OpenShift Container Platform 3.11

配置集群

OpenShift Container Platform 3.11 安装和配置

Last Updated: 2023-03-25

OpenShift Container Platform 3.11 配置集群

OpenShift Container Platform 3.11 安装和配置

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 安装和配置主题涵盖环境中安装和配置 OpenShift 的基础知识。将这些主题用于启动和运行 OpenShift 所需的一次性任务。

目录

第 1 章 概述

本指南涵盖了安装后 OpenShift Container Platform 集群的更多配置选项。

第 2 章 设置 REGISTRY

2.1. 内部 REGISTRY 概述

2.1.1. 关于 Registry

OpenShift Container Platform 可以从源代码构建 容器镜像，部署并管理它们的生命周期。为此，OpenShift Container Platform 提供了一个内部集成的容器镜像 registry，可在 OpenShift Container Platform 环境中部署以在本地管理镜像。

2.1.2. 集成的或独立 registry

在初始安装完整 OpenShift Container Platform 集群期间，可能会在安装过程中自动部署 registry。如果没有，或者您要进一步自定义 registry 的配置，请参阅在现有集群中部署 registry。

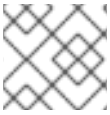
虽然它可以部署为作为完整 OpenShift Container Platform 集群集成的一部分运行，但 OpenShift Container Platform registry 可以作为独立容器镜像 registry 单独安装。

要安装独立 registry，请遵循安装独立 Registry。此安装路径部署运行 registry 和专用 Web 控制台的一体化集群。

2.2. 在现有集群中部署 REGISTRY

2.2.1. 概述

如果集成的 registry 之前在 OpenShift Container Platform 集群的初始安装过程中没有被自动部署，或者如果不再成功运行，您需要在现有集群中重新部署它，请参阅以下部分来了解部署新 registry 的选项。



注意

如果您安装了 独立 registry，则不需要这个主题。

2.2.2. 设置 Registry 主机名

您可以为内部和外部引用配置 registry 的主机名和端口。通过这样做，镜像流将为镜像提供基于主机名的推送和拉取规格，允许使用镜像与对 registry 服务 IP 的更改进行隔离，并有可能允许镜像流及其引用在集群中可移植。

要设置从集群内部引用 registry 的主机名，请在 master 配置文件的 **imagePolicyConfig** 部分中设置 **internalRegistryHostname**。外部主机名通过在同一位置设置 **externalRegistryHostname** 值来控制。

镜像策略配置

```
imagePolicyConfig:
  internalRegistryHostname: docker-registry.default.svc.cluster.local:5000
  externalRegistryHostname: docker-registry.mycompany.com
```

registry 本身必须使用相同的内部主机名值进行配置。这可以通过在 registry 署配置中设置 **REGISTRY_OPENSHIFT_SERVER_ADDR** 环境变量或设置 registry 配置的 OpenShift 部分中的值来完成。

注意

如果您已经为 registry 启用了 TLS，服务器证书必须包含您希望引用 registry 的主机名。有关添加主机名到服务器证书的说明，请参阅保护registry。

2.2.3. 部署 Registry

要部署集成的容器镜像 registry，请以具有集群管理员特权的用户身份使用 **oc adm registry** 命令。例如：

```
$ oc adm registry --config=/etc/origin/master/admin.kubeconfig \ 1
    --service-account=registry \ 2
    --images='registry.redhat.io/openshift3/ose-${component}:${version}' 3
```

1 **--config** 是 集群管理员 CLI 配置文件 的路径。

2 **--service-account** 是用于运行 registry pod 的服务帐户。

3 为 OpenShift Container Platform 拉取正确的镜像是必需的。**${component}** 和 **${version}** 在安装过程中会被动态替换。

这将创建服务和部署配置，两者均称为 **docker-registry**。部署成功后，会使用类似于 **docker-registry-1-cpty9** 的名称创建 pod。

查看创建 registry 时可以指定的完整选项列表：

```
$ oc adm registry --help
```

--fs-group 的值需要是 registry 使用的 SCC 所允许的（通常为有限制的 SCC）。

2.2.4. 将 Registry 部署为 DaemonSet

使用 **oc adm registry** 命令将 registry 部署为带有 **--daemonset** 选项的 **DaemonSet**。

DaemonSet 可确保在创建节点时，节点包含指定 pod 的副本。删除节点时，会收集 pod。

如需有关 **DaemonSet** 的更多信息，请参阅使用 Daemonsets。

2.2.5. registry 计算资源

默认情况下，创建 registry 时没有设置计算资源请求或限值。对于生产环境，强烈建议更新 registry 的部署配置，以便为 registry 容器集设置资源请求和限值。否则，registry pod 将被视为 **BestEffort** pod。

有关配置请求和限制的更多信息，请参阅计算资源。

2.2.6. Registry 的存储

registry 存储容器镜像和元数据。如果您只是使用 registry 来部署 pod，它会使用一个在 pod 退出时销毁的临时卷。任何用户已构建或推送到 registry 的镜像都会消失。

本节列出了支持的 registry 存储驱动程序。如需更多信息，请参阅容器镜像 registry 文档。

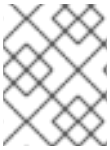
以下列表包括在 registry 配置文件中需要配置的存储驱动程序：

- 文件系统。文件系统是默认设置，不需要进行配置。
- S3.如需更多信息，请参阅 CloudFront 配置 文档。
- OpenStack Swift
- Google Cloud Storage(GCS)
- Microsoft Azure
- Aliyun OSS

支持常规 registry 存储配置选项。如需更多信息，请参阅容器镜像 registry 文档。

以下存储选项需要通过文件系统驱动程序配置：

- GlusterFS Storage
- Ceph Rados 块设备



注意

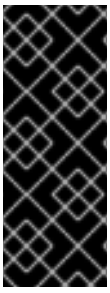
如需有关支持的持久性存储驱动程序的更多信息，请参阅配置持久性存储和持久性存储示例。

2.2.6.1. 产品使用

对于生产环境，请附加一个远程卷，或者 定义和使用您选择的持久性存储方法。

例如，使用现有持久性卷声明：

```
$ oc set volume deploymentconfigs/docker-registry --add --name=registry-storage -t pvc \
    --claim-name=<pvc_name> --overwrite
```



重要

测试显示，使用 RHEL NFS 服务器作为容器镜像 registry 的存储后端会出现问题。这包括 OpenShift Container Registry 和 Quay。因此，不建议使用 RHEL NFS 服务器来备份核心服务使用的 PV。

市场上的其他 NFS 实现可能没有这些问题。如需了解更多与此问题相关的信息，请联络相关的 NFS 厂商。

2.2.6.1.1. 使用 Amazon S3 作为存储后端

还有一个选项，可以将 Amazon Simple Storage Service 存储与内部容器镜像 registry 搭配使用。它是一种安全云存储，可通过 AWS 管理控制台进行管理。要使用它，必须手动编辑注册表的配置文件并将其挂载到 registry Pod。但是，在开始配置之前，请查看上游的建议步骤。

采用 默认 YAML 配置文件作为基础，并将 **storage** 部分中的 **filesystem** 条目替换为 **s3** 条目，如下方所示：生成的存储部分可能类似如下：

```
storage:
  cache:
    layerinfo: inmemory
```

```
delete:
  enabled: true
s3:
  accesskey: awsaccesskey (1)
  secretkey: awssecretkey (2)
  region: us-west-1
  regionendpoint: http://myobjects.local
  bucket: bucketname
  encrypt: true
  keyid: mykeyid
  secure: true
  v4auth: false
  chunksize: 5242880
  rootdirectory: /s3/object/name/prefix
```

1. 使用您的 Amazon 访问密钥替换。
2. 使用您的 Amazon secret 密钥替换。

所有 **s3** 配置选项都记录在上游 驱动程序参考文档中。

覆盖 registry 配置将引导您完成将配置文件挂载到 pod 的其他步骤。

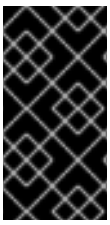
警告

当 registry 在 S3 存储后端上运行时，会报告问题。

如果要使用您正在使用的集成 registry 不支持的 S3 区域，请参阅 S3 驱动程序配置。

2.2.6.2. 非生产环境中的使用

对于非生产环境，您可以使用 **--mount-host=<path>** 选项为 registry 指定用于持久性存储的目录。然后，registry 卷在指定的 **<path>** 上创建为 host-mount。



重要

mount-host 选项可从 registry 容器所在的节点上挂载目录。如果扩展 **docker-registry** 部署配置，您的 registry Pod 和容器可能会在不同节点上运行，这可能会导致两个或多个 registry 容器，每个容器都有自己的本地存储。这会导致无法预测的行为，因为后续的拉取同一镜像的请求可能并不总是成功，具体取决于请求最终所针对的容器。

--mount-host 选项要求 registry 容器以特权模式运行。这在指定 **--mount-host** 时自动启用。但是，并非所有 pod 都默认允许运行特权容器。如果您仍然希望使用这个选项，请创建 registry 并指定它使用在安装过程中创建的 **registry** 服务帐户：

```
$ oc adm registry --service-account=registry \
    --config=/etc/origin/master/admin.kubeconfig \
    --images='registry.redhat.io/openshift3/ose-${component}:${version}' \ (1)
```

```
--mount-host=<path>
```

1 为 OpenShift Container Platform 拉取正确的镜像是必需的。**${component}** 和 **${version}** 在安装过程中会被动态替换。

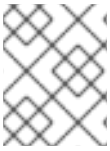
重要

容器镜像 registry 容器集以用户 1001 身份运行。此用户必须能够写入主机目录。使用以下命令，您可能需要将目录所有权改为用户 ID 1001 ：

```
$ sudo chown 1001:root <path>
```

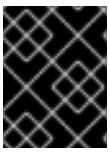
2.2.7. 启用 Registry 控制台

OpenShift Container Platform 为集成的 registry 提供了一个基于 Web 的界面。此 registry 控制台是浏览和管理镜像的可选组件。它部署为作为容器集运行的无状态服务。



注意

如果将 OpenShift Container Platform 作为独立 registry 安装，则 registry 控制台在安装过程中会自动部署和保护。



重要

如果 Cockpit 已在运行，您需要先将其关闭，然后才能继续，以避免与 registry 控制台造成端口冲突（默认为 9090）。

2.2.7.1. 部署 Registry 控制台

重要

您必须首先 公开 registry。

1. 在 **default** 项目中创建一个 passthrough 路由。下一步中创建 registry 控制台应用时，您将需要此设置。

```
$ oc create route passthrough --service registry-console \
    --port registry-console \
    -n default
```

2. 部署 registry 控制台应用。将 **<openshift_oauth_url>** 替换为 OpenShift Container Platform OAuth 供应商的 URL，通常是 master。

```
$ oc new-app -n default --template=registry-console \
    -p OPENSHIFT_OAUTH_PROVIDER_URL="https://<openshift_oauth_url>:8443" \
    -p REGISTRY_HOST=$(oc get route docker-registry -n default --template='{{ .spec.host }}') \
    -p COCKPIT_KUBE_URL=$(oc get route registry-console -n default --template='https://{{ .spec.host }}')
```

注意

如果您试图登录到 registry 控制台时重定向 URL 错误，请使用 **oc get oauthclients** 检查 OAuth 客户端。

3. 最后，使用 Web 浏览器使用路由 URI 查看控制台。

2.2.7.2. 保护 Registry 控制台

默认情况下，如果按照部署 registry 控制台中的步骤手动部署，registry 控制台会生成自签名 TLS 证书。如需更多信息，请参阅 对 Registry 控制台进行故障排除。

使用以下步骤将组织的签名证书添加为 secret 卷。假设您的证书在 **oc** 客户端主机上可用。

1. 创建一个包含证书和密钥的 ***.cert*** 文件。使用以下内容格式化文件：

   - 服务器证书和中间证书颁发机构的一个或多个 **BEGIN CERTIFICATE** 块
   - 包含该密钥的 **BEGIN PRIVATE KEY** 或类似密钥的块。密钥不能加密
     例如：

     ```
     -----BEGIN CERTIFICATE-----
     MIIDUzCCAjugAwIBAgIJAPXW+CuNYS6QMA0GCSqGSIb3DQEBCwUAMD8xKTAnBgN
     V
     BAoMIGI0OGE2NGNkNmMwNTQ1YThhZTgxOTEzZDE5YmJjMmRjMRIwEAYDVQQDD
     Als
     ...
     -----END CERTIFICATE-----
     -----BEGIN CERTIFICATE-----
     MIIDUzCCAjugAwIBAgIJAPXW+CuNYS6QMA0GCSqGSIb3DQEBCwUAMD8xKTAnBgN
     V
     BAoMIGI0OGE2NGNkNmMwNTQ1YThhZTgxOTEzZDE5YmJjMmRjMRIwEAYDVQQDD
     Als
     ...
     -----END CERTIFICATE-----
     -----BEGIN PRIVATE KEY-----
     MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCyOJ5garOYw0sm
     8TBCDSqQ/H1awGMzDYdB11xuHHsxYS2VepPMzMzryHR137l4dGFLhvdTvJUH8lUS
     ...
     -----END PRIVATE KEY-----
     ```

   - 安全 registry 应包含以下主题备用名称(SAN)列表：
     - 两个服务主机名：
       例如：

       ```
       docker-registry.default.svc.cluster.local
       docker-registry.default.svc
       ```

     - 服务 IP 地址.
       例如：

       ```
       172.30.124.220
       ```

使用以下命令获取容器镜像 registry 服务 IP 地址：

```
oc get service docker-registry --template='{{.spec.clusterIP}}'
```

- 公共主机名.
例如：

```
docker-registry-default.apps.example.com
```

使用以下命令获取容器镜像 registry 公共主机名：

```
oc get route docker-registry --template '{{.spec.host}}'
```

例如，服务器证书应包含类似如下的 SAN 详情：

```
X509v3 Subject Alternative Name:
            DNS:docker-registry-public.openshift.com, DNS:docker-registry.default.svc,
DNS:docker-registry.default.svc.cluster.local, DNS:172.30.2.98, IP
Address:172.30.2.98
```

registry 控制台从 */etc/cockpit/ws-certs.d* 目录中加载证书。它使用最后一个带有 *.cert* 扩展名的文件（按字母顺序排列）。因此，*.cert* 文件应至少包含以 OpenSSL 样式格式化的至少两个 PEM 块。

如果没有找到证书，将使用 **openssl** 命令创建自签名证书，并存储在 *0-self-signed.cert* 文件中。

2. 创建 secret：

```
$ oc create secret generic console-secret \
    --from-file=/path/to/console.cert
```

3. 将 secret 添加到 **registry-console** 部署配置中：

```
$ oc set volume dc/registry-console --add --type=secret \
    --secret-name=console-secret -m /etc/cockpit/ws-certs.d
```

这会触发 registry 控制台的新部署，使其包含您的签名证书。

2.2.7.3. 对 Registry 控制台进行故障排除

2.2.7.3.1. 调试模式

使用环境变量启用 registry 控制台调试模式。以下命令以 debug 模式重新部署 registry 控制台：

```
$ oc set env dc registry-console G_MESSAGES_DEBUG=cockpit-ws,cockpit-wrapper
```

启用调试模式允许在 registry 控制台的 pod 日志中显示更详细的日志。

2.2.7.3.2. 显示 SSL 证书路径

要检查 registry 控制台正在使用的证书，可以从控制台容器集内运行命令。

1. 列出 **default** 项目中的 pod，并找到 registry 控制台的 pod 名称：

```
$ oc get pods -n default
NAME                       READY     STATUS    RESTARTS   AGE
registry-console-1-rssrw   1/1       Running   0          1d
```

2. 使用上一命令中的容器集名称，获取 **cockpit-ws** 进程使用的证书路径。本例演示了使用自动生成的证书的控制台：

```
$ oc exec registry-console-1-rssrw remotectl certificate
certificate: /etc/cockpit/ws-certs.d/0-self-signed.cert
```

2.3. 访问 REGISTRY

2.3.1. 查看日志

要查看容器镜像 registry 的日志，请使用带有部署配置的 **oc logs** 命令：

```
$ oc logs dc/docker-registry
2015-05-01T19:48:36.300593110Z time="2015-05-01T19:48:36Z" level=info
msg="version=v2.0.0+unknown"
2015-05-01T19:48:36.303294724Z time="2015-05-01T19:48:36Z" level=info msg="redis not
configured" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
2015-05-01T19:48:36.303422845Z time="2015-05-01T19:48:36Z" level=info msg="using inmemory
layerinfo cache" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
2015-05-01T19:48:36.303433991Z time="2015-05-01T19:48:36Z" level=info msg="Using OpenShift
Auth handler"
2015-05-01T19:48:36.303439084Z time="2015-05-01T19:48:36Z" level=info msg="listening on :5000"
instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
```

2.3.2. 文件存储

标签和镜像元数据存储在 OpenShift Container Platform 中，但 registry 将层和签名数据存储在挂载到位于 ***/registry*** 的 registry 容器中的卷中。因为 **oc exec** 不在特权容器上工作，若要查看 registry 的内容，您必须手动 SSH 到 registry pod 的容器所在的节点，然后在容器本身上运行 **docker exec** ：

1. 列出当前的 pod 以查找容器镜像 registry 的 pod 名称：

```
# oc get pods
```

然后，使用 **oc describe** 来查找运行容器的节点的主机名：

```
# oc describe pod <pod_name>
```

2. 登录到所需的节点：

```
# ssh node.example.com
```

3. 从节点主机上的默认项目中列出正在运行的容器，并确定容器镜像 registry 的容器 ID：

```
# docker ps --filter=name=registry_docker-registry.*_default_
```

4. 使用 **oc rsh** 命令列出 registry 内容：

```
# oc rsh dc/docker-registry find /registry
/registry/docker
/registry/docker/registry
/registry/docker/registry/v2
/registry/docker/registry/v2/blobs (1)
/registry/docker/registry/v2/blobs/sha256
/registry/docker/registry/v2/blobs/sha256/ed
/registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331ca3d83c648c24f92cece5f
89d95ac6c34ce751111810
/registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331ca3d83c648c24f92cece5f
89d95ac6c34ce751111810/data (2)
/registry/docker/registry/v2/blobs/sha256/a3
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd9fd84406680ae93d633cb1
6422d00e8a7c22955b46d4
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd9fd84406680ae93d633cb1
6422d00e8a7c22955b46d4/data
/registry/docker/registry/v2/blobs/sha256/f7
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb26f259582bb33502bdb0fcf5
011e03c60577c4284845
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb26f259582bb33502bdb0fcf5
011e03c60577c4284845/data
/registry/docker/registry/v2/repositories (3)
/registry/docker/registry/v2/repositories/p1
/registry/docker/registry/v2/repositories/p1/pause (4)
/registry/docker/registry/v2/repositories/p1/pause/_manifests
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256/e9a2ac64189818
97b399d3709f1b4a6d2723cd38a4909215ce2752a5c068b1cf
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256/e9a2ac64189818
97b399d3709f1b4a6d2723cd38a4909215ce2752a5c068b1cf/signatures (5)
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256/e9a2ac64189818
97b399d3709f1b4a6d2723cd38a4909215ce2752a5c068b1cf/signatures/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256/e9a2ac64189818
97b399d3709f1b4a6d2723cd38a4909215ce2752a5c068b1cf/signatures/sha256/ede17b139a2
71d6b1331ca3d83c648c24f92cece5f89d95ac6c34ce751111810
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256/e9a2ac64189818
97b399d3709f1b4a6d2723cd38a4909215ce2752a5c068b1cf/signatures/sha256/ede17b139a2
71d6b1331ca3d83c648c24f92cece5f89d95ac6c34ce751111810/link (6)
/registry/docker/registry/v2/repositories/p1/pause/_uploads (7)
/registry/docker/registry/v2/repositories/p1/pause/_layers (8)
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/a3ed95caeb02ffe68cdd9fd844
06680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/a3ed95caeb02ffe68cdd9fd844
06680ae93d633cb16422d00e8a7c22955b46d4/link (9)
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/f72a00a23f01987b42cb26f25
9582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/f72a00a23f01987b42cb26f25
9582bb33502bdb0fcf5011e03c60577c4284845/link
```

(1) 此目录将所有层和签名存储为 blob。

2 此文件包含 blob 的内容。

3 此目录存储所有镜像存储库。

4 此目录适用于单个镜像存储库 **p1/pause**。

5 此目录包含特定镜像清单修订版本的签名。

6 此文件包含返回到 blob（包含签名数据）的引用。

7 此目录包含目前为给定存储库上传和暂存的任何层。

8 此目录包含指向这个存储库引用的所有层的链接。

9 此文件包含对已通过镜像链接到该存储库的特定层的引用。

2.3.3. 直接访问 Registry

对于高级用途，您可以直接访问 registry 来调用 **docker** 命令。这可让您直接使用 **docker push** 或 **docker pull** 等操作从集成的 registry 中推送镜像或从中提取镜像。为此，您必须使用 **docker login** 命令登录 registry。您可以执行的操作取决于您的用户权限，如以下各节所述。

2.3.3.1. 用户必备条件

要直接访问 registry，您使用的用户必须满足以下条件，具体取决于您的预期用途：

- 对于任何直接访问，您必须有首选身份提供程序的常规用户。普通用户可以生成登录到 registry 所需的访问令牌。系统用户（如 **system:admin** ）无法获取访问令牌，因此无法直接访问 registry。
  例如，如果您使用 **HTPASSWD** 身份验证，您可以使用以下命令创建一个：

```
# htpasswd /etc/origin/master/htpasswd <user_name>
```

- 若要调取镜像，例如使用 **docker pull** 命令，用户必须具有 **registry-viewer** 角色。添加此角色：

```
$ oc policy add-role-to-user registry-viewer <user_name>
```

- 为了编写或推送镜像，例如使用 **docker push** 命令，用户必须具有 **registry-editor** 角色。添加此角色：

```
$ oc policy add-role-to-user registry-editor <user_name>
```

如需有关用户权限的更多信息，请参阅管理角色绑定。

2.3.3.2. 登录到 Registry

注意

确保您的用户满足直接访问 registry 的先决条件。

直接登录到 registry：

1. 确保您以 **普通用户身份** 登录到 OpenShift Container Platform：

```
$ oc login
```

2. 使用您的访问令牌登录到容器镜像registry：

```
docker login -u openshift -p $(oc whoami -t) <registry_ip>:<port>
```

注意

您可以为用户名传递任何值，令牌包含所有必要的信息。传递包含冒号的用户名将导致登录失败。

2.3.3.3. 推送和拉取镜像

登录 registry 后，您可以对 registry 执行 **docker pull** 和 **docker push** 操作。

重要

您可以抓取任意镜像，但是如果已添加了**system:registry**角色，则只能将镜像推送到您自己的registry中。

在以下示例中，我们使用：

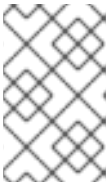
组件	值
<registry_ip>	**172.30.124.220**
<port>	**5000**
<project>	**openshift**
<image>	**busybox**
<tag>	忽略 (默认为 **latest**)

1. 抓取任意镜像：

```
$ docker pull docker.io/busybox
```

2. 使用 **<registry_ip>:<port>/<project>/<image>** 格式标记（tag）新镜像。项目名称**必须**出现在这个 pull 规范中，以供OpenShift Container Platform 把这个镜像正确放置在 registry 中，并在以后正确访问 registry 中的这个镜像：

```
$ docker tag docker.io/busybox 172.30.124.220:5000/openshift/busybox
```



注意

您的常规用户必须具有指定项目的 **system:image-builder** 角色，该角色允许用户写入或推送镜像。否则，下一步中的 **docker push** 将失败。若要进行测试，您可以创建一个新项目来推送 **busybox** 镜像。

3. 将新标记的镜像推送到registry：

```
$ docker push 172.30.124.220:5000/openshift/busybox
...
cf2616975b4a: Image successfully pushed
Digest: sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f77403cab55
```

2.3.4. 访问 Registry 指标

OpenShift Container Registry 为 Prometheus metrics 提供了一个端点。Prometheus是一个独立的开源系统监视和警报工具包。

metrics 可以通过registry端点的***/extensions/v2/metrics***路径获得。但是，必须先启用此路由；请参阅 Extended Registry Configuration 了解说明。

以下是指标查询的简单示例：

```
$ curl -s -u <user>:<secret> \ 1
    http://172.30.30.30:5000/extensions/v2/metrics | grep openshift | head -n 10

# HELP openshift_build_info A metric with a constant '1' value labeled by major, minor, git commit &
git version from which OpenShift was built.
# TYPE openshift_build_info gauge
openshift_build_info{gitCommit="67275e1",gitVersion="v3.6.0-alpha.1+67275e1-
803",major="3",minor="6+"} 1
# HELP openshift_registry_request_duration_seconds Request latency summary in microseconds for
each operation
# TYPE openshift_registry_request_duration_seconds summary
openshift_registry_request_duration_seconds{name="test/origin-
pod",operation="blobstore.create",quantile="0.5"} 0
openshift_registry_request_duration_seconds{name="test/origin-
pod",operation="blobstore.create",quantile="0.9"} 0
openshift_registry_request_duration_seconds{name="test/origin-
pod",operation="blobstore.create",quantile="0.99"} 0
openshift_registry_request_duration_seconds_sum{name="test/origin-
pod",operation="blobstore.create"} 0
openshift_registry_request_duration_seconds_count{name="test/origin-
pod",operation="blobstore.create"} 5
```

1 **<user>** 可以是任意的，但 **<secret>** 必须与 registry 配置中指定的值匹配。

访问指标的另一种方法是使用集群角色。您仍然需要启用端点，但不需要指定 **<secret>**。配置文件中与指标数据相关的部分应如下所示：

```
openshift:
  version: 1.0
  metrics:
```

```
    enabled: true
...
```

如果您还没有角色来访问指标，则必须创建一个集群角色：

```
$ cat <<EOF |
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus-scraper
rules:
- apiGroups:
  - image.openshift.io
  resources:
  - registry/metrics
  verbs:
  - get
EOF
oc create -f -
```

要将此角色添加到用户，请运行以下命令：

```
$ oc adm policy add-cluster-role-to-user prometheus-scraper <username>
```

有关更高级查询和推荐的可视化工具，请参阅 上游 Prometheus 文档。

2.4. 保护并公开 REGISTRY

2.4.1. 概述

默认情况下，OpenShift Container Platform registry 在集群安装过程中是安全的，以便通过 TLS 提供流量。默认情况下还会创建一个 passthrough 路由，以将服务公开到外部。

如果出于任何原因您的 registry 没有被保护或公开，请参阅以下部分来了解如何手动操作。

2.4.2. 手动保护 Registry

手动保护 registry 以通过 TLS 提供流量：

1. 部署 registry。
2. 获取 registry 的服务 IP 和端口：

```
$ oc get svc/docker-registry
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
docker-registry   ClusterIP   172.30.82.152   <none>        5000/TCP   1d
```

3. 您可以使用现有服务器证书，或者创建对指定 CA 签名的指定 IP 和主机名有效的密钥和服务器证书。要为 registry 服务 IP 和 **docker-registry.default.svc.cluster.local** 主机名创建服务器证书，请在 Ansible 主机清单文件中列出的第一个 master 中运行以下命令，默认为 ***/etc/ansible/hosts***：

```
$ oc adm ca create-server-cert \
```

```
    --signer-cert=/etc/origin/master/ca.crt \
    --signer-key=/etc/origin/master/ca.key \
    --signer-serial=/etc/origin/master/ca.serial.txt \
    --hostnames='docker-registry.default.svc.cluster.local,docker-
registry.default.svc,172.30.124.220' \
    --cert=/etc/secrets/registry.crt \
    --key=/etc/secrets/registry.key
```

如果路由器将在 外部公开，请在 **--hostnames** 标志中添加公共路由主机名：

```
--hostnames='mydocker-registry.example.com,docker-
registry.default.svc.cluster.local,172.30.124.220 \
```

如需有关更新默认证书的更多详细信息，请参阅重新部署 Registry 和路由器 证书，以便可从外部访问该路由。

注意

oc adm ca create-server-cert 命令会生成一个有效期为两年的证书。这可以通过 **--expire-days** 选项进行修改，但出于安全原因，建议不要超过这个值。

4. 为 registry 证书创建 secret：

```
$ oc create secret generic registry-certificates \
    --from-file=/etc/secrets/registry.crt \
    --from-file=/etc/secrets/registry.key
```

5. 将该机密添加到 registry 容器集的服务帐户（包括 **默认** 服务帐户）：

```
$ oc secrets link registry registry-certificates
$ oc secrets link default  registry-certificates
```

注意

默认情况下，“将 secret 仅限于引用它们的服务帐户”的功能被禁用。这意味着，如果在主配置文件中将 **serviceAccountConfig.limitSecretReferences** 设置为 **false**（默认设置），则不需要将 secret 连接到一个特定的服务。

6. 暂停 **docker-registry** 服务：

```
$ oc rollout pause dc/docker-registry
```

7. 将 secret 卷添加到 registry 部署配置中：

```
$ oc set volume dc/docker-registry --add --type=secret \
    --secret-name=registry-certificates -m /etc/secrets
```

8. 通过在 registry 部署配置中添加以下环境变量来启用 TLS：

```
$ oc set env dc/docker-registry \
    REGISTRY_HTTP_TLS_CERTIFICATE=/etc/secrets/registry.crt \
    REGISTRY_HTTP_TLS_KEY=/etc/secrets/registry.key
```

如需更多信息，请参阅 Docker 文档中的配置 registry 部分。

9. 将 registry 的存活度探测使用从 HTTP 更新到 HTTPS:

```
$ oc patch dc/docker-registry -p '{"spec": {"template": {"spec": {"containers":[{
    "name":"registry",
    "livenessProbe":  {"httpGet": {"scheme":"HTTPS"}}
  }]}}}}'
```

10. 如果您的 registry 最初部署在 OpenShift Container Platform 3.2 或更高版本上，请将用于 registry 的就绪度探测从 HTTP 更新到 HTTPS:

```
$ oc patch dc/docker-registry -p '{"spec": {"template": {"spec": {"containers":[{
    "name":"registry",
    "readinessProbe":  {"httpGet": {"scheme":"HTTPS"}}
  }]}}}}'
```

11. 恢复 **docker-registry** 服务：

```
$ oc rollout resume dc/docker-registry
```

12. 验证 registry 是否以 TLS 模式运行。等待最新的 **docker-registry** 部署完成，再验证 registry 容器的 Docker 日志。您应该找到 **listening on :5000, tls** 的条目。

```
$ oc logs dc/docker-registry | grep tls
time="2015-05-27T05:05:53Z" level=info msg="listening on :5000, tls" instance.id=deeba528-c478-41f5-b751-dc48e4935fc2
```

13. 将 CA 证书复制到 Docker 证书目录。这必须在集群的所有节点上完成：

```
$ dcertsdir=/etc/docker/certs.d
$ destdir_addr=$dcertsdir/172.30.124.220:5000
$ destdir_name=$dcertsdir/docker-registry.default.svc.cluster.local:5000

$ sudo mkdir -p $destdir_addr $destdir_name
$ sudo cp ca.crt $destdir_addr    (1)
$ sudo cp ca.crt $destdir_name
```

(1) ***ca.crt*** 文件是 master 上的 ***/etc/origin/master/ca.crt*** 的副本。

14. 在使用身份验证时，一些 **docker** 版本还需要将集群配置为信任操作系统级别的证书。

    a. 复制证书：

    ```
    $ cp /etc/origin/master/ca.crt /etc/pki/ca-trust/source/anchors/myregistrydomain.com.crt
    ```

    b. 运行：

    ```
    $ update-ca-trust enable
    ```

15. 只删除 ***/etc/sysconfig/docker*** 文件中的这个特定 registry 的 **--insecure-registry** 选项。然后，重新载入守护进程并重启 **docker** 服务来反映这个配置更改：

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

16. 验证 **docker** 客户端连接。运行 **docker push** 到 registry 或 registry 中的 **docker pull** 应该会成功。确保您已登录到 registry。

```
$ docker tag|push <registry/image> <internal_registry/project/image>
```

例如：

```
$ docker pull busybox
$ docker tag docker.io/busybox 172.30.124.220:5000/openshift/busybox
$ docker push 172.30.124.220:5000/openshift/busybox
...
cf2616975b4a: Image successfully pushed
Digest: sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f77403cab55
```

2.4.3. 手动公开安全 Registry

您可以通过首先保护 registry，然后使用路由公开它，而不是从 OpenShift Container Platform 集群内部登录到 OpenShift Container Platform registry。您可以使用路由地址从集群以外登陆到 registry，并使用路由主机进行镜像的 tag 和 push 操作。

1. 以下每个先决条件步骤都会在典型的集群安装过程中默认执行。如果没有，请手动执行它们：
   a. 手动部署 registry。
   b. 手动保护 registry 的安全。
   c. 手动部署路由器。
2. 在初始集群安装过程中，默认为 registry 创建 passthrough 路由：
   a. 验证路由是否存在：

```
$ oc get route/docker-registry -o yaml
apiVersion: v1
kind: Route
metadata:
  name: docker-registry
spec:
  host: <host> 1
  to:
    kind: Service
    name: docker-registry 2
  tls:
    termination: passthrough 3
```

1. 路由的主机。您必须能够通过 DNS 外部将此名称解析到路由器的 IP 地址。
2. registry 的服务名称。
3. 将此路由指定为 passthrough 路由。

注意

也支持重新加密路由来公开安全 registry。

b. 如果不存在，则通过 **oc create route passthrough** 命令创建路由，将 registry 指定为路由的服务。默认情况下，创建的路由的名称与服务名称相同：

i. 获取 **docker-registry** 服务详情：

```
$ oc get svc
NAME              CLUSTER_IP      EXTERNAL_IP   PORT(S)                   SELECTOR
AGE
docker-registry   172.30.69.167   <none>        5000/TCP                  docker-
registry=default   4h
kubernetes        172.30.0.1      <none>        443/TCP,53/UDP,53/TCP     <none>
4h
router            172.30.172.132  <none>        80/TCP                    router=router
4h
```

ii. 创建路由：

```
$ oc create route passthrough    \
    --service=docker-registry    \(1)
    --hostname=<host>
route "docker-registry" created    (2)
```

(1) 将 registry 指定为路由的服务。

(2) 路由名称与服务名称相同。

3. 接下来，您必须信任主机系统上用于 registry 的证书，以允许主机推送和拉取镜像。引用的证书是在保护 registry 时创建的。

```
$ sudo mkdir -p /etc/docker/certs.d/<host>
$ sudo cp <ca_certificate_file> /etc/docker/certs.d/<host>
$ sudo systemctl restart docker
```

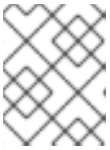
4. 使用保护 registry 的信息，登录到 registry。但是，这一次指向路由中使用的主机名，而不是您的服务 IP。登录到安全且公开的 registry 时，请确保在 **docker login** 命令中指定 registry：

```
# docker login -e user@company.com \
    -u f83j5h6 \
    -p Ju1PeM47R0B92Lk3AZp-bWJSck2F7aGCiZ66aFGZrs2 \
    <host>
```

5. 现在，您可以使用路由主机标记和推送镜像。例如，要在名为 **test** 的项目中标记和推送 **busybox** 镜像：

```
$ oc get imagestreams -n test
NAME      DOCKER REPO   TAGS      UPDATED

$ docker pull busybox
$ docker tag busybox <host>/test/busybox
```

```
$ docker push <host>/test/busybox
The push refers to a repository [<host>/test/busybox] (len: 1)
8c2e06607696: Image already exists
6ce2e90b0bc7: Image successfully pushed
cf2616975b4a: Image successfully pushed
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147d8ca31

$ docker pull <host>/test/busybox
latest: Pulling from <host>/test/busybox
cf2616975b4a: Already exists
6ce2e90b0bc7: Already exists
8c2e06607696: Already exists
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147d8ca31
Status: Image is up to date for <host>/test/busybox:latest

$ oc get imagestreams -n test
NAME      DOCKER REPO                    TAGS      UPDATED
busybox   172.30.11.215:5000/test/busybox   latest    2 seconds ago
```



注意

您的镜像流将具有 registry 服务的 IP 地址和端口，而不是路由名称和端口。详情请参阅 **oc get imagestreams**。

2.4.4. 手动公开 Non-Secure Registry

对于非生产环境的 OpenShift Container Platform，可以公开一个非安全的 registry，而不必通过安全 registry 来公开 registry。这允许您使用一个外部路由到 registry，而无需使用 SSL 证书。

警告

为外部访问非安全的 registry 只适用于非生产环境。

公开一个非安全的 registry：

1. 公开 registry：

```
# oc expose service docker-registry --hostname=<hostname> -n default
```

这会创建以下 JSON 文件：

```
apiVersion: v1
kind: Route
metadata:
  creationTimestamp: null
  labels:
    docker-registry: default
```

```
  name: docker-registry
spec:
  host: registry.example.com
  port:
    targetPort: "5000"
  to:
    kind: Service
    name: docker-registry
status: {}
```

2. 验证路由是否已创建成功：

```
# oc get route
NAME              HOST/PORT                  PATH      SERVICE          LABELS
INSECURE POLICY   TLS TERMINATION
docker-registry   registry.example.com             docker-registry   docker-registry=default
```

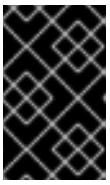
3. 检查 registry 的健康状况：

```
$ curl -v http://registry.example.com/healthz
```

预期为 HTTP 200/OK 消息。

公开 registry 后，通过将端口号添加到 **OPTIONS** 条目来更新您的 */etc/sysconfig/docker* 文件。例如：

```
OPTIONS='--selinux-enabled --insecure-registry=172.30.0.0/16 --insecure-registry
registry.example.com:80'
```



重要

上面的选项应该添加到您要从其登录的客户端上。

此外，确保 Docker 在客户端上运行。

登录到非安全且公开的 registry 时，请确保在 **docker login** 命令中指定 registry。例如：

```
# docker login -e user@company.com \
    -u f83j5h6 \
    -p Ju1PeM47R0B92Lk3AZp-bWJSck2F7aGCiZ66aFGZrs2 \
    <host>
```

2.5. 扩展 REGISTRY 配置

2.5.1. 维护 Registry IP 地址

OpenShift Container Platform 使用其服务 IP 地址来引用集成的 registry，因此如果您决定删除并重新创建 **docker-registry** 服务，您可以通过安排在新服务中重复使用旧 IP 地址来确保完全透明转换。如果无法避免新的 IP 地址，您可以通过只重启 master 来最小化集群中断。

重新使用地址

要重新使用 IP 地址，您必须在删除旧 **docker-registry** 服务前保存其 IP 地址，并安排将新分配的 IP 地址替换为新 **docker-registry** 服务中保存的 IP 地址。

1. 记录该服务的 **clusterIP**:

```
$ oc get svc/docker-registry -o yaml | grep clusterIP:
```

2. 删除服务：

```
$ oc delete svc/docker-registry dc/docker-registry
```

3. 在 ***registry.yaml*** 中创建 registry 定义，将 **<options>** 替换为非生产环境使用部分中说明的第 3 步中使用的 ：

```
$ oc adm registry <options> -o yaml > registry.yaml
```

4. 编辑 ***registry.yaml***，找到 **Service**，并将其 **clusterIP** 更改为第 1 步中记录的地址。

5. 使用修改后的 ***registry.yaml*** 创建 registry：

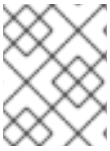
```
$ oc create -f registry.yaml
```

重启 Master

如果您无法重新使用 IP 地址，则任何使用包含旧 IP 地址的 pull specification 的操作都将失败。要最小化集群中断，您必须重启 master:

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

这样可确保从缓存中清除包括旧 IP 地址的旧 registry URL。



注意

我们建议不要重启整个集群，因为这会对 pod 造成不必要的停机时间，实际上不会清除缓存。

2.5.2. 配置外部 registry 搜索列表

您可以使用 ***/etc/containers/registries.conf*** 文件创建 Docker Registry 列表来搜索容器镜像。

/etc/containers/registries.conf 文件是 OpenShift Container Platform 在用户使用镜像简短名称（如 **myimage:latest** ）提取镜像时应对其搜索的 registry 服务器列表。您可以自定义搜索顺序，指定安全且不安全的 registry，并定义一个被阻断的 registry 列表。OpenShift Container Platform 不允许搜索或允许从被阻塞列表中的 registry 中拉取(pull)。

例如，如果用户想要拉取 **myimage:latest** 镜像，OpenShift Container Platform 会按照它们出现在列表中的顺序搜索 registry，直到找到 **myimage:latest** 镜像。

registry 搜索列表允许您策展一组可供 OpenShift Container Platform 用户下载的镜像和模板。您可以将这些镜像放在一个或多个 Docker registry 中，将 registry 添加到列表中，并将这些镜像拉取到集群中。

注意

使用 registry 搜索列表时，OpenShift Container Platform 不会从不在搜索列表中的 registry 中拉取镜像。

配置 registry 搜索列表：

1. 编辑 */etc/containers/registries.conf* 文件，根据需要添加或编辑以下参数：

```
[registries.search] 1
registries = ["reg1.example.com", "reg2.example.com"]

[registries.insecure] 2
registries = ["reg3.example.com"]

[registries.block] 3
registries = ['docker.io']
```

1. 指定用户可以使用 SSL/TLS 从中下载镜像的安全 registry。
2. 指定用户可以在不使用 TLS 的情况下下载镜像的不安全 registry。
3. 指定用户无法从中下载镜像的 registry。

2.5.3. 设置 Registry 主机名

您可以为内部和外部引用配置 registry 的主机名和端口。通过这样做，镜像流将为镜像提供基于主机名的推送和拉取规格，允许使用镜像与对 registry 服务 IP 的更改进行隔离，并有可能允许镜像流及其引用在集群中可移植。

要设置从集群内部引用 registry 的主机名，请在 master 配置文件的 **imagePolicyConfig** 部分中设置 **internalRegistryHostname**。外部主机名通过在同一位置设置 **externalRegistryHostname** 值来控制。

镜像策略配置

```
imagePolicyConfig:
  internalRegistryHostname: docker-registry.default.svc.cluster.local:5000
  externalRegistryHostname: docker-registry.mycompany.com
```

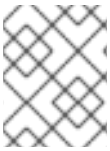
registry 本身必须使用相同的内部主机名值进行配置。这可以通过在 registry 署配置中设置 **REGISTRY_OPENSHIFT_SERVER_ADDR** 环境变量或设置 registry 配置的 OpenShift 部分中的值来完成。

注意

如果您已经为 registry 启用了 TLS，服务器证书必须包含您希望引用 registry 的主机名。有关添加主机名到服务器证书的说明，请参阅保护registry。

2.5.4. 覆盖 Registry 配置

您可以使用自己的自定义配置覆盖集成 registry 的默认配置，默认覆盖正在运行的 registry 容器中的 */config.yml* 中的默认配置。



注意

此文件中的上游配置选项也可使用环境变量覆盖。middleware 部分是一个例外，因为只有几个选项可以使用环境变量覆盖。了解如何覆盖特定的配置选项。

要启用对 registry 配置文件的管理，并使用 **ConfigMap** 部署更新的配置：

1. 部署 registry。
2. 根据需要在本地编辑 registry 配置文件。registry 中部署的初始 YAML 文件如下所示：查看支持的选项。

   registry 配置文件

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /registry
  delete:
    enabled: true
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
      options:
        acceptschema2: true
        pullthrough: true
        enforcequota: false
        projectcachettl: 1m
        blobrepositorycachettl: 10m
  storage:
    - name: openshift
openshift:
  version: 1.0
  metrics:
    enabled: false
    secret: <secret>
```

3. 创建一个 **ConfigMap**，包含此目录中每个文件的内容：

```
$ oc create configmap registry-config \
    --from-file=</path/to/custom/registry/config.yml>/
```

4. 将 **registry-config** ConfigMap 作为卷添加到 registry 的部署配置中，以在 */etc/docker/registry/* 中挂载自定义配置文件：

```
$ oc set volume dc/docker-registry --add --type=configmap \
    --configmap-name=registry-config -m /etc/docker/registry/
```

5. 通过在 registry 的部署配置中添加以下环境变量来更新 registry，以引用上一步中的配置路径：

```
$ oc set env dc/docker-registry \
    REGISTRY_CONFIGURATION_PATH=/etc/docker/registry/config.yml
```

这可以作为迭代过程执行，从而实现所需的配置。例如，在故障排除期间，可临时更新配置以将其置于 **debug** 模式。

更新现有配置：

警告

此流程将覆盖当前部署的 registry 配置。

1. 编辑本地 registry 配置文件 ***config.yml***。
2. 删除 **registry-config** configmap：

```
$ oc delete configmap registry-config
```

3. 重新创建 configmap 以引用更新的配置文件：

```
$ oc create configmap registry-config\
    --from-file=</path/to/custom/registry/config.yml>/
```

4. 重新部署 registry 以读取更新的配置：

```
$ oc rollout latest docker-registry
```

提示

在源代码控制存储库中维护配置文件。

2.5.5. registry 配置参考

上游 docker distribution 库中有许多配置选项。并非所有配置选项都受到支持或启用。覆盖 registry 配置时，请使用本节作为参考。

注意

此文件中的上游配置选项也可使用环境变量覆盖。但是，middleware 部分可能**无法**使用环境变量覆盖。了解如何覆盖特定的配置选项。

2.5.5.1. Log

支持上游选项。

例如：

```
log:
  level: debug
  formatter: text
  fields:
    service: registry
    environment: staging
```

2.5.5.2. Hook

不支持邮件 hook。

2.5.5.3. 存储

本节列出了支持的 registry 存储驱动程序。如需更多信息，请参阅容器镜像 registry 文档。

以下列表包括在 registry 配置文件中需要配置的存储驱动程序：

- 文件系统。文件系统是默认设置，不需要进行配置。
- S3.如需更多信息，请参阅 CloudFront 配置 文档。
- OpenStack Swift
- Google Cloud Storage(GCS)
- Microsoft Azure
- Aliyun OSS

支持常规 registry 存储配置选项。如需更多信息，请参阅容器镜像 registry 文档。

以下存储选项需要通过文件系统驱动程序配置：

- GlusterFS Storage
- Ceph Rados 块设备

注意

如需有关支持的持久性存储驱动程序的更多信息，请参阅配置持久性存储和持久性存储示例。

常规存储配置选项

```
storage:
  delete:
    enabled: true (1)
  redirect:
    disable: false
  cache:
```

```
    blobdescriptor: inmemory
  maintenance:
    uploadpurging:
      enabled: true
      age: 168h
      interval: 24h
      dryrun: false
    readonly:
      enabled: false
```

1 此条目对于镜像修剪正常工作是**必需的**。

2.5.5.4. Auth

不应更改 Auth 选项。**openshift** 扩展是唯一支持的选项。

```
auth:
  openshift:
    realm: openshift
```

2.5.5.5. Middleware

存储库 中间件扩展允许配置负责与 OpenShift Container Platform 和镜像代理交互的 OpenShift Container Platform 中间件。

```
middleware:
  registry:
    - name: openshift 1
  repository:
    - name: openshift 2
      options:
        acceptschema2: true 3
        pullthrough: true 4
        mirrorpullthrough: true 5
        enforcequota: false 6
        projectcachettl: 1m 7
        blobrepositorycachettl: 10m 8
  storage:
    - name: openshift 9
```

1 2 9 这些条目是必需的。其存在可确保加载所需的组件。这些值不应更改。

3 允许您在推送到 registry 期间存储 清单模式 v2。详情请查看 下方。

4 允许 registry 充当远程 Blob 的代理。详情请查看 下方。

5 允许从远程 registry 提供 registry 缓存 Blob，以便稍后进行快速访问。镜像在第一次访问 Blob 时开始。如果禁用了 pullthrough，则选项无效。

6 防止 Blob 上传超过目标项目中定义的大小限制。

7 在 registry 中缓存的限制的过期超时。值越低，限制更改传播到 registry 所需的时间就会越短。但是，registry 会更频繁地从服务器查询限制，因此推送速度会较慢。

8 blob 和 repository 之间记住关联的过期超时。值越大，快速查找的几率更高，registry 操作效率越高。另一方面，内存使用会增加为用户提供服务的风险，因为用户不再获得访问该层的权限。

2.5.5.5.1. S3 驱动程序配置

如果要使用您正在使用的集成 registry 不支持的 S3 区域，您可以指定一个 **regionendpoint** 以避免 **区域** 验证错误。

有关使用 Amazon Simple Storage Service 存储的更多信息，请参阅 Amazon S3 作为存储后端。

例如：

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
  s3:
    accesskey: BJKMSZBRESWJQXRWMAEQ
    secretkey: 5ah5I91SNXbeoUXXDasFtadRqOdy62JzlnOW1goS
    bucket: docker.myregistry.com
    region: eu-west-3
    regionendpoint: https://s3.eu-west-3.amazonaws.com
 auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
  storage:
    - name: openshift
```

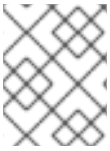
注意

验证 **region** 和 **regionendpoint** 字段本身是否一致。否则，集成的 registry 将启动，但它无法读取或写入 S3 存储。

如果您使用与 Amazon S3 不同的 S3 存储，则 **regionendpoint** 也很有用。

2.5.5.5.2. CloudFront 中间件

可以添加 **CloudFront 中间件扩展** 以支持 AWS CloudFront CDN 存储供应商。CloudFront 中间件加快了镜像内容在国际间分发。Blob 分布到世界各地的几个边缘位置。客户端始终定向到延迟最低的边缘。



