OpenShift Container Platform 3.11

第二天的操作指南

OpenShift Container Platform 3.11 第两天操作指南

Last Updated: 2023-03-25

OpenShift Container Platform 3.11 第二天的操作指南

OpenShift Container Platform 3.11 第两天操作指南

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

OpenShift Container Platform 集群管理指南更为关注配置，本指南则重点介绍常见的日常维护任务。

目录

第 1 章 概述

本节是 OpenShift Container Platform 管理员需要针对全新新安装进行的操作。

OpenShift Container Platform 集群管理指南 更为关注配置，本指南则侧重于常见的日常维护任务。

第 2 章 运行一次的任务

安装 OpenShift Container Platform 后，您的系统可能需要额外的配置，以确保主机始终平稳运行。

虽然它们被归类为运行时任务，但您可以随时执行其中的任何任务（若有情况改变）。

2.1. NTP 同步

NTP（网络时间协议）用于保持主机与全局时钟同步。时间同步对于时间敏感操作（如日志保留和时间戳）非常重要，并且强烈建议在 Kubernetes 基础上构建 OpenShift Container Platform。OpenShift Container Platform 操作包括 etcd 领导选举机制、pod 的健康检查和其他问题，有助于防止时间偏差。

注意

OpenShift Container Platform 安装 playbook 会安装、启用和配置 **ntp** 软件包，以默认提供 NTP 服务。要禁用此行为，请在清单文件中设置 **openshift_clock_enabled=false**。如果主机安装了 **chrony** 软件包，则会将其配置为提供 NTP 服务，而不使用 **ntp** 软件包。

根据您的实例，可能不会默认启用 NTP。验证主机是否已配置为使用 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
```

如果 **NTP enabled** 和 **NTP synchronized** 都为 **yes**，则代表 NTP 同步处于活动状态。

如果为 **no**，请安装并启用 **ntp** 或 **chrony** RPM 软件包。

要安装 **ntp** 软件包，请运行以下命令：

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

要安装 **chrony** 软件包，请运行以下命令：

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

重要

无论是使用 NTP 还是其它方法，均应在集群中的所有主机上启用时间同步。

有关 **timedatectl** 命令、时区和时钟配置的更多信息，请参阅配置日期和时间、时区和 DST。

2.2. 熵

OpenShift Container Platform 使用熵来为对象（如 ID 或 SSL 流量）生成随机数字。这些操作会等待到有足够的熵来完成任务。如果没有足够的熵，内核无法生成具有足够速度的随机数字，这可能导致超时以及安全连接的引用。

检查可用的熵：

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

应该在集群中的所有节点主机上验证可用的熵。理想情况下，这个值应大于 **1000**。

注意

红帽建议监控这个值，并在值低于 **800** 时发出警报。

另外，您可以使用 **rngtest** 命令检查可用的熵，但是如果您的系统也可以*提供*足够的熵：

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

rngtest 命令可从 **rng-tools** 获得

如果以上操作需要大约 30 秒才能完成，则没有足够的熵可用。

根据您的环境，可以使用不同的方式增加熵。如需更多信息，请参阅以下博客文章：
https://developers.redhat.com/blog/2017/10/05/entropy-rhel-based-cloud-instances/。

通常，您可以通过安装 **rng-tools** 软件包并启用 **rngd** 服务来增加熵：

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

当 **rngd** 服务启动后，熵应增大到足够级别。

2.3. 修改默认的存储类

要使动态置备的持久性存储可以正常工作，需要定义默认存储类。在安装过程中，为常见的云供应商（如 Amazon Web Services(AWS)、Google Cloud Platform(GCP)等等）定义了这个默认存储类。

验证定义了默认存储类：

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

以上输出来自在 GCP 上运行的 OpenShift Container Platform 实例，其中提供了两种类型的持久性存储：标准(HDD)和 SSD。注意标准存储类被配置为默认存储类。如果没有定义存储类，或者没有被设置为默认值，请参阅 Dynamic Provisioning 和 Creating Storage Classes 部分来了解如何根据建议设置存储类。

第 3 章 环境健康检查

本主题包含验证 OpenShift Container Platform 集群的整体健康状况和各种组件的步骤，以及描述预期行为的步骤。

了解各种组件的验证过程是故障排除问题的第一步。如果遇到问题，您可以使用本节中提供的检查来诊断任何问题。

3.1. 检查完整的环境健康状况

要验证 OpenShift Container Platform 集群的端到端功能，请构建和部署一个示例应用程序。

流程

1. 创建一个名为 **validate** 的新项目，以及来自 **cakephp-mysql-example** 模板的示例应用程序：

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

   您可以检查日志以遵循构建：

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

2. 构建完成后，两个 pod 应该正在运行：包括一个数据库和一个应用程序：

```
$ 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. 访问应用程序 URL。应该可以看到 Cake PHP 框架欢迎页面。URL 的格式应为 **cakephp-mysql-example-validate.<app_domain>**。

4. 验证功能后，就可以删除 **validate** 项目：

```
$ oc delete project validate
```

   另外，也会删除项目中的所有资源。

3.2. 使用 PROMETHEUS 创建警报

您可以将 OpenShift Container Platform 与 Prometheus 集成，以创建视觉和警报来帮助诊断任何环境问题。如果 pod 消耗太多 CPU 或内存，则这些问题可能包括当节点停机时。

如需更多信息，请参阅 Prometheus Cluster Monitoring。

3.3. 主机健康状况

要验证集群是否正在运行，连接到 master 实例，并运行以下命令：

```
$ 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                 1h       v1.6.1+5115d708d7
ocp-master-gjkm       Ready                 1h       v1.6.1+5115d708d7
ocp-master-wc8w       Ready                 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
```

上面的集群示例包含三个 master 主机、三个基础架构节点主机和三个节点主机。它们都在运行。此输出中应当能看到群集中的所有主机。

Ready 状态表示 master 主机可以与节点主机通信，并且节点已准备好运行 pod（不包括调度被禁用的节点）。

在运行 etcd 命令前，source ***etcd.conf*** 文件：

```
# source /etc/etcd/etcd.conf
```

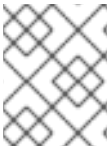
您可以使用 **etcdctl** 命令从任何 master 实例检查基本 etcd 健康状况：

```
# etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URLS 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
```

但是，要获取有关 etcd 主机的更多信息，包括关联的 master 主机：

```
# etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URLS 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
```

如果 etcd 集群与 master 服务在一起，则所有 etcd 主机都应包含 master 主机，如果 etcd 单独运行，则所有 etcd 实例都应该可以看到。



注意

etcdctl2 是 **etcdctl** 工具的别名，其中包含以 v2 数据模型查询 etcd 集群的正确标志，以及 v3 数据模型的 **etcdctl3**。

3.4. 路由器和 REGISTRY 健康状况

检查路由器服务是否在运行：

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

DESIRED 和 **CURRENT** 列中的值应与节点主机数量匹配。

使用同样的命令检查 registry 状态：

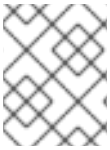
```
$ oc -n default get deploymentconfigs/docker-registry
NAME              REVISION   DESIRED   CURRENT   TRIGGERED BY
docker-registry   1          3         3         config
```

注意

如果运行容器镜像 registry 的多个实例，则后端存储需要支持多个进程的写入。如果所选的基础架构供应商不包含此功能，则需要运行单一的容器镜像 registry 实例。

验证所有 pod 都在运行，以及在哪些主机上运行：

```
$ 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
```



注意

如果 OpenShift Container Platform 使用外部容器镜像 registry，则内部 registry 服务不需要处于运行状态。

3.5. 网络连接

网络连接有两个主要网络层：用于节点交互的集群网络，以及用于 Pod 交互的软件定义型网络(SDN)。OpenShift Container Platform 支持多种网络配置，通常针对特定的基础架构提供程序进行了优化。

注意

由于网络的复杂性，本节中并不涵盖所有验证场景。

3.5.1. 在 master 主机上连接

etcd 和 master 主机

Master 服务使用 etcd "键-值"存储来保持其状态同步。master 和 etcd 服务之间的通信非常重要，无论这些 etcd 服务和 master 在同一主机上运行，还是在专业于 etcd 服务的的主机上运行。这个通信在 TCP 端口 **2379** 和 **2380** 中发生。如需了解检查此通信的方法，请参阅 Host health 部分。

SkyDNS

SkyDNS 提供在 OpenShift Container Platform 中运行的本地服务的名称解析。此服务使用 **TCP** 和 **UDP** 端口 **8053**。

验证名称解析：

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

如果回答与以下输出结果匹配，**SkyDNS** 服务可以正常工作：

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

API 服务和 Web 控制台

API 服务和 Web 控制台都共享相同的端口，通常为 **TCP 8443** 或 **443,** 具体取决于设置。这个端口需要在集群中可用，且需要与部署的环境合作的人。此端口可以访问的 URL 可能会因内部集群和外部客户端而异。

在以下示例中，**https://internal-master.example.com:443** URL 由内部集群使用，外部客户端则使用 **https://master.example.com:443** URL。在任何节点主机上：

```
$ curl -k 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"
}
```

这必须可从客户端的网络访问：

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

确定内部和外部 master URL

以下命令可用于确定内部集群和外部客户端使用的 URL。上例中可以找到 URL 示例。

要确定内部集群 URL：

```
$ grep masterURL /etc/origin/master/master-config.yaml
```

要确定外部客户端 URL：

```
$ grep masterPublicURL /etc/origin/master/master-config.yaml
```

3.5.2. 节点实例间的连接

默认情况下，SDN 在节点上连接 pod 通信时使用 **UDP** 端口 **4789**。

要验证节点主机功能，请创建新应用。以下示例确保节点可以提供容器镜像 registry，该镜像 registry 在基础架构节点上运行：

流程

1. 创建一个新项目

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

2. 部署 httpd 应用程序：

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

等待构建完成：

```
$ 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. 连接到正在运行的 pod：

```
$ oc rsh po/<pod-name>
```

例如：

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

4. 检查内部 registry 服务的 **healthz** 路径：

```
$ 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*
```

HTTP/1.1 200 OK 响应表示节点正确连接。

5. 清理 test 项目：

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

6. 节点主机正在侦听 **TCP** 端口 **10250**。此端口需要可以被任何节点上的所有 master 主机访问，如果在集群中部署了监控，基础架构节点也必须可以访问所有实例上的所有端口。可以通过以下命令检测到这个端口上的通信中断：

```
$ 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
```

在上面的输出中，master 服务无法访问 **ocp-node-w135** 节点上的节点服务，该服务由其 **NotReady** 状态表示。

7. 最后一个服务是路由器，它负责将连接路由到 OpenShift Container Platform 集群中运行的正确服务。路由器在基础架构节点上的 **TCP** 端口 **80** 和 **443** 上侦听入口流量。在路由器可以开始工作前，必须配置 DNS：

```
$ 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
```

IP 地址（本例中为 **35.xx.xx.92**）应指向将入口流量分发到所有基础架构节点的负载均衡器。要验证路由器的功能，请再次检查 registry 服务，但这次它们来自集群外部：

```
$ 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
```

3.6. 存储

Master 实例对 **/var** 目录至少需要 40 GB 的硬盘空间。使用 **df** 命令检查 master 主机的磁盘用量：

```
$ 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
```

节点实例需要至少 15 GB 空间用于 **/var** 目录，以及另外最少 15 GB 空间用于 Docker 存储（本例中为 **/var/lib/docker**）。根据集群的大小和 pod 所需的临时存储大小，应该为节点上的 **/var/lib/origin/openshift.local.volumes** 创建单独的分区。

```
$ 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
```

pod 的持久性存储应该在运行 OpenShift Container Platform 集群的实例外进行处理。pod 的持久性卷可由基础架构提供程序置备，也可以使用容器原生存储或容器就绪存储。

3.7. DOCKER 存储

Docker 存储的后端使用以下两个选项之一：第一个选项是代设备映射器的精简池逻辑卷；另一个（从 Red Hat Enterprise Linux version 7.4 开始）是使用一个 overlay2 文件系统通常建议使用 overlay2 文件系统，因为设置更轻松则性能更高。

Docker 存储磁盘挂载为 **/var/lib/docker**，并格式化为 **xfs** 文件系统。Docker 存储配置为使用 overlay2 文件系统：

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

Docker 验证此存储驱动程序是否使用了：

```
# 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.redhat.io/v1/
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
Insecure Registries:
 127.0.0.0/8
Registries: registry.redhat.io (secure), registry.redhat.io (secure), docker.io (secure)
```

3.8. API 服务状态

OpenShift API 服务在所有主实例上运行。要查看服务的状态，查看 **kube-system** 项目中的 master-api pod：

```
oc get pod -n kube-system -l openshift.io/component=api

NAME                        READY     STATUS    RESTARTS   AGE
master-api-myserver.com     1/1       Running   0          56d
```

API 服务会公开健康检查，该检查可以使用 API 主机名从外部查询。API 服务和 Web 控制台都共享相同的端口，通常是 TCP 8443 或 443，具体取决于设置。这个端口需要在集群中可用，且需要与部署的环境合作：

```
oc get pod -n kube-system -o wide
NAME                                READY     STATUS    RESTARTS   AGE       IP            NODE
master-api-myserver.com             1/1       Running   0          7h        10.240.0.16
myserver.com

$ curl -k https://myserver.com:443/healthz (1)
ok
```

1 这必须可从客户端的网络访问。本例中的 Web 控制台端口是 **443。**指定在 OpenShift Container Platform 部署前为主机清单文件中为 **openshift_master_console_port** 设置的值。如果清单文件中不包含 **openshift_master_console_port**，端口 **8443** 会被默认设置为。

3.9. 控制器角色验证

OpenShift Container Platform 控制器服务在所有 master 主机之间可用。该服务以主动/被动模式运行，这意味着该服务可以随时只在一个主机上运行。

OpenShift Container Platform 控制器执行一个流程来选择哪个主机运行该服务。当前运行的值存储在存储在 **kube-system** 项目中的特殊 **configmap** 中的注解中。

验证 master 主机以 **cluster-admin** 用户身份运行控制器服务：

```
$ 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
```

该命令在 **holderIdentity** 属性中输出 **control-plane.alpha.kubernetes.io/leader** 注解中的当前 master 控制器，如下所示：

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

使用以下命令过滤输出来查找 master 主机的主机名：

```
$ 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
```

3.10. 验证最大传输单元(MTU)大小

验证最大传输单元(MTU)可防止可能的网络错误配置伪装为 SSL 证书问题。

当数据包大于 HTTP 传输的 MTU 大小时，物理网络路由器可以将数据包分解为多个数据包来传输数据。但是，当数据包大于通过 HTTPS 传输的 MTU 大小时，路由器会被强制丢弃数据包。

安装会生成证书来为多个组件提供安全连接，其中包括：

- Master 主机
- 节点主机
- 基础架构节点
- registry
- 路由器

这些证书可以在 master 节点的 **/etc/origin/master** 目录中找到，而 **/etc/origin/node** 目录则可用于 infra 和 app 节点。

安装后，您可以使用网络连接部分中介绍的过程验证与 **REGISTRY_OPENSHIFT_SERVER_ADDR** 的连接。

先决条件

1. 在 master 主机中获取 HTTPS 地址：

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

   以上会输出 **docker-registry.default.svc:5000**。

2. 将 **/healthz** 附加到上方给出的值，使用它检查所有主机（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
```

