&1 ; do sleep 1 ; done

Operator-Lifecycle-Manager (OLM)

## Disable all default sources

oc patch operatorhub.config.openshift.io/cluster -p='{"spec":{"disableAllDefaultSources":true}}' --type=merge

## Use older version catalogs

This is sometimes useful when you want to install older content on a newer cluster (or the riskier opposite option).

I took a snapshot of the existing catalogsources and modified them to point to 4.9 then disabled default sources and applied my non default sources (under the same names) this is the procedure:

curl -s -L https://github.com/itaysk/kubectl-neat/releases/download/v2.0.3/kubectl-neat\_linux\_amd64.tar.gz | tar xvz -C ~/bin/

oc project openshift-marketplace

oc neat get catalogsource > sources.yml

sed -i 's/v4.10/v4.9/g' sources.yml

oc patch operatorhub.config.openshift.io/cluster -p='{"spec":{"disableAllDefaultSources":true}}' --type=merge

oc apply -f sources.yml

I’m using neat to ease getting cluster resources without all the “junk” around them (so included here the command to download that plugin).

Pull Secrets

## Update pull secret without reinstalling

The pull secret required be able to pull images from the Red Hat registries is stored in the pull-secret secret hosted in the openshift-config namespace.

It is just a matter of modifying that secret with the updated one (in base64):

oc edit secret -n openshift-config pull-secret

NOTE: That secret is translated by the machine-config operator into the /var/lib/kubelet/config.json file so in order to update it is required for the hosts to be rebooted (which is done automatically by the mc operator)

## Opt out telemetry

oc -n openshift-config create secret generic pull-secret --from-file=.dockerconfigjson=<(oc extract secret/pull-secret -n openshift-config --to=- | jq -M 'del(.auths["cloud.openshift.com"])') --dry-run=client -o yaml | oc -n openshift-config apply --filename=-

NOTE: This will trigger a reboot in all the nodes.

Registries

## Configure insecure registry

oc patch image.config.openshift.io/cluster -p \

'{"spec":{"allowedRegistriesForImport":[{"domainName":"my.own.registry.example.com:8888","insecure":true}],"registrySources":{"insecureRegistries":["my.own.registry.example.com:8888"]}}}' --type='merge'

## Configure custom/insecure registry to search path in OCP 4.x

In OpenShift 4, the registries configuration is managed by the [Image Registry Operator](https://docs.openshift.com/container-platform/4.3/registry/configuring-registry-operator.html). In order to modify registries parameter, it is only required to modify the image.config.openshift.io/cluster object, that manages the /etc/containers/{policy.json,registries.conf} files content under the hood.

In OpenShift versions prior to 4.4 a missing feature of the operator is the ability to remove, modify or append any additional entry to the unqualified-search-registries line in the /etc/containers/registries.conf file to allow search in insecure registries.

Meanwhile, the current workaround is to modify the /etc/containers/{policy.json,registries.conf} files using a machineconfig object instead.

You need to be aware that these files will be overwritten if the image.config.openshift.io/cluster object is modified, as they are intended to be managed by the operator. If the object is modifed, the operator will modify the 99-master/worker-<uuid>-registries machineconfig object and will bring back the previous unqualified-search-registries content to the file, so you must avoid to use the image.config.openshift.io/cluster after you applied the machineconfig.

The following snippet shows the content of the machineconfig object that modifies the unqualified-search-registries parameter:

apiVersion: machineconfiguration.openshift.io/v1

kind: MachineConfig

metadata:

labels:

machineconfiguration.openshift.io/role: <MachineConfigPool>

name: 99-<MachineConfigPool>-container-runtime

spec:

config:

ignition:

config: {}

security:

tls: {}

timeouts: {}

version: 2.2.0

networkd: {}

passwd: {}

storage:

files:

- contents:

source: data:text/plain;charset=utf-8;base64,<base64\_content>

verification: {}

filesystem: root

mode: 420

path: /etc/containers/registries.conf

- contents:

source: data:text/plain;charset=utf-8;base64,<base64\_content>

verification: {}

filesystem: root

mode: 420

path: /etc/containers/policy.json

systemd: {}

fips: false

kernelArguments: null

osImageURL: ""

NOTE: MachineConfigPool possible values are **worker** or **master**. base64\_content is the full content of the config files (including the unqualified-search-registries parameter). In order to get the entire content of the file, you can connect to any of the hosts and extract the content (you can use oc debug node/<mynode> as cluster-admin user)

