OpenShift Container Platform 3.7

Day Two Operations Guide

OpenShift Container Platform 3.7 Day Two Operations Guide

Last Updated: 2018-06-25

OpenShift Container Platform 3.7 Day Two Operations Guide

OpenShift Container Platform 3.7 Day Two Operations Guide

Legal Notice

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

While the OpenShift Container Platform Cluster administration guide is focused more on configuration, this guide will describe an overview of common daily maintenance tasks.

Table of Contents

CHAPTER 1. OVERVIEW **5**

CHAPTER 2. RUN-ONCE TASKS **6**
2.1. NTP SYNCHRONIZATION 6
2.2. ENTROPY 6
2.3. CHECKING THE DEFAULT STORAGE CLASS 7

CHAPTER 3. ENVIRONMENT HEALTH CHECKS **9**
3.1. CHECKING COMPLETE ENVIRONMENT HEALTH 9
Procedure 9
3.2. HOST HEALTH 9
3.3. ROUTER AND REGISTRY HEALTH 10
3.4. NETWORK CONNECTIVITY 11
3.4.1. Connectivity on master hosts 12
3.4.2. Connectivity on node instances 13
Procedure 13
3.5. STORAGE 15
3.6. DOCKER STORAGE 16
3.7. API SERVICE STATUS 17
3.8. CONTROLLER ROLE VERIFICATION 17
3.9. VERIFYING CORRECT MAXIMUM TRANSMISSION UNIT (MTU) SIZE 18
Prerequisites 19

CHAPTER 4. CREATING AN ENVIRONMENT-WIDE BACKUP **21**
4.1. CREATING A MASTER HOST BACKUP 21
Procedure 21
4.2. CREATING A NODE HOST BACKUP 25
Procedure 25
4.3. ETCD BACKUP 28
4.3.1. etcd configuration backup 28
4.3.2. etcd data backup 28
Prerequisites 29
Procedure 30
4.4. CREATING A PROJECT BACKUP 31
4.4.1. Back up a project 31
Procedure 31
4.4.2. Restore project 35
Procedure 35
4.5. CREATING A PVC BACKUP 39
4.5.1. Backup persistent volume claims 39
Procedure 39
4.5.2. Restore persistent volume claims 40
4.5.2.1. Restoring files to an existing PVC 40
Procedure 40
4.5.2.2. Restoring data to a new PVC 41
Procedure 41

CHAPTER 5. HOST-LEVEL TASKS **43**
5.1. ADDING A HOST TO THE CLUSTER 43
5.2. MASTER HOST TASKS 43
5.2.1. Deprecating a master host 43
5.2.1.1. Deprecating a master host without collocated etcd 43

Procedure 43
5.2.1.2. Deprecating master host with collocated etcd 45
5.2.1.3. Replacing a master host 45
5.2.2. Creating a master host backup 45
Procedure 45
5.2.3. Restoring a master host backup 49
Procedure 49
5.3. NODE HOST TASKS 50
5.3.1. Deprecating a node host 50
Prerequisites 50
Procedure 50
5.3.1.1. Replacing a node host 57
5.3.2. Creating a node host backup 57
Procedure 58
5.3.3. Restoring a node host backup 60
Procedure 60
5.3.4. Node maintenance and next steps 61
5.4. ETCD TASKS 61
5.4.1. etcd backup 61
5.4.2. etcd configuration backup 62
5.4.3. etcd data backup 62
Prerequisites 62
Procedure 64
5.4.4. etcd restore 65
5.4.4.1. Restoring etcd v2 & v3 data 65
Procedure 65
Procedure 67
5.4.4.2. Restoring etcd for v3 67
Procedure 68
5.4.5. etcd restore 68
Procedure 68
5.4.6. Scaling etcd 70
Prerequisites 70
5.4.6.1. Adding a new etcd host using Ansible 71
Procedure 71
5.4.6.2. Manually adding a new etcd host 72
Procedure 73
Procedure 74
Procedure 76
5.4.7. Removing an etcd host 77
Procedure 77
Procedure 78

CHAPTER 6. PROJECT-LEVEL TASKS 80
6.1. PROJECT BACKUP 80
6.1.1. Back up a project 80
Procedure 80
6.1.2. Restore project 84
Procedure 84
6.2. PVC BACKUP 87
6.2.1. Backup persistent volume claims 87
Procedure 87
6.2.2. Restore persistent volume claims 89

6.2.2.1. Restoring files to an existing PVC 89
Procedure 89
6.2.2.2. Restoring data to a new PVC 89
Procedure 89
6.3. PRUNING IMAGES AND CONTAINERS 90

CHAPTER 7. DOCKER TASKS 92
7.1. INCREASING DOCKER STORAGE 92
7.1.1. Evacuating the node 92
Procedure 92
7.1.2. Increasing storage 92
Prerequisites 93
Procedure 93
7.1.3. Changing the storage backend 95
7.1.3.1. Evacuating the node 95
7.2. MANAGING DOCKER CERTIFICATES 97
7.2.1. Installing a certificate authority certificate for external registries 98
Procedure 98
7.2.2. Docker certificates backup 99
Procedure 99
7.2.3. Docker certificates restore 100
7.3. MANAGING DOCKER REGISTRIES 100
7.3.1. Docker search external registries 100
Procedure 100
7.3.2. Docker external registries whitelist and blacklist 101
Procedure 101
7.3.3. Secure registries 103
7.3.4. Insecure registries 103
Procedure 103
7.3.5. Authenticated registries 104
Procedure 104
7.3.6. ImagePolicy admission plug-in 106
Procedure 106
7.3.7. Import images from external registries 107
Procedure 107
7.3.8. OpenShift Container Platform registry integration 109
7.3.8.1. Connect the registry project with the cluster 109
Procedure 110

CHAPTER 1. OVERVIEW

This section is built for OpenShift Container Platform administrators with a fresh installation.

While the OpenShift Container Platform Cluster administration guide is focused more on configuration, this guide describes an overview of common daily maintenance tasks.

CHAPTER 2. RUN-ONCE TASKS

After installing OpenShift Container Platform, your system might need extra configuration to ensure your hosts consistently run smoothly.

While these are classified as run-once tasks, you can perform any of these at any time if any circumstances change.

2.1. NTP SYNCHRONIZATION

NTP (Network Time Protocol) is for keeping hosts in sync with the world clock. Time synchronization is important for time sensitive operations, such as log keeping and time stamps, and is highly recommended for Kubernetes, which OpenShift Container Platform is built on. OpenShift Container Platform operations include etcd leader election, health checks for pods and some other issues, and helps prevent time skew problems.

Depending on your instance, NTP might not be enabled by default. To verify that a host is configured to use NTP:

```
$ timedatectl
      Local time: Thu 2017-12-21 14:58:34 UTC
  Universal time: Thu 2017-12-21 14:58:34 UTC
        RTC time: Thu 2017-12-21 14:58:34
       Time zone: Etc/UTC (UTC, +0000)
     NTP enabled: yes
NTP synchronized: yes
 RTC in local TZ: no
      DST active: n/a
```

If both **`NTP enabled`** and **`NTP synchronized`** are **`yes`**, then NTP synchronization is active.

If **no**, install and enable the **`ntp`** or **`chrony`** RPM package.

For NTP:

```
# timedatectl set-ntp true
```

For chrony:

```
# yum install chrony
# systemctl enable chronyd --now
```

IMPORTANT

Time synchronization should be enabled on all hosts in the cluster, whether using NTP or any other method.

For more information about the **`timedatectl`** command, timezones, and clock configuration, see Configuring the date and time and UTC, Timezones, and DST.

2.2. ENTROPY

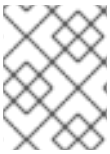
OpenShift Container Platform uses entropy to generate random numbers for objects such as IDs or

SSL traffic. These operations wait until there is enough entropy to complete the task. Without enough entropy, the kernel is not able to generate these random numbers with sufficient speed, which can lead to timeouts and the refusal of secure connections.

To check available entropy:

```
$ cat /proc/sys/kernel/random/entropy_avail
2683
```

The available entropy should be verified on all node hosts in the cluster. Ideally, this value should be above **1000**.



NOTE

Red Hat recommends monitoring this value and issuing an alert if the value is under **800**.

Alternatively, you can use the **rngtest** command to check not only the available entropy, but if your system can *feed* enough entropy as well:

```
$ cat /dev/random | rngtest -c 100
```

The **rngtest** command is available from the **rng-tools**

If the above takes around 30 seconds to complete, then there is not enough entropy available.

Depending on your environment, entropy can be increased in multiple ways. For more information, see the following blog post: https://developers.redhat.com/blog/2017/10/05/entropy-rhel-based-cloud-instances/.

Generally, you can increase entropy by installing the **rng-tools** package and enabling the **rngd** service:

```
# yum install rng-tools
# systemctl enable --now rngd
```

Once the **rngd** service has started, entropy should increase to a sufficient level.

2.3. CHECKING THE DEFAULT STORAGE CLASS

For proper functionality of dynamically provisioned persistent storage, the default storage class needs to be defined. During the installation, this default storage class is defined for common cloud providers, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and more.

To verify that the default storage class is defined:

```
$ oc get storageclass
NAME                 TYPE
ssd                  kubernetes.io/gce-pd
standard (default)   kubernetes.io/gce-pd
```

The above output is taken from an OpenShift Container Platform instance running on GCP, where two kinds of persistent storage are available: standard (HDD) and SSD. Notice the standard storage class is

configured as the default. If there is no storage class defined, or none is set as a default, see the Dynamic Provisioning and Creating Storage Classes section for instructions on how to set up a storage class as suggested.

CHAPTER 3. ENVIRONMENT HEALTH CHECKS

This topic contains steps to verify the overall health of the OpenShift Container Platform cluster and the various components, as well as describing the intended behavior.

Knowing the verification process for the various components is the first step to troubleshooting issues. If experiencing issues, you can use the checks provided in this section to diagnose any problems.

3.1. CHECKING COMPLETE ENVIRONMENT HEALTH

To verify the end-to-end functionality of an OpenShift Container Platform cluster, build and deploy an example application.

Procedure

1. Create a new project named **`validate`**, as well as an example application from the **`cakephp-mysql-example`** template:

   ```
   $ oc new-project validate
   $ oc new-app cakephp-mysql-example
   ```

   You can check the logs to follow the build:

   ```
   $ oc logs -f bc/cakephp-mysql-example
   ```

2. Once the build is complete, two pods should be running: a database and an application:

   ```
   $ oc get pods
   NAME                              READY     STATUS      RESTARTS
   AGE
   cakephp-mysql-example-1-build     0/1       Completed   0          1m
   cakephp-mysql-example-2-247xm     1/1       Running     0
   39s
   mysql-1-hbk46                     1/1       Running     0
   1m
   ```

3. Visit the application URL. The Cake PHP framework welcome page should be visible. The URL should have the following format **`cakephp-mysql-example-validate.<app_domain>`**.

4. Once the functionality has been verified, the **`validate`** project can be deleted:

   ```
   $ oc delete project validate
   ```

   All resources within the project will be deleted as well.

3.2. HOST HEALTH

To verify that the cluster is up and running, connect to a master instance, and run the following:

```
$ oc get nodes
NAME                  STATUS                     AGE       VERSION
ocp-infra-node-1clj   Ready                      1h
```

```
v1.6.1+5115d708d7
ocp-infra-node-86qr   Ready                      1h
v1.6.1+5115d708d7
ocp-infra-node-g8qw   Ready                      1h
v1.6.1+5115d708d7
ocp-master-94zd       Ready,SchedulingDisabled   1h
v1.6.1+5115d708d7
ocp-master-gjkm       Ready,SchedulingDisabled   1h
v1.6.1+5115d708d7
ocp-master-wc8w       Ready,SchedulingDisabled   1h
v1.6.1+5115d708d7
ocp-node-c5dg         Ready                      1h
v1.6.1+5115d708d7
ocp-node-ghxn         Ready                      1h
v1.6.1+5115d708d7
ocp-node-w135         Ready                      1h
v1.6.1+5115d708d7
```

The above cluster example consists of three master hosts, three infrastructure node hosts, and three node hosts. All of them are running. All hosts in the cluster should be visible in this output.

The **Ready** status means that master hosts can communicate with node hosts and that the nodes are ready to run pods (excluding the nodes in which scheduling is disabled).

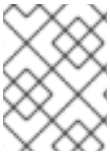
A basic etcd health status can be checked with the **etcdctl2** command from any master instance:

```
# etcdctl2 cluster-health
member 59df5107484b84df is healthy: got healthy result from
https://10.156.0.5:2379
member 6df7221a03f65299 is healthy: got healthy result from
https://10.156.0.6:2379
member fea6dfedf3eecfa3 is healthy: got healthy result from
https://10.156.0.9:2379
cluster is healthy
```

However, to get more information about etcd hosts, including the associated master host:

```
# etcdctl2 member list
295750b7103123e0: name=ocp-master-zh8d peerURLs=https://10.156.0.7:2380
clientURLs=https://10.156.0.7:2379 isLeader=true
b097a72f2610aea5: name=ocp-master-qcg3 peerURLs=https://10.156.0.11:2380
clientURLs=https://10.156.0.11:2379 isLeader=false
fea6dfedf3eecfa3: name=ocp-master-j338 peerURLs=https://10.156.0.9:2380
clientURLs=https://10.156.0.9:2379 isLeader=false
```

All etcd hosts should contain the master host name if the etcd cluster is co-located with master services, or all etcd instances should be visible if etcd is running separately.



NOTE

etcdctl2 is an alias for the **etcdctl** tool that contains the proper flags to query the etcd cluster in v2 data model, as well as, **etcdctl3** for v3 data model.

3.3. ROUTER AND REGISTRY HEALTH

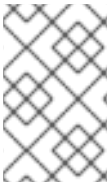
To check if a router service is running:

```
$ oc -n default get deploymentconfigs/router
NAME      REVISION   DESIRED   CURRENT   TRIGGERED BY
router    1          3         3         config
```

The values in the **DESIRED** and **CURRENT** columns should match the number of nodes hosts.

Use the same command to check the registry status:

```
$ oc -n default get deploymentconfigs/docker-registry
NAME              REVISION   DESIRED   CURRENT   TRIGGERED BY
docker-registry   1          3         3         config
```



NOTE

Multiple running instances of the container registry require backend storage supporting writes by multiple processes. If the chosen infrastructure provider does not contain this ability, running a single instance of a container registry is acceptable.

To verify that all pods are running and on which hosts:

```
$ oc -n default get pods -o wide
NAME                       READY     STATUS    RESTARTS   AGE       IP
NODE
docker-registry-1-54nhl    1/1       Running   0          2d
172.16.2.3    ocp-infra-node-tl47
docker-registry-1-jsm2t    1/1       Running   0          2d
172.16.8.2    ocp-infra-node-62rc
docker-registry-1-qbt4g    1/1       Running   0          2d
172.16.14.3   ocp-infra-node-xrtz
registry-console-2-gbhcz   1/1       Running   0          2d
172.16.8.4    ocp-infra-node-62rc
router-1-6zhf8             1/1       Running   0          2d
10.156.0.4    ocp-infra-node-62rc
router-1-ffq4g             1/1       Running   0          2d
10.156.0.10   ocp-infra-node-tl47
router-1-zqxbl             1/1       Running   0          2d
10.156.0.8    ocp-infra-node-xrtz
```

NOTE

If OpenShift Container Platform is using an external container registry, the internal registry service does not need to be running.

3.4. NETWORK CONNECTIVITY

Network connectivity has two main networking layers: the cluster network for node interaction, and the software defined network (SDN) for pod interaction. OpenShift Container Platform supports multiple network configurations, often optimized for a specific infrastructure provider.

NOTE

Due to the complexity of networking, not all verification scenarios are covered in this section.

3.4.1. Connectivity on master hosts

etcd and master hosts

Master services keep their state synchronized using the etcd key-value store. Communication between master and etcd services is important, whether those etcd services are collocated on master hosts, or running on hosts designated only for the etcd service. This communication happens on TCP ports **2379** and **2380**. See the Host health section for methods to check this communication.

SkyDNS

SkyDNS provides name resolution of local services running in OpenShift Container Platform. This service uses **TCP** and **UDP** port **8053**.

To verify the name resolution:

```
$ dig +short docker-registry.default.svc.cluster.local
172.30.150.7
```

If the answer matches the output of the following, **SkyDNS** service is working correctly:

```
$ oc get svc/docker-registry
NAME              CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
docker-registry   172.30.150.7   <none>        5000/TCP   3d
```

API service and web console

Both the API service and web console share the same port, usually **TCP 8443** or **443**, depending on the setup. This port needs to be available within the cluster and to everyone who needs to work with the deployed environment. The URLs under which this port is reachable may differ for internal cluster and for external clients.

In the following example, the **https://internal-master.example.com:443** URL is used by the internal cluster, and the **https://master.example.com:443** URL is used by external clients. On any node host:

```
$ curl https://internal-master.example.com:443/version
{
  "major": "1",
  "minor": "6",
  "gitVersion": "v1.6.1+5115d708d7",
  "gitCommit": "fff65cf",
  "gitTreeState": "clean",
  "buildDate": "2017-10-11T22:44:25Z",
  "goVersion": "go1.7.6",
  "compiler": "gc",
  "platform": "linux/amd64"
}
```

This must be reachable from client's network:

```
$ curl -k https://master.example.com:443/healthz
ok
```

3.4.2. Connectivity on node instances

The SDN connecting pod communication on nodes uses **UDP** port **4789** by default.

To verify node host functionality, create a new application. The following example ensures the node reaches the docker registry, which is running on an infrastructure node:

Procedure

1. Create a new project:

```
$ oc new-project sdn-test
```

2. Deploy an httpd application:

```
$ oc new-app centos/httpd-24-
centos7~https://github.com/openshift/httpd-ex
```

Wait until the build is complete:

```
$ oc get pods
NAME               READY     STATUS      RESTARTS   AGE
httpd-ex-1-205hz   1/1       Running     0          34s
httpd-ex-1-build   0/1       Completed   0          1m
```

3. Connect to the running pod:

```
$ oc rsh po/httpd-ex-1-205hz
```

4. Check the **healthz** path of the internal registry service:

```
$ curl -kv https://docker-
registry.default.svc.cluster.local:5000/healthz
* About to connect() to docker-registry.default.svc.cluster.locl
port 5000 (#0)
*   Trying 172.30.150.7...
* Connected to docker-registry.default.svc.cluster.local
(172.30.150.7) port 5000 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
* skipping SSL peer certificate verification
* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*  subject: CN=172.30.150.7
*  start date: Nov 30 17:21:51 2017 GMT
*  expire date: Nov 30 17:21:52 2019 GMT
*  common name: 172.30.150.7
*  issuer: CN=openshift-signer@1512059618
> GET /healthz HTTP/1.1
> User-Agent: curl/7.29.0
> Host: docker-registry.default.svc.cluster.local:5000
> Accept: */*
```

```
>
< HTTP/1.1 200 OK
< Cache-Control: no-cache
< Date: Mon, 04 Dec 2017 16:26:49 GMT
< Content-Length: 0
< Content-Type: text/plain; charset=utf-8
<
* Connection #0 to host docker-registry.default.svc.cluster.local
left intact

sh-4.2$ *exit*
```

The **HTTP/1.1 200 OK** response means the node is correctly connecting.

5. Clean up the test project:

```
$ oc delete project sdn-test
project "sdn-test" deleted
```

6. The node host is listening on **TCP** port **10250**. This port needs to be reachable by all master hosts on any node, and if monitoring is deployed in the cluster, the infrastructure nodes must have access to this port on all instances as well. Broken communication on this port can be detected with the following command:

```
$ oc get nodes
NAME                 STATUS                     AGE       VERSION
ocp-infra-node-1clj  Ready                      4d
v1.6.1+5115d708d7
ocp-infra-node-86qr  Ready                      4d
v1.6.1+5115d708d7
ocp-infra-node-g8qw  Ready                      4d
v1.6.1+5115d708d7
ocp-master-94zd      Ready,SchedulingDisabled   4d
v1.6.1+5115d708d7
ocp-master-gjkm      Ready,SchedulingDisabled   4d
v1.6.1+5115d708d7
ocp-master-wc8w      Ready,SchedulingDisabled   4d
v1.6.1+5115d708d7
ocp-node-c5dg        Ready                      4d
v1.6.1+5115d708d7
ocp-node-ghxn        Ready                      4d
v1.6.1+5115d708d7
ocp-node-w135        NotReady                   4d
v1.6.1+5115d708d7
```

In the output above, the node service on the **ocp-node-w135** node is not reachable by the master services, which is represented by its **NotReady** status.

7. The last service is the router, which is responsible for routing connections to the correct services running in the OpenShift Container Platform cluster. Routers listen on **TCP** ports **80** and **443** on infrastructure nodes for ingress traffic. Before routers can start working, DNS must be configured:

```
$ dig *.apps.example.com
```

```
; <<>> DiG 9.11.1-P3-RedHat-9.11.1-8.P3.fc27 <<>> *.apps.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45790
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;*.apps.example.com. IN A

;; ANSWER SECTION:
*.apps.example.com. 3571 IN CNAME apps.example.com.
apps.example.com. 3561 IN A 35.xx.xx.92

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Dec 05 16:03:52 CET 2017
;; MSG SIZE  rcvd: 105
```

The IP address, in this case **35.xx.xx.92**, should be pointing to the load balancer distributing ingress traffic to all infrastructure nodes. To verify the functionality of the routers, check the registry service once more, but this time from outside the cluster:

```
$ curl -kv https://docker-registry-default.apps.example.com/healthz
*   Trying 35.xx.xx.92...
* TCP_NODELAY set
* Connected to docker-registry-default.apps.example.com
(35.xx.xx.92) port 443 (#0)
...
< HTTP/2 200
< cache-control: no-cache
< content-type: text/plain; charset=utf-8
< content-length: 0
< date: Tue, 05 Dec 2017 15:13:27 GMT
<
* Connection #0 to host docker-registry-default.apps.example.com
left intact
```

3.5. STORAGE

Master instances need at least 40 GB of hard disk space for the **/var** directory. Check the disk usage of a master host using the **df** command:

```
$ df -hT
Filesystem     Type      Size  Used Avail Use% Mounted on
/dev/sda1      xfs        45G  2.8G   43G   7% /
devtmpfs       devtmpfs  3.6G     0  3.6G   0% /dev
tmpfs          tmpfs     3.6G     0  3.6G   0% /dev/shm
tmpfs          tmpfs     3.6G   63M  3.6G   2% /run
tmpfs          tmpfs     3.6G     0  3.6G   0% /sys/fs/cgroup
tmpfs          tmpfs     732M     0  732M   0% /run/user/1000
tmpfs          tmpfs     732M     0  732M   0% /run/user/0
```

Node instances need at least 15 GB space for the **/var** directory, and at least another 15 GB for Docker storage (**/var/lib/docker** in this case). Depending on the size of the cluster and the amount of ephemeral storage desired for pods, a separate partition should be created for **/var/lib/origin/openshift.local.volumes** on the nodes.

```
$ df -hT
Filesystem     Type      Size  Used Avail Use% Mounted on
/dev/sda1      xfs        25G  2.4G   23G  10% /
devtmpfs       devtmpfs  3.6G     0  3.6G   0% /dev
tmpfs          tmpfs     3.6G     0  3.6G   0% /dev/shm
tmpfs          tmpfs     3.6G  147M  3.5G   4% /run
tmpfs          tmpfs     3.6G     0  3.6G   0% /sys/fs/cgroup
/dev/sdb       xfs        25G  2.7G   23G  11% /var/lib/docker
/dev/sdc       xfs        50G   33M   50G   1%
/var/lib/origin/openshift.local.volumes
tmpfs          tmpfs     732M     0  732M   0% /run/user/1000
```

Persistent storage for pods should be handled outside of the instances running the OpenShift Container Platform cluster. Persistent volumes for pods can be provisioned by the infrastructure provider, or with the use of container native storage or container ready storage.