注意

CloudFront 中间件扩展 只能用于 S3 存储。它仅在 Blob 服务期间使用。因此，只有 Blob 下载可以加快，不能上传。

以下是带有 CloudFront 中间件的 S3 存储驱动程序的最小配置示例：

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
  s3: (1)
    accesskey: BJKMSZBRESWJQXRWMAEQ
    secretkey: 5ah5l91SNXbeoUXXDasFtadRqOdy62JzlnOW1goS
    region: us-east-1
    bucket: docker.myregistry.com
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
  storage:
    - name: cloudfront (2)
      options:
        baseurl: https://jrpbyn0k5k88bi.cloudfront.net/ (3)
        privatekey: /etc/docker/cloudfront-ABCEDFGHIJKLMNOPQRST.pem (4)
        keypairid: ABCEDFGHIJKLMNOPQRST (5)
    - name: openshift
```

(1) 无论 CloudFront 中间件如何，S3 存储都必须以相同的方式进行配置。

(2) OpenShift 中间件需要先列出 CloudFront 存储中间件。

(3) CloudFront 基本 URL。在 AWS 管理控制台中，这被列为 CloudFront 发行版本的 **Domain Name**。

(4) AWS 私钥在文件系统中的位置。这不能与 Amazon EC2 密钥对混淆。请参阅 AWS 文档 为您的可信签名者创建 CloudFront 密钥对。文件需要作为 机密 挂载到 registry 容器集。

(5) 云前端密钥对的 ID。

2.5.5.5.3. 覆盖中间件配置选项

middleware 部分无法使用环境变量覆盖。但也有一些例外：例如：

```
middleware:
```

```
repository:
 - name: openshift
   options:
     acceptschema2: true 1
     pullthrough: true 2
     mirrorpullthrough: true 3
     enforcequota: false 4
     projectcachettl: 1m 5
     blobrepositorycachettl: 10m 6
```

1 此配置选项可以被布尔值环境变量 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ACCEPTSCHEMA2** 覆盖，允许在清单放置请求时接受清单架构 v2。可识别的值为 **true** 和 **false**（适用于以下所有其他布尔值变量）。

2 此配置选项可以被布尔值环境变量 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_PULLTHROUGH** 覆盖，启用远程存储库的代理模式。

3 此配置选项可以被布尔值环境变量 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_MIRRORPULLTHROUGH** 覆盖，它指示 registry 在提供远程 Blob 时在本地镜像 Blob。

4 此配置选项可以被布尔值环境变量 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA** 覆盖，允许开启或关闭配额执行。默认情况下，配额强制是禁用的。

5 可被环境变量 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_PROJECTCACHETTL** 覆盖的配置选项，为项目配额对象指定驱除超时。它需要一个有效的持续时间字符串（如 **2m**）。如果为空，则获得默认超时。如果零(**0m**)，则禁用缓存。

6 可被环境变量 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_BLOBREPOSITORYCACHETTL** 覆盖的配置选项，为 blob 和包含存储库之间的关联指定驱除超时。值的格式与 **projectcachettl** 示例中的格式相同。

2.5.5.5.4. 镜像 Pullthrough

如果启用，registry 将尝试从远程 registry 获取请求的 blob，除非 blob 在本地存在。远程候选从存储在镜像流 状态的 **DockerImage** 条目（客户端从中拉取）计算。此类条目中的所有唯一远程 registry 引用将依次尝试，直到找到 Blob。

只有存在拉取镜像的镜像流标签时才会进行 pullthrough。例如，如果拉取的镜像为 **docker-registry.default.svc:5000/yourproject/yourimage:prod**，则 registry 将在项目 **yourproject** 中查找名为 **yourimage:prod** 的镜像流标签。如果找到镜像，它将尝试使用该镜像流标签关联的 **dockerImageReference** 来拉取镜像。

在执行 pullthrough 时，registry 将使用与所引用镜像流标签关联的项目中找到的拉取凭据。此功能还允许您拉取驻留在 registry 上的镜像，它们没有凭证可以访问，只要您有权访问引用该镜像的镜像流标签。

您必须确保 registry 具有适当的证书来信任您针对的任何外部 registry。证书需要放置在 pod 上的 */etc/pki/tls/certs* 目录中。您可以使用 配置映射 或 secret 挂载证书。请注意，必须替换整个 */etc/pki/tls/certs* 目录。您必须包含新证书，并在您挂载的 secret 或配置映射中替换系统证书。

请注意，默认情况下，镜像流标签使用引用策略类型 **Source**，这意味着镜像流引用被解析为镜像拉取规格时，使用的规格将指向镜像的来源。对于托管在外部 registry 上的镜像，这将是外部 registry，因此该

资源将引用外部 registry 并拉取镜像。例如，**registry.redhat.io/openshift3/jenkins-2-rhel7** 和 pullthrough 将不适用。为确保引用镜像流的资源使用指向内部 Registry 的拉取规格，镜像流标签应使用引用策略类型 **Local**。有关参考政策的更多信息，

这个功能默认是开启的。不过，它可以通过配置选项来禁用。

默认情况下，除非禁用 **mirrorpullthrough**，否则通过这种方式提供的所有远程 Blob 都存储在本地以进行后续的快速访问。此镜像功能的缺点是增加存储的使用。

注意

当客户端尝试至少获取一个 blob 字节时，镜像开始。要在实际需要前将特定镜像预先放入集成的 registry 中，您可以运行以下命令：

```
$ oc get imagestreamtag/${IS}:${TAG} -o jsonpath='{
.image.dockerImageLayers[*].name }' | \
  xargs -n1 -I {} curl -H "Range: bytes=0-1" -u user:${TOKEN} \
  http://${REGISTRY_IP}:${PORT}/v2/default/mysql/blobs/{}
```

注意

此 OpenShift Container Platform 镜像功能不应与上游 registry 通过缓存功能拉取(pull) 混淆，后者是类似但不同的功能。

2.5.5.5.5. 清单架构 v2 支持

每一镜像具有一个清单，用于描述其 Blob、运行它的说明以及其他元数据。清单是经过版本控制的，每个版本具有不同的结构和字段，随着时间推移而发展。同一镜像可由多个清单版本表示。每个版本都有不同的摘要。

registry 目前支持 清单 v2 模式 1 (**schema1)**和 清单 v2 模式 2 （**架构2**）。前者已被淘汰，但将会延长支持时间。

默认配置是存储 **schema2**。

您应注意与各种 Docker 客户端的兼容性问题：

- 1.9 或更早版本的 Docker 客户端仅支持 **schema1**。此客户端拉取或推送的任何清单都将是这种传统架构。
- 版本 1.10 的 Docker 客户端支持 **schema1** 和 **schema2**。默认情况下，如果支持较新的架构，它会将后者推送到 registry。

存储具有 **schema1** 的镜像的 registry 将始终不动地将其返回到客户端。**Schema2** 将仅原封不动传输到较新的 Docker 客户端。对于较旧版本，它将自行转换为 **schema1**。

这具有显著的后果。例如，通过较新的 Docker 客户端推送到 registry 的镜像无法被旧 Docker 通过其摘要拉取。这是因为存储的镜像的清单是 **schema2**，其摘要只能用于拉取此版本的清单。

旦您确信所有 registry 客户端都支持 **schema2**，就可以在 registry 中启用其支持。有关特定选项，请参见上述中间件配置参考。

2.5.5.6. OpenShift

本节回顾针对特定于 OpenShift Container Platform 的功能的全局设置配置。在以后的发行版本中，Middleware 部分中与 **openshift**相关的设置将被弃用。

目前，本节允许您配置 registry 指标集合：

```
openshift:
 version: 1.0 (1)
 server:
  addr: docker-registry.default.svc (2)
 metrics:
  enabled: false (3)
  secret: <secret> (4)
 requests:
  read:
   maxrunning: 10 (5)
   maxinqueue: 10 (6)
   maxwaitinqueue 2m (7)
  write:
   maxrunning: 10 (8)
   maxinqueue: 10 (9)
   maxwaitinqueue 2m (10)
```

(1) 指定本节配置版本的必填条目。唯一支持的值是 **1.0**。

(2) registry 的主机名。应设置为 master 上配置的相同值。它可以被环境变量 **REGISTRY_OPENSHIFT_SERVER_ADDR** 覆盖。

(3) 可以设置为 **true** 来启用指标集合。它可以被布尔值环境变量 **REGISTRY_OPENSHIFT_METRICS_ENABLED** 覆盖。

(4) 用于授权客户端请求的机密。指标客户端必须在 **Authorization** 标头中将它用作 bearer 令牌。它可以被环境变量 **REGISTRY_OPENSHIFT_METRICS_SECRET** 覆盖。

(5) 同时拉取请求的最大数量。它可以被环境变量 **REGISTRY_OPENSHIFT_REQUESTS_READ_MAXRUNNING** 覆盖。零表示无限制。

(6) 已排队拉取请求的最大数量。它可以被环境变量 **REGISTRY_OPENSHIFT_REQUESTS_READ_MAXINQUEUE** 覆盖。零表示无限制。

(7) 拉取请求在被拒绝前可在队列中等待的最长时间。它可以被环境变量 **REGISTRY_OPENSHIFT_REQUESTS_READ_MAXWAITINQUEUE** 覆盖。零表示无限制。

(8) 同时推送请求的最大数量。它可以被环境变量 **REGISTRY_OPENSHIFT_REQUESTS_WRITE_MAXRUNNING** 覆盖。零表示无限制。

(9) 最多排队的推送请求数.它可以被环境变量 **REGISTRY_OPENSHIFT_REQUESTS_WRITE_MAXINQUEUE** 覆盖。零表示无限制。

(10) 推送请求在被拒绝前可在队列中等待的最长时间。它可以被环境变量 **REGISTRY_OPENSHIFT_REQUESTS_WRITE_MAXWAITINQUEUE** 覆盖。零表示无限制。

有关使用信息，请参阅访问 Registry Metrics。

2.5.5.7. 报告

报告不受支持。

2.5.5.8. HTTP

支持上游选项。了解如何通过环境变量更改这些设置。应当仅更改 **tls** 部分。例如：

```
http:
  addr: :5000
  tls:
    certificate: /etc/secrets/registry.crt
    key: /etc/secrets/registry.key
```

2.5.5.9. 通知

支持上游选项。REST API 参考 提供了更加全面的集成选项。

例如：

```
notifications:
  endpoints:
    - name: registry
      disabled: false
      url: https://url:port/path
      headers:
        Accept:
          - text/plain
      timeout: 500
      threshold: 5
      backoff: 1000
```

2.5.5.10. Redis

不支持 Redis。

2.5.5.11. Health

支持上游选项。registry 部署配置在 **/healthz** 上提供了一个集成健康检查。

2.5.5.12. Proxy

不应启用代理配置。此功能由 OpenShift Container Platform 存储库中间件扩展 **pullthrough: true** 提供。

2.5.5.13. Cache

集成 registry 会主动缓存数据，以减少对减慢外部资源的调用数量。有两个缓存：

1. 用于缓存 Blob 元数据的存储缓存。此缓存没有过期时间，数据会一直存在，直到显式删除为止。
2. 应用缓存包含 blob 和存储库之间的关联。此缓存中的数据具有过期时间。

要完全关闭缓存，您需要更改配置：

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: "" 1
openshift:
  version: 1.0
  cache:
    disabled: true 2
    blobrepositoryttl: 10m
```

1 禁用存储后端中访问的元数据的缓存。如果没有此缓存，registry 服务器将持续访问元数据后端。

2 禁用包含 blob 和存储库关联的缓存。如果没有这个缓存，registry 服务器将持续从主 API 重新查询数据并重新计算关联。

2.6. 已知问题

2.6.1. 概述

以下是部署或使用集成 registry 时已知的问题。

2.6.2. 使用 Registry Pull-through 的并发构建

本地 **docker-registry** 部署会增加负载。默认情况下，它现在缓存 ***registry.redhat.io*** 的内容。来自为 STI 构建的 ***registry.redhat.io*** 的镜像现在存储在本地 registry 中。尝试拉取它们的结果从本地 **docker-registry** 中拉取。因此，在有些情况下，并发构建可能会造成拉取超时，构建可能会失败。要缓解此问题，请将 **docker-registry** 部署扩展到多个副本。检查构建器容器集日志中的超时。

2.6.3. 使用共享 NFS 卷扩展 registry 的镜像推送错误

当将扩展的 registry 与共享 NFS 卷搭配使用时，您可能会在推送镜像的过程中看到以下错误之一：

- **digest invalid: provided digest did not match uploaded content**
- **blob upload unknown**
- **blob upload invalid**

当 Docker 尝试推送镜像时，内部 registry 服务将返回这些错误。其原因源自于跨节点的文件属性同步。使用默认循环负载平衡配置推送镜像时可能会出现的错误，如 NFS 客户端缓存、网络延迟和层大小等因素。

您可以执行以下步骤来最大程度减少此类故障的可能性：

1. 确保 **docker-registry** 服务的 **sessionAffinity** 设置为 **ClientIP**:

```
$ oc get svc/docker-registry --template='{{.spec.sessionAffinity}}'
```

这应该会返回 **ClientIP**，这是 OpenShift Container Platform 近期版本中的默认设置。如果没有，请修改它：

```
$ oc patch svc/docker-registry -p '{"spec":{"sessionAffinity": "ClientIP"}}'
```

2. 确保 NFS 服务器上的 registry 卷的 NFS 导出行列出 **no_wdelay** 选项。**no_wdelay** 选项可防止服务器延迟写入，这极大提高了读写一致性，这是 Registry 的一项要求。

重要

测试显示，使用 RHEL NFS 服务器作为容器镜像 registry 的存储后端会出现问题。这包括 OpenShift Container Registry 和 Quay。因此，不建议使用 RHEL NFS 服务器来备份核心服务使用的 PV。

市场上的其他 NFS 实现可能没有这些问题。如需了解更多与此问题相关的信息，请联络相关的 NFS 厂商。

2.6.4. 使用 "not found" 错误来拉取内部管理的镜像故障

当拉取镜像推送到不同于从中拉取的镜像流时，会发生此错误。这是因为将构建的镜像重新标记到任意镜像流中：

```
$ oc tag srcimagestream:latest anyproject/pullimagestream:latest
```

然后，使用如下镜像引用从中提取：

```
internal.registry.url:5000/anyproject/pullimagestream:latest
```

在手动 Docker 拉取过程中，这会导致类似的错误：

```
Error: image anyproject/pullimagestream:latest not found
```

为防止这种情况，请完全避免标记内部管理的镜像，或者 手动 将构建的镜像重新推送到所需的命名空间。

2.6.5. S3 Storage 上带有 "500 Internal Server Error"的镜像 Push Fails

当 registry 在 S3 存储后端上运行时，会报告问题。推送到容器镜像 registry 有时会失败，并显示以下错误：

```
Received unexpected HTTP status: 500 Internal Server Error
```

要进行调试，您需要 查看 registry 日志。在这里，查看推送失败时出现的类似的错误消息：

```
time="2016-03-30T15:01:21.22287816-04:00" level=error msg="unknown error completing upload:
driver.Error{DriverName:\"s3\", Enclosed:(*url.Error)(0xc20901cea0)}" http.request.method=PUT
...
time="2016-03-30T15:01:21.493067808-04:00" level=error msg="response completed with error"
err.code=UNKNOWN err.detail="s3: Put https://s3.amazonaws.com/oso-tsi-
docker/registry/docker/registry/v2/blobs/sha256/ab/abe5af443833d60cf672e2ac57589410dddec060ed7
25d3e676f1865af63d2e2/data: EOF" err.message="unknown error" http.request.method=PUT
...
```

```
time="2016-04-02T07:01:46.056520049-04:00" level=error msg="error putting into main store: s3:
The request signature we calculated does not match the signature you provided. Check your key and
signing method." http.request.method=PUT
atest
```

如果您看到此类错误，请联系您的 Amazon S3 支持。您所在地区或您的特定存储桶可能有问题。

2.6.6. 镜像修剪失败

如果您在修剪镜像时遇到以下错误：

```
BLOB sha256:49638d540b2b62f3b01c388e9d8134c55493b1fa659ed84e97cb59b87a6b8e6c error
deleting blob
```

您的 registry 日志 包含以下信息：

```
error deleting blob
\"sha256:49638d540b2b62f3b01c388e9d8134c55493b1fa659ed84e97cb59b87a6b8e6c\": operation
unsupported
```

这意味着您的 自定义配置文件 缺少 storage 部分中的强制条目，即 **storage:delete:enabled** 设置为 **true**。添加它们，重新部署 registry，再重复您的镜像修剪操作。

第 3 章 设置路由器

3.1. 路由器概述

3.1.1. 关于路由器

可以通过多种方式使 流量进入集群。最常见的方法是使用 OpenShift Container Platform router 作为 OpenShift Container Platform 安装中 services 的外部流量入口点。

OpenShift Container Platform 提供和支持以下路由器插件：

- HAProxy 模板路由器是默认的插件。它使用 **openshift3/ose-haproxy-router** 镜像和 OpenShift Container Platform 上的容器内的模板路由器插件运行 HAProxy 实例。它目前支持通过 SNI 的 HTTP(S)流量和启用 TLS 的流量。路由器的容器侦听主机网络接口，这与大多数仅侦听私有 IP 的容器不同。路由器将路由名称的外部请求代理到与路由关联的服务标识的实际 pod 的 IP。
- F5 路由器 可与您环境中的现有 **F5 BIG-IP®** 系统集成，以同步路由。需要 **F5 BIG-IP®** 版本 11.4 或更高版本才能使用 F5 iControl REST API。
- 部署默认 HAProxy 路由器
- 部署自定义 HAProxy 路由器
- 将 HAProxy 路由器配置为使用 PROXY 协议
- 配置路由超时

3.1.2. 路由器服务帐户

在部署 OpenShift Container Platform 集群前，您必须有一个路由器的服务帐户，该帐户会在集群安装过程中自动创建。此服务帐户具有 安全性上下文约束 (SCC)的权限，允许它指定主机端口。

3.1.2.1. 访问标签的权限

使用 命名空间标签 时，例如在创建 路由器分片 时，路由器的服务帐户必须具有 **cluster-reader** 权限。

```
$ oc adm policy add-cluster-role-to-user \
    cluster-reader \
    system:serviceaccount:default:router
```

服务帐户就位后，您可以继续安装默认的 HAProxy 路由器 或自定义 HAProxy 路由器

3.2. 使用默认 HAPROXY 路由器

3.2.1. 概述

oc adm router 命令随管理员 CLI 一同提供，以简化在新安装中设置路由器的任务。**oc adm router** 命令创建服务和部署配置对象。使用 **--service-account** 选项指定路由器用于联系主控机的服务帐户。

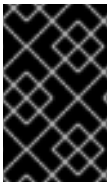
路由器服务帐户 可以提前创建，或者通过 **oc adm router --service-account** 命令创建。

OpenShift Container Platform 组件之间的每种通信都受到 TLS 的保护，并使用各种证书和身份验证方法。可以提供 **--default-certificate** .pem 格式文件，或者由 **oc adm router** 命令创建文件。创建路由时，用户可以提供路由器在处理路由时使用的路由证书。

重要

删除路由器时，请确保也删除了部署配置、服务和机密。

路由器部署到特定的节点上。这使得集群管理员和外部网络管理器能够更轻松地协调哪个 IP 地址将运行路由器以及路由器将处理的流量。路由器通过使用 节点选择器 部署到特定的节点上。



重要

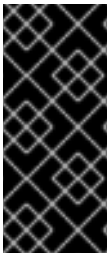
路由器默认使用主机网络，它们直接连接到主机上所有接口上的端口 80 和 443。将路由器限制为端口 80/443 可用且未被其他服务使用的主机，再使用 节点选择器 和 调度程序配置 进行设置。例如，您可以通过专用基础架构节点来运行路由器等服务来达到此目的。

重要

建议将不同的 **openshift-router** 服务帐户与路由器分开。这可以通过 **oc adm router** 命令的 **--service-account** 标志来提供。

```
$ oc adm router --dry-run --service-account=router 1
```

1 **--service-account** 是 **openshift-router** 服务帐户 的名称。



重要

使用 **oc adm router** 创建的路由器 pod 具有默认的资源请求，节点必须满足这些请求才能部署路由器 pod。为提高基础架构组件的可靠性，默认资源请求用于在不使用资源请求的情况下增加 pod 以上的路由器容器集的 QoS 层。默认值表示部署基本路由器所需的已观察到最少资源，可以在路由器部署配置中编辑，并且您可能希望根据路由器负载来增加这些资源。

3.2.2. 创建路由器

如果路由器不存在，请运行以下命令来创建路由器：

```
$ oc adm router <router_name> --replicas=<number> --service-account=router --extended-
logging=true
```

除非创建 高可用性 配置，否则 **--replicas** 通常为 **1**。

--extended-logging=true 配置路由器，将 HAProxy 生成的日志转发到 syslog 容器。

查找路由器的主机 IP 地址：

```
$ oc get po <router-pod>  --template={{.status.hostIP}}
```

您还可以使用路由器分片来确保路由器被过滤到特定的命名空间或路由，或者在路由器创建后设置任何环境变量。本例中为每个分片创建一个路由器。

3.2.3. 其他基本路由器命令

检查默认路由器

名为 **router** 的默认路由器服务帐户会在集群安装过程中自动创建。验证此帐户是否已存在：

```
$ oc adm router --dry-run --service-account=router
```

查看默认路由器

查看创建的默认路由器会是什么样子：

```
$ oc adm router --dry-run -o yaml --service-account=router
```

将路由器配置为转发 HAProxy 日志

您可以将路由器配置为将 HAProxy 生成的日志转发到 rsyslog sidecar 容器。**--extended-logging=true** 参数附加 syslog 容器将 HAProxy 日志转发到标准输出。

```
$ oc adm router --extended-logging=true
```

以下示例是使用 **--extended-logging=true** 的路由器的配置：

```
$ oc get pod router-1-xhdb9 -o yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  - env:

    ....

    - name: ROUTER_SYSLOG_ADDRESS (1)
      value: /var/lib/rsyslog/rsyslog.sock

    ....

  - command: (2)
    - /sbin/rsyslogd
    - -n
    - -i
    - /tmp/rsyslog.pid
    - -f
    - /etc/rsyslog/rsyslog.conf
    image: registry.redhat.io/openshift3/ose-haproxy-router:v3.11.188
    imagePullPolicy: IfNotPresent
    name: syslog
```

1. **--extended-logging=true** 参数为日志创建一个套接字文件。
2. **--extended-logging=true** 参数向路由器添加一个容器。在容器中，rsyslog 进程作为 **/sbin/rsyslogd -n -i /tmp/rsyslog.pid -f /etc/rsyslog/rsyslog.conf** 运行。

使用以下命令查看 HAProxy 日志：

```
$ oc set env dc/test-router ROUTER_LOG_LEVEL=info (1)
$ oc logs -f <pod-name> -c syslog (2)
```

1. 将日志级别设置为 **info** 或 **debug**。默认值为 **warning**。
2. 指定要查看日志的路由器 pod 的名称。

HAProxy 日志的格式如下：

```
2020-04-14T03:05:36.629527+00:00 test-311-node-1 haproxy[43]: 10.0.151.166:59594
[14/Apr/2020:03:05:36.627] fe_no_sni~ be_secure:openshift-console:console/pod:console-
b475748cb-t6qkq:console:10.128.0.5:8443 0/0/1/1/2 200 393 - - --NI 2/1/0/1/0 0/0 "HEAD / HTTP/1.1"
2020-04-14T03:05:36.633024+00:00 test-311-node-1 haproxy[43]: 10.0.151.166:59594
[14/Apr/2020:03:05:36.528] public_ssl be_no_sni/fe_no_sni 95/1/104 2793 -- 1/1/0/0/0 0/0
```

将路由器部署到标记的节点

将路由器部署到与指定 节点标签 匹配的任何节点上：

```
$ oc adm router <router_name> --replicas=<number> --selector=<label> \
    --service-account=router
```

例如，如果要创建一个名为 **router** 的路由器，并将其放置到带有 **node-role.kubernetes.io/infra=true** 标签的节点上：

```
$ oc adm router router --replicas=1 --selector='node-role.kubernetes.io/infra=true' \
  --service-account=router
```

在集群安装过程中，**openshift_router_selector** 和 **openshift_registry_selector** Ansible 设置默认设置为 **node-role.kubernetes.io/infra=true**。只有在存在与 **node-role.kubernetes.io/infra=true** 标签匹配的节点时，才会自动部署默认路由器和 registry。

有关更新标签的详情，请参阅更新节点上的标签。

根据调度程序策略，在不同的主机上创建多个实例。

使用不同的路由器镜像

使用其他路由器镜像并查看要使用的路由器配置：

```
$ oc adm router <router_name> -o <format> --images=<image> \
    --service-account=router
```

例如：

```
$ oc adm router region-west -o yaml --images=myrepo/somerouter:mytag \
    --service-account=router
```

3.2.4. 过滤到特定路由器的路由

使用 **ROUTE_LABELS** 环境变量，您可以过滤路由，使其仅供特定路由器使用。

例如，如果您有多个路由器和 100 个路由，您可以将标签附加到路由，以便一部分标签由一个路由器处理，而其余则由另一个路由器处理。

1. 创建路由器 后，使用 **ROUTE_LABELS** 环境变量标记路由器：

```
$ oc set env dc/<router=name>  ROUTE_LABELS="key=value"
```

2. 将标签添加到所需路由中：

```
oc label route <route=name> key=value
```

3. 要验证标签是否已附加到路由，请检查路由配置：

```
$ oc describe route/<route_name>
```

设置最大并发连接数

默认情况下，路由器可以处理最多 20000 连接。您可以根据需要更改该限制。由于连接太少，健康检查无法正常工作，从而导致不必要的重启。您需要配置系统，以支持最大连接数。**'sysctl fs.nr_open'** 和 **'sysctl fs.file-max'** 中显示的限值必须足够大。否则，HAproxy 将不会启动。

创建路由器时，**--max-connections=** 选项会设置所需的限制：

```
$ oc adm router --max-connections=10000  ....
```

编辑路由器部署配置中的 **ROUTER_MAX_CONNECTIONS** 环境变量，以更改值。路由器容器集使用新值重启。如果 **ROUTER_MAX_CONNECTIONS** 不存在，则使用默认值 20000。

注意

连接包括 frontend 和 internal 后端。这表示两个连接。务必将 **ROUTER_MAX_CONNECTIONS** 设置为两倍，超过您要创建的连接数。

3.2.5. HAProxy Strict SNI

HAProxy **strict-sni** 可以通过路由器部署配置中的 **ROUTER_STRICT_SNI** 环境变量来控制。也可以使用 **--strict-sni** 命令行选项创建路由器时设置它。

```
$ oc adm router --strict-sni
```

3.2.6. TLS 密码套件

在创建路由器时使用 **--ciphers** 选项设置路由器密码套件 ：

```
$ oc adm router --ciphers=modern  ....
```

值包括：**modern**, **intermediate**, 或 **old**，**intermediate** 为默认值。或者，可以提供一组":"分隔的密码。密码必须来自以下集合：

```
$ openssl ciphers
```

或者，将 **ROUTER_CIPHERS** 环境变量用于现有路由器。

3.2.7. 双向 TLS 身份验证

可以使用 mutual TLS 身份验证限制对路由器和后端服务的客户端访问。路由器将拒绝来自不在 **通过*身份验证*** 的集合中的客户端的请求。双向 TLS 身份验证是在客户端证书上实施的，可以根据签发证书的证书颁发机构(CA)控制，证书撤销列表和/或任何证书主题过滤器。在创建路由器时，使用 mutual tls 配置选项 **--mutual-tls-auth**, **--mutual-tls-auth-ca**, **--mutual-tls-auth-crl** 和 **--mutual-tls-auth-filter**：

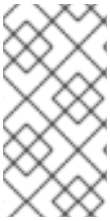
```
$ oc adm router --mutual-tls-auth=required  \
            --mutual-tls-auth-ca=/local/path/to/cacerts.pem   ....
```

--mutual-tls-auth 的值是 **required**, **optional**, 或 **none**，**none** 是默认值。**--mutual-tls-auth-ca** 值指定包含一个或多个 CA 证书的文件。路由器使用这些 CA 证书来验证客户端的证书。

可以使用 **--mutual-tls-auth-crl** 指定证书撤销列表，以处理证书（由有效认证机构签发）被撤销的情况。

```
$ oc adm router --mutual-tls-auth=required  \
    --mutual-tls-auth-ca=/local/path/to/cacerts.pem  \
    --mutual-tls-auth-filter='^/CN=my.org/ST=CA/C=US/O=Security/OU=OSE$'  \
    ....
```

--mutual-tls-auth-filter 值可用于根据证书主题进行精细的访问控制。该值是一个正则表达式，用于匹配证书的主题。



注意

上面的 mutual TLS 身份验证过滤器显示了一个限制性正则表达式(regex)，包括在 **^** 和 **$** 之间，这会**完全准确**匹配证书的主题。如果您决定使用限制较小的正则表达式，请注意这可能与您认为有效的任何 CA 颁发的证书匹配。建议您使用 **--mutual-tls-auth-ca** 选项，以便您更精细地控制发布的证书。

使用 **--mutual-tls-auth=required** 可确保您***只允许***经过身份验证的客户端访问后端资源。这意味着始终***需要*** 客户端来提供身份验证信息（提供一个客户端证书）。要使 mutual TLS 身份验证可选，请使用 **--mutual-tls-auth=optional** （或使用 **none** 禁用它，这是默认设置）。请注意，**optional** 意味着您 ***不需要*** 客户端提供任何身份验证信息，如果客户端提供任何身份验证信息，这会在 **X-SSL*** HTTP 标头中传递给后端。

```
$ oc adm router --mutual-tls-auth=optional  \
    --mutual-tls-auth-ca=/local/path/to/cacerts.pem  \
    ....
```

启用 mutual TLS 身份验证支持时（使用 **--mutual-tls-auth** 标志的 **required** 值或 **optional** 值），客户端身份验证信息将以 **X-SSL*** HTTP 标头的形式传递给后端。

X-SSL* HTTP 标头 **X-SSL-Client-DN** 示例：证书主题的完整可分辨名称(DN)。**X-SSL-Client-NotBefore**：客户端证书启动日期为 YYMMDDhhmmss[Z] 格式。**X-SSL-Client-NotAfter** ：客户端证书结束日期，采用 YYMMDDhhmmss[Z] 格式。**X-SSL-Client-SHA1** ：客户端证书的 SHA-1 指纹。**X-SSL-Client-DER**：提供客户端证书的完整访问权限。包含以 base-64 格式编码的 DER 格式的客户端证书。

3.2.8. 高可用路由器

您可以使用 IP 故障切换在 OpenShift Container Platform 集群中设置高可用性路由器。此设置在不同节点上有多个副本，因此如果当前副本失败，故障切换软件可以切换到另一个副本。

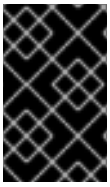
3.2.9. 自定义路由器服务端口

您可以通过设置环境变量 **ROUTER_SERVICE_HTTP_PORT** 和 **ROUTER_SERVICE_HTTPS_PORT** 来自定义模板路由器绑定的服务端口。这可以通过创建模板路由器，然后编辑其部署配置来完成。

以下示例创建带有 **0** 个副本的路由器部署，并自定义路由器服务 HTTP 和 HTTPS 端口，然后相应地将其扩展（到 **1** 个副本）。

```
$ oc adm router --replicas=0 --ports='10080:10080,10443:10443' 1
$ oc set env dc/router ROUTER_SERVICE_HTTP_PORT=10080 \
                ROUTER_SERVICE_HTTPS_PORT=10443
$ oc scale dc/router --replicas=1
```

1 确保为使用容器网络模式 **--host-network=false** 的路由器正确设置公开端口。



重要

如果您确实自定义模板路由器服务端口，您还需要确保运行路由器 Pod 的节点在防火墙中打开这些自定义端口（通过 Ansible 或 **iptables**，或通过 **firewall-cmd**使用的任何其他自定义方式）。

以下是使用 **iptables** 打开自定义路由器服务端口的示例：

```
$ iptables -A OS_FIREWALL_ALLOW -p tcp --dport 10080 -j ACCEPT
$ iptables -A OS_FIREWALL_ALLOW -p tcp --dport 10443 -j ACCEPT
```

3.2.10. 使用多个路由器

管理员可以创建具有相同定义的多个路由器，为同一组路由提供服务。每个路由器将位于不同的节点，并且具有不同的 IP 地址。网络管理员需要获取到每个节点所需的流量。

可以分组多个路由器来在集群中分发路由负载，并将租户单独的租户分发到不同的路由器或 分片。组中的每个路由器或分片都根据路由器中的选择器接受路由。管理员可以使用 **ROUTE_LABELS** 在整个集群中创建分片。用户可以使用 **NAMESPACE_LABELS** 在命名空间（项目）上创建分片。

3.2.11. 在部署配置中添加 Node Selector

将特定路由器部署到特定的节点上需要两个步骤：

1. 为所需节点添加 标签 ：

```
$ oc label node 10.254.254.28 "router=first"
```

2. 将节点选择器添加到路由器部署配置中：

```
$ oc edit dc <deploymentConfigName>
```

使用与标签对应的键和值添加 **template.spec.nodeSelector** 字段：

```
...
  template:
    metadata:
      creationTimestamp: null
      labels:
```

```
      router: router1
  spec:
    nodeSelector:          1
      router: "first"
...
```

1 键和值分别是 **router** 和 **first**，对应于 **router=first** 标签。

3.2.12. 使用路由器共享

*路由器分片*使用 **NAMESPACE_LABELS** 和 **ROUTE_LABELS** 来过滤路由器命名空间和路由。这可让您通过多个路由器部署分发路由的子集。通过使用非覆盖的子集，您可以有效地对一组路由进行分区。或者，您可以定义由重叠的路由子集组成的分片。

默认情况下，路由器从所有项目（命名空间）中选择所有路由。分片涉及将标签添加到路由或命名空间，以及向路由器添加标签选择器。每个路由器分片都由一组特定标签选择器或属于由特定标签选择器选择的命名空间来选择的路由组成。

注意

路由器服务帐户必须设置 [**集群读取器**] 权限，以允许访问其他命名空间中的标签。

路由器划分和 DNS

由于需要外部 DNS 服务器将请求路由到所需的分片，因此管理员负责为项目中的每个路由器创建单独的 DNS 条目。路由器不会将未知路由转发到另一个路由器。

考虑以下示例：

- 路由器 A 驻留在主机 192.168.0.5 上，并且具有 ***.foo.com** 的路由。
- 路由器 B 驻留在主机 192.168.1.9 上，并且具有 ***.example.com** 的路由。

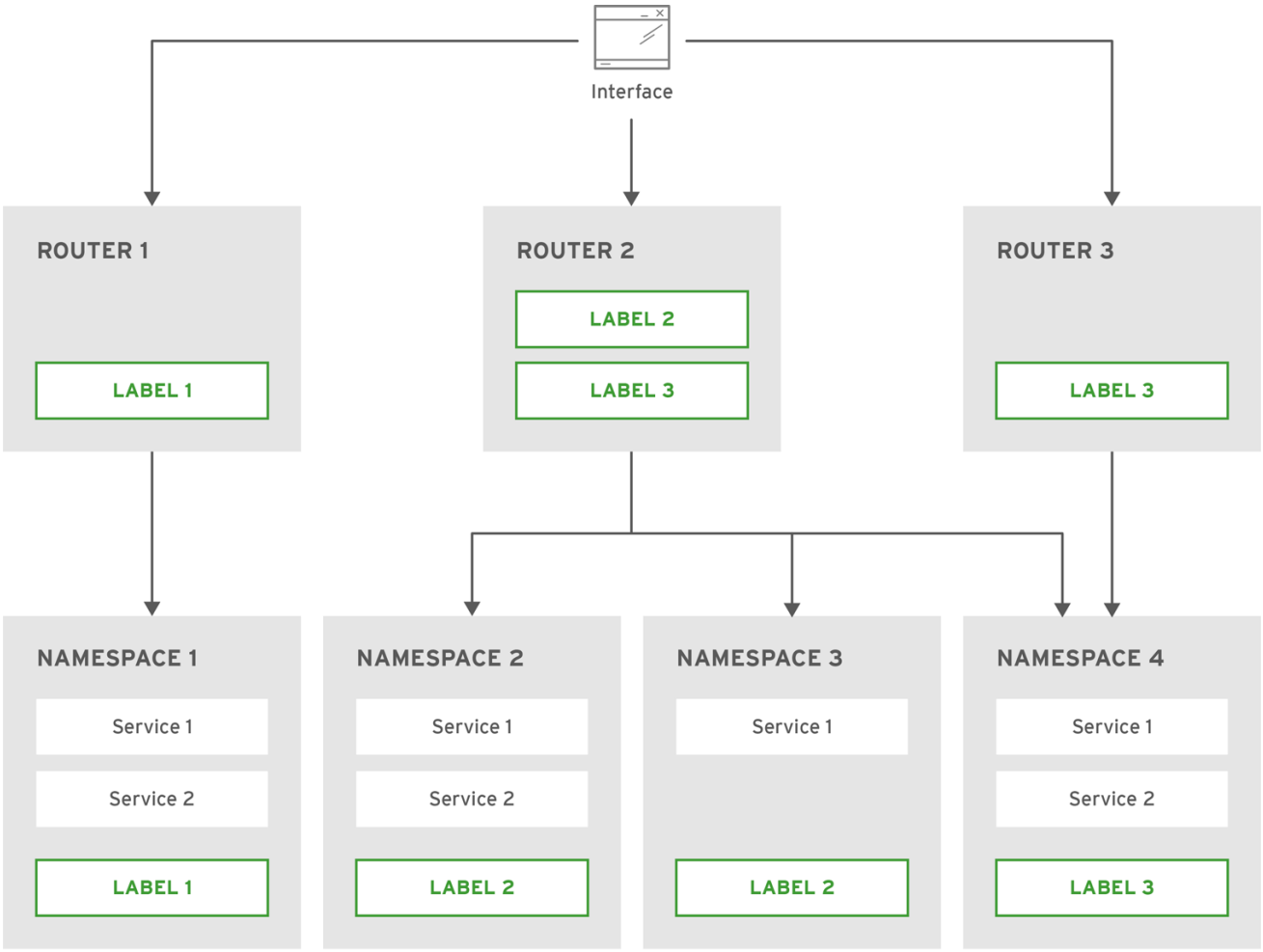
单独的 DNS 条目必须将 *.foo.com 解析为托管 Router A 和 *.example.com 的节点：

- ***.foo.com A IN 192.168.0.5**
- ***.example.com A IN 192.168.1.9**

路由器分片示例

本节论述了使用命名空间和路由标签进行路由器分片。

图 3.1. 路由器划分基于命名空间标签



1. 使用命名空间标签选择器配置路由器：

```
$ oc set env dc/router NAMESPACE_LABELS="router=r1"
```

2. 由于该路由器在命名空间上有一个选择器，因此路由器将仅处理匹配的命名空间的路由。要使此选择器与命名空间匹配，请相应地标记命名空间：

```
$ oc label namespace default "router=r1"
```

3. 现在，如果您在 default 命名空间中创建路由，该路由位于默认路由器中：

```
$ oc create -f route1.yaml
```

4. 创建新项目（命名空间）并创建路由 **route2** ：

```
$ oc new-project p1
$ oc create -f route2.yaml
```

注意路由在路由器中不可用。

5. 标签命名空间 **p1** 带有 **router=r1**

```
$ oc label namespace p1 "router=r1"
```

添加此标签使路由在路由器中可用。

示例

一个路由器部署 **finops-router** 使用标签选择器 **NAMESPACE_LABELS="name in (finance, ops)"** 配置，一个路由器部署 **dev-router** 使用标签选择器 **NAMESPACE_LABELS="name=dev"** 配置。如果所有路由都位于标有 **name=finance**、**name=ops** 和 **name=dev 的命名空间** 中，则此配置会在两个路由器部署之间有效分发您的路由。

在上面的场景中，分片成为分区的一种特殊情况，没有重叠的子集。路由在路由器分片之间划分。

路由选择标准适用于路由的分布方式。有可能在路由器部署之间具有重叠的路由子集。

示例

除了上例中的 **finops-router** 和 **dev-router** 外，您还有 **devops-router**，它使用标签选择器 **NAMESPACE_LABELS="name in (dev, ops)"** 配置。
标签为 **name=dev** 或 **name=ops** 的命名空间中的路由现在由两个不同的路由器部署服务。在这种情况下，您定义了重叠的路由子集，如 基于命名空间标签的路由器交换过程中所示。

另外，这可让您创建更复杂的路由规则，允许将优先级更高的流量转移到专用的 **finops-router**，同时将较低优先级的流量发送到 **devops-router**。

基于路由标签的路由器划分

NAMESPACE_LABELS 允许过滤项目来服务并从这些项目中选择所有路由，但您可能希望根据与路由本身相关的其他标准对路由进行分区。**ROUTE_LABELS** 选择器允许您对路由本身进行分片。

示例

使用标签选择器 **ROUTE_LABELS="mydeployment=prod"** 配置路由器部署 **prod-router**，路由器部署 **devtest-router** 则配置了标签选择器 **ROUTE_LABELS="mydeployment in(dev, test)"**。
此配置根据路由的标签（无论命名空间如何）在两个路由器部署之间路由。

这个示例假设您有要服务的所有路由，并带有标签 **"mydeployment=<tag>"**。

3.2.12.1. 创建路由器分片

本节描述了路由器分片的高级示例。假设有 26 个路由，名为 **a** 到 **z**，且具有不同的标签：

路由上可能的标签

```
sla=high    geo=east    hw=modest    dept=finance
sla=medium  geo=west    hw=strong    dept=dev
sla=low                              dept=ops
```

这些标签表达了包括服务级别协议、地理位置、硬件要求和部门在内的概念。路由最多可在每列中有一个标签。有些路由可能具有其他标签，或者根本没有标签。

名称	SLA	地域	HW	部门	其他标签
a	**high**	东部	中等	**finance**	**type=static**
b		**西部**	强		**type=dynamic**

名称	SLA	地域	HW	部门	其他标签
c、d、e	低		中等		type=static
g 到 k	中		强	dev	
l 到 s	high		中等	ops	
t – z		西部			type=dynamic

以下是一个便捷脚本 *mkshard*，它演示了如何一起使用 **oc adm router**、**oc set env** 和 **oc scale** 来创建路由器分片。

```
#!/bin/bash
# Usage: mkshard ID SELECTION-EXPRESSION
id=$1
sel="$2"
router=router-shard-$id          (1)
oc adm router $router --replicas=0   (2)
dc=dc/router-shard-$id           (3)
oc set env   $dc ROUTE_LABELS="$sel"   (4)
oc scale $dc --replicas=3        (5)
```

1. 创建的路由器具有名称 **router-shard-<id>**。
2. 暂时不要进行扩展。
3. 路由器的部署配置。
4. 使用 **oc set env** 设置选择表达式。选择表达式是 **ROUTE_LABELS** 环境变量的值。
5. 向上扩展。

运行 *mkshard* 几次创建多个路由器：

路由器	选择表达式	Routes
router-shard-1	sla=high	a, l – s
router-shard-2	geo=west	b, t – z
router-shard-3	dept=dev	g 到 k

3.2.12.2. 修改路由器分片

由于路由器分片是基于标签的结构，因此您可以修改标签（通过 **oc label**）或选择表达式（通过 **oc set env**）。

本节扩展了创建路由器分片一节中启动的示例，演示了如何更改选择表达式。

以下是可修改现有路由器以使用新选择表达式的便捷脚本 *modshard* ：

```
#!/bin/bash
# Usage: modshard ID SELECTION-EXPRESSION...
id=$1
shift
router=router-shard-$id      1
dc=dc/$router                2
oc scale $dc --replicas=0    3
oc set env   $dc "$@"        4
oc scale $dc --replicas=3    5
```

1. 修改后的路由器具有名称 **router-shard-<id>**。
2. 进行修改的部署配置。
3. 缩减规模。
4. 使用 **oc set env** 设置新的选择表达式。与创建路由器分片部分中的 **mkshard** 不同，在 **modshard** 中指定为非**ID** 参数的选择表达式必须包含环境变量名称及其值。
5. 纵向扩展。

注意

在 **modshard** 中，如果 **router-shard-<id>**部署策略 是 **Rolling**，则不需要 **oc scale** 命令。

例如，将 **router-shard-3** 的部门扩展到包含 **ops** 和 **dev** ：

```
$ modshard 3 ROUTE_LABELS='dept in (dev, ops)'
```

结果是，**router-shard-3** 现在选择路由 **g – s** (**g – k** 和 **l – s** 的组合)。

本例考虑了本例中只有三个部门，并指定了一个部门离开分片，从而得到与上例相同的结果：

```
$ modshard 3 ROUTE_LABELS='dept != finance'
```

这个示例指定了三个用逗号分开的功能，并只选择路由 **b** ：

```
$ modshard 3 ROUTE_LABELS='hw=strong,type=dynamic,geo=west'
```

与涉及路由标签的 **ROUTE_LABELS** 类似，您可以使用 **NAMESPACE_LABELS** 环境变量根据路由命名空间的标签选择路由。本例修改 **router-shard-3** 以提供命名空间具有标签 **frequency=weekly** 的路由：

```
$ modshard 3 NAMESPACE_LABELS='frequency=weekly'
```

最后一个示例组合了 **ROUTE_LABELS** 和 **NAMESPACE_LABELS**，以选择带有标签 **sla=low** 的路由，其命名空间具有标签 **frequency=weekly** ：

```
$ modshard 3 \
    NAMESPACE_LABELS='frequency=weekly' \
    ROUTE_LABELS='sla=low'
```

3.2.13. 查找路由器的主机名

在公开服务时，用户可以使用来自外部用户用于访问应用的 DNS 名称相同的路由。外部网络的网络管理员必须确保主机名解析为已接受该路由的路由器的名称。用户可以使用指向此主机名的 CNAME 设置 DNS。但是，用户可能不知道路由器的主机名。当不知道时，集群管理员可以提供它。

在创建路由器时，集群管理员可以使用 **--router-canonical-hostname** 选项和路由器的规范主机名。例如：

```
# oc adm router myrouter --router-canonical-hostname="rtr.example.com"
```

这会在包含路由器主机名的路由器部署配置中创建 **ROUTER_CANONICAL_HOSTNAME** 环境变量。

对于已存在的路由器，集群管理员可以编辑路由器的部署配置，并添加 **ROUTER_CANONICAL_HOSTNAME** 环境变量：

```
spec:
  template:
    spec:
      containers:
        - env:
          - name: ROUTER_CANONICAL_HOSTNAME
            value: rtr.example.com
```

ROUTER_CANONICAL_HOSTNAME 值显示在接受该路由的所有路由器的路由状态中。每次重新载入路由器时，路由状态都会刷新。

当用户创建路由时，所有活跃的路由器都会评估路由，如果满足条件，则接受路由。当定义 **ROUTER_CANONICAL_HOSTNAME** 环境变量的路由器接受该路由时，路由器会将该值放在路由状态的 **routerCanonicalHostname** 字段中。用户可以检查路由状态，以确定路由器是否允许该路由，从列表中选择路由器，再查找要传给网络管理员的路由器的主机名。

```
status:
  ingress:
    conditions:
      lastTransitionTime: 2016-12-07T15:20:57Z
      status: "True"
      type: Admitted
      host: hello.in.mycloud.com
      routerCanonicalHostname: rtr.example.com
      routerName: myrouter
      wildcardPolicy: None
```

在可用时 **oc describe** 会包括主机名：

```
$ oc describe route/hello-route3
...
Requested Host: hello.in.mycloud.com exposed on router myroute (host rtr.example.com) 12 minutes
ago
```

利用上述信息，用户可以要求 DNS 管理员将路由的主机 **hello.in.mycloud.com** 中的 CNAME 设置为路由器的规范主机名 **rtr.example.com**。这将导致指向 **hello.in.mycloud.com** 的任何流量到达用户的应用。

3.2.14. 自定义默认路由子域

您可以通过修改 master 配置文件（默认为 */etc/origin/master/master-config.yaml* 文件）来自定义用作环境默认路由子域的后缀。没有指定主机名的路由会有一个使用该默认路由子域生成的路由。

以下示例演示了如何将配置的后缀设置为 **v3.openshift.test** ：

```
routingConfig:
  subdomain: v3.openshift.test
```

注意

如果 master 正在运行，这个更改需要重启。

运行上述配置的 OpenShift Container Platform master 时，对于名为 **no-route-hostname** 的示例路由（没有主机名添加到命名空间 **mynamespace**）生成的主机名将是：

```
no-route-hostname-mynamespace.v3.openshift.test
```

3.2.15. 将路由主机名强制到自定义路由子域

如果管理员希望限制到特定路由子域的所有路由，他们可以将 **--force-subdomain** 选项传递给 **oc adm router** 命令。这会强制路由器覆盖路由中指定的任何主机名，并根据提供给 **--force-subdomain** 选项的模板生成一个主机名。

以下示例运行路由器，它使用自定义子域模板 **${name}-${namespace}.apps.example.com** 来覆盖路由主机名。

```
$ oc adm router --force-subdomain='${name}-${namespace}.apps.example.com'
```

3.2.16. 使用通配符证书

不包含证书的 TLS 路由改为使用路由器的默认证书。在大多数情况下，此证书应由可信证书颁发机构提供，但为了方便起见，您可以使用 OpenShift Container Platform CA 创建证书。例如：

```
$ CA=/etc/origin/master
$ oc adm ca create-server-cert --signer-cert=$CA/ca.crt \
      --signer-key=$CA/ca.key --signer-serial=$CA/ca.serial.txt \
      --hostnames='*.cloudapps.example.com' \
      --cert=cloudapps.crt --key=cloudapps.key
```

注意

oc adm ca create-server-cert 命令会生成一个有效期为两年的证书。这可以通过 **--expire-days** 选项进行修改，但出于安全原因，建议不要超过这个值。

仅从 Ansible 主机清单文件中列出的第一个 master 运行 **oc adm** 命令，默认为 */etc/ansible/hosts*。

路由器预期证书和密钥在单个文件中采用 PEM 格式：

```
$ cat cloudapps.crt cloudapps.key $CA/ca.crt > cloudapps.router.pem
```