上例输出中显示了用来确保 SSL 连接正确使用的 MTU 大小。尝试连接成功，然后建立连接并完成后，使用 certpath 初始化 NSS 以及有关 *docker-registry* 的所有服务器证书信息即可完成。

不正确的 MTU 大小会导致超时：

```
$ 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
```

上例显示连接已建立，但无法完成使用 certpath 初始化 NSS。这个问题与在相关的节点配置映射中不正确的 MTU 大小相关。

要解决这个问题，将节点配置映射中的 MTU 大小调整为比 OpenShift SDN 以太网设备使用的 MTU 大小小 50 字节。

3. 查看所需以太网设备的 MTU 大小（例如 **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
```

以上显示了 MTU 设置为 1500。

4. 要更改 MTU 大小，请修改适当的节点 配置映射 并设置比 **ip** 命令提供的输出小 50 字节的值。例如，如果 MTU 大小被设置为 1500，在节点配置映射中将 MTU 大小调整为 1450：

```
networkConfig:
  mtu: 1450
```

5. 保存更改并重启节点：

注意

您必须在所有 master 和作为 OpenShift Container Platform SDN 一部分的节点上更改 MTU 大小。另外，tun0 接口的 MTU 大小必须在属于集群的所有节点中相同。

6. 一旦节点恢复在线后，通过重新运行原始 **curl** 命令来确认问题已不存在。

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

如果仍然有超时问题，请继续以 50 字节的增量调整 MTU 大小，然后重复该过程。

第 4 章 创建环境范围内的备份

创建环境范围内的备份涉及复制重要数据，以便在崩溃实例或损坏数据时帮助恢复数据。创建备份后，可以恢复到相关组件的新安装的版本。

在 OpenShift Container Platform 中，您可以在集群级别进行 *备份*，并将状态保存到独立的存储中。环境备份的完整状态包括：

- 集群数据文件
- 每个 master 上的 etcd 数据
- API 对象
- registry 存储
- 卷存储

定期执行备份以防止数据丢失。

重要

以下流程描述了备份应用程序和 OpenShift Container Platform 集群的通用方法。它不能考虑自定义要求。使用以下步骤作为集群的完整备份和恢复流程的基础。您必须采取所有必要的措施以防止数据丢失。

无法保证备份和恢复。您负责备份自己的数据。

4.1. 创建 MASTER 主机备份

在对 OpenShift Container Platform 基础架构进行任何更改之前执行此备份过程，如系统更新、升级或其他显著修改。定期备份数据，以确保在出错时有最新的数据可用。

OpenShift Container Platform 文件

主（master）实例运行重要的服务，如 API 和控制器。**/etc/origin/master** 目录存储很多重要文件：

- 配置、API、控制器、服务等
- 安装生成的证书
- 所有与云供应商相关的配置
- 密钥及其他身份验证文件，例如，如果使用 htpasswd 时为 **htpasswd**
- 更多

您可以自定义 OpenShift Container Platform 服务，如增加日志级别或使用代理。配置文件存储在 **/etc/sysconfig** 目录中。

因为 master 也是节点，所以备份整个 **/etc/origin** 目录。

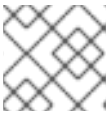
流程

重要

您必须在每个 master 节点上执行以下步骤。

1. 创建 pod 定义备份（在此）。.
2. 创建 master 主机配置文件的备份：

```
$ 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/ ${MYBACKUPDIR}/etc/sysconfig/
```



注意

master 配置文件为/etc/origin/master/master-config.yaml。

警告

/etc/origin/master/ca.serial.txt 文件仅对 Ansible 主机清单中列出的第一个 master 生成。如果您弃用了第一个 master 主机，请在进程前将 **/etc/origin/master/ca.serial.txt** 文件复制到其余 master 主机中。

重要

在运行多个 master 的 OpenShift Container Platform 3.11 集群中，其中一个 master 节点在 **/etc/origin/master**、**/etc/etcd/ca** 和 **/etc/etcd/generated_certs** 中包括额外的 CA 证书。这些是应用程序节点和 etcd 节点扩展操作所必需的，且需要在另一个 master 节点上恢复，它应该永久不可用。这些目录默认包含在此处的备份过程中。

3. 在规划备份时需要考虑的其他重要文件包括：

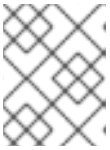
File	描述
/etc/cni/*	容器网络接口配置（如果使用）
/etc/sysconfig/iptables	存储 **iptables** 规则的地方
/etc/sysconfig/docker-storage-setup	**container-storage-setup** 命令的输入文件
/etc/sysconfig/docker	**docker** 配置文件
/etc/sysconfig/docker-network	**docker** 网络配置（如 MTU）
/etc/sysconfig/docker-storage	**docker** 存储配置（由 **container-storage-setup** 生成）

/etc/dnsmasq.conf	**dnsmasq** 的主要配置文件
/etc/dnsmasq.d/*	不同的 **dnsmasq** 配置文件
/etc/sysconfig/flanneld	**flannel** 配置文件（如果使用）
/etc/pki/ca-trust/source/anchors/	添加到系统的证书（例如，外部 registry）

创建这些文件的备份：

```
$ 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/
```

4. 如果软件包被意外删除，或者您需要重新整合 **rpm** 软件包中所含的文件，则系统中安装的 **rhel** 软件包列表会很有用。



注意

如果您使用 Red Hat Satellite 功能（如内容视图或事实存储），请提供正确机制来重新安装缺少的软件包和系统中安装的软件包的历史数据。

要创建系统中当前 **rhel** 软件包的列表：

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

5. 如果您使用前面的步骤，则备份目录中存在以下文件：

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

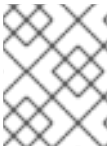
```
etc/origin/master/serviceaccounts.private.key
etc/origin/master/serviceaccounts.public.key
etc/origin/master/openshift-master.crt
etc/origin/master/openshift-master.key
etc/origin/master/openshift-master.kubeconfig
etc/origin/master/master.server.crt
etc/origin/master/master.server.key
etc/origin/master/master.kubelet-client.crt
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
```

如果需要，您可以压缩文件以节省空间：

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

-

要从头开始创建任何这些文件，**openshift-ansible-contrib** 存储库包含 **backup_master_node.sh** 脚本，它将执行前面的步骤。该脚本在运行 脚本的主机上创建一个目录，并复制前面提到的所有文件。



注意

红帽不支持 **openshift-ansible-contrib** 脚本，但可以在架构团队执行测试以确保代码运行正常且安全时作为参考。

您可以使用以下方法在每个 master 主机上运行该脚本：

```
$ 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. 创建节点主机备份

创建节点主机的备份与备份 master 主机的用例不同。由于 master 主机包含许多重要文件，因此强烈建议创建备份。但是，节点的性质是，在故障切换时，任何特殊操作都会在节点上复制，它们通常不包含运行环境所需的数据。如果节点的备份包含运行环境所需的一些内容，则建议使用创建备份。

备份过程会在基础架构进行任何更改之前执行，如系统更新、升级或任何其他显著的修改。备份应定期执行，确保在出现故障时最新的数据可用。

OpenShift Container Platform 文件

节点实例以 pod 的形式运行应用程序，这些 pod 基于容器。**/etc/origin/** 和 **/etc/origin/node** 目录包括了重要文件，例如：

- 节点服务的配置
- 安装生成的证书
- 与云供应商相关的配置
- 密钥和其他身份验证文件，如 **dnsmasq** 配置

可以自定义 OpenShift Container Platform 服务来提高日志级别、使用代理等，并且配置文件存储在 **/etc/sysconfig** 目录中。

流程

1. 创建节点配置文件的备份：

```
$ 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 使用在规划备份策略时必须考虑的特定文件，包括：

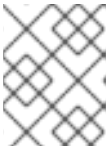
File	描述

/etc/cni/*	容器网络接口配置（如果使用）
/etc/sysconfig/iptables	存储 **iptables** 规则的地方
/etc/sysconfig/docker-storage-setup	**container-storage-setup** 命令的输入文件
/etc/sysconfig/docker	**docker** 配置文件
/etc/sysconfig/docker-network	**docker** 网络配置（如 MTU）
/etc/sysconfig/docker-storage	**docker** 存储配置（由 **container-storage-setup** 生成）
/etc/dnsmasq.conf	**dnsmasq** 的主要配置文件
/etc/dnsmasq.d/*	不同的 **dnsmasq** 配置文件
/etc/sysconfig/flanneld	**flannel** 配置文件（如果使用）
/etc/pki/ca-trust/source/anchors/	添加到系统的证书（例如，外部 registry）

创建这些文件：

```
$ 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. 如果意外删除软件包，或者应该恢复 **rpm** 软件包中包含的文件，系统中安装的 **rhel** 软件包列表很有用。



注意

如果使用 Red Hat Satellite 功能（如内容视图或事实存储），请提供正确机制来重新安装缺少的软件包和系统中安装的软件包的历史数据。

要创建系统中当前 **rhel** 软件包的列表：

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

4. 备份目录中现在应存在以下文件：

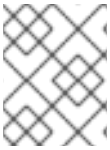
```
$ MYBACKUPDIR=/backup/$(hostname)/$(date +%Y%m%d)
```

```
$ 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
```

如果需要，可以压缩文件以节省空间：

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

要从头开始创建任何这些文件，**openshift-ansible-contrib** 存储库包含 **backup_master_node.sh** 脚本，它将执行前面的步骤。该脚本在运行 脚本的主机上创建一个目录，并复制前面提到的所有文件。



注意

红帽不支持 **openshift-ansible-contrib** 脚本，但可以在架构团队执行测试以确保代码运行正常且安全时作为参考。

可以使用以下方法在每个 master 主机上执行该脚本：

```
$ 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. 备份 REGISTRY 证书

如果使用一个外部安全 registry，您必须保存所有 registry 证书。默认情况下，registry 会被保护。

重要

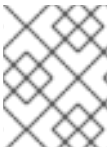
您必须在每个集群节点上执行以下步骤。

流程

1. 备份 registry 证书：

```
# cd /etc/docker/certs.d/
# tar cf /tmp/docker-registry-certs-$(hostname).tar *
```

2. 将备份移到外部位置。



注意

使用一个或多个外部安全 registry 时，拉取或推送镜像的任何主机都必须信任 registry 的证书来运行 pod。

4.4. 备份其他安装文件

备份用于安装 OpenShift Container Platform 的文件。

流程

1. 因为恢复过程涉及完整的重新安装，因此请保存初始安装中使用的所有文件。这些文件可能包括：
   - 来自集群安装的 Ansible playbook 和清单文件
   - 断开连接的安装方法中的 */etc/yum.repos.d/ose.repo*
2. 备份安装后步骤的步骤。有些安装可能涉及安装程序中没有包括的步骤。这些步骤可能包括对 OpenShift Container Platform 控制外的服务的更改或安装额外的服务，如监控代理。高级安装程序尚不支持的其他配置也可能会受到影响，比如使用多个身份验证提供程序。

4.5. 备份应用程序数据

在很多情况下，您可以使用 **oc rsync** 命令备份应用程序数据，假设容器镜像中安装 **rsync**。Red Hat **rhel7** 基础镜像包含 **rsync**.因此，所有基于 **rhel7** 的镜像也都包含它。请参阅 对 CLI 操作进行故障排除和调试 - rsync。

警告

这是应用程序数据的*通用*备份，不会考虑特定于应用程序的备份过程，如数据库系统的特殊导出和导入步骤。

其他备份方法可能取决于您使用的持久性卷类型，如 Cinder、NFS 或 Gluster。

备份的路径也是*特定于应用程序*的路径。您可以通过查看 **deploymentconfig** 中的卷的 **mountPath** 来确定要备份的路径。

注意

只有在应用程序 pod 运行时，才能执行这类应用程序数据备份。

流程

备份 Jenkins 部署的应用程序数据的示例

1. 从 **deploymentconfig** 获取应用程序数据 **mountPath**:

```
$ oc get dc/jenkins -o jsonpath='{ .spec.template.spec.containers[?
(@.name=="jenkins")].volumeMounts[?(@.name=="jenkins-data")].mountPath }'
/var/lib/jenkins
```

2. 获取当前运行的 pod 的名称：

```
$ oc get pod --selector=deploymentconfig=jenkins -o jsonpath='{ .metadata.name }'
jenkins-1-37nux
```

3. 使用 **oc rsync** 命令复制应用程序数据：

```
$ oc rsync jenkins-1-37nux:/var/lib/jenkins /tmp/jenkins-backup/
```

4.6. ETCD 备份

etcd 是所有对象定义的键值存储，以及持久 master 状态。其他组件会监视更改，然后进入所需的状态。

3.5 之前的 OpenShift Container Platform 版本使用 etcd 版本 2(v2)，而 3.5 及更新版本使用版本 3(v3)。两个版本的 etcd 之间的数据模型不同。etcd v3 可以使用 v2 和 v3 数据模型，而 etcd v2 只能使用 v2 数据模型。在 etcd v3 服务器中，v2 和 v3 数据存储可以并行存在，并相互独立。

对于 v2 和 v3 操作，您可以使用 **ETCDCTL_API** 环境变量来使用正确的 API：

```
$ etcdctl -v
etcdctl version: 3.2.28
API version: 2

$ ETCDCTL_API=3 etcdctl version
etcdctl version: 3.2.28
API version: 3.2
```

如需有关如何迁移到 v3 的信息，请参阅 OpenShift Container Platform 3.7 文档中的迁移 etcd 数据（v2 到 v3）部分。

在 OpenShift Container Platform 版本 3.10 及更新版本中，您可以在单独的主机上安装 etcd，或作为静态 pod 在 master 主机上运行。如果没有指定独立的 etcd 主机，etcd 会作为静态 pod 在 master 主机上运行。因此，如果您使用静态 pod，备份过程会有所不同。

etcd 备份过程由两个不同的步骤组成：

- 配置备份：包含所需的 etcd 配置和证书
- 数据备份：包含 v2 和 v3 数据模型。

您可以在任何已连接到 etcd 集群的主机上执行数据备份过程，其中提供了正确的证书，以及安装 **etcdctl** 工具的位置。

注意

备份文件必须复制到外部系统，最好在 OpenShift Container Platform 环境之外，然后进行加密。

请注意，etcd 备份仍具有当前存储卷的所有引用。恢复 etcd 时，OpenShift Container Platform 开始在节点上启动以前的 pod 并重新连接相同的存储。此过程与从集群中删除节点的过程不同，并在其位置添加一个新节点。附加到该节点的任何节点上的任何节点都会重新附加到 pod。

4.6.1. 备份 etcd

当备份 etcd 时，您必须备份 etcd 配置文件和 etcd 数据。

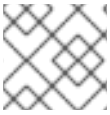
4.6.1.1. 备份 etcd 配置文件

要保留的 etcd 配置文件都存储在运行 etcd 的实例的 **/etc/etcd** 目录中。这包括 etcd 配置文件 (**/etc/etcd/etcd.conf**)和集群通信所需的证书。所有这些文件都是在安装时由 Ansible 安装程序生成的。

流程
对于集群的每个 etcd 成员，备份 etcd 配置。