3.6. DOCKER STORAGE

Docker Storage can be backed by one of two options. The first is a thin pool logical volume with device mapper, the second, since Red Hat Enterprise Linux version 7.4, is an overlay2 file system. The overlay2 file system is generally recommended due to the ease of setup and increased performance.

The Docker storage disk is mounted as **/var/lib/docker** and formatted with **xfs** file system. Docker storage is configured to use overlay2 filesystem:

```
$ cat /etc/sysconfig/docker-storage
DOCKER_STORAGE_OPTIONS='--storage-driver overlay2'
```

To verify this storage driver is used by Docker:

```
# docker info
Containers: 4
 Running: 4
 Paused: 0
 Stopped: 0
Images: 4
Server Version: 1.12.6
Storage Driver: overlay2
 Backing Filesystem: xfs
Logging Driver: journald
Cgroup Driver: systemd
Plugins:
 Volume: local
 Network: overlay host bridge null
 Authorization: rhel-push-plugin
Swarm: inactive
Runtimes: docker-runc runc
Default Runtime: docker-runc
Security Options: seccomp selinux
```

```
Kernel Version: 3.10.0-693.11.1.el7.x86_64
Operating System: Employee SKU
OSType: linux
Architecture: x86_64
Number of Docker Hooks: 3
CPUs: 2
Total Memory: 7.147 GiB
Name: ocp-infra-node-1clj
ID: T7T6:IQTG:WTUX:7BRU:5FI4:XUL5:PAAM:4SLW:NWKL:WU2V:NQOW:JPHC
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://registry.access.redhat.com/v1/
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
Insecure Registries:
 127.0.0.0/8
Registries: registry.access.redhat.com (secure),
registry.access.redhat.com (secure), docker.io (secure)
```

3.7. API SERVICE STATUS

The OpenShift API service, **`atomic-openshift-master-api.service`**, runs on all master instances. To see the status of the service:

```
$ systemctl status atomic-openshift-master-api.service
● atomic-openshift-master-api.service - Atomic OpenShift Master API
   Loaded: loaded (/usr/lib/systemd/system/atomic-openshift-master-
api.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2017-11-30 11:40:19 EST; 5 days ago
     Docs: https://github.com/openshift/origin
 Main PID: 30047 (openshift)
   Memory: 65.0M
   CGroup: /system.slice/atomic-openshift-master-api.service
           └─30047 /usr/bin/openshift start master api --
config=/etc/origin/master/ma...

Dec 06 09:15:49 ocp-master-94zd atomic-openshift-master-api[30047]: I1206
09:15:49.85...
Dec 06 09:15:50 ocp-master-94zd atomic-openshift-master-api[30047]: I1206
09:15:50.96...
Dec 06 09:15:52 ocp-master-94zd atomic-openshift-master-api[30047]: I1206
09:15:52.34...
```

The API service exposes a health check, which can be queried externally with:

```
$ curl -k https://master.example.com/healthz
ok
```

3.8. CONTROLLER ROLE VERIFICATION

The OpenShift Container Platform controller service, **`atomic-openshift-master-controllers.service`**, is available across all master hosts. The service runs in active/passive mode, meaning it should only be running on one master at any time.

The OpenShift Container Platform controllers execute a procedure to choose which host runs the service. The current running value is stored in an annotation in a special **configmap** stored in the **kube-system** project.

Verify the master host running the **atomic-openshift-master-controllers** service as a **cluster-admin** user:

```
$ oc get -n kube-system cm openshift-master-controllers -o yaml
apiVersion: v1
kind: ConfigMap
metadata:
  annotations:
    control-plane.alpha.kubernetes.io/leader: '{"holderIdentity":"master-
ose-master-0.example.com-10.19.115.212-
dnwrtcl4","leaseDurationSeconds":15,"acquireTime":"2018-02-
17T18:16:54Z","renewTime":"2018-02-19T13:50:33Z","leaderTransitions":16}'
  creationTimestamp: 2018-02-02T10:30:04Z
  name: openshift-master-controllers
  namespace: kube-system
  resourceVersion: "17349662"
  selfLink: /api/v1/namespaces/kube-system/configmaps/openshift-master-
controllers
  uid: 08636843-0804-11e8-8580-fa163eb934f0
```

The command outputs the current master controller in the **control-plane.alpha.kubernetes.io/leader** annotation, within the **holderIdentity** property as:

```
master-<hostname>-<ip>-<8_random_characters>
```

Find the hostname of the master host by filtering the output using the following:

```
$ oc get -n kube-system cm openshift-master-controllers -o json | jq -r
'.metadata.annotations[] | fromjson.holderIdentity | match("^master-(.*)-
[0-9.]*-[0-9a-z]{8}$") | .captures[0].string'
ose-master-0.example.com
```

3.9. VERIFYING CORRECT MAXIMUM TRANSMISSION UNIT (MTU) SIZE

Verifying the maximum transmission unit (MTU) prevents a possible networking misconfiguration that can masquerade as an SSL certificate issue.

When a packet is larger than the MTU size that is transmitted over HTTP, the physical network router is able to break the packet into multiple packets to transmit the data. However, when a packet is larger than the MTU size is that transmitted over HTTPS, the router is forced to drop the packet.

Installation produces certificates that provide secure connections to multiple components that include:

- master hosts
- node hosts
- infrastructure nodes

- registry
- router

These certificates can be found within the **/etc/origin/master** directory for the master nodes and **/etc/origin/node** directory for the infra and app nodes.

After installation, you can verify connectivity to the **REGISTRY_OPENSHIFT_SERVER_ADDR** using the process outlined in the Network connectivity section.

Prerequisites

1. From a master host, get the HTTPS address:

```
$ oc get dc docker-registry -o
jsonpath='{.spec.template.spec.containers[].env[?
(@.name=="OPENSHIFT_DEFAULT_REGISTRY")].value}{"\n"}'
docker-registry.default.svc:5000
```

   The above gives the output of **docker-registry.default.svc:5000**.

2. Append **/healthz** to the value given above, use it to check on all hosts (master, infrastructure, node):

```
$ curl -v https://docker-registry.default.svc:5000/healthz
* About to connect() to docker-registry.default.svc port 5000 (#0)
*   Trying 172.30.11.171...
* Connected to docker-registry.default.svc (172.30.11.171) port 5000
(#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
*   CAfile: /etc/pki/tls/certs/ca-bundle.crt
  CApath: none
* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*  subject: CN=172.30.11.171
*  start date: Oct 18 05:30:10 2017 GMT
*  expire date: Oct 18 05:30:11 2019 GMT
*  common name: 172.30.11.171
*  issuer: CN=openshift-signer@1508303629
> GET /healthz HTTP/1.1
> User-Agent: curl/7.29.0
> Host: docker-registry.default.svc:5000
> Accept: */*
>
< HTTP/1.1 200 OK
< Cache-Control: no-cache
< Date: Tue, 24 Oct 2017 19:42:35 GMT
< Content-Length: 0
< Content-Type: text/plain; charset=utf-8
<
* Connection #0 to host docker-registry.default.svc left intact
```

   The above example output shows the MTU size being used to ensure the SSL connection is correct. The attempt to connect is successful, followed by connectivity being established and completes with initializing the NSS with the certpath and all the server certificate information regarding the *docker-registry*.

An improper MTU size results in a timeout:

```
$ curl -v https://docker-registry.default.svc:5000/healthz
* About to connect() to docker-registry.default.svc port 5000 (#0)
*   Trying 172.30.11.171...
* Connected to docker-registry.default.svc (172.30.11.171) port 5000
(#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
```

The above example shows that the connection is established, but it cannot finish initializing NSS with certpath. The issue deals with improper MTU size set within the **`/etc/origin/node/node-config.yaml`** file.

To fix this issue, adjust the MTU size within the **`/etc/origin/node/node-config.yaml`** to 50 bytes smaller than the MTU size being used by the OpenShift SDN Ethernet device.

3. View the MTU size of the desired Ethernet device (i.e. **`eth0`**):

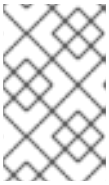
```
$ ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP mode DEFAULT qlen 1000
    link/ether fa:16:3e:92:6a:86 brd ff:ff:ff:ff:ff:ff
```

The above shows MTU set to 1500.

4. To change the MTU size, modify the **`/etc/origin/node/node-config.yaml`** file and set a value that is 50 bytes smaller than output provided by the **`ip`** command.
For example, if the MTU size is set to 1500, adjust the MTU size to 1450 within the **`/etc/origin/node/node-config.yaml`** file:

```
networkConfig:
   mtu: 1450
```

5. Save the changes and reboot the node:



NOTE

You must change the MTU size on all masters and nodes that are part of the OpenShift Container Platform SDN. Also, the MTU size of the tun0 interface must be the same across all nodes that are part of the cluster.

6. Once the node is back online, confirm the issue no longer exists by re-running the original **`curl`** command.

```
$ curl -v https://docker-registry.default.svc:5000/healthz
```

If the timeout persists, continue to adjust the MTU size in increments of 50 bytes and repeat the process.

CHAPTER 4. CREATING AN ENVIRONMENT-WIDE BACKUP

Creating an environment-wide backup involves copying important data to assist with restoration in the case of crashing instances, or corrupt data. After backups have been created, they can be restored onto a newly installed version of the relevant component.

Perform a back up on a regular basis to prevent data loss.

4.1. CREATING A MASTER HOST BACKUP

The backup process is to be performed before any change to the infrastructure, such as a system update, upgrade, or any other significant modification. Backups should be performed on a regular basis to ensure the most recent data is available if a failure occurs.

OpenShift Container Platform files

The master instances run important services, such as the API, controllers. The **/etc/origin/master** directory stores many important files:

- The configuration, the API, controllers, services, and more
- Certificates generated by the installation
- All cloud provider-related configuration
- Keys and other authentication files, such as **htpasswd** if you use htpasswd
- And more

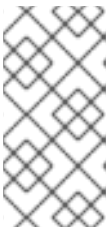
The OpenShift Container Platform services can be customized to increase the log level, use proxies, and so on. The configuration files are stored in the **/etc/sysconfig** directory.

Because the masters are also unschedulable nodes, back up the entire **/etc/origin** directory.

Procedure

1. Create a backup of the master host configuration files:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo cp -aR /etc/origin ${MYBACKUPDIR}/etc
$ sudo cp -aR /etc/sysconfig/atomic-* ${MYBACKUPDIR}/etc/sysconfig/
```



NOTE

On a single master cluster installation the configuration file is stored in the **/etc/sysconfig/atomic-openshift-master**, whereas in a multi-master environment **/etc/sysconfig/atomic-openshift-master-api**, and **/etc/sysconfig/atomic-openshift-master-controllers** are used.

WARNING

At the time of writing, the `/etc/origin/master/ca.serial.txt` file is generated onto only the first master listed in the Ansible host inventory. This is being investigated to be fixed for future OpenShift Container Platform releases in the 1469358 bugzilla. If deprecating the first master host, copy the `/etc/origin/master/ca.serial.txt` file to the rest of master hosts before the process.

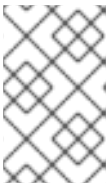
2. Other important files that need to be considered when planning a backup include:

File	Description
`/etc/cni/*`	Container Network Interface configuration (if used)
`/etc/sysconfig/iptables`	Where the `iptables` rules are stored
`/etc/sysconfig/docker-storage-setup`	The input file for `container-storage-setup` command
`/etc/sysconfig/docker`	The `docker` configuration file
`/etc/sysconfig/docker-network`	`docker` networking configuration (i.e. MTU)
`/etc/sysconfig/docker-storage`	`docker` storage configuration (generated by `container-storage-setup`)
`/etc/dnsmasq.conf`	Main configuration file for `dnsmasq`
`/etc/dnsmasq.d/*`	Different `dnsmasq` configuration files
`/etc/sysconfig/flanneld`	`flannel` configuration file (if used)
`/etc/pki/ca-trust/source/anchors/`	Certificates added to the system (i.e. for external registries)

Create a backup of those files:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo mkdir -p ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors
$ sudo cp -aR /etc/sysconfig/{iptables,docker-*,flanneld} \
    ${MYBACKUPDIR}/etc/sysconfig/
$ sudo cp -aR /etc/dnsmasq* /etc/cni ${MYBACKUPDIR}/etc/
$ sudo cp -aR /etc/pki/ca-trust/source/anchors/* \
    ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/
```

3. If a package is accidentally removed, or a file included in an **rpm** package should be restored, having a list of **rhel** packages installed on the system can be useful.



NOTE

If using Red Hat Satellite features, such as content views or the facts store, provide a proper mechanism to reinstall the missing packages and a historical data of packages installed in the systems.

To create a list of the current **rhel** packages installed in the system:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}
$ rpm -qa | sort | sudo tee $MYBACKUPDIR/packages.txt
```

4. If using the previous steps, the following files should now be present in the backup directory:

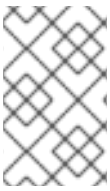
```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo find ${MYBACKUPDIR} -mindepth 1 -type f -printf '%P\n'
etc/sysconfig/atomic-openshift-master
etc/sysconfig/atomic-openshift-master-api
etc/sysconfig/atomic-openshift-master-controllers
etc/sysconfig/atomic-openshift-node
etc/sysconfig/flanneld
etc/sysconfig/iptables
etc/sysconfig/docker-network
etc/sysconfig/docker-storage
etc/sysconfig/docker-storage-setup
etc/sysconfig/docker-storage-setup.rpmnew
etc/origin/master/ca.crt
etc/origin/master/ca.key
etc/origin/master/ca.serial.txt
etc/origin/master/ca-bundle.crt
etc/origin/master/master.proxy-client.crt
etc/origin/master/master.proxy-client.key
etc/origin/master/service-signer.crt
etc/origin/master/service-signer.key
etc/origin/master/serviceaccounts.private.key
etc/origin/master/serviceaccounts.public.key
etc/origin/master/openshift-master.crt
etc/origin/master/openshift-master.key
etc/origin/master/openshift-master.kubeconfig
etc/origin/master/master.server.crt
etc/origin/master/master.server.key
etc/origin/master/master.kubelet-client.crt
etc/origin/master/master.kubelet-client.key
etc/origin/master/admin.crt
etc/origin/master/admin.key
etc/origin/master/admin.kubeconfig
etc/origin/master/etcd.server.crt
etc/origin/master/etcd.server.key
etc/origin/master/master.etcd-client.key
etc/origin/master/master.etcd-client.csr
```

```
etc/origin/master/master.etcd-client.crt
etc/origin/master/master.etcd-ca.crt
etc/origin/master/policy.json
etc/origin/master/scheduler.json
etc/origin/master/htpasswd
etc/origin/master/session-secrets.yaml
etc/origin/master/openshift-router.crt
etc/origin/master/openshift-router.key
etc/origin/master/registry.crt
etc/origin/master/registry.key
etc/origin/master/master-config.yaml
etc/origin/generated-configs/master-master-
1.example.com/master.server.crt
...[OUTPUT OMITTED]...
etc/origin/cloudprovider/openstack.conf
etc/origin/node/system:node:master-0.example.com.crt
etc/origin/node/system:node:master-0.example.com.key
etc/origin/node/ca.crt
etc/origin/node/system:node:master-0.example.com.kubeconfig
etc/origin/node/server.crt
etc/origin/node/server.key
etc/origin/node/node-dnsmasq.conf
etc/origin/node/resolv.conf
etc/origin/node/node-config.yaml
etc/origin/node/flannel.etcd-client.key
etc/origin/node/flannel.etcd-client.csr
etc/origin/node/flannel.etcd-client.crt
etc/origin/node/flannel.etcd-ca.crt
etc/pki/ca-trust/source/anchors/openshift-ca.crt
etc/pki/ca-trust/source/anchors/registry-ca.crt
etc/dnsmasq.conf
etc/dnsmasq.d/origin-dns.conf
etc/dnsmasq.d/origin-upstream-dns.conf
etc/dnsmasq.d/node-dnsmasq.conf
packages.txt
```

If needed, the files can be compressed to save space:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo tar -zcvf /backup/$(hostname)-$(date +%Y%m%d).tar.gz
$MYBACKUPDIR
$ sudo rm -Rf ${MYBACKUPDIR}
```

To create any of these files from scratch, the **openshift-ansible-contrib** repository contains the **backup_master_node.sh** script, which performs the previous steps. The script creates a directory on the host running the script and copies all the files previously mentioned.



NOTE

The **openshift-ansible-contrib** script is not supported by Red Hat, but the reference architecture team performs testing to ensure the code operates as defined and is secure.

The script can be executed on every master host with:

-

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-contrib.git
$ cd openshift-ansible-contrib/reference-architecture/day2ops/scripts
$ ./backup_master_node.sh -h
```

4.2. CREATING A NODE HOST BACKUP

Creating a backup of a node host is a different use case from backing up a master host. Because master hosts contain many important files, creating a backup is highly recommended. However, the nature of nodes is that anything special is replicated over the nodes in case of failover, and they typically do not contain data that is necessary to run an environment. If a backup of a node contains something necessary to run an environment, then a creating a backup is recommended.

The backup process is to be performed before any change to the infrastructure, such as a system update, upgrade, or any other significant modification. Backups should be performed on a regular basis to ensure the most recent data is available if a failure occurs.

OpenShift Container Platform files

Node instances run applications in the form of pods, which are based on containers. The **`/etc/origin/`** and **`/etc/origin/node`** directories house important files, such as:

- The configuration of the node services
- Certificates generated by the installation
- Cloud provider-related configuration
- Keys and other authentication files, such as the **`dnsmasq`** configuration

The OpenShift Container Platform services can be customized to increase the log level, use proxies, and more, and the configuration files are stored in the **`/etc/sysconfig`** directory.

Procedure

1. Create a backup of the node configuration files:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo cp -aR /etc/origin ${MYBACKUPDIR}/etc
$ sudo cp -aR /etc/sysconfig/atomic-openshift-node
${MYBACKUPDIR}/etc/sysconfig/
```

2. OpenShift Container Platform uses specific files that must be taken into account when planning the backup policy, including:

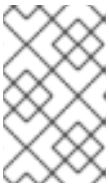
File	Description
`/etc/cni/*`	Container Network Interface configuration (if used)
`/etc/sysconfig/iptables`	Where the **`iptables`** rules are stored

/etc/sysconfig/docker-storage-setup	The input file for **container-storage-setup** command
/etc/sysconfig/docker	The **docker** configuration file
/etc/sysconfig/docker-network	**docker** networking configuration (i.e. MTU)
/etc/sysconfig/docker-storage	**docker** storage configuration (generated by **container-storage-setup**)
/etc/dnsmasq.conf	Main configuration file for **dnsmasq**
/etc/dnsmasq.d/*	Different **dnsmasq** configuration files
/etc/sysconfig/flanneld	**flannel** configuration file (if used)
/etc/pki/ca-trust/source/anchors/	Certificates added to the system (i.e. for external registries)

To create those files:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo mkdir -p ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors
$ sudo cp -aR /etc/sysconfig/{iptables,docker-*,flanneld} \
    ${MYBACKUPDIR}/etc/sysconfig/
$ sudo cp -aR /etc/dnsmasq* /etc/cni ${MYBACKUPDIR}/etc/
$ sudo cp -aR /etc/pki/ca-trust/source/anchors/* \
    ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/
```

3. If a package is accidentally removed, or a file included in an **rpm** package should be restored, having a list of **rhel** packages installed on the system can be useful.



NOTE

If using Red Hat Satellite features, such as content views or the facts store, provide a proper mechanism to reinstall the missing packages and a historical data of packages installed in the systems.

To create a list of the current **rhel** packages installed in the system:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}
$ rpm -qa | sort | sudo tee $MYBACKUPDIR/packages.txt
```

4. The following files should now be present in the backup directory:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo find ${MYBACKUPDIR} -mindepth 1 -type f -printf '%P\n'
etc/sysconfig/atomic-openshift-node
```

```
etc/sysconfig/flanneld
etc/sysconfig/iptables
etc/sysconfig/docker-network
etc/sysconfig/docker-storage
etc/sysconfig/docker-storage-setup
etc/sysconfig/docker-storage-setup.rpmnew
etc/origin/node/system:node:app-node-0.example.com.crt
etc/origin/node/system:node:app-node-0.example.com.key
etc/origin/node/ca.crt
etc/origin/node/system:node:app-node-0.example.com.kubeconfig
etc/origin/node/server.crt
etc/origin/node/server.key
etc/origin/node/node-dnsmasq.conf
etc/origin/node/resolv.conf
etc/origin/node/node-config.yaml
etc/origin/node/flannel.etcd-client.key
etc/origin/node/flannel.etcd-client.csr
etc/origin/node/flannel.etcd-client.crt
etc/origin/node/flannel.etcd-ca.crt
etc/origin/cloudprovider/openstack.conf
etc/pki/ca-trust/source/anchors/openshift-ca.crt
etc/pki/ca-trust/source/anchors/registry-ca.crt
etc/dnsmasq.conf
etc/dnsmasq.d/origin-dns.conf
etc/dnsmasq.d/origin-upstream-dns.conf
etc/dnsmasq.d/node-dnsmasq.conf
packages.txt
```

If needed, the files can be compressed to save space:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo tar -zcvf /backup/$(hostname)-$(date +%Y%m%d).tar.gz
$MYBACKUPDIR
$ sudo rm -Rf ${MYBACKUPDIR}
```

To create any of these files from scratch, the **openshift-ansible-contrib** repository contains the **backup_master_node.sh** script, which performs the previous steps. The script creates a directory on the host running the script and copies all the files previously mentioned.

NOTE

The **openshift-ansible-contrib** script is not supported by Red Hat, but the reference architecture team performs testing to ensure the code operates as defined and is secure.