在这里您可以使用 **--default-cert** 标志：

```
$ oc adm router --default-cert=cloudapps.router.pem --service-account=router
```

注意

浏览器只考虑一个级别深度的子域有效通配符。因此在此示例中，证书对 *a.cloudapps.example.com* 有效，但不适用于 *a.b.cloudapps.example.com*。

3.2.17. 手动重新部署证书

手动重新部署路由器证书：

1. 检查包含默认路由器证书的 secret 是否已添加到路由器中：

```
$ oc set volume dc/router

deploymentconfigs/router
  secret/router-certs as server-certificate
    mounted at /etc/pki/tls/private
```

   如果添加了证书，请跳过以下步骤并覆盖 secret。

2. 确保为以下变量 **DEFAULT_CERTIFICATE_DIR** 设置了一个默认证书目录：

```
$ oc set env dc/router --list

DEFAULT_CERTIFICATE_DIR=/etc/pki/tls/private
```

   如果没有，请使用以下命令创建该目录：

```
$ oc set env dc/router DEFAULT_CERTIFICATE_DIR=/etc/pki/tls/private
```

3. 将证书导出到 PEM 格式：

```
$ cat custom-router.key custom-router.crt custom-ca.crt > custom-router.crt
```

4. 覆盖或创建路由器证书 secret：
   如果证书 secret 添加到路由器，请覆盖该 secret。如果没有，请创建一个新 secret。

   要覆盖 secret，请运行以下命令：

```
$ oc create secret generic router-certs --from-file=tls.crt=custom-router.crt --from-
file=tls.key=custom-router.key --type=kubernetes.io/tls -o json --dry-run | oc replace -f -
```

   要创建新 secret，请运行以下命令：

```
$ oc create secret generic router-certs --from-file=tls.crt=custom-router.crt --from-
```

```
file=tls.key=custom-router.key --type=kubernetes.io/tls

$ oc set volume dc/router --add --mount-path=/etc/pki/tls/private --secret-name='router-certs'
--name router-certs
```

5. 部署路由器。

```
$ oc rollout latest dc/router
```

3.2.18. 使用安全路由

目前，不支持密码保护的密钥文件。启动后，HAProxy 会提示输入密码，且无法自动执行此过程。要从密钥文件中删除密码短语，您可以运行以下命令：

```
# openssl rsa -in <passwordProtectedKey.key> -out <new.key>
```

以下是如何在流量代理到目的地之前使用发生在路由器上发生 TLS 终止的安全边缘终止路由的示例：安全边缘终止路由指定 TLS 证书和密钥信息。TLS 证书由路由器前端提供。

首先，启动一个路由器实例：

```
# oc adm router --replicas=1 --service-account=router
```

接下来，为我们边缘安全路由创建私钥、csr 和证书。有关如何执行此操作的说明将特定于您的证书颁发机构和提供商。有关名为 **www.example.test** 的域的简单自签名证书，请参考以下示例：

```
# sudo openssl genrsa -out example-test.key 2048
#
# sudo openssl req -new -key example-test.key -out example-test.csr  \
  -subj "/C=US/ST=CA/L=Mountain View/O=OS3/OU=Eng/CN=www.example.test"
#
# sudo openssl x509 -req -days 366 -in example-test.csr  \
      -signkey example-test.key -out example-test.crt
```

使用上述证书和密钥生成路由。

```
$ oc create route edge --service=my-service \
    --hostname=www.example.test \
    --key=example-test.key --cert=example-test.crt
route "my-service" created
```

看一下其定义。

```
$ oc get route/my-service -o yaml
apiVersion: v1
kind: Route
metadata:
  name:  my-service
spec:
  host: www.example.test
  to:
    kind: Service
    name: my-service
```

```
  tls:
    termination: edge
    key: |
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

确保 **www.example.test** 的 DNS 条目指向您的路由器实例，并且到您的域的路由应可用。以下示例使用 curl 和本地解析器模拟 DNS 查找：

```
# routerip="4.1.1.1"  #  replace with IP address of one of your router instances.
# curl -k --resolve www.example.test:443:$routerip https://www.example.test/
```

3.2.19. 使用通配符路由（用于子域）

HAProxy 路由器支持通配符路由，这通过将 **ROUTER_ALLOW_WILDCARD_ROUTES** 环境变量设置为 **true** 来启用。具有 **Subdomain** 通配符策略且通过路由器准入检查的任何路由都将由 HAProxy 路由器提供服务。然后，HAProxy 路由器根据路由的通配符策略公开相关的服务（用于路由）。

重要

要更改路由的通配符策略，您必须删除路由并使用更新的通配符策略重新创建路由。仅编辑路由的 ***.yaml*** 文件中的路由通配符策略无法正常工作。

```
$ oc adm router --replicas=0 ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
$ oc scale dc/router --replicas=1
```

了解如何为通配符路由配置 Web 控制台。

使用安全通配符边缘终止路由

这个示例反映了在流量代理到目的地之前在路由器上发生的 TLS 终止。发送到子域 **example.org(*.example.org**)中的任何主机的流量将代理到公开的服务。

安全边缘终止路由指定 TLS 证书和密钥信息。TLS 证书由与子域(***.example.org**)匹配的所有主机的路由器前端提供。

1. 启动路由器实例：

```
$ oc adm router --replicas=0 --service-account=router
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
$ oc scale dc/router --replicas=1
```

2. 为边缘安全路由创建私钥、证书签名请求(CSR)和证书。
有关如何执行此操作的说明特定于您的证书颁发机构和供应商。有关名为 ***.example.test** 的域的简单自签名证书，请查看以下示例：

```
# sudo openssl genrsa -out example-test.key 2048
#
```

```
# sudo openssl req -new -key example-test.key -out example-test.csr  \
  -subj "/C=US/ST=CA/L=Mountain View/O=OS3/OU=Eng/CN=*.example.test"
#
# sudo openssl x509 -req -days 366 -in example-test.csr  \
     -signkey example-test.key -out example-test.crt
```

3. 使用上述证书和密钥生成通配符路由：

```
$ cat > route.yaml  <<REOF
apiVersion: v1
kind: Route
metadata:
  name:  my-service
spec:
  host: www.example.test
  wildcardPolicy: Subdomain
  to:
    kind: Service
    name: my-service
  tls:
    termination: edge
    key: "$(perl -pe 's/\n/\\n/' example-test.key)"
    certificate: "$(perl -pe 's/\n/\\n/' example-test.cert)"
REOF
$ oc create -f route.yaml
```

确保 ***.example.test** 的 DNS 条目指向您的路由器实例，并且到域的路由可用。

这个示例使用带有本地解析器的 **curl** 来模拟 DNS 查找：

```
# routerip="4.1.1.1"  #  replace with IP address of one of your router instances.
# curl -k --resolve www.example.test:443:$routerip https://www.example.test/
# curl -k --resolve abc.example.test:443:$routerip https://abc.example.test/
# curl -k --resolve anyname.example.test:443:$routerip https://anyname.example.test/
```

对于允许通配符路由的路由器（**ROUTER_ALLOW_WILDCARD_ROUTES** 设置为 **true**），存在一些注意事项，即与通配符路由关联的子域的所有权。

在通配符路由之前，所有权基于针对任何其他声明赢得路由最旧的命名空间的主机名提出的声明。例如，如果路由 **r1** 比路由 **r2** 老，对于 主机名 **one.example.test**，命名空间 **ns1** 中的路由 **r1**（有一个 **one.example.test** 的声明）会优于命令空间 **ns2** 中具有相同声明的路由 **r2**。

另外，其他命名空间中的路由也被允许声明非覆盖的主机名。例如，命名空间 **ns1** 中的路由 **rone** 可以声明 **www.example.test**，命名空间 **d2** 中的另一个路由 **rtwo** 可以声明 **c3po.example.test**。

如果*没有*任何通配符路由声明同一子域（上例中的**example.test** ），则仍会出现这种情况。

但是，通配符路由需要声明子域中的所有主机名（以 ***.example.test**格式的主机名）。通配符路由的声明会根据该子域的最旧路由(**example.test**)是否与通配符路由在同一命名空间中被允许或拒绝。最旧的路由可以是常规路由，也可以是通配符路由。

例如，如果已有一个路由 **eldest** 存在于 **s1** 命名空间中，声明一个名为 **owner.example.test** 的主机，如果稍后某个时间点上添加了一个新的通配符用来**通配**子域 (**example.test**) 中的路由，则通配路由的声明*仅在*与拥有的路由处于相同命名空间 (**ns1**)时才被允许。

以下示例演示了通配符路由的声明将成功或失败的各种情景。

在以下示例中，只要通配符路由未声明子域，允许通配符路由的路由器将允许对子域 **example.test** 中的主机进行非覆盖声明。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
$ oc expose service myservice --hostname=aname.example.test
$ oc expose service myservice --hostname=bname.example.test

$ oc project ns2
$ oc expose service anotherservice --hostname=second.example.test
$ oc expose service anotherservice --hostname=cname.example.test

$ oc project otherns
$ oc expose service thirdservice --hostname=emmy.example.test
$ oc expose service thirdservice --hostname=webby.example.test
```

在以下示例中，允许通配符路由的路由器不允许 **owner.example.test** 或 **aname.example.test** 的声明成功，因为拥有的命名空间是 **ns1**。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
$ oc expose service myservice --hostname=aname.example.test

$ oc project ns2
$ oc expose service secondservice --hostname=bname.example.test
$ oc expose service secondservice --hostname=cname.example.test

$ # Router will not allow this claim with a different path name `/p1` as
$ # namespace `ns1` has an older route claiming host `aname.example.test`.
$ oc expose service secondservice --hostname=aname.example.test --path="/p1"

$ # Router will not allow this claim as namespace `ns1` has an older route
$ # claiming host name `owner.example.test`.
$ oc expose service secondservice --hostname=owner.example.test

$ oc project otherns

$ # Router will not allow this claim as namespace `ns1` has an older route
$ # claiming host name `aname.example.test`.
$ oc expose service thirdservice --hostname=aname.example.test
```

在以下示例中，允许通配符路由的路由器将允许 **'*.example.test** 声明成功，因为拥有的命名空间是 **ns1**，通配符路由属于同一命名空间。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
```

```
$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  router will allow this claim.
```

在以下示例中，允许通配符路由的路由器不允许 **'*.example.test** 声明成功，因为拥有的命名空间是 **ns1**，并且通配符路由属于另一个命名空间 **cyclone**。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ # Switch to a different namespace/project.
$ oc project cyclone

$ # Reusing the route.yaml from a prior example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  router will deny (_NOT_ allow) this claim.
```

同样，当具有通配符路由的命名空间声明子域后，只有该命名空间中的路由才能声明同一子域中的任何主机。

在以下示例中，当命名空间 **ns1** 中带有通配符路由声明子域 **example.test** 的路由之后，只有命名空间 **ns1** 中的路由可以声明同一子域中的任何主机。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ oc project otherns

$ # namespace `otherns` is allowed to claim for other.example.test
$ oc expose service otherservice --hostname=other.example.test

$ oc project ns1

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  Router will allow this claim.

$ #  In addition, route in namespace otherns will lose its claim to host
$ #  `other.example.test` due to the wildcard route claiming the subdomain.
```

```
$ # namespace `ns1` is allowed to claim for deux.example.test
$ oc expose service mysecondservice --hostname=deux.example.test

$ # namespace `ns1` is allowed to claim for deux.example.test with path /p1
$ oc expose service mythirdservice --hostname=deux.example.test --path="/p1"

$ oc project otherns

$ # namespace `otherns` is not allowed to claim for deux.example.test
$ # with a different path '/otherpath'
$ oc expose service otherservice --hostname=deux.example.test --path="/otherpath"

$ # namespace `otherns` is not allowed to claim for owner.example.test
$ oc expose service yetanotherservice --hostname=owner.example.test

$ # namespace `otherns` is not allowed to claim for unclaimed.example.test
$ oc expose service yetanotherservice --hostname=unclaimed.example.test
```

在以下示例中，显示了不同的场景，其中删除所有者路由并在命名空间内和跨命名空间传递所有权。虽然命名空间 **ns1** 中存在声明主机 **eldest.example.test** 的路由，但该命名空间中的通配符路由可以声明子域 **example.test**。当删除主机 **eldest.example.test** 的路由时，下一个最旧的路由 **senior.example.test** 将成为最旧的路由，不会影响任何其他路由。删除主机 **senior.example.test** 的路由后，下一个最旧的路由 **junior.example.test** 将成为最旧路由并阻止通配符路由。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=eldest.example.test
$ oc expose service seniorservice --hostname=senior.example.test

$ oc project otherns

$ # namespace `otherns` is allowed to claim for other.example.test
$ oc expose service juniorservice --hostname=junior.example.test

$ oc project ns1

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  Router will allow this claim.

$ #  In addition, route in namespace otherns will lose its claim to host
$ #  `junior.example.test` due to the wildcard route claiming the subdomain.

$ # namespace `ns1` is allowed to claim for dos.example.test
$ oc expose service mysecondservice --hostname=dos.example.test

$ # Delete route for host `eldest.example.test`, the next oldest route is
$ # the one claiming `senior.example.test`, so route claims are unaffacted.
$ oc delete route myservice
```

```
$ # Delete route for host `senior.example.test`, the next oldest route is
$ # the one claiming `junior.example.test` in another namespace, so claims
$ # for a wildcard route would be affected. The route for the host
$ # `dos.example.test` would be unaffected as there are no other wildcard
$ # claimants blocking it.
$ oc delete route seniorservice
```

3.2.20. 使用 Container Network Stack

OpenShift Container Platform 路由器在容器内运行，默认的行为是使用主机的网络堆栈（即，路由器容器运行的节点）。这个默认行为有利于性能，因为来自远程客户端的网络流量不需要通过用户空间获取多个跃点来访问目标服务和容器。

另外，这个默认行为可让路由器获取远程连接的实际源 IP 地址，而不是获取节点的 IP 地址。这可用于定义基于原始 IP、支持粘性会话和监控流量以及其他用途的入口规则。

此主机网络行为由 **--host-network** 路由器命令行选项控制，默认行为等同于使用 **--host-network=true**。如果要使用容器网络堆栈运行路由器，请在创建路由器时使用 **--host-network=false** 选项。例如：

```
$ oc adm router --service-account=router --host-network=false
```

在内部，这意味着路由器容器必须发布 80 和 443 端口，以便外部网络与路由器通信。

注意

使用容器网络堆栈运行意味着路由器将连接的源 IP 地址视为节点的 NATed IP 地址，而不是实际的远程 IP 地址。

注意

在使用多租户网络隔离的 OpenShift Container Platform 集群中，带有 **--host-network=false** 选项的非默认命名空间中的路由器将加载集群中的所有路由，但命名空间之间的路由会因为网络隔离而无法访问。使用 **--host-network=true** 选项时，路由会绕过容器网络，并且可以访问集群中的任何 pod。在这种情况下，如果需要隔离，则不要在命名空间间添加路由。

3.2.21. 使用动态配置管理器

您可以配置 HAProxy 路由器来支持动态配置管理器。

动态配置管理器带来某些类型的在线路由，无需 HAProxy 重新加载停机时间。它处理任何路由和端点生命周期事件，如路由和端点附加 |**deletion**|**update**。

通过将 **ROUTER_HAPROXY_CONFIG_MANAGER** 环境变量设置为 **true** 来启用动态配置管理器：

```
$ oc set env dc/<router_name> ROUTER_HAPROXY_CONFIG_MANAGER='true'
```

如果动态配置管理器无法动态配置 HAProxy，它会重写配置并重新加载 HAProxy 进程。例如，如果新路由包含自定义注解，如自定义超时，或者路由需要自定义 TLS 配置。

动态配置内部使用 HAProxy 套接字和配置 API，以及预分配的路由和后端服务器池。预分配的路由池使用路由蓝图创建。默认蓝图集支持不安全的路由、边缘安全路由，无需任何自定义 TLS 配置和直通路由。

重要

重新加密 路由需要自定义 TLS 配置信息，因此需要额外的配置才能将其用于动态配置管理器。

通过设置 **ROUTER_BLUEPRINT_ROUTE_NAMESPACE** 以及可选的 **ROUTER_BLUEPRINT_ROUTE_LABELS** 环境变量来扩展动态配置管理器可以使用的蓝图。

蓝图路由命名空间中的所有路由或路由标签的路由作为与默认蓝图集类似的自定义蓝图处理。这包括使用自定义注解或路由的**重新加密**路由或自定义 TLS 配置的路由。

以下流程假定您已创建了三个路由对象：**reencrypt-blueprint**, **annotated-edge-blueprint**, 和 **annotated-unsecured-blueprint**。如需不同路由类型对象的示例，请参阅 Route Types。

流程

1. 创建一个新项目

```
$ oc new-project namespace_name
```

2. 创建新路由。此方法公开现有的服务：

```
$ oc create route edge edge_route_name --key=/path/to/key.pem \
    --cert=/path/to/cert.pem --service=<service> --port=8443
```

3. 标记路由：

```
$ oc label route edge_route_name type=route_label_1
```

4. 创建与路由对象定义不同的路由 ：所有都具有标签 **type=route_label_1**:

```
$ oc create -f reencrypt-blueprint.yaml
$ oc create -f annotated-edge-blueprint.yaml
$ oc create -f annotated-unsecured-blueprint.json
```

您还可以从路由中删除标签，这会阻止它用作蓝图路由。例如，防止 **annotated-unsecured-blueprint** 用作蓝图路由：

```
$ oc label route annotated-unsecured-blueprint type-
```

5. 创建用于蓝图池的新路由器：

```
$ oc adm router
```

6. 为新路由器设置环境变量：

```
$ oc set env dc/router ROUTER_HAPROXY_CONFIG_MANAGER=true     \
                       ROUTER_BLUEPRINT_ROUTE_NAMESPACE=namespace_name  \
                       ROUTER_BLUEPRINT_ROUTE_LABELS="type=route_label_1"
```

处理命名空间或带有 **type=route_label_1** 标签的项目 **namespace_name** 中的所有路由，并用作自定义蓝图。

请注意，您还可以通过管理该命名空间 **namespace_name** 中的路由来添加、更新或删除蓝图。动态配置管理器会监视命名空间 **namespace_name** 中路由的更改，类似于路由器监视**路由**和**服务**的方式。

7. 预分配的路由和后端服务器的池大小可以通过 **ROUTER_BLUEPRINT_ROUTE_POOL_SIZE** 控制，默认为 **10**， 而 **ROUTER_MAX_DYNAMIC_SERVERS** 默认为 **5** 环境变量。您还可以控制动态配置管理器所做的更改的频率，即当重新编写 HAProxy 配置并重新加载 HAProxy 进程时。默认值为 1 小时或 3600 秒，或者当动态配置管理器用尽池空间时。**COMMIT_INTERVAL** 环境变量控制此设置：

```
$ oc set env dc/router -c router ROUTER_BLUEPRINT_ROUTE_POOL_SIZE=20 \
    ROUTER_MAX_DYNAMIC_SERVERS=3 COMMIT_INTERVAL=6h
```

这个示例将每个蓝图路由的池大小增加到 **20**，将动态服务器的数量减少到 **3**，并将提交间隔增加到 **6** 小时。

3.2.22. 公开路由器指标

HAProxy 路由器指标 默认以 Prometheus 格式 公开或发布，供外部指标收集和聚合系统（如 Prometheus、statsd）使用。指标数据也可直接从 HAProxy 路由器 以自己的 HTML 格式获取，以便在浏览器或 CSV 下载中查看。这些指标包括 HAProxy 原生指标和一些控制器指标。

当您使用以下命令创建路由器时，OpenShift Container Platform 以 Prometheus 格式在 stats 端口（默认为 1936）上提供指标数据。

```
$ oc adm router --service-account=router
```

- 要提取 Prometheus 格式的原始统计信息，请运行以下命令：

```
curl <user>:<password>@<router_IP>:<STATS_PORT>
```

例如：

```
$ curl admin:sLzdR6SgDJ@10.254.254.35:1936/metrics
```

您可以获取访问路由器服务注解中的指标所需的信息：

```
$ oc edit service <router-name>

apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/port: "1936"
    prometheus.io/scrape: "true"
    prometheus.openshift.io/password: IImoDqON02
    prometheus.openshift.io/username: admin
```

prometheus.io/port 是 stats 端口，默认为 1936。您可能需要配置防火墙以允许访问。使用前面的用户名和密码来访问指标。路径是 */metrics*。

```
$ curl <user>:<password>@<router_IP>:<STATS_PORT>
for example:
$ curl admin:sLzdR6SgDJ@10.254.254.35:1936/metrics
```

```
...
# HELP haproxy_backend_connections_total Total number of connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-
route"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-
route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-
route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked and the current
threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_no_sni",service=""}
0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="default",pod="docker-registry-5-
nk5fz",route="docker-registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-vkjqx",route="hello-
route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...
```

- 在浏览器中获取指标：

  1. 从路由器部署配置文件中删除以下 环境变量 ：

     ```
     $ oc edit dc router

     - name: ROUTER_LISTEN_ADDR
       value: 0.0.0.0:1936
     - name: ROUTER_METRICS_TYPE
       value: haproxy
     ```

  2. 对路由器就绪度探测进行补丁，以使用与存活度探测相同的路径，它现在由 haproxy 路由器提供：

     ```
     $ oc patch dc router -p '"spec": {"template": {"spec": {"containers": [{"name":
     "router","readinessProbe": {"httpGet": {"path": "/healthz"}}}]}}}'
     ```

  3. 在浏览器中使用以下 URL 启动 stats 窗口，其中 **STATS_PORT** 值默认为 **1936**:

     ```
     http://admin:<Password>@<router_IP>:<STATS_PORT>
     ```

     您可以通过在 URL 中添加 **;csv** 来获得 CSV 格式的统计信息：

例如：

```
http://admin:<Password>@<router_IP>:1936;csv
```

获取路由器 IP、管理员名称和密码：

```
oc describe pod <router_pod>
```

- 禁止指标收集：

```
$ oc adm router --service-account=router --stats-port=0
```

3.2.23. 大型集群的 ARP 缓存调优

在具有大量路由的 OpenShift Container Platform 集群中（超过 **net.ipv4.neigh.default.gc_thresh3** 的值 **65536**），您必须增加运行路由器 pod 的集群中每个节点上的 sysctl 变量默认值，以允许 ARP 缓存中更多条目。

当出现问题时，内核信息类似如下：

```
[ 1738.811139] net_ratelimit: 1045 callbacks suppressed
[ 1743.823136] net_ratelimit: 293 callbacks suppressed
```

当出现这个问题时，**oc** 命令可能会因为以下错误开始失败：

```
Unable to connect to the server: dial tcp: lookup <hostname> on <ip>:<port>: write udp <ip>:<port>->
<ip>:<port>: write: invalid argument
```

要验证 IPv4 的 ARP 条目的实际数量，请运行以下内容：

```
# ip -4 neigh show nud all | wc -l
```

如果数字开始接近 **net.ipv4.neigh.default.gc_thresh3** 阈值，则增加值。运行以下命令来获取当前值：

```
# sysctl net.ipv4.neigh.default.gc_thresh1
net.ipv4.neigh.default.gc_thresh1 = 128
# sysctl net.ipv4.neigh.default.gc_thresh2
net.ipv4.neigh.default.gc_thresh2 = 512
# sysctl net.ipv4.neigh.default.gc_thresh3
net.ipv4.neigh.default.gc_thresh3 = 1024
```

以下 sysctl 将变量设置为 OpenShift Container Platform 当前默认值。

```
# sysctl net.ipv4.neigh.default.gc_thresh1=8192
# sysctl net.ipv4.neigh.default.gc_thresh2=32768
# sysctl net.ipv4.neigh.default.gc_thresh3=65536
```

要使这些设置永久生效，请查看本文档。

3.2.24. 保护 DDoS Attacks

向默认 HAProxy 路由器镜像添加 **超时 http-request**，以防止部署遭受分布式拒绝服务（例如，slowloris）攻击：

```
# and the haproxy stats socket is available at /var/run/haproxy.stats
global
  stats socket ./haproxy.stats level admin

defaults
  option http-server-close
  mode http
  timeout http-request 5s
  timeout connect 5s 1
  timeout server 10s
  timeout client 30s
```

1 **超时 http-request** 设置为 5 秒。HAProxy 为客户端 5 秒提供 * 以发送其整个 HTTP 请求。否则，HAProxy 会关闭连接并显示 *an 错误。

另外，当设置环境变量 **ROUTER_SLOWLORIS_TIMEOUT** 时，它会限制客户端发送整个 HTTP 请求所需的时间。否则，HAProxy 将关闭连接。

通过设置 环境变量，可以将信息捕获为路由器部署配置的一部分，不需要手动修改模板，而手动添加 HAProxy 设置要求您重新构建路由器 Pod 和维护路由器模板文件。

使用注解在 HAProxy 模板路由器中实施基本的 DDoS 保护，包括限制以下功能：

- 并发 TCP 连接数
- 客户端可以请求 TCP 连接的速率
- 可以发出 HTTP 请求的速率

这些是在每个路由上启用的，因为应用可能有完全不同的流量模式。

表 3.1. HAProxy 模板路由器设置

设置	描述
haproxy.router.openshift.io/rate-limit-connections	启用设置（例如，设置为 **true** ）。
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	此路由上相同 IP 地址可执行的并发 TCP 连接数。
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	客户端 IP 可以打开的 TCP 连接数。
haproxy.router.openshift.io/rate-limit-connections.rate-http	客户端 IP 在 3 秒期间内可以进行的 HTTP 请求数。

3.2.25. 启用 HAProxy 线程

使用 **--threads** 标志启用线程处理。此标志指定 HAProxy 路由器将使用的线程数量。

3.3. 部署自定义 HAPROXY 路由器

3.3.1. 概述

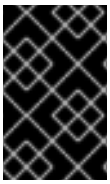
默认 HAProxy 路由器旨在满足大多数用户的需求。但是，它不会公开 HAProxy 的所有功能。因此，用户可能需要根据自己的需求修改路由器。

您可能需要在应用后端内实施新功能，或修改当前的操作。router 插件提供了进行此自定义所需的所有功能。

路由器容器集使用模板文件来创建所需的 HAProxy 配置文件。模板文件是 golang 模板。在处理模板时，路由器可以访问 OpenShift Container Platform 信息，包括路由器的部署配置、接受路由集和一些帮助程序功能。

当路由器 pod 启动时，并且每次重新加载时，它会创建一个 HAProxy 配置文件，然后启动 HAProxy。HAProxy 配置手册 描述了 HAProxy 的所有功能以及如何构建有效的配置文件。

configMap 可用于添加新模板到路由器 pod。通过这种方法，修改路由器部署配置，将 **configMap** 挂载为路由器 Pod 中的卷。**TEMPLATE_FILE** 环境变量设置为路由器 Pod 中模板文件的完整路径名称。



重要

无法保证在升级 OpenShift Container Platform 后路由器模板自定义仍然可以正常工作。

此外，路由器模板自定义必须应用到运行的路由器的模板版本。

或者，您也可以构建自定义路由器镜像，并在部署部分或所有路由器时使用它。不需要所有路由器来运行同一镜像。为此，请修改 ***haproxy-template.config*** 文件，再重新构建 路由器 镜像。新镜像推送到集群的 Docker 存储库，路由器的部署配置 **image:** 字段则使用新名称更新。更新集群时，需要重新构建并推送镜像。

在这两种情况下，路由器容器集都以模板文件开头。

3.3.2. 获取路由器配置模板

HAProxy 模板文件非常大且复杂。对于某些更改，修改现有模板可能比编写完整的替换版本更简单。您可以通过在 master 上运行并引用路由器 pod，从正在运行的路由器获取 ***haproxy-config.template*** 文件：

```
# oc get po
NAME                     READY     STATUS    RESTARTS   AGE
router-2-40fc3           1/1       Running   0          11d
# oc exec router-2-40fc3 cat haproxy-config.template > haproxy-config.template
# oc exec router-2-40fc3 cat haproxy.config > haproxy.config
```

另外，您可以登录到运行路由器的节点：

```
# docker run --rm --interactive=true --tty --entrypoint=cat \
    registry.redhat.io/openshift3/ose-haproxy-router:v{product-version} haproxy-config.template
```

镜像名称来自**容器镜像**。

将此内容保存到文件中，以用作自定义模板的基础。保存的 ***haproxy.config*** 显示实际运行的内容。

3.3.3. 修改路由器配置模板

3.3.3.1. 背景信息

模板基于 golang 模板。它可以引用路由器部署配置中的任何环境变量、下面描述的任何配置信息，以及路由器提供的帮助程序功能。

模板文件的结构镜像生成的 HAProxy 配置文件。在处理模板时，所有未被 **{{" something "}}** 包括的内容会直接复制到配置文件。被 **{{" something "}}** 包括的内容会被评估。生成的文本（如果有）复制到配置文件。

3.3.3.2. Go 模板操作

define 操作命名将包含已处理模板的文件。

```
{{define "/var/lib/haproxy/conf/haproxy.config"}}pipeline{{end}}
```

表 3.2. 模板路由器功能

功能	含义
processEndpointsForAlias(alias ServiceAliasConfig, svc ServiceUnit, action string) []Endpoint	返回有效端点列表。当操作为"shuffle"时，端点的顺序是随机的。
env(variable, default ...string) string	尝试从容器集获取 named 环境变量。如果未定义或为空，它将返回可选的第二个参数。否则，它将返回空字符串。
matchPattern(pattern, s string) bool	第一个参数是包含正则表达式的字符串，第二个参数是要测试的变量。返回一个布尔值，指示作为第一个参数提供的正则表达式是否与作为第二个参数提供的字符串匹配。
isInteger(s string) bool	确定给定变量是否为整数。
firstMatch(s string, allowedValues ...string) bool	将给定字符串与允许字符串的列表进行比较。通过列表向右返回第一个匹配扫描.
matchValues(s string, allowedValues ... string) bool	将给定字符串与允许字符串的列表进行比较。如果字符串为允许的值，则返回 "true"，否则返回 false。
generateRouteRegexp(hostname, path string, wildcard bool) string	生成与路由主机（和路径）匹配的正则表达式。第一个参数是主机名，第二个参数是路径，第三个参数是通配符布尔值。
genCertificateHostName(hostname string, wildcard bool) string	生成用于服务/匹配证书的主机名。第一个参数是主机名，第二个参数是通配符布尔值。
isTrue(s string) bool	确定给定变量是否包含"true"。

这些功能由 HAProxy 模板路由器插件提供。

3.3.3.3. 路由器提供的信息

本节回顾路由器在模板中提供的 OpenShift Container Platform 信息。路由器配置参数是 HAProxy 路由器插件提供的一组数据。这些字段可由 **(dot).Fieldname** 访问。

路由器配置参数下方的表格根据各种字段的定义展开。特别是 **.State** 拥有一组可接受的路由。

表 3.3. 路由器配置参数

字段	类型	描述
WorkingDir	字符串	文件要写入的目录，默认为 */var/lib/containers/router*
State	**map[string] (ServiceAliasConfig)**	路由。
ServiceUnits	**map[string]ServiceUnit**	服务查找。
DefaultCertificate	字符串	以 pem 格式到默认证书的完整路径名称。
PeerEndpoints	**[]Endpoint**	对等。
StatsUser	字符串	公开统计数据的用户名（如果模板支持）。
StatsPassword	字符串	公开统计数据的密码（如果模板支持）。
StatsPort	int	公开统计信息的端口（如果模板支持）。
BindPorts	bool	路由器是否应绑定默认端口。

表 3.4. 路由器 ServiceAliasConfig(A Route)

字段	类型	描述
Name	字符串	路由特定于用户的名称。
Namespace	字符串	路由的命名空间。
Host	字符串	主机名。例如：**www.example.com**。

字段	类型	描述
Path	字符串	可选路径。例如：**www.example.com/myservice**，其中 **myservice** 是路径。
TLSTermination	**routeapi.TLSTerminationType**	此后端的终止策略；驱动映射文件和路由器配置。
Certificates	**map[string]Certificate**	用于保护此后端的证书。按证书 ID 的键。
Status	**ServiceAliasConfigStatus**	指明需要永久保留的配置状态。
PreferPort	字符串	指明用户要公开的端口。如果为空，将为服务选择一个端口。
InsecureEdgeTerminationPolicy	**routeapi.InsecureEdgeTerminationPolicyType**	指明到边缘终端路由的不安全连接所需的行为：**none** (或 **disable**), **allow**, 或 **redirect**。
RoutingKeyName	字符串	用于模糊 cookie ID 的路由 + 命名空间名称的哈希值。
IsWildcard	bool	表示此服务单元需要通配符支持。
Annotations	**map[string]string**	附加到此路由的注解。
ServiceUnitNames	**map[string]int32**	支持此路由的服务集合，按服务名称键，并根据映射中其他条目的权重加上相应的权重。
ActiveServiceUnits	int	权重为非零的 **ServiceUnitName** 的数量。

ServiceAliasConfig 是服务的路由。由主机 + 路径唯一标识指定。默认模板使用 **{{range $cfgIdx、$cfg := .State }}** 迭代路由。在这样的 **{{range}}** 块中，模板可以使用 **$cfg.Field** 引用当前 **ServiceAliasConfig** 的任何字段。

表 3.5. Router ServiceUnit

字段	类型	描述
Name	字符串	对应于服务名称 + namespace 的名称。由 **ServiceUnit** 唯一标识。
EndpointTable	**[]Endpoint**	支持该服务的端点。这转换为路由器的最终后端实施。

ServiceUnit 是一种服务封装、支持该服务的端点，以及指向服务的路由。这是驱动创建路由器配置文件的数据

表 3.6. 路由器端点

字段	类型
ID	字符串
IP	字符串
Port	字符串
TargetName	字符串
PortName	字符串
IdHash	字符串
NoHealthCheck	bool

Endpoint 是 Kubernetes 端点的内部表示。

表 3.7. Router Certificate, ServiceAliasConfigStatus

字段	类型	描述
Certificate	字符串	代表一个公钥/私钥对。它通过 ID 来标识，该 ID 将成为文件名。CA 证书将不会设置 **PrivateKey**。
ServiceAliasConfigStatus	字符串	表示此配置所需的文件已持久保存到磁盘。有效值："saved", ""。

表 3.8. 路由器证书类型

字段	类型	描述
ID	字符串	
内容	字符串	证书。
PrivateKey	字符串	私钥。

表 3.9. 路由器 TLSTerminationType

字段	类型	描述
TLSTerminationType	字符串	指定安全通信将停止的位置。
InsecureEdgeTerminationPolicyType	字符串	表示路由不安全连接所需的行为。虽然每个路由器可能会对要公开的端口做出自己的决定，但通常这是端口 80。

TLSTerminationType 和 **InsecureEdgeTerminationPolicyType** 指定安全通信将停止的位置。

表 3.10. 路由器 TLSTerminationType 值

常数	值	含义
TLSTerminationEdge	**edge**	终止边缘路由器上的加密。
TLSTerminationPassthrough	**passthrough**	目的地终止加密，目的地负责解密流量。
TLSTerminationReencrypt	**reencrypt**	在边缘路由器中终止加密，并使用目的地提供的新证书重新加密。

表 3.11. Router InsecureEdgeTerminationPolicyType Values

类型	含义
Allow	流量发送到不安全端口（默认）上的服务器。
Disable	不安全的端口不允许流量。
Redirect	客户端重定向到安全端口。

None (**""**) 与 **Disable** 相同。

3.3.3.4. 注解

每个路由都可以附加注解。每个注释仅是一个名称和一个值。

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms
[...]
```

名称可以是与现有 Annotations 没有冲突的任何内容。值是任意字符串。字符串可以具有多个令牌，用空格分开。例如，**aa bb cc**.该模板使用 **{{index}}** 来提取注释的值。例如：

```
{{$balanceAlgo := index $cfg.Annotations "haproxy.router.openshift.io/balance"}}
```

这是如何将其用于相互客户端授权的示例。

```
{{ with $cnList := index $cfg.Annotations "whiteListCertCommonName" }}
  {{   if ne $cnList "" }}
    acl test ssl_c_s_dn(CN) -m str {{ $cnList }}
    http-request deny if !test
  {{   end }}
{{ end }}
```

然后，您可以使用此命令处理列入白名单的 CN。

```
$ oc annotate route <route-name> --overwrite whiteListCertCommonName="CN1 CN2 CN3"
```

如需更多信息，请参阅特定于路由的注解。

3.3.3.5. 环境变量

该模板可以使用路由器 pod 中存在的任何环境变量。环境变量可以在部署配置中设置。可以添加新的环境变量。

它们由 **env** 函数引用：

```
{{env "ROUTER_MAX_CONNECTIONS" "20000"}}
```

第一个字符串是 变量，第二个字符串是变量缺失或 **nil** 时的默认值。当 **ROUTER_MAX_CONNECTIONS** 未设置或为 **nil** 时，则使用 20000。环境变量是一个映射，其中键是环境变量名称，内容是 变量的值。

如需更多信息，请参阅特定于路由的环境变量。

3.3.3.6. 用法示例

以下是基于 HAProxy 模板文件的简单模板：

从注释开始：

```
{{/*
  Here is a small example of how to work with templates
  taken from the HAProxy template file.
*/}}
```

模板可以创建任意数量的输出文件。使用定义结构来创建输出文件。文件名指定为要定义的参数，定义块内直到匹配端的所有内容都会被写入为该文件的内容。

```
{{ define "/var/lib/haproxy/conf/haproxy.config" }}
global
{{ end }}
```

上方会将 **global** 复制到 */var/lib/haproxy/conf/haproxy.config* 文件，然后关闭该文件。

根据环境变量设置日志记录。

```
{{ with (env "ROUTER_SYSLOG_ADDRESS" "") }}
  log {{.}} {{env "ROUTER_LOG_FACILITY" "local1"}} {{env "ROUTER_LOG_LEVEL" "warning"}}
{{ end }}
```

env 函数提取环境变量的值。如果未定义环境变量或 **nil**，则返回第二个参数。

通过，使用 将 块中的 "."(dot)值设置为作为使用 的参数提供的任何值。**with** 操作测试点为 **nil**。如果没有 **nil**，则会处理到**最后**。在上面，假设 **ROUTER_SYSLOG_ADDRESS** 包含 */var/log/msg*、**ROUTER_LOG_FACILITY** 未定义，并且 **ROUTER_LOG_LEVEL** 包含 **info**。以下命令将复制到输出文件中：

```
  log /var/log/msg local1 info
```

每个接受的路由最终都会在配置文件中生成行。使用 **范围** 来进入接受的路由：

```
{{ range $cfgIdx, $cfg := .State }}
  backend be_http_{{$cfgIdx}}
{{end}}
```

.State 是 **ServiceAliasConfig** 的映射，其中键是路由名称. **范围** 步骤通过映射，每个传递都使用 **键** 设置 **$cfgIdx**，并将 **$cfg** 设置为指向描述路由的 **ServiceAliasConfig**。如果有两个路由名为 **myroute** 和 **heroute**，则上述命令会将以下内容复制到输出文件中：

```
  backend be_http_myroute
  backend be_http_hisroute
```

Route Annotations、**$cfg.Annotations** 也是注解名称作为键的映射，内容字符串作为值。路由可以根据需要拥有更多注解，其使用由模板作者定义。用户将注解代码到路由中，模板作者则自定义 HAProxy 模板来处理注解。

常见用法是索引注解以获取值。

```
{{$balanceAlgo := index $cfg.Annotations "haproxy.router.openshift.io/balance"}}
```

索引提取给定注解的值（若有）。因此，**$balanceAlgo** 将包含与注解或 **nil** 关联的字符串。如上所示，您可以测试非空**字符串**，并使用 **with** 结构对其执行操作。

```
{{ with $balanceAlgo }}
  balance $balanceAlgo
{{ end }}
```

此处，如果 **$balanceAlgo** 不是 **nil**，**平衡 $balanceAlgo** 将复制到输出文件中。

在第二个示例中，您想要根据注释中设置的超时值设置服务器超时。

```
$value := index $cfg.Annotations "haproxy.router.openshift.io/timeout"
```

现在 **$value** 会被评估，以确保它包含正确构建的字符串。**matchPattern** 函数接受正则表达式，如果参数满足表达式要求，则返回 **true**。

```
  matchPattern "[1-9][0-9]*(us\|ms\|s\|m\|h\|d)?" $value
```

这会接受 **5000ms**，但不接受 **7y**。结果可用于测试。

```
{{if (matchPattern "[1-9][0-9]*(us\|ms\|s\|m\|h\|d)?" $value) }}
  timeout server  {{$value}}
{{ end }}
```

它还可用于匹配令牌：

```
matchPattern "roundrobin|leastconn|source" $balanceAlgo
```

另外，也可以使用 **matchValues** 来匹配令牌：

```
matchValues $balanceAlgo "roundrobin" "leastconn" "source"
```

3.3.4. 使用 ConfigMap 替换路由器配置模板

您可以使用 ConfigMap 来自定义路由器实例，而无需重新构建路由器镜像。可以修改 ***haproxy-config.template***、***reload-haproxy*** 和其他脚本，以及创建和修改路由器环境变量。

1. 复制您要修改的 ***haproxy-config.template***，如上所述根据需要进行修改。
2. 创建 ConfigMap：

   ```
   $ oc create configmap customrouter --from-file=haproxy-config.template
   ```

   customrouter ConfigMap 现在包含修改后的 ***haproxy-config.template*** 文件的副本。
3. 修改路由器部署配置，将 ConfigMap 挂载为文件，并将 **TEMPLATE_FILE** 环境变量指向该文件。这可以通过 **oc set env** 和 **oc set volume** 命令完成，或者通过编辑路由器部署配置来完成。

   使用 oc 命令

   ```
   $ oc set volume dc/router --add --overwrite \
      --name=config-volume \
      --mount-path=/var/lib/haproxy/conf/custom \
      --source='{"configMap": { "name": "customrouter"}}'
   $ oc set env dc/router \
      TEMPLATE_FILE=/var/lib/haproxy/conf/custom/haproxy-config.template
   ```

   编辑路由器部署配置

   使用 **oc edit dc router**，使用文本编辑器编辑路由器部署配置。

   ```
   ...
          - name: STATS_USERNAME
            value: admin
          - name: TEMPLATE_FILE 1
            value: /var/lib/haproxy/conf/custom/haproxy-config.template
          image: openshift/origin-haproxy-routerp
   ...
          terminationMessagePath: /dev/termination-log
          volumeMounts: 2
          - mountPath: /var/lib/haproxy/conf/custom
            name: config-volume
        dnsPolicy: ClusterFirst
   ```

```
...
    terminationGracePeriodSeconds: 30
    volumes: 3
    - configMap:
        name: customrouter
      name: config-volume
...
```

1. 在 **spec.container.env** 字段中，添加 **TEMPLATE_FILE** 环境变量以指向挂载的 *haproxy-config.template* 文件。
2. 添加 **spec.container.volumeMounts** 字段以创建挂载点。
3. 添加新的 **spec.volumes** 字段来引用 ConfigMap。

保存更改并退出编辑器。这将重新启动路由器。

3.3.5. 使用粘滞表

以下示例自定义可在高可用性路由设置中使用，以使用在对等点间同步的粘滞表。

添加 Peer 部分

要在对等点间同步粘滞位，您必须在 HAProxy 配置中定义对等部分。本节决定了 HAProxy 如何识别并连接到同级服务器。插件在 **.PeerEndpoints** 变量下向模板提供数据，以便您可以轻松地识别路由器服务的成员。您可以通过添加以下内容，将 peer 部分添加到路由器镜像中的 *haproxy-config.template* 文件中：

```
{{ if (len .PeerEndpoints) gt 0 }}
peers openshift_peers
  {{ range $endpointID, $endpoint := .PeerEndpoints }}
    peer {{$endpoint.TargetName}} {{$endpoint.IP}}:1937
  {{ end }}
{{ end }}
```

更改重新加载脚本

使用粘滞位时，您可以选择告知 HAProxy 在 peer 部分中应考虑本地主机的名称。在创建端点时，插件会尝试将 **TargetName** 设置为端点的 **TargetRef.Name** 的值。如果没有设置 **TargetRef**，它会将 **TargetName** 设置为 IP 地址。**TargetRef.Name** 与 Kubernetes 主机名对应，因此您可以将 **-L** 选项添加到 **reload-haproxy** 脚本中，以标识 peer 部分中的本地主机。

```
peer_name=$HOSTNAME 1

if [ -n "$old_pid" ]; then
  /usr/sbin/haproxy -f $config_file -p $pid_file -L $peer_name -sf $old_pid
else
  /usr/sbin/haproxy -f $config_file -p $pid_file -L $peer_name
fi
```

1. 必须与 peer 部分中使用的端点目标名称匹配。

修改后端

最后，若要在后端使用 stick-tables，您可以修改 HAProxy 配置以使用 stick-tables 和 peer 设置。以下是将 TCP 连接的现有后端更改为使用 stick-table 的示例：

```
        {{ if eq $cfg.TLSTermination "passthrough" }}
backend be_tcp_{{$cfgIdx}}
  balance leastconn
  timeout check 5000ms
  stick-table type ip size 1m expire 5m{{ if (len $.PeerEndpoints) gt 0 }} peers openshift_peers {{ end
}}
  stick on src
        {{ range $endpointID, $endpoint := $serviceUnit.EndpointTable }}
  server {{$endpointID}} {{$endpoint.IP}}:{{$endpoint.Port}} check inter 5000ms
        {{ end }}
      {{ end }}
```

在修改后，您可以重建路由器。

3.3.6. 重建路由器

若要重建路由器，您需要复制正在运行的路由器上存在的多个文件。创建一个工作目录并从路由器复制文件：

```
# mkdir - myrouter/conf
# cd myrouter
# oc get po
NAME                    READY     STATUS    RESTARTS   AGE
router-2-40fc3          1/1       Running   0          11d
# oc exec router-2-40fc3 cat haproxy-config.template > conf/haproxy-config.template
# oc exec router-2-40fc3 cat error-page-503.http > conf/error-page-503.http
# oc exec router-2-40fc3 cat default_pub_keys.pem > conf/default_pub_keys.pem
# oc exec router-2-40fc3 cat ../Dockerfile > Dockerfile
# oc exec router-2-40fc3 cat ../reload-haproxy > reload-haproxy
```

您可以编辑或替换其中任何一个文件。但是，*conf/haproxy-config.template* 和 *reload-haproxy* 最有可能被修改。

更新文件后：

```
# docker build -t openshift/origin-haproxy-router-myversion .
# docker tag openshift/origin-haproxy-router-myversion 172.30.243.98:5000/openshift/haproxy-router-
myversion (1)
# docker push 172.30.243.98:5000/openshift/origin-haproxy-router-pc:latest (2)
```

1. 使用存储库标记 version。在本例中，存储库是 **172.30.243.98:5000**。
2. 将标记的版本推送到存储库。可能需要先 **docker login** 存储库。

要使用新路由器，请通过更改 **image:** 字符串或将 **--images=<repo>/<image>:<tag>** 标志添加到 **oc adm router** 命令来编辑路由器部署配置。

调试更改时，设置 **imagePullPolicy:Always** 在部署配置中，强制在每次 pod 创建时拉取镜像。调试完成后，您可以将其改回到 **imagePullPolicy:IfNotPresent** 以避免每次 pod 启动时都拉取。

3.4. 将 HAPROXY 路由器配置为使用 PROXY 协议

3.4.1. 概述

默认情况下，HAProxy 路由器要求进入到不安全、边缘和重新加密路由的连接才能使用 HTTP。但是，您可以使用 PROXY 协议 将路由器配置为预期传入请求。本节论述了如何将 HAProxy 路由器和外部负载均衡器配置为使用 PROXY 协议。

3.4.2. 为什么使用 PROXY 协议？

当代理服务器或负载平衡器等中间服务转发 HTTP 请求时，它会将连接的源地址附加到请求的"Forwarded"标头，从而将此信息提供给后续请求，以及最终将请求转发到的后端服务。但是，如果连接被加密，则无法修改"Forwarded"标头。在这种情况下，HTTP 标头无法在转发请求时准确传达原始源地址。

为了解决这个问题，一些负载均衡器使用 PROXY 协议封装 HTTP 请求，作为只转发 HTTP 的替代选择。封装使负载平衡器能够在不修改转发请求本身的情况下向请求添加信息。特别是，这意味着负载均衡器即使在转发加密的连接时也能通信源地址。

HAProxy 路由器可以配置为接受 PROXY 协议并解封 HTTP 请求。由于路由器终止对边缘和再加密路由的加密，因此路由器随后可以更新请求中的 "Forwarded" HTTP 标头（及相关的 HTTP 标头），附加使用 PROXY 协议通信的任何源地址。

警告

PROXY 协议和 HTTP 不兼容，不可混合。如果您在路由器前面使用负载均衡器，则必须使用 PROXY 协议或 HTTP。将一个协议配置为使用一个协议，另一个协议使用其他协议会导致路由失败。

3.4.3. 使用 PROXY 协议

默认情况下，HAProxy 路由器不使用 PROXY 协议。可以使用 **ROUTER_USE_PROXY_PROTOCOL** 环境变量配置路由器，以预期传入连接的 PROXY 协议：

启用 PROXY 协议

```
$ oc set env dc/router ROUTER_USE_PROXY_PROTOCOL=true
```

将变量设置为 **true** 或 **TRUE** 以外的任何值，以禁用 PROXY 协议：

禁用 PROXY 协议

```
$ oc set env dc/router ROUTER_USE_PROXY_PROTOCOL=false
```

如果您在路由器中启用 PROXY 协议，则必须将路由器前面的负载均衡器配置为使用 PROXY 协议。以下是配置 Amazon Elastic Load Balancer(ELB)服务以使用 PROXY 协议的示例。本例假定 ELB 将端口 80(HTTP)、443(HTTPS)和 5000（镜像注册表）转发到一个或多个 EC2 实例上运行的路由器。