```
$ ssh master-0 1
# mkdir -p /backup/etcd-config-$(date +%Y%m%d)/
# cp -R /etc/etcd/ /backup/etcd-config-$(date +%Y%m%d)/
```

1 将 **master-0** 替换为 etcd 成员的名称。



注意

每个 etcd 集群成员上的证书和配置文件是唯一的。

4.6.1.2. 备份 etcd 数据

先决条件

注意

OpenShift Container Platform 安装程序创建别名，以避免为 etcd v2 任务输入名为 **etcdctl2** 的所有标志，以及用于 etcd v3 任务的 **etcdctl3**。

但是，**etcdctl3** 别名不会向 **etcdctl** 命令提供完整的端点列表，因此您必须指定 **--endpoints** 选项并列出所有端点。

备份 etcd 之前：

- **etcdctl** 二进制文件必须可用，或者在容器化安装中，**rhel7/etcd** 容器必须可用。
- 确保 OpenShift Container Platform API 服务正在运行。
- 确保与 etcd 集群的连接（端口 2379/tcp）。
- 确保正确的证书以连接到 etcd 集群。
- 通过检查其健康状况来确保 etcd 集群正常工作：
  - 检查端点的健康状况：

```
# etcdctl3 --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" \ 1
        endpoint health
```

1 将这些值替换为集群的端点。

输**出示例**

```
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
```

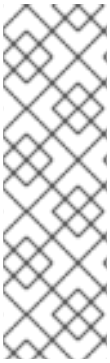
  - 检查成员列表。

```
# etcdctl3 member list
```

输**出示例**

```
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
```

流程

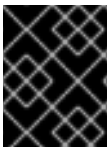


注意

虽然 **etcdctl backup** 命令用于执行备份，etcd v3 没有 *备份* 的概念。您可以使用 **etcdctl snapshot save** 命令对一个实时成员进行 *快照*，或从 etcd 数据目录中复制 **member/snap/db** 文件。

etcdctl backup 命令重写了备份中所含的一些元数据，特别是节点 ID 和集群 ID，这意味着在备份中，节点会丢失它的以前的身份。要从备份重新创建集群，您可以创建一个新的单节点集群，然后将其他节点的其余部分添加到集群中。元数据被重写以防止新节点加入现有集群。

备份 etcd 数据：



重要

从以前的 OpenShift Container Platform 版本升级的集群可能包含 v2 数据存储。备份所有 etcd 数据存储。

1. 从静态 pod 清单获取 etcd 端点 IP 地址：

```
$ export ETCD_POD_MANIFEST="/etc/origin/node/pods/etcd.yaml"
```

```
$ export ETCD_EP=$(grep https ${ETCD_POD_MANIFEST} | cut -d '/' -f3)
```

2. 以管理员身份登录：

```
$ oc login -u system:admin
```

3. 获取 etcd pod 名称：

```
$ export ETCD_POD=$(oc get pods -n kube-system | grep -o -m 1 '^master-etcd\S*')
```

4. 进入 **kube-system** 项目：

```
$ oc project kube-system
```

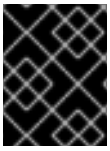
5. 在 pod 中生成 etcd 数据快照并将其保存在本地：

```
$ oc exec ${ETCD_POD} -c etcd -- /bin/bash -c "ETCDCTL_API=3 etcdctl \
    --cert /etc/etcd/peer.crt \
    --key /etc/etcd/peer.key \
    --cacert /etc/etcd/ca.crt \
    --endpoints $ETCD_EP \
    snapshot save /var/lib/etcd/snapshot.db" (1)
```

(1) 您必须将快照写入 **/var/lib/etcd/** 下的目录中。

4.7. 备份项目

对所有相关数据创建备份涉及导出所有重要信息，然后恢复到一个新项目中。



重要

因为 **oc get all** 命令只返回某些项目资源，所以您必须单独备份其他资源，包括 PVC 和 Secret，如以下步骤所示。

流程

1. 列出要备份的项目数据：

```
$ oc get all
```

输**出示例**

```
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. 将项目对象导出到 **project.yaml** 文件。

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

3. 在项目中导出其他对象，如角色绑定、secret、服务帐户和持久性卷声明。
您可以使用以下命令导出项目中的所有命名空间对象：

```
$ for object in $(oc api-resources --namespaced=true -o name)
```

```
do
  oc get -o yaml --export $object > $object.yaml
done
```

请注意，无法导出某些资源，并显示 **MethodNotAllowed** 错误。

4. 一些导出的对象可以依赖特定元数据或对项目中唯一 ID 的引用。这是对重新创建的对象可用性的一个限制。
使用**镜像流**时，**deploymentconfig** 的 **image** 参数可以指向已恢复环境中不存在的内部 registry 中镜像的特定 **sha** checksum。例如，以 **oc new-app centos/ruby-22-centos7~https://github.com/sclorg/ruby-ex.git** 运行示例 "ruby-ex" 作为 oc new-app centos7~https://github.com/sclorg/ruby-ex.git 创建一个 **镜像流 ruby-ex** 来托管该镜像：

```
$ oc get dc ruby-ex -o jsonpath="{.spec.template.spec.containers[].image}"
10.111.255.221:5000/myproject/ruby-ex@sha256:880c720b23c8d15a53b01db52f7abdcbb2280e03f686a5c8edfef1a2a7b21cee
```

如果导入使用 **oc get --export** 导出的 **deploymentconfig**，在镜像不存在时会失败。

4.8. 备份持久性卷声明

您可以从容器内部将持久数据同步到服务器。

重要

根据托管 OpenShift Container Platform 环境的供应商，也可以为备份和恢复目的启动第三方快照服务。因为 OpenShift Container Platform 没有能够启动这些服务，所以本指南并不描述了这些步骤。

如需特定应用程序的正确备份流程，请参阅任何产品文档。例如，复制 mysql 数据目录本身并不会创建可用的备份。反之，运行关联的应用程序的特定备份过程，然后同步任何数据。这包括使用 OpenShift Container Platform 托管平台提供的快照解决方案。

流程

1. 查看项目和 pod：

```
$ 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. 描述所需的 pod，以查找持久性卷目前使用的卷：

```
$ 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
```

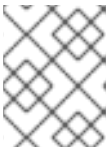
```
IP:   172.16.12.5
Controllers:  ReplicationController/demo-2
Containers:
  demo:
    Container ID:
docker://201f3e55b373641eb36945d723e1e212ecab847311109b5cee1fd0109424217a
    Image:  docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20dc9ff6f350436f935
968b0c83fcb98a7a8c381a
    Image ID:  docker-pullable://docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20dc9ff6f350436f935
968b0c83fcb98a7a8c381a
    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...
```

此输出显示持久性数据位于 **/opt/app-root/src/uploaded** 目录中。

3. 在本地复制数据：

```
$ 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
```

ocp_sop.txt 文件下载到本地系统，以通过备份软件或其他备份机制备份。



注意

如果 pod 启动而无需使用 **pvc**，但稍后您决定需要 **pvc**，则可以使用前面的步骤。您可以保留数据，然后使用剩余的进程填充新存储。

第 5 章 主机级任务

5.1. 在集群中添加主机

有关向集群添加 master 或节点主机的详情，请参考安装和配置指南中的将主机添加到现有集群部分。

5.2. MASTER 主机任务

5.2.1. 弃用 master 主机

弃用 master 主机会从 OpenShift Container Platform 环境中删除它。

弃用或缩减 master 主机的原因包括硬件需要重新调整大小，或替换底层基础架构。

高可用性 OpenShift Container Platform 环境需要至少三个 master 主机和三个 etcd 节点。通常，master 主机与 etcd 服务在一起。如果您弃用了 master 主机，您也从该主机中删除 etcd 静态 pod。

重要

由于在这些服务间发生的投票机制，请确保始终以奇数数字部署 master 和 etcd 服务。

5.2.1.1. 创建 master 主机备份

在对 OpenShift Container Platform 基础架构进行任何更改之前执行此备份过程，如系统更新、升级或其他显著修改。定期备份数据，以确保在出错时有最新的数据可用。

OpenShift Container Platform 文件

主（master）实例运行重要的服务，如 API 和控制器。**/etc/origin/master** 目录存储很多重要文件：

- 配置、API、控制器、服务等
- 安装生成的证书
- 所有与云供应商相关的配置
- 密钥及其他身份验证文件，例如，如果使用 htpasswd 时为 **htpasswd**
- 更多

您可以自定义 OpenShift Container Platform 服务，如增加日志级别或使用代理。配置文件存储在 **/etc/sysconfig** 目录中。

因为 master 也是节点，所以备份整个 **/etc/origin** 目录。

流程

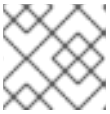
重要

您必须在每个 master 节点上执行以下步骤。

1. 创建 pod 定义备份（在此）。.

2. 创建 master 主机配置文件的备份：

```
$ 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/ ${MYBACKUPDIR}/etc/sysconfig/
```



注意

master 配置文件为/etc/origin/master/master-config.yaml。

警告

/etc/origin/master/ca.serial.txt 文件仅对 Ansible 主机清单中列出的第一个 master 生成。如果您弃用了第一个 master 主机，请在进程前将 **/etc/origin/master/ca.serial.txt** 文件复制到其余 master 主机中。

重要

在运行多个 master 的 OpenShift Container Platform 3.11 集群中，其中一个 master 节点在 **/etc/origin/master**、**/etc/etcd/ca** 和 **/etc/etcd/generated_certs** 中包括额外的 CA 证书。这些是应用程序节点和 etcd 节点扩展操作所必需的，且需要在另一个 master 节点上恢复，它应该永久不可用。这些目录默认包含在此处的备份过程中。

3. 在规划备份时需要考虑的其他重要文件包括：

File	描述
/etc/cni/*	容器网络接口配置（如果使用）
/etc/sysconfig/iptables	存储 **iptables** 规则的地方
/etc/sysconfig/docker-storage-setup	**container-storage-setup** 命令的输入文件
/etc/sysconfig/docker	**docker** 配置文件
/etc/sysconfig/docker-network	**docker** 网络配置（如 MTU）
/etc/sysconfig/docker-storage	**docker** 存储配置（由 **container-storage-setup** 生成）
/etc/dnsmasq.conf	**dnsmasq** 的主要配置文件
/etc/dnsmasq.d/*	不同的 **dnsmasq** 配置文件

/etc/sysconfig/flanneld	**flannel** 配置文件（如果使用）
/etc/pki/ca-trust/source/anchors/	添加到系统的证书（例如，外部 registry）

创建这些文件的备份：

```
$ 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/
```

4. 如果软件包被意外删除，或者您需要重新整合 **rpm** 软件包中所含的文件，则系统中安装的 **rhel** 软件包列表会很有用。

注意

如果您使用 Red Hat Satellite 功能（如内容视图或事实存储），请提供正确机制来重新安装缺少的软件包和系统中安装的软件包的历史数据。

要创建系统中当前 **rhel** 软件包的列表：

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

5. 如果您使用前面的步骤，则备份目录中存在以下文件：

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

如果需要，您可以压缩文件以节省空间：

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

要从头开始创建任何这些文件，**openshift-ansible-contrib** 存储库包含 **backup_master_node.sh** 脚本，它将执行前面的步骤。该脚本在运行 脚本的主机上创建一个目录，并复制前面提到的所有文件。

注意

红帽不支持 **openshift-ansible-contrib** 脚本，但可以在架构团队执行测试以确保代码运行正常且安全时作为参考。

您可以使用以下方法在每个 master 主机上运行该脚本：

```
$ 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
```

5.2.1.2. 备份 etcd

当备份 etcd 时，您必须备份 etcd 配置文件和 etcd 数据。

5.2.1.2.1. 备份 etcd 配置文件

要保留的 etcd 配置文件都存储在运行 etcd 的实例的 **/etc/etcd** 目录中。这包括 etcd 配置文件 (**/etc/etcd/etcd.conf**)和集群通信所需的证书。所有这些文件都是在安装时由 Ansible 安装程序生成的。

流程
对于集群的每个 etcd 成员，备份 etcd 配置。

```
$ ssh master-0 1
# mkdir -p /backup/etcd-config-$(date +%Y%m%d)/
# cp -R /etc/etcd/ /backup/etcd-config-$(date +%Y%m%d)/
```

1 将 **master-0** 替换为 etcd 成员的名称。

注意

每个 etcd 集群成员上的证书和配置文件是唯一的。

5.2.1.2.2. 备份 etcd 数据

先决条件

注意

OpenShift Container Platform 安装程序创建别名，以避免为 etcd v2 任务输入名为 **etcdctl2** 的所有标志，以及用于 etcd v3 任务的 **etcdctl3**。

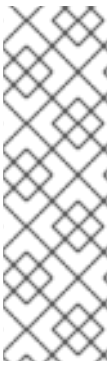
但是，**etcdctl3** 别名不会向 **etcdctl** 命令提供完整的端点列表，因此您必须指定 **--endpoints** 选项并列出所有端点。

备份 etcd 之前：

- **etcdctl** 二进制文件必须可用，或者在容器化安装中，**rhel7/etcd** 容器必须可用。
- 确保 OpenShift Container Platform API 服务正在运行。

- 确保与 etcd 集群的连接（端口 2379/tcp）。
- 确保正确的证书以连接到 etcd 集群。

流程



注意

虽然 **etcdctl backup** 命令用于执行备份，etcd v3 没有 *备份* 的概念。您可以使用 **etcdctl snapshot save** 命令对一个实时成员进行 *快照*，或从 etcd 数据目录中复制 **member/snap/db** 文件。

etcdctl backup 命令重写了备份中所含的一些元数据，特别是节点 ID 和集群 ID，这意味着在备份中，节点会丢失它的以前的身份。要从备份重新创建集群，您可以创建一个新的单节点集群，然后将其他节点的其余部分添加到集群中。元数据被重写以防止新节点加入现有集群。

备份 etcd 数据：

重要

从以前的 OpenShift Container Platform 版本升级的集群可能包含 v2 数据存储。备份所有 etcd 数据存储。

1. 从静态 pod 清单获取 etcd 端点 IP 地址：

```
$ export ETCD_POD_MANIFEST="/etc/origin/node/pods/etcd.yaml"
```

```
$ export ETCD_EP=$(grep https ${ETCD_POD_MANIFEST} | cut -d '/' -f3)
```

2. 以管理员身份登录：

```
$ oc login -u system:admin
```

3. 获取 etcd pod 名称：

```
$ export ETCD_POD=$(oc get pods -n kube-system | grep -o -m 1 '^master-etcd\S*')
```

4. 进入 **kube-system** 项目：