The script can be executed on every master host with:

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-contrib.git
$ cd openshift-ansible-contrib/reference-architecture/day2ops/scripts
$ ./backup_master_node.sh -h
```

4.3. ETCD BACKUP

etcd is the key value store for all object definitions, as well as the persistent master state. Other components watch for changes, then bring themselves into the desired state.

OpenShift Container Platform versions prior to 3.5 use etcd version 2 (v2), while 3.5 and later use version 3 (v3). The data model between the two versions of etcd is different. etcd v3 can use both the v2 and v3 data models, whereas etcd v2 can only use the v2 data model. In an etcd v3 server, the v2 and v3 data stores exist in parallel and are independent.

For both v2 and v3 operations, you can use the **ETCDCTL_API** environment variable to use the proper API:

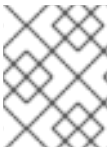
```
$ etcdctl -v
etcdctl version: 3.2.5
API version: 2
$ ETCDCTL_API=3 etcdctl version
etcdctl version: 3.2.5
API version: 3.2
```

See Migrating etcd Data (v2 to v3) section for information about how to migrate to v3.

The etcd backup process is composed of two different procedures:

- Configuration backup: Including the required etcd configuration and certificates
- Data backup: Including both v2 and v3 data model.

You can perform the data backup process on any host that has connectivity to the etcd cluster, where the proper certificates are provided, and where the **etcdctl** tool is installed.



NOTE

The backup files must be copied to an external system, ideally outside the OpenShift Container Platform environment, and then encrypted.

4.3.1. etcd configuration backup

The etcd configuration files to be preserved are all stored in the **/etc/etcd** directory of the instances where etcd is running. This includes the etcd configuration file (**/etc/etcd/etcd.conf**) and the required certificates for cluster communication. All those files are generated at installation time by the Ansible installer.

For each etcd member of the cluster, back up the etcd configuration.

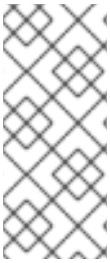
```
$ ssh master-0
# mkdir -p /backup/etcd-config-$(date +%Y%m%d)/
# cp -R /etc/etcd/ /backup/etcd-config-$(date +%Y%m%d)/
```

NOTE

The certificates and configuration files on each etcd cluster member are unique.

4.3.2. etcd data backup

Prerequisites



NOTE

The OpenShift Container Platform installer creates aliases to avoid typing all the flags named **etcdctl2** for etcd v2 tasks and **etcdctl3** for etcd v3 tasks.

However, the **etcdctl3** alias does not provide the full endpoint list to the **etcdctl** command, so the **--endpoints** option with all the endpoints must be provided.

Before backing up etcd:

- **etcdctl** binaries should be available or, in containerized installations, the **rhel7/etcd** container should be available
- Ensure connectivity with the etcd cluster (port 2379/tcp)
- Ensure the proper certificates to connect to the etcd cluster

1. To ensure the etcd cluster is working, check its health.
   - If you use the etcd v2 API, run the following command:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
          --key-file=/etc/etcd/peer.key \
          --ca-file=/etc/etcd/ca.crt \
          --peers="https://*master-0.example.com*:2379,\
          https://*master-1.example.com*:2379,\
          https://*master-2.example.com*:2379"\
          cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

   - If you use the etcd v3 API, run the following command:

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
          --key=/etc/etcd/peer.key \
          --cacert="/etc/etcd/ca.crt" \
          --endpoints="https://*master-0.example.com*:2379,\
            https://*master-1.example.com*:2379,\
            https://*master-2.example.com*:2379"
            endpoint health
https://master-0.example.com:2379 is healthy: successfully
committed proposal: took = 5.011358ms
https://master-1.example.com:2379 is healthy: successfully
committed proposal: took = 1.305173ms
https://master-2.example.com:2379 is healthy: successfully
committed proposal: took = 1.388772ms
```

2. Check the member list.

- If you use the etcd v2 API, run the following command:

```
# etcdctl2 member list
2a371dd20f21ca8d: name=master-1.example.com
peerURLs=https://192.168.55.12:2380
clientURLs=https://192.168.55.12:2379 isLeader=false
40bef1f6c79b3163: name=master-0.example.com
peerURLs=https://192.168.55.8:2380
clientURLs=https://192.168.55.8:2379 isLeader=false
95dc17ffcce8ee29: name=master-2.example.com
peerURLs=https://192.168.55.13:2380
clientURLs=https://192.168.55.13:2379 isLeader=true
```

- If you use the etcd v3 API, run the following command:

```
# etcdctl3 member list
2a371dd20f21ca8d, started, master-1.example.com,
https://192.168.55.12:2380, https://192.168.55.12:2379
40bef1f6c79b3163, started, master-0.example.com,
https://192.168.55.8:2380, https://192.168.55.8:2379
95dc17ffcce8ee29, started, master-2.example.com,
https://192.168.55.13:2380, https://192.168.55.13:2379
```

Procedure

NOTE

While the **`etcdctl backup`** command is used to perform the backup, etcd v3 has no concept of a *backup*. Instead, you either take a *snapshot* from a live member with the **`etcdctl snapshot save`** command or copy the **`member/snap/db`** file from an etcd data directory.

The **`etcdctl backup`** command rewrites some of the metadata contained in the backup, specifically, the node ID and cluster ID, which means that in the backup, the node loses its former identity. To recreate a cluster from the backup, you create a new, single-node cluster, then add the rest of the nodes to the cluster. The metadata is rewritten to prevent the new node from joining an existing cluster.

1. Back up the etcd data:

- If you use the v2 API, take the following actions:

a. Stop all etcd services:

```
# systemctl stop etcd.service
```

b. Create the etcd data backup and copy the etcd **db** file:

```
# mkdir -p /backup/etcd-$(date +%Y%m%d)
# etcdctl2 backup \
    --data-dir /var/lib/etcd \
    --backup-dir /backup/etcd-$(date +%Y%m%d)
# cp /var/lib/etcd/member/snap/db /backup/etcd-$(date +%Y%m%d)
```

- If you use the v3 API, run the following command:

```
# mkdir -p /backup/etcd-$(date +%Y%m%d)
# etcdctl3 snapshot save */backup/etcd-$(date +%Y%m%d)*/db
Snapshot saved at /backup/etcd-<date>/db
# systemctl stop etcd.service
# etcdctl2 backup \
    --data-dir /var/lib/etcd \
    --backup-dir /backup/etcd-$(date +%Y%m%d)
# systemctl start etcd.service
```

NOTE

The **`etcdctl snapshot save`** command requires the etcd service to be running.

In these commands, a **`/backup/etcd-<date>/`** directory is created, where **`<date>`** represents the current date, which must be an external NFS share, S3 bucket, or any external storage location.

In the case of an all-in-one cluster, the etcd data directory is located in the **`/var/lib/origin/openshift.local.etcd`** directory.

4.4. CREATING A PROJECT BACKUP

Creating a backup of all relevant data involves exporting all important information, then restoring into a new project.

NOTE

Currently, a OpenShift Container Platform project back up and restore tool is being developed by Red Hat. See the following bug for more information:

- bugzilla 1303205.

4.4.1. Back up a project

Procedure

1. To list all the relevant data to backup:

```
$ oc get all
NAME         TYPE      FROM      LATEST
bc/ruby-ex   Source    Git       1

NAME               TYPE      FROM          STATUS     STARTED
DURATION
builds/ruby-ex-1   Source    Git@c457001   Complete   2 minutes ago
35s

NAME                 DOCKER REPO
TAGS      UPDATED
is/guestbook         10.111.255.221:5000/myproject/guestbook
latest    2 minutes ago
```

```
is/hello-openshift   10.111.255.221:5000/myproject/hello-openshift
latest    2 minutes ago
is/ruby-22-centos7   10.111.255.221:5000/myproject/ruby-22-centos7
latest    2 minutes ago
is/ruby-ex           10.111.255.221:5000/myproject/ruby-ex
latest    2 minutes ago

NAME                 REVISION   DESIRED   CURRENT   TRIGGERED BY
dc/guestbook         1          1         1
config,image(guestbook:latest)
dc/hello-openshift   1          1         1
config,image(hello-openshift:latest)
dc/ruby-ex           1          1         1
config,image(ruby-ex:latest)

NAME                   DESIRED   CURRENT   READY     AGE
rc/guestbook-1         1         1         1         2m
rc/hello-openshift-1   1         1         1         2m
rc/ruby-ex-1           1         1         1         2m

NAME                  CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
svc/guestbook         10.111.105.84    <none>        3000/TCP
2m
svc/hello-openshift   10.111.230.24    <none>
8080/TCP,8888/TCP   2m
svc/ruby-ex           10.111.232.117   <none>        8080/TCP
2m

NAME                          READY     STATUS      RESTARTS   AGE
po/guestbook-1-c010g          1/1       Running     0          2m
po/hello-openshift-1-4zw2q    1/1       Running     0          2m
po/ruby-ex-1-build            0/1       Completed   0          2m
po/ruby-ex-1-rxc74            1/1       Running     0          2m
```

2. Export the project objects into a **`project.yaml`** file in **`yaml`** format:

```
$ oc export all -o yaml > project.yaml
```

Or, in **`json`**:

```
$ oc export all -o json > project.json
```

3. The above creates a **`yaml`** or **`json`** file with the project content. This, however, does not export all objects, such as **`role bindings`**, **`secrets`**, **`service accounts`**, or **`persistent volume claims`**. To export these, run:

```
$ for object in rolebindings serviceaccounts secrets imagestreamtags
podpreset cms egressnetworkpolicies rolebindingrestrictions
limitranges resourcequotas pvcs templates cronjobs statefulsets hpas
deployments replicasets poddisruptionbudget endpoints
do
  oc export $object -o yaml > $object.yaml
done
```

4. Some exported objects can rely on specific metadata or references to unique IDs in the project. This is a limitation on the usability of the recreated objects.
When using **imagestreams**, the **image** parameter of a **deploymentconfig** can point to a specific **sha** checksum of an image in the internal registry that would not exist in a restored environment. For instance, running the sample "ruby-ex" as **oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git** creates an **imagestream ruby-ex** using the internal registry to host the image:

```
$ oc get dc ruby-ex -o jsonpath="
{.spec.template.spec.containers[].image}"
10.111.255.221:5000/myproject/ruby-
ex@sha256:880c720b23c8d15a53b01db52f7abdcbb2280e03f686a5c8edfef1a2a7
b21cee
```

If importing the **deploymentconfig** as it is exported with **oc export** it fails if the image does not exist.

To create those exports, use the **project_export.sh** in the **openshift-ansible-contrib** repository, which creates all the project objects in different files. The script creates a directory on the host running the script with the project name and a **json** file for every object type in that project.

NOTE

The code in the **openshift-ansible-contrib** repository referenced below is not explicitly supported by Red Hat but the Reference Architecture team performs testing to ensure the code operates as defined and is secure.

The script runs on Linux and requires **jq** and the **oc** commands installed and the system should be logged in to the OpenShift Container Platform environment as a user that can read all the objects in that project.

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-
contrib.git
$ cd openshift-ansible-contrib/reference-
architecture/day2ops/scripts
$ ./project_export.sh <projectname>
```

For example:

```
$ ./project_export.sh myproject
Exporting namespace to project-demo/ns.json
Exporting rolebindings to project-demo/rolebindings.json
Exporting serviceaccounts to project-demo/serviceaccounts.json
Exporting secrets to project-demo/secrets.json
Exporting deploymentconfigs to project-demo/dc_*.json
Patching DC...
Exporting buildconfigs to project-demo/bcs.json
Exporting builds to project-demo/builds.json
Exporting imagestreams to project-demo/iss.json
Exporting imagestreamtags to project-demo/imagestreamtags.json
Exporting replicationcontrollers to project-demo/rcs.json
```

```
Exporting services to project-demo/svc_*.json
Exporting pods to project-demo/pods.json
Exporting podpreset to project-demo/podpreset.json
Exporting configmaps to project-demo/cms.json
Exporting egressnetworkpolicies to project-
demo/egressnetworkpolicies.json
Exporting rolebindingrestrictions to project-
demo/rolebindingrestrictions.json
Exporting limitranges to project-demo/limitranges.json
Exporting resourcequotas to project-demo/resourcequotas.json
Exporting pvcs to project-demo/pvcs.json
Exporting routes to project-demo/routes.json
Exporting templates to project-demo/templates.json
Exporting cronjobs to project-demo/cronjobs.json
Exporting statefulsets to project-demo/statefulsets.json
Exporting hpas to project-demo/hpas.json
Exporting deployments to project-demo/deployments.json
Exporting replicasets to project-demo/replicasets.json
Exporting poddisruptionbudget to project-
demo/poddisruptionbudget.json
```

5. Once executed, review the files to verify that the content has been properly exported:

```
$ cd <projectname>
$ ls -1
bcs.json
builds.json
cms.json
cronjobs.json
dc_ruby-ex.json
dc_ruby-ex_patched.json
deployments.json
egressnetworkpolicies.json
endpoint_external-mysql-service.json
hpas.json
imagestreamtags.json
iss.json
limitranges.json
ns.json
poddisruptionbudget.json
podpreset.json
pods.json
pvcs.json
rcs.json
replicasets.json
resourcequotas.json
rolebindingrestrictions.json
rolebindings.json
routes.json
secrets.json
serviceaccounts.json
statefulsets.json
svc_external-mysql-service.json
svc_ruby-ex.json
```

```
templates.json
$ less bcs.json
...
```

NOTE

If the original object does not exist, empty files will be created when exporting.

6. If using **imagestreams**, the script modifies the **deploymentconfig** to use the image reference instead the image **sha**, creating a different **json** file than the exported using the **_patched** appendix:

```
$ diff dc_hello-openshift.json dc_hello-openshift_patched.json
45c45
<             "image": "docker.io/openshift/hello-
openshift@sha256:42b59c869471a1b5fdacadf778667cecbaa79e002b7235f8091
540ae612f0e14",
---
>             "image": "hello-openshift:latest",
```

WARNING

The script does not support multiple container pods currently, use it with caution.

4.4.2. Restore project

To restore a project, create the new project, then restore any exported files with **oc create -f pods.json**. However, restoring a project from scratch requires a specific order, because some objects are dependent on others. For example, the **configmaps** must be created before any **pods**.

Procedure

1. If the project has been exported as a single file, it can be imported as:

```
$ oc new-project <projectname>
$ oc create -f project.yaml
$ oc create -f secret.yaml
$ oc create -f serviceaccount.yaml
$ oc create -f pvc.yaml
$ oc create -f rolebindings.yaml
```

WARNING

Some resources can fail to be created (for example, pods and default service accounts).

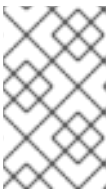
2. If the project was initially exported using the **`project_export.sh`** script, the files are located in the **`projectname`** directory, and can be imported using the same **`project_import.sh`** script that performs the **`oc create`** process in the proper order:

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-
contrib.git
$ cd openshift-ansible-contrib/reference-
architecture/day2ops/scripts
$ ./project_import.sh <projectname_path>
```

For example:

```
$ ls ~/backup/myproject
bcs.json           dc_guestbook_patched.json        dc_ruby-
ex_patched.json  pvcs.json          secrets.json
builds.json        dc_hello-openshift.json          iss.json
rcs.json           serviceaccounts.json
cms.json           dc_hello-openshift_patched.json  ns.json
rolebindings.json  svcs.json
dc_guestbook.json  dc_ruby-ex.json                  pods.json
routes.json        templates.json

$ ./project_import.sh ~/backup/myproject
namespace "myproject" created
rolebinding "admin" created
rolebinding "system:deployers" created
rolebinding "system:image-builders" created
rolebinding "system:image-pullers" created
secret "builder-dockercfg-mqhs6" created
secret "default-dockercfg-51xb9" created
secret "deployer-dockercfg-6kvz7" created
Error from server (AlreadyExists): error when creating
"myproject//serviceaccounts.json": serviceaccounts "builder" already
exists
Error from server (AlreadyExists): error when creating
"myproject//serviceaccounts.json": serviceaccounts "default" already
exists
Error from server (AlreadyExists): error when creating
"myproject//serviceaccounts.json": serviceaccounts "deployer"
already exists
error: no objects passed to create
service "guestbook" created
service "hello-openshift" created
service "ruby-ex" created
```

```
imagestream "guestbook" created
imagestream "hello-openshift" created
imagestream "ruby-22-centos7" created
imagestream "ruby-ex" created
error: no objects passed to create
error: no objects passed to create
buildconfig "ruby-ex" created
build "ruby-ex-1" created
deploymentconfig "guestbook" created
deploymentconfig "hello-openshift" created
deploymentconfig "ruby-ex" created
replicationcontroller "ruby-ex-1" created
Error from server (AlreadyExists): error when creating
"myproject//rcs.json": replicationcontrollers "guestbook-1" already
exists
Error from server (AlreadyExists): error when creating
"myproject//rcs.json": replicationcontrollers "hello-openshift-1"
already exists
pod "guestbook-1-c010g" created
pod "hello-openshift-1-4zw2q" created
pod "ruby-ex-1-rxc74" created
Error from server (AlreadyExists): error when creating
"myproject//pods.json": object is being deleted: pods "ruby-ex-1-
build" already exists
error: no objects passed to create
```



NOTE

AlreadyExists errors can appear, because some objects as **serviceaccounts** and secrets are created automatically when creating the project.

3. If you are using **buildconfigs**, the builds are not triggered automatically and the applications are not executed:

```
$ oc get bc
NAME      TYPE      FROM      LATEST
ruby-ex   Source    Git       1
$ oc get pods
NAME                        READY     STATUS    RESTARTS   AGE
guestbook-1-plnnq           1/1       Running   0          26s
hello-openshift-1-g4g0j     1/1       Running   0          26s
```

To trigger the builds, run the **oc start-build** command:

```
$ for bc in $(oc get bc -o jsonpath="{.items[*].metadata.name}")
do
    oc start-build ${bc}
done
```

The pods will deploy once the build completes.

4. To verify the project was restored:

```
$ oc get all
NAME         TYPE      FROM      LATEST
bc/ruby-ex   Source    Git       2

NAME               TYPE      FROM          STATUS
STARTED              DURATION
builds/ruby-ex-1   Source    Git           Error (BuildPodDeleted)
About a minute ago
builds/ruby-ex-2   Source    Git@c457001   Complete
55 seconds ago       12s

NAME                 DOCKER REPO
TAGS      UPDATED
is/guestbook         10.111.255.221:5000/myproject/guestbook
latest    About a minute ago
is/hello-openshift   10.111.255.221:5000/myproject/hello-openshift
latest    About a minute ago
is/ruby-22-centos7   10.111.255.221:5000/myproject/ruby-22-centos7
latest    About a minute ago
is/ruby-ex           10.111.255.221:5000/myproject/ruby-ex
latest    43 seconds ago

NAME                 REVISION   DESIRED   CURRENT   TRIGGERED BY
dc/guestbook         1          1         1
config,image(guestbook:latest)
dc/hello-openshift   1          1         1
config,image(hello-openshift:latest)
dc/ruby-ex           1          1         1
config,image(ruby-ex:latest)

NAME                   DESIRED   CURRENT   READY     AGE
rc/guestbook-1         1         1         1         1m
rc/hello-openshift-1   1         1         1         1m
rc/ruby-ex-1           1         1         1         43s

NAME                  CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
svc/guestbook         10.111.126.115   <none>        3000/TCP
1m
svc/hello-openshift   10.111.23.21     <none>
8080/TCP,8888/TCP   1m
svc/ruby-ex           10.111.162.157   <none>        8080/TCP
1m

NAME                          READY     STATUS      RESTARTS   AGE
po/guestbook-1-plnnq          1/1       Running     0          1m
po/hello-openshift-1-g4g0j    1/1       Running     0          1m
po/ruby-ex-1-h99np            1/1       Running     0          42s
po/ruby-ex-2-build            0/1       Completed   0          55s
```

NOTE

The services and pods IPs are different, because they are assigned dynamically at creation time.

4.5. CREATING A PVC BACKUP

This topic describes how to synchronize persistent data from inside of a container to a server and then restore the data onto a new persistent volume claim.

IMPORTANT

Depending on the provider that is hosting the OpenShift Container Platform environment, the ability to launch third party snapshot services for backup and restore purposes also exists. As OpenShift Container Platform does not have the ability to launch these services, this guide does not describe these steps.

Consult any product documentation for the correct backup procedures of specific applications. For example, copying the mysql data directory itself would not be a usable backup. Instead, run the specific backup procedures of the associated application and then synchronize any data. This includes using snapshot solutions provided by the OpenShift Container Platform hosting platform.

4.5.1. Backup persistent volume claims

Procedure

1. View the project and pods:

```
$ oc get pods
NAME           READY     STATUS      RESTARTS   AGE
demo-1-build   0/1       Completed   0          2h
demo-2-fxx6d   1/1       Running     0          1h
```

2. Describe the desired pod to find the volumes currently being used by a persistent volume:

```
$ oc describe pod demo-2-fxx6d
Name:   demo-2-fxx6d
Namespace:  test
Security Policy: restricted
Node:   ip-10-20-6-20.ec2.internal/10.20.6.20
Start Time:  Tue, 05 Dec 2017 12:54:34 -0500
Labels:   app=demo
   deployment=demo-2
   deploymentconfig=demo
Status:   Running
IP:   172.16.12.5
Controllers:  ReplicationController/demo-2
Containers:
  demo:
    Container ID:
docker://201f3e55b373641eb36945d723e1e212ecab847311109b5cee1fd010942
4217a
    Image:  docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Image ID:  docker-pullable://docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Port:  8080/TCP
    State:  Running
```

```
      Started:  Tue, 05 Dec 2017 12:54:52 -0500
    Ready:  True
    Restart Count: 0
    Volume Mounts:
      */opt/app-root/src/uploaded from persistent-volume (rw)*
      /var/run/secrets/kubernetes.io/serviceaccount from default-
token-8mmrk (ro)
    Environment Variables: <none>
...omitted...
```

The above shows that the persistent data is currently located in the **/opt/app-root/src/uploaded** directory.

3. Copy the data locally:

```
$ oc rsync demo-2-fxx6d:/opt/app-root/src/uploaded ./demo-app
receiving incremental file list
uploaded/
uploaded/ocp_sop.txt
uploaded/lost+found/

sent 38 bytes  received 190 bytes  152.00 bytes/sec
total size is 32  speedup is 0.14
```

The **ocp_sop.txt** file has been pulled down to the local system to be backed up by backup software or to another backup mechanism.

NOTE

The steps above can also be used in the event that a pod starts without needing to use a **pvc**, but then decides a **pvc** is necessary. This would allow for the data to be preserved and the restore procedures to be used to populate the new storage.

4.5.2. Restore persistent volume claims

This topic describes two methods for restoring data. The first involves deleting the file, then placing the file back in the expected location. The second example shows migrating persistent volume claims. The migration would occur in the event that the storage needs to be moved or in a disaster scenario when the backend storage no longer exists.

Check with the restore procedures for the specific application on any steps required to restore data to the application.

4.5.2.1. Restoring files to an existing PVC

Procedure

1. Delete the file:

```
$ oc rsh demo-2-fxx6d
sh-4.2$ ls */opt/app-root/src/uploaded/*
lost+found  ocp_sop.txt
```

```
sh-4.2$ *rm -rf /opt/app-root/src/uploaded/ocp_sop.txt*
sh-4.2$ *ls /opt/app-root/src/uploaded/*
lost+found
```

2. Replace the file from the server containing the rsync backup of the files that were in the pvc:

```
$ oc rsync uploaded demo-2-fxx6d:/opt/app-root/src/
```

3. Validate that the file is back on the pod by using **oc rsh** to connect to the pod and view the contents of the directory:

```
$ oc rsh demo-2-fxx6d
sh-4.2$ *ls /opt/app-root/src/uploaded/*
lost+found  ocp_sop.txt
```

4.5.2.2. Restoring data to a new PVC

The following steps assume that a new **pvc** has been created.

Procedure

1. Overwrite the currently defined **claim-name**:

```
$ oc volume dc/demo --add --name=persistent-volume \
  --type=persistentVolumeClaim --claim-name=filestore \ --mount-
path=/opt/app-root/src/uploaded --overwrite
```

2. Validate that the pod is using the new PVC:

```
$ oc describe dc/demo
Name:  demo
Namespace: test
Created: 3 hours ago
Labels:  app=demo
Annotations: openshift.io/generated-by=OpenShiftNewApp
Latest Version: 3
Selector: app=demo,deploymentconfig=demo
Replicas: 1
Triggers: Config, Image(demo@latest, auto=true)
Strategy: Rolling
Template:
  Labels: app=demo
  deploymentconfig=demo
  Annotations: openshift.io/container.demo.image.entrypoint=
["container-entrypoint","/bin/sh","-c","$STI_SCRIPTS_PATH/usage"]
  openshift.io/generated-by=OpenShiftNewApp
  Containers:
   demo:
    Image: docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Port: 8080/TCP
    Volume Mounts:
      /opt/app-root/src/uploaded from persistent-volume (rw)
```

```
    Environment Variables: <none>
  Volumes:
   persistent-volume:
    Type: PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
    *ClaimName: filestore*
    ReadOnly: false
...omitted...
```

3. Now that the new **pvc** is being used by the deployment configuration, use **oc rsync** to place the files onto the new **pvc**:

```
$ oc rsync uploaded demo-3-2b8gs:/opt/app-root/src/
sending incremental file list
uploaded/
uploaded/ocp_sop.txt
uploaded/lost+found/

sent 181 bytes  received 39 bytes  146.67 bytes/sec
total size is 32  speedup is 0.15
```

4. Validate that the file is back on the pod by using **oc rsh** to connect to the pod and view the contents of the directory.

```
$ oc rsh demo-3-2b8gs
sh-4.2$ ls /opt/app-root/src/uploaded/
lost+found  ocp_sop.txt
```

CHAPTER 5. HOST-LEVEL TASKS

5.1. ADDING A HOST TO THE CLUSTER

For information on adding master or node hosts to a cluster, see the Adding hosts to an existing cluster section in the Install and configuration guide.