将 Amazon ELB 配置为使用 PROXY 协议

1. 要简化后续步骤，首先设置一些 shell 变量：

```
$ lb='infra-lb' 1
$ instances=( 'i-079b4096c654f563c' ) 2
$ secgroups=( 'sg-e1760186' ) 3
$ subnets=( 'subnet-cf57c596' ) 4
```

1. 您的 ELB 的名称。
2. 运行路由器的实例。
3. 此 ELB 的安全组或组。
4. 此 ELB 的子网。

2. 接下来，使用适当的监听程序、安全组和子网创建 ELB。

注意

您必须将所有监听程序配置为使用 TCP 协议，而不是 HTTP 协议。

```
$ aws elb create-load-balancer --load-balancer-name "$lb" \
  --listeners \
   'Protocol=TCP,LoadBalancerPort=80,InstanceProtocol=TCP,InstancePort=80' \
   'Protocol=TCP,LoadBalancerPort=443,InstanceProtocol=TCP,InstancePort=443' \
   'Protocol=TCP,LoadBalancerPort=5000,InstanceProtocol=TCP,InstancePort=5000' \
  --security-groups $secgroups \
  --subnets $subnets
{
   "DNSName": "infra-lb-2006263232.us-east-1.elb.amazonaws.com"
}
```

3. 使用 ELB 注册路由器实例或实例：

```
$ aws elb register-instances-with-load-balancer --load-balancer-name "$lb" \
  --instances $instances
{
   "Instances": [
      {
         "InstanceId": "i-079b4096c654f563c"
      }
   ]
}
```

4. 配置 ELB 的健康检查：

```
$ aws elb configure-health-check --load-balancer-name "$lb" \
  --health-check
'Target=HTTP:1936/healthz,Interval=30,UnhealthyThreshold=2,HealthyThreshold=2,Timeout=5
'
{
   "HealthCheck": {
```

```
        "HealthyThreshold": 2,
        "Interval": 30,
        "Target": "HTTP:1936/healthz",
        "Timeout": 5,
        "UnhealthyThreshold": 2
      }
}
```

5. 最后，创建一个启用了 **ProxyProtocol** 属性的负载均衡器策略，并在 ELB 的 TCP 端口 80 和 443 中配置它：

```
$ aws elb create-load-balancer-policy --load-balancer-name "$lb" \
  --policy-name "${lb}-ProxyProtocol-policy" \
  --policy-type-name 'ProxyProtocolPolicyType' \
  --policy-attributes 'AttributeName=ProxyProtocol,AttributeValue=true'
$ for port in 80 443
  do
    aws elb set-load-balancer-policies-for-backend-server \
     --load-balancer-name "$lb" \
     --instance-port "$port" \
     --policy-names "${lb}-ProxyProtocol-policy"
  done
```

验证配置

您可以按如下所示检查负载均衡器以验证配置是否正确：

```
$ aws elb describe-load-balancers --load-balancer-name "$lb" |
    jq '.LoadBalancerDescriptions| [.[]|.ListenerDescriptions]'
[
  [
    {
      "Listener": {
        "InstancePort": 80,
        "LoadBalancerPort": 80,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": ["infra-lb-ProxyProtocol-policy"] 1
    },
    {
      "Listener": {
        "InstancePort": 443,
        "LoadBalancerPort": 443,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": ["infra-lb-ProxyProtocol-policy"] 2
    },
    {
      "Listener": {
        "InstancePort": 5000,
        "LoadBalancerPort": 5000,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
```

```
      },
      "PolicyNames": [] 3
    }
  ]
]
```

1 TCP 端口 80 的监听器应具有使用 PROXY 协议的策略。

2 TCP 端口 443 的监听程序应具有相同的策略。

3 TCP 端口 5000 的监听程序 **不应** 具有该策略。

或者，如果您已经配置了 ELB，但没有配置为使用 PROXY 协议，则需要更改 TCP 端口 80 的现有监听程序以使用 TCP 协议而不是 HTTP（TCP 端口 443 应已使用 TCP 协议）：

```
$ aws elb delete-load-balancer-listeners --load-balancer-name "$lb" \
  --load-balancer-ports 80
$ aws elb create-load-balancer-listeners --load-balancer-name "$lb" \
  --listeners 'Protocol=TCP,LoadBalancerPort=80,InstanceProtocol=TCP,InstancePort=80'
```

验证协议更新

验证协议是否已更新，如下所示：

```
$ aws elb describe-load-balancers --load-balancer-name "$lb" |
  jq '[.LoadBalancerDescriptions[]|.ListenerDescriptions]'
[
  [
    {
      "Listener": {
        "InstancePort": 443,
        "LoadBalancerPort": 443,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    },
    {
      "Listener": {
        "InstancePort": 5000,
        "LoadBalancerPort": 5000,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    },
    {
      "Listener": {
        "InstancePort": 80,
        "LoadBalancerPort": 80,
        "Protocol": "TCP", 1
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
```

```
    }
  ]
]
```

1 所有监听器（包括 TCP 端口 80 的监听器）都应使用 TCP 协议。

然后，创建一个负载均衡器策略，并将其添加到 ELB 中，如上一步第 5 步所述。

第 4 章 部署 RED HAT CLOUDFORMS

4.1. 在 OPENSHIFT CONTAINER PLATFORM 上部署 RED HAT CLOUDFORMS

4.1.1. 简介

OpenShift Container Platform 安装程序包括 Ansible 角色 **openshift-management**，以及用于在 OpenShift Container Platform 上部署 Red Hat CloudForms 4.6（CloudForms Management Engine 5.9 或 CFME）的 playbook。

警告

当前实施与 Red Hat CloudForms 4.5 的技术预览部署过程不兼容，如 OpenShift Container Platform 3.6 文档 中所述。

在 OpenShift Container Platform 上部署 Red Hat CloudForms 时，需要做出两个主要决策：

1. 您是否想要外部或容器化（也称为 *podified*）PostgreSQL 数据库？
2. 哪个存储类将支持您的持久性卷(PV)？

对于第一个决定，您可以使用两种方式之一部署 Red Hat CloudForms，具体取决于 Red Hat CloudForms 使用的 PostgreSQL 数据库的位置：

Deployment Variant	描述
完全容器化	所有应用程序服务和 PostgreSQL 数据库都作为 pod 在 OpenShift Container Platform 上运行。
外部数据库	应用程序使用外部托管的 PostgreSQL 数据库服务器，所有其他服务则作为 pod 在 OpenShift Container Platform 上运行。

对于第二个决定，**openshift-management** 角色提供了覆盖许多默认部署参数的自定义选项。这包括以下存储类选项来支持 PV：

Storage class	描述
NFS（默认）	本地，在集群上
NFS 外部	NFS（如存储设备）

Storage class	描述
云供应商	使用云供应商（Google Cloud Engine、Amazon Web Services 或 Microsoft Azure）的自动存储置备。
预配置（高级）	假设您提前创建了所有内容

本指南的主题包括：在 OpenShift Container Platform 上运行 Red Hat CloudForms 的要求、可用配置变量的说明，以及在初始 OpenShift Container Platform 安装期间或置备集群之后运行安装程序的说明。

4.2. OPENSHIFT CONTAINER PLATFORM 上 RED HAT CLOUDFORMS 的要求

下表中列出了默认要求。可以通过 自定义模板参数 来覆盖它们。

重要

如果无法满足这些要求，应用性能将会受到影响，甚至可能无法部署。

表 4.1. 默认要求

项	要求	描述	自定义参数
应用程序内存	≥ 4.0 Gi	应用程序所需的最小内存	**APPLICATION_MEM_REQ**
应用程序存储	≥ 5.0 Gi	应用程序所需的最小 PV 大小	**APPLICATION_VOLUME_CAPACITY**
PostgreSQL 内存	≥ 6.0 Gi	数据库所需的最小内存	**POSTGRESQL_MEM_REQ**
PostgreSQL 存储	≥ 15.0 Gi	数据库所需的最小 PV 大小	**DATABASE_VOLUME_CAPACITY**
集群主机	≥ 3	集群中的主机数量	N/A

满足这些要求：

- 您必须具有几个集群节点。
- 您的集群节点必须具有大量可用内存。
- 您必须有多个 GiB 的可用存储，无论是本地还是云供应商。
- 可以通过为模板参数提供覆盖值来更改 PV 大小。

4.3. 配置角色变量

4.3.1. 概述

以下小节描述了 Ansible 清单文件中可能使用的角色变量，用于控制 运行安装程序 时红帽 CloudForms 安装的行为。

4.3.2. 常规变量

变量	必填	默认	描述
openshift_management_install_management	否	**false**	布尔值，设置为 **true** 以安装应用程序。
openshift_management_app_template	是	**cfme-template**	要安装的红帽 CloudForms 的部署变体。为容器化数据库设置 **cfme-template**，或为外部数据库设置 **cfme-template-ext-db**。
openshift_management_project	否	**openshift-management**	红帽 CloudForms 安装的命名空间（项目）
openshift_management_project_description	否	**CloudForms Management Engine**	命名空间（项目）描述。
openshift_management_username	否	**admin**	默认管理用户名。更改此值不会更改用户名；只有在您更改了名称并运行集成脚本（如 添加容器提供程序的脚本）时，才会更改此值。
openshift_management_password	否	**smartvm**	默认管理密码。更改此值不会更改密码；只有在您更改了密码并且正在运行集成脚本（如用于 添加容器提供程序的脚本）时，才会更改此值。

4.3.3. 自定义模板参数

您可以使用 **openshift_management_template_parameters** Ansible 角色变量来指定要在应用程序或 PV 模板中覆盖的任何模板参数。

例如，如果要降低 PostgreSQL pod 的内存要求，您可以设置以下内容：

```
openshift_management_template_parameters={'POSTGRESQL_MEM_REQ': '1Gi'}
```

处理红帽 CloudForms 模板后，**1Gi** 将用于 **POSTGRESQL_MEM_REQ** 模板参数的值。

并非所有模板参数都存在于 *两种* 模板变体中（容器化或外部数据库）。例如，虽然 podified 数据库模板具有 **POSTGRESQL_MEM_REQ** 参数，但外部 db 模板中没有这样的参数，因为不需要此信息，因为没有要求 pod 的数据库。

因此，如果您要覆盖模板参数，请非常小心。包含模板中未定义的参数将导致错误。如果在 **Ensure the Management App is created** 任务期间收到错误，请在再次运行安装程序前先运行 卸载脚本。

4.3.4. 数据库变量

4.3.4.1. 容器化（pod）数据库

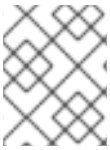
cfme-template.yaml 文件中任何 **POSTGRES _* 或DATABASE_*** 模板参数都可以通过清单文件中的 **openshift_management_template_parameters** 哈希进行自定义。

4.3.4.2. 外部数据库

cfme-template-ext-db.yaml 文件中的任何 **POSTGRES _*** 或 **DATABASE_*** 模板参数都可以通过清单文件中的 **openshift_management_template_parameters** 哈希进行自定义。

外部 PostgreSQL 数据库要求您提供数据库连接参数。您必须在清单中的 **openshift_management_template_parameters** 参数中设置所需的连接密钥。以下键是必需的：

- **DATABASE_USER**
- **DATABASE_PASSWORD**
- **DATABASE_IP**
- **DATABASE_PORT** （大多数 PostgreSQL 服务器在端口 **5432**上运行）
- **DATABASE_NAME**



注意

确保您的外部数据库正在运行 PostgreSQL 9.5，或者您可能无法成功部署 CloudForms 应用。

您的清单将包含类似如下的行：

```
[OSEv3:vars]
openshift_management_app_template=cfme-template-ext-db (1)
openshift_management_template_parameters={'DATABASE_USER': 'root',
'DATABASE_PASSWORD': 'mypassword', 'DATABASE_IP': '10.10.10.10', 'DATABASE_PORT':
'5432', 'DATABASE_NAME': 'cfme'}
```

(1) 将 **openshift_management_app_template** 参数设置为 **cfme-template-ext-db**。

4.3.5. 存储类变量

变量	必填	默认	描述
openshift_management_storage_class	否	**nfs**	要使用的存储类型。选项包括 **nfs**、**nfs_external**、**preconfigured** 或 **cloudprovider**。
openshift_management_storage_nfs_external_hostname	否	**false**	如果您使用的是外部 NFS 服务器，如 NetApp 设备，则必须在此处设置主机名。如果没有使用外部 NFS，则该值保留为 **false**。另外，外部 NFS 要求您创建将支持应用程序 PV 的 NFS 导出以及数据库 PV（可选）。
openshift_management_storage_nfs_base_dir	否	**/exports/**	如果您使用外部 NFS，则可以在此处将基本路径设置为导出位置。对于本地 NFS，如果您要更改用于本地 NFS 导出的默认路径，您还可以更改此值。
openshift_management_storage_nfs_local_hostname	否	**false**	如果您的清单中没有 **[nfs]** 组，或者只想手动定义集群中的本地 NFS 主机，请将这个参数设置为首选 NFS 服务器的主机名。服务器必须是 OpenShift Container Platform 集群的一部分。

4.3.5.1. NFS（默认）

NFS 存储类最适合概念验证和测试部署。它也是部署的默认存储类。选择时不需要额外的配置。

此存储类将集群主机上的 NFS（默认为清单文件中的第一个 master）配置为支持所需的 PV。应用需要一个 PV，而数据库（可能外部托管）可能需要一秒钟。Red Hat CloudForms 应用所需的最小 PV 大小为 5GiB，PostgreSQL 数据库为 15GiB（如果专用于 NFS，则为卷或分区上的最小可用空间为 20GiB）。

自定义通过以下角色变量提供：

- **openshift_management_storage_nfs_base_dir**
- **openshift_management_storage_nfs_local_hostname**

4.3.5.2. NFS 外部

外部 NFS 依靠预先配置的 NFS 服务器为所需的 PV 提供导出。对于外部 NFS，您必须有一个 **cfme-app** 和可选的一个 **cfme-db** （容器化数据库）导出。

配置通过以下角色变量提供：

- **openshift_management_storage_nfs_external_hostname**
- **openshift_management_storage_nfs_base_dir**

openshift_management_storage_nfs_external_hostname 参数必须设置为外部 NFS 服务器的主机名或 IP。

如果 */exports* 不是您的导出的父目录，则必须通过 **openshift_management_storage_nfs_base_dir** 参数设置基础目录。

例如，如果您的服务器导出为 */exports/hosted/prod/cfme-app*，则必须设置 **openshift_management_storage_nfs_base_dir=/exports/hosted/prod**。

4.3.5.3. 云供应商

如果您要将 OpenShift Container Platform 云供应商集成用于存储类，Red Hat CloudForms 也可以使用云供应商存储来支持其所需的 PV。要使此功能正常工作，您必须已配置了 **openshift_cloudprovider_kind** 变量（用于 AWS 或 GCE），以及特定于您所选云供应商的所有关联参数。

当使用此存储类创建应用程序时，需要使用配置的云供应商存储集成自动置备所需的 PV。

没有额外的变量可用于配置此存储类的行为。

4.3.5.4. 预配置（高级）

预配置 存储类意味着您准确知道您的操作，并且所有存储要求都已提前处理。通常，这意味着您已创建了正确大小的 PV。安装程序不会进行任何操作来修改任何存储设置。

没有额外的变量可用于配置此存储类的行为。

4.4. 运行安装程序

4.4.1. 在 OpenShift Container Platform 安装期间或之后部署 Red Hat CloudForms

您可以选择在初始 OpenShift Container Platform 安装过程中或置备集群后部署 Red Hat CloudForms：

1. 确定在清单文件 **[OSEv3:vars]** 部分下的清单文件中将 **openshift_management_install_management** 设置为 **true** ：

```
[OSEv3:vars]
openshift_management_install_management=true
```

2. 在清单文件中设置任何其他红帽 CloudForms 角色变量，如 配置角色变量 中所述。清单文件示例中提供了有助于执行此操作 的资源。

3. 根据是否已置备 OpenShift Container Platform，选择要运行的 playbook:

a. 如果要在安装 OpenShift Container Platform 集群的同时安装 Red Hat CloudForms，请调用标准 ***config.yml*** playbook，如 运行安装 Playbook 所述以开始 OpenShift Container Platform 集群和 Red Hat CloudForms 安装。

b. 如果要在已置备的 OpenShift Container Platform 集群上安装 Red Hat CloudForms，请切换到 playbook 目录，并直接调用 Red Hat CloudForms playbook 开始安装：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v [-i /path/to/inventory] \
    playbooks/openshift-management/config.yml
```

4.4.2. 清单文件示例

以下小节演示了在 OpenShift Container Platform 中显示 Red Hat CloudForms 的各种配置的清单文件示例片段，可帮助您开始。

注意

如需完整的变量描述，请参阅配置角色变量。

4.4.2.1. 所有默认值

本例最简单的方法是使用所有默认值和选项。这将实现完全容器化（指定）红帽 CloudForms 安装。所有应用程序组件以及 PostgreSQL 数据库都是在 OpenShift Container Platform 中创建的：

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
```

4.4.2.2. 外部 NFS 存储

如前例所示，除了在集群中使用本地 NFS 服务外，它使用的是现有的外部 NFS 服务器（如存储设备）。请注意两个新参数：

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_storage_class=nfs_external (1)
openshift_management_storage_nfs_external_hostname=nfs.example.com (2)
```

1. 将设置为 **nfs_external**。
2. 设置为 NFS 服务器的主机名。

如果外部 NFS 主机在不同的父目录下导出目录，如 ***/exports/hosted/prod***，请添加以下额外变量：

```
openshift_management_storage_nfs_base_dir=/exports/hosted/prod
```

4.4.2.3. 覆盖 PV 大小

这个示例覆盖持久性卷(PV)大小。PV 大小必须通过 **openshift_management_template_parameters** 设置，这样可确保应用程序和数据库能够在创建的 PV 上发出声明，而无需相互干扰：

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_template_parameters={'APPLICATION_VOLUME_CAPACITY': '10Gi',
'DATABASE_VOLUME_CAPACITY': '25Gi'}
```

4.4.2.4. 覆盖内存要求

在测试或概念验证安装中，您可能需要降低应用程序和数据库内存要求，以符合您的容量。请注意，减少内存限值可能会导致性能降低或完全失败来初始化应用程序：

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_template_parameters={'APPLICATION_MEM_REQ': '3000Mi',
'POSTGRESQL_MEM_REQ': '1Gi', 'ANSIBLE_MEM_REQ': '512Mi'}
```

本例指示安装程序处理应用程序模板，参数 **APPLICATION_MEM_REQ** 设置为 **3000Mi**，**POSTGRESQL_MEM_REQ** 设置为 **1Gi**，**ANSIBLE_MEM_REQ** 设置为 **512Mi**。

这些参数可以和上一示例 覆盖 PV Sizes 中显示的参数结合使用。

4.4.2.5. 外部 PostgreSQL 数据库

要使用外部数据库，您必须将 **openshift_management_app_template** 参数值改为 **cfme-template-ext-db**。

此外，必须使用 **openshift_management_template_parameters** 变量提供数据库连接信息。如需了解更多详细信息，请参阅配置角色变量。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template-ext-db
openshift_management_template_parameters={'DATABASE_USER': 'root',
'DATABASE_PASSWORD': 'mypassword', 'DATABASE_IP': '10.10.10.10', 'DATABASE_PORT':
'5432', 'DATABASE_NAME': 'cfme'}
```

重要

确保您正在运行 PostgreSQL 9.5，或者您可能无法成功部署应用。

4.5. 启用容器提供程序集成

4.5.1. 添加单一容器提供程序

在 OpenShift Container Platform 上部署 Red Hat CloudForms 后，如 运行安装程序 所述，有两种方法可用于启用容器供应商集成。您可以将 OpenShift Container Platform 手动添加为容器供应商，也可以尝试使用此角色中包含的 playbook。

4.5.1.1. 手动添加

有关将 OpenShift Container Platform 集群手动添加为容器供应商的步骤，请参阅以下 Red Hat CloudForms 文档：

- 与 OpenShift Container Platform 集成

4.5.1.2. 自动添加

可以使用此角色中包含的 playbook 来完成自动化容器提供程序集成。

此 playbook：

1. 收集所需的身份验证 secret。
2. 查找指向红帽 CloudForms 应用程序和集群 API 的公共路由。
3. 发出 REST 调用，以将 OpenShift Container Platform 集群添加为容器供应商。

进入 playbook 目录并运行容器供应商 playbook：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v [-i /path/to/inventory] \
    openshift-management/add_container_provider.yml
```

4.5.2. 多个容器供应商

除了提供 playbook 以将当前 OpenShift Container Platform 集群集成到 Red Hat CloudForms 部署中，此角色包含一个脚本，允许您在任何任意红帽 CloudForms 服务器中添加多个容器平台作为容器供应商。容器平台可以是 OpenShift Container Platform 或 OpenShift Origin。

在运行 playbook 时，使用多个提供程序脚本需要在 CLI 上手动配置和设置 **EXTRA_VARS** 参数。

4.5.2.1. 准备脚本

要准备多个供应商脚本，请完成以下手动配置：

1. 复制 */usr/share/ansible/openshift-ansible/roles/openshift_management/files/examples/container_providers.yml* 示例，如 */tmp/cp.yml*。您将修改此文件。
2. 如果您更改了红帽 CloudForms 名称或密码，请更新您复制的 *container_providers.yml* 文件中的 **management_server** 键中的 **hostname**、user**和 password** 参数。
3. 填写您要添加为容器供应商的每个容器平台集群的 **container_providers** 键下的条目。
   a. 必须配置以下参数：
      - **auth_key** - 这是具有 **cluster-admin** 权限的服务帐户的令牌。
      - **hostname** - 这是指向集群 API 的主机名。每个容器提供程序必须具有唯一的主机名。
      - **name -** 这是要在红帽 CloudForms 服务器容器提供程序概览页面中显示的集群名称。这必须是唯一的。

      提示

      从集群中获取 **auth_key** bearer 令牌：

      ```
      $ oc serviceaccounts get-token -n management-infra management-admin
      ```

   b. 可选择性地配置以下参数：

- **port** - 如果您的容器平台集群在 **8443** 之外的端口上运行 API，则更新此密钥。
- **endpoint** - 您可以启用 SSL验证(**verify_ssl**)，或将验证设置更改为 **ssl-with-validation**。目前不支持自定义可信 CA 证书。

4.5.2.1.1. 示例

例如，请考虑以下情况：

- 您可以将 *container_providers.yml* 文件复制到 */tmp/cp.yml*。
- 您需要添加两个 OpenShift Container Platform 集群。
- 您的红帽 CloudForms 服务器在 **mgmt.example.com**上运行

在这种情况下，您将自定义 */tmp/cp.yml*，如下所示：

```
container_providers:
  - connection_configurations:
      - authentication: {auth_key: "<token>", authtype: bearer, type: AuthToken} (1)
        endpoint: {role: default, security_protocol: ssl-without-validation, verify_ssl: 0}
    hostname: "<provider_hostname1>"
    name: <display_name1>
    port: 8443
    type: "ManageIQ::Providers::Openshift::ContainerManager"
  - connection_configurations:
      - authentication: {auth_key: "<token>", authtype: bearer, type: AuthToken} (2)
        endpoint: {role: default, security_protocol: ssl-without-validation, verify_ssl: 0}
    hostname: "<provider_hostname2>"
    name: <display_name2>
    port: 8443
    type: "ManageIQ::Providers::Openshift::ContainerManager"
management_server:
  hostname: "<hostname>"
  user: <user_name>
  password: <password>
```

(1)(2) 将 **<token>** 替换为此集群的管理令牌。

4.5.2.2. 运行 Playbook

要运行 multi-providers 集成脚本，您必须提供容器提供程序配置文件的路径，作为 **ansible-playbook** 命令的 **EXTRA_VARS** 参数。使用 **-e** （或 **--extra-vars**）参数将 **container_providers_config** 设置为配置文件路径。进入 playbook 目录并运行 playbook:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v [-i /path/to/inventory] \
    -e container_providers_config=/tmp/cp.yml \
    playbooks/openshift-management/add_many_container_providers.yml
```

playbook 完成后，您应在红帽 CloudForms 服务中找到两个新的容器提供程序。导航到 **Compute** → **Containers** → **Providers** 页面，查看概述。

4.5.3. 刷新供应商

添加单个或多个容器提供程序后，必须在红帽 CloudForms 中刷新新的提供程序，以获取关于容器提供程序和所管理容器的所有最新数据。这涉及导航到红帽 CloudForms Web 控制台中的各个提供程序，然后单击每个控制台的刷新按钮。

有关步骤，请参阅以下 Red Hat CloudForms 文档：

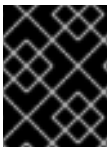
- 管理提供程序

4.6. 卸载 RED HAT CLOUDFORMS

4.6.1. 运行卸载 Playbook

要从 OpenShift Container Platform 卸载并删除部署的 Red Hat CloudForms 安装，请切换到 playbook 目录并运行 ***uninstall.yml*** playbook:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v [-i /path/to/inventory] \
    playbooks/openshift-management/uninstall.yml
```



重要

NFS 导出定义和数据不会自动删除。在尝试初始化新部署之前，您必须手动从旧应用或数据库部署中清除任何数据。

4.6.2. 故障排除

无法擦除旧的 PostgreSQL 数据可能会导致级联错误，从而导致 **postgresql** pod 进入 **crashloopbackoff** 状态。这会阻止 **cfme** pod 启动。**crashloopbackoff** 的原因是，之前部署期间创建的数据库 NFS 导出的文件权限不正确。

要继续，请从 PostgreSQL 导出中删除所有数据，并删除 pod（而不是部署器 Pod）。例如，如果您有以下 pod：

```
$ oc get pods
NAME                READY     STATUS             RESTARTS   AGE
httpd-1-cx7fk       1/1       Running            1          21h
cfme-0              0/1       Running            1          21h
memcached-1-vkc7p   1/1       Running            1          21h
postgresql-1-deploy 1/1       Running            1          21h
postgresql-1-6w2t4  0/1       CrashLoopBackOff   1          21h
```

然后，您将：

1. 从数据库 NFS 导出中删除数据。
2. 运行：

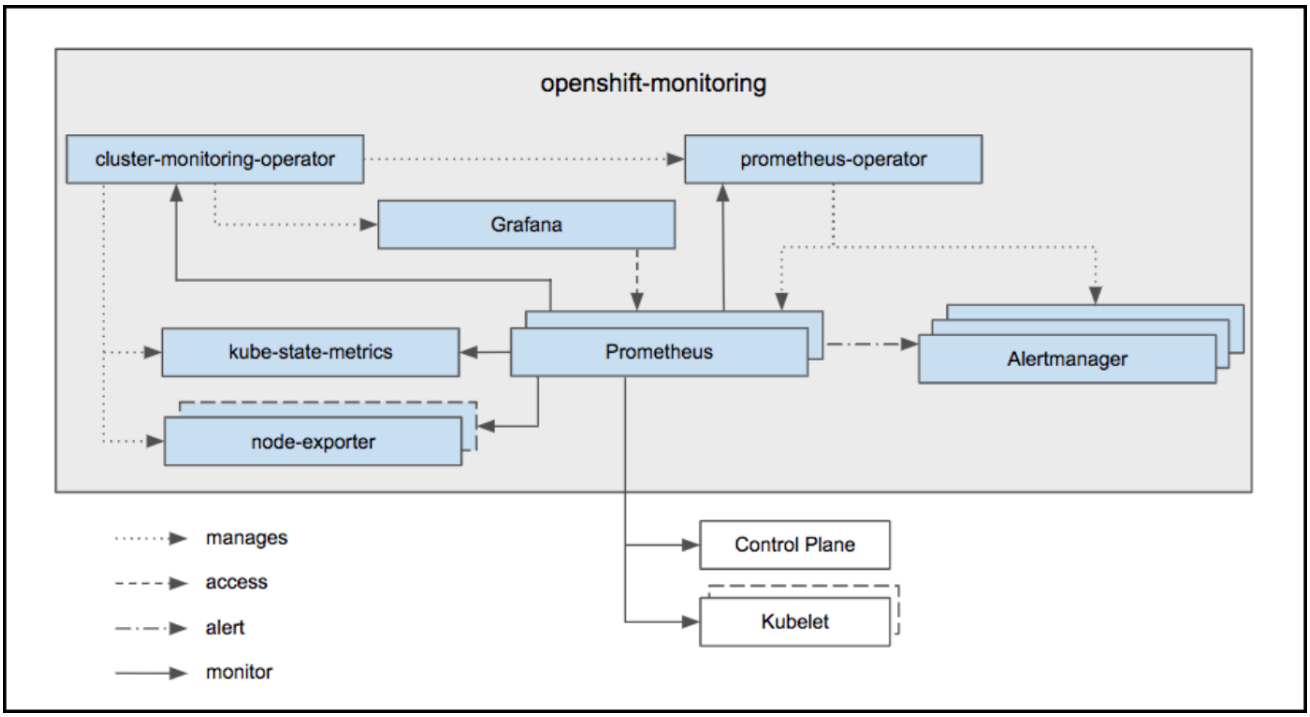
```
$ oc delete postgresql-1-6w2t4
```

PostgreSQL 部署器容器集将尝试扩展新的 **postgresql 容器集**，以取代您删除的容器集。在 **postgresql** 容器集运行后，**cfme** 容器集将停止阻止并开始应用初始化。

第 5 章 PROMETHEUS CLUSTER MONITORING

5.1. 概述

OpenShift Container Platform 附带一个预先配置和自我更新的监控堆栈，它基于 Prometheus 开源项目及其更广泛的生态系统。它提供对集群组件的监控，并附带一组警报，以便立即通知集群管理员任何出现的问题，以及一组 Grafana 仪表板。



上图中突出显示，监控堆栈的核心是 OpenShift Container Platform Cluster Monitoring Operator(CMO)，它监视部署的监控组件和资源，并确保它们始终保持最新状态。

Prometheus Operator (PO) 可以创建、配置和管理 Prometheus 和 Alertmanager 实例。还能根据熟悉的 Kubernetes 标签查询来自动生成监控目标配置。

除了 Prometheus 和 Alertmanager 外，OpenShift Container Platform 监控还包括 node-exporter 和 kube-state-metrics。node-exporter 是部署在每个节点上的代理，用于收集有关它的指标。kube-state-metrics 导出器代理将 Kubernetes 对象转换为 Prometheus 可使用的指标。

作为集群监控的一部分监控的目标有：

- Prometheus 本身
- Prometheus-Operator
- cluster-monitoring-operator
- Alertmanager 集群实例
- Kubernetes apiserver
- kubelet（kubelet 为每个容器指标嵌入 cAdvisor）
- kube-controllers
- kube-state-metrics

- node-exporter
- etcd（如果启用了 etcd 监控）

所有这些组件都会自动更新。

如需有关 OpenShift Container Platform Cluster Monitoring Operator 的更多信息，请参阅 Cluster Monitoring Operator GitHub 项目。

注意

为了能够提供具有保证兼容性的更新，OpenShift Container Platform 监控堆栈的可配置性仅限于明确可用的选项。

5.2. 配置 OPENSHIFT CONTAINER PLATFORM 集群监控

OpenShift Container Platform Ansible **openshift_cluster_monitoring_operator** 角色使用清单文件中的变量配置和部署 Cluster Monitoring Operator。

表 5.1. Ansible 变量

变量	描述
openshift_cluster_monitoring_operator_install	如果为 **true**，部署 Cluster Monitoring Operator。否则，取消部署。默认将此变量设置为 **true**。
openshift_cluster_monitoring_operator_prometheus_storage_capacity	每个 Prometheus 实例的持久性卷声明大小。只有在 **openshift_cluster_monitoring_operator_prometheus_storage_enabled** 被设置为 **true** 时才会应用这个变量。默认值为 **50Gi**。
openshift_cluster_monitoring_operator_alertmanager_storage_capacity	每个 Alertmanager 实例的持久性卷声明大小。只有在 **openshift_cluster_monitoring_operator_alertmanager_storage_enabled** 被设置为 **true** 时才会应用这个变量。默认值为 **2Gi**。
openshift_cluster_monitoring_operator_node_selector	设置为所需的现有 节点选择器，以确保 pod 放置到具有特定标签的节点上。默认为 **node-role.kubernetes.io/infra=true**。
openshift_cluster_monitoring_operator_alertmanager_config	配置 Alertmanager。
openshift_cluster_monitoring_operator_prometheus_storage_enabled	启用 Prometheus 时间序列数据的持久性存储。默认将此变量设置为 **false**。
openshift_cluster_monitoring_operator_alertmanager_storage_enabled	启用 Alertmanager 通知和静默的持久性存储。默认将此变量设置为 **false**。

变量	描述
openshift_cluster_monitoring_operator_prometheus_storage_class_name	如果启用了 **openshift_cluster_monitoring_operator_prometheus_storage_enabled** 选项，请设置特定的 StorageClass 以确保 pod 被配置为使用该 **storageclass** 的 **PVC**。默认值为 **none**，它应用默认存储类名称。
openshift_cluster_monitoring_operator_alertmanager_storage_class_name	如果启用了 **openshift_cluster_monitoring_operator_alertmanager_storage_enabled** 选项，请设置特定的 StorageClass 以确保 pod 被配置为使用该 **storageclass** 的 **PVC**。默认值为 **none**，它应用默认存储类名称。

5.2.1. 监控先决条件

监控堆栈会带来额外的资源需求。详情请查看计算资源建议。

5.2.2. 安装监控堆栈

监控堆栈默认安装有 OpenShift Container Platform。您可以防止安装它。为此，请在 Ansible 清单文件中将此变量设置为 **false** ：

openshift_cluster_monitoring_operator_install

您可以通过运行以下命令完成：

```
$ ansible-playbook [-i </path/to/inventory>] <OPENSHIFT_ANSIBLE_DIR>/playbooks/openshift-monitoring/config.yml \
  -e openshift_cluster_monitoring_operator_install=False
```

Ansible 目录的常用路径是 **/usr/share/ansible/openshift-ansible/**。在本例中，配置文件的路径为 **/usr/share/ansible/openshift-ansible/playbooks/openshift-monitoring/config.yml**。

5.2.3. 持久性存储

使用持久性存储运行集群监控意味着您的指标存储在 持久性卷中，并可在 Pod 重新启动或重新创建后保留。如果您需要预防指标或警报数据丢失，这是理想方案。在生产环境中，强烈建议您使用块存储技术配置永久存储。

5.2.3.1. 启用持久性存储

默认情况下，对于 Prometheus 时间序列数据和 Alertmanager 通知和静默禁用持久性存储。您可以将集群配置为永久存储其中任何一个或两者。

- 要启用 Prometheus 时间序列数据的持久性存储，请在 Ansible 清单文件中将此变量设置为 **true** ：

  openshift_cluster_monitoring_operator_prometheus_storage_enabled

- 要启用 Alertmanager 通知和静默的持久性存储，请在 Ansible 清单文件中将此变量设置为 **true** ：
  openshift_cluster_monitoring_operator_alertmanager_storage_enabled

5.2.3.2. 确定需要多少存储

您需要的存储量取决于 Pod 的数目。管理员负责投入足够的存储来确保磁盘未满。如需有关持久性存储的系统要求的信息，请参阅 Cluster Monitoring Operator 的容量规划。

5.2.3.3. 设置持久性存储大小

要为 Prometheus 和 Alertmanager 指定持久性卷声明的大小，请更改这些 Ansible 变量：

- **openshift_cluster_monitoring_operator_prometheus_storage_capacity** (default:50Gi)
- **openshift_cluster_monitoring_operator_alertmanager_storage_capacity** (default:2Gi)

只有将其对应的 **storage_enabled** 变量设置为 **true** 时，每个变量才会生效。

5.2.3.4. 分配充足的持久性卷

除非使用动态置备的存储，否则您需要确保 PVC 准备好声明持久性卷(PV)，每个副本一个 PV。Prometheus 有两个副本，Alertmanager 有三个副本，它们相当于五个 PV。

5.2.3.5. 启用动态置备的存储

您可以使用动态置备的存储，而不是静态置备的存储。详情请参阅动态卷置备。

要为 Prometheus 和 Alertmanager 启用动态存储，请在 Ansible 清单文件中将以下参数设置为 **true** ：

- **openshift_cluster_monitoring_operator_prometheus_storage_enabled** (Default: false)
- **openshift_cluster_monitoring_operator_alertmanager_storage_enabled** (Default: false)

启用动态存储后，您还可以在 Ansible 清单文件中为每个组件设置**存储类** ：

- **openshift_cluster_monitoring_operator_prometheus_storage_class_name** (default: "")
- **openshift_cluster_monitoring_operator_alertmanager_storage_class_name** (default: "")

只有将其对应的 **storage_enabled** 变量设置为 **true** 时，每个变量才会生效。

5.2.4. 支持的配置

配置 OpenShift Container Platform Monitoring 的支持方法是使用本指南中介绍的选项进行配置。除了这些明确的配置选项外，还可以将其他配置注入到堆栈中。但不受支持，因为配置范例可能会在 Prometheus 发行版本间有所变化，只有在控制了所有可能的配置时，才能安全地应对这样的情况。

明确不支持的情形包括：

- 在 **openshift-monitoring** 命名空间中创建额外的 **ServiceMonitor** 对象，从而扩展集群监控 Prometheus 实例提取的目标。这可能导致无法考量的冲突和负载差异，因此 Prometheus 设置可能会不稳定。

- 创建额外的 **ConfigMap** 对象，导致集群监控 Prometheus 实例包含额外的警报和记录规则。请注意，如果应用此行为，这个行为会导致行为中断，因为 Prometheus 2.0 将附带新的规则文件语法。

5.3. 配置 ALERTMANAGER

Alertmanager 管理传入的警报；这包括银级、禁止、聚合和通过电子邮件、PagerDuty 和 HipChat 等方法发送通知。

OpenShift Container Platform Monitoring Alertmanager 集群的默认配置是：

```
global:
  resolve_timeout: 5m
route:
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: default
  routes:
  - match:
      alertname: DeadMansSwitch
    repeat_interval: 5m
    receiver: deadmansswitch
receivers:
- name: default
- name: deadmansswitch
```

可以使用 **openshift_cluster_monitoring_operator** 角色中的 Ansible 变量 **openshift_cluster_monitoring_operator_alertmanager_config** 覆盖此配置。

以下示例将 PagerDuty 配置为通知。如需了解如何检索 **service_key**，请参阅 Alertmanager 的 PagerDuty 文档。

```
openshift_cluster_monitoring_operator_alertmanager_config: |+
  global:
    resolve_timeout: 5m
  route:
    group_wait: 30s
    group_interval: 5m
    repeat_interval: 12h
    receiver: default
    routes:
    - match:
        alertname: DeadMansSwitch
      repeat_interval: 5m
      receiver: deadmansswitch
    - match:
        service: example-app
      routes:
      - match:
          severity: critical
        receiver: team-frontend-page
  receivers:
  - name: default
  - name: deadmansswitch
```

```
- name: team-frontend-page
  pagerduty_configs:
  - service_key: "<key>"
```

子路由仅匹配严重性为 **critical** 的警报，并使用名为 **team-frontend-page** 的接收器发送它们。如名称所示，对于关键警报，应传出某人。参阅 Alertmanager 配置来配置通过不同警报接收器发送警报。

5.3.1. 死人开关

OpenShift Container Platform Monitoring 附带了一个 *死人开关*，用于确保监控基础架构的可用性。

死人开关是始终触发的简单 Prometheus 警报规则。Alertmanager 持续向支持此功能的通知提供程序发送死人交换机的通知。这也可确保 Alertmanager 和通知提供程序之间的通信正常工作。

PagerDuty 支持这种机制，以在监控系统本身停机时发出警报。如需更多信息，请参阅下面的死人开关 PagerDuty。

5.3.2. 分组警报

在警报针对 Alertmanager 触发后，必须将其配置为了解如何在逻辑上分组它们。

在本例中，添加了一个新的路由来反映 **frontend** 团队的警报路由。

流程

1. 添加新路由。可以在原始路由下添加多个路由，通常用于定义通知的接收方。以下示例使用匹配器来确保只使用来自服务 **example-app** 的警报：

```
global:
  resolve_timeout: 5m
route:
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: default
  routes:
  - match:
      alertname: DeadMansSwitch
    repeat_interval: 5m
    receiver: deadmansswitch
  - match:
      service: example-app
    routes:
    - match:
        severity: critical
      receiver: team-frontend-page
receivers:
- name: default
- name: deadmansswitch
```

   子路由仅匹配严重性为 **critical** 的警报，并使用名为 **team-frontend-page** 的接收器发送它们。如名称所示，对于关键警报，应传出某人。

5.3.3. 死人开关 PagerDuty

PagerDuty 通过名为 死人开关 的集成来支持此机制。只需将 **PagerDuty** 配置添加到默认的 **deadmanswitch 接收器**。使用上述流程添加此配置。

如果死人开关警报静默 15 分钟，将死人开关配置为页面运算符。使用默认的 Alertmanager 配置时，死人开关警报每五分钟重复一次。如果死人开关在 15 分钟后触发，这表明通知失败至少两次。

了解如何 为 PagerDuty 配置死人开关。

5.3.4. 警报规则

OpenShift Container Platform Cluster Monitoring 附带了以下默认配置的警报规则。目前无法添加自定义警报规则。

有些警报规则的名称相同。这是有意设计的。它们会警告同一事件，它们具有不同的阈值，或严重性不同。在禁止规则中，触发较高的严重性时会禁止较低严重性。

有关警报规则的详情，请查看配置文件。

警报	重要性	描述
ClusterMonitoringOperatorErrors	**critical**	Cluster Monitoring Operator 会出现 *X*% 错误。
AlertmanagerDown	**critical**	Alertmanager 已从 Prometheus 目标发现中消失。
ClusterMonitoringOperatorDown	**critical**	ClusterMonitoringOperator 已从 Prometheus 目标发现中消失。
KubeAPIDown	**critical**	KubeAPI 已从 Prometheus 目标发现中消失。
KubeControllerManagerDown	**critical**	kubecontrollermanager 已从 Prometheus 目标发现中消失。
KubeSchedulerDown	**critical**	kubescheduler 已从 Prometheus 目标发现中消失。
KubeStateMetricsDown	**critical**	kubeStateMetrics 已从 Prometheus 目标发现中消失。
KubeletDown	**critical**	kubelet 已从 Prometheus 目标发现中消失。
NodeExporterDown	**critical**	NodeExporter 已从 Prometheus 目标发现中消失。
PrometheusDown	**critical**	Prometheus 已从 Prometheus 目标发现中消失。

警报	重要性	描述
PrometheusOperatorDown	**critical**	PrometheusOperator 已从 Prometheus 目标发现中消失。
KubePodCrashLooping	**critical**	*Namespace/Pod* (*Container*) 重启 *times* / second
KubePodNotReady	**critical**	*Namespace/Pod* 未就绪。
KubeDeploymentGenerationMismatch	**critical**	部署 *Namespace/Deployment* 生成不匹配
KubeDeploymentReplicasMismatch	**critical**	部署 *Namespace/Deployment* 副本不匹配
KubeStatefulSetReplicasMismatch	**critical**	StatefulSet *Namespace/StatefulSet* 副本不匹配
KubeStatefulSetGenerationMismatch	**critical**	StatefulSet *Namespace/StatefulSet* 生成不匹配
KubeDaemonSetRolloutStuck	**critical**	只有调度并准备好用于守护进程设置 *Namespace/DaemonSet* 的所需 pod 的 *X*%
KubeDaemonSetNotScheduled	**warning**	没有调度 daemonset *Namespace/DaemonSet* 的 pod。
KubeDaemonSetMisScheduled	**warning**	许多 daemonset *Namespace/DaemonSet* 的 pod 在不应该运行的位置运行。
KubeCronJobRunning	**warning**	CronJob *Namespace/CronJob* 需要 1h 以上才能完成。
KubeJobCompletion	**warning**	Job *Namespaces/Job* 需要超过 1h 的时间才能完成。
KubeJobFailed	**warning**	Job *Namespaces/Job* 无法完成。
KubeCPUOvercommit	**warning**	Pod 上过量使用的 CPU 资源请求无法容忍节点失败。
KubeMemOvercommit	**warning**	Pod 上过量使用的内存资源请求，无法容忍节点失败。

警报	重要性	描述
KubeCPUOvercommit	**warning**	命名空间上过量使用的 CPU 资源请求配额。
KubeMemOvercommit	**warning**	命名空间上过量使用的内存资源请求配额。
alerKubeQuotaExceeded	**warning**	命名空间 *Namespace* 中的 *X*% 的*资源*已使用。
KubePersistentVolumeUsageCritical	**critical**	命名空间 *Namespace* 中的 *PersistentVolumeClaim* 声明的持久性卷有 *X*% free。
KubePersistentVolumeFullInFourDays	**critical**	根据最近的抽样，命名空间 *Namespace* 中的 *PersistentVolumeClaim* 声明的持久性卷应该在四天内填满。*X* 字节当前可用。
KubeNodeNotReady	**warning**	*节点* 已就绪一小时以上
KubeVersionMismatch	**warning**	运行 *X* 种不同版本的 Kubernetes 组件。
KubeClientErrors	**warning**	Kubernetes API 服务器客户端的 '*Job/Instance*' 正在遇到 *X*% 错误。
KubeClientErrors	**warning**	Kubernetes API 服务器客户端的 '*Job/Instance*' 正在遇到 *X* 错误/sec'。
KubeletTooManyPods	**warning**	kubelet *实例*正在运行 *X* pod，接近 110。
KubeAPILatencyHigh	**warning**	API 服务器具有 99% 的 *Verb 资源* 延迟 *X* 秒。
KubeAPILatencyHigh	**critical**	API 服务器具有 99% 的 *Verb 资源* 延迟 *X* 秒。
KubeAPIErrorsHigh	**critical**	API 服务器针对 *X*% 的请求出错。
KubeAPIErrorsHigh	**warning**	API 服务器针对 *X*% 的请求出错。
KubeClientCertificateExpiration	**warning**	Kubernetes API 证书将在不到 7 天后过期。

警报	重要性	描述
KubeClientCertificateExpiration	**critical**	Kubernetes API 证书将在不到 1 天后过期。
AlertmanagerConfigInconsistent	**critical**	Summary：配置不同步.描述：Alertmanager 集群 ***服务*** 的实例配置不同步。
AlertmanagerFailedReload	**warning**	Summary：Alertmanager 的配置重新加载失败。描述：重新加载 Alertmanager 的配置对于 *Namespace/Pod* 失败。
TargetDown	**warning**	Summary：目标已停机。描述：*X*% 的*作业* 目标为 down。
DeadMansSwitch	**none**	Summary：通知 DeadMansSwitch.描述：这是一个 DeadMansSwitch，可确保整个 Alerting 管道正常工作。
NodeDiskRunningFull	**warning**	node-exporter *Namespace/Pod* 的设备 *设备*在接下来 24 小时内完全运行。
NodeDiskRunningFull	**critical**	node-exporter *Namespace/Pod* 的设备 *设备*在接下来 2 小时内完全运行。
PrometheusConfigReloadFailed	**warning**	Summary：重新载入 Prometheus 配置失败。描述：为 *Namespace/Pod* 重新载入 Prometheus 配置失败
PrometheusNotificationQueueRunningFull	**warning**	Summary：Prometheus 的警报通知队列已满运行。描述：Prometheus 的警报通知队列已完全针对 *Namespace/Pod* 运行
PrometheusErrorSendingAlerts	**warning**	Summary：从 Prometheus 发送警报时出错。描述：将警报从 Prometheus *Namespace/Pod* 发送到 *Alertmanager* 时出错
PrometheusErrorSendingAlerts	**critical**	Summary：从 Prometheus 发送警报时出错。描述：将警报从 Prometheus *Namespace/Pod* 发送到 *Alertmanager* 时出错

警报	重要性	描述
PrometheusNotConnectedToAlertmanagers	**warning**	Summary：Prometheus 没有连接到任何 Alertmanager。描述：Prometheus *Namespace/Pod* 没有连接到任何 Alertmanager
PrometheusTSDBReloadsFailing	**warning**	Summary：Prometheus 在从磁盘重新载入数据块时遇到问题。描述：在过去四个小时内，*实例*中的*作业*有 *X* 重新加载失败。
PrometheusTSDBCompactionsFailing	**warning**	Summary：Prometheus 在压缩示例块时遇到问题。描述：*实例*的*作业*过去四小时内出现 *X* 紧凑故障。
PrometheusTSDBWALCorruptions	**warning**	Summary：Prometheus write-ahead 日志已被损坏。描述：*Instance* 中的 *作业* 具有损坏的 write-ahead 日志(WAL)。
PrometheusNotIngestingSamples	**warning**	Summary：Prometheus 不捕获示例。描述：Prometheus *Namespace/Pod* 不嵌套示例。
PrometheusTargetScrapesDuplicate	**warning**	Summary：Prometheus 有许多示例被拒绝。描述：*Namespace/Pod* 因为时间戳重复但不同的值而拒绝多个示例
EtcdInsufficientMembers	**critical**	Etcd 集群 "*Job*": insufficient members (*X*).
EtcdNoLeader	**critical**	Etcd 集群 "*Job*": member *Instance* 没有 leader。
EtcdHighNumberOfLeaderChanges	**warning**	Etcd 集群 "*Job*": 实例 *Instance* 在过去一小时内看到 *X* leader 改变。
EtcdHighNumberOfFailedGRPCRequests	**warning**	Etcd 集群 "*Job*":*X*% 的 *GRPC_Method* 请求在 etcd 实例 *Instance* 上失败。
EtcdHighNumberOfFailedGRPCRequests	**critical**	Etcd 集群 "*Job*":*X*% 的 *GRPC_Method* 请求在 etcd 实例 *Instance* 上失败。
EtcdGRPCRequestsSlow	**critical**	Etcd 集群 "*Job*": 到 *GRPC_Method* 的 gRPC 请求在 *X_s on etcd instance _Instance*。

警报	重要性	描述
EtcdMemberCommunicationSlow	**warning**	Etcd 集群 "*Job*": 成员与 *To* 通信正在 *X_s on etcd instance _Instance*。
EtcdHighNumberOfFailedProposals	**warning**	Etcd 集群 "*Job*":*X* 提议在 etcd 实例 *Instance* 的最后一小时内失败。
EtcdHighFsyncDurations	**warning**	Etcd 集群 "*Job*":99th percentile fync durations 是 *X_s on etcd instance _Instance*。
EtcdHighCommitDurations	**warning**	Etcd 集群 "*Job*":99 percentile 的提交持续时间为 *X_s on etcd instance _Instance*。
FdExhaustionClose	**warning**	*Job* instance *Instance* 很快会耗尽其文件描述符
FdExhaustionClose	**critical**	*Job* instance *Instance* 很快会耗尽其文件描述符

5.4. 配置 ETCD 监控

如果 **etcd** 服务没有正确运行，则整个 OpenShift Container Platform 集群的成功操作将处于危险之中。因此，最好为 **etcd** 配置监控。

按照以下步骤配置 **etcd** 监控 ：

流程

1. 验证监控堆栈是否正在运行：