```
$ oc project kube-system
```

5. 在 pod 中生成 etcd 数据快照并将其保存在本地：

```
$ oc exec ${ETCD_POD} -c etcd -- /bin/bash -c "ETCDCTL_API=3 etcdctl \
    --cert /etc/etcd/peer.crt \
    --key /etc/etcd/peer.key \
    --cacert /etc/etcd/ca.crt \
    --endpoints $ETCD_EP \
    snapshot save /var/lib/etcd/snapshot.db" (1)
```

(1) 您必须将快照写入 **/var/lib/etcd/** 下的目录中。

5.2.1.3. 弃用 master 主机

Master 主机运行重要的服务，如 OpenShift Container Platform API 和控制器服务。为了弃用 master 主机，必须停止这些服务。

OpenShift Container Platform API 服务是一个主动/主动服务，因此只要请求发送到单独的 master 服务器，停止该服务不会影响环境。但是，OpenShift Container Platform 控制器服务是一个主动/被动服务，服务使用 etcd 来决定活跃的 master。

在多主控机架构中弃用 master 主机包括从负载均衡器池中移除 master，以避免尝试使用该 master 的新连接。这个过程很大程度上依赖于使用的负载均衡器。以下步骤显示从 **haproxy** 中删除 master 的详细信息。如果 OpenShift Container Platform 在云供应商上运行或使用 **F5** 设备，请查看特定的产品文档来从轮转中删除 master。

流程

1. 删除 **/etc/haproxy/haproxy.cfg** 配置文件中的 **backend** 部分。例如，如果使用 **haproxy** 弃用名为 **master-0.example.com** 的 master，请确保从以下内容中删除主机名：

```
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. 然后，重新启动 **haproxy** 服务。

```
$ sudo systemctl restart haproxy
```

3. 从负载均衡器中删除 master 后，通过将定义文件移出静态 pod dir /etc/origin/node/pods 来禁用 API 和控制器服务：

```
# mkdir -p /etc/origin/node/pods/disabled
# mv /etc/origin/node/pods/controller.yaml /etc/origin/node/pods/disabled/:
+
```

4. 由于 master 主机是一个可调度的 OpenShift Container Platform 节点，因此请按照 弃用节点主机 部分中的步骤进行操作。

5. 从 **/etc/ansible/hosts** Ansible 清单文件中的 **[masters]** 和 **[nodes]** 组移除 master 主机，以避免在使用该清单文件运行任何 Ansible 任务时出现问题。

警告

弃用 Ansible 清单中列出的第一个 master 主机需要额外的举措。

/etc/origin/master/ca.serial.txt 文件仅对 Ansible 主机清单中列出的第一个 master 生成。如果您弃用了第一个 master 主机，请在进程前将 **/etc/origin/master/ca.serial.txt** 文件复制到其余 master 主机中。

重要

在运行多个 master 的 OpenShift Container Platform 3.11 集群中，其中一个 master 节点会包括额外的 CA 证书（**/etc/origin/master**, **/etc/etcd/ca**, 和 **/etc/etcd/generated_certs**）。这些是应用程序节点和 etcd 节点扩展操作所必需的，如果 CA 主机 master 已被弃用，则必须在另一个 master 节点上恢复。

6. **kubernetes** 服务将 master 主机 IP 作为端点包含在内。要验证 master 已被正确弃用，请查看 **kubernetes** 服务输出并查看已弃用的 master 是否已被删除：

```
$ 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>
```

主机成功弃用后，可以安全地删除之前运行主控机的主机。

5.2.1.4. 删除 etcd 主机

如果 etcd 主机无法恢复，将其从集群中移除。

在所有 master 主机上执行的步骤

流程

1. 从 etcd 集群中删除其他 etcd 主机。为每个 etcd 节点运行以下命令：

```
# etcdctl3 --endpoints=https://<surviving host IP>:2379
  --cacert=/etc/etcd/ca.crt
  --cert=/etc/etcd/peer.crt
  --key=/etc/etcd/peer.key member remove <failed member ID>
```

2. 在每个 master 上重启 master API 服务：

```
# master-restart api restart-master controller
```

在当前 etcd 集群中执行的步骤

流程

1. 从集群中删除失败的主机：

```
# 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) **remove** 命令需要 etcd ID，而不是主机名。

2. 要确保 etcd 配置在 etcd 服务重启时不使用失败的主机，修改所有剩余的 etcd 主机上的 **/etc/etcd/etcd.conf** 文件，并在 **ETCD_INITIAL_CLUSTER** 变量的值中删除失败主机：

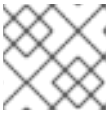
```
# vi /etc/etcd/etcd.conf
```

例如：

```
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
```

成为：

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



注意

不需要重启 etcd 服务，因为失败的主机是使用 **etcdctl** 被删除。

3. 修改 Ansible 清单文件，以反映集群的当前状态，并避免在重新运行 playbook 时出现问题：

```
[OSEv3:children]
masters
nodes
etcd

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

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

4. 如果您使用 Flannel，请修改每个主机上 **/etc/sysconfig/flanneld** 的 **flanneld** 服务配置并删除 etcd 主机：

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

5. 重启 **flanneld** 服务：

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

5.2.2. 创建 master 主机备份

在对 OpenShift Container Platform 基础架构进行任何更改之前执行此备份过程，如系统更新、升级或其他显著修改。定期备份数据，以确保在出错时有最新的数据可用。

OpenShift Container Platform 文件

主（master）实例运行重要的服务，如 API 和控制器。**/etc/origin/master** 目录存储很多重要文件：

- 配置、API、控制器、服务等
- 安装生成的证书
- 所有与云供应商相关的配置
- 密钥及其他身份验证文件，例如，如果使用 htpasswd 时为 **htpasswd**
- 更多

您可以自定义 OpenShift Container Platform 服务，如增加日志级别或使用代理。配置文件存储在 **/etc/sysconfig** 目录中。

因为 master 也是节点，所以备份整个 **/etc/origin** 目录。

流程

重要

您必须在每个 master 节点上执行以下步骤。

1. 创建 pod 定义备份（在此）。.
2. 创建 master 主机配置文件的备份：

```
$ 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/ ${MYBACKUPDIR}/etc/sysconfig/
```

注意

master 配置文件为/etc/origin/master/master-config.yaml。

警告

/etc/origin/master/ca.serial.txt 文件仅对 Ansible 主机清单中列出的第一个 master 生成。如果您弃用了第一个 master 主机，请在进程前将 **/etc/origin/master/ca.serial.txt** 文件复制到其余 master 主机中。

重要

在运行多个 master 的 OpenShift Container Platform 3.11 集群中，其中一个 master 节点在 **/etc/origin/master**、**/etc/etcd/ca** 和 **/etc/etcd/generated_certs** 中包括额外的 CA 证书。这些是应用程序节点和 etcd 节点扩展操作所必需的，且需要在另一个 master 节点上恢复，它应该永久不可用。这些目录默认包含在此处的备份过程中。

3. 在规划备份时需要考虑的其他重要文件包括：

File	描述
/etc/cni/*	容器网络接口配置（如果使用）
/etc/sysconfig/iptables	存储 **iptables** 规则的地方
/etc/sysconfig/docker-storage-setup	**container-storage-setup** 命令的输入文件
/etc/sysconfig/docker	**docker** 配置文件
/etc/sysconfig/docker-network	**docker** 网络配置（如 MTU）
/etc/sysconfig/docker-storage	**docker** 存储配置（由 **container-storage-setup** 生成）
/etc/dnsmasq.conf	**dnsmasq** 的主要配置文件
/etc/dnsmasq.d/*	不同的 **dnsmasq** 配置文件
/etc/sysconfig/flanneld	**flannel** 配置文件（如果使用）
/etc/pki/ca-trust/source/anchors/	添加到系统的证书（例如，外部 registry）

创建这些文件的备份：

```
$ 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/
```

4. 如果软件包被意外删除，或者您需要重新整合 **rpm** 软件包中所含的文件，则系统中安装的 **rhel** 软件包列表会很有用。

注意

如果您使用 Red Hat Satellite 功能（如内容视图或事实存储），请提供正确机制来重新安装缺少的软件包和系统中安装的软件包的历史数据。

要创建系统中当前 **rhel** 软件包的列表：

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

5. 如果您使用前面的步骤，则备份目录中存在以下文件：

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

如果需要，您可以压缩文件以节省空间：

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

要从头开始创建任何这些文件，**openshift-ansible-contrib** 存储库包含 **backup_master_node.sh** 脚本，它将执行前面的步骤。该脚本在运行 脚本的主机上创建一个目录，并复制前面提到的所有文件。

注意

红帽不支持 **openshift-ansible-contrib** 脚本，但可以在架构团队执行测试以确保代码运行正常且安全时作为参考。

您可以使用以下方法在每个 master 主机上运行该脚本：

```
$ 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. 恢复 master 主机备份

在创建了重要 master 主机文件的备份后，如果它们被破坏或意外删除，您可以通过将文件复制到 master，确保文件包含正确的内容并重新启动受影响的服务来恢复文件。

流程

1. 恢复 **/etc/origin/master/master-config.yaml** 文件：

```
# 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
# master-restart api
# master-restart controllers
```

警告

重新启动主控服务可能会导致停机。但是，您可以从高可用性负载均衡器池中移除 master 主机，然后执行恢复操作。正确恢复该服务后，您可以将 master 主机重新添加到负载均衡器池中。

注意

对受影响的实例执行完全重启，以恢复 **iptables** 配置。

2. 如果因为缺少软件包而无法重启 OpenShift Container Platform，请重新安装软件包。

   a. 获取当前安装的软件包列表：

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

   b. 查看软件包列表之间的区别：

```
$ diff /tmp/current_packages.txt ${MYBACKUPDIR}/packages.txt

> ansible-2.4.0.0-5.el7.noarch
```

   c. 重新安装缺少的软件包：

```
# yum reinstall -y <packages> (1)
```

   (1) 将 **<packages>** 替换为软件包列表之间不同的软件包。

3. 通过将证书复制到 **/etc/pki/ca-trust/source/anchors/** 目录并执行 **update-ca-trust** 来恢复系统证书：

```
$ MYBACKUPDIR=*/backup/$(hostname)/$(date +%Y%m%d)*
$ sudo cp ${MYBACKUPDIR}/etc/pki/ca-trust/source/anchors/<certificate> /etc/pki/ca-trust/source/anchors/ (1)
$ sudo update-ca-trust
```

(1) 将 **<certificate>** 替换为要恢复的系统证书的文件名。

注意

始终确保返回文件时恢复用户 ID 和组 ID，以及 **SELinux** 上下文。

5.3. 节点主机任务

5.3.1. 弃用节点主机

无论弃用基础架构节点或应用程序节点，步骤都是一样的。

先决条件

确保有足够的容量，以便将现有 pod 从节点集合中移除。只有在删除基础架构节点时，才建议删除基础架构节点。

流程

1. 列出所有可用的节点，以查找要弃用的节点：

```
$ 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
```

例如，本节弃用 **ocp-infra-node-b7pl** 基础架构节点。

2. 描述节点及其运行的服务：

```
$ 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>
```

上面的输出显示节点正在运行两个 pod： **router-1-vzlzq** 和 **docker-registry-1-5szjs**。有两个基础架构节点可用于迁移这两个 pod。

注意

上述集群是一个高可用性集群，这意味着 **router** 和 **docker-registry** 服务在所有基础架构节点上运行。

3. 将节点标记为不可调度并撤离其所有 pod：

```
$ 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
```

如果 pod 已附加了本地存储（例如 **EmptyDir**），则必须提供 **--delete-local-data** 选项。通常，在生产环境中运行的 pod 应该只将本地存储用于临时或缓存文件，但不适用于任何重要或持久的。对于常规存储，应用程序应使用对象存储或持久性卷。在这种情况下，**docker-registry** pod 的本地存储为空，因为对象存储被用来存储容器镜像。

注意

以上操作会删除节点上运行的现有 pod。然后，根据复制控制器创建新的 pod。

通常，每个应用都应该使用部署配置进行部署，这将利用复制控制器创建 pod。

oc adm drain 不会删除任何不是镜像 pod 的裸机 pod（镜像 pod，或由 **ReplicationController**、**ReplicaSet**、**DaemonSet**、**StatefulSet** 或作业进行管理的 pod）。要做到这一点，需要 **--force** 选项。请注意，在此操作过程中，不会在其他节点上重新创建裸机 pod，数据可能会丢失。

以下示例显示了 registry 的复制控制器的输出：

```
$ 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
```

输出底部的事件显示有关新 pod 创建的信息。因此，当列出所有 pod 时：

```
$ 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. 现在，在已弃用节点上运行的 **docker-registry-1-5szjs** 和 **router-1-vzlzq** pod 不再可用。相反，创建了两个新 pod：**docker-registry-1-dprp5** 和 **router-1-2gshr**。如上所示，新的路由器 Pod 是 **router-1-2gshr**，但处于 **Pending** 状态。这是因为每个节点只能在一个路由器中运行，并绑定到主机的端口 80 和 443。

5. 观察新创建的 registry pod 时，以下示例显示了 **ocp-infra-node-rghb** 节点上已创建了 pod，它与弃用节点的不同：

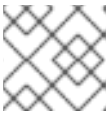
```
$ oc describe pod docker-registry-1-dprp5
Name:   docker-registry-1-dprp5
Namespace:  default
Security Policy: hostnetwork
Node:   ocp-infra-node-rghb/10.156.0.10
...
```

弃用基础架构和应用程序节点的唯一区别在于，在基础架构节点被撤离后，如果没有计划替换该节点，在基础架构节点上运行的服务可以缩减：

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

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

6. 现在，每个基础架构节点只运行一个 pod 中的一类：

```
$ 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
```



注意

要提供完整的高可用性集群，至少有三个基础架构节点应当始终可用。

7. 验证节点上的调度是否已禁用：

```
$ 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
```

节点没有包含任何 pod:

```
$ 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. 从 **/etc/haproxy/haproxy.cfg** 配置文件中的 **backend** 部分删除基础架构实例：

```
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. 然后，重新启动 **haproxy** 服务。

```
$ sudo systemctl restart haproxy
```

10. 在所有 pod 都被逐出后，使用以下命令从集群中删除节点：

```
$ 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
```

注意

如需有关撤离和排空 pod 或节点的更多信息，请参阅节点维护部分。

5.3.1.1. 替换节点主机

如果需要添加节点以替代已弃用节点，请按照 将主机添加到现有集群 部分。

5.3.2. 创建节点主机备份

创建节点主机的备份与备份 master 主机的用例不同。由于 master 主机包含许多重要文件，因此强烈建议创建备份。但是，节点的性质是，在故障切换时，任何特殊操作都会在节点上复制，它们通常不包含运行环境所需的数据。如果节点的备份包含运行环境所需的一些内容，则建议使用创建备份。

备份过程会在基础架构进行任何更改之前执行，如系统更新、升级或任何其他显著的修改。备份应定期执行，确保在出现故障时最新的数据可用。

OpenShift Container Platform 文件

节点实例以 pod 的形式运行应用程序，这些 pod 基于容器。**/etc/origin/** 和 **/etc/origin/node** 目录包括了重要文件，例如：

- 节点服务的配置
- 安装生成的证书
- 与云供应商相关的配置
- 密钥和其他身份验证文件，如 **dnsmasq** 配置

可以自定义 OpenShift Container Platform 服务来提高日志级别、使用代理等，并且配置文件存储在 **/etc/sysconfig** 目录中。