5.2. MASTER HOST TASKS

5.2.1. Deprecating a master host

Deprecating a master host removes it from the OpenShift Container Platform environment.

The reasons to deprecate or scale down master hosts include hardware re-sizing or replacing the underlying infrastructure.

Highly available OpenShift Container Platform environments require at least three master hosts and three etcd nodes. Usually, the master hosts are collocated with the etcd services. This topic describes the deprecation process for master hosts with or without collocated etcd.

You should create a backup of the configuration and data files prior to any important task such as deprecating a master host. See the Creating a master host backup and etcd tasks sections for more information.

IMPORTANT

Ensure that the master and etcd services are always deployed in odd numbers due to the voting mechanisms that take place among those services.

5.2.1.1. Deprecating a master host without collocated etcd

Master hosts run important services, such as the OpenShift Container Platform API and controllers services (if multiple masters are present). In order to deprecate a master host, these services must be stopped.

The OpenShift Container Platform API service is an active/active service, so stopping the service does not affect the environment as long as the requests are sent to a separate master server. However, the OpenShift Container Platform controllers service is an active/passive service, where the services leverage etcd to decide the active master.

Deprecating a master host in a multi-master architecture includes removing the master from the load balancer pool to avoid new connections attempting to use that master. This process depends heavily on the load balancer used. The steps below show the details of removing the master from **haproxy**. In the event that OpenShift Container Platform is running on a cloud provider, or using a **F5** appliance, see the specific product documents to remove the master from rotation.

Procedure

1. Remove the **backend** section in the **/etc/haproxy/haproxy.cfg** configuration file. For example, if deprecating a master named **master-0.example.com** using **haproxy**, ensure the host name is removed from the following:

```
backend mgmt8443
    balance source
```

```
    mode tcp
    # MASTERS 8443
    server master-1.example.com 192.168.55.12:8443 check
    server master-2.example.com 192.168.55.13:8443 check
```

2. Then, restart the **haproxy** service.

```
$ sudo systemctl restart haproxy
```

3. Once the master is removed from the load balancer, disable the API and controller services:

```
$ sudo systemctl disable --now atomic-openshift-master-api
$ sudo systemctl disable --now atomic-openshift-master-controllers
```

4. Because the master host is a unschedulable OpenShift Container Platform node, follow the steps in the Deprecating a node host section.

5. Remove the master host from the **[masters]** and **[nodes]** groups in the **/etc/ansible/hosts** Ansible inventory file to avoid issues if running any Ansible tasks using that inventory file.

WARNING

Deprecating the first master host listed in the Ansible inventory file requires extra precautions.

At the time of writing, the **/etc/origin/master/ca.serial.txt** file is generated onto only the first master listed in the Ansible host inventory. This is being investigated to be fixed in future OpenShift Container Platform releases in the 1469358 bugzilla. If deprecating the first master host, copy the **/etc/origin/master/ca.serial.txt** file to the rest of master hosts before the process.

6. The **kubernetes** service includes the master host IPs as endpoints. To verify that the master has been properly deprecated, review the **kubernetes** service output and see if the deprecated master has been removed:

```
$ oc describe svc kubernetes -n default
Name:   kubernetes
Namespace:  default
Labels:   component=apiserver
   provider=kubernetes
Annotations:  <none>
Selector:  <none>
Type:   ClusterIP
IP:   10.111.0.1
Port:   https 443/TCP
Endpoints:  192.168.55.12:8443,192.168.55.13:8443
Port:   dns 53/UDP
```

```
Endpoints:  192.168.55.12:8053,192.168.55.13:8053
Port:   dns-tcp 53/TCP
Endpoints:  192.168.55.12:8053,192.168.55.13:8053
Session Affinity: ClientIP
Events:   <none>
```

Once the master has been successfully deprecated, the host where the master was previously running can be safely deleted.

5.2.1.2. Deprecating master host with collocated etcd

To deprecate a master host running an etcd service, execute the previous steps in Deprecating a master host without collocated etcd, as well as the steps in Removing an etcd host.

5.2.1.3. Replacing a master host

In the event of replacing a broken master host, follow the process in Deprecating a master host without collocated etcd, then scale up the master hosts using the scale up Ansible playbook following the steps in Adding hosts to an existing cluster.

If the master host has a collocated etcd, use the Deprecating master host with collocated etcd steps, then the Adding hosts to an existing cluster as well as Scaling etcd.

5.2.2. Creating a master host backup

The backup process is to be performed before any change to the infrastructure, such as a system update, upgrade, or any other significant modification. Backups should be performed on a regular basis to ensure the most recent data is available if a failure occurs.

OpenShift Container Platform files

The master instances run important services, such as the API, controllers. The **`/etc/origin/master`** directory stores many important files:

- The configuration, the API, controllers, services, and more
- Certificates generated by the installation
- All cloud provider-related configuration
- Keys and other authentication files, such as **`htpasswd`** if you use htpasswd
- And more

The OpenShift Container Platform services can be customized to increase the log level, use proxies, and so on. The configuration files are stored in the **`/etc/sysconfig`** directory.

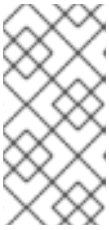
Because the masters are also unschedulable nodes, back up the entire **`/etc/origin`** directory.

Procedure

1. Create a backup of the master host configuration files:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo cp -aR /etc/origin ${MYBACKUPDIR}/etc
```

```
$ sudo cp -aR /etc/sysconfig/atomic-* ${MYBACKUPDIR}/etc/sysconfig/
```



NOTE

On a single master cluster installation the configuration file is stored in the **`/etc/sysconfig/atomic-openshift-master`**, whereas in a multi-master environment **`/etc/sysconfig/atomic-openshift-master-api`**, and **`/etc/sysconfig/atomic-openshift-master-controllers`** are used.

WARNING

At the time of writing, the **`/etc/origin/master/ca.serial.txt`** file is generated onto only the first master listed in the Ansible host inventory. This is being investigated to be fixed for future OpenShift Container Platform releases in the 1469358 bugzilla. If deprecating the first master host, copy the **`/etc/origin/master/ca.serial.txt`** file to the rest of master hosts before the process.

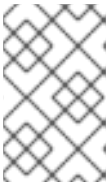
2. Other important files that need to be considered when planning a backup include:

File	Description
`/etc/cni/*`	Container Network Interface configuration (if used)
`/etc/sysconfig/iptables`	Where the **`iptables`** rules are stored
`/etc/sysconfig/docker-storage-setup`	The input file for **`container-storage-setup`** command
`/etc/sysconfig/docker`	The **`docker`** configuration file
`/etc/sysconfig/docker-network`	**`docker`** networking configuration (i.e. MTU)
`/etc/sysconfig/docker-storage`	**`docker`** storage configuration (generated by **`container-storage-setup`**)
`/etc/dnsmasq.conf`	Main configuration file for **`dnsmasq`**
`/etc/dnsmasq.d/*`	Different **`dnsmasq`** configuration files
`/etc/sysconfig/flanneld`	**`flannel`** configuration file (if used)
`/etc/pki/ca-trust/source/anchors/`	Certificates added to the system (i.e. for external registries)

Create a backup of those files:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo mkdir -p ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors
$ sudo cp -aR /etc/sysconfig/{iptables,docker-*,flanneld} \
    ${MYBACKUPDIR}/etc/sysconfig/
$ sudo cp -aR /etc/dnsmasq* /etc/cni ${MYBACKUPDIR}/etc/
$ sudo cp -aR /etc/pki/ca-trust/source/anchors/* \
    ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/
```

3. If a package is accidentally removed, or a file included in an **rpm** package should be restored, having a list of **rhel** packages installed on the system can be useful.



NOTE

If using Red Hat Satellite features, such as content views or the facts store, provide a proper mechanism to reinstall the missing packages and a historical data of packages installed in the systems.

To create a list of the current **rhel** packages installed in the system:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}
$ rpm -qa | sort | sudo tee $MYBACKUPDIR/packages.txt
```

4. If using the previous steps, the following files should now be present in the backup directory:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo find ${MYBACKUPDIR} -mindepth 1 -type f -printf '%P\n'
etc/sysconfig/atomic-openshift-master
etc/sysconfig/atomic-openshift-master-api
etc/sysconfig/atomic-openshift-master-controllers
etc/sysconfig/atomic-openshift-node
etc/sysconfig/flanneld
etc/sysconfig/iptables
etc/sysconfig/docker-network
etc/sysconfig/docker-storage
etc/sysconfig/docker-storage-setup
etc/sysconfig/docker-storage-setup.rpmnew
etc/origin/master/ca.crt
etc/origin/master/ca.key
etc/origin/master/ca.serial.txt
etc/origin/master/ca-bundle.crt
etc/origin/master/master.proxy-client.crt
etc/origin/master/master.proxy-client.key
etc/origin/master/service-signer.crt
etc/origin/master/service-signer.key
etc/origin/master/serviceaccounts.private.key
etc/origin/master/serviceaccounts.public.key
etc/origin/master/openshift-master.crt
etc/origin/master/openshift-master.key
etc/origin/master/openshift-master.kubeconfig
etc/origin/master/master.server.crt
```

```
etc/origin/master/master.server.key
etc/origin/master/master.kubelet-client.crt
etc/origin/master/master.kubelet-client.key
etc/origin/master/admin.crt
etc/origin/master/admin.key
etc/origin/master/admin.kubeconfig
etc/origin/master/etcd.server.crt
etc/origin/master/etcd.server.key
etc/origin/master/master.etcd-client.key
etc/origin/master/master.etcd-client.csr
etc/origin/master/master.etcd-client.crt
etc/origin/master/master.etcd-ca.crt
etc/origin/master/policy.json
etc/origin/master/scheduler.json
etc/origin/master/htpasswd
etc/origin/master/session-secrets.yaml
etc/origin/master/openshift-router.crt
etc/origin/master/openshift-router.key
etc/origin/master/registry.crt
etc/origin/master/registry.key
etc/origin/master/master-config.yaml
etc/origin/generated-configs/master-master-
1.example.com/master.server.crt
...[OUTPUT OMITTED]...
etc/origin/cloudprovider/openstack.conf
etc/origin/node/system:node:master-0.example.com.crt
etc/origin/node/system:node:master-0.example.com.key
etc/origin/node/ca.crt
etc/origin/node/system:node:master-0.example.com.kubeconfig
etc/origin/node/server.crt
etc/origin/node/server.key
etc/origin/node/node-dnsmasq.conf
etc/origin/node/resolv.conf
etc/origin/node/node-config.yaml
etc/origin/node/flannel.etcd-client.key
etc/origin/node/flannel.etcd-client.csr
etc/origin/node/flannel.etcd-client.crt
etc/origin/node/flannel.etcd-ca.crt
etc/pki/ca-trust/source/anchors/openshift-ca.crt
etc/pki/ca-trust/source/anchors/registry-ca.crt
etc/dnsmasq.conf
etc/dnsmasq.d/origin-dns.conf
etc/dnsmasq.d/origin-upstream-dns.conf
etc/dnsmasq.d/node-dnsmasq.conf
packages.txt
```

If needed, the files can be compressed to save space:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo tar -zcvf /backup/$(hostname)-$(date +%Y%m%d).tar.gz
$MYBACKUPDIR
$ sudo rm -Rf ${MYBACKUPDIR}
```

To create any of these files from scratch, the **openshift-ansible-contrib** repository contains the **backup_master_node.sh** script, which performs the previous steps. The script creates a directory on the host running the script and copies all the files previously mentioned.

NOTE

The **openshift-ansible-contrib** script is not supported by Red Hat, but the reference architecture team performs testing to ensure the code operates as defined and is secure.

The script can be executed on every master host with:

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-contrib.git
$ cd openshift-ansible-contrib/reference-architecture/day2ops/scripts
$ ./backup_master_node.sh -h
```

5.2.3. Restoring a master host backup

After creating a backup of important master host files, if they become corrupted or accidentally removed, you can restore the file by copying back the file, ensuring it contains the proper content and restart the affected services.

Procedure

1. Restore the **/etc/origin/master/master-config.yaml** file:

```
# MYBACKUPDIR=*/backup/$(hostname)/$(date +%Y%m%d)*
# cp /etc/origin/master/master-config.yaml
/etc/origin/master/master-config.yaml.old
# cp /backup/$(hostname)/$(date +%Y%m%d)/origin/master/master-
config.yaml /etc/origin/master/master-config.yaml
# systemctl restart atomic-openshift-master-api
# systemctl restart atomic-openshift-master-controllers
```

WARNING

Restarting the master services can lead to downtime. However, you can remove the master host from the highly available load balancer pool, then perform the restore operation. Once the service has been properly restored, you can add the master host back to the load balancer pool.

NOTE

Perform a full reboot of the affected instance to restore the **iptables** configuration.

2. If the issue is an accidental package and its dependencies are removed, reinstall the package. Get the list of the current installed packages:

```
$ rpm -qa | sort > /tmp/current_packages.txt
```

3. Get the differences:

```
$ diff /tmp/current_packages.txt ${MYBACKUPDIR}/packages.txt
1a2
> ansible-2.4.0.0-5.el7.noarch
```

4. Reinstall the missing packages:

```
# yum reinstall -y *ansible-2.4.0.0-5.el7.noarch*
```

5. Restore a system certificate by copying the certificate to the **/etc/pki/ca-trust/source/anchors/** directory and execute the **update-ca-trust**:

```
$ MYBACKUPDIR=*/backup/$(hostname)/$(date +%Y%m%d)*
$ sudo cp ${MYBACKUPDIR}/external_certificates/my_company.crt
/etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust
```

NOTE

Always ensure the user ID and group ID are restored when the files are copied back, as well as the **SELinux** context.

5.3. NODE HOST TASKS

5.3.1. Deprecating a node host

The procedure is the same whether deprecating an infrastructure node or an application node.

Prerequisites
Ensure enough capacity is available to migrate the existing pods from the node set to be removed. Removing an infrastructure node is advised only when at least two more nodes will stay online after the infrastructure node is removed.

Procedure

1. List all available nodes to find the node to deprecate:

```
$ oc get nodes
NAME                 STATUS                     AGE       VERSION
ocp-infra-node-b7pl  Ready                      23h
v1.6.1+5115d708d7
ocp-infra-node-p5zj  Ready                      23h
v1.6.1+5115d708d7
ocp-infra-node-rghb  Ready                      23h
v1.6.1+5115d708d7
ocp-master-dgf8      Ready,SchedulingDisabled   23h
v1.6.1+5115d708d7
```

```
ocp-master-q1v2       Ready,SchedulingDisabled   23h
v1.6.1+5115d708d7
ocp-master-vq70       Ready,SchedulingDisabled   23h
v1.6.1+5115d708d7
ocp-node-020m         Ready                      23h
v1.6.1+5115d708d7
ocp-node-7t5p         Ready                      23h
v1.6.1+5115d708d7
ocp-node-n0dd         Ready                      23h
v1.6.1+5115d708d7
```

As an example, this topic deprecates the **ocp-infra-node-b7pl** infrastructure node.

2. Describe the node and its running services:

```
$ oc describe node ocp-infra-node-b7pl
Name:   ocp-infra-node-b7pl
Role:
Labels:   beta.kubernetes.io/arch=amd64
   beta.kubernetes.io/instance-type=n1-standard-2
   beta.kubernetes.io/os=linux
   failure-domain.beta.kubernetes.io/region=europe-west3
   failure-domain.beta.kubernetes.io/zone=europe-west3-c
   kubernetes.io/hostname=ocp-infra-node-b7pl
   role=infra
Annotations:  volumes.kubernetes.io/controller-managed-attach-
detach=true
Taints:   <none>
CreationTimestamp: Wed, 22 Nov 2017 09:36:36 -0500
Phase:
Conditions:
  ...
Addresses:  10.156.0.11,ocp-infra-node-b7pl
Capacity:
 cpu:  2
 memory: 7494480Ki
 pods:  20
Allocatable:
 cpu:  2
 memory: 7392080Ki
 pods:  20
System Info:
 Machine ID:   bc95ccf67d047f2ae42c67862c202e44
 System UUID:   9762CC3D-E23C-AB13-B8C5-FA16F0BCCE4C
 Boot ID:   ca8bf088-905d-4ec0-beec-8f89f4527ce4
 Kernel Version:  3.10.0-693.5.2.el7.x86_64
 OS Image:   Employee SKU
 Operating System:  linux
 Architecture:   amd64
 Container Runtime Version: docker://1.12.6
 Kubelet Version:  v1.6.1+5115d708d7
 Kube-Proxy Version:  v1.6.1+5115d708d7
ExternalID:   437740049672994824
Non-terminated Pods:  (2 in total)
  Namespace   Name    CPU Requests CPU Limits Memory Requests Memory
Limits
```

```
  ---------   ----    ------------ ---------- --------------- ---
----------
  default   docker-registry-1-5szjs  100m (5%) 0 (0%)  256Mi (3%)0
(0%)
  default   router-1-vzlzq   100m (5%) 0 (0%)  256Mi (3%)0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests CPU Limits Memory Requests Memory Limits
  ------------ ---------- --------------- -------------
  200m (10%) 0 (0%)  512Mi (7%) 0 (0%)
Events:  <none>
```

The output above shows that the node is running two pods: **router-1-vzlzq** and **docker-registry-1-5szjs**. Two more infrastructure nodes are available to migrate these two pods.

NOTE

The cluster described above is a highly available cluster, this means both the **router** and **docker-registry** services are running on all infrastructure nodes.

3. Mark a node as unschedulable and evacuate all of its pods:

```
$ oc adm drain ocp-infra-node-b7pl --delete-local-data
node "ocp-infra-node-b7pl" cordoned
WARNING: Deleting pods with local storage: docker-registry-1-5szjs
pod "docker-registry-1-5szjs" evicted
pod "router-1-vzlzq" evicted
node "ocp-infra-node-b7pl" drained
```

If the pod has attached local storage (for example, **EmptyDir**), the **--delete-local-data** option must be provided. Generally, pods running in production should use the local storage only for temporary or cache files, but not for anything important or persistent. For regular storage, applications should use object storage or persistent volumes. In this case, the **docker-registry** pod's local storage is empty, because the object storage is used instead to store the container images.

NOTE

The above operation deletes existing pods running on the node. Then, new pods are created according to the replication controller.

In general, every application should be deployed with a deployment configuration, which creates pods using the replication controller.

oc adm drain will not delete any bare pods (pods that are neither mirror pods nor managed by **ReplicationController**, **ReplicaSet**, **DaemonSet**, **StatefulSet**, or a job). To do so, the **--force** option is required. Be aware that the bare pods will not be recreated on other nodes and data may be lost during this operation.

The example below shows the output of the replication controller of the registry:

```
$ oc describe rc/docker-registry-1
```

```
Name:  docker-registry-1
Namespace: default
Selector: deployment=docker-registry-1,deploymentconfig=docker-
registry,docker-registry=default
Labels:  docker-registry=default
  openshift.io/deployment-config.name=docker-registry
Annotations: ...
Replicas: 3 current / 3 desired
Pods Status: 3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  deployment=docker-registry-1
   deploymentconfig=docker-registry
   docker-registry=default
  Annotations:  openshift.io/deployment-config.latest-version=1
   openshift.io/deployment-config.name=docker-registry
   openshift.io/deployment.name=docker-registry-1
  Service Account: registry
  Containers:
   registry:
    Image: openshift3/ose-docker-registry:v3.6.173.0.49
    Port: 5000/TCP
    Requests:
      cpu: 100m
      memory: 256Mi
    Liveness: http-get https://:5000/healthz delay=10s timeout=5s
period=10s #success=1 #failure=3
    Readiness: http-get https://:5000/healthz delay=0s timeout=5s
period=10s #success=1 #failure=3
    Environment:
      REGISTRY_HTTP_ADDR:     :5000
      REGISTRY_HTTP_NET:     tcp
      REGISTRY_HTTP_SECRET:
tyGEnDZmc8dQfioP3WkNd5z+Xbdfy/JVXf/NLo3s/zE=
      REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA: false
      REGISTRY_HTTP_TLS_KEY:     /etc/secrets/registry.key
      OPENSHIFT_DEFAULT_REGISTRY:    docker-
registry.default.svc:5000
      REGISTRY_CONFIGURATION_PATH:    /etc/registry/config.yml
      REGISTRY_HTTP_TLS_CERTIFICATE:    /etc/secrets/registry.crt
    Mounts:
      /etc/registry from docker-config (rw)
      /etc/secrets from registry-certificates (rw)
      /registry from registry-storage (rw)
  Volumes:
   registry-storage:
    Type: EmptyDir (a temporary directory that shares a pod's
lifetime)
    Medium:
   registry-certificates:
    Type: Secret (a volume populated by a Secret)
    SecretName: registry-certificates
    Optional: false
   docker-config:
    Type: Secret (a volume populated by a Secret)
    SecretName: registry-config
    Optional: false
```

```
Events:
  FirstSeen LastSeen Count From   SubObjectPath Type  Reason
Message
  --------- -------- ----- ----   ------------- -------- ------
-------
  49m  49m  1 replication-controller   Normal  SuccessfulCreate
Created pod: docker-registry-1-dprp5
```

The event at the bottom of the output displays information about new pod creation. So, when listing all pods:

```
$ oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-1-dprp5    1/1       Running   0          52m
docker-registry-1-kr8jq    1/1       Running   0          1d
docker-registry-1-ncpl2    1/1       Running   0          1d
registry-console-1-g4nqg   1/1       Running   0          1d
router-1-2gshr             0/1       Pending   0          52m
router-1-85qm4             1/1       Running   0          1d
router-1-q5sr8             1/1       Running   0          1d
```

4. The **docker-registry-1-5szjs** and **router-1-vzlzq** pods that were running on the now deprecated node are no longer available. Instead, two new pods have been created: **docker-registry-1-dprp5** and **router-1-2gshr**. As shown above, the new router pod is **router-1-2gshr**, but is in the **Pending** state. This is because every node can be running only on one single router and is bound to the ports 80 and 443 of the host.

5. When observing the newly created registry pod, the example below shows that the pod has been created on the **ocp-infra-node-rghb** node, which is different from the deprecating node:

```
$ oc describe pod docker-registry-1-dprp5
Name:   docker-registry-1-dprp5
Namespace:  default
Security Policy: hostnetwork
Node:   ocp-infra-node-rghb/10.156.0.10
...
```

The only difference between deprecating the infrastructure and the application node is that once the infrastructure node is evacuated, and if there is no plan to replace that node, the services running on infrastructure nodes can be scaled down:

```
$ oc scale dc/router --replicas 2
deploymentconfig "router" scaled

$ oc scale dc/docker-registry --replicas 2
deploymentconfig "docker-registry" scaled
```

6. Now, every infrastructure node is running only one kind of each pod:

```
$ oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-1-kr8jq    1/1       Running   0          1d
docker-registry-1-ncpl2    1/1       Running   0          1d
```

```
registry-console-1-g4nqg   1/1       Running   0          1d
router-1-85qm4             1/1       Running   0          1d
router-1-q5sr8             1/1       Running   0          1d

$ oc describe po/docker-registry-1-kr8jq | grep Node:
Node:   ocp-infra-node-p5zj/10.156.0.9

$ oc describe po/docker-registry-1-ncpl2 | grep Node:
Node:   ocp-infra-node-rghb/10.156.0.10
```

NOTE

To provide a full highly available cluster, at least three infrastructure nodes should always be available.