```
$ oc -n openshift-monitoring get pods
NAME                                           READY     STATUS              RESTARTS   AGE
alertmanager-main-0                            3/3       Running             0          34m
alertmanager-main-1                            3/3       Running             0          33m
alertmanager-main-2                            3/3       Running             0          33m
cluster-monitoring-operator-67b8797d79-sphxj   1/1       Running             0          36m
grafana-c66997f-pxrf7                          2/2       Running             0          37s
kube-state-metrics-7449d589bc-rt4mq            3/3       Running             0          33m
node-exporter-5tt4f                            2/2       Running             0          33m
node-exporter-b2mrp                            2/2       Running             0          33m
node-exporter-fd52p                            2/2       Running             0          33m
node-exporter-hfqgv                            2/2       Running             0          33m
prometheus-k8s-0                               4/4       Running             1          35m
prometheus-k8s-1                               0/4       ContainerCreating   0          21s
prometheus-operator-6c9fddd47f-9jfgk           1/1       Running             0          36m
```

2. 打开集群监控堆栈的配置文件：

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

3. 在 **config.yaml: |+** 下，添加 **etcd** 部分。

   a. 如果在 master 节点上运行静态 pod 的 **etcd**，您可以使用选择器指定 **etcd** 节点：

```
...
data:
  config.yaml: |+
    ...
    etcd:
      targets:
        selector:
          openshift.io/component: etcd
          openshift.io/control-plane: "true"
```

   b. 如果在单独的主机上运行 **etcd**，则需要使用 IP 地址指定节点：

```
...
data:
  config.yaml: |+
    ...
    etcd:
      targets:
       ips:
       - "127.0.0.1"
       - "127.0.0.2"
       - "127.0.0.3"
```

   如果 **etcd** 节点的 IP 地址有变化，您必须更新此列表。

4. 验证 **etcd** 服务监控器现在是否正在运行：

```
$ oc -n openshift-monitoring get servicemonitor
NAME                 AGE
alertmanager         35m
etcd                 1m (1)
kube-apiserver       36m
kube-controllers     36m
kube-state-metrics   34m
kubelet              36m
node-exporter        34m
prometheus           36m
prometheus-operator  37m
```

   (1) **etcd** 服务监控器。

   etcd 服务监控器最多可能需要一分钟才能启动。

5. 现在，您可以进入 Web 界面来查看有关 **etcd** 监控状态的更多信息。

   a. 要获取 URL，请运行：

```
$ oc -n openshift-monitoring get routes
```

```
NAME                HOST/PORT                                                                    PATH
SERVICES            PORT      TERMINATION   WILDCARD
...
prometheus-k8s      prometheus-k8s-openshift-monitoring.apps.msvistun.origin-
gce.dev.openshift.com                  prometheus-k8s      web       reencrypt     None
```

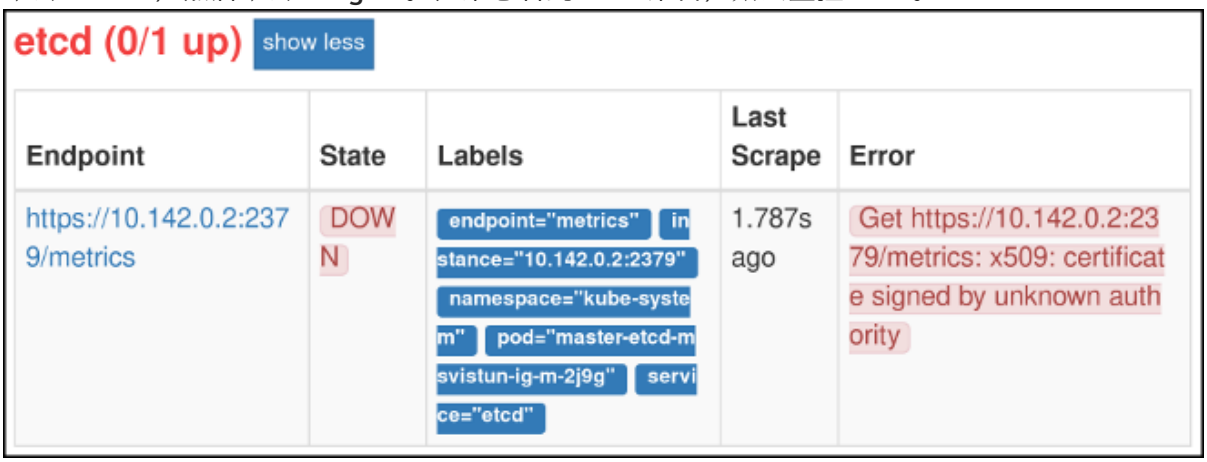
b. 使用 **https**，导航到为 **prometheus-k8s** 列出的 URL。登录。

6. 确保该用户属于 **cluster-monitoring-view** 角色。此角色提供查看集群监控 UI 的访问权限。例如，要将用户 **developer** 添加到 **cluster-monitoring-view** 角色中，请运行：

```
$ oc adm policy add-cluster-role-to-user cluster-monitoring-view developer
```

7. 在 Web 界面中，以属于 **cluster-monitoring-view** 角色的用户身份登录。

8. 单击 **Status**，然后单击 **Targets**。如果您看到 **etcd** 条目，则会监控 **etcd**。



9. 虽然 **etcd** 被监控，但 Prometheus 还无法通过 **etcd** 进行身份验证，因此无法收集指标数据。针对 **etcd** 配置 Prometheus 身份验证：

a. 将 **/etc/etcd/ca/ca.crt** 和 **/etc/etcd/ca/ca.key** 凭证文件从 master 节点复制到本地机器：

```
$ ssh -i gcp-dev/ssh-privatekey cloud-user@35.237.54.213
```

b. 创建包含以下内容的 **openssl.cnf** 文件：

```
[ req ]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[ req_distinguished_name ]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, keyEncipherment, digitalSignature
extendedKeyUsage=serverAuth, clientAuth
```

c. 生成 **etcd.key** 私钥文件：

```
$ openssl genrsa -out etcd.key 2048
```

d. 生成 **etcd.csr** 证书签名请求文件：

```
$ openssl req -new -key etcd.key -out etcd.csr -subj "/CN=etcd" -config openssl.cnf
```

e. 生成 **etcd.crt** 证书文件：

```
$ openssl x509 -req -in etcd.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out etcd.crt -
days 365 -extensions v3_req -extfile openssl.cnf
```

f. 将凭证置于 OpenShift Container Platform 使用的格式：

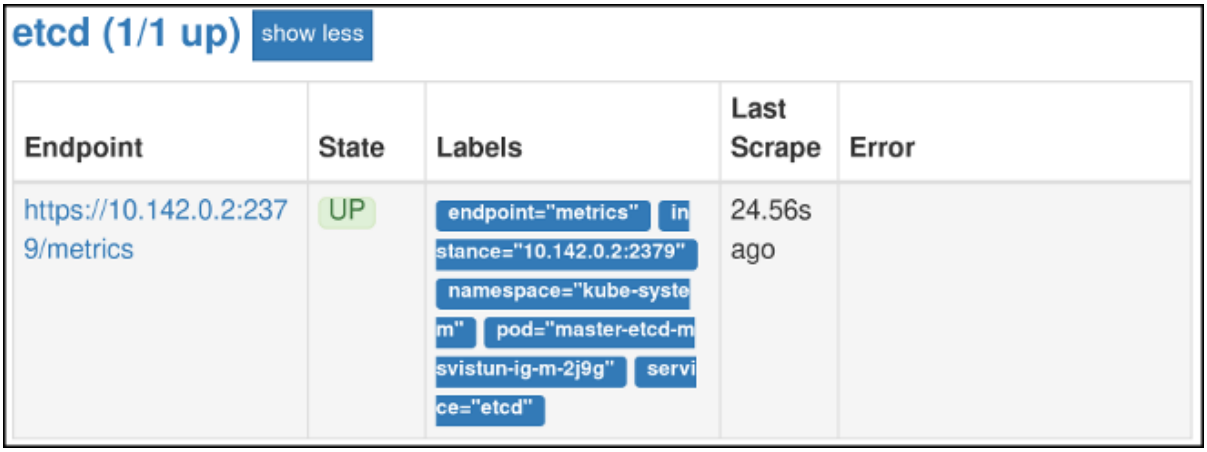
```
$ cat <<-EOF > etcd-cert-secret.yaml
apiVersion: v1
data:
  etcd-client-ca.crt: "$(cat ca.crt | base64 --wrap=0)"
  etcd-client.crt: "$(cat etcd.crt | base64 --wrap=0)"
  etcd-client.key: "$(cat etcd.key | base64 --wrap=0)"
kind: Secret
metadata:
  name: kube-etcd-client-certs
  namespace: openshift-monitoring
type: Opaque
EOF
```

这会创建 *etcd-cert-secret.yaml* 文件

g. 将凭证文件应用到集群：

```
$ oc apply -f etcd-cert-secret.yaml
```

10. 现在您已配置了身份验证，请再次访问 Web 界面的 **Targets** 页面。验证 **etcd** 现在是否正确监控。可能需要几分钟后更改才会生效。



11. 如果您希望在更新 OpenShift Container Platform 时自动更新 **etcd** 监控，请将 Ansible 清单文件中的这个变量设置为 **true** ：

```
openshift_cluster_monitoring_operator_etcd_enabled=true
```

如果您在单独的主机上运行 **etcd**，请按照 IP 地址使用此 Ansible 变量指定节点：

```
openshift_cluster_monitoring_operator_etcd_hosts=[<address1>, <address2>, ...]
```

如果 **etcd** 节点的 IP 地址改变，您必须更新此列表。

5.5. 访问 PROMETHEUS、ALERTMANAGER 和 GRAFANA。

OpenShift Container Platform Monitoring 附带了一个用于集群监控的 Prometheus 实例和一个中央 Alertmanager 集群。除了 Prometheus 和 Alertmanager 外，OpenShift Container Platform Monitoring 还包含一个 Grafana 实例，以及用于集群监控故障排除的预构建仪表板。由监控堆栈提供的 Grafana 实例及其仪表板是只读的。

获取用于访问 Prometheus、Alertmanager 和 Grafana Web UI 的地址：

流程

1. 运行以下命令：

```
$ oc -n openshift-monitoring get routes
NAME                HOST/PORT
alertmanager-main   alertmanager-main-openshift-monitoring.apps._url_.openshift.com
grafana             grafana-openshift-monitoring.apps._url_.openshift.com
prometheus-k8s      prometheus-k8s-openshift-monitoring.apps._url_.openshift.com
```

   确保将 **https://** 添加到这些地址。您无法使用未加密的连接访问 Web UI。

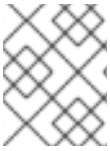
2. 根据 OpenShift Container Platform 身份进行身份验证，并使用与 OpenShift Container Platform 其他位置相同的凭证或验证方式。您必须使用具有所有命名空间的读取访问权限的角色，如 **cluster-monitoring-view** 集群角色。

第 6 章 访问并配置 RED HAT REGISTRY

6.1. 启用身份验证的 RED HAT REGISTRY

Red Hat Container Catalog （**registry.access.redhat.com**)是一个托管的镜像 registry，通过它可以获得所需的容器镜像。OpenShift Container Platform 3.11 Red Hat Container Catalog 从 **registry.access.redhat.com** 移到 **registry.redhat.io**。

新的 registry（**Registry.redhat.io**）需要进行身份验证才能访问OpenShift Container Platform上的镜像及内容。当迁移到新registry后，现有的registry仍将在一段时间内可用。



注意

OpenShift Container Platform从**Registry.redhat.io中**提取（pull）镜像，因此需要配置集群以使用它。

新registry使用标准的OAuth机制进行身份验证：

- **身份验证令牌。**令牌（token）是服务帐户，由管理员生成。系统可以使用它们与容器镜像registry进行身份验证。服务帐户不受用户帐户更改的影响，因此使用令牌进行身份验证是一个可靠且具有弹性的方法。这是生产环境集群中唯一受支持的身份验证选项。
- **Web用户名和密码。**这是用于登录到诸如**access.redhat.com**之类的资源的标准凭据集。虽然可以在OpenShift Container Platform上使用此身份验证方法，但在生产环境部署中不支持此方法。此身份验证方法应该只限于在OpenShift Container Platform之外的独立项目中使用。

您可以在 **docker login** 中使用您的凭证（用户名和密码，或身份验证令牌）来访问新 registry 中的内容。

所有镜像流均指向新的registry。由于新 registry 需要进行身份验证才能访问，因此 OpenShift 命名空间中有一个名为 **imagestreamsecret** 的新机密。

您需要将凭据放在两个位置：

- **OpenShift 命名空间** 。您的凭据必须存在于 OpenShift 命名空间中，以便 OpenShift 命名空间中的镜像流可以导入。
- **您的主机**。您的凭据必须存在于主机上，因为在抓取（pull）镜像时，Kubernetes会使用主机中的凭据。

访问新 registry:

- 验证镜像导入 secret(**imagestreamsecret**)是否位于 OpenShift 命名空间中。该 secret 具有允许您访问新 registry 的凭证。
- 验证所有集群节点都有一个 **/var/lib/origin/.docker/config.json**，可以从 master 中复制，供您访问红帽 registry。

6.1.1. 创建用户帐户

如果您是有权使用红帽产品的红帽客户，则拥有具有适用用户凭证的帐户。这些是您用于登录到红帽客户门户的用户名和密码。

如果您没有帐户，可以通过注册以下选项之一获取免费帐户：

- 红帽开发人员计划.此帐户可让您访问开发人员工具和计划。
- 30 天试用订阅.此帐户为您提供 30 天的试用订阅，可访问选定的红帽软件产品。

6.1.2. 为 Red Hat Registry 创建服务帐户和身份验证令牌

如果您的组织管理共享帐户，则必须创建令牌。管理员可以创建、查看和删除与组织关联的所有令牌。

先决条件

- 用户凭证

流程

要创建令牌以完成 **docker login**，请执行以下操作 ：

1. 导航到 **registry.redhat.io**。
2. 使用您的红帽网络(RHN)用户名和密码登录。
3. 出现提示时接受条款.
   - 如果未立即提示您接受条款，则在继续以下步骤时会提示您。
4. 在 Registry Service Accounts 页面中点 Create Service Account
   a. 为服务帐户提供名称。它将带有一个随机字符串。
   b. 输入描述。
   c. 单击 create。
5. 切回到您的服务帐户。
6. 点您创建的服务帐户。
7. 复制用户名，包括前缀字符串。
8. 复制令牌。

6.1.3. 管理用于安装和升级的 registry 凭证

您还可以在安装过程中使用 Ansible 安装程序管理 registry 凭据。

这将设置以下内容：

- OpenShift 命名空间中的 **imagestreamsecret**。
- 所有节点上的凭据。

当您将 **registry.redhat.io** 的默认值用于 **openshift_examples_registryurl** 或 **oreg_url** 时，Ansible 安装程序将需要凭证。

先决条件

- 用户凭证

- 服务帐户
- 服务帐户令牌

流程

要在安装过程中使用 Ansible 安装程序管理 registry 凭证：

- 在安装或升级过程中，指定安装程序清单中的 **oreg_auth_user** 和 **oreg_auth_password** 变量。

注意

如果您已创建了令牌，请将 **oreg_auth_password** 设置为令牌的值。

需要访问其他经过身份验证的 registry 的集群可以通过设置 **openshift_additional_registry_credentials** 来配置 registry 列表。每个 registry 都需要主机和密码值，您可以通过设置用户来指定用户名。默认情况下，通过尝试检查指定 registry 上的镜像 **openshift3/ose-pod** 来验证指定的凭证。

要指定备用镜像，请执行以下操作：

- 设置 **test_image**。
- 通过将 **test_login** 设置为 False 来禁用凭据验证。

如果 registry 不安全，则将 **tls_verify** 设置为 False。

此列表中的所有凭据都将在 OpenShift 命名空间中创建 **imagestreamsecret**，并部署到所有节点的凭据。

例如：

```
openshift_additional_registry_credentials=
[{'host':'registry.example.com','user':'name','password':'pass1','test_login':'False'},
{'host':'registry2.example.com','password':'token12345','tls_verify':'False','test_image':'mongodb/mongodb'}]
```

6.1.4. 在 Red Hat Registry 中使用服务帐户

在为 Red Hat Registry 创建服务帐户和生成的令牌后，您可以执行其他任务。

注意

本节提供了手动步骤，可以通过提供 *管理 registry Credentials for Installation 和 Upgrade* 部分中概述的清单变量来自动执行这些步骤。

先决条件

- 用户凭证
- 服务帐户
- 服务帐户令牌

流程

在 Registry Service Accounts 页面中点击您的帐户名称。在这里，您可以执行以下任务：

- 在 **Token Information** 选项卡中，您可以查看您的用户名（您提供的带有随机字符串的名称）和密码（令牌）。在此选项卡中，您可以重新生成令牌。
- 在 **OpenShift Secret** 选项卡中，您可以：
  a. 单击选项卡中的链接，以下载该机密。
  b. 将 secret 提交到集群：

```
# oc create -f <account-name>-secret.yml --namespace=openshift
```

  c. 使用 **imagePullSecrets** 字段在 Kubernetes pod 配置中添加对 secret 的引用来更新 Kubernetes 配置，例如：

```
apiVersion: v1
kind: Pod
metadata:
  name: somepod
  namespace: all
  spec:
    containers:
      - name: web
      image: registry.redhat.io/REPONAME

    imagePullSecrets:
      - name: <numerical-string-account-name>-pull-secret
```

- 在 **Docker Login** 选项卡中，您可以运行 **docker login**。例如：

```
# docker login -u='<numerical-string|account-name>'
  -p=<token>
```

成功登录后，将 **~/.docker/config.json** 复制到 **/var/lib/origin/.docker/config.json**，然后重新启动节点。

```
# cp -r ~/.docker /var/lib/origin/
  systemctl restart atomic-openshift-node
```

- 在 **Docker Configuration** 选项卡中，您可以：
  a. 单击选项卡中的链接，下载凭据配置。
  b. 通过将文件放入 Docker 配置目录中，将配置写入到磁盘。这将覆盖现有的凭据。例如：

```
# mv <account-name>-auth.json ~/.docker/config.json
```

第 7 章 MASTER 和节点配置

7.1. 安装后自定义 MASTER 和节点配置

openshift start 命令（用于 master 服务器）和 **Hyperkube** 命令（用于节点服务器）采用一组有限的参数，足以在开发或实验环境中启动服务器。但是，这些参数不足以描述和控制生产环境中所需的整套配置和安全选项。

您必须在 master 配置文件(*/etc/origin/master/master-config.yaml*) 和 节点配置映射中提供这些选项。这些文件定义了选项，包括覆盖默认插件、连接到 etcd、自动创建服务帐户、构建镜像名称、自定义项目请求、配置卷插件等。

本节介绍了自定义 OpenShift Container Platform master 和节点主机的可用选项，并演示了如何在安装后更改配置。

在不使用默认值的情况下，将完全指定这些文件。因此，一个空值表示您要使用该参数的空值启动。这样可以轻松地说明您配置的确切原因，但也难以记住要指定的所有选项。为此，可以使用 **--write-config** 选项创建配置文件，然后与 **--config** 选项一起使用。

7.2. 安装依赖项

生产环境应该使用标准 集群安装 过程进行安装。在生产环境中，最好将 多个 master 用于 高可用性 (HA)。建议使用三个 master 的集群架构，并且推荐使用 HAproxy。

小心

如果在 *master 主机上* 安装 etcd，则必须将集群配置为使用至少三个 master，因为 etcd 无法决定哪个是权威的。成功运行两个 master 的唯一方法是在 master 以外的主机上安装 etcd。

7.3. 配置主控机和节点

配置 master 和节点配置文件的方法必须与用于安装 OpenShift Container Platform 集群的方法匹配。如果您遵循标准 集群安装 过程，请在 Ansible 清单文件中进行配置更改。

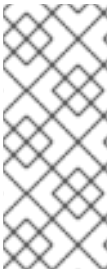
7.4. 使用 ANSIBLE 进行配置更改

对于本节，假设您熟悉 Ansible。

只有一部分可用的主机配置选项会 公开给 Ansible。安装 OpenShift Container Platform 后，Ansible 会使用一些替换的值创建一个清单文件。修改此清单文件并重新运行 Ansible 安装程序 playbook 是为了自定义 OpenShift Container Platform 集群。

虽然 OpenShift Container Platform 支持使用 Ansible playbook 和清单文件进行集群安装，但您也可以使用其他管理工具，如 Puppet、Chef 或 Salt。

使用案例：将集群配置为使用 HTPasswd 身份验证



注意

- 此用例假定您已为 playbook 中引用的所有节点设置了 SSH 密钥。
- **htpasswd** 工具程序位于 **httpd-tools** 软件包中：

```
# yum install httpd-tools
```

修改 Ansible 清单并进行配置更改：

1. 打开 *./hosts* 清单文件。
2. 在文件的 **[OSEv3:vars]** 部分添加以下新变量：

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge':
'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
#openshift_master_htpasswd_users={'<name>': '<hashed-password>', '<name>': '<hashed-
password>'}
# or
#openshift_master_htpasswd_file=/etc/origin/master/htpasswd
```

对于 HTPasswd 身份验证，**openshift_master_identity_providers** 变量启用身份验证类型。您可以使用 HTPasswd 配置三个不同的身份验证选项：

- 如果主机上已配置了 **/etc/origin/master/htpasswd**，则仅**指定 openshift_master_identity_providers**。
- 指定 **openshift_master_identity_providers** 和 **openshift_master_htpasswd_file** 将本地 htpasswd 文件复制到主机。
- 指定 **openshift_master_identity_providers** 和 **openshift_master_htpasswd_users**，以在主机上生成新的 htpasswd 文件。

由于 OpenShift Container Platform 需要哈希密码来配置 HTPasswd 身份验证，因此您可以使用 **htpasswd** 命令(如以下小结所述)为用户生成哈希密码，或使用用户和相关哈希密码创建平面文件。

以下示例将身份验证方法从默认 **拒绝所有** 设置到 **htpasswd**，并使用指定的文件为 **jsmith** 和 **blob 用户可以生成用户 ID 和密码**。

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge':
'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
openshift_master_htpasswd_users={'jsmith': '$apr1$wIwXkFLI$bAygtKGmPOqaJftB',
'bloblaw': '7IRJ$2ODmeLoxf4I6sUEKfiA$2aDJqLJe'}
# or
#openshift_master_htpasswd_file=/etc/origin/master/htpasswd
```

3. 重新运行 ansible playbook 以使这些修改生效：

```
$ ansible-playbook -b -i ./hosts ~/src/openshift-ansible/playbooks/deploy_cluster.yml
```

playbook 更新配置，并重启 OpenShift Container Platform master 服务以应用更改。

您现在已使用 Ansible 修改了 master 和节点配置文件，但这只是一个简单的用例。从此处您可以看到哪些 master 和 节点配置选项 公开给 Ansible，并自定义您自己的 Ansible 清单。

7.4.1. 使用 htpasswd 命令

要将 OpenShift Container Platform 集群配置为使用 HTPasswd 身份验证，您至少需要一个带有哈希密码的用户才能包含在 清单文件中。

您可以：

- 生成用户名和密码直接添加到 *./hosts* 清单文件。
- 创建一个平面文件，将凭据传递到 *./hosts* 清单文件。

创建用户和散列密码：

1. 运行以下命令来添加指定用户：

```
$ htpasswd -n <user_name>
```

注意

您可以包含 **-b** 选项，用于在命令行中提供密码：

```
$ htpasswd -nb <user_name> <password>
```

2. 输入并确认用户的明文密码。
例如：

```
$ htpasswd -n myuser
New password:
Re-type new password:
myuser:$apr1$vdW.cI3j$WSKIOzUPs6Q
```

该命令将生成散列版本的密码。

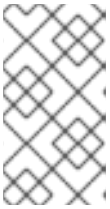
然后您可以在配置 HTPasswd 身份验证时使用哈希密码。hashed 密码是字符串，位于 **:** 后面。在上例中，请输入：

```
openshift_master_htpasswd_users={'myuser': '$apr1$wIwXkFLI$bAygtISk2eKGmqaJftB'}
```

创建带有用户名和散列密码的平面文件：

1. 执行以下命令：

```
$ htpasswd -c /etc/origin/master/htpasswd <user_name>
```



注意

您可以包含 **-b** 选项，用于在命令行中提供密码：

```
$ htpasswd -c -b <user_name> <password>
```

2. 输入并确认用户的明文密码。
例如：

```
htpasswd -c /etc/origin/master/htpasswd user1
New password:
Re-type new password:
Adding password for user user1
```

命令会生成一个文件，其中包含用户名以及用户密码的散列版本。

然后，您可以在配置 HTPasswd 身份验证时使用密码文件。

注意

如需有关 **htpasswd** 命令的更多信息，请参阅 HTPasswd 身份提供程序。

7.5. 进行手动配置更改

使用案例：配置集群以使用 HTPasswd 身份验证

手动修改配置文件：

1. 打开您要修改的配置文件，本例中为 */etc/origin/master/master-config.yaml* 文件：
2. 将以下新变量添加到文件的 **identityProviders** 小节中：

```
oauthConfig:
  ...
  identityProviders:
  - name: my_htpasswd_provider
    challenge: true
    login: true
    mappingMethod: claim
    provider:
      apiVersion: v1
      kind: HTPasswdPasswordIdentityProvider
      file: /etc/origin/master/htpasswd
```

3. 保存您的更改并关闭该文件。
4. 重启 master 以使更改生效：

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

您现在已手动修改 master 和节点配置文件，但这只是一个简单的用例。在这里，您可以通过进行进一步的修改来查看所有 master 和节点配置选项，并进一步自定义自己的集群。

注意

要修改集群中的节点，请根据需要更新节点配置映射。不要手动编辑 *node-config.yaml* 文件。

7.6. 主配置文件

本节介绍了 *master-config.yaml* 文件中提到的参数。

您可以 创建新的 master 配置文件 来查看已安装 OpenShift Container Platform 版本的有效选项。

重要

每当修改 *master-config.yaml* 文件时，您必须重启 master 才能使更改生效。请参阅 重启 OpenShift Container Platform 服务。

7.6.1. 准入控制配置

表 7.1. 准入控制配置参数

参数名称	描述
AdmissionConfig	包含 准入插件 配置。OpenShift Container Platform 具有可在创建或修改 API 对象时触发的准入控制器插件的可配置列表。此选项允许您覆盖默认插件列表；例如，禁用一些插件、添加其他插件、更改顺序以及指定配置。插件及其配置列表都可从 Ansible 控制。

APIServerArguments	键值对将直接传递给与 API 服务器的命令行参数匹配的 Kube API 服务器。它们不会被迁移，但如果您引用了不存在服务器的值，则不会启动。这些值可能会覆盖 **KubernetesMasterConfig** 中的其他设置，这可能会导致无效的配置。使用带有 **event-ttl** 值的 **APIServerArguments** 将事件存储在 etcd 中。默认值为 **2h**，但可将其设置为 less 以防止内存增长：
	
	`apiServerArguments:`
	`  event-ttl:`
	`  - "15m"`

控制器参数	键值对将直接传递给与控制器管理器的命令行参数匹配的 Kube 控制器管理器。它们不会被迁移，但如果您引用了不存在服务器的值，则不会启动。这些值可能会覆盖 **KubernetesMasterConfig** 中的其他设置，这可能会导致无效的配置。
DefaultAdmissionConfig	用于启用或禁用各种准入插件。当此类型作为 **pluginConfig** 下的 **configuration** 对象以及准入插件支持时，这会导致启用 **默认的** 准入插件。
PluginConfig	允许为每个准入插件指定配置文件。
PluginOrderOverride	master 上将安装的准入插件名称列表。订购非常显著。如果为空，则使用默认插件列表。

参数名称	描述
SchedulerArguments	键值对将直接传递给与调度程序的命令行参数匹配的 Kube 调度程序。它们不会被迁移，但如果您引用了不存在服务器的值，则不会启动。这些值可能会覆盖 **KubernetesMasterConfig** 中的其他设置，这可能会导致无效的配置。

7.6.2. 资产配置

表 7.2. 资产配置参数

参数名称	描述
AssetConfig	如果存在，则资产服务器会根据定义的参数启动。例如： assetConfig: logoutURL: "" masterPublicURL: https://master.ose32.example.com:8443 publicURL: https://master.ose32.example.com:8443/console/ servingInfo: bindAddress: 0.0.0.0:8443 bindNetwork: tcp4 certFile: master.server.crt clientCA: "" keyFile: master.server.key maxRequestsInFlight: 0 requestTimeoutSeconds: 0
corsAllowedOrigins	要使用其他主机名从 web 应用访问 API 服务器，您必须通过将配置字段中的 corsAllowedOrigins 指定 **corsAllowedOrigins** 或在 **openshift start** 中指定 **--cors-allowed-origins** 选项来列入主机名。不会对值进行固定或转义。如需示例用法，请参阅 Web 控制台。
DisabledFeatures	不应启动的功能列表。您可能想将其设置为 **null**。不太可能每个人都希望手动禁用功能，不建议这样做。
扩展	从子上下文下资产服务器文件系统提供服务的文件。
扩展开发	当设置为 **true** 时，告诉资产服务器为每个请求重新载入扩展脚本和样式表，而不是只在启动时使用。它使您可以开发扩展，无需每次更改重启服务器。
ExtensionProperties	key-(string)和 value-（字符串）对，这些对将注入到全局变量 **OPENSHIFT_EXTENSION_PROPERTIES** 下。
ExtensionScripts	Web 控制台加载时，资产服务器文件中的文件路径作为脚本加载。
ExtensionStylesheets	Web 控制台加载时，资产服务器上的文件路径以作为风格的表加载。

参数名称	描述
LoggingPublicURL	用于日志记录的公共端点（可选）。
LogoutURL	在登出 Web 控制台后将 Web 浏览器重定向到的一个可选的绝对 URL。如果未指定，则会显示内置注销页面。
MasterPublicURL	Web 控制台如何访问 OpenShift Container Platform 服务器。
MetricsPublicURL	指标的公共端点（可选）。
PublicURL	资产服务器的 URL。

7.6.3. 认证和授权配置

表 7.3. 认证和授权参数

参数名称	描述
authConfig	包含身份验证和授权配置选项。
AuthenticationCacheSize	指明应缓存多少个验证结果。如果为 0，则使用默认的缓存大小。
AuthorizationCacheTTL	表示应缓存授权结果的时长。它接受有效的持续时间字符串（例如："5m").如果为空，则获得默认超时。如果为 0 (例如"0m"), 禁用缓存。

7.6.4. Controller 配置

表 7.4. 控制器配置参数

参数名称	描述
Controllers	应该启动的控制器列表。如果设置为 **none**，则不会自动启动任何控制器。默认值为 *，它将启动所有控制器。当使用 * 时，您可以通过在其名称前加上 **-** 来排除控制器。此时无法识别其他值。
ControllerLeaseTTL	启用控制器选择，指示 master 在控制器启动并在该值定义的秒数内尝试获取租期。将此值设置为非负值会强制 **pauseControllers=true**。这个值默认为 off（0 或 omitted），控制器选择可以使用 -1 来禁用控制器选择。
PauseControllers	指示 master 不会自动启动控制器，而是等待接收通知到服务器，然后再启动它们。

7.6.5. etcd 配置

表 7.5. etcd 配置参数

参数名称	描述
Address	advertised host:port 用于客户端连接到 etcd。
etcdClientInfo	包含有关如何连接到 etcd 的信息。指定 etcd 是否作为嵌入式或非嵌入运行，以及主机。其余的配置由 Ansible 清单处理。例如： etcdClientInfo: ca: ca.crt certFile: master.etcd-client.crt keyFile: master.etcd-client.key urls: - https://m1.aos.example.com:4001
etcdConfig	如果存在，etcd 会根据定义的参数启动。例如： etcdConfig: address: master.ose32.example.com:4001 peerAddress: master.ose32.example.com:7001 peerServingInfo: bindAddress: 0.0.0.0:7001 certFile: etcd.server.crt clientCA: ca.crt keyFile: etcd.server.key servingInfo: bindAddress: 0.0.0.0:4001 certFile: etcd.server.crt clientCA: ca.crt keyFile: etcd.server.key storageDirectory: /var/lib/origin/openshift.local.etcd
etcdStorageConfig	包含 API 资源如何在 etcd 中存储的信息。这些值只有在 etcd 是集群的后备存储时才相关。
KubernetesStoragePrefix	etcd 中的路径，Kubernetes 资源将在其下作为根（root）。如果更改，这个值将意味着 *etcd* 中的现有对象将不再存在。默认值为 **kubernetes.io**。
KubernetesStorageVersion	*etcd* 中的 Kubernetes 资源应该被序列化为的 API 版本。当从 etcd 读取的集群的所有客户端都有相应的代码才能读取新版本时，这个值**不应该**为高级。
OpenShiftStoragePrefix	OpenShift Container Platform 资源根到的 etcd 中的路径。如果更改，这个值将意味着 etcd 中的现有对象将不再存在。默认值为 **openshift.io**。
OpenShiftStorageVersion	*etcd* 中 OS 资源的 API 版本应序列化为。当从*etcd* 读取的集群的所有客户端都有相应的代码才能读取新版本时，这个值**不应该为**高级。
PeerAddress	到 *etcd* 的对等点连接的广告 host:port。

参数名称	描述
PeerServingInfo	描述如何开始为 *etcd* peer 提供服务。
ServingInfo	描述如何开始服务。例如： servingInfo: bindAddress: 0.0.0.0:8443 bindNetwork: tcp4 certFile: master.server.crt clientCA: ca.crt keyFile: master.server.key maxRequestsInFlight: 500 requestTimeoutSeconds: 3600
StorageDir	*etcd* 存储目录的路径。

7.6.6. 授权配置

表 7.6. 授予配置参数

参数名称	描述
GrantConfig	描述如何处理补贴。
GrantHandlerAuto	自动批准客户端授权请求。
GrantHandlerDeny	自动拒绝客户端授权请求。
GrantHandlerPrompt	提示用户批准新的客户端授权请求。
Method	决定在 OAuth 客户端请求授权时使用的默认策略。只有特定的 OAuth 客户端不提供自己策略，则使用这种方法。有效的授权处理方法有： ● auto：始终批准授权请求，对受信任的客户端很有用 ● prompt： 提示用户输入授权请求的批准，对第三方客户端很有用 ● deny：始终拒绝授权请求，对黑名单的客户端很有用

7.6.7. 镜像配置

表 7.7. 镜像配置参数

参数名称	描述
Format	为系统组件构建的名称格式。

参数名称	描述
Latest	决定是否将从 registry 中拉取 latest 标签。

7.6.8. 镜像策略配置

表 7.8. 镜像策略配置参数

参数名称	描述
DisableScheduledImport	允许禁用镜像的调度后台导入。
MaxImagesBulkImportedPerRepository	控制用户执行 Docker 存储库批量导入时导入的镜像数量。此数字默认为 5，以防止用户意外导入大量镜像。设置 **-1** 代表没有限制。
MaxScheduledImageImportsPerMinute	每分钟将在后台导入的最大调度镜像流数。默认值为 60。
ScheduledImageImportMinimumIntervalSeconds	针对上游存储库检查调度后台导入的镜像流时可达最少的秒数。默认值为 15 分钟。
AllowedRegistriesForImport	限制普通用户可从中导入镜像的 Docker docker。将此列表设置为您信任包含有效 Docker 镜像且希望应用程序能够从中导入的 registry。有权通过 API 创建镜像或 ImageStreamMappings 的用户不受此策略的影响 - 通常只有管理员或系统集成将具有这些权限。
AdditionalTrustedCA	指定到 PEM 编码文件的文件路径，它列出了在镜像流导入期间应被信任的额外证书颁发机构。此文件需要可以被 API 服务器进程访问。根据集群安装的方式，这可能需要将文件挂载到 API 服务器 pod 中。
InternalRegistryHostname	设置默认内部镜像 registry 的主机名。该值必须采用 **hostname[:port]** 格式。为实现向后兼容，用户仍然可以使用 **OPENSHIFT_DEFAULT_REGISTRY** 环境变量，但此设置会覆盖环境变量。当设定此参数时，内部 registry 还需要设置其主机名。如需了解更多详细信息，请参阅设置 registry 主机名。
ExternalRegistryHostname	ExternalRegistryHostname 设置默认外部镜像 registry 的主机名。只有在镜像 registry 对外公开时才应设置外部主机名。该值用于 ImageStreams 中的 **publicDockerImageRepository** 字段。该值必须采用 **hostname[:port]** 格式。

7.6.9. Kubernetes 的主配置

表 7.9. Kubernetes 主配置参数

参数名称	描述
APILevels	应该在启动时启用的 API 级别的列表 v1 作为示例。

参数名称	描述
DisabledAPIGroupVersions	组映射到应禁用的版本（或 *）的映射。
KubeletClientInfo	包含有关如何连接到 kubelet 的信息。
KubernetesMasterConfig	包含如何连接到 kubelet 的 KubernetesMasterConfig 的信息。如果存在，则使用此流程启动 kubernetes master。
MasterCount	应正在运行的预期 master 数量。这个值默认为 1，可以被设置为正整数，或者设置为 -1，这表示这是集群的一部分。
MasterIP	Kubernetes 资源的公共 IP 地址。如果为空，来自 **net.InterfaceAddrs** 的第一个结果将被使用。
MasterKubeConfig	描述如何将此节点连接到 master 的 ***.kubeconfig*** 文件的文件名。
PodEvictionTimeout	控制删除失败节点上的 pod 的宽限期。它需要有效的持续时间字符串。如果为空，您会收到默认的 pod 驱除超时。默认值为 **5m0s**。

ProxyClientInfo	指定代理到 pod 时使用的客户端证书/密钥。例如：
	`proxyClientInfo:`
	`  certFile: master.proxy-client.crt`
	`  keyFile: master.proxy-client.key`

ServicesNodePortRange	用于分配主机上服务公共端口的范围。默认 30000-32767。
ServicesSubnet	用于分配服务 IP 的子网。
StaticNodeNames	静态已知的节点列表。

7.6.10. 网络配置

仔细选择以下参数中的 CIDR，因为 IPv4 地址空间由节点的所有用户共享。OpenShift Container Platform 为自己的使用保留 IPv4 地址空间中的 CIDR，并为外部用户和集群间共享的地址保留来自 IPv4 地址空间的 CIDR。

表 7.10. 网络配置参数

参数名称	描述
ClusterNetworkCIDR	指定全局覆盖网络的 L3 空间的 CIDR 字符串。这为集群网络的内部使用保留。

参数名称	描述
externalIPNetworkCIDRs	控制服务外部 IP 字段可接受哪些值。如果为空，则不能设置 **externalIP**。它可以包含检查访问的 CIDR 列表。如果 CIDR 的前缀为 **!**，则该 CIDR 中的 IP 将被拒绝。首先应用拒绝，然后针对其中一个允许的 CIDR 检查的 IP。为了安全起见，您必须确保此范围不会与节点、Pod 或服务 CIDR 重叠。
HostSubnetLength	分配给每一主机子网的位数。例如，8 表示主机上的 /24 网络。
ingressIPNetworkCIDR	控制为裸机上的类型为 **LoadBalancer** 的服务分配入口 IP 的范围。它可以包含从其中分配的一个 CIDR。默认情况下，配置了 **172.46.0.0/16**。为安全起见，您应该确保此范围与为外部 IP、节点、Pod 或服务保留的 CIDR 重叠。
HostSubnetLength	分配给每一主机子网的位数。例如，8 表示主机上的 /24 网络。

NetworkConfig	传递给 compiled-in-network 的插件。此处的许多选项可以在 Ansible 清单中控制。
	• **NetworkPluginName** (字符串)
	• **ClusterNetworkCIDR** （字符串）
	• **hostSubnetLength** （无符号整数）
	• **ServiceNetworkCIDR** (字符串)
	• **externalIPNetworkCIDRs** （字符串数组）：控制服务外部 IP 字段可以接受哪些值。如果为空，则不能设置外部 IP。它可以包含检查访问的 CIDR 列表。如果 CIDR 的前缀为 **!**，则该 CIDR 中的 IP 会拒绝。首先应用拒绝，然后针对其中一个允许的 CIDR 检查 IP。为了安全起见，您应该确保此范围与节点、Pod 或服务 CIDR 不重叠。
	例如：
	`networkConfig:`
	`  clusterNetworks`
	`  - cidr: 10.3.0.0/16`
	`    hostSubnetLength: 8`
	`  networkPluginName: example/openshift-ovs-subnet`
	`# serviceNetworkCIDR must match`
	`kubernetesMasterConfig.servicesSubnet`
	`  serviceNetworkCIDR: 179.29.0.0/16`

NetworkPluginName	要使用的网络插件的名称。
ServiceNetwork	用于指定服务网络的 CIDR 字符串。

7.6.11. OAuth 身份验证配置

表 7.11. OAuth 配置参数

参数名称	描述
AlwaysShowProviderSelection	强制供应商选择页面，即使只有一个提供程序，也要呈现。
AssetPublicURL	用于构建有效客户端重定向 URL 用于外部访问。
Error	一个包含 go 模板的文件路径，用于在身份验证期间呈现错误页面，或授权流（如果没有指定），则会使用默认错误页面。
IdentityProviders	订购用户以识别自身的方式列表。
Login	包含用来呈现登录页面的 go 模板的文件路径。如果未指定，则使用默认登录页面。
MasterCA	用于验证到 **MasterURL** 的 TLS 连接的 CA。
MasterPublicURL	用于构建有效客户端重定向 URL 用于外部访问。
MasterURL	用于让服务器对服务器调用来交换访问令牌的授权代码。

OAuthConfig	如果存在，/oauth 端点会根据定义的参数启动。例如：
	
	oauthConfig:
	  assetPublicURL:
	https://master.ose32.example.com:8443/console/
	  grantConfig:
	  method: auto
	  identityProviders:
	  - challenge: true
	  login: true
	  mappingMethod: claim
	  name: htpasswd_all
	  provider:
	  apiVersion: v1
	  kind: HTPasswdPasswordIdentityProvider
	  file: /etc/origin/openshift-passwd
	  masterCA: ca.crt
	  masterPublicURL: https://master.ose32.example.com:8443
	  masterURL: https://master.ose32.example.com:8443
	  sessionConfig:
	  sessionMaxAgeSeconds: 3600
	  sessionName: ssn
	  sessionSecretsFile: /etc/origin/master/session-secrets.yaml
	  tokenConfig:
	  accessTokenMaxAgeSeconds: 86400
	  authorizeTokenMaxAgeSeconds: 500

OAuthTemplates	允许自定义登录页面等页面。

参数名称	描述
ProviderSelection	包含用来呈现提供程序选择页面的 go 模板的文件路径。如果未指定，则使用默认供应商选择页面。
SessionConfig	包含有关配置会话的信息。
Templates	允许您自定义登录页面等页面。
TokenConfig	包含授权和访问令牌的选项。

7.6.12. 项目配置

表 7.12. 项目配置参数

参数名称	描述
DefaultNodeSelector	包含默认项目节点选择器。

ProjectConfig	包含有关项目创建和默认值的信息：
	• **defaultNodeSelector** （字符串）：包含默认项目节点选择器。
	• **projectRequestMessage** （字符串）：当用户无法通过 projectrequest API 端点请求一个项目时，向用户显示的字符串。
	• **ProjectRequestTemplate** (字符串):响应 projectrequest 以创建项目时使用的模板。采用 **<namespace>/<template>** 格式。它是可选的；如果没有指定，则使用默认模板。
	• **SecurityAllocator**:控制为项目自动分配 UID 和 MCS 标签。如果为 nil，则禁用分配：
	  ◦ **mcsAllocatorRange** （字符串）：定义将分配给命名空间的 MCS 类别范围。格式为 **<prefix>/<numberOfLabels>[,<maxCategory>]**。默认值为 **s0/2**，并从 c0 → c1023 分配，这表示总计有 535k 标签。如果此值在启动后改变，新项目可能会接收已经分配给其他项目的标签。前缀可以是任何有效的 SELinux 术语集合（包括 user、role 和 type）。但是，使用默认前缀可让服务器自动设置它们。例如，**s0:/2** 分配 s0:c0,c0 到 s0:c511,c511 的标签，而 **s0:/2,512** 会将标签从 s0:c0,c0,c0 分配给 s0:c511,c511,511。
	  ◦ **mcsLabelsPerProject** (整数):定义每个项目要保留的标签数。默认值是 **5**，用于匹配默认 UID 和 MCS 范围。
	  ◦ **uidAllocatorRange** (字符串):定义自动分配给项目的 Unix 用户 ID(UID)，以及每个命名空间获取的块的大小。例如，**1000-1999/10** 将为每个命名空间分配十个 UID，并能够在用尽空间前分配最多 100 个块。默认值为在 10k 块中分配 10 亿到 20 亿的块，这是用户命名空间启动时为容器镜像的预期范围范围。

参数名称	描述
ProjectRequestMessage	当用户无法通过项目请求 API 端点来请求项目时向用户显示的字符串。
ProjectRequestTemplate	响应 **projectrequest** 以创建项目时使用的模板。它采用 namespace/template 格式，它是可选的。如果没有指定，则使用默认模板。

7.6.13. 调度程序配置

表 7.13. 调度程序配置参数

参数名称	描述
SchedulerConfigFile	指向描述如何设置调度程序的文件。如果为空，您可以获得默认的调度规则

7.6.14. 安全分配器配置

表 7.14. 安全分配器参数

参数名称	描述
MCSAllocatorRange	定义将分配给命名空间的 MCS 类别范围。格式为 **<prefix>/<numberOfLabels>[,<maxCategory>]**。默认值为 **s0/2**，并从 c0 到 c1023 分配，这表示有总计 535k 个标签（1024 选择 2 ~ 535k）。如果此值在启动后改变，新项目可能会接收已经分配给其他项目的标签。前缀可以是任何有效的 SELinux 术语集合（包括 user、role 和 type），虽然它们保留为默认值，但允许服务器自动设置它们。
SecurityAllocator	控制为项目自动分配 UID 和 MCS 标签。如果为 nil，则禁用分配。
UIDAllocatorRange	定义会自动分配给项目的 Unix 用户 ID(UID)，以及每个命名空间获取的块的大小。例如，1000-1999/10 将根据每个命名空间分配十个 UID，并在用尽空间前分配最多 100 个块。默认值为在 10k 块中分配 10 亿到 20 亿的块（这是在用户命名空间启动后使用范围容器镜像的预期大小）。

7.6.15. 服务帐户配置

表 7.15. 服务帐户配置参数

参数名称	描述
LimitSecretReferences	控制是否允许服务帐户在没有显式引用命名空间中引用任何 secret。
ManagedNames	在每个命名空间中创建的服务帐户名称列表。如果没有指定名称，则不会启动 **ServiceAccountsController**。

参数名称	描述
MasterCA	用于验证 TLS 连接到主主机的 CA。服务帐户控制器会自动将此文件的内容注入到 pod 中，以便它们能够验证与主控机的连接。
PrivateKeyFile	包含 PEM 编码私有 RSA 密钥的文件，用于签署服务帐户令牌。如果没有指定私钥，则不会启动服务帐户 **TokensController**。
PublicKeyFiles	文件列表，各自包含 PEM 编码的公共 RSA 密钥。如果有任何文件包含私钥，则使用密钥的公钥部分。公钥列表用于验证所提供的服务帐户令牌。每个密钥按顺序尝试，直到列表耗尽或验证成功为止。如果没有指定密钥，则不会使用服务帐户身份验证。
ServiceAccountConfig	包含与服务帐户相关的选项： • **LimitSecretReferences** (boolean):控制是否允许服务帐户在没有显式引用命名空间中引用任何 secret。 • **ManagedNames** （字符串）：在每个命名空间中创建的服务帐户名称列表。如果没有指定名称，则不会启动 **ServiceAccountsController**。 • **MasterCA** （字符串）：用于验证 TLS 连接到主设备的证书颁发机构。服务帐户控制器会自动将此文件的内容注入到 pod 中，以便它们能够验证与主控机的连接。 • **privateKeyFile** （字符串）：包含 PEM 编码的私有 RSA 密钥，用于签署服务帐户令牌。如果没有指定私钥，则不会启动服务帐户 **TokensController**。 • **publicKeyFiles** （字符串）：文件列表，各自包含 PEM 编码的公共 RSA 密钥。如果有任何文件包含私钥，OpenShift Container Platform 将使用密钥的公钥部分。公钥列表用于验证服务帐户令牌 ; 每个密钥按顺序尝试，直到列表耗尽或验证成功为止。如果没有指定密钥，则无法使用服务帐户身份验证。

7.6.16. 服务信息配置

表 7.16. 服务信息配置参数

参数名称	描述
AllowRecursiveQueries	允许主服务器上的 DNS 服务器以递归方式应答查询。请注意，打开解析器可用于 DNS 模糊攻击，并且主 DNS 不应该被公共网络访问。
BindAddress	要服务的 ip:port。
BindNetwork	控制导入镜像的限值和行为。
CertFile	包含 PEM 编码证书的文件。
CertInfo	用于提供安全流量的 TLS 证书信息。

参数名称	描述
ClientCA	您识别传入客户端证书的所有签名人的证书捆绑包。

dnsConfig	如果存在，则根据定义的参数启动 DNS 服务器。例如：
	`dnsConfig:`
	`  bindAddress: 0.0.0.0:8053`
	`  bindNetwork: tcp4`

DNSDomain	包含域后缀。
DNSIP	包含 IP。
KeyFile	包含由 **CertFile** 指定的证书的 PEM 编码私钥的文件。
MasterClientConnectionOverrides	为用于连接 master 的客户端连接提供覆盖。这个参数不被支持。要设置 QPS 和 burst 值，请参阅设置节点 QPS 和 Burst 值。
MaxRequestsInFlight	允许到服务器的并发请求数。如果零，则不限制。
NamedCertificates	用于保护到特定主机名请求的证书列表。
RequestTimeoutSecond	请求超时前的秒数。默认值为 60 分钟。如果 -1，请求没有限制。
ServingInfo	资产的 HTTP 服务信息。

7.6.17. 卷配置

表 7.17. 卷配置参数

参数名称	描述
DynamicProvisioningEnabled	启用或禁用动态置备的布尔值。默认为 **true**。
FSGroup	为每个 FSGroup 在每个节点上启用 本地存储配额。目前，这仅适用于 emptyDir 卷，如果底层 **volumeDirectory** 位于 XFS 文件系统中。
MasterVolumeConfig	包含用于在 master 节点上配置卷插件的选项。
NodeVolumeConfig	包含用于在节点上配置卷的选项。
VolumeConfig	包含用于在节点中配置卷插件的选项： • **DynamicProvisioningEnabled**（布尔值）：默认值为 **true**，并在 **false** 时关闭动态置备。

参数名称	描述
VolumeDirectory	卷所在的目录。使用 **openshift_node_group_data_dir** 参数更改此值。

7.6.18. 基本审计

审计提供一组安全相关的按时间排序的记录，记录各个用户、管理员或其他系统组件影响系统的一系列活动。

审计在 API 服务器级别运作，记录所有传入到服务器的请求。每个审计日志包含两个条目：

1. 请求行中包含：
    a. 用于匹配响应行的唯一 ID（参见 #2）
    b. 请求的源 IP
    c. 被调用的 HTTP 方法
    d. 调用该操作的原始用户
    e. 操作的模拟用户（**self** 表示自己）
    f. 操作的模拟组（**lookup** 表示用户的组）
    g. 请求的命名空间或 <none>
    h. 请求的 URI
2. 响应行中包括：
    a. #1 中的唯一 ID
    b. 响应代码

用户 **admin** 询问 pod 列表的输出示例：