流程

1. 创建节点配置文件的备份：

```
$ 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 使用在规划备份策略时必须考虑的特定文件，包括：

File	描述
/etc/cni/*	容器网络接口配置（如果使用）
/etc/sysconfig/iptables	存储 **iptables** 规则的地方
/etc/sysconfig/docker-storage-setup	**container-storage-setup** 命令的输入文件
/etc/sysconfig/docker	**docker** 配置文件
/etc/sysconfig/docker-network	**docker** 网络配置（如 MTU）
/etc/sysconfig/docker-storage	**docker** 存储配置（由 **container-storage-setup** 生成）
/etc/dnsmasq.conf	**dnsmasq** 的主要配置文件
/etc/dnsmasq.d/*	不同的 **dnsmasq** 配置文件
/etc/sysconfig/flanneld	**flannel** 配置文件（如果使用）
/etc/pki/ca-trust/source/anchors/	添加到系统的证书（例如，外部 registry）

创建这些文件：

```
$ 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. 如果意外删除软件包，或者应该恢复 **rpm** 软件包中包含的文件，系统中安装的 **rhel** 软件包列表很有用。

注意

如果使用 Red Hat Satellite 功能（如内容视图或事实存储），请提供正确机制来重新安装缺少的软件包和系统中安装的软件包的历史数据。

要创建系统中当前 **rhel** 软件包的列表：

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

4. 备份目录中现在应存在以下文件：

```
$ 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
```

如果需要，可以压缩文件以节省空间：

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

要从头开始创建任何这些文件，**openshift-ansible-contrib** 存储库包含 **backup_master_node.sh** 脚本，它将执行前面的步骤。该脚本在运行 脚本的主机上创建一个目录，并复制前面提到的所有文件。

注意

红帽不支持 **openshift-ansible-contrib** 脚本，但可以在架构团队执行测试以确保代码运行正常且安全时作为参考。

可以使用以下方法在每个 master 主机上执行该脚本：

```
$ 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. 恢复节点主机备份

创建重要节点主机文件的备份后，如果它们被破坏或意外删除，您可以通过复制 文件来恢复文件，确保该文件包含正确的内容并重新启动受影响的服务。

流程

1. 恢复 **/etc/origin/node/node-config.yaml** 文件：

```
# 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
# reboot
```

警告

重新启动服务可能会导致停机。如需有关如何简化流程的提示，请参阅节点维护。

注意

对受影响的实例执行完全重启，以恢复 **iptables** 配置。

1. 如果因为缺少软件包而无法重启 OpenShift Container Platform，请重新安装软件包。
   a. 获取当前安装的软件包列表：

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

   b. 查看软件包列表之间的区别：

```
$ diff /tmp/current_packages.txt ${MYBACKUPDIR}/packages.txt

> ansible-2.4.0.0-5.el7.noarch
```

   c. 重新安装缺少的软件包：

```
# yum reinstall -y <packages> 1
```

   1 将 **<packages>** 替换为软件包列表之间不同的软件包。

2. 通过将证书复制到 **/etc/pki/ca-trust/source/anchors/** 目录并执行 **update-ca-trust** 来恢复系统证书：

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

将 **<certificate>** 替换为要恢复的系统证书的文件名。

注意

始终确保在文件返回时恢复正确的用户 ID 和组 ID，以及 **SELinux** 上下文。

5.3.4. 节点维护及后续步骤

有关各种节点管理选项，请参阅管理节点或管理 pod 主题。它们是：

- 将节点标记为 Unschedulable 或 Schedulable
- 在节点上清空 pod
- 设置 Pod Disruption 预算

节点可以保留一部分资源供特定组件使用。这包括 kubelet、kube-proxy、Docker 或其他剩余的系统组件，如 **sshd** 和 **NetworkManager**。如需更多信息，请参阅 Cluster Administrator 指南中的 Allocating node resources 部分。

5.4. ETCD 任务

5.4.1. etcd 备份

etcd 是所有对象定义的键值存储，以及持久 master 状态。其他组件会监视更改，然后进入所需的状态。

3.5 之前的 OpenShift Container Platform 版本使用 etcd 版本 2(v2)，而 3.5 及更新版本使用版本 3(v3)。两个版本的 etcd 之间的数据模型不同。etcd v3 可以使用 v2 和 v3 数据模型，而 etcd v2 只能使用 v2 数据模型。在 etcd v3 服务器中，v2 和 v3 数据存储可以并行存在，并相互独立。

对于 v2 和 v3 操作，您可以使用 **ETCDCTL_API** 环境变量来使用正确的 API：

```
$ etcdctl -v
etcdctl version: 3.2.28
API version: 2

$ ETCDCTL_API=3 etcdctl version
etcdctl version: 3.2.28
API version: 3.2
```

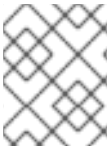
如需有关如何迁移到 v3 的信息，请参阅 OpenShift Container Platform 3.7 文档中的迁移 etcd 数据（v2 到 v3）部分。

在 OpenShift Container Platform 版本 3.10 及更新版本中，您可以在单独的主机上安装 etcd，或作为静态 pod 在 master 主机上运行。如果没有指定独立的 etcd 主机，etcd 会作为静态 pod 在 master 主机上运行。因此，如果您使用静态 pod，备份过程会有所不同。

etcd 备份过程由两个不同的步骤组成：

- 配置备份：包含所需的 etcd 配置和证书
- 数据备份：包含 v2 和 v3 数据模型。

您可以在任何已连接到 etcd 集群的主机上执行数据备份过程，其中提供了正确的证书，以及安装 **etcdctl** 工具的位置。



注意

备份文件必须复制到外部系统，最好在 OpenShift Container Platform 环境之外，然后进行加密。

请注意，etcd 备份仍具有当前存储卷的所有引用。恢复 etcd 时，OpenShift Container Platform 开始在节点上启动以前的 pod 并重新连接相同的存储。此过程与从集群中删除节点的过程不同，并在其位置添加一个新节点。附加到该节点的任何节点上的任何节点都会重新附加到 pod。

5.4.1.1. 备份 etcd

当备份 etcd 时，您必须备份 etcd 配置文件和 etcd 数据。

5.4.1.1.1. 备份 etcd 配置文件

要保留的 etcd 配置文件都存储在运行 etcd 的实例的 **/etc/etcd** 目录中。这包括 etcd 配置文件 (**/etc/etcd/etcd.conf**)和集群通信所需的证书。所有这些文件都是在安装时由 Ansible 安装程序生成的。

流程
对于集群的每个 etcd 成员，备份 etcd 配置。

```
$ ssh master-0 1
# mkdir -p /backup/etcd-config-$(date +%Y%m%d)/
# cp -R /etc/etcd/ /backup/etcd-config-$(date +%Y%m%d)/
```

1 将 **master-0** 替换为 etcd 成员的名称。

注意

每个 etcd 集群成员上的证书和配置文件是唯一的。

5.4.1.1.2. 备份 etcd 数据

先决条件

注意

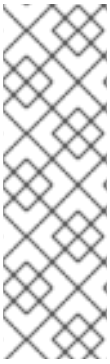
OpenShift Container Platform 安装程序创建别名，以避免为 etcd v2 任务输入名为 **etcdctl2** 的所有标志，以及用于 etcd v3 任务的 **etcdctl3**。

但是，**etcdctl3** 别名不会向 **etcdctl** 命令提供完整的端点列表，因此您必须指定 **--endpoints** 选项并列出所有端点。

备份 etcd 之前：

- **etcdctl** 二进制文件必须可用，或者在容器化安装中，**rhel7/etcd** 容器必须可用。
- 确保 OpenShift Container Platform API 服务正在运行。
- 确保与 etcd 集群的连接（端口 2379/tcp）。
- 确保正确的证书以连接到 etcd 集群。

流程



注意

虽然 **etcdctl backup** 命令用于执行备份，etcd v3 没有 *备份* 的概念。您可以使用 **etcdctl snapshot save** 命令对一个实时成员进行 *快照*，或从 etcd 数据目录中复制 **member/snap/db** 文件。

etcdctl backup 命令重写了备份中所含的一些元数据，特别是节点 ID 和集群 ID，这意味着在备份中，节点会丢失它的以前的身份。要从备份重新创建集群，您可以创建一个新的单节点集群，然后将其他节点的其余部分添加到集群中。元数据被重写以防止新节点加入现有集群。

备份 etcd 数据：

重要

从以前的 OpenShift Container Platform 版本升级的集群可能包含 v2 数据存储。备份所有 etcd 数据存储。

1. 从静态 pod 清单获取 etcd 端点 IP 地址：

```
$ export ETCD_POD_MANIFEST="/etc/origin/node/pods/etcd.yaml"
```

```
$ export ETCD_EP=$(grep https ${ETCD_POD_MANIFEST} | cut -d '/' -f3)
```

2. 以管理员身份登录：

```
$ oc login -u system:admin
```

3. 获取 etcd pod 名称：

```
$ export ETCD_POD=$(oc get pods -n kube-system | grep -o -m 1 '^master-etcd\S*')
```

4. 进入 **kube-system** 项目：

```
$ oc project kube-system
```

5. 在 pod 中生成 etcd 数据快照并将其保存在本地：

```
$ oc exec ${ETCD_POD} -c etcd -- /bin/bash -c "ETCDCTL_API=3 etcdctl \
    --cert /etc/etcd/peer.crt \
    --key /etc/etcd/peer.key \
```

```
    --cacert /etc/etcd/ca.crt \
    --endpoints $ETCD_EP \
    snapshot save /var/lib/etcd/snapshot.db" 1
```

1 您必须将快照写入 **/var/lib/etcd/** 下的目录中。

5.4.2. 恢复 etcd

5.4.2.1. 恢复 etcd 配置文件

如果 etcd 主机已损坏，并且 **/etc/etcd/etcd.conf** 文件丢失，请按照以下步骤恢复该文件：

1. 访问 etcd 主机：

```
$ ssh master-0 1
```

1 将 **master-0** 替换为 etcd 主机的名称。

2. 将备份 **etcd.conf** 文件复制到 **/etc/etcd/** 中：

```
# cp /backup/etcd-config-<timestamp>/etcd/etcd.conf /etc/etcd/etcd.conf
```

3. 在文件中设置所需的权限和 selinux 上下文：

```
# restorecon -RvF /etc/etcd/etcd.conf
```

在本例中，备份文件存储在 **/backup/etcd-config-<timestamp>/etcd/etcd.conf** 路径中，其中可用作外部 NFS 共享、S3 存储桶或其他存储解决方案。

恢复 etcd 配置文件后，您必须重启静态 pod。这是在恢复 etcd 数据后完成的。

5.4.2.2. 恢复 etcd 数据

在静态 pod 中恢复 etcd 前：

- **etcdctl** 二进制文件必须可用，或者在容器化安装中，**rhel7/etcd** 容器必须可用。
  您可以运行以下命令，使用 etcd 软件包安装 **etcdctl** 二进制文件：

```
# yum install etcd
```

该软件包还会安装 systemd 服务。禁用并屏蔽该服务，使其在 etcd 在静态 pod 中运行时不会作为 systemd 服务运行。通过禁用并屏蔽该服务，确保您不会意外启动该服务，并防止它在重启系统时自动重启该服务。

```
# systemctl disable etcd.service
```

```
# systemctl mask etcd.service
```

在静态 pod 上恢复 etcd:

1. 如果 pod 正在运行，通过将 pod 清单 YAML 文件移到另一个目录中来停止 etcd pod：

```
# mkdir -p /etc/origin/node/pods-stopped
```

```
# mv /etc/origin/node/pods/etcd.yaml /etc/origin/node/pods-stopped
```

2. 移动所有旧数据：

```
# mv /var/lib/etcd /var/lib/etcd.old
```

   您可以使用 etcdctl 在恢复 pod 的节点中重新创建数据。

3. 将 etcd 快照恢复到 etcd pod 的挂载路径：

```
# export ETCDCTL_API=3
```

```
# etcdctl snapshot restore /etc/etcd/backup/etcd/snapshot.db \
  --data-dir /var/lib/etcd/ \
  --name ip-172-18-3-48.ec2.internal \
  --initial-cluster "ip-172-18-3-48.ec2.internal=https://172.18.3.48:2380" \
  --initial-cluster-token "etcd-cluster-1" \
  --initial-advertise-peer-urls https://172.18.3.48:2380 \
  --skip-hash-check=true
```

   从 ***etcd.conf*** 文件获取集群的适当值。

4. 在数据目录中设置所需的权限和 selinux 上下文：

```
# restorecon -RvF /var/lib/etcd/
```

5. 通过将 pod 清单 YAML 文件移到所需目录中来重启 etcd pod：

```
# mv /etc/origin/node/pods-stopped/etcd.yaml /etc/origin/node/pods/
```

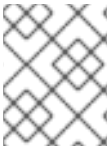
5.4.3. 替换 etcd 主机

要替换 etcd 主机，扩展 etcd 集群，然后删除主机。如果替换过程中会丢失 etcd 主机，这个过程可确保保留仲裁。

警告

etcd 集群必须在替换操作过程中维护仲裁。这意味着，至少有一个主机始终处于操作状态。

如果在 etcd 集群维护仲裁时执行主机替换操作，集群操作通常不受影响。如果需要复制大量的 etcd 数据，有些操作可能会减慢。

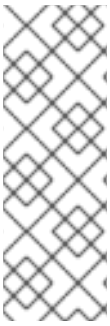


注意

在启动涉及 etcd 集群的任何流程前，您必须备份 etcd 数据和配置文件，以便在流程失败时可以恢复集群。

5.4.4. 扩展 etcd

您可以通过在 etcd 主机中添加更多资源或通过添加更多 etcd 主机来纵向扩展 etcd 集群。



注意

由于 etcd 使用的投票系统，集群必须始终包含奇数个成员数。

拥有奇数 etcd 主机的集群可考虑容错。etcd 主机数量为奇数不会改变仲裁所需的数量，但会增加故障的容错能力。例如，对于包含三个成员的群集，仲裁为 2，保留一个失败容错能力。这可确保集群在两个成员健康时继续运行。

建议具有三个 etcd 主机的生产环境级别的集群。

新主机需要一个全新的 Red Hat Enterprise Linux 7 专用主机。etcd 存储应位于 SSD 磁盘上，以实现最佳性能和在 **/var/lib/etcd** 中挂载的专用磁盘上。

先决条件

1. 在添加新的 etcd 主机前，备份 etcd 配置和数据以防止数据丢失。
2. 检查当前的 etcd 集群状态，以避免添加新主机到不健康的集群。运行这个命令：

```
# 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. 在运行 **scaleup** playbook 前，请确保新主机已注册到正确的红帽软件频道：

```
# 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 托管在 **rhel-7-server-extras-rpms** 软件频道中。

4. 确保从 etcd 集群中删除所有未使用的 etcd 成员。这必须在运行 **scaleup** playbook 之前完成。

a. 列出 etcd 成员：

```
# etcdctl --cert="/etc/etcd/peer.crt" --key="/etc/etcd/peer.key" \
  --cacert="/etc/etcd/ca.crt" --endpoints=ETCD_LISTEN_CLIENT_URLS member list -w
table
```

复制未使用的 etcd 成员 ID（如果适用）。

b. 通过使用以下命令指定其 ID 来删除未使用的成员：

```
# etcdctl --cert="/etc/etcd/peer.crt" --key="/etc/etcd/peer.key" \
  --cacert="/etc/etcd/ca.crt" --endpoints=ETCD_LISTEN_CLIENT_URL member remove
UNUSED_ETCD_MEMBER_ID
```

5. 在当前 etcd 节点上升级 etcd 和 iptables：

```
# yum update etcd iptables-services
```

6. 为 etcd 主机备份 **/etc/etcd** 配置。

7. 如果新的 etcd 成员也是 OpenShift Container Platform 节点，将所需的主机数量添加到集群中。

8. 此流程的其余部分假定您添加了一台主机，但是如果您添加多个主机，请在每个主机上执行所有步骤。

5.4.4.1. 使用 Ansible 添加新 etcd 主机

流程

1. 在 Ansible 清单文件中，创建一个名为 **[new_etcd]** 的新组，再添加新主机。然后，将 **new_etcd** 组添加为 **[OSEv3]** 组的子组：

```
[OSEv3:children]
masters
nodes
etcd
new_etcd (1)

... [OUTPUT ABBREVIATED] ...