Once the previous file is properly created with the proper file content and the roles to be applied (master, worker, etc.), the MachineConfig needs to be applied to the cluster as:

oc create -f 99\_<MachineConfigPool>\_container\_runtime.yaml

The modification will trigger a reboot on the hosts, so in order to wait for the nodes to be ready, you can use the command below:

oc wait mcp/<MachineConfigPool> --for condition=updated --timeout=600s

Routers

## Scale routers

oc patch \

--namespace=openshift-ingress-operator \

--patch='{"spec": {"replicas": 1}}' \

--type=merge \

ingresscontroller/default

## Moving routers to a specific (infra) node

Label the desired nodes with a particular label (infra) and add a taint to those nodes:

oc label node worker1 node-role.kubernetes.io/infra=

oc label node worker2 node-role.kubernetes.io/infra=

oc adm taint nodes -l node-role.kubernetes.io/infra infra=reserved:NoSchedule infra=reserved:NoExecute

Patch the ingresscontroller to use the nodes with specific nodeselector and a toleration for the previous taint:

oc patch ingresscontroller/default -n openshift-ingress-operator --type=merge -p '{"spec":{"nodePlacement": {"nodeSelector": {"matchLabels": {"node-role.kubernetes.io/infra": ""}},"tolerations": [{"effect":"NoSchedule","key": "infra","value": "reserved"},{"effect":"NoExecute","key": "infra","value": "reserved"}]}}}'

Storage

## Get default StorageClass name

oc get sc -o jsonpath='{.items[?(@.metadata.annotations.storageclass\.kubernetes\.io/is-default-class=="true")].metadata.name}'

## Unbound an existing pvc from one pod to be used by another pod and retaining data

Scale pods to 0

oc scale --replicas=0 deployment/victoria

Edit deployment and delete volumes that make a reference to pvc

oc edit deployment/victoria

apiVersion: apps/v1

kind: Deployment

metadata:

annotations:

deployment.kubernetes.io/revision: "7"

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

...

spec:

containers:

- args:

- -selfScrapeInterval=10s

- -dedup.minScrapeInterval=60s

image: victoriametrics/victoria-metrics

imagePullPolicy: Always

name: victoria

ports:

- containerPort: 8428

protocol: TCP

resources: {}

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

volumeMounts:

- mountPath: /victoria-metrics-data

name: victoria-data

dnsPolicy: ClusterFirst

restartPolicy: Always

schedulerName: default-scheduler

securityContext: {}

terminationGracePeriodSeconds: 30

volumes:

- name: victoria-data

persistentVolumeClaim:

claimName: pvc-victoria-data2

Remove claimRef to pvc on pv and make sure persistentVolumeReclaimPolicy: Retain

oc edit pv/pv-victoria-data

apiVersion: v1

kind: PersistentVolume

metadata:

annotations:

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

...

spec:

accessModes:

- ReadWriteMany

capacity:

storage: 10Gi

claimRef:

apiVersion: v1

kind: PersistentVolumeClaim

name: pvc-victoria-data2

namespace: victoria

resourceVersion: "33452972"

uid: 9d4fb2ac-b09d-4110-b337-1d93a34279f7

nfs:

path: /export/data

server: helper.ocp4.info.net

persistentVolumeReclaimPolicy: Retain

volumeMode: Filesystem

status:

phase: Bound

Make sure PV is available

oc get pv

NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE

pv-victoria-data 10Gi RWX Retain Available

At this point PV is ready to be used by another pod

Troubleshooting

## Get the status of all the operators in your cluster

The following command is the easiest way to see the status of the cluster:

oc get clusteroperators

A little addon to the previous command very useful when you are upgrading your cluster:

watch -n5 oc get clusteroperators

## Get pods not running nor completed

A handy one liner to see the pods having issues (such as CrashLoopBackOff):

oc get pods -A -o wide | grep -v -E 'Completed|Running'

## Get cluster and operators status

Combining the previous two tips and adding some more data:

watch -n 30 "oc get nodes; oc get pods -A -o wide | grep -v -E 'Completed|Running'; oc get clusteroperators | grep -v 'True[[:space:]]\+False[[:space:]]\+False'; oc get clusterversion; oc get machines -A; oc get machineconfigpool"

## Get node logs

Display node journal:

oc adm node-logs <node>

Tail 10 lines from node journal:

oc adm node-logs --tail=10 <node>

Get kubelet journal logs only:

oc adm node-logs -u kubelet.service <node>

Grep kernel word on node journal:

oc adm node-logs --grep=kernel <node>

List /var/log contents:

oc adm node-logs --path=/ <node>

Get /var/log/audit/audit.log from node:

oc adm node-logs --path=audit/audit.log <node>

## Debug node issues

OCP 4.1 is based on RHCOS and it is encouraged to not ssh into the hosts. Instead:

oc debug node/<node>

...

cat /host/etc/redhat-release

# If you want to use the node binaries you can:

# chroot /host

## Run debugging tools in the RHCOS hosts

oc debug node/<node>

chroot /host

podman run -it --name rhel-tools --privileged \

--ipc=host --net=host --pid=host -e HOST=/host \

-e NAME=rhel-tools -e IMAGE=rhel7/rhel-tools \

-v /run:/run -v /var/log:/var/log \

-v /etc/localtime:/etc/localtime -v /:/host rhel7/rhel-tools

or you can specify the image used for the debug pod as:

oc debug node/<node> --image=rhel7/rhel-tools

This will allow you to run tcpdump and other tools. Use it with caution!!!