```
AUDIT: id="5c3b8227-4af9-4322-8a71-542231c3887b" ip="127.0.0.1" method="GET" user="admin"
as="<self>" asgroups="<lookup>" namespace="default" uri="/api/v1/namespaces/default/pods"
AUDIT: id="5c3b8227-4af9-4322-8a71-542231c3887b" response="200"
```

7.6.18.1. 启用基本审计

以下流程在安装后启用基本审计。

注意

高级审计必须在安装过程中启用。

1. 编辑所有 master 节点上的 */etc/origin/master/master-config.yaml* 文件，如下例所示：

```
auditConfig:
```

```
auditFilePath: "/var/log/origin/audit-ocp.log"
enabled: true
maximumFileRetentionDays: 14
maximumFileSizeMegabytes: 500
maximumRetainedFiles: 15
```

2. 重启集群中的 API pod。

```
# /usr/local/bin/master-restart api
```

要在安装过程中启用基本审计，请在清单文件中添加以下变量声明。根据情况调整值。

```
openshift_master_audit_config={"enabled": true, "auditFilePath": "/var/lib/origin/openpaas-oscp-audit.log", "maximumFileRetentionDays": 14, "maximumFileSizeMegabytes": 500, "maximumRetainedFiles": 5}
```

审计配置采用以下参数：

表 7.18. Audit 配置参数

参数名称	描述
enabled	启用或禁用审计日志的布尔值。默认为 **false**。
auditFilePath	请求应记录到的文件路径。如果没有设置，日志会被输出到 master 日志中。
maximumFileRetentionDays	根据文件名中编码的时间戳，指定用于保留旧审计日志文件的最大天数。
maximumRetainedFiles	指定要保留的旧审计日志文件的最大数量。
maximumFileSizeMegabytes	在轮转日志文件前，指定日志文件的最大大小（以 MB 为单位）。默认值为 100MB。

Audit 配置示例

```
auditConfig:
  auditFilePath: "/var/log/origin/audit-ocp.log"
  enabled: true
  maximumFileRetentionDays: 14
  maximumFileSizeMegabytes: 500
  maximumRetainedFiles: 15
```

当您定义 **auditFilePath** 参数时，如果该目录不存在，则会创建该目录。

7.6.19. 高级审计

高级审计功能对 基本审计功能 进行了一些改进，包括精细的事件过滤和多个输出后端。

要启用高级审计功能，您可以创建一个审计策略文件，并在 **openshift_master_audit_config** 和 **openshift_master_audit_policyfile** 参数中指定以下值：

```
openshift_master_audit_config={"enabled": true, "auditFilePath": "/var/log/origin/audit-ocp.log",
"maximumFileRetentionDays": 14, "maximumFileSizeMegabytes": 500, "maximumRetainedFiles": 5,
"policyFile": "/etc/origin/master/adv-audit.yaml", "logFormat":"json"}
openshift_master_audit_policyfile="/<path>/adv-audit.yaml"
```

重要

您必须在安装集群前创建 ***adv-audit.yaml*** 文件，并指定其在集群清单文件中的位置。

下表包含您可以使用的附加选项。

表 7.19. 高级审计配置参数

参数名称	描述
policyFile	定义审计策略配置的文件的路径。
policyConfiguration	嵌入式审计策略配置。
logFormat	指定保存的审计日志的格式。允许的值是 **legacy** （基本审计中使用的格式）和 **json**。
webHookKubeConfig	定义审计 Webhook 配置的 **.kubeconfig** 格式文件的路径，事件发送到其中。
webHookMode	指定发送审计事件的策略。允许的值是 **block** （块处理另一个事件，直到之前已完全处理）和 **batch** （以批处理的形式缓冲事件并交付）。

重要

要启用高级审计功能，您必须提供 **policyFile 或 policyConfiguration** 描述审计规则：

Audit 策略配置示例

```
apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:

  # Do not log watch requests by the "system:kube-proxy" on endpoints or services
  - level: None 1
    users: ["system:kube-proxy"] 2
    verbs: ["watch"] 3
    resources: 4
    - group: ""
      resources: ["endpoints", "services"]

  # Do not log authenticated requests to certain non-resource URL paths.
  - level: None
    userGroups: ["system:authenticated"] 5
    nonResourceURLs: 6
```

```
  - "/api*" # Wildcard matching.
  - "/version"

# Log the request body of configmap changes in kube-system.
- level: Request
  resources:
  - group: "" # core API group
    resources: ["configmaps"]
  # This rule only applies to resources in the "kube-system" namespace.
  # The empty string "" can be used to select non-namespaced resources.
  namespaces: ["kube-system"] 7

# Log configmap and secret changes in all other namespaces at the metadata level.
- level: Metadata
  resources:
  - group: "" # core API group
    resources: ["secrets", "configmaps"]

# Log all other resources in core and extensions at the request level.
- level: Request
  resources:
  - group: "" # core API group
  - group: "extensions" # Version of group should NOT be included.

# A catch-all rule to log all other requests at the Metadata level.
- level: Metadata 8

# Log login failures from the web console or CLI. Review the logs and refine your policies.
- level: Metadata
  nonResourceURLs:
  - /login* 9
  - /oauth* 10
```

1 8 每个事件都可以记录四个可能级别：

- **None** - 不记录与此规则匹配的日志事件。
- **Metadata** - 日志请求元数据（请求用户、时间戳、资源、操作等），但不请求或响应正文。这与基本审计中使用的级别相同。
- **Request** - 日志记录事件元数据和请求正文，但不包括响应的正文。
- **RequestResponse** - 日志记录事件元数据、请求和响应正文。

2 适用于规则的用户列表。一个空列表表示每个用户。

3 此规则应用到的操作动词列表。一个空列表表示每个动词。这是与 API 请求关联的 Kubernetes 动词（包括 **get**, **list**, **watch**, **create**, **update**, **patch**, **delete**, **deletecollection**, 和 **proxy**）。

4 适用于该规则的资源列表。一个空列表表示每个资源。每个资源都被指定为分配给的组（例如，Kubernetes 核心 API、批处理、build.openshift.io 等）的空项，以及该组中的资源列表。

5 适用于该规则的组列表。一个空列表表示每个组。

6 规则应用到的非资源 URL 列表。

7 适用于规则的命名空间列表。一个空列表表示每个命名空间。

9 Web 控制台使用的端点。

10 CLI 使用的端点。

如需有关高级审计的更多信息，请参阅 Kubernetes 文档

7.6.20. 为 etcd 指定 TLS 密码

您可以指定在 master 和 etcd 服务器之间的通信时使用的 支持的 TLS 密码。

1. 在每个 etcd 节点上，升级 etcd：

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

2. 确认 etcd 版本为 3.2.22 或更高版本：

```
# etcd --version
etcd Version: 3.2.22
```

3. 在每个 master 主机上，指定在 **/etc/origin/master/master-config.yaml** 文件中启用的密码：

```
servingInfo:
  ...
  minTLSVersion: VersionTLS12
  cipherSuites:
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_RSA_WITH_AES_256_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
...
```

4. 在每个 master 主机上，重启 master 服务：

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

5. 确认应用了密码。例如，对于 TLSv1.2 密码 **ECDHE-RSA-AES128-GCM-SHA256**，请运行以下命令：

```
# openssl s_client -connect etcd1.example.com:2379 1
CONNECTED(00000003)
depth=0 CN = etcd1.example.com
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN = etcd1.example.com
verify error:num=21:unable to verify the first certificate
verify return:1
139905367488400:error:14094412:SSL routines:ssl3_read_bytes:sslv3 alert bad
certificate:s3_pkt.c:1493:SSL alert number 42
139905367488400:error:140790E5:SSL routines:ssl23_write:ssl handshake
failure:s23_lib.c:177:
---
```

```
Certificate chain
 0 s:/CN=etcd1.example.com
   i:/CN=etcd-signer@1529635004
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEkjCCAnqgAwIBAgIBATANBgkqhkiG9w0BAQsFADAhMR8wHQYDVQQDDBZldGNk
.........
....
eif87qttt0Sl1vS8DG1KQO1oOBlNkg==
-----END CERTIFICATE-----
subject=/CN=etcd1.example.com
issuer=/CN=etcd-signer@1529635004
---
Acceptable client certificate CA names
/CN=etcd-signer@1529635004
Client Certificate Types: RSA sign, ECDSA sign
Requested Signature Algorithms:
RSA+SHA256:ECDSA+SHA256:RSA+SHA384:ECDSA+SHA384:RSA+SHA1:ECDSA+SHA1

Shared Requested Signature Algorithms:
RSA+SHA256:ECDSA+SHA256:RSA+SHA384:ECDSA+SHA384:RSA+SHA1:ECDSA+SHA1

Peer signing digest: SHA384
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 1666 bytes and written 138 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID:
    Session-ID-ctx:
    Master-Key:
1EFA00A91EE5FC5EDDCFC67C8ECD060D44FD3EB23D834EDED929E4B74536F273C0F
9299935E5504B562CD56E76ED208D
    Key-Arg   : None
    Krb5 Principal: None
    PSK identity: None
    PSK identity hint: None
    Start Time: 1529651744
    Timeout   : 300 (sec)
    Verify return code: 21 (unable to verify the first certificate)
```

1 **etcd1.example.com** 是 etcd 主机的名称。

7.7. 节点配置文件

在安装过程中，OpenShift Container Platform 为每类节点组在 **openshift-node** 项目中创建一个 configmap：

- node-config-master
- node-config-infra
- node-config-compute
- node-config-all-in-one
- node-config-master-infra

若要对现有节点进行配置更改，请编辑相应的配置映射。各个节点上的 *同步 pod* 监视配置映射的变化。在安装过程中，使用 *sync Daemonsets* 和一个 ***/etc/origin/node/node-config.yaml*** 文件（节点的配置参数所在的文件）创建的同步 pod 被添加到每个节点。当同步 pod 检测到配置映射更改时，它会在该节点组的所有节点上更新 ***node-config.yaml***，并在适当的节点上重启 **atomic-openshift-node.service**。

```
$ oc get cm -n openshift-node
```

输出示例

```
NAME                   DATA      AGE
node-config-all-in-one     1         1d
node-config-compute        1         1d
node-config-infra          1         1d
node-config-master         1         1d
node-config-master-infra   1         1d
```

node-config-compute 组的配置映射示例

```
apiVersion: v1
authConfig:                                    (1)
  authenticationCacheSize: 1000
  authenticationCacheTTL: 5m
  authorizationCacheSize: 1000
  authorizationCacheTTL: 5m
dnsBindAddress: 127.0.0.1:53
dnsDomain: cluster.local
dnsIP: 0.0.0.0                                 (2)
dnsNameservers: null
dnsRecursiveResolvConf: /etc/origin/node/resolv.conf
dockerConfig:
  dockerShimRootDirectory: /var/lib/dockershim
  dockerShimSocket: /var/run/dockershim.sock
  execHandlerName: native
enableUnidling: true
imageConfig:
  format: registry.reg-aws.openshift.com/openshift3/ose-${component}:${version}
  latest: false
iptablesSyncPeriod: 30s
kind: NodeConfig
kubeletArguments:                              (3)
  bootstrap-kubeconfig:
  - /etc/origin/node/bootstrap.kubeconfig
```

```
  cert-dir:
  - /etc/origin/node/certificates
  cloud-config:
  - /etc/origin/cloudprovider/aws.conf
  cloud-provider:
  - aws
  enable-controller-attach-detach:
  - 'true'
  feature-gates:
  - RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true
  node-labels:
  - node-role.kubernetes.io/compute=true
  pod-manifest-path:
  - /etc/origin/node/pods 4
  rotate-certificates:
  - 'true'
masterClientConnectionOverrides:
  acceptContentTypes: application/vnd.kubernetes.protobuf,application/json
  burst: 40
  contentType: application/vnd.kubernetes.protobuf
  qps: 20
masterKubeConfig: node.kubeconfig
networkConfig: 5
  mtu: 8951
  networkPluginName: redhat/openshift-ovs-subnet 6
servingInfo: 7
  bindAddress: 0.0.0.0:10250
  bindNetwork: tcp4
  clientCA: client-ca.crt 8
volumeConfig:
  localQuota:
    perFSGroup: null
volumeDirectory: /var/lib/origin/openshift.local.volumes
```

1. 身份验证和授权配置选项.
2. 附加到 pod 的 */etc/resolv.conf* 的 IP 地址。
3. 直接传递给与 Kubelet 命令行参数匹配的 Kubelet 的键值对。
4. pod 清单文件或目录的路径。目录必须包含一个或多个清单文件。OpenShift 容器平台使用清单文件在节点上创建 pod。
5. 节点上的 pod 网络设置。
6. 软件定义型网络(SDN)插件.为 **ovs-subnet** 插件设置为 **redhat/openshift-ovs-subnet**，为 **ovs-multitenant** 插件设置为 **redhat/openshift-ovs-multitenant**，为 **ovs-networkpolicy** 插件设置为 **redhat/openshift-ovs-networkpolicy**。
7. 节点的证书信息。
8. 可选：PEM 编码的证书捆绑包。如果设置，则必须根据指定文件中的证书颁发机构显示并验证有效的客户端证书，然后才能检查请求标头中的用户名。

注意

不要手动修改 */etc/origin/node/node-config.yaml* 文件。

节点配置文件决定了节点的资源。如需更多信息，请参阅 Cluster Administrator 指南中的 Allocating node resources 部分。

7.7.1. Pod 和节点配置

表 7.20. Pod 和节点配置参数

参数名称	描述
NodeConfig	完全指定的配置启动 OpenShift Container Platform 节点。
NodeName	用于标识集群中的这个特定节点的值。如果可能，则这应该是您的完全限定主机名。如果您要描述一组静态节点到 master 节点，则这个值必须与列表中的其中一个值匹配。

7.7.2. Docker 配置

表 7.21. Docker 配置参数

参数名称	描述
AllowDisabledDocker	如果为 true，kubelet 将忽略 Docker 的错误。这意味着，节点可以在没有启动 docker 的机器上启动。
DockerConfig	保留 Docker 相关配置选项
ExecHandlerName	在容器中执行命令的处理程序。

7.7.3. 本地存储配置

您可以使用 XFS quota 子系统 基于 **emptyDir** 卷（如 secret 和配置映射）限制 **emptyDir** 卷和卷的大小。

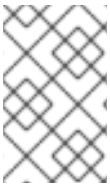
要限制 XFS 文件系统中的 **emptyDir** 卷大小，使用 **openshift-node** 项目中的 *node-config-compute* 配置映射为每个唯一 FSGroup 配置本地卷配额。

```
apiVersion: kubelet.config.openshift.io/v1
kind: VolumeConfig
  localQuota: 1
    perFSGroup: 1Gi 2
```

1 包含控制节点上本地卷配额的选项。

2 将此值设置为代表每个 [FSGroup] 的资源数量，每个节点（如 **1Gi**、**512Mi** 等）。要求 **volumeDirectory** 位于通过 **grpquota** 选项挂载的 XFS 文件系统上。匹配的安全性上下文约束 **fsGroup** 类型 必须设为 **MustRunAs**。

如果没有指定 FSGroup，这表示请求与 **RunAsAny** 的 SCC 匹配，则会跳过配额应用程序。



注意

不要直接编辑 */etc/origin/node/volume-config.yaml* 文件。该文件从 *node-config-compute* 配置映射创建。使用 *node-config-compute* 配置映射在 *volume-config.yaml* 文件中创建或编辑 paramaters。

7.7.4. 设置节点 Queries per Second (QPS) Limits 和 Burst 值

kubelet 与 API 服务器进行交互的频率取决于 qps 和 burst 值。如果每个节点上运行的 pod 有限，默认值就足够了。如果节点上有足够 CPU 和内存资源，可以在 */etc/origin/node/node-config.yaml* 文件中调整 qps 和 burst 值：

```
kubeletArguments:
  kube-api-qps:
  - "20"
  kube-api-burst:
  - "40"
```

注意

以上 qps 和 burst 值是 OpenShift Container Platform 的默认值。

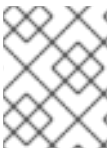
表 7.22. QPS 和 Burst 配置参数

参数名称	描述
kube-api-qps	Kubelet 与 APIServer 通信的 QPS 速率。默认值为 **20**。
kube-api-burst	Kubelet 与 APIServer 通信的突发率。默认值为 **40**。
ExecHandlerName	在容器中执行命令的处理程序。

然后，重启 OpenShift Container Platform 节点服务。

7.7.5. 使用 Docker 1.9+ 并行镜像拉取(pull)

如果使用 Docker 1.9+，您可能需要考虑启用并行镜像拉取，因为默认值是一次拉取镜像。



注意

Docker 1.9 之前的数据损坏潜在的问题。但是，从 1.9 开始，崩溃问题已解决，它可以安全地切换到并行拉取。

```
kubeletArguments:
  serialize-image-pulls:
  - "false" 1
```

1 更改为 **true** 以禁用并行拉取。这是默认配置。

7.8. 密码和其他敏感数据

对于一些身份验证配置，需要一个 LDAP **bindPassword** 或 OAuth **clientSecret** 值。这些值不是直接在主配置文件中指定，而是以环境变量、外部文件或加密文件形式提供。

环境变量示例

```
bindPassword:
  env: BIND_PASSWORD_ENV_VAR_NAME
```

外部文件示例

```
bindPassword:
  file: bindPassword.txt
```

加密的外部文件示例

```
bindPassword:
  file: bindPassword.encrypted
  keyFile: bindPassword.key
```

为以上示例创建加密的文件和密钥文件：

```
$ oc adm ca encrypt --genkey=bindPassword.key --out=bindPassword.encrypted
> Data to encrypt: B1ndPass0rd!
```

仅从 Ansible 主机清单文件中列出的第一个 master 运行 **oc adm** 命令，默认为 */etc/ansible/hosts*。

警告

加密的数据仅像解密密钥一样安全。请小心谨慎，以限制文件系统权限和对密钥文件的访问。

7.9. 创建新配置文件

从头开始定义 OpenShift Container Platform 配置时，请先创建新的配置文件。

对于 master 主机配置文件，使用 **openshift start** 命令和 **--write-config** 选项来写入配置文件。对于节点主机，使用 **oc adm create-node-config** 命令写入配置文件。

以下命令将相关的启动配置文件、证书文件和任何其他必要的文件写入指定的 **--write-config** 或 **--node-dir** 目录。

生成的证书文件在两年内有效，而证书颁发机构(CA)证书在五年内有效。这可以通过 **--expire-days** 和 **--signer-expire-days** 选项进行修改，但为了安全起见，建议您不要使其大于这些值。

在指定目录中为一体化服务器（master 和一个节点）创建配置文件：

```
$ openshift start --write-config=/openshift.local.config
```

要在指定目录中创建 master 配置文件和其他所需的文件：

```
$ openshift start master --write-config=/openshift.local.config/master
```

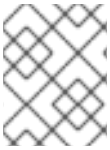
在指定目录中创建节点配置文件和其他相关文件：

```
$ oc adm create-node-config \
    --node-dir=/openshift.local.config/node-<node_hostname> \
    --node=<node_hostname> \
    --hostnames=<node_hostname>,<ip_address> \
    --certificate-authority="/path/to/ca.crt" \
    --signer-cert="/path/to/ca.crt" \
    --signer-key="/path/to/ca.key"
    --signer-serial="/path/to/ca.serial.txt"
    --node-client-certificate-authority="/path/to/ca.crt"
```

在创建节点配置文件时，**--hostnames** 选项接受以逗号分隔的每个主机名或 IP 地址列表，用于有效服务器证书。

7.10. 使用配置文件启动服务器

将主控机和节点配置文件修改为您的规格后，您可以在启动服务器时将它们指定为参数来使用。如果您指定配置文件，则您传递的其他命令行选项都不会被遵守。



注意

要修改集群中的节点，请根据需要更新节点配置映射。不要手动编辑 ***node-config.yaml*** 文件。

1. 使用 master 配置文件启动 master 服务器：

```
$ openshift start master \
    --config=/openshift.local.config/master/master-config.yaml
```

2. 使用节点配置文件和 ***node.kubeconfig*** 文件启动网络代理和 SDN 插件：

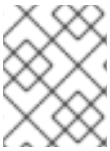
```
$ openshift start network \
    --config=/openshift.local.config/node-<node_hostname>/node-config.yaml \
    --kubeconfig=/openshift.local.config/node-<node_hostname>/node.kubeconfig
```

3. 使用节点配置文件启动节点服务器：

```
$ hyperkube kubelet \
    $(/usr/bin/openshift-node-config \
    --config=/openshift.local.config/node-<node_hostname>/node-config.yaml)
```

7.11. 查看 MASTER 和节点日志

OpenShift Container Platform 使用 **systemd-journald.service** 作为节点和名为 **master-logs** 的脚本，为 master 收集调试的日志消息。



注意

web 控制台中显示的行的数量在 5000 中被硬编码，且无法更改。要查看整个日志，可使用 CLI。

日志记录根据 Kubernetes 日志记录惯例使用五个日志消息，如下所示：

表 7.23. 日志级别选项

选项	描述
0	仅限错误和警告
2	普通信息
4	调试级别信息
6	API 级调试信息（请求/响应）
8	正文级别 API 调试信息

您可以根据需要 更改独立于 master 或节点的日志级别。

查看节点日志

要查看节点系统的日志，请运行以下命令：

```
# journalctl -r -u <journal_name>
```

使用 **-r** 选项显示最新的条目。

查看 master 日志

要查看 master 组件的日志，请运行以下命令：

```
# /usr/local/bin/master-logs <component> <container>
```

例如：

```
# /usr/local/bin/master-logs controllers controllers
# /usr/local/bin/master-logs api api
# /usr/local/bin/master-logs etcd etcd
```

将 master 日志重定向到文件

要将 master 日志的输出重定向到文件，请运行以下命令：

```
master-logs api api 2> file
```

7.11.1. 配置日志记录级别

您可以通过在 master 或 */etc/sysconfig/atomic-openshift-node* 文件中的 */etc/origin/master/master.env* 文件中设置 **DEBUG_LOGLEVEL** 选项来控制哪些 INFO 信息。将日志配置为收集所有信息可能会导致大型日志被解释，并可能会占用过量空间。仅在需要调试集群时收集所有消息。

注意

不论日志配置是什么，日志中会出现带有 FATAL、ERROR、WARNING 和一些 INFO 严重性的消息。

更改日志记录级别：

1. 编辑 master 或 */etc/sysconfig/atomic-openshift-node* 文件的 */etc/origin/master/master.env* 文件。
2. 在 **DEBUG_LOGLEVEL** 字段中，输入 **Log Level Options** 表中的值。例如：

```
DEBUG_LOGLEVEL=4
```

3. 根据需要重启 master 或节点主机。请参阅重启 OpenShift Container Platform 服务。

重启后，所有新日志消息均符合新设置。旧的消息不会更改。

注意

可以使用标准集群安装过程来设置默认日志级别。如需更多信息，请参阅 集群变量。

以下示例是不同日志级别重定向的主日志文件的摘录。已从这些示例中删除系统信息。

loglevel=2 的 master-logs api api 2> 文件 输出摘录

```
W1022 15:08:09.787705       1 server.go:79] Unable to keep dnsmasq up to date, 0.0.0.0:8053 must
point to port 53
I1022 15:08:09.787894       1 logs.go:49] skydns: ready for queries on cluster.local. for
tcp4://0.0.0.0:8053 [rcache 0]
I1022 15:08:09.787913       1 logs.go:49] skydns: ready for queries on cluster.local. for
udp4://0.0.0.0:8053 [rcache 0]
I1022 15:08:09.889022       1 dns_server.go:63] DNS listening at 0.0.0.0:8053
I1022 15:08:09.893156       1 feature_gate.go:190] feature gates: map[AdvancedAuditing:true]
I1022 15:08:09.893500       1 master.go:431] Starting OAuth2 API at /oauth
I1022 15:08:09.914759       1 master.go:431] Starting OAuth2 API at /oauth
I1022 15:08:09.942349       1 master.go:431] Starting OAuth2 API at /oauth
W1022 15:08:09.977088       1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
W1022 15:08:09.977176       1 swagger.go:38] No API exists for predefined swagger description
/api/v1
[restful] 2018/10/22 15:08:09 log.go:33: [restful/swagger] listing is available at
https://openshift.com:443/swaggerapi
[restful] 2018/10/22 15:08:09 log.go:33: [restful/swagger] https://openshift.com:443/swaggerui/ is
mapped to folder /swagger-ui/
I1022 15:08:10.231405       1 master.go:431] Starting OAuth2 API at /oauth
W1022 15:08:10.259523       1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
```

```
W1022 15:08:10.259555       1 swagger.go:38] No API exists for predefined swagger description
/api/v1
I1022 15:08:23.895493       1 logs.go:49] http: TLS handshake error from 10.10.94.10:46322: EOF
I1022 15:08:24.449577       1 crdregistration_controller.go:110] Starting crd-autoregister controller
I1022 15:08:24.449916       1 controller_utils.go:1019] Waiting for caches to sync for crd-autoregister
controller
I1022 15:08:24.496147       1 logs.go:49] http: TLS handshake error from 127.0.0.1:39140: EOF
I1022 15:08:24.821198       1 cache.go:39] Caches are synced for APIServiceRegistrationController
controller
I1022 15:08:24.833022       1 cache.go:39] Caches are synced for AvailableConditionController
controller
I1022 15:08:24.865087       1 controller.go:537] quota admission added evaluator for: { events}
I1022 15:08:24.865393       1 logs.go:49] http: TLS handshake error from 127.0.0.1:39162: read tcp4
127.0.0.1:443->127.0.0.1:39162: read: connection reset by peer
I1022 15:08:24.966917       1 controller_utils.go:1026] Caches are synced for crd-autoregister
controller
I1022 15:08:24.967961       1 autoregister_controller.go:136] Starting autoregister controller
I1022 15:08:24.967977       1 cache.go:32] Waiting for caches to sync for autoregister controller
I1022 15:08:25.015924       1 controller.go:537] quota admission added evaluator for: {
serviceaccounts}
I1022 15:08:25.077984       1 cache.go:39] Caches are synced for autoregister controller
W1022 15:08:25.304265       1 lease_endpoint_reconciler.go:176] Resetting endpoints for master
service "kubernetes" to [10.10.94.10]
E1022 15:08:25.472536       1 memcache.go:153] couldn't get resource list for
servicecatalog.k8s.io/v1beta1: the server could not find the requested resource
E1022 15:08:25.550888       1 memcache.go:153] couldn't get resource list for
servicecatalog.k8s.io/v1beta1: the server could not find the requested resource
I1022 15:08:29.480691       1 healthz.go:72] /healthz/log check
I1022 15:08:30.981999       1 controller.go:105] OpenAPI AggregationController: Processing item
v1beta1.servicecatalog.k8s.io
E1022 15:08:30.990914       1 controller.go:111] loading OpenAPI spec for
"v1beta1.servicecatalog.k8s.io" failed with: OpenAPI spec does not exists
I1022 15:08:30.990965       1 controller.go:119] OpenAPI AggregationController: action for item
v1beta1.servicecatalog.k8s.io: Rate Limited Requeue.
I1022 15:08:31.530473       1 trace.go:76] Trace[1253590531]: "Get /api/v1/namespaces/openshift-
infra/serviceaccounts/serviceaccount-controller" (started: 2018-10-22 15:08:30.868387562 +0000
UTC m=+24.277041043) (total time: 661.981642ms):
Trace[1253590531]: [661.903178ms] [661.89217ms] About to write a response
I1022 15:08:31.531366       1 trace.go:76] Trace[83808472]: "Get /api/v1/namespaces/aws-
sb/secrets/aws-servicebroker" (started: 2018-10-22 15:08:30.831296749 +0000 UTC
m=+24.239950203) (total time: 700.049245ms):
```

loglevel=4 的 master-logs api api 2> 文件 输出摘录

```
I1022 15:08:09.746980       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: AlwaysDeny.
I1022 15:08:09.747597       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: ResourceQuota.
I1022 15:08:09.748038       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: openshift.io/ClusterResourceQuota.
I1022 15:08:09.786771       1 start_master.go:458] Starting master on 0.0.0.0:443 (v3.10.45)
I1022 15:08:09.786798       1 start_master.go:459] Public master address is https://openshift.com:443
I1022 15:08:09.786844       1 start_master.go:463] Using images from
"registry.access.redhat.com/openshift3/ose-<component>:v3.10.45"
W1022 15:08:09.787046       1 dns_server.go:37] Binding DNS on port 8053 instead of 53, which may
```

```
not be resolvable from all clients
W1022 15:08:09.787705       1 server.go:79] Unable to keep dnsmasq up to date, 0.0.0.0:8053 must
point to port 53
I1022 15:08:09.787894       1 logs.go:49] skydns: ready for queries on cluster.local. for
tcp4://0.0.0.0:8053 [rcache 0]
I1022 15:08:09.787913       1 logs.go:49] skydns: ready for queries on cluster.local. for
udp4://0.0.0.0:8053 [rcache 0]
I1022 15:08:09.889022       1 dns_server.go:63] DNS listening at 0.0.0.0:8053
I1022 15:08:09.893156       1 feature_gate.go:190] feature gates: map[AdvancedAuditing:true]
I1022 15:08:09.893500       1 master.go:431] Starting OAuth2 API at /oauth
I1022 15:08:09.914759       1 master.go:431] Starting OAuth2 API at /oauth
I1022 15:08:09.942349       1 master.go:431] Starting OAuth2 API at /oauth
W1022 15:08:09.977088       1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
W1022 15:08:09.977176       1 swagger.go:38] No API exists for predefined swagger description
/api/v1
[restful] 2018/10/22 15:08:09 log.go:33: [restful/swagger] listing is available at
https://openshift.com:443/swaggerapi
[restful] 2018/10/22 15:08:09 log.go:33: [restful/swagger] https://openshift.com:443/swaggerui/ is
mapped to folder /swagger-ui/
I1022 15:08:10.231405       1 master.go:431] Starting OAuth2 API at /oauth
W1022 15:08:10.259523       1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
W1022 15:08:10.259555       1 swagger.go:38] No API exists for predefined swagger description
/api/v1
[restful] 2018/10/22 15:08:10 log.go:33: [restful/swagger] listing is available at
https://openshift.com:443/swaggerapi
[restful] 2018/10/22 15:08:10 log.go:33: [restful/swagger] https://openshift.com:443/swaggerui/ is
mapped to folder /swagger-ui/
I1022 15:08:10.444303       1 master.go:431] Starting OAuth2 API at /oauth
W1022 15:08:10.492409       1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
W1022 15:08:10.492507       1 swagger.go:38] No API exists for predefined swagger description
/api/v1
[restful] 2018/10/22 15:08:10 log.go:33: [restful/swagger] listing is available at
https://openshift.com:443/swaggerapi
[restful] 2018/10/22 15:08:10 log.go:33: [restful/swagger] https://openshift.com:443/swaggerui/ is
mapped to folder /swagger-ui/
I1022 15:08:10.774824       1 master.go:431] Starting OAuth2 API at /oauth
I1022 15:08:23.808685       1 logs.go:49] http: TLS handshake error from 10.128.0.11:39206: EOF
I1022 15:08:23.815311       1 logs.go:49] http: TLS handshake error from 10.128.0.14:53054: EOF
I1022 15:08:23.822286       1 customresource_discovery_controller.go:174] Starting
DiscoveryController
I1022 15:08:23.822349       1 naming_controller.go:276] Starting NamingConditionController
I1022 15:08:23.822705       1 logs.go:49] http: TLS handshake error from 10.128.0.14:53056: EOF
+24.277041043) (total time: 661.981642ms):
Trace[1253590531]: [661.903178ms] [661.89217ms] About to write a response
I1022 15:08:31.531366       1 trace.go:76] Trace[83808472]: "Get /api/v1/namespaces/aws-
sb/secrets/aws-servicebroker" (started: 2018-10-22 15:08:30.831296749 +0000 UTC
m=+24.239950203) (total time: 700.049245ms):
Trace[83808472]: [700.049245ms] [700.04027ms] END
I1022 15:08:31.531695       1 trace.go:76] Trace[1916801734]: "Get /api/v1/namespaces/aws-
sb/secrets/aws-servicebroker" (started: 2018-10-22 15:08:31.031163449 +0000 UTC
m=+24.439816907) (total time: 500.514208ms):
Trace[1916801734]: [500.514208ms] [500.505008ms] END
I1022 15:08:44.675371       1 healthz.go:72] /healthz/log check
```

```
I1022 15:08:46.589759       1 controller.go:537] quota admission added evaluator for: { endpoints}
I1022 15:08:46.621270       1 controller.go:537] quota admission added evaluator for: { endpoints}
I1022 15:08:57.159494       1 healthz.go:72] /healthz/log check
I1022 15:09:07.161315       1 healthz.go:72] /healthz/log check
I1022 15:09:16.297982       1 trace.go:76] Trace[2001108522]: "GuaranteedUpdate etcd3:
*core.Node" (started: 2018-10-22 15:09:15.139820419 +0000 UTC m=+68.548473981) (total time:
1.158128974s):
Trace[2001108522]: [1.158012755s] [1.156496534s] Transaction committed
I1022 15:09:16.298165       1 trace.go:76] Trace[1124283912]: "Patch /api/v1/nodes/master-
0.com/status" (started: 2018-10-22 15:09:15.139695483 +0000 UTC m=+68.548348970) (total time:
1.158434318s):
Trace[1124283912]: [1.158328853s] [1.15713683s] Object stored in database
I1022 15:09:16.298761       1 trace.go:76] Trace[24963576]: "GuaranteedUpdate etcd3: *core.Node"
(started: 2018-10-22 15:09:15.13159057 +0000 UTC m=+68.540244112) (total time: 1.167151224s):
Trace[24963576]: [1.167106144s] [1.165570379s] Transaction committed
I1022 15:09:16.298882       1 trace.go:76] Trace[222129183]: "Patch /api/v1/nodes/node-
0.com/status" (started: 2018-10-22 15:09:15.131269234 +0000 UTC m=+68.539922722) (total time:
1.167595526s):
Trace[222129183]: [1.167517296s] [1.166135605s] Object stored in database
```

loglevel=8 中的 master-logs api api 2> 文件 输出摘录

```
1022 15:11:58.829357       1 plugins.go:84] Registered admission plugin "NamespaceLifecycle"
I1022 15:11:58.839967       1 plugins.go:84] Registered admission plugin "Initializers"
I1022 15:11:58.839994       1 plugins.go:84] Registered admission plugin
"ValidatingAdmissionWebhook"
I1022 15:11:58.840012       1 plugins.go:84] Registered admission plugin
"MutatingAdmissionWebhook"
I1022 15:11:58.840025       1 plugins.go:84] Registered admission plugin "AlwaysAdmit"
I1022 15:11:58.840082       1 plugins.go:84] Registered admission plugin "AlwaysPullImages"
I1022 15:11:58.840105       1 plugins.go:84] Registered admission plugin
"LimitPodHardAntiAffinityTopology"
I1022 15:11:58.840126       1 plugins.go:84] Registered admission plugin "DefaultTolerationSeconds"
I1022 15:11:58.840146       1 plugins.go:84] Registered admission plugin "AlwaysDeny"
I1022 15:11:58.840176       1 plugins.go:84] Registered admission plugin "EventRateLimit"
I1022 15:11:59.850825       1 feature_gate.go:190] feature gates: map[AdvancedAuditing:true]
I1022 15:11:59.859108       1 register.go:154] Admission plugin AlwaysAdmit is not enabled.  It will not
be started.
I1022 15:11:59.859284       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: AlwaysAdmit.
I1022 15:11:59.859809       1 register.go:154] Admission plugin NamespaceAutoProvision is not
enabled.  It will not be started.
I1022 15:11:59.859939       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: NamespaceAutoProvision.
I1022 15:11:59.860594       1 register.go:154] Admission plugin NamespaceExists is not enabled.  It
will not be started.
I1022 15:11:59.860778       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: NamespaceExists.
I1022 15:11:59.863999       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: NamespaceLifecycle.
I1022 15:11:59.864626       1 register.go:154] Admission plugin EventRateLimit is not enabled.  It will
not be started.
I1022 15:11:59.864768       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: EventRateLimit.
I1022 15:11:59.865259       1 register.go:154] Admission plugin ProjectRequestLimit is not enabled.  It
```

```
will not be started.
I1022 15:11:59.865376       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: ProjectRequestLimit.
I1022 15:11:59.866126       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: OriginNamespaceLifecycle.
I1022 15:11:59.866709       1 register.go:154] Admission plugin openshift.io/RestrictSubjectBindings
is not enabled.  It will not be started.
I1022 15:11:59.866761       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: openshift.io/RestrictSubjectBindings.
I1022 15:11:59.867304       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: openshift.io/JenkinsBootstrapper.
I1022 15:11:59.867823       1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: openshift.io/BuildConfigSecretInjector.
I1022 15:12:00.015273       1 master_config.go:476] Initializing cache sizes based on 0MB limit
I1022 15:12:00.015896       1 master_config.go:539] Using the lease endpoint reconciler with
TTL=15s and interval=10s
I1022 15:12:00.018396       1 storage_factory.go:285] storing { apiServerIPInfo} in v1, reading as
__internal from storagebackend.Config{Type:"etcd3", Prefix:"kubernetes.io", ServerList:
[]string{"https://master-0.com:2379"}, KeyFile:"/etc/origin/master/master.etcd-client.key",
CertFile:"/etc/origin/master/master.etcd-client.crt", CAFile:"/etc/origin/master/master.etcd-ca.crt",
Quorum:true, Paging:true, DeserializationCacheSize:0, Codec:runtime.Codec(nil),
Transformer:value.Transformer(nil), CompactionInterval:300000000000,
CountMetricPollPeriod:60000000000}
I1022 15:12:00.037710       1 storage_factory.go:285] storing { endpoints} in v1, reading as __internal
from storagebackend.Config{Type:"etcd3", Prefix:"kubernetes.io", ServerList:[]string{"https://master-
0.com:2379"}, KeyFile:"/etc/origin/master/master.etcd-client.key",
CertFile:"/etc/origin/master/master.etcd-client.crt", CAFile:"/etc/origin/master/master.etcd-ca.crt",
Quorum:true, Paging:true, DeserializationCacheSize:0, Codec:runtime.Codec(nil),
Transformer:value.Transformer(nil), CompactionInterval:300000000000,
CountMetricPollPeriod:60000000000}
I1022 15:12:00.054112       1 compact.go:54] compactor already exists for endpoints [https://master-
0.com:2379]
I1022 15:12:00.054678       1 start_master.go:458] Starting master on 0.0.0.0:443 (v3.10.45)
I1022 15:12:00.054755       1 start_master.go:459] Public master address is https://openshift.com:443
I1022 15:12:00.054837       1 start_master.go:463] Using images from
"registry.access.redhat.com/openshift3/ose-<component>:v3.10.45"
W1022 15:12:00.056957       1 dns_server.go:37] Binding DNS on port 8053 instead of 53, which may
not be resolvable from all clients
W1022 15:12:00.065497       1 server.go:79] Unable to keep dnsmasq up to date, 0.0.0.0:8053 must
point to port 53
I1022 15:12:00.066061       1 logs.go:49] skydns: ready for queries on cluster.local. for
tcp4://0.0.0.0:8053 [rcache 0]
I1022 15:12:00.066265       1 logs.go:49] skydns: ready for queries on cluster.local. for
udp4://0.0.0.0:8053 [rcache 0]
I1022 15:12:00.158725       1 dns_server.go:63] DNS listening at 0.0.0.0:8053
I1022 15:12:00.167910       1 htpasswd.go:118] Loading htpasswd file /etc/origin/master/htpasswd...
I1022 15:12:00.168182       1 htpasswd.go:118] Loading htpasswd file /etc/origin/master/htpasswd...
I1022 15:12:00.231233       1 storage_factory.go:285] storing {apps.openshift.io deploymentconfigs}
in apps.openshift.io/v1, reading as apps.openshift.io/__internal from
storagebackend.Config{Type:"etcd3", Prefix:"openshift.io", ServerList:[]string{"https://master-
0.com:2379"}, KeyFile:"/etc/origin/master/master.etcd-client.key",
CertFile:"/etc/origin/master/master.etcd-client.crt", CAFile:"/etc/origin/master/master.etcd-ca.crt",
Quorum:true, Paging:true, DeserializationCacheSize:0, Codec:runtime.Codec(nil),
Transformer:value.Transformer(nil), CompactionInterval:300000000000,
CountMetricPollPeriod:60000000000}
I1022 15:12:00.248136       1 compact.go:54] compactor already exists for endpoints [https://master-
```

```
0.com:2379]
I1022 15:12:00.248697       1 store.go:1391] Monitoring deploymentconfigs.apps.openshift.io count at
<storage-prefix>//deploymentconfigs
W1022 15:12:00.256861       1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
W1022 15:12:00.258106       1 swagger.go:38] No API exists for predefined swagger description
/api/v1
```

7.12. 重启 MASTER 和节点服务

要应用 master 或节点配置更改，您必须重启相应的服务。

要重新载入 master 配置更改，请使用 **master-restart** 命令重启在 control plane 静态 pod 中运行的主服务：

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

要重新载入节点配置更改，重启节点主机上的节点服务：

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

第 8 章 OPENSHIFT ANSIBLE BROKER 配置

8.1. 概述

当在集群中部署 OpenShift Ansible 代理 (OAB)时，其行为主要由代理在启动时加载的配置文件决定。代理的配置作为 ConfigMap 对象存储在代理的命名空间中（默认为**openshift-ansible-service-broker**）。

OpenShift Ansible Broker 配置文件示例

```
registry: 1
  - type: dockerhub
    name: docker
    url: https://registry.hub.docker.com
    org: <dockerhub_org>
    fail_on_error: false
  - type: rhcc
    name: rhcc
    url: https://registry.redhat.io
    fail_on_error: true
    white_list:
      - "^foo.*-apb$"
      - ".*-apb$"
    black_list:
      - "bar.*-apb$"
      - "^my-apb$"
  - type: local_openshift
    name: lo
    namespaces:
      - openshift
    white_list:
      - ".*-apb$"
dao: 2
  etcd_host: localhost
  etcd_port: 2379
log: 3
  logfile: /var/log/ansible-service-broker/asb.log
  stdout: true
  level: debug
  color: true
openshift: 4
  host: ""
  ca_file: ""
  bearer_token_file: ""
  image_pull_policy: IfNotPresent
  sandbox_role: "edit"
  keep_namespace: false
  keep_namespace_on_error: true
broker: 5
  bootstrap_on_startup: true
  dev_broker: true
  launch_apb_on_bind: false
  recovery: true
  output_request: true
```

```
    ssl_cert_key: /path/to/key
    ssl_cert: /path/to/cert
    refresh_interval: "600s"
    auth:
      - type: basic
        enabled: true
  secrets: 6
    - title: Database credentials
      secret: db_creds
      apb_name: dh-rhscl-postgresql-apb
```

1. 详情请参阅 Registry 配置。
2. 详情请参阅 DAO 配置。
3. 详情请参阅日志配置。
4. 有关详细信息，请参阅 OpenShift 配置。
5. 详情请参阅 Broker 配置。
6. 详情请参阅 Secret 配置。

8.2. 在 RED HAT PARTNER CONNECT REGISTRY 进行身份验证

在配置 Automation Broker 前，您必须在 OpenShift Container Platform 集群的所有节点上运行以下命令，才能使用 Red Hat Partner Connect：

```
$ docker --config=/var/lib/origin/.docker login -u <registry-user> -p <registry-password>
registry.connect.redhat.com
```

8.3. 修改 OPENSHIFT ANSIBLE BROKER 配置

在部署后修改 OAB 的默认代理配置：

1. 以具有 **cluster-admin** 权限的用户身份编辑 OAB 命名空间中的 **broker-config** ConfigMap 对象：

   ```
   $ oc edit configmap broker-config -n openshift-ansible-service-broker
   ```

2. 保存任何更新后，请重新部署 OAB 的部署配置以使更改生效：

   ```
   $ oc rollout latest dc/asb -n openshift-ansible-service-broker
   ```

8.4. REGISTRY 配置

registry 部分允许您定义代理应该用于 APB 的 registry。

表 8.1. registry 部分配置选项

字段	描述	必填
name	registry 的名称。代理用来识别来自此 registry 的 APB。	Y
user	用于向 registry 进行身份验证的用户名。当 **auth_type** 被设置为 **secret** 或 **file** 时，不使用。	N
pass	用于向 registry 进行身份验证的密码。当 **auth_type** 被设置为 **secret** 或 **file** 时，不使用。	N
auth_type	如果没有通过 **user** 和 **pass** 在代理配置中定义，代理应如何读取 registry 凭证可以是 **secret** （在代理命名空间中使用 secret） 或 **file** （使用挂载的文件系统）。	N
auth_name	存储应读取的 registry 凭证的机密或文件的名称。将 **auth_type** 设置为 **secret** 时使用。	N，仅当 **auth_type** 设置为 **secret** 或 **file** 时才需要。
org	镜像中包含的命名空间或机构。	N
type	registry 的类型。可用的适配器有 **mock**、**rhcc**、**openshift**、**dockerhub** 和 **local_openshift**。	Y
命名空间	用于配置 **local_openshift** registry 类型的命名空间列表。默认情况下，用户应使用 **openshift**。	N
url	用于检索镜像信息的 URL。适用于 RHCC 的广泛使用，而 **dockerhub** 类型则使用硬编码 URL。	N
fail_on_error	此 registry 失败，如果失败 bootstrap 请求。将停止执行其他 registry 加载。	N
white_list	用于定义应允许哪些镜像名称的正则表达式列表。必须有一个白名单，才能将 APB 添加到目录中。如果要检索 registry 中的所有 APB，您可以使用的最宽松的正则表达式为 **.*-apb$**。如需了解更多详细信息，请参阅 APB 过滤。	N
black_list	用于定义应不允许哪些镜像名称的正则表达式列表。如需了解更多详细信息，请参阅 APB 过滤。	N
images	要与 OpenShift Container Registry 搭配使用的镜像列表。	N

8.4.1. 生产或开发

一个*生产环境*代理配置设计为指向可信容器分发 registry，如 Red Hat Container Catalog(RHCC)：

```
registry:
```

```
  - name: rhcc
    type: rhcc
    url: https://registry.redhat.io
    tag: v3.11
    white_list:
      - ".*-apb$"
  - type: local_openshift
    name: localregistry
    namespaces:
      - openshift
    white_list: []
```

但是，*开发* 代理配置主要供使用代理的开发人员使用。要启用开发人员设置，将 registry 名称设置为 **dev**，并将 **broker** 部分中的 **dev_broker** 字段设置为 **true** ：

```
registry:
  name: dev
```

```
broker:
  dev_broker: true
```

8.4.2. 存储 registry 凭证

代理配置决定了代理应该如何读取任何 registry 凭证。它们可以从 **registry** 中的 **user** 和 **pass** 值读取，例如：