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

[new_etcd] (2)
etcd0.example.com (3)
```

(1)(2)(3)添加以下行：

注意

将旧的 **etcd 主机**条目替换为清单文件中的新 **etcd** 主机条目。在替换旧的 **etcd 主机**时，必须创建一个 **/etc/etcd/ca/** 目录的副本。另外，您可以在扩展 **etcd 主机**前重新部署 etcd ca 和 certs。

2. 在安装 OpenShift Container Platform 并托管 Ansible 清单文件的主机中，切换到 playbook 目录并运行 etcd **scaleup** playbook:

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

3. 在 playbook 运行后，通过将新 etcd 主机从 **[new_etcd]** 组移到 **[etcd]** 组来修改清单文件，使其反映当前的状态：

```
[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. 如果使用 Flannel，修改位于 **/etc/sysconfig/flanneld** 主机上的 **flanneld** 服务配置，使其包含新的 etcd 主机：

```
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. 重启 **flanneld** 服务：

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

5.4.4.2. 手动添加新 etcd 主机

如果您没有以静态 pod 用户身份在 master 节点上运行 etcd，您可能需要添加另一个 etcd 主机。

流程

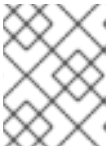
修改当前的 etcd 集群

要创建 etcd 证书，请运行 **openssl** 命令，将值替换为您环境中的值。

1. 创建一些环境变量：

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

export CN=$NEW_ETCD_HOSTNAME
```

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



注意

用作 **etcd_v3_ca_*** 的自定义 **openssl** 扩展包括 $SAN 环境变量作为 **subjectAltName**。如需更多信息，请参阅 **/etc/etcd/ca/openssl.cnf**。

2. 创建用于存储配置和证书的目录：

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

3. 创建服务器证书请求并为其签名：（*server.csr* 和 *server.crt*）

```
# 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. 创建对等证书请求并将其签名：（*peer.csr* 和 *peer.crt*）

```
# 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. 从当前节点复制当前的 etcd 配置和 **ca.crt** 文件，作为稍后修改的示例：

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

6. 当仍处于存活的 etcd 主机上时，将新主机添加到集群中。要在集群中添加额外的 etcd 成员，您必须首先调整第一个成员的 **peerURLs** 值中的默认 **localhost** peer：

   a. 使用 **member list** 命令获取第一个成员的成员 ID：

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

```
    --peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://172.18.0.75:2379"
\ 1
    member list
```

1 请确定在 **--peers** 参数值中只指定活跃的 etcd 成员的 URL。

b. 获取 etcd 侦听集群对等点的 IP 地址：

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

c. 通过传递从上一步中获取的成员 ID 和 IP 地址，使用 **etcdctl member update** 命令更新 **peerURLs** 的值：

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
    --key-file=/etc/etcd/peer.key \
    --ca-file=/etc/etcd/ca.crt \
    --peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://172.18.0.75:2379"
\
    member update 511b7fb6cc0001 https://172.18.1.18:2380
```

d. 重新运行 **member list** 命令，并确保对等 URL 不再包含 **localhost**。

7. 将新主机添加到 etcd 集群。请注意，新主机尚未配置，因此在您配置新主机之前，状态将保持为**未启动**。

警告

您必须添加每个成员，并一次性使他们在线。将每个额外成员添加到集群时，您必须调整当前对等点的 **peerURLs** 列表。**peerURLs** 列表会针对每个添加的成员增长一个。**etcdctl member add** 命令输出在 *etcd.conf* 文件中设置的值，以添加每个成员，如下说明所述。

```
# etcdctl -C https://${CURRENT_ETCD_HOST}:2379 \
  --ca-file=/etc/etcd/ca.crt     \
  --cert-file=/etc/etcd/peer.crt     \
  --key-file=/etc/etcd/peer.key member add ${NEW_ETCD_HOSTNAME}
https://${NEW_ETCD_IP}:2380 1

Added member named 10.3.9.222 with ID 4e1db163a21d7651 to cluster

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"
```

1 在这一行中，**10.3.9.222** 是 etcd 成员的标签。您可以指定主机名、IP 地址或简单名称。

8. 更新示例 **${PREFIX}/etcd.conf** 文件。

   a. 将以下值替换为上一步中生成的值：

      - ETCD_NAME
      - ETCD_INITIAL_CLUSTER
      - ETCD_INITIAL_CLUSTER_STATE

   b. 使用上一步中输出中的新主机 IP 修改以下变量：您可以使用 **${NEW_ETCD_IP}** 作为值。

```
ETCD_LISTEN_PEER_URLS
ETCD_LISTEN_CLIENT_URLS
ETCD_INITIAL_ADVERTISE_PEER_URLS
ETCD_ADVERTISE_CLIENT_URLS
```

   c. 如果您之前使用 member 系统作为 etcd 节点，您必须覆盖 */etc/etcd/etcd.conf* 文件中的当前值。

   d. 检查文件中的语法错误或缺少 IP 地址，否则 etcd 服务可能会失败：

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

9. 在托管安装文件的节点上，更新 */etc/ansible/hosts* 清单文件中的 **[etcd]** 主机组。删除旧的 etcd 主机并添加新主机。

10. 创建一个包含证书、示例配置文件和 **ca** 的 **tgz** 文件，并将其复制到新主机上：

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

修改新的 etcd 主机

1. 安装 **iptables-services** 以提供 iptables 工具为 etcd 打开所需的端口：

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

2. 创建 **OS_FIREWALL_ALLOW** 防火墙规则以允许 etcd 进行通信：

   - 客户端的端口 2379/tcp
   - 端口 2380/tcp 用于对等通信

```
# 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
```

注意

在本例中，创建了一个新的链 **OS_FIREWALL_ALLOW**，这是 OpenShift Container Platform 安装程序用于防火墙规则的标准命名。

警告

如果环境托管在 IaaS 环境中，请修改实例的安全组，以允许传入到这些端口的流量。

3. 安装 etcd：

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

确保安装了 **etcd-2.3.7-4.el7.x86_64** 或更高的版本，

4. 通过删除 etcd pod 定义来确保 etcd 服务没有运行：

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
```

5. 删除所有 etcd 配置和数据：

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

6. 提取证书和配置文件：

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

7. 在新主机上启动 etcd:

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

8. 验证主机是否是集群的一部分以及当前的集群健康状况：

- 如果使用 v2 etcd api，请运行以下命令：

```
# 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
```

- 如果使用 v3 etcd api，请运行以下命令：

```
# 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
```

修改每个 OpenShift Container Platform master

1. 修改每个 master 上 **/etc/origin/master/master-config.yaml** 文件的 **etcClientInfo** 部分中的 master 配置。将新的 etcd 主机添加到 OpenShift Container Platform 用来存储数据的 etcd 服务器列表中，并删除所有失败的 etcd 主机：

```
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. 重启 master API 服务：

- 在每个 master 上：

```
# master-restart api
# master-restart controllers
```

警告

etcd 节点的数量必须是奇数，因此您必须至少添加两个主机。

3. 如果使用 Flannel，修改每个 OpenShift Container Platform 主机的 **/etc/sysconfig/flanneld** 服务的 **flanneld** 服务配置，使其包含新的 etcd 主机：

```
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
```

4. 重启 **flanneld** 服务：

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

5.4.5. 删除 etcd 主机

如果 etcd 主机无法恢复，将其从集群中移除。

在所有 master 主机上执行的步骤

流程

1. 从 etcd 集群中删除其他 etcd 主机。为每个 etcd 节点运行以下命令：

```
# etcdctl3 --endpoints=https://<surviving host IP>:2379
  --cacert=/etc/etcd/ca.crt
  --cert=/etc/etcd/peer.crt
  --key=/etc/etcd/peer.key member remove <failed member ID>
```

2. 在每个 master 上重启 master API 服务：

```
# master-restart api restart-master controller
```

在当前 etcd 集群中执行的步骤

流程

1. 从集群中删除失败的主机：

```
# 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 **remove** 命令需要 etcd ID，而不是主机名。

2. 要确保 etcd 配置在 etcd 服务重启时不使用失败的主机，修改所有剩余的 etcd 主机上的 **/etc/etcd/etcd.conf** 文件，并在 **ETCD_INITIAL_CLUSTER** 变量的值中删除失败主机：

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

例如：

```
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
```

成为：

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

注意

不需要重启 etcd 服务，因为失败的主机是使用 **etcdctl** 被删除。

3. 修改 Ansible 清单文件，以反映集群的当前状态，并避免在重新运行 playbook 时出现问题：

```
[OSEv3:children]
masters
nodes
etcd

... [OUTPUT ABBREVIATED] ...

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

4. 如果您使用 Flannel，请修改每个主机上 **/etc/sysconfig/flanneld** 的 **flanneld** 服务配置并删除 etcd 主机：

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

5. 重启 **flanneld** 服务：

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

第 6 章 项目级别任务

6.1. 备份项目

对所有相关数据创建备份涉及导出所有重要信息，然后恢复到一个新项目中。

重要

因为 **oc get all** 命令只返回某些项目资源，所以您必须单独备份其他资源，包括 PVC 和 Secret，如以下步骤所示。

流程

1. 列出要备份的项目数据：

```
$ oc get all
```

输**出示例**

```
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. 将项目对象导出到 **project.yaml** 文件。

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

3. 在项目中导出其他对象，如角色绑定、secret、服务帐户和持久性卷声明。
您可以使用以下命令导出项目中的所有命名空间对象：

```
$ for object in $(oc api-resources --namespaced=true -o name)
do
  oc get -o yaml --export $object > $object.yaml
done
```

请注意，无法导出某些资源，并显示 **MethodNotAllowed** 错误。

4. 一些导出的对象可以依赖特定元数据或对项目中唯一 ID 的引用。这是对重新创建的对象可用性的一个限制。
使用**镜像流**时，**deploymentconfig** 的 **image** 参数可以指向已恢复环境中不存在的内部 registry 中镜像的特定 **sha** checksum。例如，以 **oc new-app centos/ruby-22-centos7~https://github.com/sclorg/ruby-ex.git** 运行示例 "ruby-ex" 作为 oc new-app centos7~https://github.com/sclorg/ruby-ex.git 创建一个 **镜像流 ruby-ex** 来托管该镜像：

```
$ oc get dc ruby-ex -o jsonpath="{.spec.template.spec.containers[].image}"
10.111.255.221:5000/myproject/ruby-
ex@sha256:880c720b23c8d15a53b01db52f7abdcbb2280e03f686a5c8edfef1a2a7b21cee
```

如果导入使用 **oc get --export** 导出的 **deploymentconfig**，在镜像不存在时会失败。

6.2. 恢复项目

要恢复项目，请创建新项目，然后运行 **oc create -f <file_name>** 来恢复所有导出的文件。

流程

1. 创建项目：

```
$ oc new-project <project_name> 1
```

1 此 **<project_name>** 值必须与备份的项目的名称匹配。

2. 导入项目对象：

```
$ oc create -f project.yaml
```

3. 导入您在备份项目时导出的任何其他资源，如角色绑定、secret、服务帐户和持久性卷声明：

```
$ oc create -f <object>.yaml
```

如果某些资源需要存在另一个对象，则可能无法导入。如果发生了这种情况，请查看错误消息以确定必须首先导入哪些资源。

警告

某些资源（如 pod 和默认服务帐户）可能无法创建。

6.3. 备份持久性卷声明

您可以从容器内部将持久数据同步到服务器。

重要

根据托管 OpenShift Container Platform 环境的供应商，也可以为备份和恢复目的启动第三方快照服务。因为 OpenShift Container Platform 没有能够启动这些服务，所以本指南并不描述了这些步骤。

如需特定应用程序的正确备份流程，请参阅任何产品文档。例如，复制 mysql 数据目录本身并不会创建可用的备份。反之，运行关联的应用程序的特定备份过程，然后同步任何数据。这包括使用 OpenShift Container Platform 托管平台提供的快照解决方案。

流程

1. 查看项目和 pod：

```
$ 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. 描述所需的 pod，以查找持久性卷目前使用的卷：

```
$ 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://201f3e55b373641eb36945d723e1e212ecab847311109b5cee1fd0109424217a
   Image:  docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20dc9ff6f350436f935
968b0c83fcb98a7a8c381a
   Image ID:  docker-pullable://docker-
registry.default.svc:5000/test/demo@sha256:0a9f2487a0d95d51511e49d20dc9ff6f350436f935
```

```
968b0c83fcb98a7a8c381a
    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...
```

此输出显示持久性数据位于 **/opt/app-root/src/uploaded** 目录中。

3. 在本地复制数据：

```
$ 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
```

ocp_sop.txt 文件下载到本地系统，以通过备份软件或其他备份机制备份。

注意

如果 pod 启动而无需使用 **pvc**，但稍后您决定需要 **pvc**，则可以使用前面的步骤。您可以保留数据，然后使用剩余的进程填充新存储。

6.4. 恢复持久性卷声明

您可以恢复您备份的持久性卷声明(PVC)数据。您可以删除该文件，然后将文件放回预期的位置，或迁移持久性卷声明。如果您需要移动存储或在灾难场景中迁移，当后端存储不再存在时，可能会迁移。

有关特定应用程序的正确恢复步骤，请参阅任何产品文档。

6.4.1. 将文件恢复到现有 PVC

流程

1. 删除该文件：

```
$ 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. 替换包含 pvc 中文件的 rsync 备份的服务器中的文件：

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

3. 使用 **oc rsh** 连接到 pod 并查看目录的内容，验证该文件是否在 pod 上恢复：

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

6.4.2. 将数据恢复到新 PVC

下列步骤假定已创建新的 **pvc**。

流程

1. 覆盖当前定义的 **claim-name**:

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

2. 验证 pod 是否在使用新 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@sha256:0a9f2487a0d95d51511e49d20dc9ff6f350436f935
968b0c83fcb98a7a8c381a
    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. 现在，部署配置使用新的 **pvc**，运行 **oc rsync** 将文件放在新的 **pvc** 中：

```
$ 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. 使用 **oc rsh** 连接到 pod 并查看目录的内容，验证该文件是否在 pod 上恢复：

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

6.5. 修剪镜像和容器

如需有关修剪收集的数据和旧版本对象的信息，请参阅修剪资源。

第 7 章 DOCKER 任务

OpenShift Container Platform 使用容器引擎（CRI-O 或 Docker）来运行由任意数量的容器组成的 pod 中的应用程序。

作为集群管理员，有时容器引擎需要一些额外的配置，才能高效地运行 OpenShift Container Platform 安装元素。

7.1. 增加容器存储

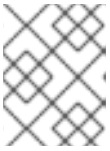
增加可用存储量可确保继续部署，而不会造成任何中断。为此，必须提供一个包含相应可用容量的空闲容量的空闲分区。

7.1.1. 清空节点

流程

1. 从 master 实例或集群管理员，允许从该节点驱除任何 pod，并禁用该节点上调度其他 pod:

```
$ 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
```



注意

如果存在运行没有迁移的本地卷的容器，请运行以下命令： **oc adm drain ${NODE} --ignore-daemonsets --delete-local-data**。

2. 列出节点上的 pod，以验证是否已移除它们：

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

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

NAME      READY     STATUS    RESTARTS   AGE
```

3. 对每个节点重复前面的两个步骤。

如需有关撤离和排空 pod 或节点的更多信息，请参阅节点维护。

7.1.2. 增加存储

您可以通过两种方式来增加 Docker 存储：附加新磁盘或扩展现有磁盘。

使用新磁盘增加存储

先决条件

- 需要更多存储的现有实例都必须有新磁盘。在以下步骤中，原始磁盘被标记为 **/dev/xvdb**，新磁盘被标记为 **/dev/xvdd**，如 */etc/sysconfig/docker-storage-setup* 文件所示：

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

注意

此过程可能因底层的 OpenShift Container Platform 基础架构而异。

流程

1. 停止 **docker** ：

```
# systemctl stop docker
```

2. 通过删除 pod 定义并重启主机来停止节点服务：

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
```

3. 运行 **docker-storage-setup** 命令以扩展与容器存储关联的卷组和逻辑卷：

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

   a. 有关 **精简池** 设置，您应该看到以下输出并继续下一步：

```
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
```

   b. 对于使用 Overlay2 文件系统的 **XFS** 设置，在以上输出中显示的增加将不可见。
   您必须执行以下步骤来扩展和增大 XFS 存储：

      i. 运行 **lvextend** 命令，以增大逻辑卷以使用卷组中的所有可用空间：

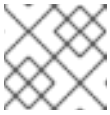
```
# lvextend -r -l +100%FREE /dev/mapper/docker_vol-dockerlv
```

注意

如果要求使用较少的空间，请相应地选择 **FREE** 百分比。

      ii. 运行 **xfs_growfs** 命令增大文件系统以使用可用空间：

```
# xfs_growfs /dev/mapper/docker_vol-dockerlv
```



注意

XFS 文件系统无法缩小。

iii. 再次运行 **docker-storage-setup** 命令：

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

现在，您应该在输出中看到扩展卷组和逻辑卷。

```
INFO: Device /dev/vdb is already partitioned and is part of volume group docker_vg
INFO: Found an already configured thin pool /dev/mapper/docker_vg-docker--pool in
/etc/sysconfig/docker-storage
INFO: Device node /dev/mapper/docker_vg-docker--pool exists.
  Logical volume docker_vg/docker-pool changed.
```

4. 启动 Docker 服务：

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

5. 恢复 pod 定义：

```
# mv /etc/origin/node/pods-stopped/* /etc/origin/node/pods/
```

6. 通过重启主机来重启节点服务：

```
# systemctl restart atomic-openshift-node.service
```

7. 在添加新卷组并重新运行 **docker-storage-setup** 时，添加磁盘的一个优点是，在添加新存储后，系统中使用的镜像仍然存在：

```
# container images
REPOSITORY                                TAG             IMAGE ID          CREATED
SIZE
docker-registry.default.svc:5000/tet/perl             latest          8b0b0106fb5e        13
minutes ago      627.4 MB
registry.redhat.io/rhscl/perl-524-rhel7         <none>          912b01ac7570        6 days ago
559.5 MB
registry.redhat.io/openshift3/ose-deployer      v3.6.173.0.21       89fd398a337d        5 weeks
ago         970.2 MB
registry.redhat.io/openshift3/ose-pod           v3.6.173.0.21       63accd48a0d7        5 weeks
ago         208.6 MB
```

8. 随着存储容量增加，使节点可以调度，以接受新的传入的 pod。
作为集群管理员，从 master 实例运行以下 ：

```
$ 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
```

为现有磁盘扩展存储

1. 按照前面的步骤撤离节点。
2. 停止 **docker** ：

```
# systemctl stop docker
```

3. 通过删除 pod 定义来停止节点服务：

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
```

4. 根据需要重新定义现有磁盘的大小。这可取决于您的环境：
   - 如果您使用 LVM（逻辑卷管理器）：
     - 删除逻辑卷 ：

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

     - 删除 Docker 卷组 ：

```
# vgremove docker_vg
```

     - 删除物理卷 ：

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

   - 如果使用云供应商，您可以分离磁盘、销毁磁盘，然后创建一个新的较大的磁盘，并将它附加到实例。
   - 对于非云环境，可以调整磁盘和文件系统的大小。详情请查看以下解决方案：
     - https://access.redhat.com/solutions/199573
5. 通过检查设备名称、大小等，验证 */etc/sysconfig/container-storage-setup* 文件是否已为新磁盘正确配置。
6. 运行 **docker-storage-setup** 以重新配置新磁盘：

```
# 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
```

7. 启动 Docker 服务：

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

8. 恢复 pod 定义：

```
# mv /etc/origin/node/pods-stopped/* /etc/origin/node/pods/
```

9. 通过重启主机来重启节点服务：

```
# systemctl restart atomic-openshift-node.service
```

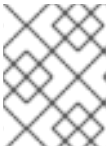
7.1.3. 更改存储后端

随着服务和文件系统的进步，可能需要更改存储后端才能利用新功能。以下步骤提供了将设备映射器后端改为 **overlay2** 存储后端的示例。与传统设备映射器相比，**overlay2** 提供更高的速度和密度。

7.1.3.1. 清空节点

1. 从 master 实例或集群管理员，允许从该节点驱除任何 pod，并禁用该节点上调度其他 pod:

```
$ 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
```



注意

如果存在运行没有迁移的本地卷的容器，请运行以下命令： **oc adm drain ${NODE} --ignore-daemonsets --delete-local-data**

2. 列出节点上的 pod，以验证是否已移除它们：

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

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

NAME      READY     STATUS    RESTARTS   AGE
```

如需有关撤离和排空 pod 或节点的更多信息，请参阅节点维护。

3. 如果没有容器目前在实例上运行的容器，停止 **docker** 服务：

```
# systemctl stop docker
```

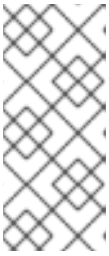
4. 停止 **atomic-openshift-node** 服务：

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

5. 验证卷组、逻辑卷名称和物理卷名称的名称：

```
# 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
```

6. 修改 **docker-storage-setup** 文件，以指定 **STORAGE_DRIVER**。



注意

当从 Red Hat Enterprise Linux 版本 7.3 升级到 7.4 时，**docker** 服务会尝试使用带有 STORAGE_DRIVER 的 **STORAGE_DRIVER** 的 **/var**。使用 extfs 作为 **STORAGE_DRIVER** 会导致错误。有关错误的更多信息，请参见以下程序错误：

- 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
```

7. 设置存储：

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

8. 启动 **docker** ：

```
# systemctl start docker
```

9. 重启 **atomic-openshift-node** 服务：

```
# systemctl restart atomic-openshift-node.service
```

10. 通过将存储修改为使用 **overlay2**，使该节点可以调度来接受新的传入的 pod。
从 master 实例或集群管理员：

```
$ 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. 管理容器 REGISTRY 证书

OpenShift Container Platform 内部 registry 创建为 pod。但是，如果需要，可以从外部 Registry 中拉取容器。默认情况下，registry 侦听 TCP 端口 5000。registry 提供通过 TLS 保护公开镜像的选项，或运行 registry而不加密流量。

警告

Docker 会将 **.crt** 文件解释为 CA 证书，**.cert** 文件作为客户端证书。任何 CA 的扩展都**必须是 .crt**。

7.2.1. 为外部 registry 安装证书颁发机构证书

要将 OpenShift Container Platform 与外部 registry 搭配使用，registry 证书颁发机构(CA)证书必须信任从 registry 中拉取镜像的所有节点。

注意

根据 Docker 版本，信任容器镜像 registry 的进程会有所不同。Docker 根证书颁发机构的最新版本与系统默认值合并。在 **docker** 版本 1.13 之前，只有在没有其他自定义 root 证书时才使用系统默认证书。

流程

1. 将 CA 证书复制到 **/etc/pki/ca-trust/source/anchors/** 中：

```
$ sudo cp myregistry.example.com.crt /etc/pki/ca-trust/source/anchors/
```

2. 提取 CA 证书并将其添加到可信证书颁发机构列表中：

```
$ sudo update-ca-trust extract
```

3. 使用 **openssl** 命令验证 SSL 证书：

```
$ openssl verify myregistry.example.com.crt
myregistry.example.com.crt: OK
```

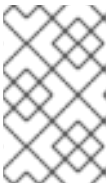
4. 证书被放置并信任被更新后，重启 **docker** 服务以确保正确设置新证书：

```
$ sudo systemctl restart docker.service
```

对于 1.13 之前的 Docker 版本，需要执行以下额外步骤以信任颁发机构的证书：

1. 在每个节点上在 **/etc/docker/certs.d** 中创建一个新目录，其中目录的名称是容器镜像 registry 的主机名：

```
$ sudo mkdir -p /etc/docker/certs.d/myregistry.example.com
```



注意

除非容器镜像 registry 在无需端口号的情况下无法访问容器镜像 registry，否则不需要端口号。将端口寻址到原始 Docker registry，如下所示：**myregistry.example.com:port**

2. 通过 IP 地址访问容器镜像 registry 需要在 **/etc/docker/certs.d** 中创建新目录，其中目录的名称为容器镜像 registry 的 IP：

```
$ sudo mkdir -p /etc/docker/certs.d/10.10.10.10
```

3. 将 CA 证书复制到前面步骤中新创建的 Docker 目录中：

```
$ 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. 复制证书后，重启 **docker** 服务以确保使用新证书：

```
$ sudo systemctl restart docker.service
```

7.2.2. Docker 证书备份

执行节点主机备份时，请确保包含外部 registry 的证书。

流程

1. 如果使用 **/etc/docker/certs.d**，请复制目录中包含的所有证书并存储这些文件：

```
$ sudo tar -czvf docker-registry-certs-$(hostname)-$(date +%Y%m%d).tar.gz
/etc/docker/certs.d/
```

2. 如果使用系统信任，请在将证书添加到系统信任之前存储证书。存储完成后，使用 **trust** 命令提取证书以进行恢复。识别系统信任 CA 并记下 **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. 解压 **pem** 格式证书，并为其提供名称。例如，**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. 使用 **openssl** 检查已正确解压了证书：

```
$ openssl verify myca.crt
```

5. 对所有必需的证书重复这些步骤，并将文件存储在远程位置。

7.2.3. Docker 证书恢复

如果外部 registry 的 Docker 证书删除或损坏，恢复机制将使用之前执行备份中的文件与安装方法相同的步骤。

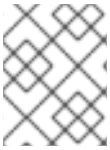
7.3. 管理容器 REGISTRY

您可以将 OpenShift Container Platform 配置为使用外部 **docker** registry 来拉取镜像。但是，您可以使用配置文件来允许或拒绝某些镜像或 registry。

如果外部 Registry 使用了证书用于网络流量，它可以将其称为安全 registry。否则，registry 和 host 之间的流量是纯文本而不是加密，这意味着它是一个不安全的 registry。

7.3.1. Docker 搜索外部 registry

默认情况下，**docker** 守护进程能够从任何 registry 中拉取镜像，但搜索操作针对 **docker.io**/ 和 **registry.redhat.io** 执行。守护进程可以配置为通过 **docker** 守护进程使用 **--add-registry** 选项从其他 registry 中搜索镜像。



注意

Red Hat Enterprise Linux **docker** 软件包中默认存在从 Red Hat Registry **registry.redhat.io** 搜索镜像的功能。

流程

1. 要允许用户在其他 registry 中使用 **docker search** 搜索镜像，请将这些 registry 添加到 **registry** 参数下的 **/etc/containers/registries.conf** 文件中：

```
registries:
  - registry.redhat.io
  - my.registry.example.com
```

2. 重启 **docker** 守护进程以允许使用 **my.registry.example.com** ：

```
$ sudo systemctl restart docker.service
```

重启 **docker** 守护进程会导致 **docker** 容器重启。

3. 使用 Ansible 安装程序，这可以使用 Ansible 主机文件中的 **openshift_docker_additional_registries** 变量进行配置：

```
openshift_docker_additional_registries=registry.redhat.io,my.registry.example.com
```

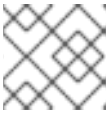
7.3.2. Docker 外部 registry 白名单和黑名单

通过为 **docker** 守护进程配置 **registries** 和 **block_registries** 标志，可以将 Docker 配置为阻止来自外部 registry 的操作。

流程

1. 使用 **registries** 标记将允许的 registry 添加到 **/etc/containers/registries.conf** 文件中：

```
registries:
  - registry.redhat.io
  - my.registry.example.com
```



注意

docker.io registry 可使用相同的方法添加。

2. 阻止其余的 registry：

```
block_registries:
   - all
```

3. 阻止旧版本中的 registry 的其余部分：

```
BLOCK_REGISTRY='--block-registry=all'
```

4. 重启 **docker** 守护进程：

```
$ sudo systemctl restart docker.service
```

重启 **docker** 守护进程会导致 **docker** 容器重启。

5. 在本例中，**docker.io** registry 已被列入黑名单，因此有关该 registry 的任何操作都会失败：

```
$ sudo docker pull hello-world
Using default tag: latest
Trying to pull repository registry.redhat.io/hello-world ...
```

```
Trying to pull repository my.registry.example.com/hello-world ...
Trying to pull repository registry.redhat.io/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.
```

通过再次修改文件并重新启动服务，将 **docker.io** 添加会 **registry** 变量。

```
registries:
  - registry.redhat.io
  - my.registry.example.com
  - docker.io
block_registries:
  - all
```

或者

```
ADD_REGISTRY="--add-registry=registry.redhat.io --add-registry=my.registry.example.com --add-registry=docker.io"
BLOCK_REGISTRY='--block-registry=all'
```

6. 重启 Docker 服务：

```
$ sudo systemctl restart docker
```

7. 验证镜像现在可拉取(pull):

```
$ 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:
sha256:0e06ef5e1945a718b02a8c319e15bae44f47039005530bc617a5d071190ed3fc
```