7. To verify that the scheduling on the node is disabled:

```
$ oc get nodes
NAME                   STATUS                     AGE       VERSION
ocp-infra-node-b7pl    Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-infra-node-p5zj    Ready                      1d
v1.6.1+5115d708d7
ocp-infra-node-rghb    Ready                      1d
v1.6.1+5115d708d7
ocp-master-dgf8        Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-master-q1v2        Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-master-vq70        Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-node-020m          Ready                      1d
v1.6.1+5115d708d7
ocp-node-7t5p          Ready                      1d
v1.6.1+5115d708d7
ocp-node-n0dd          Ready                      1d
v1.6.1+5115d708d7
```

And that the node does not contain any pods:

```
$ oc describe node ocp-infra-node-b7pl
Name:   ocp-infra-node-b7pl
Role:
Labels:   beta.kubernetes.io/arch=amd64
   beta.kubernetes.io/instance-type=n1-standard-2
   beta.kubernetes.io/os=linux
   failure-domain.beta.kubernetes.io/region=europe-west3
   failure-domain.beta.kubernetes.io/zone=europe-west3-c
   kubernetes.io/hostname=ocp-infra-node-b7pl
   role=infra
Annotations:  volumes.kubernetes.io/controller-managed-attach-
detach=true
Taints:   <none>
CreationTimestamp: Wed, 22 Nov 2017 09:36:36 -0500
Phase:
```

```
Conditions:
  ...
Addresses:  10.156.0.11,ocp-infra-node-b7pl
Capacity:
 cpu:  2
 memory: 7494480Ki
 pods:  20
Allocatable:
 cpu:  2
 memory: 7392080Ki
 pods:  20
System Info:
 Machine ID:   bc95ccf67d047f2ae42c67862c202e44
 System UUID:   9762CC3D-E23C-AB13-B8C5-FA16F0BCCE4C
 Boot ID:   ca8bf088-905d-4ec0-beec-8f89f4527ce4
 Kernel Version:  3.10.0-693.5.2.el7.x86_64
 OS Image:   Employee SKU
 Operating System:  linux
 Architecture:   amd64
 Container Runtime Version: docker://1.12.6
 Kubelet Version:  v1.6.1+5115d708d7
 Kube-Proxy Version:  v1.6.1+5115d708d7
ExternalID:   437740049672994824
Non-terminated Pods:  (0 in total)
  Namespace   Name  CPU Requests CPU Limits Memory Requests Memory
Limits
  ---------   ----  ------------ ---------- --------------- -----
--------
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests CPU Limits Memory Requests Memory Limits
  ------------ ---------- --------------- -------------
  0 (0%) 0 (0%)  0 (0%)  0 (0%)
Events:  <none>
```

8. Remove the infrastructure instance from the **backend** section in the **`/etc/haproxy/haproxy.cfg`** configuration file:

```
backend router80
    balance source
    mode tcp
    server infra-1.example.com 192.168.55.12:80 check
    server infra-2.example.com 192.168.55.13:80 check

backend router443
    balance source
    mode tcp
    server infra-1.example.com 192.168.55.12:443 check
    server infra-2.example.com 192.168.55.13:443 check
```

9. Then, restart the **haproxy** service.

```
$ sudo systemctl restart haproxy
```

10. Remove the node from the cluster after all pods are evicted with command:

```
$ oc delete node ocp-infra-node-b7pl
node "ocp-infra-node-b7pl" deleted
```

```
$ oc get nodes
NAME                  STATUS                     AGE       VERSION
ocp-infra-node-p5zj   Ready                      1d
v1.6.1+5115d708d7
ocp-infra-node-rghb   Ready                      1d
v1.6.1+5115d708d7
ocp-master-dgf8       Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-master-q1v2       Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-master-vq70       Ready,SchedulingDisabled   1d
v1.6.1+5115d708d7
ocp-node-020m         Ready                      1d
v1.6.1+5115d708d7
ocp-node-7t5p         Ready                      1d
v1.6.1+5115d708d7
ocp-node-n0dd         Ready                      1d
v1.6.1+5115d708d7
```

NOTE

For more information on evacuating and draining pods or nodes, see Node maintenance section.

5.3.1.1. Replacing a node host

In the event that a node would need to be added in place of the deprecated node, follow the Adding hosts to an existing cluster section.

5.3.2. Creating a node host backup

Creating a backup of a node host is a different use case from backing up a master host. Because master hosts contain many important files, creating a backup is highly recommended. However, the nature of nodes is that anything special is replicated over the nodes in case of failover, and they typically do not contain data that is necessary to run an environment. If a backup of a node contains something necessary to run an environment, then a creating a backup is recommended.

The backup process is to be performed before any change to the infrastructure, such as a system update, upgrade, or any other significant modification. Backups should be performed on a regular basis to ensure the most recent data is available if a failure occurs.

OpenShift Container Platform files

Node instances run applications in the form of pods, which are based on containers. The `/etc/origin/` and `/etc/origin/node` directories house important files, such as:

- The configuration of the node services
- Certificates generated by the installation
- Cloud provider-related configuration

- Keys and other authentication files, such as the **dnsmasq** configuration

The OpenShift Container Platform services can be customized to increase the log level, use proxies, and more, and the configuration files are stored in the **/etc/sysconfig** directory.

Procedure

1. Create a backup of the node configuration files:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo cp -aR /etc/origin ${MYBACKUPDIR}/etc
$ sudo cp -aR /etc/sysconfig/atomic-openshift-node
${MYBACKUPDIR}/etc/sysconfig/
```

2. OpenShift Container Platform uses specific files that must be taken into account when planning the backup policy, including:

File	Description
/etc/cni/*	Container Network Interface configuration (if used)
/etc/sysconfig/iptables	Where the **iptables** rules are stored
/etc/sysconfig/docker-storage-setup	The input file for **container-storage-setup** command
/etc/sysconfig/docker	The **docker** configuration file
/etc/sysconfig/docker-network	**docker** networking configuration (i.e. MTU)
/etc/sysconfig/docker-storage	**docker** storage configuration (generated by **container-storage-setup**)
/etc/dnsmasq.conf	Main configuration file for **dnsmasq**
/etc/dnsmasq.d/*	Different **dnsmasq** configuration files
/etc/sysconfig/flanneld	**flannel** configuration file (if used)
/etc/pki/ca-trust/source/anchors/	Certificates added to the system (i.e. for external registries)

To create those files:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}/etc/sysconfig
$ sudo mkdir -p ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors
$ sudo cp -aR /etc/sysconfig/{iptables,docker-*,flanneld} \
    ${MYBACKUPDIR}/etc/sysconfig/
$ sudo cp -aR /etc/dnsmasq* /etc/cni ${MYBACKUPDIR}/etc/
```

```
$ sudo cp -aR /etc/pki/ca-trust/source/anchors/* \
    ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/
```

3. If a package is accidentally removed, or a file included in an **rpm** package should be restored, having a list of **rhel** packages installed on the system can be useful.

NOTE

If using Red Hat Satellite features, such as content views or the facts store, provide a proper mechanism to reinstall the missing packages and a historical data of packages installed in the systems.

To create a list of the current **rhel** packages installed in the system:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo mkdir -p ${MYBACKUPDIR}
$ rpm -qa | sort | sudo tee $MYBACKUPDIR/packages.txt
```

4. The following files should now be present in the backup directory:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
$ sudo find ${MYBACKUPDIR} -mindepth 1 -type f -printf '%P\n'
etc/sysconfig/atomic-openshift-node
etc/sysconfig/flanneld
etc/sysconfig/iptables
etc/sysconfig/docker-network
etc/sysconfig/docker-storage
etc/sysconfig/docker-storage-setup
etc/sysconfig/docker-storage-setup.rpmnew
etc/origin/node/system:node:app-node-0.example.com.crt
etc/origin/node/system:node:app-node-0.example.com.key
etc/origin/node/ca.crt
etc/origin/node/system:node:app-node-0.example.com.kubeconfig
etc/origin/node/server.crt
etc/origin/node/server.key
etc/origin/node/node-dnsmasq.conf
etc/origin/node/resolv.conf
etc/origin/node/node-config.yaml
etc/origin/node/flannel.etcd-client.key
etc/origin/node/flannel.etcd-client.csr
etc/origin/node/flannel.etcd-client.crt
etc/origin/node/flannel.etcd-ca.crt
etc/origin/cloudprovider/openstack.conf
etc/pki/ca-trust/source/anchors/openshift-ca.crt
etc/pki/ca-trust/source/anchors/registry-ca.crt
etc/dnsmasq.conf
etc/dnsmasq.d/origin-dns.conf
etc/dnsmasq.d/origin-upstream-dns.conf
etc/dnsmasq.d/node-dnsmasq.conf
packages.txt
```

If needed, the files can be compressed to save space:

```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
```

```
$ sudo tar -zcvf /backup/$(hostname)-$(date +%Y%m%d).tar.gz
$MYBACKUPDIR
$ sudo rm -Rf ${MYBACKUPDIR}
```

To create any of these files from scratch, the **openshift-ansible-contrib** repository contains the **backup_master_node.sh** script, which performs the previous steps. The script creates a directory on the host running the script and copies all the files previously mentioned.

NOTE

The **openshift-ansible-contrib** script is not supported by Red Hat, but the reference architecture team performs testing to ensure the code operates as defined and is secure.

The script can be executed on every master host with:

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-contrib.git
$ cd openshift-ansible-contrib/reference-architecture/day2ops/scripts
$ ./backup_master_node.sh -h
```

5.3.3. Restoring a node host backup

After creating a backup of important node host files, if they become corrupted or accidentally removed, you can restore the file by copying back the file, ensuring it contains the proper content and restart the affected services.

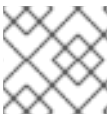
Procedure

1. Restore the **/etc/origin/node/node-config.yaml** file:

```
# MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
# cp /etc/origin/node/node-config.yaml /etc/origin/node/node-
config.yaml.old
# cp /backup/$(hostname)/$(date +%Y%m%d)/etc/origin/node/node-
config.yaml /etc/origin/node/node-config.yaml
# systemctl restart atomic-openshift-node
```

WARNING

Restarting the services can lead to downtime. See Node maintenance, for tips on how to ease the process.



NOTE

Perform a full reboot of the affected instance to restore the **iptables** configuration.

1. If the issue is an accidental package and its dependencies are removed, reinstall the package. Get the list of the current installed packages:

```
$ rpm -qa | sort > /tmp/current_packages.txt
```

2. Get the differences:

```
$ diff /tmp/current_packages.txt ${MYBACKUPDIR}/packages.txt
1a2
> ansible-2.4.0.0-5.el7.noarch
```

3. Reinstall the missing packages:

```
# yum reinstall -y ansible-2.4.0.0-5.el7.noarch
```

4. Restore a system certificate by copying the certificate to the **/etc/pki/ca-trust/source/anchors/** directory and execute the **update-ca-trust**:

```
$ MYBACKUPDIR=*/backup/$(hostname)/$(date +%Y%m%d)*
$ sudo cp ${MYBACKUPDIR}/etc/pki/ca-
trust/source/anchors/my_company.crt /etc/pki/ca-
trust/source/anchors/
$ sudo update-ca-trust
```

NOTE

Always ensure proper user ID and group ID are restored when the files are copied back, as well as the **SELinux** context.

5.3.4. Node maintenance and next steps

See Managing nodes or Managing pods topics for various node management options. These include:

- Marking Nodes as Unschedulable or Schedulable
- Evacuating Pods on Nodes
- Setting Pod Disruption Budgets

A node can reserve a portion of its resources to be used by specific components. These include the kubelet, kube-proxy, Docker, or other remaining system components such as **sshd** and **NetworkManager**. See the Allocating node resources section in the Cluster Administrator guide for more information.

5.4. ETCD TASKS

5.4.1. etcd backup

etcd is the key value store for all object definitions, as well as the persistent master state. Other components watch for changes, then bring themselves into the desired state.

OpenShift Container Platform versions prior to 3.5 use etcd version 2 (v2), while 3.5 and later use version 3 (v3). The data model between the two versions of etcd is different. etcd v3 can use both the

v2 and v3 data models, whereas etcd v2 can only use the v2 data model. In an etcd v3 server, the v2 and v3 data stores exist in parallel and are independent.

For both v2 and v3 operations, you can use the **ETCDCTL_API** environment variable to use the proper API:

```
$ etcdctl -v
etcdctl version: 3.2.5
API version: 2
$ ETCDCTL_API=3 etcdctl version
etcdctl version: 3.2.5
API version: 3.2
```

See Migrating etcd Data (v2 to v3) section for information about how to migrate to v3.

The etcd backup process is composed of two different procedures:

- Configuration backup: Including the required etcd configuration and certificates
- Data backup: Including both v2 and v3 data model.

You can perform the data backup process on any host that has connectivity to the etcd cluster, where the proper certificates are provided, and where the **etcdctl** tool is installed.

NOTE

The backup files must be copied to an external system, ideally outside the OpenShift Container Platform environment, and then encrypted.

5.4.2. etcd configuration backup

The etcd configuration files to be preserved are all stored in the **/etc/etcd** directory of the instances where etcd is running. This includes the etcd configuration file (**/etc/etcd/etcd.conf**) and the required certificates for cluster communication. All those files are generated at installation time by the Ansible installer.

For each etcd member of the cluster, back up the etcd configuration.

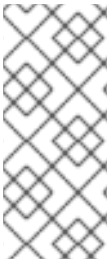
```
$ ssh master-0
# mkdir -p /backup/etcd-config-$(date +%Y%m%d)/
# cp -R /etc/etcd/ /backup/etcd-config-$(date +%Y%m%d)/
```

NOTE

The certificates and configuration files on each etcd cluster member are unique.

5.4.3. etcd data backup

Prerequisites



NOTE

The OpenShift Container Platform installer creates aliases to avoid typing all the flags named **etcdctl2** for etcd v2 tasks and **etcdctl3** for etcd v3 tasks.

However, the **etcdctl3** alias does not provide the full endpoint list to the **etcdctl** command, so the **--endpoints** option with all the endpoints must be provided.

Before backing up etcd:

- **etcdctl** binaries should be available or, in containerized installations, the **rhel7/etcd** container should be available
- Ensure connectivity with the etcd cluster (port 2379/tcp)
- Ensure the proper certificates to connect to the etcd cluster

1. To ensure the etcd cluster is working, check its health.
   - If you use the etcd v2 API, run the following command:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
          --key-file=/etc/etcd/peer.key \
          --ca-file=/etc/etcd/ca.crt \
          --peers="https://*master-0.example.com*:2379,\
          https://*master-1.example.com*:2379,\
          https://*master-2.example.com*:2379"\
          cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

   - If you use the etcd v3 API, run the following command:

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
          --key=/etc/etcd/peer.key \
          --cacert="/etc/etcd/ca.crt" \
          --endpoints="https://*master-0.example.com*:2379,\
            https://*master-1.example.com*:2379,\
            https://*master-2.example.com*:2379"
            endpoint health
https://master-0.example.com:2379 is healthy: successfully
committed proposal: took = 5.011358ms
https://master-1.example.com:2379 is healthy: successfully
committed proposal: took = 1.305173ms
https://master-2.example.com:2379 is healthy: successfully
committed proposal: took = 1.388772ms
```

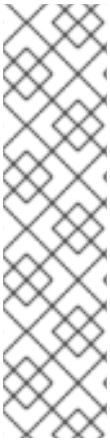
2. Check the member list.
   - If you use the etcd v2 API, run the following command:

```
# etcdctl2 member list
2a371dd20f21ca8d: name=master-1.example.com
peerURLs=https://192.168.55.12:2380
clientURLs=https://192.168.55.12:2379 isLeader=false
40bef1f6c79b3163: name=master-0.example.com
peerURLs=https://192.168.55.8:2380
clientURLs=https://192.168.55.8:2379 isLeader=false
95dc17ffcce8ee29: name=master-2.example.com
peerURLs=https://192.168.55.13:2380
clientURLs=https://192.168.55.13:2379 isLeader=true
```

- If you use the etcd v3 API, run the following command:

```
# etcdctl3 member list
2a371dd20f21ca8d, started, master-1.example.com,
https://192.168.55.12:2380, https://192.168.55.12:2379
40bef1f6c79b3163, started, master-0.example.com,
https://192.168.55.8:2380, https://192.168.55.8:2379
95dc17ffcce8ee29, started, master-2.example.com,
https://192.168.55.13:2380, https://192.168.55.13:2379
```

Procedure



NOTE

While the **`etcdctl backup`** command is used to perform the backup, etcd v3 has no concept of a *backup*. Instead, you either take a *snapshot* from a live member with the **`etcdctl snapshot save`** command or copy the **`member/snap/db`** file from an etcd data directory.

The **`etcdctl backup`** command rewrites some of the metadata contained in the backup, specifically, the node ID and cluster ID, which means that in the backup, the node loses its former identity. To recreate a cluster from the backup, you create a new, single-node cluster, then add the rest of the nodes to the cluster. The metadata is rewritten to prevent the new node from joining an existing cluster.

1. Back up the etcd data:

   - If you use the v2 API, take the following actions:

     a. Stop all etcd services:

     ```
     # systemctl stop etcd.service
     ```

     b. Create the etcd data backup and copy the etcd **db** file:

     ```
     # mkdir -p /backup/etcd-$(date +%Y%m%d)
     # etcdctl2 backup \
         --data-dir /var/lib/etcd \
         --backup-dir /backup/etcd-$(date +%Y%m%d)
     # cp /var/lib/etcd/member/snap/db /backup/etcd-$(date +%Y%m%d)
     ```

   - If you use the v3 API, run the following command:

```
# mkdir -p /backup/etcd-$(date +%Y%m%d)
# etcdctl3 snapshot save */backup/etcd-$(date +%Y%m%d)*/db
Snapshot saved at /backup/etcd-<date>/db
# systemctl stop etcd.service
# etcdctl2 backup \
    --data-dir /var/lib/etcd \
    --backup-dir /backup/etcd-$(date +%Y%m%d)
# systemctl start etcd.service
```

NOTE

The **`etcdctl snapshot save`** command requires the etcd service to be running.

In these commands, a **`/backup/etcd-<date>/`** directory is created, where **`<date>`** represents the current date, which must be an external NFS share, S3 bucket, or any external storage location.

In the case of an all-in-one cluster, the etcd data directory is located in the **`/var/lib/origin/openshift.local.etcd`** directory.

5.4.4. etcd restore

The restore procedure for etcd configuration files replaces the appropriate files, then restarts the service.

If an etcd host has become corrupted and the **`/etc/etcd/etcd.conf`** file is lost, restore it using:

```
$ ssh master-0
# cp /backup/yesterday/master-0-files/etcd.conf /etc/etcd/etcd.conf
# restorecon -Rv /etc/etcd/etcd.conf
# systemctl restart etcd.service
```

In this example, the backup file is stored in the **`/backup/yesterday/master-0-files/etcd.conf`** path where it can be used as an external NFS share, S3 bucket, or other storage solution.

5.4.4.1. Restoring etcd v2 & v3 data

The following process restores healthy data files and starts the etcd cluster as a single node, then adds the rest of the nodes if an etcd cluster is required.

Procedure

1. Stop all etcd services:

   ```
   # systemctl stop etcd.service
   ```

2. To ensure the proper backup is restored, delete the etcd directories:

   - To back up the current etcd data before you delete the directory, run the following command:

```
# mv /var/lib/etcd /var/lib/etcd.old
# mkdir /var/lib/etcd
# chown -R etcd.etcd /var/lib/etcd/
# restorecon -Rv /var/lib/etcd/
```

- Or, to delete the directory and the etcd, data, run the following command:

```
# rm -Rf /var/lib/etcd/*
```

NOTE

In an all-in-one cluster, the etcd data directory is located in the **`/var/lib/origin/openshift.local.etcd`** directory.

3. Restore a healthy backup data file to each of the etcd nodes. Perform this step on all etcd hosts, including master hosts collocated with etcd.

```
# cp -R /backup/etcd-xxx/* /var/lib/etcd/
# mv /var/lib/etcd/db /var/lib/etcd/member/snap/db
```

4. Run the etcd service on each host, forcing a new cluster.
This creates a custom file for the etcd service, which overwrites the execution command adding the **`--force-new-cluster`** option:

```
# mkdir -p /etc/systemd/system/etcd.service.d/
# echo "[Service]" > /etc/systemd/system/etcd.service.d/temp.conf
# echo "ExecStart=" >> /etc/systemd/system/etcd.service.d/temp.conf
# sed -n '/ExecStart/s/"$/ --force-new-cluster"/p' \
    /usr/lib/systemd/system/etcd.service \
    >> /etc/systemd/system/etcd.service.d/temp.conf

# systemctl daemon-reload
# systemctl restart etcd
```

5. Check for error messages:

```
$ journalctl -fu etcd.service
```

6. Check for health status:

```
# etcdctl2 cluster-health
member 5ee217d17301 is healthy: got healthy result from
https://192.168.55.8:2379
cluster is healthy
```

7. Restart the etcd service in cluster mode:

```
# rm -f /etc/systemd/system/etcd.service.d/temp.conf
# systemctl daemon-reload
# systemctl restart etcd
```

8. Check for health status and member list:

```
# etcdctl2 cluster-health
member 5ee217d17301 is healthy: got healthy result from
https://192.168.55.8:2379
cluster is healthy

# etcdctl2 member list
5ee217d17301: name=master-0.example.com
peerURLs=http://localhost:2380 clientURLs=https://192.168.55.8:2379
isLeader=true
```

9. After the first instance is running, you can restore the rest of your etcd servers.

Fix the peerURLS parameter

After restoring the data and creating a new cluster, the **peerURLs** parameter shows **localhost** instead of the IP where etcd is listening for peer communication:

```
# etcdctl2 member list
5ee217d17301: name=master-0.example.com peerURLs=http://*localhost*:2380
clientURLs=https://192.168.55.8:2379 isLeader=true
```

Procedure

1. Get the member ID using **etcdctl member list**:

```
`etcdctl member list`
```

2. Get the IP where etcd listens for peer communication:

```
$ ss -l4n | grep 2380
```

3. Update the member information with that IP:

```
# etcdctl2 member update 5ee217d17301 https://192.168.55.8:2380
Updated member with ID 5ee217d17301 in cluster
```

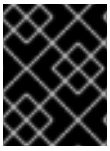
4. To verify, check that the IP is in the member list:

```
$ etcdctl2 member list
5ee217d17301: name=master-0.example.com
peerURLs=https://*192.168.55.8*:2380
clientURLs=https://192.168.55.8:2379 isLeader=true
```

5.4.4.2. Restoring etcd for v3

The restore procedure for v3 data is similar to the restore procedure for the v2 data.

Snapshot integrity may be optionally verified at restore time. If the snapshot is taken with **etcdctl snapshot save**, it will have an integrity hash that is checked by **etcdctl snapshot restore**. If the snapshot is copied from the data directory, there is no integrity hash and it will only restore by using **--skip-hash-check**.



IMPORTANT

The procedure to restore only the v3 data must be performed on a single etcd host. You can then add the rest of the nodes to the cluster.