```
registry:
  - name: isv
    type: openshift
    url: https://registry.connect.redhat.com
    user: <user>
    pass: <password>
```

如果要确保这些凭据不可公开访问，**registry** 部分中的 **auth_type** 字段可以设置为 **secret** 或 **file** 类型。**secret** 类型将 registry 配置为使用代理命名空间中的 secret，而 **file** 类型将 registry 配置为使用已挂载为卷的 secret。

使用 **secret** 或 **file** 类型 ：

1. 关联的机密应定义值**用户名**和**密码**。在使用 secret 时，您必须确保 **openshift-ansible-service-broker** 命名空间存在，因为这是从中读取 secret 的位置。
   例如，创建一个 ***reg-creds.yaml*** 文件：

   ```
   $ cat reg-creds.yaml
   ---
   username: <user_name>
   password: <password>
   ```

2. 从 **openshift-ansible-service-broker** 命名空间中的此文件创建 secret：

   ```
   $ oc create secret generic \
       registry-credentials-secret \
       --from-file reg-creds.yaml \
   ```

```
  -n openshift-ansible-service-broker
```

3. 选择是否要使用 **secret** 或 **file** 类型 ：

- 使用 **secret** 类型：

a. 在代理配置中，将 **auth_type** 设置为 **secret**，**auth_name** 设置为 secret 的名称：

```
registry:
  - name: isv
    type: openshift
    url: https://registry.connect.redhat.com
    auth_type: secret
    auth_name: registry-credentials-secret
```

b. 设置 secret 所在的命名空间：

```
openshift:
  namespace: openshift-ansible-service-broker
```

- 使用 **file** 类型：

a. 编辑 **asb** 部署配置，将文件挂载到 */tmp/registry-credentials/reg-creds.yaml* 中：

```
$ oc edit dc/asb -n openshift-ansible-service-broker
```

在 **containers.volumeMounts** 部分添加：

```
volumeMounts:
  - mountPath: /tmp/registry-credentials
    name: reg-auth
```

在 **volumes** 部分，添加：

```
    volumes:
      - name: reg-auth
        secret:
          defaultMode: 420
          secretName: registry-credentials-secret
```

b. 在代理配置中，将 **auth_type** 设置为 **file**，并将 **auth_type** 设置为文件的位置：

```
registry:
  - name: isv
    type: openshift
    url: https://registry.connect.redhat.com
    auth_type: file
    auth_name: /tmp/registry-credentials/reg-creds.yaml
```

8.4.3. APB 过滤

APB 可使用 **white_list** 或 **black_list** 参数的组合（根据代理配置中的 registry）组合来筛选其镜像名称。

这两者都是可选的正则表达式列表，这些表达式将在给定 registry 的总发现的 APB 中运行，以确定匹配项。

表 8.2. APB 过滤器行为

存在	允许	阻塞
只有白名单	匹配列表中的正则表达式。	任何不匹配的 APB。
只有黑名单	所有不匹配的 APB。	与列表中正则表达式匹配的 APB。
两者都存在	匹配白名单中的正则表达式，当不在黑名单中。	与黑名单中的正则表达式匹配的 APB。
无	没有来自 registry 的 APB。	来自该 registry 的所有 APB。

例如：

仅有白名单

```
white_list:
  - "foo.*-apb$"
  - "^my-apb$"
```

对于 **foo.*-apb$** 的任何匹配，在这种情况下，仅允许使用 **my-apb**。所有其他 APB 都会被拒绝。

仅有黑名单

```
black_list:
  - "bar.*-apb$"
  - "^foobar-apb$"
```

任何与 **bar.*-apb$** 以及本例中只有 **foobar-apb** 的匹配都会被阻断。所有其他 APB 都将被允许。

白名单和黑名单

```
white_list:
  - "foo.*-apb$"
  - "^my-apb$"
black_list:
  - "^foo-rootkit-apb$"
```

此处，**foo-rootkit-apb** 由黑名单明确阻止，尽管白名单匹配会被覆盖。

否则，只允许与 **foo.*-apb$** 和 **my-apb** 匹配的通过。

代理配置 registry 示例部分：

```
registry:
  - type: dockerhub
    name: dockerhub
    url: https://registry.hub.docker.com
```

```
    user: <user>
    pass: <password>
    org: <org>
    white_list:
      - "foo.*-apb$"
      - "^my-apb$"
    black_list:
      - "bar.*-apb$"
      - "^foobar-apb$"
```

8.4.4. 模拟 Registry

模拟 registry 有助于读取本地 APB 规格。这使用的是本地 spec 列表，而不是通过 registry 搜索镜像规格。将 registry 的名称设置为 **mock** 以使用模拟 registry。

```
registry:
  - name: mock
    type: mock
```

8.4.5. Dockerhub Registry

dockerhub 类型允许您从 DockerHub 中的特定机构加载 APB。例如， ansibleplaybookbundle 组织。

```
registry:
  - name: dockerhub
    type: dockerhub
    org: ansibleplaybookbundle
    user: <user>
    pass: <password>
    white_list:
      - ".*-apb$"
```

8.4.6. Ansible Galaxy Registry

galaxy 类型允许您使用 Ansible Galaxy 的 APB 角色。您还可以选择指定机构。

```
registry:
  - name: galaxy
    type: galaxy
    # Optional:
    # org: ansibleplaybookbundle
    runner: docker.io/ansibleplaybookbundle/apb-base:latest
    white_list:
      - ".*$"
```

8.4.7. 本地 OpenShift Container Registry

通过使用 **local_openshift** 类型，您可以从 OpenShift Container Platform 集群内部的 OpenShift Container Registry 加载 APB。您可以配置您要在其中查找公布的 APB 的命名空间。

```
registry:
  - type: local_openshift
```

```
    name: lo
    namespaces:
      - openshift
    white_list:
      - ".*-apb$"
```

8.4.8. Red Hat Container Catalog Registry

使用 **rhcc** 类型将允许您加载发布到 Red Hat Container Catalog (RHCC)registry 中的 APB。

```
registry:
  - name: rhcc
    type: rhcc
    url: https://registry.redhat.io
    white_list:
      - ".*-apb$"
```

8.4.9. Red Hat Partner Connect Registry

Red Hat Container Catalog 中的第三方镜像是从 Red Hat 合作伙伴 Connect Registry 提供的，地址为 https://registry.connect.redhat.com。**partner_rhcc** 类型允许代理从合作伙伴注册表引导，以检索 APB 列表并加载其 spec。合作伙伴 Registry 需要身份验证才能使用有效的红帽客户门户网站用户名和密码拉取镜像。

```
registry:
  - name: partner_reg
    type: partner_rhcc
    url:  https://registry.connect.redhat.com
    user: <registry_user>
    pass: <registry_password>
    white_list:
      - ".*-apb$"
```

由于合作伙伴 registry 需要身份验证，因此还需要以下手动步骤来配置代理以使用合作伙伴 Registry URL：

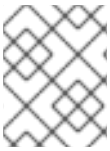
1. 在 OpenShift Container Platform 集群的所有节点上运行以下命令：

```
# docker --config=/var/lib/origin/.docker \
    login -u <registry_user> -p <registry_password> \
    registry.connect.redhat.com
```

8.4.10. Helm Chart Registry

使用 **helm** 类型时，您可以使用 Helm Chart 仓库中的 Helm Chart。

```
registry:
  - name: stable
    type: helm
    url: "https://kubernetes-charts.storage.googleapis.com"
    runner: "docker.io/automationbroker/helm-runner:latest"
    white_list:
      - ".*"
```



注意

stable 存储库中的许多 Helm chart 不适用于 OpenShift Container Platform，如果使用它们，则会失败。

8.4.11. API V2 Docker Registry

通过使用 **apiv2** 类型，您可以使用 Docker Registry HTTP API V2 协议的 docker registry 中的镜像。

```
registry:
  - name: <registry_name>
    type: apiv2
    url:  <registry_url>
    user: <registry-user>
    pass: <registry-password>
    white_list:
      - ".*-apb$"
```

如果 registry 需要身份验证来拉取镜像，这可以通过在现有集群的每个节点中运行以下命令来实现：

```
$ docker --config=/var/lib/origin/.docker login -u <registry-user> -p <registry-password> <registry_url>
```

8.4.12. Quay Docker Registry

使用 **quay** 类型可以加载发布到 CoreOS Quay Registry 的 APB。如果提供了身份验证令牌，则将加载令牌配置为访问的专用存储库。指定机构中的公共存储库不需要令牌来加载。

```
registry:
  - name: quay_reg
    type: quay
    url:  https://quay.io
    token: <for_private_repos>
    org: <your_org>
    white_list:
      - ".*-apb$"
```

如果 Quay registry 需要身份验证来拉取镜像，这可通过在现有集群的每个节点中运行以下命令来实现：

```
$ docker --config=/var/lib/origin/.docker login -u <registry-user> -p <registry-password> quay.io
```

8.4.13. 多个 registry

您可以使用多个 registry 将 APB 分开到逻辑机构中，并从同一代理管理它们。registry 必须具有唯一的非空名称。如果没有唯一名称，则服务代理将失败，并显示出错信息警报。

```
registry:
  - name: dockerhub
    type: dockerhub
    org: ansibleplaybookbundle
    user: <user>
    pass: <password>
    white_list:
      - ".*-apb$"
```

```
  - name: rhcc
    type: rhcc
    url: <rhcc_url>
    white_list:
      - ".*-apb$"
```

8.5. 代理身份验证

代理支持身份验证，这意味着连接到代理时，调用者必须提供每个请求的 Basic Auth 或 Bearer Auth 凭证。使用 **curl**，它像提供一样简单：

```
-u <user_name>:<password>
```

或者

```
-h "Authorization: bearer <token>
```

至命令。服务目录必须配置有包含用户名和密码组合的 secret 或 bearer 令牌的 secret。

8.5.1. 基本验证

要启用 Basic Auth 使用，在代理配置中设置以下内容：

```
broker:
  ...
  auth:
    - type: basic (1)
      enabled: true (2)
```

(1) **type** 字段指定要使用的身份验证类型。

(2) **enable** 字段允许您禁用特定的验证类型。这样，您只需要删除 **auth** 的整个部分才能禁用它。

8.5.1.1. 部署模板和 Secret

通常，代理是使用部署模板中的 ConfigMap 进行配置。您提供身份验证配置的方式与文件配置文件中相同。

以下是部署模板的示例：

```
auth:
  - type: basic
    enabled: ${ENABLE_BASIC_AUTH}
```

Basic Auth 的另一个部分是用于向代理进行身份验证的用户名和密码。虽然基本 Auth 实施可以由不同的后端服务提供支持，但当前支持的一个 secret 由 *secret* 提供支持。secret 必须通过卷挂载（在 ***/var/run/asb_auth*** 位置）注入 pod。这是代理读取用户名和密码的位置。

在 部署模板 中，必须指定 secret。例如：

```
- apiVersion: v1
  kind: Secret
```

```
  metadata:
    name: asb-auth-secret
    namespace: openshift-ansible-service-broker
  data:
    username: ${BROKER_USER}
    password: ${BROKER_PASS}
```

secret 必须包含用户名和密码。值必须采用 base64 编码。为这些条目生成值的最简单方法是使用 **echo** 和 **base64** 命令：

```
$ echo -n admin | base64 (1)
YWRtaW4=
```

(1) **n** 选项非常重要。

此 secret 现在必须通过卷挂载传递给 pod。这在部署模板中也配置：

```
spec:
  serviceAccount: asb
  containers:
  - image: ${BROKER_IMAGE}
    name: asb
    imagePullPolicy: IfNotPresent
    volumeMounts:
      ...
      - name: asb-auth-volume
        mountPath: /var/run/asb-auth
```

然后，在 **volumes** 部分挂载 secret:

```
volumes:
  ...
  - name: asb-auth-volume
    secret:
      secretName: asb-auth-secret
```

上述命令将创建一个位于 */var/run/asb-auth* 的卷挂载。此卷将有两个文件：由 **asb-auth-secret** 机密写入的用户名和密码。

8.5.1.2. 配置服务目录和代理通信

现在，代理配置为使用 Basic Auth，您必须告诉服务目录如何与代理通信。这通过代理资源的 **authInfo** 部分来完成。

以下是在服务目录中 **创建代理** 资源的示例。**spec** 告知服务目录代理正在侦听的 URL。**authInfo** 告诉它读取哪个 secret 获取身份验证信息。

```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: Broker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb-1338-openshift-ansible-service-broker.172.17.0.1.nip.io
```

```
  authInfo:
    basicAuthSecret:
      namespace: openshift-ansible-service-broker
      name: asb-auth-secret
```

从服务目录的 v0.0.17 开始，代理资源配置更改：

```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: ServiceBroker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb-1338-openshift-ansible-service-broker.172.17.0.1.nip.io
  authInfo:
    basic:
      secretRef:
        namespace: openshift-ansible-service-broker
        name: asb-auth-secret
```

8.5.2. Bearer Auth

默认情况下，如果没有指定身份验证，代理将使用 bearer 令牌身份验证(Bearer Auth)。Bearer Auth 使用 Kubernetes **apiserver** 库的委托身份验证。

配置通过 Kubernetes RBAC 角色和角色绑定授予对 URL 前缀的访问权限。代理添加了配置选项 **cluster_url** 以指定 **url_prefix**。这个值默认为 **openshift-ansible-service-broker**。

集群角色示例

```
- apiVersion: authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    name: access-asb-role
  rules:
  - nonResourceURLs: ["/ansible-service-broker", "/ansible-service-broker/*"]
    verbs: ["get", "post", "put", "patch", "delete"]
```

8.5.2.1. 部署模板和 Secret

以下是创建服务目录可以使用的 secret 的示例。本例假定角色 **access-asb-role** 已创建。来自部署模板：

```
- apiVersion: v1
  kind: ServiceAccount
  metadata:
    name: ansibleservicebroker-client
    namespace: openshift-ansible-service-broker

- apiVersion: authorization.openshift.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: ansibleservicebroker-client
  subjects:
  - kind: ServiceAccount
    name: ansibleservicebroker-client
```

```
    namespace: openshift-ansible-service-broker
  roleRef:
    kind: ClusterRole
    name: access-asb-role

- apiVersion: v1
  kind: Secret
  metadata:
    name: ansibleservicebroker-client
    annotations:
      kubernetes.io/service-account.name: ansibleservicebroker-client
  type: kubernetes.io/service-account-token
```

上例中会创建一个服务帐户，授予 **access-asb-role** 的访问权限，并为该服务帐户令牌创建 secret。

8.5.2.2. 配置服务目录和代理通信

现在，代理配置为使用 Bearer Auth 令牌，您必须告知服务目录如何与代理通信。这通过 **代理** 资源的 **authInfo** 部分来完成。

以下是在服务目录中 **创建代理** 资源的示例。**spec** 告知服务目录代理正在侦听的 URL。**authInfo** 告诉它读取哪个 secret 获取身份验证信息。

```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: ServiceBroker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb.openshift-ansible-service-broker.svc:1338${BROKER_URL_PREFIX}/
  authInfo:
    bearer:
      secretRef:
        kind: Secret
        namespace: openshift-ansible-service-broker
        name: ansibleservicebroker-client
```

8.6. DAO 配置

字段	描述	必填
etcd_host	etcd 主机的 URL。	Y
etcd_port	与 **etcd_host** 通信时使用的端口。	Y

8.7. 日志配置

字段	描述	必填
logfile	写入代理日志的位置。	Y

字段	描述	必填
stdout	将日志写入 stdout。	Y
level	日志输出的级别。	Y
color	颜色日志。	Y

8.8. OPENSHIFT 配置

字段	描述	必填
主机	OpenShift Container Platform 主机。	N
ca_file	证书颁发机构文件的位置。	N
bearer_token_file	要使用的 bearer 令牌的位置。	N
image_pull_policy	拉取镜像的时间。	Y
namespace	代理部署到的命名空间。诸如通过 secret 传递参数值等事项非常重要。	Y
sandbox_role	为 APB 沙盒环境提供的角色。	Y
keep_namespace	始终在执行 APB 后保留命名空间。	N
keep_namespace_on_error	在 APB 执行出错后保留命名空间。	N

8.9. 代理配置

broker 部分告诉代理应该启用和禁用功能。它还将告知代理在磁盘上查找要启用完整功能的文件。

字段	描述	默认值	必填
dev_broker	允许访问开发路由。	**false**	N
launch_apb_on_bind	允许绑定为 no-op。	**false**	N
bootstrap_on_startup	允许代理在启动时尝试 bootstrap 本身。将从配置的 registry 中检索 APB。	**false**	N

字段	描述	默认值	必填
recovery	允许代理通过处理 etcd 中记录的待处理作业来尝试恢复其自身。	**false**	N
output_request	允许代理输出对日志文件的请求，以便更轻松地调试。	**false**	N
ssl_cert_key	告知代理在哪里查找 TLS 密钥文件。如果没有设置，API 服务器将尝试创建一个。	**""**	N
ssl_cert	告知代理在哪里查找 TLS *.crt* 文件。如果没有设置，API 服务器将尝试创建一个。	**""**	N
refresh_interval	查询新镜像规格的 registry 的时间间隔。	**"600s"**	N
auto_escalate	允许代理在运行 APB 时升级用户的权限。	**false**	N
cluster_url	为代理预期的 URL 设置前缀。	**openshift-ansible-service-broker**	N

注意

async bind 和 unbind 是一个实验性功能，不受支持或默认启用。如果没有 async 绑定，将 **launch_apb_on_bind** 设置为 **true** 可能会导致 bind 操作超时，并会重试。代理会使用 "409 Conflicts" 处理这个问题，因为它使用不同的参数使用相同的绑定请求。

8.10. SECRET 配置

secrets 部分会在代理的命名空间和代理运行的 APB 中创建 secret 间的关联。代理使用这些规则将 secret 挂载到运行的 APB 中，允许用户使用 secret 传递参数，而无需将它们公开给目录或用户。

部分是每个条目具有以下结构的列表：

字段	描述	必填
title	规则的标题。这仅用于显示和输出目的。	Y
apb_name	与指定 secret 关联的 APB 名称。这是完全限定名称(<**registry_name**>-<**image_name**>)。	Y
secret	要从中拉取参数的 secret 名称。	Y

您可以下载并使用 *create_broker_secret.py* 文件来创建和格式化这个配置部分。

```
secrets:
- title: Database credentials
  secret: db_creds
  apb_name: dh-rhscl-postgresql-apb
```

8.11. 在代理后面运行

在代理 OpenShift Container Platform 集群内运行 OAB 时，务必要了解其核心概念，并在用于外部网络访问的代理上下文中考虑它们。

作为概述，代理本身作为集群中的 pod 运行。它具有外部网络访问权限的要求，具体取决于其 registry 的配置方式。

8.11.1. registry Adapter Whitelists

代理配置的 registry 适配器必须能够与外部 registry 通信，才能成功引导并加载远程 APB 清单。这些请求可以通过代理进行，但代理必须确保可以访问所需的远程主机。

需要白名单的主机示例：

registry Adapter Type	将主机列入白名单
rhcc	**registry.redhat.io,access.redhat.com**
dockerhub	**docker.io**

8.11.2. 使用 Ansible 配置代理成为代理

如果在初始安装过程中，您可以将 OpenShift Container Platform 集群配置为在代理后运行（请参阅" 配置全局代理选项"），当部署 OAB 时：

- 自动继承这些集群范围的代理设置，
- 生成所需的 **NO_PROXY** 列表，包括 **cidr** 字段和 **serviceNetworkCIDR**，

不需要进一步配置。

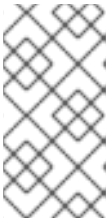
8.11.3. 手动配置代理成为代理

如果在初始安装过程中或部署代理前没有配置集群的全局代理选项，或者修改了全局代理设置，则必须手动配置代理以便通过代理进行外部访问：

1. 在尝试在代理后面运行 OAB 之前，请参阅使用 HTTP Proxies 工作，并确保集群进行相应配置，以便在代理后面运行。
   特别是，集群必须配置为 *不* 代理内部集群请求。这通常配置有 **NO_PROXY** 设置：

   ```
   .cluster.local,.svc,<serviceNetworkCIDR_value>,<master_IP>,<master_domain>,.default
   ```

   除了任何其他所需的 **NO_PROXY** 设置之外。如需了解更多详细信息，请参阅配置 NO_PROXY。



注意

部署未指定版本的 或 v1 APBs *还必须将* **172.30.0.1** 添加到其 **NO_PROXY** 列表中。v2 之前的 APB 通过 **exec** HTTP 请求从运行 APB pod 中提取凭证，而不是通过 secret 交换。除非在 OpenShift Container Platform 3.9 之前运行带有实验代理支持的代理，否则您可能不必担心。

2. 以具有 **cluster-admin** 权限的用户身份登录代理的 **DeploymentConfig** ：

```
$ oc edit dc/asb -n openshift-ansible-service-broker
```

3. 设置以下环境变量：

- **HTTP_PROXY**
- **HTTPS_PROXY**
- **NO_PROXY**

注意

如需更多信息，请参阅在 Pod 中设置代理环境变量。

4. 保存任何更新后，请重新部署 OAB 的部署配置以使更改生效：

```
$ oc rollout latest dc/asb -n openshift-ansible-service-broker
```

8.11.4. 在 Pod 中设置代理环境变量

APB pod 本身还需要通过代理进行外部访问。如果代理识别具有代理配置，它会将这些环境变量透明地应用到它生成的 APB pod。只要 APB 中使用的模块通过环境变量处理代理配置，APB 也将使用这些设置来执行其工作。

最后，APB 生成的服务可能还需要通过代理访问外部网络。如果在其自己的执行环境中识别这些环境变量，则*必须*明确编写 APB 来设置这些环境变量，或者集群操作员必须手动修改所需的服务才能将其注入其环境中。

第 9 章 将主机添加到现有集群

9.1. 添加主机

您可以通过运行 ***scaleup.yml*** playbook 添加新主机到集群。此 playbook 查询 master，为新主机生成和发布新证书，然后仅在新主机上运行配置 playbook。在运行 ***scaleup.yml*** playbook 之前，请完成所有必备的主机准备步骤。

重要

scaleup.yml playbook 仅配置新主机。它不会更新 master 服务的 ***NO_PROXY***，也不会重启 master 服务。

您必须有一个现有的清单文件，如 ***/etc/ansible/hosts***，它代表当前集群配置，才能运行 ***scaleup.yml*** playbook。如果您之前使用 **atomic-openshift-installer** 命令来运行安装，您可以检查 ***~/.config/openshift/hosts*** 查找安装程序生成的最后一个清单文件，并将该文件用作清单文件。您可以根据需要修改此文件。然后，在运行 **ansible-playbook** 时，您必须使用 **-i** 指定文件位置。

重要

有关推荐的最大节点数，请参阅集群最大值部分。

流程

1. 通过更新 **openshift-ansible** 软件包来确保您有最新的 playbook：

```
# yum update openshift-ansible
```

2. 编辑 ***/etc/ansible/hosts*** 文件，并将 **new_<host_type>** 添加到 **[OSEv3:children]** 部分。例如，要添加新节点主机，请添加 **new_nodes** ：

```
[OSEv3:children]
masters
nodes
new_nodes
```

若要添加新的 master 主机，可添加 **new_masters**。

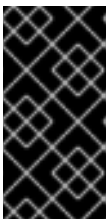
3. 创建一个 **[new_<host_type>]** 部分来为新主机指定主机信息。将此部分格式化为现有部分，如下例所示：

```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
node3.example.com openshift_node_group_name='node-config-infra'
```

如需了解更多选项，请参阅配置主机变量。

在添加新 master 时，将主机添加到 **[new_masters]** 部分和 **[new_nodes]** 部分，以确保新 master 主机是 OpenShift SDN 的一部分：

```
[masters]
master[1:2].example.com

[new_masters]
master3.example.com

[nodes]
master[1:2].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
master3.example.com
```



重要

如果您使用 **node-role.kubernetes.io/infra=true** 标签标记 master 主机，且没有其他专用基础架构节点，还必须通过向条目中添加 **openshift_schedulable=true** 明确将该主机标记为可以调度。否则，registry 和路由器 Pod 将无法放置到任何节点。

4. 更改到 playbook 目录，再运行 ***openshift_node_group.yml*** playbook。如果您的清单文件位于 ***/etc/ansible/hosts*** 默认以外的位置，请使用 **-i** 选项指定位置：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i /path/to/file] \
  playbooks/openshift-master/openshift_node_group.yml
```

这会为新节点组创建 ConfigMap，并最终为主机上的节点配置文件。

注意

运行 ***openshift_node_group.yaml*** playbook 只会更新新节点。无法运行它来更新集群中的现有节点。

5. 运行 ***scaleup.yml*** playbook。如果您的清单文件位于默认 ***/etc/ansible/hosts*** 以外的位置，请使用 **-i** 选项指定位置。

- 对于额外的节点：

```
$ ansible-playbook [-i /path/to/file] \
    playbooks/openshift-node/scaleup.yml
```

- 对于额外的 master：

```
$ ansible-playbook [-i /path/to/file] \
    playbooks/openshift-master/scaleup.yml
```

6. 如果您在集群中部署了 EFK 堆栈，请将节点标签设置为 **logging-infra-fluentd=true**：

```
# oc label node/new-node.example.com logging-infra-fluentd=true
```

7. 在 playbook 运行后，验证安装。

8. 将您在 **[new_<host_type>]** 部分定义的任何主机移动到它们的相应部分。通过移动这些主机，后续的 playbook 运行使用此清单文件正确处理节点。您可以保留空 **[new_<host_type>]** 部分。例如，添加新节点时：

```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
node3.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
```

9.2. 在现有集群中添加 ETCD 主机

您可以通过运行 etcd *扩展 playbook 来在集群中添加新的 etcd* 主机。此 playbook 查询 master，为新主机生成并分发新证书，然后仅在新主机上运行配置 playbook。在运行 etcd ***scaleup.yml*** playbook 前，请完成所有预备 主机准备步骤。

警告

这些步骤会将 Ansible 清单中的设置与集群同步。确保 Ansible 清单中显示了任何本地更改。

将 etcd 主机添加到现有集群中：

1. 通过更新 **openshift-ansible** 软件包来确保您有最新的 playbook：

```
# yum update openshift-ansible
```

2. 编辑 */etc/ansible/hosts* 文件，将 **new_<host_type>** 添加到 **[OSEv3: Child]** 组，并在 **new_<host_type>** 组下添加主机。例如，要添加新的 etcd，请添加 **new_etcd**：

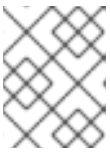
```
[OSEv3:children]
masters
nodes
etcd
new_etcd

[etcd]
etcd1.example.com
etcd2.example.com
```

```
[new_etcd]
etcd3.example.com
```

3. 更改到 playbook 目录，再运行 ***openshift_node_group.yml*** playbook。如果您的清单文件位于 ***/etc/ansible/hosts*** 默认以外的位置，请使用 **-i** 选项指定位置：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i /path/to/file] \
  playbooks/openshift-master/openshift_node_group.yml
```

这会为新节点组创建 ConfigMap，并最终为主机上的节点配置文件。



注意

运行 ***openshift_node_group.yaml*** playbook 只会更新新节点。无法运行它来更新集群中的现有节点。

4. 运行 etcd ***scaleup.yml*** playbook。如果您的清单文件位于 ***/etc/ansible/hosts*** 默认以外的位置，请使用 **-i** 选项指定位置：

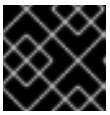
```
$ ansible-playbook [-i /path/to/file] \
  playbooks/openshift-etcd/scaleup.yml
```

5. playbook 成功完成后，验证安装。

9.3. 将现有的 MASTER 替换为 ETCD COLOCATED

在将机器迁移到不同的数据中心以及分配给它的网络和 IP 将改变时，请按照以下步骤操作。

1. 备份主 etcd 和 master 节点。



重要

确保您备份 ***/etc/etcd/*** 目录，如 etcd 备份 说明中所述。

2. 在需要替换 master 时，置备任意数量的新机器。

3. 添加或扩展集群。例如，如果要添加 3 个带有 etcd 在一起的 master，扩展 3 个 master 节点。

重要

在 OpenShift Container Platform 版本 3.11 的初始发行版本中，***scaleup.yml*** playbook 不会扩展 etcd。这个问题已在 OpenShift Container Platform 3.11.59 及更高版本中解决。

a. 添加 master。在这一流程的第 3 步，在 **[new_masters]** 和 **[new_nodes]** 中添加新数据中心的主机，运行 ***openshift_node_group.yml*** playbook，并运行 master ***scaleup.yml*** playbook。

b. 将相同的主机放在 etcd 部分中，运行 ***openshift_node_group.yml*** playbook，并运行 etcd ***scaleup.yml*** playbook。

c. 验证主机是否已添加：

```
# oc get nodes
```

d. 验证 master 主机 IP 是否已添加：

```
# oc get ep kubernetes
```

e. 验证已添加了 etcd。**ETCDCTL_API** 的值取决于所使用的版本：

```
# source /etc/etcd/etcd.conf
# ETCDCTL_API=2 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
```

f. 将 */etc/origin/master/ca.serial.txt* 从 */etc/origin/master* 目录复制到清单文件中最先列出的新 master 主机。默认情况下，这是 */etc/ansible/hosts*。

1. 删除 etcd 主机。

g. 将 */etc/etcd/ca* 目录复制到清单文件中第一个列出的新 etcd 主机中。默认情况下，这是 */etc/ansible/hosts*。

h. 从 *master-config.yaml* 文件中删除旧的 etcd 客户端：

```
# grep etcdClientInfo -A 11 /etc/origin/master/master-config.yaml
```

i. 重启 master：

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

j. 从集群中删除旧的 etcd 成员。**ETCDCTL_API** 的值取决于所使用的版本：

```
# source /etc/etcd/etcd.conf
# ETCDCTL_API=2 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
```

k. 获取以上命令的输出中的 ID，并使用 ID 删除旧的成员：

```
# etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URL member remove
1609b5a3a078c227
```

l. 通过删除 etcd pod 定义来停止旧 etcd 主机上的 etcd 服务：

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

1. 通过将定义文件从静态 pod dir /etc/origin/node/pods 中移出，关闭旧的 master 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/:
```

2. 从 HA 代理配置中删除 master 节点，该配置在原生安装过程中默认作为负载均衡器安装。

3. 弃用机器。

m. 通过删除 pod 定义并重启主机来停止 master 上的节点服务：

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

n. 删除节点资源：

```
# oc delete node
```

9.4. 迁移节点

您可以单独对节点进行迁移，或以组的形式迁移节点（例如 2 个、5 个 10 个为一组等），具体取决于您对节点的服务运行和扩展的方式。

1. 要迁移节点，在新的数据中心中置备节点使用的虚拟机。

2. 要添加新节点，请扩展基础架构。确保正确设置新节点的标签，并将您的新 API 服务器添加到负载均衡器中，并成功提供流量。

3. 评估和缩减。

   a. 在旧数据中心将当前节点标记为未调度。

   b. 撤离节点，以便其上的 pod 调度到其他节点。

   c. 验证撤离的服务在新节点上运行。

4. 删除节点。

   a. 验证节点为空，且没有运行任何进程。

   b. 停止服务或删除节点。

第 10 章 添加默认镜像流和模板

10.1. 概述

如果您在有 x86_64 架构的服务器上安装了 OpenShift Container Platform，您的集群包含了红帽提供的镜像流和模板 组，以便开发人员轻松创建新应用程序。默认情况下，集群安装过程会在 **openshift** 项目中创建这些集合，它是所有用户具有查看访问权限的默认全局项目。

如果您在使用 IBM POWER 架构的服务器上安装 OpenShift Container Platform，您可以 在集群中添加镜像流和模板。

10.2. 按订阅类型提供

根据您的红帽帐户上的活跃订阅，红帽提供了以下镜像流和模板集并被红帽支持。请联系红帽销售代表了解更多详情。

10.2.1. OpenShift Container Platform 订阅

带有一个活跃的 *OpenShift Container Platform 订阅*，并提供支持了镜像流和模板的核心集合。这包括以下技术：

类型	技术
语言和框架	• .NET Core • Node.js • Perl • PHP • Python • Ruby
数据库	• MariaDB • MongoDB • MySQL • PostgreSQL
中间件服务	• Red Hat JBoss Web Server(Tomcat) • Red Hat Single Sign-on

类型	技术
其他服务	• Jenkins • Jenkins Slaves

10.2.2. xPaaS 中间件附加订阅

xPaaS 中间件镜像支持由 *xPaaS 中间件附加组件订阅*提供，它是每个 xPaaS 产品的独立订阅。如果您的帐户上已有相关的订阅，则会为以下技术提供镜像流和模板支持：

类型	技术
中间件服务	• Red Hat JBoss A-MQ • Red Hat JBoss BPM Suite Intelligent Process Server • Red Hat JBoss BRMS Decision Server • Red Hat JBoss Data Grid • Red Hat JBoss EAP • Red Hat Fuse on OpenShift • Red Hat JBoss Data Virtualization

10.3. 开始前

在考虑执行本主题中的任务前，请通过执行以下操作之一确认这些镜像流和模板已在 OpenShift Container Platform 集群中注册：

- 登录 Web 控制台，再点 **Add to Project**。
- 使用 CLI 列出 **openshift** 项目：

```
$ oc get is -n openshift
$ oc get templates -n openshift
```

如果删除或更改了默认镜像流和模板，可以根据这里的内容自行创建默认对象。否则，不需要以下说明。

10.4. 先决条件

在创建默认镜像流和模板之前：

- 必须在 OpenShift Container Platform 安装中部署集成的容器镜像 registry 服务。
- 您必须使用 **cluster-admin** 特权运行 **oc create** 命令，因为它们在默认的 **openshift** 项目 上运行。

- 您必须已安装了 **openshift-ansible** RPM 软件包。具体步骤请参阅 软件先决条件。
- 对于 IBM POWER8 或 IBM POWER9 服务器中的内部安装，在 **openshift** 命名空间中为 **registry.redhat.io** 创建 secret。
- 为包含镜像流和模板的目录定义 shell 变量。这可显著缩短以下部分中的命令。要做到这一点：
  - 对于 x86_64 服务器中的云安装和内部安装：

```
$ IMAGESTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/x86_64/image-streams"; \
    XPAASSTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/x86_64/xpaas-streams"; \
    XPAASTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/x86_64/xpaas-templates"; \
    DBTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/x86_64/db-templates"; \
    QSTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/x86_64/quickstart-templates"
```

- 对于 IBM POWER8 或 IBM POWER9 服务器中的内部安装：

```
IMAGESTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/ppc64le/image-streams"; \
    DBTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/ppc64le/db-templates"; \
    QSTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/ppc64le/quickstart-templates"
```

10.5. 为 OPENSHIFT CONTAINER PLATFORM 镜像创建镜像流

如果您的节点主机使用 Red Hat Subscription Manager 订阅，且您想要使用使用 Red Hat Enterprise Linux(RHEL)7 的镜像的核心镜像流集合：

```
$ oc create -f $IMAGESTREAMDIR/image-streams-rhel7.json -n openshift
```

另外，要创建使用基于 CentOS 7 的镜像的核心镜像流集合：

```
$ oc create -f $IMAGESTREAMDIR/image-streams-centos7.json -n openshift
```

无法同时创建 CentOS 和 RHEL 镜像流集，因为它们使用相同的名称。要将镜像流集可供用户使用，可在另一项目中创建一个集合，或者编辑其中一个文件，并修改镜像流名称以使其唯一。

10.6. 为 XPAAS 中间件镜像创建镜像流

xPaaS 中间件镜像流为 **JBoss EAP**、**JBoss JWS**、**JBoss A-MQ**、**红帽 Fuse on OpenShift**、**Decision Server**、**JBoss Data Virtualization** 和 **JBoss Data Grid** 提供图像。它们可用于使用提供的模板为这些平台构建应用程序。

创建 xPaaS 中间件镜像流集：

```
$ oc create -f $XPAASSTREAMDIR/jboss-image-streams.json -n openshift
```

注意

访问这些镜像流引用的镜像需要相关的 xPaaS 中间件订阅。

10.7. 创建数据库服务模板

借助数据库服务模板，可以更轻松地运行可供其他组件使用的数据库镜像。对于每个数据库（MongoDB、MySQL 和 PostgreSQL）,定义了两个模板。

一个模板使用容器中的临时存储，这意味着如果容器重启，存储的数据将会丢失，例如，如果 pod 移动。此模板仅用于演示目的。

其他模板定义了用于存储的持久性卷，但 OpenShift Container Platform 安装需要配置 持久性卷。

创建数据库模板的核心集合：

```
$ oc create -f $DBTEMPLATES -n openshift
```

创建模板后，用户可以轻松地实例化各种模板，使它们快速访问数据库部署。

10.8. 创建 INSTANT APP 和 QUICKSTART 模板

Instant App 和 Quickstart 模板为正在运行的应用程序定义一组完整的对象。它们是：

- 从 GitHub 公共存储库中的源构建应用的构建配置
- 用于在 应用程序镜像构建后部署应用程序镜像的部署配置。
- 用于为应用 pod 提供负载均衡的服务。
- 提供对应用程序的外部访问的路由。

有些模板也定义了数据库部署和服务，以便应用程序能够执行数据库操作。

注意

定义数据库的模板将临时存储用于数据库内容。这些模板仅限于演示目的，因为如果数据库 pod 因任何原因重启，所有数据库数据都将丢失。

通过使用这些模板，用户可以使用 OpenShift Container Platform 提供的各种语言镜像轻松实例化完整的应用程序。它们也可以在实例化过程中自定义模板参数，以便从其自己的存储库构建源，而非示例存储库，从而为构建新应用提供了简单起点。

创建核心 Instant App 和 Quickstart 模板：

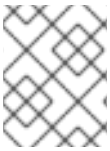
```
$ oc create -f $QSTEMPLATES -n openshift
```

还有一组模板，可使用各种 xPaaS 中间件产品（**JBoss EAP**, **JBoss JWS**, **JBoss A-MQ**, **Red Hat Fuse on OpenShift**, **Decision Server**, 和 **JBoss Data Grid**）创建应用程序，通过运行以下命令：

```
$ oc create -f $XPAASTEMPLATES -n openshift
```

注意

xPaaS 中间件模板需要 xPaaS 中间件镜像流，后者需要相关的 xPaaS 中间件订阅。



注意

定义数据库的模板将临时存储用于数据库内容。这些模板仅限于演示目的，因为如果数据库 pod 因任何原因重启，所有数据库数据都将丢失。

10.9. 下一步是什么？

创建这些工件后，开发人员现在可以登录到 web 控制台，并根据模板创建过程进行操作。可以选择任何数据库或应用程序模板，在当前项目中创建正在运行的数据库服务或应用程序。请注意，一些应用程序模板也定义了自己的数据库服务。

示例应用都构建了默认在模板中引用的 GitHub 存储库，如 **SOURCE_REPOSITORY_URL** 参数值中所示。这些存储库可以进行分叉，并且分叉可以在从模板创建时作为 **SOURCE_REPOSITORY_URL** 参数值提供。这使得开发人员能够尝试创建自己的应用程序。

您可以将开发人员定向到开发者指南中的使用 Instant App 和 Quickstart Templates 部分。

第 11 章 配置自定义证书

11.1. 概述

管理员可以为 OpenShift Container Platform API 和 Web 控制台的 公共主机名配置自定义服务证书。这可在集群安装过程中完成，也可以在安装后配置。

11.2. 配置证书链

如果使用证书链，则必须手动将所有证书链接到一个命名的证书文件中。这些证书必须按以下顺序放置：

- OpenShift Container Platform master 主机证书
- 中间 CA 证书
- Root CA 证书
- 第三方证书

要创建此证书链，请将证书串联为通用文件。您必须为每个证书运行这个命令，并确保它们按之前定义的顺序运行。

```
$ cat <certificate>.pem >> ca-chain.cert.pem
```

11.3. 在安装过程中配置自定义证书

在集群安装过程中，可以使用 **openshift_master_named_certificates** 和 **openshift_master_overwrite_named_certificates** 参数来配置自定义证书，它们可在清单文件中配置。有关使用 Ansible 配置自定义证书 的更多详细信息。

自定义证书配置参数

```
openshift_master_overwrite_named_certificates=true (1)
openshift_master_named_certificates=[{"certfile": "/path/on/host/to/crt-file", "keyfile":
"/path/on/host/to/key-file", "names": ["public-master-host.com"], "cafile": "/path/on/host/to/ca-file"}] (2)
openshift_hosted_router_certificate={"certfile": "/path/on/host/to/app-crt-file", "keyfile":
"/path/on/host/to/app-key-file", "cafile": "/path/on/host/to/app-ca-file"} (3)
```

(1) 如果您为 **openshift_master_named_certificates** 参数提供值，请将此参数设置为 **true**。

(2) 置备 主 API 证书。如有必要，将组成 证书链 的所有必需文件串联为提供给 **certFile** 参数的证书文件。

(3) 置备 路由器通配符证书。

master API 证书的参数示例：

```
openshift_master_overwrite_named_certificates=true
openshift_master_named_certificates=[{"names": ["master.148.251.233.173.nip.io"], "certfile":
"/home/cloud-user/master.148.251.233.173.nip.io.cert.pem", "keyfile": "/home/cloud-
user/master.148.251.233.173.nip.io.key.pem", "cafile": "/home/cloud-user/master-bundle.cert.pem"}]
```

路由器通配符证书的参数示例：

```
openshift_hosted_router_certificate={"certfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.cert.pem", "keyfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.key.pem", "cafile": "/home/cloud-user/ca-chain.cert.pem"}
```

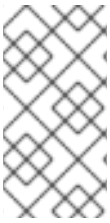
11.4. 为 WEB 控制台或 CLI 配置自定义证书

您可以通过主配置文件 的 **serviceInfo** 部分为 web 控制台和 CLI 指定自定义证书：

- **serviceInfo.namedCertificates** 部分为 web 控制台提供自定义证书。
- **serviceInfo** 部分提供 CLI 和其它 API 调用的自定义证书。

您可以以这种方式配置多个证书，每个证书都可与多个主机名、多个路由器或 OpenShift Container Platform 镜像 registry 关联。

除了 **namedCertificates** 外，还必须在 **serviceInfo.certFile** 和 **serviceInfo.keyFile** 配置部分中配置默认证书。



注意

namedCertificates 部分应仅为与 */etc/origin/master/master-config.yaml* 文件中的 **masterPublicURL** 和 **oauthConfig.assetPublicURL** 设置关联的主机名配置。将自定义服务证书用于与 **masterURL** 关联的主机名将导致 TLS 错误，因为基础架构组件将尝试使用内部 masterURL 主机联系 master API。

自定义证书配置

```
servingInfo:
  logoutURL: ""
  masterPublicURL: https://openshift.example.com:8443
  publicURL: https://openshift.example.com:8443/console/
  bindAddress: 0.0.0.0:8443
  bindNetwork: tcp4
  certFile: master.server.crt (1)
  clientCA: ""
  keyFile: master.server.key (2)
  maxRequestsInFlight: 0
  requestTimeoutSeconds: 0
  namedCertificates:
    - certFile: wildcard.example.com.crt (3)
      keyFile: wildcard.example.com.key (4)
      names:
        - "openshift.example.com"
  metricsPublicURL: "https://metrics.os.example.com/hawkular/metrics"
```

1. CLI 和其他 API 调用的证书文件的路径。
2. CLI 和其他 API 调用的密钥文件的路径。
3. OpenShift Container Platform API 和 Web 控制台的公共主机名的证书文件的路径。如有必要，将组成 证书链 的所有必需文件串联为提供给 **certFile** 参数的证书文件。

4 OpenShift Container Platform API 和 Web 控制台的公共主机名的密钥文件的路径。

Ansible 清单文件（默认为 **/etc/ansible/hosts**）中的 **openshift_master_cluster_public_hostname** 和 **openshift_master_cluster_hostname** 参数必须是不同的。如果它们相同，命名的证书将失败，您需要重新安装它们。

```
# Native HA with External LB VIPs
openshift_master_cluster_hostname=internal.paas.example.com
openshift_master_cluster_public_hostname=external.paas.example.com
```

有关在 OpenShift Container Platform 中使用 DNS 的更多信息，请参阅 DNS 安装先决条件。

这种方法允许您利用 OpenShift Container Platform 生成的自签名证书，并根据需要将自定义可信证书添加到各个组件中。

请注意，内部基础架构证书仍是自签名的，这可能被某些安全或 PKI 团队视为不当做法。但是，这里的风险最小，因为信任这些证书的唯一客户端是集群中的其他组件。所有外部用户和系统都使用自定义可信证书。

相对路径基于主配置文件的位置解析。重启服务器以获取配置更改。

11.5. 配置自定义 MASTER 主机证书

为了促进与 OpenShift Container Platform 的外部用户的可信连接，您可以置备一个与 Ansible 清单文件（默认为 **/etc/ansible/hosts**）中的 **openshift_master_cluster_public_hostname** 参数提供的域名匹配的命名证书。

您必须将此证书放在 Ansible 可访问的目录中，并在 Ansible 清单文件中添加路径，如下所示：

```
openshift_master_named_certificates=[{"certfile": "/path/to/console.ocp-c1.myorg.com.crt", "keyfile":
"/path/to/console.ocp-c1.myorg.com.key", "names": ["console.ocp-c1.myorg.com"]}]
```

如果参数值是：

- **cert** File 是包含 OpenShift Container Platform 自定义主 API 证书的文件的路径。
- **keyfile** 是包含 OpenShift Container Platform 自定义主 API 证书密钥的文件的路径。
- **name** 是集群的公共主机名。

对于 Ansible 允许，文件路径需要是系统的本地路径。证书被复制到 master 主机，并部署到 */etc/origin/master* 目录中。

在保护 registry 时，将服务主机名和 IP 地址添加到 registry 的服务器证书中。主题备用名称(SAN)必须包含以下内容：

- 两个服务主机名：

  ```
  docker-registry.default.svc.cluster.local
  docker-registry.default.svc
  ```

- 服务 IP 地址.
  例如：

```
172.30.252.46
```

使用以下命令获取容器镜像 registry 服务 IP 地址：

```
oc get service docker-registry --template='{{.spec.clusterIP}}'
```

- 公共主机名.

```
docker-registry-default.apps.example.com
```

使用以下命令获取容器镜像 registry 公共主机名：

```
oc get route docker-registry --template '{{.spec.host}}'
```

例如，服务器证书应包含类似如下的 SAN 详情：

```
X509v3 Subject Alternative Name:
            DNS:docker-registry-public.openshift.com, DNS:docker-registry.default.svc, DNS:docker-registry.default.svc.cluster.local, DNS:172.30.2.98, IP Address:172.30.2.98
```

11.6. 为默认路由器配置自定义通配符证书

您可以使用默认通配符证书配置 OpenShift Container Platform 默认路由器。默认通配符证书为在 OpenShift Container Platform 中部署的应用程序提供了一种便捷方式，而无需自定义证书，即可使用默认加密。

注意

不建议在非生产环境中使用默认通配符证书。

要配置默认通配符证书，置备一个对 ***.<app_domain>** 有效的证书，其中 **<app_domain>** 是 Ansible 清单文件 中的 **openshift_master_default_subdomain** 的值，默认为 **/etc/ansible/hosts**。置备后，将证书、密钥和 ca 证书文件放在 Ansible 主机上，并将下面这一行添加到您的 Ansible 清单文件。

```
openshift_hosted_router_certificate={"certfile": "/path/to/apps.c1-ocp.myorg.com.crt", "keyfile": "/path/to/apps.c1-ocp.myorg.com.key", "cafile": "/path/to/apps.c1-ocp.myorg.com.ca.crt"}
```

例如：

```
openshift_hosted_router_certificate={"certfile": "/home/cloud-user/star-apps.148.251.233.173.nip.io.cert.pem", "keyfile": "/home/cloud-user/star-apps.148.251.233.173.nip.io.key.pem", "cafile": "/home/cloud-user/ca-chain.cert.pem"}
```

参数值是：

- **cert** File 是包含 OpenShift Container Platform 路由器通配符证书的文件的路径。
- **keyfile** 是包含 OpenShift Container Platform 路由器通配符证书密钥的文件的路径。
- **CAfile** 是包含此密钥和证书 root CA 的文件的路径。如果使用了中间 CA，则该文件应包含中间和 root CA。

如果这些证书文件是 OpenShift Container Platform 集群的新文件，请切换到 playbook 目录，并运行 Ansible ***deploy_router.yml*** playbook，将这些文件添加到 OpenShift Container Platform 配置文件中。playbook 将证书文件添加到 ***/etc/origin/master/*** 目录中。

```
# ansible-playbook [-i /path/to/inventory] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-hosted/deploy_router.yml
```

如果 证书不是新的，例如，要更改现有证书或替换过期的证书，请切换到 playbook 目录并运行以下 playbook：

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/redeploy-certificates.yml
```

注意

要使此 playbook 运行，证书名称不得更改。如果证书名称改变，请重新运行 Ansible ***deploy_cluster.yml*** playbook，就像证书是新证书一样。

11.7. 为 IMAGE REGISTRY 配置自定义证书

OpenShift Container Platform 镜像 registry 是一个可促进构建和部署的内部服务。大多数与 registry 的通信都由 OpenShift Container Platform 中的内部组件处理。因此，您不应该需要替换 registry 服务本身使用的证书。

但是，默认情况下，registry 使用路由来允许外部系统和用户拉取和推送镜像。您可以使用带有提供给外部用户的自定义证书的*重新加密路由*，而不使用内部自签名证书。

要配置此功能，请将以下行添加到 Ansible 清单文件的 **[OSEv3:vars]** 部分，默认为 ***/etc/ansible/hosts*** 文件。指定要与 registry 路由一起使用的证书。