8. 如果需要使用外部 registry，例如修改所有需要使用该 registry 的节点主机中的 **docker** 守护进程配置文件，请在这些节点上创建一个黑名单以避免恶意容器被执行。
使用 Ansible 安装程序，这可以使用 Ansible 主机文件中的 **openshift_docker_additional_registries** 和 **openshift_docker_blocked_registries** 变量进行配置：

```
openshift_docker_additional_registries=registry.redhat.io,my.registry.example.com
openshift_docker_blocked_registries=all
```

7.3.3. 安全 registry

为了能够从外部 registry 拉取镜像，需要信任 registry 证书，否则 pull 镜像操作会失败。

要这样做，请参阅为外部 registry 安装证书颁发机构证书部分。

如果使用白名单，则外部 registry 应该添加到 **registry** 变量中，如上文所述。

7.3.4. 不安全的 registry

应避免使用非可信证书的外部 registry（或根本没有证书）。

但是，任何不安全的 registry 都应使用 **--insecure-registry** 选项来添加，以允许 **docker** 守护进程从存储库拉取镜像。这与 **--add-registry** 选项相同，但不会验证 **docker** 操作。

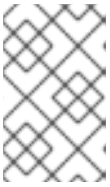
该 registry 应该使用这两个选项添加来启用搜索的选项；如果有黑名单，则应该执行其他操作，如拉取镜像。

出于测试目的，以下示例显示了如何添加一个 **localhost** 不安全的 registry。

流程

1. 修改 **/etc/containers/registries.conf** 配置文件以添加 localhost 不安全的 registry:

```
[registries.search]
registries = ['registry.redhat.io', 'my.registry.example.com', 'docker.io', 'localhost:5000' ]

[registries.insecure]
registries = ['localhost:5000']

[registries.block]
registries = ['all']
```



注意

将不安全的 registry 添加到 **registry.search** 部分以及 **registry.insecure** 部分，以确保它们标记为不安全并列入白名单。添加到 **registeries.block** 部分的任何 registry 都会被阻断，除非也被添加到 **registry.search** 部分。

2. 重启 **docker** 守护进程以使用 registry：

```
$ sudo systemctl restart docker.service
```

重启 **docker** 守护进程会导致 **docker** 容器被重启。

3. 在 **localhost** 运行容器镜像 registry pod:

```
$ sudo docker run -p 5000:5000 registry:2
```

4. 拉取镜像：

```
$ sudo docker pull openshift/hello-openshift
```

5. 标记镜像：

```
$ sudo docker tag docker.io/openshift/hello-openshift:latest localhost:5000/hello-openshift-local:latest
```

6. 将镜像推送到本地 registry：

```
$ sudo docker push localhost:5000/hello-openshift-local:latest
```

7. 使用 Ansible 安装程序，这可以使用 **Ansible** 主机文件中的 **openshift_docker_additional_registries**、**openshift_docker_blocked_registries** 和 **openshift_docker_insecure_registries** 变量进行配置：

```
openshift_docker_additional_registries=registry.redhat.io,my.registry.example.com,localhost:500
0
openshift_docker_insecure_registries=localhost:5000
openshift_docker_blocked_registries=all
```

注意

您还可以将 **openshift_docker_insecure_registries** 变量设置为主机的 IP 地址。**0.0.0.0/0** 不是一个有效设置。

7.3.5. 经过身份验证的 registry

将经过身份验证的 registry 与 **docker** 搭配使用需要 **docker** 守护进程才能登录到 registry。在 OpenShift Container Platform 中，必须执行一组不同的步骤，因为用户无法在主机上运行 **docker login** 命令。经过身份验证的 registry 可用于限制用户能够拉取或能够访问外部 registry 的镜像。

如果外部 **Docker** registry 需要身份验证，请在使用该 registry 的项目中创建特殊 secret，然后使用该 secret 执行 **docker** 操作。

流程

1. 在用户要登录到 **docker** registry 的项目中创建 **dockercfg** secret：

```
$ oc project <my_project>
$ oc create secret docker-registry <my_registry> --docker-server=
<my.registry.example.com> --docker-username=<username> --docker-password=
<my_password> --docker-email=<me@example.com>
```

2. 如果存在 **.dockercfg** 文件，请使用 **oc** 命令创建 secret：

```
$ oc create secret generic <my_registry> --from-file=.dockercfg=<path/to/.dockercfg> --
type=kubernetes.io/dockercfg
```

3. 填充 **$HOME/.docker/config.json** 文件：

```
$ oc create secret generic <my_registry> --from-file=.dockerconfigjson=<path/to/.dockercfg>
--type=kubernetes.io/dockerconfigjson
```

4. 通过将 secret 链接到执行 pull 操作的服务帐户，使用 **dockercfg** secret 从经过身份验证的 registry 拉取镜像。用于拉取镜像的默认服务帐户名为 **default** ：

```
$ oc secrets link default <my_registry> --for=pull
```

5. 要使用 S2I 功能推送镜像，**dockercfg** secret 被挂载到 S2I pod 中，因此需要链接到执行该构建的正确服务帐户。用于构建镜像的默认服务帐户名为 **builder**。

```
$ oc secrets link builder <my_registry>
```

6. 在 **buildconfig** 中，应指定 secret 进行 push 或 pull 操作：

```
"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. 如果外部 Registry 将身份验证委派给外部服务，则创建 **dockercfg** secret：registry 使用 registry URL 和外部身份验证系统，并使用自己的 URL。这两个 secret 都应添加到服务帐户。

```
$ oc project <my_project>
$ oc create secret docker-registry <my_registry> --docker-server=*
<my_registry_example.com> --docker-username=<username> --docker-password=
<my_password> --docker-email=<me@example.com>
$ oc create secret docker-registry <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 准入插件

准入插件截获对 API 的请求，并根据配置的规则执行检查，并根据这些规则允许或拒绝某些操作。OpenShift Container Platform 可以使用 **ImagePolicy** 准入插件限制环境中运行的镜像：

- 镜像源：可以用于拉取镜像的 registry
- 镜像解析：强制 pod 使用不可变摘要运行，以确保因为重新标记而不会改变镜像
- 容器镜像标签限制：强制镜像具有或没有特定标签
- 镜像注解限制：强制集成容器 registry 中的镜像具有或没有特定注解

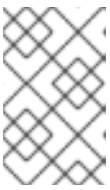
警告

ImagePolicy 准入插件目前被视为 beta。

流程

1. 如果启用了 **ImagePolicy** 插件，则需要修改它，以允许在每个 master 节点上修改 **/etc/origin/master/master-config.yaml** 文件来允许使用外部 registry：

```
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.redhat.io
```



注意

启用 **ImagePolicy** 要求用户在部署应用程序时指定 registry，如 **oc new-app docker.io/kubernetes/guestbook** 替代 **oc new-app kubernetes/guestbook**，否则会失败。

2. 要在安装时启用准入插件，**openshift_master_admission_plugin_config** 变量可用于 **json** 格式的字符串，包括所有 **pluginConfig** 配置：

```
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.redhat.io"]}]}}}
```

7.3.7. 从外部 registry 导入镜像

应用程序开发人员可以使用 **oc import-image** 命令导入镜像来创建**镜像流**，并且可以配置 OpenShift Container Platform 来允许或拒绝来自外部 registry 的镜像导入。

流程

1. 要配置可以导入镜像的允许的 registry，请将以下内容添加到 **/etc/origin/master/master-config.yaml** 文件中：

```
imagePolicyConfig:
```

```
  allowedRegistriesForImport:
  - domainName: docker.io
  - domainName: '\*.docker.io'
  - domainName: '*.redhat.com'
  - domainName: 'my.registry.example.com'
```

2. 要从外部经过身份验证的用户导入镜像，请在所需项目中创建一个 secret。

3. 即使不建议使用外部身份验证的 registry，或者无法信任证书，**oc import-image** 命令可以与 **--insecure=true** 选项一起使用。
   如果外部经过身份验证的用户的 registry 是安全的，则 registry 证书应该在 master 主机中运行 registry 导入控制器时被信任：

   在 **/etc/pki/ca-trust/source/anchors/** 中复制证书：

```
$ sudo cp <my.registry.example.com.crt> /etc/pki/ca-
trust/source/anchors/<my.registry.example.com.crt>
```

4. 运行 **update-ca-trust** 命令：

```
$ sudo update-ca-trust
```

5. 重启所有 master 主机上的 master 服务：

```
$ sudo master-restart api
$ sudo master-restart controllers
```

6. 外部 registry 的证书应该在 OpenShift Container Platform registry 中信任：

```
$ 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
```

警告

目前，没有将证书添加到 registry pod 的官方流程，但可以使用上述临时解决方案。

这个临时解决方案是，在运行这些命令的系统中使用所有可信证书创建 **configmap**，因此建议是从只信任所需证书的干净系统中运行它。

7. 或者，修改 registry 镜像，以信任使用 **Dockerfile** 重建镜像的适当证书：

```
FROM registry.redhat.io/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. 重建镜像，将其推送到 **docker** registry，并将该镜像用作 registry **deploymentconfig** 中的 **spec.template.spec.containers["name":"registry"].image**:

```
$ oc patch dc docker-registry -p '{"spec":{"template":{"spec":{"containers":
[{"name":"registry","image":"*myregistry.example.com/openshift3/ose-docker-
registry:latest*"}]}}}}'
```

注意

要在安装时添加 **imagePolicyConfig** 配置，**openshift_master_image_policy_config** 变量可用于 **json** 格式的字符串，包括所有 **imagePolicyConfig** 配置，如下所示：

```
openshift_master_image_policy_config={"imagePolicyConfig":
{"allowedRegistriesForImport":[{"domainName":"docker.io"},
{"domainName":"\*.docker.io"},{"domainName":"*.redhat.com"},
{"domainName":"*my.registry.example.com*"}]}}
```

有关 **ImagePolicy** 的更多信息，请参阅 **ImagePolicy** 准入插件 部分。

7.3.8. OpenShift Container Platform registry 集成

您可以将 OpenShift Container Platform 安装为独立容器镜像 registry，以仅提供 registry 功能，但具有在 OpenShift Container Platform 平台中运行的优点。

如需有关 OpenShift Container Platform registry 的更多信息，请参阅安装独立 OpenShift Container Registry 部署。

要集成 OpenShift Container Platform registry，前面所有部分都适用。从 OpenShift Container Platform 的视角来看，它被视为外部 registry，但有些额外的任务需要执行，因为它是一个多租户 registry，OpenShift Container Platform 中的授权模型会应用，因此在创建新项目时，registry 不会在其环境中创建项目。

7.3.8.1. 将 registry 项目与集群连接

因为 registry 是一个包括 registry pod 和 Web 界面的完整 OpenShift Container Platform 环境，所以在 registry 中创建一个新项目，或使用 **oc new-project** 或 **oc create** 命令行执行。

创建项目后，会自动创建常规服务帐户（**builder**、**default** 和 **deployer**），以及项目管理员用户被授予权限。不同的用户可以授权推送/提取镜像以及"匿名"用户。

可能存在几个用例，比如允许所有用户从 registry 中的这个新项目拉取镜像，但是如果您想要在 OpenShift Container Platform 和 registry 之间有一个 1:1 项目关系，用户可以在该特定项目中推送和拉取镜像，需要一些步骤。

警告

registry Web 控制台显示要用于拉取/推送操作的令牌，但令牌显示有一个会话令牌，因此它过期。通过创建具有特定权限的服务帐户，管理员可以限制服务帐户的权限，例如：不同的服务帐户可用于推送或拉取镜像。然后，用户不必配置令牌过期时间、secret 重新创建和其他任务，因为服务帐户令牌不会过期。

流程

1. 创建一个新项目

```
$ oc new-project <my_project>
```

2. 创建 registry 项目：

```
$ oc new-project <registry_project>
```

3. 在 registry 项目中创建服务帐户：

```
$ oc create serviceaccount <my_serviceaccount> -n <registry_project>
```

4. 授予使用 **registry-editor** 角色推送和拉取镜像的权限：

```
$ oc adm policy add-role-to-user registry-editor -z <my_serviceaccount> -n <registry_project>
```

如果只需要拉取权限，则可以使用 **registry-viewer** 角色。

5. 获取服务帐户令牌：

```
$ TOKEN=$(oc sa get-token <my_serviceaccount> -n <registry_project>)
```

6. 使用令牌作为密码来创建 **dockercfg** secret：

```
$ oc create secret docker-registry <my_registry> \
  --docker-server=<myregistry.example.com> --docker-username=<notused> --docker-
password=${TOKEN} --docker-email=<me@example.com>
```

7. 通过将 secret 链接到执行 pull 操作的服务帐户，使用 **dockercfg** secret 从 registry 中拉取镜像。用于拉取镜像的默认服务帐户名为 **default** ：

```
$ oc secrets link default <my_registry> --for=pull
```

8. 要使用 S2I 功能推送镜像，**dockercfg** secret 被挂载到 S2I pod 中，因此需要链接到执行该构建的正确服务帐户。用于构建镜像的默认服务帐户名为 **builder** ：

```
$ oc secrets link builder <my_registry>
```

9. 在 **buildconfig** 中，应指定 secret 进行 push 或 pull 操作：

-

```
"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]...
```

第 8 章 管理证书

在 OpenShift Container Platform 集群的生命周期中，证书将进入其生命周期的不同阶段。以下流程描述了如何管理该生命周期的各个部分。

如需有关查看证书过期和重新部署证书的信息，请参阅重新部署证书。

8.1. 将应用程序的自签名证书改为 CA 签名证书

有些应用模板创建一个自签名证书，然后由应用程序直接向客户端提供该证书。例如，默认情况下，作为 OpenShift Container Platform Ansible 安装程序部署过程的一部分，指标部署器会创建自签名证书。

这些自签名证书不能被浏览器识别。要缓解这个问题，请使用公开签名的证书，然后将其配置为使用自签名证书重新加密流量。

1. 删除现有路由：

```
$ oc delete route hawkular-metrics -n openshift-infra
```

删除路由后，需要使用 re-encrypt 策略的新路由中使用的证书必须从现有的通配符和指标部署器创建的自签名证书中编译。以下证书必须可用：

- 通配符 CA 证书
- 通配符私钥
- 通配符证书
- Hawkular CA 证书
  每个证书都必须作为新路由的文件系统中的文件使用。

  您可以执行以下命令，检索 Hawkular CA 并将其存储在文件中：

```
$ oc get secrets hawkular-metrics-certs \
    -n openshift-infra \
    -o jsonpath='{.data.ca\.crt}' | base64 \
    -d > hawkular-internal-ca.crt
```

2. 找到通配符私钥、证书和 CA 证书。每个都分别放在一个单独的文件中，如 ***wildcard.key***、***wildcard.crt*** 和 ***wildcard.ca***。

3. 创建新的重新加密路由：

```
$ oc create route reencrypt hawkular-metrics-reencrypt \
    -n openshift-infra \
    --hostname hawkular-metrics.ocp.example.com \
    --key wildcard.key \
    --cert wildcard.crt \
    --ca-cert wildcard.ca \
    --service hawkular-metrics \
    --dest-ca-cert hawkular-internal-ca.crt
```