Procedure

1. Stop all etcd services:

```
# systemctl stop etcd.service
```

2. Clear all old data, because **etcdctl** recreates it in the node where the restore procedure is going to be performed:

```
# rm -Rf /var/lib/etcd
```

3. Run the **snapshot restore** command, substituting the values from the **/etc/etcd/etcd.conf** file:

```
# etcdctl3 snapshot restore /backup/etcd-xxxxxx/backup.db \
  --data-dir /var/lib/etcd \
  --name master-0.example.com \
  --initial-cluster "master-0.example.com=https://192.168.55.8:2380"
\ --initial-cluster-token "etcd-cluster-1" \
  --initial-advertise-peer-urls https://192.168.55.8:2380

2017-10-03 08:55:32.440779 I | mvcc: restore compact to 1041269
2017-10-03 08:55:32.468244 I | etcdserver/membership: added member
40bef1f6c79b3163 [https://192.168.55.8:2380] to cluster
26841ebcf610583c
```

4. Restore permissions and **selinux** context to the restored files:

```
# chown -R etcd.etcd /var/lib/etcd/
# restorecon -Rv /var/lib/etcd
```

5. Start the etcd service:

```
# systemctl start etcd
```

6. Check for any error messages:

```
$ journalctl -fu etcd.service
```

5.4.5. etcd restore

After the first instance is running, you can add multiple etcd servers to your cluster.

Procedure

1. Get the etcd name for the instance in the **ETCD_NAME** variable:

```
# grep ETCD_NAME /etc/etcd/etcd.conf
```

2. Get the IP address where etcd listens for peer communication:

```
# grep ETCD_INITIAL_ADVERTISE_PEER_URLS /etc/etcd/etcd.conf
```

3. If the node was previously part of a etcd cluster, delete the previous etcd data:

```
# rm -Rf /var/lib/etcd/*
```

4. On the etcd host where etcd is properly running, add the new member:

   - If you use the v2 etcd api, run this command:

```
$ etcdctl2 member add <name> <advertise_peer_urls>
```

   - If you use the v3 etcd api, run this command:

```
# etcdctl3 member add *<name>* \
  --peer-urls="*<advertise_peer_urls>*"
```

   The command outputs some variables. For example:

```
ETCD_NAME="master2"
ETCD_INITIAL_CLUSTER="master-
0.example.com=https://192.168.55.8:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

5. Add the values from the previous command to the **/etc/etcd/etcd.conf** file of the new host:

```
# vi /etc/etc/etcd.conf
```

6. Start the etcd service in the node joining the cluster:

```
# systemctl start etcd.service
```

7. Check for error messages:

```
$ journalctl -fu etcd.service
```

8. Repeat the previous steps for every etcd node to be added.

9. Once you add all the nodes, verify the cluster status and cluster health:

   - If you use the v2 etcd api, run the following commands:

```
# etcdctl2 member list
5cd050b4d701: name=master1 peerURLs=https://10.0.0.7:2380
clientURLs=https://10.0.0.7:2379 isLeader=true
d0c57659d8990cbd: name=master2 peerURLs=https://10.0.0.5:2380
clientURLs=https://10.0.0.5:2379 isLeader=false
e4696d637de3eb2d: name=master3 peerURLs=https://10.0.0.6:2380
clientURLs=https://10.0.0.6:2379 isLeader=false
```

```
# etcdctl2 cluster-health
member 5cd050b4d701 is healthy: got healthy result from
https://10.0.0.7:2379
member d0c57659d8990cbd is healthy: got healthy result from
https://10.0.0.5:2379
member e4696d637de3eb2d is healthy: got healthy result from
https://10.0.0.6:2379
cluster is healthy
```

- If you use the v3 etcd api, run the following commands:

```
# etcdctl3 endpoint health
https://master-0.example.com:2379 is healthy: successfully
committed proposal: took = 1.423459ms
https://master-1.example.com:2379 is healthy: successfully
committed proposal: took = 1.767481ms
https://master-2.example.com:2379 is healthy: successfully
committed proposal: took = 1.599694ms

# etcdctl3 endpoint status
https://master-0.example.com:2379, 40bef1f6c79b3163, 3.2.5, 28
MB, true, 9, 2878
https://master-1.example.com:2379, 1ea57201a3ff620a, 3.2.5, 28
MB, false, 9, 2878
https://master-2.example.com:2379, 59229711e4bc65c8, 3.2.5, 28
MB, false, 9, 2878
```

5.4.6. Scaling etcd

You can scale the etcd cluster vertically by adding more resources to the etcd hosts or horizontally by adding more etcd hosts.

NOTE

Due to the voting system etcd uses, the cluster must always contain an odd number of members.

The new host requires a fresh Red Hat Enterprise Linux version 7 dedicated host. The etcd storage should be located on an SSD disk to achieve maximum performance and on a dedicated disk mounted in `/var/lib/etcd`.

NOTE

OpenShift Container Platform version 3.7 ships with an automated way to add a new etcd host using Ansible.

Prerequisites

1. Before adding a new etcd host, perform a backup of both etcd configuration and data to prevent data loss.

2. Check the current etcd cluster status to avoid adding new hosts to an unhealthy cluster:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
```

```
          --key-file=/etc/etcd/peer.key \
          --ca-file=/etc/etcd/ca.crt \
          --peers="https://*master-0.example.com*:2379,\
          https://*master-1.example.com*:2379,\
          https://*master-2.example.com*:2379"\
          cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

Or, using etcd v3 API:

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
          --key=/etc/etcd/peer.key \
          --cacert="/etc/etcd/ca.crt" \
          --endpoints="https://*master-0.example.com*:2379,\
            https://*master-1.example.com*:2379,\
            https://*master-2.example.com*:2379"
            endpoint health
https://master-0.example.com:2379 is healthy: successfully committed
proposal: took = 5.011358ms
https://master-1.example.com:2379 is healthy: successfully committed
proposal: took = 1.305173ms
https://master-2.example.com:2379 is healthy: successfully committed
proposal: took = 1.388772ms
```

3. Before running the **scaleup** playbook, ensure the new host is registered to the proper Red Hat software channels:

```
# subscription-manager register \
    --username=*<username>* --password=*<password>*
# subscription-manager attach --pool=*<poolid>*
# subscription-manager repos --disable="*"
# subscription-manager repos \
    --enable=rhel-7-server-rpms \
    --enable=rhel-7-server-extras-rpms
```

etcd is hosted in the **rhel-7-server-extras-rpms** software channel.

5.4.6.1. Adding a new etcd host using Ansible

Procedure

1. Modify the Ansible inventory file and create a new group named **[new_etcd]** and add the new host. Then, add the **new_etcd** group as a child of the **[OSEv3]** group:

```
[OSEv3:children]
masters
nodes
etcd
<new_etcd>
```

```
... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com

[new_etcd]
etcd0.example.com
```

2. Run the etcd **scaleup** playbook from the host that executed the initial installation and where the Ansible inventory file is:

```
$ ansible-playbook  /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-etcd/scaleup.yml
```

3. After the previous step above has completed, modify the inventory file to reflect the current status by moving the new etcd host from the **[new_etcd]** group to the **[etcd]** group:

```
[OSEv3:children]
masters
nodes
etcd
<new_etcd>

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com
etcd0.example.com
```

4. If you are using Flannel, modify the **flanneld** service configuration, located at **/etc/sysconfig/flanneld** on every OpenShift Container Platform host, to include the new etcd host:

```
FLANNEL_ETCD_ENDPOINTS=https://master-
0.example.com:2379,https://master-1.example.com:2379,https://master-
2.example.com:2379,https://etcd0.example.com:2379
```

5. Restart the **flanneld** service:

```
# systemctl restart flanneld.service
```

5.4.6.2. Manually adding a new etcd host

The following steps can be performed on any etcd member. If using the Ansible installer, the first host provided in the **[etcd]** Ansible inventory is used to generate the etcd configuration and certificates stored in **/etc/etcd/generated_certs**, so perform the next steps in that etcd host.

Steps to be performed on the current etcd cluster

Procedure
In order to create the etcd certificates, run the **openssl** command replacing the values with those from your environment.

1. Create some environment variables:

```
export NEW_ETCD_HOSTNAME="*etcd0.example.com*"
export NEW_ETCD_IP="192.168.55.21"

export CN=$NEW_ETCD_HOSTNAME
export SAN="IP:${NEW_ETCD_IP}"
export PREFIX="/etc/etcd/generated_certs/etcd-$CN/"
export OPENSSLCFG="/etc/etcd/ca/openssl.cnf"
```

NOTE

The custom **openssl** extensions used as **etcd_v3_ca_*** include the $SAN environment variable as **subjectAltName**. See **/etc/etcd/ca/openssl.cnf** for more information.

2. Create the directory to store the configuration and certificates:

```
# mkdir -p ${PREFIX}
```

3. Create the server certificate request and sign it:

```
# openssl req -new -config ${OPENSSLCFG} \
    -keyout ${PREFIX}server.key  \
    -out ${PREFIX}server.csr \
    -reqexts etcd_v3_req -batch -nodes \
    -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
    -out ${PREFIX}server.crt \
    -in ${PREFIX}server.csr \
    -extensions etcd_v3_ca_server -batch
```

4. Create the peer certificate request and sign it:

```
# openssl req -new -config ${OPENSSLCFG} \
    -keyout ${PREFIX}peer.key \
    -out ${PREFIX}peer.csr \
    -reqexts etcd_v3_req -batch -nodes \
    -subj /CN=$CN

# openssl ca -name etcd_ca -config ${OPENSSLCFG} \
  -out ${PREFIX}peer.crt \
  -in ${PREFIX}peer.csr \
  -extensions etcd_v3_ca_peer -batch
```

5. Copy the current etcd configuration and **ca.crt** files from the current node as examples to be modified later:

```
# cp /etc/etcd/etcd.conf ${PREFIX}
# cp /etc/etcd/ca.crt ${PREFIX}
```

6. Add the new host to the etcd cluster. Note the new host is not configured yet so the status stays as **`unstarted`** until the new host is properly configured:

```
# etcdctl2 member add ${NEW_ETCD_HOSTNAME}
https://${NEW_ETCD_IP}:2380
```

This command outputs the following variables:

```
ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="
<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
<CLUSTERMEMBER1_NAME>=https:/<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER2_NAME>=https:/<CLUSTERMEMBER2_IP>:2380,
<CLUSTERMEMBER3_NAME>=https:/<CLUSTERMEMBER3_IP>:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

7. Those values must be overwritten by the current ones in the sample **`${PREFIX}/etcd.conf`** file. Also, modify the following variables with the new host IP (**`${NEW_ETCD_IP}`** can be used) in that file from the output of the last command:

```
ETCD_LISTEN_PEER_URLS
ETCD_LISTEN_CLIENT_URLS
ETCD_INITIAL_ADVERTISE_PEER_URLS
ETCD_ADVERTISE_CLIENT_URLS
```

8. Modify the **`${PREFIX}/etcd.conf`** file, checking for syntax errors or missing IPs, otherwise the etcd service might fail:

```
# vi ${PREFIX}/etcd.conf
```

9. Once the file has been properly modified, a **`tgz`** file with the certificates, the sample configuration file, and the **`ca`** is created and copied to the new host:

```
# tar -czvf /etc/etcd/generated_certs/${CN}.tgz -C ${PREFIX} .
# scp /etc/etcd/generated_certs/${CN}.tgz ${CN}:/tmp/
```

Steps to be performed on the new etcd host

Procedure

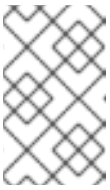
1. Install **`iptables-services`** to provide iptables utilities to open the required ports for etcd:

```
# yum install -y iptables-services
```

2. Create the **`OS_FIREWALL_ALLOW`** firewall rules to allow etcd to communicate:

- Port 2379/tcp for clients
- Port 2380/tcp for peer communication

```
# systemctl enable iptables.service --now
# iptables -N OS_FIREWALL_ALLOW
# iptables -t filter -I INPUT -j OS_FIREWALL_ALLOW
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m
tcp --dport 2379 -j ACCEPT
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m
tcp --dport 2380 -j ACCEPT
# iptables-save | tee /etc/sysconfig/iptables
```



NOTE

In this example, a new chain **OS_FIREWALL_ALLOW** is created, which is the standard naming the OpenShift Container Platform installer uses for firewall rules.

WARNING

If the environment is hosted in an IaaS environment, modify the security groups for the instance to allow incoming traffic to those ports as well.

3. Install etcd:

```
# yum install -y etcd
```

4. Ensure the etcd service is not running:

```
# systemctl disable etcd --now
```

5. Remove any etcd configuration and data:

```
# rm -Rf /etc/etcd/*
# rm -Rf /var/lib/etcd/*
```

6. Untar the certificates and configuration files:

```
# tar xzvf /tmp/etcd0.example.com.tgz -C /etc/etcd/
```

7. Modify the file ownership permissions:

```
# chown -R etcd.etcd /etc/etcd/
# chown -R etcd.etcd /var/lib/etcd/
```

8. Start etcd on the new host:

```
# systemctl enable etcd --now
```

9. Verify the host has been added to the cluster and the current cluster health:

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
          --key-file=/etc/etcd/peer.key \
          --ca-file=/etc/etcd/ca.crt \
          --peers="https://*master-0.example.com*:2379,\
          https://*master-1.example.com*:2379,\
          https://*master-2.example.com*:2379,\
          https://*etcd0.example.com*:2379"\
          cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member 8b8904727bf526a5 is healthy: got healthy result from
https://192.168.55.21:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

Or, using etcd v3 API:

```
# ETCDCTL_API=3 etcdctl --cert="/etc/etcd/peer.crt" \
          --key=/etc/etcd/peer.key \
          --cacert="/etc/etcd/ca.crt" \
          --endpoints="https://*master-0.example.com*:2379,\
            https://*master-1.example.com*:2379,\
            https://*master-2.example.com*:2379,\
            https://*etcd0.example.com*:2379"\
            endpoint health
https://master-0.example.com:2379 is healthy: successfully committed
proposal: took = 5.011358ms
https://master-1.example.com:2379 is healthy: successfully committed
proposal: took = 1.305173ms
https://master-2.example.com:2379 is healthy: successfully committed
proposal: took = 1.388772ms
https://etcd0.example.com:2379 is healthy: successfully committed
proposal: took = 1.498829ms
```

Steps to be performed on all OpenShift Container Platform masters

Procedure

1. Modify the master configuration to add the new etcd host to the list of the etcd servers OpenShift Container Platform uses to store the data, located in the **`etcClientInfo`** section of the **`/etc/origin/master/master-config.yaml`** file on every master:

```
etcdClientInfo:
  ca: master.etcd-ca.crt
  certFile: master.etcd-client.crt
  keyFile: master.etcd-client.key
  urls:
    - https://master-0.example.com:2379
```

```
    - https://master-1.example.com:2379
    - https://master-2.example.com:2379
    - https://etcd0.example.com:2379
```

2. Restart the master API service:

   - On every master:

     ```
     # systemctl restart atomic-openshift-master-api
     ```

   - Or, on a single master cluster installation:

     ```
     # systemctl restart atomic-openshift-master
     ```

WARNING

The number of etcd nodes must be odd, so at least two hosts must be added.

1. If you are using Flannel, the **`flanneld`** service configuration located at **`/etc/sysconfig/flanneld`** on every OpenShift Container Platform host must be modified to include the new etcd host:

   ```
   FLANNEL_ETCD_ENDPOINTS=https://master-
   0.example.com:2379,https://master-1.example.com:2379,https://master-
   2.example.com:2379,https://etcd0.example.com:2379
   ```

2. Restart the **`flanneld`** service:

   ```
   # systemctl restart flanneld.service
   ```

5.4.7. Removing an etcd host

If an etcd host fails beyond restoration, remove it from the cluster.

Steps to be performed on all masters hosts

Procedure

1. Remove each other etcd host from the etcd cluster. Run the following command for each etcd node:

   ```
   # etcdctl -C https://<surviving host IP address>:2379 \
     --ca-file=/etc/etcd/ca.crt     \
     --cert-file=/etc/etcd/peer.crt     \
     --key-file=/etc/etcd/peer.key member remove <failed member ID>
   ```

The host to remove.

1. Restart the master API service on every master:

```
# systemctl restart atomic-openshift-master-api
```

Or, if using a single master cluster installation:

```
# systemctl restart atomic-openshift-master
```

Steps to be performed in the current etcd cluster

Procedure

1. Remove the failed host from the cluster:

```
# etcdctl2 cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
failed to check the health of member 8372784203e11288 on
https://192.168.55.21:2379: Get https://192.168.55.21:2379/health:
dial tcp 192.168.55.21:2379: getsockopt: connection refused
member 8372784203e11288 is unreachable: [https://192.168.55.21:2379]
are all unreachable
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy

# etcdctl2 member remove 8372784203e11288 (1)
Removed member 8372784203e11288 from cluster

# etcdctl2 cluster-health
member 5ee217d19001 is healthy: got healthy result from
https://192.168.55.12:2379
member 2a529ba1840722c0 is healthy: got healthy result from
https://192.168.55.8:2379
member ed4f0efd277d7599 is healthy: got healthy result from
https://192.168.55.13:2379
cluster is healthy
```

(1) The **remove** command requires the etcd ID, not the hostname.

2. To ensure the etcd configuration does not use the failed host when the etcd service is restarted, modify the **/etc/etcd/etcd.conf** file on all remaining etcd hosts and remove the failed host in the value for the **ETCD_INITIAL_CLUSTER** variable:

```
# vi /etc/etcd/etcd.conf
```

For example:

```
ETCD_INITIAL_CLUSTER=master-
0.example.com=https://192.168.55.8:2380,master-
1.example.com=https://192.168.55.12:2380,master-
2.example.com=https://192.168.55.13:2380
```

becomes:

```
ETCD_INITIAL_CLUSTER=master-
0.example.com=https://192.168.55.8:2380,master-
1.example.com=https://192.168.55.12:2380
```

NOTE

Restarting the etcd services is not required, because the failed host is removed using **etcdctl**.

3. Modify the Ansible inventory file to reflect the current status of the cluster and to avoid issues when re-running a playbook:

```
[OSEv3:children]
masters
nodes
etcd

... [OUTPUT ABBREVIATED] ...

[etcd]
master-0.example.com
master-1.example.com
```

4. If you are using Flannel, modify the **flanneld** service configuration located at **/etc/sysconfig/flanneld** on every host and remove the etcd host:

```
FLANNEL_ETCD_ENDPOINTS=https://master-
0.example.com:2379,https://master-1.example.com:2379,https://master-
2.example.com:2379
```

5. Restart the **flanneld** service:

```
# systemctl restart flanneld.service
```

CHAPTER 6. PROJECT-LEVEL TASKS

6.1. PROJECT BACKUP

Creating a backup of all relevant data involves exporting all important information, then restoring into a new project.

NOTE

Currently, a OpenShift Container Platform project back up and restore tool is being developed by Red Hat. See the following bug for more information:

- bugzilla 1303205.

6.1.1. Back up a project

Procedure

1. To list all the relevant data to backup:

```
$ oc get all
NAME         TYPE      FROM      LATEST
bc/ruby-ex   Source    Git       1

NAME               TYPE      FROM          STATUS     STARTED
DURATION
builds/ruby-ex-1   Source    Git@c457001   Complete   2 minutes ago
35s

NAME                 DOCKER REPO
TAGS      UPDATED
is/guestbook         10.111.255.221:5000/myproject/guestbook
latest    2 minutes ago
is/hello-openshift   10.111.255.221:5000/myproject/hello-openshift
latest    2 minutes ago
is/ruby-22-centos7   10.111.255.221:5000/myproject/ruby-22-centos7
latest    2 minutes ago
is/ruby-ex           10.111.255.221:5000/myproject/ruby-ex
latest    2 minutes ago

NAME                 REVISION   DESIRED   CURRENT   TRIGGERED BY
dc/guestbook         1          1         1
config,image(guestbook:latest)
dc/hello-openshift   1          1         1
config,image(hello-openshift:latest)
dc/ruby-ex           1          1         1
config,image(ruby-ex:latest)

NAME                   DESIRED   CURRENT   READY     AGE
rc/guestbook-1         1         1         1         2m
rc/hello-openshift-1   1         1         1         2m
rc/ruby-ex-1           1         1         1         2m

NAME                 CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
```

```
svc/guestbook          10.111.105.84    <none>        3000/TCP
2m
svc/hello-openshift    10.111.230.24    <none>
8080/TCP,8888/TCP   2m
svc/ruby-ex            10.111.232.117   <none>        8080/TCP
2m

NAME                           READY     STATUS      RESTARTS   AGE
po/guestbook-1-c010g           1/1       Running     0          2m
po/hello-openshift-1-4zw2q     1/1       Running     0          2m
po/ruby-ex-1-build             0/1       Completed   0          2m
po/ruby-ex-1-rxc74             1/1       Running     0          2m
```

2. Export the project objects into a **project.yaml** file in **yaml** format:

```
$ oc export all -o yaml > project.yaml
```

Or, in **json**:

```
$ oc export all -o json > project.json
```

3. The above creates a **yaml** or **json** file with the project content. This, however, does not export all objects, such as **role bindings**, **secrets**, **service accounts**, or **persistent volume claims**. To export these, run:

```
$ for object in rolebindings serviceaccounts secrets imagestreamtags
podpreset cms egressnetworkpolicies rolebindingrestrictions
limitranges resourcequotas pvcs templates cronjobs statefulsets hpas
deployments replicasets poddisruptionbudget endpoints
do
  oc export $object -o yaml > $object.yaml
done
```

4. Some exported objects can rely on specific metadata or references to unique IDs in the project. This is a limitation on the usability of the recreated objects.
When using **imagestreams**, the **image** parameter of a **deploymentconfig** can point to a specific **sha** checksum of an image in the internal registry that would not exist in a restored environment. For instance, running the sample "ruby-ex" as **oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git** creates an **imagestream ruby-ex** using the internal registry to host the image:

```
$ oc get dc ruby-ex -o jsonpath="
{.spec.template.spec.containers[].image}"
10.111.255.221:5000/myproject/ruby-
ex@sha256:880c720b23c8d15a53b01db52f7abdcbb2280e03f686a5c8edfef1a2a7
b21cee
```

If importing the **deploymentconfig** as it is exported with **oc export** it fails if the image does not exist.

To create those exports, use the **project_export.sh** in the **openshift-ansible-contrib** repository, which creates all the project objects in different files. The script creates a directory on the host running the script with the project name and a **json** file for every object

type in that project.

NOTE

The code in the **openshift-ansible-contrib** repository referenced below is not explicitly supported by Red Hat but the Reference Architecture team performs testing to ensure the code operates as defined and is secure.

The script runs on Linux and requires **jq** and the **oc** commands installed and the system should be logged in to the OpenShift Container Platform environment as a user that can read all the objects in that project.

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-
contrib.git
$ cd openshift-ansible-contrib/reference-
architecture/day2ops/scripts
$ ./project_export.sh <projectname>
```

For example:

```
$ ./project_export.sh myproject
Exporting namespace to project-demo/ns.json
Exporting rolebindings to project-demo/rolebindings.json
Exporting serviceaccounts to project-demo/serviceaccounts.json
Exporting secrets to project-demo/secrets.json
Exporting deploymentconfigs to project-demo/dc_*.json
Patching DC...
Exporting buildconfigs to project-demo/bcs.json
Exporting builds to project-demo/builds.json
Exporting imagestreams to project-demo/iss.json
Exporting imagestreamtags to project-demo/imagestreamtags.json
Exporting replicationcontrollers to project-demo/rcs.json
Exporting services to project-demo/svc_*.json
Exporting pods to project-demo/pods.json
Exporting podpreset to project-demo/podpreset.json
Exporting configmaps to project-demo/cms.json
Exporting egressnetworkpolicies to project-
demo/egressnetworkpolicies.json
Exporting rolebindingrestrictions to project-
demo/rolebindingrestrictions.json
Exporting limitranges to project-demo/limitranges.json
Exporting resourcequotas to project-demo/resourcequotas.json
Exporting pvcs to project-demo/pvcs.json
Exporting routes to project-demo/routes.json
Exporting templates to project-demo/templates.json
Exporting cronjobs to project-demo/cronjobs.json
Exporting statefulsets to project-demo/statefulsets.json
Exporting hpas to project-demo/hpas.json
Exporting deployments to project-demo/deployments.json
Exporting replicasets to project-demo/replicasets.json
Exporting poddisruptionbudget to project-
demo/poddisruptionbudget.json
```

5. Once executed, review the files to verify that the content has been properly exported:

```
$ cd <projectname>
$ ls -1
bcs.json
builds.json
cms.json
cronjobs.json
dc_ruby-ex.json
dc_ruby-ex_patched.json
deployments.json
egressnetworkpolicies.json
endpoint_external-mysql-service.json
hpas.json
imagestreamtags.json
iss.json
limitranges.json
ns.json
poddisruptionbudget.json
podpreset.json
pods.json
pvcs.json
rcs.json
replicasets.json
resourcequotas.json
rolebindingrestrictions.json
rolebindings.json
routes.json
secrets.json
serviceaccounts.json
statefulsets.json
svc_external-mysql-service.json
svc_ruby-ex.json
templates.json
$ less bcs.json
...
```

NOTE

If the original object does not exist, empty files will be created when exporting.