## Copy a file to a node

(Ab)using oc debug

echo "test" >> ./myfile

oc debug node/<node> --image rhel7/rhel-tools -- \

bash -c 'cat > host/tmp/myfile-remote' <(cat myfile )

## Modify kubelet log level

The kubelet configuration is provided by the systemd unit file in /etc/systemd/system/kubelet.service which is created by the 01-worker-kubelet (for workers) or 01-master-kubelet machineconfig. In current OpenShift versions, that unit sets the -v parameter as per KUBELET\_LOG\_LEVEL environment variable, so customizing the log level is as simple as setting that variable through a drop-in for the kubelet systemd service unit, like this:

Connect to the node via oc debug node

oc debug node/<node>

...

chroot /host

Create a systemd drop-in that sets KUBELET\_LOG\_LEVEL to the desired value (4 in our example)

cat <<EOF > /etc/systemd/system/kubelet.service.d/40-logging.conf

[Service]

Environment="KUBELET\_LOG\_LEVEL=4"

EOF

Reload systemd and restart the service:

systemctl daemon-reload

systemctl restart kubelet

Alternatively, this drop-in could be specified via machineconfig if the log levels of all the nodes need to be changed.

## Get MCP rendered ignition

curl -k -H "Accept: application/vnd.coreos.ignition+json; version=3.1.0" https://<api\_ip>:22623/config/<poolname>

for example:

curl -k -H "Accept: application/vnd.coreos.ignition+json; version=3.1.0" https://<api\_ip>:22623/config/master

or

curl -k -H "Accept: application/vnd.coreos.ignition+json; version=3.1.0" https://<api\_ip>:22623/config/worker

## Using netcat for file transfer from emergency shell

Sometimes things go so badly that we end up with node in emergency shell. With this we can copy off journal (or any other relevant file) outside of that shell so we can attach it to a bug report or examine it with other tools. First off save the journal to a file:

journalctl > journal.log

On the receving end run:

nc -l -p 1234 > journal.log

And then on the emergency console:

nc -w 3 [destination] 1234 < journal.log

You’ll end up with journal.log on the destination

Upgrades

## Upgrade cluster to latest

oc adm upgrade --to-latest

## Force the update to a specific version/hash

1. Get the hash of the image version

CHANNEL='prerelease-4.1'

ARCH='amd64'

curl -sH 'Accept: application/json' "https://api.openshift.com/api/upgrades\_info/v1/graph?channel=${CHANNEL}&${ARCH}" | jq .

1. Apply the update

oc adm upgrade --allow-explicit-upgrade --force=true --to-image=quay.io/openshift-release-dev/ocp-release@sha256:7e1e73c66702daa39223b3e6dd2cf5e15c057ef30c988256f55fae27448c3b01

## Verify the available upgrade versions

Depending on the OCP version you can upgrade to some specific versions.

For 4.1.10 for amd64:

curl -s -XGET "https://api.openshift.com/api/upgrades\_info/v1/graph?channel=stable-4.1&arch=amd64" --header 'Accept:application/json' |jq '. as $graph | $graph.nodes | map(.version == "4.1.10") | index(true) as $orig | $graph.edges | map(select(.[0] == $orig)[1]) | map($graph.nodes[.])'

Output is something similar to:

[

{

"version": "4.1.11",

"payload": "quay.io/openshift-release-dev/ocp-release@sha256:bfca31dbb518b35f312cc67516fa18aa40df9925dc84fdbcd15f8bbca425d7ff",

"metadata": {

"description": "",

"url": "https://access.redhat.com/errata/RHBA-2019:2417",

"io.openshift.upgrades.graph.release.manifestref": "sha256:bfca31dbb518b35f312cc67516fa18aa40df9925dc84fdbcd15f8bbca425d7ff",

"io.openshift.upgrades.graph.release.channels": "stable-4.1"

}

}

]

## Switch OpenShift Channel and trigger a refresh

For instance, switching from stable-4.6 to fast-4.6:

oc patch clusterversion version --type="merge" -p '{"spec":{"channel":"fast-4.6"}}'

oc delete po -n openshift-cluster-version -l k8s-app=cluster-version-operator

Then you can update to latest as:

oc adm upgrade --to-latest