6. If using **imagestreams**, the script modifies the **deploymentconfig** to use the image reference instead the image **sha**, creating a different **json** file than the exported using the **_patched** appendix:

```
$ diff dc_hello-openshift.json dc_hello-openshift_patched.json
45c45
<             "image": "docker.io/openshift/hello-
openshift@sha256:42b59c869471a1b5fdacadf778667cecbaa79e002b7235f8091
540ae612f0e14",
---
>             "image": "hello-openshift:latest",
```

WARNING

The script does not support multiple container pods currently, use it with caution.

6.1.2. Restore project

To restore a project, create the new project, then restore any exported files with **`oc create -f pods.json`**. However, restoring a project from scratch requires a specific order, because some objects are dependent on others. For example, the **`configmaps`** must be created before any **`pods`**.

Procedure

1. If the project has been exported as a single file, it can be imported as:

```
$ oc new-project <projectname>
$ oc create -f project.yaml
$ oc create -f secret.yaml
$ oc create -f serviceaccount.yaml
$ oc create -f pvc.yaml
$ oc create -f rolebindings.yaml
```

WARNING

Some resources can fail to be created (for example, pods and default service accounts).

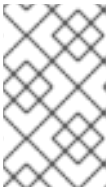
2. If the project was initially exported using the **`project_export.sh`** script, the files are located in the **`projectname`** directory, and can be imported using the same **`project_import.sh`** script that performs the **`oc create`** process in the proper order:

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/openshift/openshift-ansible-
contrib.git
$ cd openshift-ansible-contrib/reference-
architecture/day2ops/scripts
$ ./project_import.sh <projectname_path>
```

For example:

```
$ ls ~/backup/myproject
bcs.json           dc_guestbook_patched.json        dc_ruby-
ex_patched.json  pvcs.json          secrets.json
builds.json        dc_hello-openshift.json          iss.json
rcs.json           serviceaccounts.json
```

```
cms.json           dc_hello-openshift_patched.json  ns.json
rolebindings.json  svcs.json
dc_guestbook.json  dc_ruby-ex.json                  pods.json
routes.json        templates.json

$ ./project_import.sh ~/backup/myproject
namespace "myproject" created
rolebinding "admin" created
rolebinding "system:deployers" created
rolebinding "system:image-builders" created
rolebinding "system:image-pullers" created
secret "builder-dockercfg-mqhs6" created
secret "default-dockercfg-51xb9" created
secret "deployer-dockercfg-6kvz7" created
Error from server (AlreadyExists): error when creating
"myproject//serviceaccounts.json": serviceaccounts "builder" already
exists
Error from server (AlreadyExists): error when creating
"myproject//serviceaccounts.json": serviceaccounts "default" already
exists
Error from server (AlreadyExists): error when creating
"myproject//serviceaccounts.json": serviceaccounts "deployer"
already exists
error: no objects passed to create
service "guestbook" created
service "hello-openshift" created
service "ruby-ex" created
imagestream "guestbook" created
imagestream "hello-openshift" created
imagestream "ruby-22-centos7" created
imagestream "ruby-ex" created
error: no objects passed to create
error: no objects passed to create
buildconfig "ruby-ex" created
build "ruby-ex-1" created
deploymentconfig "guestbook" created
deploymentconfig "hello-openshift" created
deploymentconfig "ruby-ex" created
replicationcontroller "ruby-ex-1" created
Error from server (AlreadyExists): error when creating
"myproject//rcs.json": replicationcontrollers "guestbook-1" already
exists
Error from server (AlreadyExists): error when creating
"myproject//rcs.json": replicationcontrollers "hello-openshift-1"
already exists
pod "guestbook-1-c010g" created
pod "hello-openshift-1-4zw2q" created
pod "ruby-ex-1-rxc74" created
Error from server (AlreadyExists): error when creating
"myproject//pods.json": object is being deleted: pods "ruby-ex-1-
build" already exists
error: no objects passed to create
```



NOTE

AlreadyExists errors can appear, because some objects as **serviceaccounts** and secrets are created automatically when creating the project.

3. If you are using **buildconfigs**, the builds are not triggered automatically and the applications are not executed:

```
$ oc get bc
NAME      TYPE      FROM      LATEST
ruby-ex   Source    Git       1
$ oc get pods
NAME                     READY     STATUS    RESTARTS   AGE
guestbook-1-plnnq        1/1       Running   0          26s
hello-openshift-1-g4g0j  1/1       Running   0          26s
```

To trigger the builds, run the **oc start-build** command:

```
$ for bc in $(oc get bc -o jsonpath="{.items[*].metadata.name}")
do
    oc start-build ${bc}
done
```

The pods will deploy once the build completes.

4. To verify the project was restored:

```
$ oc get all
NAME         TYPE      FROM      LATEST
bc/ruby-ex   Source    Git       2

NAME               TYPE      FROM          STATUS
STARTED              DURATION
builds/ruby-ex-1   Source    Git           Error (BuildPodDeleted)
About a minute ago
builds/ruby-ex-2   Source    Git@c457001   Complete
55 seconds ago       12s

NAME                 DOCKER REPO
TAGS      UPDATED
is/guestbook         10.111.255.221:5000/myproject/guestbook
latest    About a minute ago
is/hello-openshift   10.111.255.221:5000/myproject/hello-openshift
latest    About a minute ago
is/ruby-22-centos7   10.111.255.221:5000/myproject/ruby-22-centos7
latest    About a minute ago
is/ruby-ex           10.111.255.221:5000/myproject/ruby-ex
latest    43 seconds ago

NAME                 REVISION   DESIRED   CURRENT   TRIGGERED BY
dc/guestbook         1          1         1
config,image(guestbook:latest)
dc/hello-openshift   1          1         1
config,image(hello-openshift:latest)
```

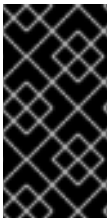
```
dc/ruby-ex           1          1         1
config,image(ruby-ex:latest)

NAME                    DESIRED   CURRENT   READY     AGE
rc/guestbook-1          1         1         1         1m
rc/hello-openshift-1    1         1         1         1m
rc/ruby-ex-1            1         1         1         43s

NAME                   CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
svc/guestbook          10.111.126.115   <none>        3000/TCP
1m
svc/hello-openshift    10.111.23.21     <none>
8080/TCP,8888/TCP   1m
svc/ruby-ex            10.111.162.157   <none>        8080/TCP
1m

NAME                           READY     STATUS      RESTARTS   AGE
po/guestbook-1-plnnq           1/1       Running     0          1m
po/hello-openshift-1-g4g0j     1/1       Running     0          1m
po/ruby-ex-1-h99np             1/1       Running     0          42s
po/ruby-ex-2-build             0/1       Completed   0          55s
```

NOTE

The services and pods IPs are different, because they are assigned dynamically at creation time.

6.2. PVC BACKUP

This topic describes how to synchronize persistent data from inside of a container to a server and then restore the data onto a new persistent volume claim.



IMPORTANT

Depending on the provider that is hosting the OpenShift Container Platform environment, the ability to launch third party snapshot services for backup and restore purposes also exists. As OpenShift Container Platform does not have the ability to launch these services, this guide does not describe these steps.

Consult any product documentation for the correct backup procedures of specific applications. For example, copying the mysql data directory itself would not be a usable backup. Instead, run the specific backup procedures of the associated application and then synchronize any data. This includes using snapshot solutions provided by the OpenShift Container Platform hosting platform.

6.2.1. Backup persistent volume claims

Procedure

1. View the project and pods:

```
$ oc get pods
NAME           READY     STATUS      RESTARTS   AGE
demo-1-build   0/1       Completed   0          2h
```

```
demo-2-fxx6d   1/1       Running     0          1h
```

2. Describe the desired pod to find the volumes currently being used by a persistent volume:

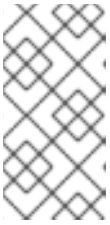
```
$ oc describe pod demo-2-fxx6d
Name:   demo-2-fxx6d
Namespace:  test
Security Policy: restricted
Node:   ip-10-20-6-20.ec2.internal/10.20.6.20
Start Time:  Tue, 05 Dec 2017 12:54:34 -0500
Labels:   app=demo
   deployment=demo-2
   deploymentconfig=demo
Status:   Running
IP:   172.16.12.5
Controllers:  ReplicationController/demo-2
Containers:
  demo:
    Container ID:
docker://201f3e55b373641eb36945d723e1e212ecab847311109b5cee1fd010942
4217a
    Image:  docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Image ID:  docker-pullable://docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Port:  8080/TCP
    State:  Running
      Started:  Tue, 05 Dec 2017 12:54:52 -0500
    Ready:  True
    Restart Count: 0
    Volume Mounts:
      */opt/app-root/src/uploaded from persistent-volume (rw)*
      /var/run/secrets/kubernetes.io/serviceaccount from default-
token-8mmrk (ro)
    Environment Variables: <none>
...omitted...
```

The above shows that the persistent data is currently located in the **/opt/app-root/src/uploaded** directory.

3. Copy the data locally:

```
$ oc rsync demo-2-fxx6d:/opt/app-root/src/uploaded ./demo-app
receiving incremental file list
uploaded/
uploaded/ocp_sop.txt
uploaded/lost+found/

sent 38 bytes  received 190 bytes  152.00 bytes/sec
total size is 32  speedup is 0.14
```

The **ocp_sop.txt** file has been pulled down to the local system to be backed up by backup software or to another backup mechanism.



NOTE

The steps above can also be used in the event that a pod starts without needing to use a **pvc**, but then decides a **pvc** is necessary. This would allow for the data to be preserved and the restore procedures to be used to populate the new storage.

6.2.2. Restore persistent volume claims

This topic describes two methods for restoring data. The first involves deleting the file, then placing the file back in the expected location. The second example shows migrating persistent volume claims. The migration would occur in the event that the storage needs to be moved or in a disaster scenario when the backend storage no longer exists.

Check with the restore procedures for the specific application on any steps required to restore data to the application.

6.2.2.1. Restoring files to an existing PVC

Procedure

1. Delete the file:

```
$ oc rsh demo-2-fxx6d
sh-4.2$ ls */opt/app-root/src/uploaded/*
lost+found  ocp_sop.txt
sh-4.2$ *rm -rf /opt/app-root/src/uploaded/ocp_sop.txt*
sh-4.2$ *ls /opt/app-root/src/uploaded/*
lost+found
```

2. Replace the file from the server containing the rsync backup of the files that were in the pvc:

```
$ oc rsync uploaded demo-2-fxx6d:/opt/app-root/src/
```

3. Validate that the file is back on the pod by using **oc rsh** to connect to the pod and view the contents of the directory:

```
$ oc rsh demo-2-fxx6d
sh-4.2$ *ls /opt/app-root/src/uploaded/*
lost+found  ocp_sop.txt
```

6.2.2.2. Restoring data to a new PVC

The following steps assume that a new **pvc** has been created.

Procedure

1. Overwrite the currently defined **claim-name**:

```
$ oc volume dc/demo --add --name=persistent-volume \
  --type=persistentVolumeClaim --claim-name=filestore \ --mount-
path=/opt/app-root/src/uploaded --overwrite
```

2. Validate that the pod is using the new PVC:

```
$ oc describe dc/demo
Name:  demo
Namespace: test
Created: 3 hours ago
Labels:  app=demo
Annotations: openshift.io/generated-by=OpenShiftNewApp
Latest Version: 3
Selector: app=demo,deploymentconfig=demo
Replicas: 1
Triggers: Config, Image(demo@latest, auto=true)
Strategy: Rolling
Template:
  Labels: app=demo
  deploymentconfig=demo
  Annotations: openshift.io/container.demo.image.entrypoint=
["container-entrypoint","/bin/sh","-c","$STI_SCRIPTS_PATH/usage"]
  openshift.io/generated-by=OpenShiftNewApp
  Containers:
   demo:
    Image: docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20
dc9ff6f350436f935968b0c83fcb98a7a8c381a
    Port: 8080/TCP
    Volume Mounts:
      /opt/app-root/src/uploaded from persistent-volume (rw)
    Environment Variables: <none>
  Volumes:
   persistent-volume:
    Type: PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
    *ClaimName: filestore*
    ReadOnly: false
...omitted...
```

3. Now that the new **pvc** is being used by the deployment configuration, use **oc rsync** to place the files onto the new **pvc**:

```
$ oc rsync uploaded demo-3-2b8gs:/opt/app-root/src/
sending incremental file list
uploaded/
uploaded/ocp_sop.txt
uploaded/lost+found/

sent 181 bytes  received 39 bytes  146.67 bytes/sec
total size is 32  speedup is 0.15
```

4. Validate that the file is back on the pod by using **oc rsh** to connect to the pod and view the contents of the directory.

```
$ oc rsh demo-3-2b8gs
sh-4.2$ ls /opt/app-root/src/uploaded/
lost+found  ocp_sop.txt
```

6.3. PRUNING IMAGES AND CONTAINERS

See the Pruning Resources topic for information about pruning collected data and older versions of objects.

CHAPTER 7. DOCKER TASKS

OpenShift Container Platform uses Docker to run applications in pods that are composed by any number of containers.

As a cluster administrator, sometimes Docker requires some extra configuration in order to efficiently run elements of the OpenShift Container Platform installation.

7.1. INCREASING DOCKER STORAGE

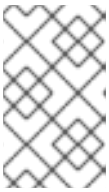
Increasing the amount of storage available ensures continued deployment without any outages. To do so, a free partition must be made available that contains an appropriate amount of free capacity.

7.1.1. Evacuating the node

Procedure

1. From a master instance, or as a cluster administrator, allow the evacuation of any pod from the node and disable scheduling of other pods on that node:

```
$ NODE=ose-app-node01.example.com
$ oc adm manage-node ${NODE} --schedulable=false
NAME                          STATUS                     AGE
VERSION
ose-app-node01.example.com   Ready,SchedulingDisabled   20m
v1.6.1+5115d708d7

$ oc adm drain ${NODE} --ignore-daemonsets
node "ose-app-node01.example.com" already cordoned
pod "perl-1-build" evicted
pod "perl-1-3lnsh" evicted
pod "perl-1-9jzd8" evicted
node "ose-app-node01.example.com" drained
```



NOTE

If there are containers running with local volumes that will not migrate, run the following command: **`oc adm drain ${NODE} --ignore-daemonsets --delete-local-data`**.

2. List the pods on the node to verify that they have been removed:

```
$ oc adm manage-node ${NODE} --list-pods

Listing matched pods on node: ose-app-node01.example.com

NAME      READY     STATUS    RESTARTS   AGE
```

For more information on evacuating and draining pods or nodes, see Node maintenance.

7.1.2. Increasing storage

You can increase Docker storage in two ways: attaching a new disk, or extending the existing disk.

Increasing storage with a new disk

Prerequisites

- A new disk must be available to the existing instance that requires more storage. In the following steps, the original disk is labeled **/dev/xvdb**, and the new disk is labeled **/dev/xvdd**, as shown in the ***/etc/sysconfig/docker-storage-setup*** file:

```
# vi /etc/sysconfig/docker-storage-setup
DEVS="/dev/xvdb /dev/xvdd"
```

NOTE

The process may differ depending on the underlying OpenShift Container Platform infrastructure.

Procedure

1. Stop the **docker** and **atomic-openshift-node** services:

```
# systemctl stop docker atomic-openshift-node
```

2. Run the **docker-storage-setup** command to extend the volume groups and logical volumes associated with container storage:

```
# docker-storage-setup
INFO: Volume group backing root filesystem could not be determined
INFO: Device /dev/xvdb is already partitioned and is part of volume
group docker_vol
INFO: Device node /dev/xvdd1 exists.
  Physical volume "/dev/xvdd1" successfully created.
  Volume group "docker_vol" successfully extended
```

3. Start the Docker services:

```
# systemctl start docker
# vgs
  VG         #PV #LV #SN Attr   VSize  VFree
  docker_vol   2   1   0 wz--n- 64.99g <55.00g
```

4. A benefit in adding a disk compared to creating a new volume group and re-running **docker-storage-setup** is that the images that were used on the system still exist after the new storage has been added:

```
# docker images
REPOSITORY                                                  TAG
IMAGE ID            CREATED             SIZE
docker-registry.default.svc:5000/tet/perl                   latest
8b0b0106fb5e        13 minutes ago      627.4 MB
registry.access.redhat.com/rhscl/perl-524-rhel7             <none>
912b01ac7570        6 days ago          559.5 MB
registry.access.redhat.com/openshift3/ose-deployer
v3.6.173.0.21       89fd398a337d        5 weeks ago         970.2
MB
```

```
registry.access.redhat.com/openshift3/ose-sti-builder
v3.6.173.0.21       99ab8895d88a        5 weeks ago         970.2
MB
registry.access.redhat.com/openshift3/ose-pod
v3.6.173.0.21       63accd48a0d7        5 weeks ago         208.6
MB
```

5. With the increase in storage capacity, enable the node to be schedulable in order to accept new incoming pods.
   As a cluster administrator, run the following from a master instance:

```
$ oc adm manage-node ${NODE} --schedulable=true

ose-master01.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-master02.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-master03.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-infra-node01.example.com   Ready                      24m
v1.6.1+5115d708d7
ose-infra-node02.example.com   Ready                      24m
v1.6.1+5115d708d7
ose-infra-node03.example.com   Ready                      24m
v1.6.1+5115d708d7
ose-app-node01.example.com   Ready                        24m
v1.6.1+5115d708d7
ose-app-node02.example.com   Ready                        24m
v1.6.1+5115d708d7
```

Increasing storage with a new disk

1. Evacuate the node following the previous steps.
2. Stop the **docker** and **atomic-openshift-node** services:

```
# systemctl stop docker atomic-openshift-node
```

3. Resize the existing disk as desired. This can can depend on your environment:
   - If you are using LVM (Logical Volume Manager):
     - Remove the logical volume:

```
# lvremove /dev/docker_vg/docker/lv
```

     - Remove the Docker volume group:

```
# vgremove docker_vg
```

     - Remove the physical volume:

```
# pvremove /dev/<my_previous_disk_device>
```

- If you are using a cloud provider, you can detach the disk, destroy the disk, then create a new bigger disk, and attach it to the instance.
- For a non-cloud environment, the disk and file system can be resized. See the following solution for more information:
  - https://access.redhat.com/solutions/199573

4. Verify that the */etc/sysconfig/container-storage-setup* file is correctly configured for the new disk by checking the device name, size, etc.
5. Run **`docker-storage-setup`** to reconfigure the new disk:

```
# docker-storage-setup
INFO: Volume group backing root filesystem could not be determined
INFO: Device /dev/xvdb is already partitioned and is part of volume
group docker_vol
INFO: Device node /dev/xvdd1 exists.
  Physical volume "/dev/xvdd1" successfully created.
  Volume group "docker_vol" successfully extended
```

6. Start the Docker services:

```
# systemctl start docker
# vgs
  VG         #PV #LV #SN Attr   VSize  VFree
  docker_vol   2   1   0 wz--n- 64.99g <55.00g
```

7. Start the **`atomic-openshift-node`** service:

```
# systemctl start atomic-openshift-node
```

7.1.3. Changing the storage backend

With the advancements of services and file systems, changes in a storage backend may be necessary to take advantage of new features. The following steps provide an example of changing a device mapper backend to an **`overlay2`** storage backend. **`overlay2`** offers increased speed and density over traditional device mapper.

7.1.3.1. Evacuating the node

1. From a master instance, or as a cluster administrator, allow the evacuation of any pod from the node and disable scheduling of other pods on that node:

```
$ NODE=ose-app-node01.example.com
$ oc adm manage-node ${NODE} --schedulable=false
NAME                          STATUS                     AGE
VERSION
ose-app-node01.example.com    Ready,SchedulingDisabled   20m
v1.6.1+5115d708d7

$ oc adm drain ${NODE} --ignore-daemonsets
node "ose-app-node01.example.com" already cordoned
pod "perl-1-build" evicted
```

```
pod "perl-1-3lnsh" evicted
pod "perl-1-9jzd8" evicted
node "ose-app-node01.example.com" drained
```

NOTE

If there are containers running with local volumes that will not migrate, run the following command: **`oc adm drain ${NODE} --ignore-daemonsets --delete-local-data`**

2. List the pods on the node to verify that they have been removed:

```
$ oc adm manage-node ${NODE} --list-pods

Listing matched pods on node: ose-app-node01.example.com

NAME      READY     STATUS    RESTARTS   AGE
```

For more information on evacuating and draining pods or nodes, see Node maintenance.

3. With no containers currently running on the instance, stop the **`docker`** and **`atomic-openshift-node service`** services:

```
# systemctl stop docker atomic-openshift-node
```

4. Verify the name of the volume group, logical volume name, and physical volume name:

```
# vgs
  VG         #PV #LV #SN Attr   VSize   VFree
  docker_vol   1   1   0 wz--n- <25.00g 15.00g

# lvs
LV       VG         Attr       LSize   Pool Origin Data%  Meta%
Move Log Cpy%Sync Convert
 dockerlv docker_vol -wi-ao---- <10.00g

# lvremove /dev/docker_vol/docker-pool  -y
# vgremove docker_vol -y
# pvs
  PV         VG         Fmt  Attr PSize   PFree
  /dev/xvdb1 docker_vol lvm2 a--  <25.00g 15.00g

# pvremove /dev/xvdb1 -y
# rm -Rf /var/lib/docker/*
# rm -f /etc/sysconfig/docker-storage
```

5. Modify the **`docker-storage-setup`** file to specify the **`STORAGE_DRIVER`**.

NOTE

When a system is upgraded from Red Hat Enterprise Linux version 7.3 to 7.4, the **docker** service attempts to use **/var** with the **STORAGE_DRIVER** of extfs. The use of extfs as the **STORAGE_DRIVER** causes errors. See the following bug for more info regarding the error:

- Bugzilla ID: 1490910

```
DEVS=/dev/xvdb
VG=docker_vol
DATA_SIZE=95%VG
STORAGE_DRIVER=overlay2
CONTAINER_ROOT_LV_NAME=dockerlv
CONTAINER_ROOT_LV_MOUNT_PATH=/var/lib/docker
CONTAINER_ROOT_LV_SIZE=100%FREE
```

6. Set up the storage:

```
# docker-storage-setup
```

7. Start the **docker** and **atomic-openshift-node** services:

```
# systemctl start docker atomic-openshift-node
```

8. With the storage modified to use **overlay2**, enable the node to be schedulable in order to accept new incoming pods.
From a master instance, or as a cluster administrator:

```
$ oc adm manage-node ${NODE} --schedulable=true

ose-master01.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-master02.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-master03.example.com   Ready,SchedulingDisabled   24m
v1.6.1+5115d708d7
ose-infra-node01.example.com   Ready                     24m
v1.6.1+5115d708d7
ose-infra-node02.example.com   Ready                     24m
v1.6.1+5115d708d7
ose-infra-node03.example.com   Ready                     24m
v1.6.1+5115d708d7
ose-app-node01.example.com   Ready                       24m
v1.6.1+5115d708d7
ose-app-node02.example.com   Ready                       24m
v1.6.1+5115d708d7
```

7.2. MANAGING DOCKER CERTIFICATES

An OpenShift Container Platform internal registry is created as a pod. However, containers may be pulled from external registries if desired. By default, registries listen on TCP port 5000. Registries provide the option of securing exposed images via TLS or running a registry without encrypting traffic.

WARNING

Docker interprets `.crt` files as CA certificates and `.cert` files as client certificates. Any CA extensions **must** be `.crt`.

7.2.1. Installing a certificate authority certificate for external registries

In order to use OpenShift Container Platform with an external registry, the registry certificate authority (CA) certificate must be trusted for all the nodes that can pull images from the registry.

NOTE

Depending on the Docker version, the process to trust a Docker registry varies. The latest versions of Docker's root certificate authorities are merged with system defaults. Prior to **docker** version 1.13, the system default certificate is used only when no other custom root certificates exist.

Procedure

1. Copy the CA certificate to **/etc/pki/ca-trust/source/anchors/**:

```
$ sudo cp myregistry.example.com.crt /etc/pki/ca-
trust/source/anchors/
```

2. Extract and add the CA certificate to the list of trusted certificates authorities:

```
$ sudo update-ca-trust extract
```

3. Verify the SSL certificate using the **openssl** command:

```
$ openssl verify myregistry.example.com.crt
myregistry.example.com.crt: OK
```

4. Once the certificate is in place and the trust is updated, restart the **docker** service to ensure the new certificates are properly set:

```
$ sudo systemctl restart docker.service
```

For Docker versions prior to 1.13, perform the following additional steps for trusting certificates of authority:

1. On every node create a new directory in **/etc/docker/certs.d** where the name of the directory is the host name of the Docker registry:

```
$ sudo mkdir -p /etc/docker/certs.d/myregistry.example.com
```

NOTE

The port number is not required unless the Docker registry cannot be accessed without a port number. Addressing the port to the original Docker registry is as follows: **`myregistry.example.com:port`**

2. Accessing the Docker registry via IP address requires the creation of a new directory within **`/etc/docker/certs.d`** on every node where the name of the directory is the IP of the Docker registry:

```
$ sudo mkdir -p /etc/docker/certs.d/10.10.10.10
```

3. Copy the CA certificate to the newly created Docker directories from the previous steps:

```
$ sudo cp myregistry.example.com.crt \
  /etc/docker/certs.d/myregistry.example.com/ca.crt

$ sudo cp myregistry.example.com.crt
/etc/docker/certs.d/10.10.10.10/ca.crt
```

4. Once the certificates have been copied, restart the **`docker`** service to ensure the new certificates are used:

```
$ sudo systemctl restart docker.service
```

7.2.2. Docker certificates backup

When performing a node host backup, ensure to include the certificates for external registries.

Procedure

1. If using **`/etc/docker/certs.d`**, copy all the certificates included in the directory and store the files:

```
$ sudo tar -czvf docker-registry-certs-$(hostname)-$(date
+%Y%m%d).tar.gz /etc/docker/certs.d/
```

2. If using a system trust, store the certificates prior to adding them within the system trust. Once the store is complete, extract the certificate for restoration using the **`trust`** command. Identify the system trust CAs and note the **`pkcs11`** ID:

```
$ trust list
...[OUTPUT OMMITED]...
pkcs11:id=%a5%b3%e1%2b%2b%49%b6%d7%73%a1%aa%94%f5%01%e7%73%65%4c%ac%
50;type=cert
    type: certificate
    label: MyCA
    trust: anchor
    category: authority
...[OUTPUT OMMITED]...
```

3. Extract the certificate in **`pem`** format and provide it a name. For example, **`myca.crt`**.

```
$ trust extract --format=pem-bundle \
 --
filter="%a5%b3%e1%2b%2b%49%b6%d7%73%a1%aa%94%f5%01%e7%73%65%4c%ac%50
;type=cert" myca.crt
```

4. Verify the certificate has been properly extracted via **openssl**:

```
$ openssl verify myca.crt
```

5. Repeat the procedure for all the required certificates and store the files in a remote location.

7.2.3. Docker certificates restore

In the event of the deletion or corruption of the Docker certificates for the external registries, the restore mechanism uses the same steps as the installation method using the files from the backups performed previously.

7.3. MANAGING DOCKER REGISTRIES

You can configure OpenShift Container Platform to use external **docker** registries to pull images. However, you can use configuration files to allow or deny certain images or registries.

If the external registry is exposed using certificates for the network traffic, it can be named as a secure registry. Otherwise, traffic between the registry and host is plain text and not encrypted, meaning it is an insecure registry.

7.3.1. Docker search external registries

By default, the **docker** daemon has the ability to pull images from any registry, but the search operation is performed against **docker.io/** and **registry.access.redhat.com**. The daemon can be be configured to search images from other registries using the **--add-registry** option with the **docker** daemon.

NOTE

The ability to search images from the Red Hat Registry **registry.access.redhat.com** exists by default in the Red Hat Enterprise Linux **docker** package.

Procedure

1. To allow users to search for images using **docker search** with other registries, add those registries to the **/etc/containers/registries.conf** file under the **registries** parameter:

```
registries:
  - registry.access.redhat.com
  - my.registry.example.com
```

Prior to OpenShift Container Platform version 3.6, this was accomplished using **/etc/sysconfig/docker** with the following options:

```
ADD_REGISTRY="--add-registry=registry.access.redhat.com --add-
registry=my.registry.example.com"
```

The first registry added is the first registry searched.

2. Restart the **docker** daemon to allow for **my.registry.example.com** to be used:

```
$ sudo systemctl restart docker.service
```

Restarting the **docker** daemon causes the **docker** containers to restart.

3. Using the Ansible installer, this can be configured using the **openshift_docker_additional_registries** variable in the Ansible hosts file:

```
openshift_docker_additional_registries=registry.access.redhat.com,my
.registry.example.com
```

7.3.2. Docker external registries whitelist and blacklist

Docker can be configured to block operations from external registries by configuring the **registries** and **block_registries** flags for the **docker** daemon.

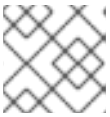
Procedure

1. Add the allowed registries to the **/etc/containers/registries.conf** file with the **registries** flag:

```
registries:
  - registry.access.redhat.com
  - my.registry.example.com
```

Prior to 3.6, the **/etc/sysconfig/docker** file is modified instead:

```
ADD_REGISTRY="--add-registry=registry.access.redhat.com --add-
registry=my.registry.example.com"
```



NOTE

The **docker.io** registry can be added using the same method.

2. Block the rest of the registries:

```
block_registries:
   - all
```

3. Block the rest of the registries in older versions:

```
BLOCK_REGISTRY='--block-registry=all'
```

4. Restart the **docker** daemon:

```
$ sudo systemctl restart docker.service
```

Restarting the **docker** daemon causes the **docker** containers to restart.

5. In this example, the **docker.io** registry has been blacklisted, so any operation regarding that registry fails:

```
$ sudo docker pull hello-world
Using default tag: latest
Trying to pull repository registry.access.redhat.com/hello-world ...
Trying to pull repository my.registry.example.com/hello-world ...
Trying to pull repository registry.access.redhat.com/hello-world ...
unknown: Not Found
$ sudo docker pull docker.io/hello-world
Using default tag: latest
Trying to pull repository docker.io/library/hello-world ...
All endpoints blocked.
```

Add **docker.io** back to the **registries** variable by modifying the file again and restarting the service.

```
registries:
  - registry.access.redhat.com
  - my.registry.example.com
  - docker.io
block_registries:
  - all
```

or

```
ADD_REGISTRY="--add-registry=registry.access.redhat.com --add-
registry=my.registry.example.com --add-registry=docker.io"
BLOCK_REGISTRY='--block-registry=all'
```

6. Restart the Docker service:

```
$ sudo systemctl restart docker
```

7. To verify that the image is now available to be pulled:

```
$ sudo docker pull docker.io/hello-world
Using default tag: latest
Trying to pull repository docker.io/library/hello-world ...
latest: Pulling from docker.io/library/hello-world

9a0669468bf7: Pull complete
Digest:
sha256:0e06ef5e1945a718b02a8c319e15bae44f47039005530bc617a5d071190ed
3fc
```

8. If using an external registry is required, for example to modify the **docker** daemon configuration file in all the node hosts that require to use that registry, create a blacklist on those nodes to avoid malicious containers from being executed.

Using the Ansible installer, this can be configured using the **openshift_docker_additional_registries** and **openshift_docker_blocked_registries** variables in the Ansible hosts file:

```
openshift_docker_additional_registries=registry.access.redhat.com,my
.registry.example.com
openshift_docker_blocked_registries=all
```

7.3.3. Secure registries

In order to be able to pull images from an external registry, it is required to trust the registry certificates, otherwise the pull image operation fails.

In order to do so, see the Installing a Certificate Authority Certificate for External Registries section.

If using a whitelist, the external registries should be added to the **registries** variable, as explained above.

7.3.4. Insecure registries

External registries that use non-trusted certificates, or without certificates at all, should be avoided.

However, any insecure registries should be added using the **--insecure-registry** option to allow for the **docker** daemon to pull images from the repository. This is the same as the **--add-registry** option, but the **docker** operation is not verified.

The registry should be added using both options to enable search, and, if there is a blacklist, to perform other operations, such as pulling images.

For testing purposes, an example is shown on how to add a **localhost** insecure registry.

Procedure

1. Modify **/etc/containers/registries.conf** configuration file to add the localhost insecure registry:

```
registries:
  - registry.access.redhat.com
  - my.registry.example.com
  - docker.io
insecure_registries:
  - localhost:5000
block_registries:
  - all
```

Prior to 3.6, modify the **/etc/sysconfig/docker** configuration file to add the localhost:

```
ADD_REGISTRY="--add-registry=registry.access.redhat.com --add-
registry=my.registry.example.com --add-registry=docker.io --add-
registry=localhost:5000"
INSECURE_REGISTRY="--insecure-registry=localhost:5000"
BLOCK_REGISTRY='--block-registry=all'
```

2. Restart the **docker** daemon to use the registry:

```
$ sudo systemctl restart docker.service
```

Restarting the **docker** daemon causes the **docker** containers to be restarted.

3. Run a Docker registry pod at **localhost**:

```
$ sudo docker run -p 5000:5000 registry:2
```

4. Pull an image:

```
$ sudo docker pull openshift/hello-openshift
```

5. Tag the image:

```
$ sudo docker tag docker.io/openshift/hello-openshift:latest
localhost:5000/hello-openshift-local:latest
```

6. Push the image to the local registry:

```
$ sudo docker push localhost:5000/hello-openshift-local:latest
```

7. Using the Ansible installer, this can be configured using the **openshift_docker_additional_registries**, **openshift_docker_blocked_registries**, and **openshift_docker_insecure_registries** variables in the **Ansible** hosts file:

```
openshift_docker_additional_registries=registry.access.redhat.com,my
.registry.example.com,localhost:5000
openshift_docker_insecure_registries=localhost:5000
openshift_docker_blocked_registries=all
```

7.3.5. Authenticated registries

Using authenticated registries with **docker** requires the **docker** daemon to log in to the registry. With OpenShift Container Platform, a different set of steps must be performed, because the users can not run **docker login** commands on the host. Authenticated registries can be used to limit the images users can pull or who can access the external registries.

If an external **docker** registry requires authentication, create a special secret in the project that uses that registry and then use that secret to perform the **docker** operations.

Procedure

1. Create a **dockercfg** secret in the project where the user is going to log in to the **docker** registry:

```
$ oc project <my_project>
$ oc secrets new-dockercfg <my_registry> --docker-server=
<my.registry.example.com> --docker-username=<username> --docker-
password=<my_password> --docker-email=<me@example.com>
```

2. If a **.dockercfg** file exists, create the secret using the **oc** command:

```
$ oc secrets new <my_registry> .dockercfg=<.dockercfg>
```

3. Populate the **$HOME/.docker/config.json** file:

```
$ oc secrets new <my_registry> .dockerconfigjson=
<.docker/config.json>
```

4. Use the **dockercfg** secret to pull images from the authenticated registry by linking the secret to the service account performing the pull operations. The default service account to pull images is named **default**:

```
$ oc secrets link default <my_registry> --for=pull
```

5. For pushing images using the S2I feature, the **dockercfg** secret is mounted in the S2I pod, so it needs to be linked to the proper service account that performs the build. The default service account used to build images is named **builder**.

```
$ oc secrets link builder <my_registry>
```

6. In the **buildconfig**, the secret should be specified for push or pull operations:

```
"type": "Source",
"sourceStrategy": {
    "from": {
        "kind": "DockerImage",
        "name": "*my.registry.example.com*/myproject/myimage:stable"
    },
    "pullSecret": {
        "name": "*mydockerregistry*"
    },
...[OUTPUT ABBREVIATED]...
"output": {
    "to": {
        "kind": "DockerImage",
        "name": "*my.registry.example.com*/myproject/myimage:latest"
    },
    "pushSecret": {
        "name": "*mydockerregistry*"
    },
...[OUTPUT ABBREVIATED]...
```

7. If the external registry delegates authentication to external services, create both **dockercfg** secrets: the registry one using the registry URL and the external authentication system using its own URL. Both secrets should be added to the service accounts.

```
$ oc project <my_project>
$ oc secrets new-dockercfg <my_registry> --docker-server=*
<my_registry_example.com> --docker-username=<username> --docker-
password=<my_password> --docker-email=<me@example.com>
$ oc secrets new-dockercfg <my_docker_registry_ext_auth> --docker-
server=<my.authsystem.example.com> --docker-username=<username> --
docker-password=<my_password> --docker-email=<me@example.com>
$ oc secrets link default <my_registry> --for=pull
```

```
$ oc secrets link default <my_docker_registry_ext_auth> --for=pull
$ oc secrets link builder <my_registry>
$ oc secrets link builder <my_docker_registry_ext_auth>
```

7.3.6. ImagePolicy admission plug-in

An admission control plug-in intercepts requests to the API, and performs checks depending on the configured rules and allows or denies certain actions based on those rules. OpenShift Container Platform can limit the allowed images running in the environment using the **ImagePolicy** admission plug-in where it can control:

- The source of images: which registries can be used to pull images
- Image resolution: force pods to run with immutable digests to ensure the image does not change due to a re-tag
- Container image label restrictions: force an image to have or not have particular labels
- Image annotation restrictions: force an image in the integrated container registry to have or not have particular annotations

WARNING

ImagePolicy admission plug-in is currently considered beta.

Procedure

1. If the **ImagePolicy** plug-in is enabled, it needs to be modified to allow the external registries to be used by modifying the **/etc/origin/master/master-config.yaml** file on every master node:

```
admissionConfig:
  pluginConfig:
    openshift.io/ImagePolicy:
      configuration:
        kind: ImagePolicyConfig
        apiVersion: v1
        executionRules:
        - name: allow-images-from-other-registries
          onResources:
          - resource: pods
          - resource: builds
          matchRegistries:
          - docker.io
          - <my.registry.example.com>
          - registry.access.redhat.com
```

NOTE

Enabling **ImagePolicy** requires users to specify the registry when deploying an application like **oc new-app docker.io/kubernetes/guestbook** instead **oc new-app kubernetes/guestbook**, otherwise it fails.

2. To enable the admission plug-ins at installation time, the **openshift_master_admission_plugin_config** variable can be used with a **json** formatted string including all the **pluginConfig** configuration:

```
openshift_master_admission_plugin_config=
{"openshift.io/ImagePolicy":{"configuration":
{"kind":"ImagePolicyConfig","apiVersion":"v1","executionRules":
[{"name":"allow-images-from-other-registries","onResources":
[{"resource":"pods"},{"resource":"builds"}],"matchRegistries":
["docker.io","*my.registry.example.com*","registry.access.redhat.com
"]}]}}}
```

WARNING

There is a current issue to be fixed in OpenShift Container Platform 3.6.1 where **ImagePolicy** pods can not be deployed using default templates, and give the following error message **Failed create | Error creating: Pod "" is invalid: spec.containers[0].\image: Forbidden: this image is prohibited by policy.**

See the Image Policy is not working as expected Red Hat Knowledgebase article for a workaround.

7.3.7. Import images from external registries

Application developers can import images to create **imagestreams** using the **oc import-image** command, and OpenShift Container Platform can be configured to allow or deny image imports from external registries.

Procedure

1. To configure the allowed registries where users can import images, add the following to the **/etc/origin/master/master-config.yaml** file:

```
imagePolicyConfig:
  allowedRegistriesForImport:
  - domainName: docker.io
  - domainName: '\*.docker.io'
  - domainName: '*.redhat.com'
  - domainName: 'my.registry.example.com'
```

2. To import images from an external authenticated registry, create a secret within the desired project.

3. Even if not recommended, if the external authenticated registry is insecure or the certificates can not be trusted, the **`oc import-image`** command can be used with the **`--insecure=true`** option.
   If the external authenticated registry is secure, the registry certificate should be trusted in the master hosts as they run the registry import controller as:

   Copy the certificate in the **`/etc/pki/ca-trust/source/anchors/`**:

   ```
   $ sudo cp <my.registry.example.com.crt> /etc/pki/ca-
   trust/source/anchors/<my.registry.example.com.crt>
   ```

4. Run **`update-ca-trust`** command:

   ```
   $ sudo update-ca-trust
   ```

5. Restart the master services on all the master hosts:

   ```
   $ sudo systemctl restart atomic-openshift-master-api
   $ sudo systemctl restart atomic-openshift-master-controllers
   ```

6. The certificate for the external registry should be trusted in the OpenShift Container Platform registry:

   ```
   $ for i in pem openssl java; do
     oc create configmap ca-trust-extracted-${i} --from-file
   /etc/pki/ca-trust/extracted/${i}
     oc set volume dc/docker-registry --add -m /etc/pki/ca-
   trust/extracted/${i} --configmap-name=ca-trust-extracted-${i} --name
   ca-trust-extracted-${i}
   done
   ```

   WARNING

   There is no official procedure currently for adding the certificate to the registry pod, but the above workaround can be used.

   This workaround creates **`configmaps`** with all the trusted certificates from the system running those commands, so the recommendation is to run it from a clean system where just the required certificates are trusted.

7. Alternatively, modify the registry image in order to trust the proper certificates rebuilding the image using a **`Dockerfile`** as:

   ```
   FROM registry.access.redhat.com/openshift3/ose-docker-registry:v3.6
   ADD <my.registry.example.com.crt> /etc/pki/ca-trust/source/anchors/
   USER 0
   RUN update-ca-trust extract
   USER 1001
   ```

8. Rebuild the image, push it to a **docker** registry, and use that image as **spec.template.spec.containers["name":"registry"].image** in the registry **deploymentconfig**:

```
$ oc patch dc docker-registry -p '{"spec":{"template":{"spec":
{"containers":
[{"name":"registry","image":"*myregistry.example.com/openshift3/ose-
docker-registry:latest*"}]}}}}'
```

NOTE

To add the **imagePolicyConfig** configuration at installation, the **openshift_master_image_policy_config** variable can be used with a **json** formatted string including all the **imagePolicyConfig** configuration, like:

```
openshift_master_image_policy_config={"imagePolicyConfig":
{"allowedRegistriesForImport":[{"domainName":"docker.io"},
{"domainName":"\*.docker.io"},{"domainName":"*.redhat.com"},
{"domainName":"*my.registry.example.com*"}]}}
```

For more information about the **ImagePolicy**, see the **ImagePolicy** admission plug-in section.

7.3.8. OpenShift Container Platform registry integration

You can install OpenShift Container Platform as a stand-alone container registry to provide only the registry capabilities, but with the advantages of running in an OpenShift Container Platform platform.

For more information about the OpenShift Container Platform registry, see Installing a Stand-alone Deployment of OpenShift Container Registry.

To integrate the OpenShift Container Platform registry, all previous sections apply. From the OpenShift Container Platform point of view, it is treated as an external registry, but there are some extra tasks that need to be performed, because it is a multi-tenant registry and the authorization model from OpenShift Container Platform applies so when a new project is created, the registry does not create a project within its environment as it is independent.

7.3.8.1. Connect the registry project with the cluster

As the registry is a full OpenShift Container Platform environment with a registry pod and a web interface, the process to create a new project in the registry is performed using the **oc new-project** or **oc create** command line or via the web interface.

Once the project has been created, the usual service accounts (**builder**, **default**, and **deployer**) are created automatically, as well as the project administrator user is granted permissions. Different users can be authorized to push/pull images as well as "anonymous" users.

There can be several use cases, such as allowing all the users to pull images from this new project within the registry, but if you want to have a 1:1 project relationship between OpenShift Container Platform and the registry, where the users can push and pull images from that specific project, some steps are required.

WARNING

The registry web console shows a token to be used for pull/push operations, but the token showed there is a session token, so it expires. Creating a service account with specific permissions allows the administrator to limit the permissions for the service account, so that, for example, different service accounts can be used for push or pull images. Then, a user does not have to configure for token expiration, secret recreation, and other tasks, as the service account tokens will not expire.

Procedure

1. Create a new project:

```
$ oc new-project <my_project>
```

2. Create a registry project:

```
$ oc new-project <registry_project>
```

3. Create a service account in the registry project:

```
$ oc create serviceaccount <my_serviceaccount> -n <registry_project>
```

4. Give permissions to push and pull images using the **`registry-editor`** role:

```
$ oc adm policy add-role-to-user registry-editor -z
<my_serviceaccount> -n <registry_project>
```

   If only pull permissions are required, the **`registry-viewer`** role can be used.

5. Get the service account token:

```
$ TOKEN=$(oc sa get-token <my_serviceaccount> -n <registry_project>)
```

6. Use the token as the password to create a **`dockercfg`** secret:

```
$ oc secrets new-dockercfg <my_registry>
  --docker-server=<myregistry.example.com> --docker-username=
<notused> --docker-password=${TOKEN} --docker-email=<me@example.com>
```

7. Use the **`dockercfg`** secret to pull images from the registry by linking the secret to the service account performing the pull operations. The default service account to pull images is named **`default`**:

```
$ oc secrets link default <my_registry> --for=pull
```

8. For pushing images using the S2I feature, the **dockercfg** secret is mounted in the S2I pod, so it needs to be linked to the proper service account that performs the build. The default service account used to build images is named **builder**:

```
$ oc secrets link builder <my_registry>
```

9. In the **buildconfig**, the secret should be specified for push or pull operations:

```
"type": "Source",
"sourceStrategy": {
    "from": {
        "kind": "DockerImage",
        "name": "
<myregistry.example.com/registry_project/my_image:stable>"
    },
    "pullSecret": {
        "name": "<my_registry>"
    },
...[OUTPUT ABBREVIATED]...
"output": {
    "to": {
        "kind": "DockerImage",
        "name": "
<myregistry.example.com/registry_project/my_image:latest>"
    },
    "pushSecret": {
        "name": "<my_registry>"
    },
...[OUTPUT ABBREVIATED]...
```