```
openshift_hosted_registry_routehost=registry.apps.c1-ocp.myorg.com 1
openshift_hosted_registry_routecertificates={"certfile": "/path/to/registry.apps.c1-ocp.myorg.com.crt",
"keyfile": "/path/to/registry.apps.c1-ocp.myorg.com.key", "cafile": "/path/to/registry.apps.c1-
ocp.myorg.com-ca.crt"} 2
openshift_hosted_registry_routetermination=reencrypt 3
```

1. registry 的主机名。
2. **cacert**, **cert**, 和 **key** 文件的位置。
   - **cert** File 是包含 OpenShift Container Platform registry 证书的文件的路径。
   - **keyfile** 是包含 OpenShift Container Platform registry 证书密钥的文件的路径。
   - **CAfile** 是包含此密钥和证书 root CA 的文件的路径。如果使用了中间 CA，则该文件应包含中间和 root CA。
3. 指定执行加密的位置：
   - 设置为 **reencrypt**，使用*重新加密路由*在边缘路由器中终止加密，并使用目的地提供的新证书重新加密。
   - 设置为 **passthrough** 会在目的地终止加密。目的地负责对数据进行解密。

11.8. 为负载均衡器配置自定义证书

如果您的 OpenShift Container Platform 集群使用默认的负载均衡器或企业级负载均衡器，您可以使用自定义证书使 Web 控制台和 API 使用公开签名的自定义证书。保留内部端点的现有内部证书。

以这种方式将 OpenShift Container Platform 配置为使用自定义证书：

1. 编辑 master 配置文件的 **servingInfo** 部分：

```
servingInfo:
  logoutURL: ""
  masterPublicURL: https://openshift.example.com:8443
  publicURL: https://openshift.example.com:8443/console/
  bindAddress: 0.0.0.0:8443
  bindNetwork: tcp4
  certFile: master.server.crt
  clientCA: ""
  keyFile: master.server.key
  maxRequestsInFlight: 0
  requestTimeoutSeconds: 0
  namedCertificates:
    - certFile: wildcard.example.com.crt (1)
      keyFile: wildcard.example.com.key (2)
      names:
        - "openshift.example.com"
  metricsPublicURL: "https://metrics.os.example.com/hawkular/metrics"
```

(1) OpenShift Container Platform API 和 Web 控制台的公共主机名的证书文件的路径。如有必要，将组成 证书链 的所有必需文件串联为提供给 **certFile** 参数的证书文件。

(2) OpenShift Container Platform API 和 Web 控制台的公共主机名的密钥文件的路径。

注意

仅为与 **masterPublicURL** 和 **oauthConfig.assetPublicURL** 设置关联的主机名配置 **namedCertificates** 部分。将自定义服务证书用于与 **masterURL** 关联的主机名会导致 TLS 错误，因为基础架构组件尝试使用内部 masterURL 主机联系主 API。

2. 在 Ansible 清单文件中指定 **openshift_master_cluster_public_hostname** 和 **openshift_master_cluster_hostname** 参数，默认为 */etc/ansible/hosts*。这些值必须不同。如果它们相同，则命名证书将失败。

```
# Native HA with External LB VIPs
openshift_master_cluster_hostname=paas.example.com (1)
openshift_master_cluster_public_hostname=public.paas.example.com (2)
```

(1) 为 SSL 透传配置的内部负载均衡器的 FQDN。

(2) 带有自定义(public)证书的外部负载均衡器的 FQDN。

有关负载均衡器环境的信息，请参阅 供应商和自定义证书 SSL 终止（产品） 的 OpenShift Container Platform 参考架构。

11.9. 将自定义证书重新引入到集群中

您可以将自定义 master 和自定义路由器证书重新创建到现有的 OpenShift Container Platform 集群中。

11.9.1. 将自定义 Master 证书重新引入到集群中

重新引进自定义证书 ：

1. 编辑 Ansible 清单文件，以设置 **openshift_master_overwrite_named_certificates=true**。
2. 使用 **openshift_master_named_certificates** 参数指定证书的路径。

```
openshift_master_overwrite_named_certificates=true
openshift_master_named_certificates=[{"certfile": "/path/on/host/to/crt-file", "keyfile":
"/path/on/host/to/key-file", "names": ["public-master-host.com"], "cafile": "/path/on/host/to/ca-
file"}] (1)
```

(1) 到一个 master API 证书的路径。如有必要，将组成 证书链 的所有必需文件串联为提供给 **certFile** 参数的证书文件。

3. 进入 playbook 目录并运行以下 playbook：

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/redeploy-certificates.yml
```

4. 如果您使用命名的证书：
   a. 更新每个 master 节点上的 ***master-config.yaml*** 文件中的证书参数。
   b. 重启 OpenShift Container Platform master 服务以应用更改。

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

11.9.2. 将自定义路由器证书重新引入到集群中

重新引进自定义路由器证书：

1. 编辑 Ansible 清单文件，以设置 **openshift_master_overwrite_named_certificates=true**。
2. 使用 **openshift_hosted_router_certificate** 参数指定证书的路径。

```
openshift_master_overwrite_named_certificates=true
openshift_hosted_router_certificate={"certfile": "/path/on/host/to/app-crt-file", "keyfile":
"/path/on/host/to/app-key-file", "cafile": "/path/on/host/to/app-ca-file"} (1)
```

(1) 路由器通配符证书 的路径。

3. 进入 playbook 目录并运行以下 playbook：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook playbooks/openshift-hosted/redeploy-router-certificates.yml
```

11.10. 将自定义证书与其他组件一起使用

如需有关其他组件（如日志记录和指标等）如何使用自定义证书，请参阅证书管理。

第 12 章 重新部署证书

12.1. 概述

OpenShift Container Platform 使用证书为以下组件提供安全连接：

- master（API 服务器和控制器）
- etcd
- 节点
- registry
- 路由器

您可以使用由安装程序提供的 Ansible playbook 来自动检查集群证书的过期日期。此外，还提供了 playbook 来自动备份和恢复这些证书，从而可以修复常见的证书错误。

重新部署证书的可能用例包括：

- 安装程序检测到错误的主机名，发现问题的时间太晚。
- 证书已过期，您需要更新它们。
- 您有新的 CA 并想要改用证书。

12.2. 检查证书过期

您可以使用安装程序在可配置的窗口内过期任何证书，并通知您了解已经过期的任何证书。证书到期 playbook 使用 Ansible 角色 **openshift_certificate_expiry**。

角色检查的证书包括：

- Master 和节点服务证书
- 来自 etcd secret 的路由器和 registry 服务证书
- **cluster-admin** 用户的 master、node、router、registry 和 ***kubeconfig*** 文件
- etcd 证书（包括嵌入式）

了解如何列出所有 OpenShift TLS 证书到期日期。

12.2.1. 角色变量

openshift_certificate_expiry 角色使用以下变量：

表 12.1. 核心变量

变量名称	默认值	描述
openshift_certificate_expiry_config_base	**/etc/origin**	OpenShift Container Platform 基础配置目录。

变量名称	默认值	描述
openshift_certificate_expiry_warning_days	**365**	标记将在从现在开始的这个天数后过期的证书。
openshift_certificate_expiry_show_all	**否**	在结果中包含健康状态（非过期和非警告）的证书。

表 12.2. 可选变量

变量名称	默认值	描述
openshift_certificate_expiry_generate_html_report	**否**	生成到期检查结果的 HTML 报告。
openshift_certificate_expiry_html_report_path	**$HOME/cert-expiry-report.yyyymmddTHHMMSS.html**	保存 HTML 报告的完整路径。默认为由报告文件的主目录和时间戳后缀组成。
openshift_certificate_expiry_save_json_results	**否**	将到期检查结果保存为 JSON 文件。
openshift_certificate_expiry_json_results_path	**$HOME/cert-expiry-report.yyyymmddTHHMMSS.json**	保存 JSON 报告的完整路径。默认为由报告文件的主目录和时间戳后缀组成。

12.2.2. 运行证书过期 Playbook

OpenShift Container Platform 安装程序提供一组示例证书过期 playbook，它使用了 **openshift_certificate_expiry** 角色的不同配置集合。

这些 playbook 必须与代表集群 的清单文件 一起使用。要获得最佳结果，请使用 **-v** 选项运行 **ansible-playbook**。

使用 ***easy-mode.yaml*** 示例 playbook，您可以根据需要尝试角色在对规格进行调整。此 playbook：

- 在 ***$HOME*** 目录中生成 JSON 和 stylized HTML 报告。
- 将警告窗口设置得非常大，以保证您将总能得到结果。
- 在结果中包含所有证书（健康或非）。

进入 playbook 目录并运行 ***easy-mode.yaml*** playbook：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v -i <inventory_file> \
    playbooks/openshift-checks/certificate_expiry/easy-mode.yaml
```

其他 Playbook 示例
其它示例 playbook 还可用于直接从 */usr/share/ansible/openshift-ansible/playbooks/certificate_expiry/* 目录运行。

表 12.3. 其他 Playbook 示例

文件名	使用方法
default.yaml	生成 **openshift_certificate_expiry** 角色的默认行为。
html_and_json_default_paths.yaml	在默认路径中生成 HTML 和 JSON 工件。
longer_warning_period.yaml	将到期警告窗口更改为 1500 天。
longer-warning-period-json-results.yaml	将过期警告窗口更改为 1500 天，并将结果保存为 JSON 文件。

运行其中任何示例 playbook：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v -i <inventory_file> \
    playbooks/openshift-checks/certificate_expiry/<playbook>
```

12.2.3. 输出格式

如上方所述，可以通过两种方式格式化检查报告。在用于机器解析的 JSON 格式，或作为风格的 HTML 页面，用于简单skimming。

HTML 报告
安装程序提供了 HTML 报告示例。您可以在浏览器中打开以下文件以查看它：

/usr/share/ansible/openshift-ansible/roles/openshift_certificate_expiry/examples/cert-expiry-report.html

JSON 报告
保存的 JSON 结果中有两个顶层键： **数据和 摘要**。

data 键是一个哈希值，其中键是检查每个主机的名称，值是每个对应主机上标识的证书的检查结果。

summary 键是一个哈希，用于总结证书的总数：

- 在整个集群中检查
- 这是正确的
- 在配置的警告窗口中过期
- 已过期

有关完整 JSON 报告示例，请参阅 */usr/share/ansible/openshift-ansible/roles/openshift_certificate_expiry/examples/cert-expiry-report.json*。

可以使用各种命令行工具，轻松检查 JSON 数据的摘要，是否有警告或过期时间。例如，使用 **grep** 查找单词 **summary**，并在匹配项后打印出两行 (**-A2**)：

```
$ grep -A2 summary $HOME/cert-expiry-report.yyyymmddTHHMMSS.json
   "summary": {
      "warning": 16,
      "expired": 0
```

如果可用，也可以使用 **jq** 工具查找特定值。下面的前两个示例显示了如何仅选择一个值，可以是 **warning** 或 **已过期**。第三个示例演示了如何同时选择这两个值：

```
$ jq '.summary.warning' $HOME/cert-expiry-report.yyyymmddTHHMMSS.json
16

$ jq '.summary.expired' $HOME/cert-expiry-report.yyyymmddTHHMMSS.json
0

$ jq '.summary.warning,.summary.expired' $HOME/cert-expiry-report.yyyymmddTHHMMSS.json
16
0
```

12.3. 重新部署证书

警告

重新部署 playbook 重启 control plane 服务，并可能导致集群停机。一个服务中的错误可能会导致 playbook 失败并影响集群健康状况。如果 playbook 失败，您可能需要手动解决问题并重新启动 playbook。playbook 必须按顺序完成所有任务才能成功。

使用以下 playbook 在所有相关主机上重新部署 master、etcd、node、registry 和路由器证书。您可以一次使用当前的 CA 重新部署所有这些证书，只为特定组件重新部署证书，或者自行重新部署新生成的或自定义 CA。

与证书到期 playbook 一样，这些 playbook 必须使用代表集群 的清单文件 运行。

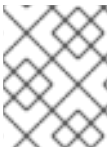
特别是，清单必须通过以下变量指定或覆盖所有主机名和 IP 地址，以便它们与当前的集群配置匹配：

- **openshift_public_hostname**
- **openshift_public_ip**
- **openshift_master_cluster_hostname**
- **openshift_master_cluster_public_hostname**

您需要的 playbook 通过以下方式提供：

-

```
# yum install openshift-ansible
```



注意

任何证书在重新部署时自动生成的任何证书的有效性（以天为单位），也可以通过 Ansible 配置。请参阅 配置证书有效期。

注意

OpenShift Container Platform CA 和 etcd 证书在 5 年后过期。签名的 OpenShift Container Platform 证书在两年后过期。

12.3.1. 使用当前 OpenShift Container Platform 和 etcd CA 重新部署所有证书

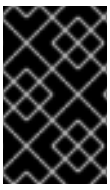
redeploy-certificates.yml playbook *不会* 重新生成 OpenShift Container Platform CA 证书。使用当前 CA 证书创建新 master、etcd、node、registry 和路由器证书，为新证书签名。

这还包括以下串行重启：

- etcd
- 主服务
- 节点服务

要使用当前的 OpenShift Container Platform CA 重新部署 master、etcd 和节点证书，请切换到 playbook 目录并运行此 playbook，指定您的清单文件：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
    playbooks/redeploy-certificates.yml
```



重要

如果 OpenShift Container Platform CA 被重新部署为 ***openshift-master/redeploy-openshift-ca.yml*** playbook，则必须将 **-e openshift_redeploy_openshift_ca=true** 添加到这个命令。

12.3.2. 重新部署新的或自定义 OpenShift Container Platform CA

openshift-master/redeploy-openshift-ca.ymlplaybook 通过生成新的 CA 证书并将更新的捆绑包分发到所有组件，包括客户端 ***kubeconfig*** 文件和可信 CA(CA-trust) 的节点数据库 (CA-trust)。

这还包括以下串行重启：

- 主服务
- 节点服务
- docker

另外，您可以在重新部署证书时 指定自定义 CA 证书，而不依赖于 OpenShift Container Platform 生成的 CA。

当主服务重启时，registry 和路由器可以继续与 master 通信，而没有重新部署，因为主设备的 serving 证书相同，并且注册表和路由器仍然有效。

要重新部署新生成的或自定义 CA：

1. 如果要使用自定义 CA，请在清单文件中设置以下变量。要使用当前的 CA，请跳过这一步。

```
# Configure custom ca certificate
# NOTE: CA certificate will not be replaced with existing clusters.
# This option may only be specified when creating a new cluster or
# when redeploying cluster certificates with the redeploy-certificates
# playbook.
openshift_master_ca_certificate={'certfile': '</path/to/ca.crt>', 'keyfile': '</path/to/ca.key>'}
```

   如果 CA 证书由中间 CA 发布，则捆绑的证书必须包含 CA 的完整链（中间和 root 证书）来验证子证书。

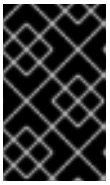
   例如：

```
$ cat intermediate/certs/intermediate.cert.pem \
    certs/ca.cert.pem >> intermediate/certs/ca-chain.cert.pem
```

2. 进入 playbook 目录并运行 ***openshift-master/redeploy-openshift-ca.yml*** playbook，指定您的清单文件：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/openshift-master/redeploy-openshift-ca.yml
```

   新的 OpenShift Container Platform CA 已就位，每当您想要在所有组件上重新部署由新 CA 签名的证书时，请使用 ***redeploy-certificates.yml*** playbook。



重要

在新的 OpenShift Container Platform CA 之后使用 ***redeploy-certificates.yml*** playbook 时，您必须在 playbook 命令中添加 **-e openshift_redeploy_openshift_ca=true**。

12.3.3. 重新部署新的 etcd CA

openshift-etcd/redeploy-ca.yml playbook 通过生成新的 CA 证书来重新部署 etcd CA 证书，并将更新的捆绑包分发到所有 etcd peers 和 master 客户端。

这还包括以下串行重启：

- etcd
- 主服务

重新部署新生成的 etcd CA：

1. 运行 ***openshift-etcd/redeploy-ca.yml*** playbook，指定您的清单文件：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
    playbooks/openshift-etcd/redeploy-ca.yml
```

重要

首次运行 **playbook/openshift-etcd/redeploy-ca.yml** playbook 后，包含 CA 符号器的压缩捆绑包将保留到 **/etc/etcd/etcd_ca.tgz** 中。因为生成新 etcd 证书需要 CA 签名者，所以备份它们非常重要。

如果 playbook 再次运行，因为预注意它不会在磁盘上覆盖这个捆绑包。要再次运行 playbook，请备份并移动该路径中的捆绑包，然后运行 playbook。

当新的 etcd CA 已就绪后，每当您希望使用由 etcd 对等和 master 客户端上的新的 etcd CA 签发的证书进行重新部署时，您可以自行决定使用 *openshift-etcd/redeploy-certificates.yml* playbook。另外，除了 etcd peers 和 master 客户端外，您还可以使用 *redeploy-certificates.yml* playbook 为 OpenShift Container Platform 组件重新部署证书。

注意

etcd 证书重新部署可能会导致将 **串行** 复制到所有 master 主机上。

12.3.4. 重新部署 Master 和 Web 控制台证书

openshift-master/redeploy-certificates.yml playbook 会重新部署 master 证书和密钥。这包括 master 服务的串行重启。

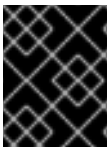
要重新部署 master 证书和密钥，请切换到 playbook 目录并运行此 playbook，指定您的清单文件：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
    playbooks/openshift-master/redeploy-certificates.yml
```

注意

如果使用命名证书，您必须更新每个 *master-config.yaml* 文件中的 命名证书参数。如有必要，将组成 证书链 的所有必需文件串联为提供给 **certFile** 参数的证书文件。

然后，重启 OpenShift Container Platform master 服务以应用更改。



重要

运行此 playbook 后，您必须通过删除包含服务证书的现有 secret 或密钥对 来重新生成任何服务签名证书或密钥对，或删除并重新添加注解到适当的服务。

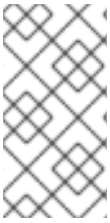
如果需要，您可以在清单文件中设置 **openshift_redeploy_service_signer=false** 参数，以跳过对服务签名证书的重新部署。如果在清单文件中设置了 **openshift_redeploy_openshift_ca=true** 和 **openshift_redeploy_service_signer=true**，则服务签名证书会在重新部署 master 证书时重新部署。如果设置了 **openshift_redeploy_openshift_ca=false** 或省略了参数，则服务 signer 证书永远不会重新部署。

12.3.5. 仅重新部署指定证书

openshift-master/redeploy-named-certificates.yml playbook 只重新部署命名的证书。运行此 playbook 也会完成 master 服务的串行重启。

要仅重新部署指定证书，请更改到包含 playbook 的目录，并运行此 playbook。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
    playbooks/openshift-master/redeploy-named-certificates.yml
```



注意

ansible 清单文件中的 _ **openshift_master_named_certificates_** 参数必须包含与 ***master-config.yaml*** 文件中同名的证书。如果更改了 **certfile** 和 **keyfile** 的名称，您必须更新每个 ***master-config.yaml*** 文件中的命名证书参数，然后重新启动 **api** 和 **controllers** 服务。带有完整 ca 链的 **cafile** 已添加到 **/etc/origin/master/ca-bundle.crt** 中。

12.3.6. 只重新部署 etcd 证书

openshift-etcd/redeploy-certificates.yml playbook 只重新部署包含 master 客户端证书的 etcd 证书。

这还包括以下串行重启：

- etcd
- 主服务。

要重新部署 etcd 证书，请切换到 playbook 目录并运行此 playbook，指定您的清单文件：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
    playbooks/openshift-etcd/redeploy-certificates.yml
```

12.3.7. 重新部署节点证书

默认情况下，节点证书有效期为一年。OpenShift Container Platform 会在节点接近到期时自动轮转节点证书。如果没有配置自动批准，您必须手动批准证书签名请求(CSR)。

如果您需要重新部署证书，因为 CA 证书已被更改，您可以使用 **-e openshift_redeploy_openshift_ca=true** 标记的 ***playbooks/redeploy-certificates.yml*** playbook。详情请参阅 使用当前 OpenShift Container Platform 和 etcd CA 重新部署所有证书 。在运行此 playbook 时，CSR 会被自动批准。

12.3.8. 只重新部署 Registry 或路由器证书

openshift-hosted/redeploy-registry-certificates.yml 和 *openshift-hosted/redeploy-router-certificates.yml* playbook 替换 registry 和路由器的安装程序创建的证书。如果自定义证书用于这些组件，请参阅 重新部署自定义 registry 或路由器证书 来手动替换它们。

12.3.8.1. 只重新部署 registry 证书

要重新部署 registry 证书，请切换到 playbook 目录并运行以下 playbook，指定您的清单文件：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
    playbooks/openshift-hosted/redeploy-registry-certificates.yml
```

12.3.8.2. 只重新部署路由器证书

要重新部署路由器证书，请切换到 playbook 目录并运行以下 playbook，指定您的清单文件：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
    playbooks/openshift-hosted/redeploy-router-certificates.yml
```

12.3.9. 重新部署自定义 registry 或路由器证书

当因为重新部署了 CA 节点被撤离时，registry 和路由器 pod 会重启。如果 registry 和路由器证书也未被新的 CA 重新部署，这可能会导致中断，因为它们无法使用其旧证书访问 master。

12.3.9.1. 手动重新部署 registry 证书

要手动重新部署 registry 证书，您必须将新的 registry 证书添加到名为 **registry-certificates** 的 secret 中，然后重新部署 registry：

1. 使用以下步骤的其余部分切换到 **default** 项目：

   ```
   $ oc project default
   ```

2. 如果您的 registry 最初是在 OpenShift Container Platform 3.1 或更早版本上创建，它可能仍在使用环境变量来存储证书（这已被淘汰使用 secret）。

   a. 运行以下命令，并查找 **OPENSHIFT_CA_DATA**、**OPENSHIFT_CERT_DATA**、**OPENSHIFT_KEY_DATA** 环境变量：

      ```
      $ oc set env dc/docker-registry --list
      ```

   b. 如果不存在，请跳过这一步。如果这样做，请创建以下 **ClusterRoleBinding** ：

      ```
      $ cat <<EOF |
      apiVersion: v1
      groupNames: null
      kind: ClusterRoleBinding
      metadata:
        creationTimestamp: null
        name: registry-registry-role
      roleRef:
        kind: ClusterRole
        name: system:registry
      subjects:
      - kind: ServiceAccount
        name: registry
        namespace: default
      userNames:
      ```

```
- system:serviceaccount:default:registry
EOF
oc create -f -
```

然后，运行以下命令来删除环境变量：

```
$ oc set env dc/docker-registry OPENSHIFT_CA_DATA- OPENSHIFT_CERT_DATA-
OPENSHIFT_KEY_DATA- OPENSHIFT_MASTER-
```

3. 本地设置以下环境变量使其更复杂：

```
$ REGISTRY_IP=`oc get service docker-registry -o jsonpath='{.spec.clusterIP}'`
$ REGISTRY_HOSTNAME=`oc get route/docker-registry -o jsonpath='{.spec.host}'`
```

4. 创建新 registry 证书：

```
$ oc adm ca create-server-cert \
   --signer-cert=/etc/origin/master/ca.crt \
   --signer-key=/etc/origin/master/ca.key \
   --hostnames=$REGISTRY_IP,docker-registry.default.svc,docker-
registry.default.svc.cluster.local,$REGISTRY_HOSTNAME \
   --cert=/etc/origin/master/registry.crt \
   --key=/etc/origin/master/registry.key \
   --signer-serial=/etc/origin/master/ca.serial.txt
```

仅从 Ansible 主机清单文件中列出的第一个 master 运行 **oc adm** 命令，默认为 */etc/ansible/hosts*。

5. 使用新的 registry 证书更新 **registry-certificates** secret：

```
$ oc create secret generic registry-certificates \
   --from-file=/etc/origin/master/registry.crt,/etc/origin/master/registry.key \
   -o json --dry-run | oc replace -f -
```

6. 重新部署 registry：

```
$ oc rollout latest dc/docker-registry
```

12.3.9.2. 手动重新部署路由器证书

要手动重新部署路由器证书，您必须将新的路由器证书添加到名为 **router-certs** 的 secret 中，然后重新部署路由器：

1. 使用以下步骤的其余部分切换到 **default** 项目：

```
$ oc project default
```

2. 如果您的路由器最初是在 OpenShift Container Platform 3.1 或更早版本创建的，它可能仍然使用环境变量来存储证书，这已被使用 service serving 证书 secret。

   a. 运行以下命令并查找 **OPENSHIFT_CA_DATA**、**OPENSHIFT_CERT_DATA**、**OPENSHIFT_KEY_DATA** 环境变量：

   -

```
$ oc set env dc/router --list
```

b. 如果这些变量存在，请创建以下 **ClusterRoleBinding** ：

```
$ cat <<EOF |
apiVersion: v1
groupNames: null
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: router-router-role
roleRef:
  kind: ClusterRole
  name: system:router
subjects:
- kind: ServiceAccount
  name: router
  namespace: default
userNames:
- system:serviceaccount:default:router
EOF
oc create -f -
```

c. 如果存在这些变量，请运行以下命令删除它们：

```
$ oc set env dc/router OPENSHIFT_CA_DATA- OPENSHIFT_CERT_DATA-
OPENSHIFT_KEY_DATA- OPENSHIFT_MASTER-
```

3. 获取证书。

- 如果您使用外部证书颁发机构(CA)为证书进行签名，请创建新的证书并根据内部流程将其提供给 OpenShift Container Platform。
- 如果使用内部 OpenShift Container Platform CA 签署证书，请运行以下命令：

重要

以下命令生成内部签名的证书。它仅被信任 OpenShift Container Platform CA 的客户端信任。

```
$ cd /root
$ mkdir cert ; cd cert
$ oc adm ca create-server-cert \
    --signer-cert=/etc/origin/master/ca.crt \
    --signer-key=/etc/origin/master/ca.key \
    --signer-serial=/etc/origin/master/ca.serial.txt \
    --hostnames='*.hostnames.for.the.certificate' \
    --cert=router.crt \
    --key=router.key \
```

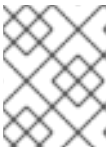
这些命令生成以下文件：

- 名为 ***router.crt*** 的新证书。

- 签名 CA 证书链 */etc/origin/master/ca.crt* 的副本。如果您使用中间 CA，则此链可以包含多个证书。
- 对应的名为 *router.key* 的私钥。

4. 创建新文件以串联生成的证书：

```
$ cat router.crt /etc/origin/master/ca.crt router.key > router.pem
```



注意

只有在使用由 OpenShift CA 签名的证书时，此步骤才有效。如果使用自定义证书，则应该使用正确的 CA 链的文件而不是 **/etc/origin/master/ca.crt**。

5. 在生成新的 secret 前，备份当前 secret：

```
$ oc get -o yaml --export secret router-certs > ~/old-router-certs-secret.yaml
```

6. 创建新 secret 以容纳新证书和密钥，并替换现有 secret 的内容：

```
$ oc create secret tls router-certs --cert=router.pem \ 1
    --key=router.key -o json --dry-run | \
    oc replace -f -
```

1 *router.pem* 是包含您生成的证书的串联的文件。

7. 重新部署路由器：

```
$ oc rollout latest dc/router
```

最初部署路由器时，会将注解添加到路由器的服务，该服务自动创建一个名为 **router-metrics-tls** 的 服务证书 secret。

要手动重新部署 **router-metrics-tls** 证书，可通过删除 secret、删除并重新添加注解到路由器服务来重新创建服务证书，然后重新部署 **router-metrics-tls** secret：

8. 从 **router** 服务中删除以下注解：

```
$ oc annotate service router \
    service.alpha.openshift.io/serving-cert-secret-name- \
    service.alpha.openshift.io/serving-cert-signed-by-
```

9. 删除现有的 **router-metrics-tls** secret。

```
$ oc delete secret router-metrics-tls
```

10. 重新添加注解：

```
$ oc annotate service router \
    service.alpha.openshift.io/serving-cert-secret-name=router-metrics-tls
```

12.4. 管理证书签名请求

集群管理员可以查看证书签名请求(CSR)并批准或拒绝它们。

12.4.1. 查看证书签名请求

您可以查看证书签名请求(CSR)的列表。

- 获取当前 CSR 列表。

```
$ oc get csr
```

- 查看 CSR 的详细信息以验证其是否有效：

```
$ oc describe csr <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

12.4.2. 批准证书签名请求

您可以使用 **oc certificate approve** 命令手动批准证书签名请求(CSR)。

- 批准 CSR：

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

- 批准所有待处理的 CSR：

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}{{end}}
{{end}}' | xargs oc adm certificate approve
```

12.4.3. 拒绝证书签名请求

您可以使用 **oc certificate deny** 命令手动拒绝证书签名请求(CSR)。

- 拒绝 CSR：

```
$ oc adm certificate deny <csr_name> 1
```

1 **<csr_name>** 是当前 CSR 列表中 CSR 的名称。

12.4.4. 配置证书签名请求的自动批准

您可以在安装集群时将以下参数添加到 Ansible 清单文件，来配置节点证书签名请求(CSR)：

```
openshift_master_bootstrap_auto_approve=true
```

添加此参数可让使用 bootstrap 凭证生成的所有 CSR，或者从之前验证的节点中批准，而无需管理员干预。

如需更多信息，请参阅配置集群变量。

第 13 章 配置身份验证和用户代理

13.1. 概述

OpenShift Container Platform master 包含内置的 OAuth 服务器。开发人员和管理员获取 OAuth 访问令牌，以完成自身的 API 身份验证。

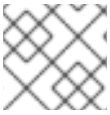
作为管理员，您可以使用 master 配置文件 配置 OAuth，以指定身份提供程序。最好 在集群安装过程中配置身份提供程序，但您可以在安装后配置它。

注意

OpenShift Container Platform 用户名不能包括 /、: 和 %。

Deny All identity provider 被默认使用，它拒绝对所有用户名和密码的访问。要允许访问，您必须选择不同的身份提供程序并配置 master 配置文件（默认为 */etc/origin/master/master-config.yaml* ）。

当您运行没有配置文件的 master 时，Allow All identity provider 会被默认使用，它允许任何非空用户名和密码登录。这可用于测试目的。要使用其他身份提供程序，或者修改任何 token, grant, 或 session options，您必须从配置文件运行 master。



注意

需要分配的角色以通过外部用户管理设置。

在修改了身份提供程序后，您必须重启 master 服务才能使更改生效：

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

13.2. 身份提供程序参数

所有身份提供程序都有四个参数：

参数	描述
name	此提供程序名称作为前缀放在提供程序用户名前，以此组成身份名称。
challenge	为 **true** 时，来自非 Web 客户端（如 CLI）的未经身份验证的令牌请求会发送 **WWW-Authenticate** challenge 标头。不支持所有身份提供程序。 要防止跨站点请求伪造 (CSRF) 攻击，只有在请求上存在 **X-CSRF-Token** 标头时，才会发送浏览器客户端基本身份验证质询。希望接收基本 **WWW-Authenticate** 质询的客户端应将此标头设置为非空值。

参数	描述
login	为 **true** 时，来自 Web 客户端（如 Web 控制台）的未经身份验证的令牌请求会重定向到由该供应商支持的登录页面。不支持所有身份提供程序。 如果您希望在重定向到身份提供程序登录前将用户发送到品牌页面，然后在 master 配置文件中设置 **oauthConfig → alwaysShowProviderSelection: true**。此提供程序选择页面 可以自定义。
mappingMethod	定义在用户登录时如何将新身份映射到用户。输入以下值之一： **claim** 默认值。使用身份的首选用户名置备用户。如果具有该用户名的用户已映射到另一身份，则失败。 **lookup** 查找现有的身份、用户身份映射和用户，但不自动置备用户或身份。这允许集群管理员手动或使用外部流程设置身份和用户。使用此方法需要手动置备用户。在使用 Lookup 映射方法时，请参阅手动置备用户。 **generate** 使用身份的首选用户名置备用户。如果拥有首选用户名的用户已映射到现有的身份，则生成一个唯一用户名。例如：**myuser2**。此方法不应与需要在 OpenShift Container Platform 用户名和身份提供程序用户名（如 LDAP 组同步）之间完全匹配的外部流程一同使用。 **add** 使用身份的首选用户名置备用户。如果已存在具有该用户名的用户，此身份将映射到现有用户，添加到该用户的现有身份映射中。如果配置了多个身份提供程序并且它们标识同一组用户并映射到相同的用户名，则需要进行此操作。

注意

在添加或更改身份提供程序时，您可以通过把 **mappingMethod** 参数设置为 **add**，将新提供程序中的身份映射到现有的用户。

13.3. 配置身份提供程序

OpenShift Container Platform 不支持为同一身份提供程序配置多个 LDAP 服务器。但是，您可以为更复杂的配置（如 LDAP 故障转移 ）扩展基本身份验证。

您可以使用这些参数在安装过程中或安装后定义身份提供程序。

13.3.1. 使用 Ansible 配置身份提供程序

对于初始集群安装，但 Deny All identity provider 被默认配置，但可在安装过程中配置 **openshift_master_identity_providers** 参数。OAuth 配置中的会话选项也可在清单文件中配置。

使用 Ansible 的用户身份供应商配置示例

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
#openshift_master_htpasswd_users={'user1': '<pre-hashed password>', 'user2': '<pre-hashed
```

```
password>'}
# or
#openshift_master_htpasswd_file=/etc/origin/master/htpasswd

# Allow all auth
#openshift_master_identity_providers=[{'name': 'allow_all', 'login': 'true', 'challenge': 'true', 'kind':
'AllowAllPasswordIdentityProvider'}]

# LDAP auth
#openshift_master_identity_providers=[{'name': 'my_ldap_provider', 'challenge': 'true', 'login': 'true',
'kind': 'LDAPPasswordIdentityProvider', 'attributes': {'id': ['dn'], 'email': ['mail'], 'name': ['cn'],
'preferredUsername': ['uid']}, 'bindDN': '', 'bindPassword': '', 'insecure': 'false', 'url':
'ldap://ldap.example.com:389/ou=users,dc=example,dc=com?uid'}]
# Configuring the ldap ca certificate (1)
#openshift_master_ldap_ca=<ca text>
# or
#openshift_master_ldap_ca_file=<path to local ca file to use> (2)

# Available variables for configuring certificates for other identity providers:
#openshift_master_openid_ca
#openshift_master_openid_ca_file (3)
#openshift_master_request_header_ca
#openshift_master_request_header_ca_file (4)
```

(1) 如果您只为 LDAP 身份提供程序在 **openshift_master_identity_providers** 参数中指定了 **'insecure': 'true'**，您可以省略 CA 证书。

(2) (3) (4) 如果您在运行 playbook 的主机上指定一个文件，则其内容将复制到 */etc/origin/master/<identity_provider_name>_<identity_provider_type>_ca.crt* 文件中。身份供应商的名称是 **openshift_master_identity_providers** 参数的值， **ldap**, **openid**, 或 **request_header**。如果您没有指定 CA 文本或本地 CA 文件的路径，您必须将 CA 证书放在该位置。如果指定多个身份提供程序，则必须手动放置每个供应商的 CA 证书。您无法更改此位置。

您可以指定多个身份提供程序。如果这样做，您必须将每个身份提供程序的 CA 证书放在 */etc/origin/master/* 目录中。例如，您可以在 **openshift_master_identity_providers** 值中包含以下供应商：

```
openshift_master_identity_providers:
- name: foo
  provider:
    kind: OpenIDIdentityProvider
    ...
- name: bar
  provider:
    kind: OpenIDIdentityProvider
    ...
- name: baz
  provider:
    kind: RequestHeaderIdentityProvider
    ...
```

您必须将这些身份提供程序的 CA 证书放在以下文件中：

- */etc/origin/master/foo_openid_ca.crt*

- */etc/origin/master/bar_openid_ca.crt*
- */etc/origin/master/baz_requestheader_ca.crt*

13.3.2. 在 master 配置文件中配置身份提供程序

您可以通过修改 master 主配置文件，配置 master 主机以使用所需的身份提供程序进行身份验证。

例 13.1. master 配置文件中的身份提供程序配置示例

```
...
oauthConfig:
  identityProviders:
  - name: htpasswd_auth
    challenge: true
    login: true
    mappingMethod: "claim"
...
```

当设置为默认的 **声明** 值时，如果身份映射到之前存在的用户名，OAuth 将失败。

13.3.2.1. 使用 lookup 映射方法时手动置备用户

使用 **lookup** 映射方法时，用户置备由外部系统通过 API 进行。通常，身份在登录时自动映射到用户。'lookup' 映射方法自动禁用这个自动映射，这需要您手动置备用户。

如需有关身份对象的更多信息，请参阅 Identity user API obejct。

如果使用 **lookup** 映射方法，请在配置身份提供程序后为每个用户执行以下步骤：

1. 如果还没有创建，创建一个 OpenShift Container Platform 用户：

   ```
   $ oc create user <username>
   ```

   例如，以下命令会创建一个 OpenShift Container Platform 用户 **bob** ：

   ```
   $ oc create user bob
   ```

2. 如果尚未创建，请创建一个 OpenShift Container Platform 身份。使用身份提供程序的名称，并在身份提供程序范围内唯一代表此身份的名称：

   ```
   $ oc create identity <identity-provider>:<user-id-from-identity-provider>
   ```

   & **lt;identity-provider** > 是 master 配置中身份提供程序的名称，如下面的相应身份提供程序部分所示。

   例如，以下命令会创建一个身份提供程序 **ldap_provider** 和身份提供程序用户名 **bob_s** 的身份。

   ```
   $ oc create identity ldap_provider:bob_s
   ```

3. 为创建用户和身份创建 user/identity 映射：

```
$ oc create useridentitymapping <identity-provider>:<user-id-from-identity-provider>
<username>
```

例如，以下命令将身份映射到用户：

```
$ oc create useridentitymapping ldap_provider:bob_s bob
```

13.3.3. 允许所有

在 **identityProviders** 小节中设置 **AllowAllPasswordIdentityProvider**，以允许任何非空用户名和密码登录。

例 13.2. 使用 AllowAllPasswordIdentityProvider 的 master 配置

```
oauthConfig:
  ...
  identityProviders:
  - name: my_allow_provider 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: AllowAllPasswordIdentityProvider
```

1. 此提供程序名称作为前缀放在提供程序用户名前，以此组成身份名称。
2. 为 **true** 时，来自非 Web 客户端（如 CLI）的未经身份验证的令牌请求会为此提供程序发送 **WWW-Authenticate** 质询标头。
3. 为 **true** 时，来自 Web 客户端（如 Web 控制台）的未经身份验证的令牌请求会重定向到由该供应商支持的登录页面。
4. 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。

13.3.4. 拒绝所有

在 **identityProviders** 小节中设置 **DenyAllPasswordIdentityProvider**，以拒绝对所有用户名和密码的访问。

例 13.3. 使用 DenyAllPasswordIdentityProvider 的 master 配置

```
oauthConfig:
  ...
  identityProviders:
  - name: my_deny_provider 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
```

```
  provider:
    apiVersion: v1
    kind: DenyAllPasswordIdentityProvider
```

1. 此提供程序名称作为前缀放在提供程序用户名前，以此组成身份名称。
2. 为 **true** 时，来自非 Web 客户端（如 CLI）的未经身份验证的令牌请求会为此提供程序发送 **WWW-Authenticate** 质询标头。
3. 为 **true** 时，来自 Web 客户端（如 Web 控制台）的未经身份验证的令牌请求会重定向到由该供应商支持的登录页面。
4. 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。

13.3.5. HTPasswd

在 **identityProviders** 小节中设置 **HTPasswdPasswordIdentityProvider**，针对使用 **htpasswd** 生成的文件验证用户名和密码。

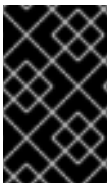
注意

htpasswd 工具程序位于 **httpd-tools** 软件包中：

```
# yum install httpd-tools
```

OpenShift Container Platform 支持 Bcrypt、SHA-1 和 MD5 加密哈希功能，而 MD5 是 **htpasswd** 的默认值。目前不支持明文、加密文本和其他散列功能。

如果修改时间更改，则无格式文件重新读取，无需重新启动服务器。



重要

由于 OpenShift Container Platform master API 现在作为静态 pod 运行，因此您必须在 */etc/origin/master/* 中创建 **HTPasswdPasswordIdentityProvider** htpasswd 文件，使其可以被容器读取。

使用 htpasswd 命令：

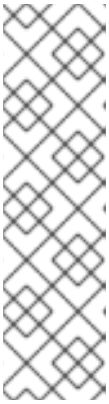
- 要创建带有用户名和散列密码的平面文件，请运行：

```
$ htpasswd -c /etc/origin/master/htpasswd <user_name>
```

然后，输入 并确认该用户的明文密码。该命令将生成散列版本的密码。

例如：

```
htpasswd -c /etc/origin/master/htpasswd user1
New password:
Re-type new password:
Adding password for user user1
```



注意

您可以包含 **-b** 选项，用于在命令行中提供密码：

```
$ htpasswd -c -b <user_name> <password>
```

例如：

```
$ htpasswd -c -b file user1 MyPassword!
Adding password for user user1
```

- 要在文件中添加或更新登录，请运行：

```
$ htpasswd /etc/origin/master/htpasswd <user_name>
```

- 要从文件中删除登录，请运行：

```
$ htpasswd -D /etc/origin/master/htpasswd <user_name>
```

例 13.4. 使用 HTPasswdPasswordIdentityProvider 的 master 配置

```
oauthConfig:
  ...
  identityProviders:
  - name: my_htpasswd_provider (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: HTPasswdPasswordIdentityProvider
      file: /etc/origin/master/htpasswd (5)
```

1. 此提供程序名称作为前缀放在提供程序用户名前，以此组成身份名称。
2. 为 **true** 时，来自非 Web 客户端（如 CLI）的未经身份验证的令牌请求会为此提供程序发送 **WWW-Authenticate** 质询标头。
3. 为 **true** 时，来自 Web 客户端（如 Web 控制台）的未经身份验证的令牌请求会重定向到由该供应商支持的登录页面。
4. 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。
5. 使用 **htpasswd** 生成的文件。

13.3.6. Keystone

Keystone 是一个提供身份、令牌、目录和策略服务的 OpenStack 项目。您可以将 OpenShift Container Platform 集群与 Keystone 集成，以便通过配置为将用户存储在内部数据库中的 OpenStack Keystone v3 服务器启用共享身份验证。此配置允许用户使用其 Keystone 凭证登录 OpenShift Container Platform。

您可以配置与 Keystone 的集成，以便 OpenShift Container Platform 的新用户基于 Keystone 用户名或者唯一 Keystone ID。使用这两种方法时，用户可以输入其 Keystone 用户名和密码进行登录。使 OpenShift Container Platform 用户基于 Keystone ID 更为安全。如果删除了某一 Keystone 用户并使用其用户名创建了新的 Keystone 用户，新用户或许能够访问旧用户的资源。

13.3.6.1. 在 master 上配置身份验证

1. 如果您有：

   - 已完成 Openshift 的安装，然后将 */etc/origin/master/master-config.yaml* 文件复制到新目录中，例如：

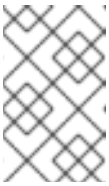
     ```
     $ cd /etc/origin/master
     $ mkdir keystoneconfig; cp master-config.yaml keystoneconfig
     ```

   - 尚未安装 OpenShift Container Platform，然后启动 OpenShift Container Platform API 服务器，指定(future)OpenShift Container Platform master 的主机名，以及一个用于存储由 start 命令创建的配置文件的目录：

     ```
     $ openshift start master --public-master=<apiserver> --write-config=<directory>
     ```

     例如：

     ```
     $ openshift start master --public-master=https://myapiserver.com:8443 --write-
     config=keystoneconfig
     ```

     注意

     如果要使用 Ansible 安装，您必须将 **identityProvider** 配置添加到 Ansible playbook 中。如果在使用 Ansible 安装后使用以下步骤手动修改配置，那么每当您重新运行安装工具或升级时，您都会丢失任何修改。

2. 编辑新的 *keystoneconfig/master-config.yaml* 文件的 **identityProviders** 小节，并复制示例 **KeystonePasswordIdentityProvider** 配置并粘贴它来替换现有的小节：

   ```
   oauthConfig:
    ...
    identityProviders:
    - name: my_keystone_provider (1)
      challenge: true (2)
      login: true (3)
      mappingMethod: claim (4)
      provider:
        apiVersion: v1
        kind: KeystonePasswordIdentityProvider
        domainName: default (5)
        url: http://keystone.example.com:5000 (6)
        ca: ca.pem (7)
        certFile: keystone.pem (8)
        keyFile: keystonekey.pem (9)
        useKeystoneIdentity: false (10)
   ```

1. 此提供程序名称作为前缀放在提供程序用户名前，以此组成身份名称。
2. 为 **true** 时，来自非 Web 客户端（如 CLI）的未经身份验证的令牌请求会为此提供程序发送 **WWW-Authenticate** 质询标头。
3. 为 **true** 时，来自 Web 客户端（如 Web 控制台）的未经身份验证的令牌请求会重定向到由该供应商支持的登录页面。
4. 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。
5. Keystone 域名。在 Keystone 中，用户名是特定于域的。只支持一个域。
6. 用于连接到 Keystone 服务器的 URL（必需）。
7. 可选：用于验证所配置 URL 的服务器证书的证书捆绑包。
8. 可选：向配置的 URL 发出请求时要出现的客户端证书。
9. 客户端证书的密钥。如果指定了 **certFile**，则需要此项。
10. 为 **true** 时，表示用户通过 Keystone ID 进行身份验证，而不是由 Keystone 用户名进行身份验证。设置为 **false** 以根据用户名进行身份验证。

3. 对 **identityProviders** 小节进行以下修改：
   a. 更改提供程序 **名称** ("my_keystone_provider")以匹配您的 Keystone 服务器。此名称作为前缀放在提供程序用户名前，以此组成身份名称。
   b. 如果需要，更改 **mappingMethod** 来控制如何在提供程序的身份和用户对象之间建立映射。
   c. 将 **domainName** 更改为 OpenStack Keystone 服务器的域名。在 Keystone 中，用户名是特定于域的。只支持一个域。
   d. 指定用于连接 OpenStack Keystone 服务器的 **url**。
   e. 另外，要根据 Keystone ID（而非 Keystone 用户名）验证用户身份，**可将KeystoneIdentity** 设置为 **true**。
   f. （可选）将 **ca** 更改为证书捆绑包，以用于验证所配置 URL 的服务器证书。
   g. （可选）将 **certFile** 更改为客户端证书，使其在向配置的 URL 发出请求时显示。
   h. 如果指定了 **certFile**，您必须将 **keyFile** 更改为客户端证书的密钥。
4. 保存您的更改并关闭该文件。
5. 启动 OpenShift Container Platform API 服务器，指定您刚才修改的配置文件：

```
$ openshift start master --config=<path/to/modified/config>/master-config.yaml
```

配置之后，系统会提示您使用其 Keystone 凭据登录 OpenShift Container Platform Web 控制台。

13.3.6.2. 使用 Keystone 身份验证创建用户

在 OpenShift Container Platform 中与外部身份验证供应商（如本例中为 Keystone）集成时，您不会在 OpenShift Container Platform 中创建用户。Keystone 是记录系统，表示用户在 Keystone 数据库中定义用户，并且配置的身份验证服务器的有效 Keystone 用户名的任何用户都可以登录。

要为 OpenShift Container Platform 添加用户，用户必须存在于 Keystone 数据库中，如果需要，必须为该用户创建新的 Keystone 帐户。

13.3.6.3. 验证用户

登录一个或多个用户后，您可以运行 **oc get users** 来查看用户列表并验证用户是否已成功创建：

例 13.5. oc get users 命令的输出

```
$ oc get users
NAME        UID                                    FULL NAME   IDENTITIES
bobsmith    a0c1d95c-1cb5-11e6-a04a-002186a28631   Bob Smith   keystone:bobsmith (1)
```

1. OpenShift Container Platform 中的身份由作为 Keystone 用户名前缀的身份提供商名称组成。

在这里，您可能需要 了解如何管理用户角色。

13.3.7. LDAP 身份验证

在 **identityProviders** 小节中设置 **LDAPPasswordIdentityProvider**，以使用简单的绑定身份验证根据 LDAPv3 服务器验证用户名和密码。

注意

如果您需要 LDAP 服务器故障转移，而不是按照以下步骤，通过 为 LDAP 故障转移配置 SSSD 来扩展基本验证方法。

在身份验证过程中，搜索 LDAP 目录中与提供的用户名匹配的条目。如果找到一个唯一匹配项，则尝试使用该条目的可分辨名称 (DN) 以及提供的密码进行简单绑定。

执行下面这些步骤：

1. 通过将配置的 **url** 中的属性和过滤器与用户提供的用户名组合来生成搜索过滤器。
2. 使用生成的过滤器搜索目录。如果搜索返回的不是一个条目，则拒绝访问。
3. 尝试使用搜索所获条目的 DN 和用户提供的密码绑定到 LDAP 服务器。
4. 如果绑定失败，则拒绝访问。
5. 如果绑定成功，则将配置的属性用作身份、电子邮件地址、显示名称和首选用户名来构建一个身份。

配置的 **url** 是 RFC 2255 URL，指定要使用的 LDAP 主机和搜索参数。URL 的语法是：

```
ldap://host:port/basedn?attribute?scope?filter
```

对于以上示例：

URL 组件	描述
ldap	对于常规 LDAP，使用 **ldap** 字符串。对于安全 LDAP (LDAPS)，改为使用 **ldaps**。
host:port	LDAP 服务器的名称和端口。LDAP 默认为 **localhost:389**，LDAPS 则默认为 **localhost:636**。
basedn	所有搜索都应从中开始的目录分支的 DN。至少，这必须是目录树的顶端，但也可指定目录中的子树。
attribute	要搜索的属性。虽然 RFC 2255 允许使用逗号分隔属性列表，但无论提供多少个属性，都仅使用第一个属性。如果没有提供任何属性，则默认使用 **uid**。建议选择一个在您使用的子树中的所有条目间是唯一的属性。
scope	搜索的范围。可以是 **one** 或 **sub**。如果未提供范围，则默认使用 **sub** 范围。
filter	有效的 LDAP 搜索过滤器。如果未提供，则默认为 **(objectClass=*)**

在进行搜索时，属性、过滤器和提供的用户名会组合在一起，创建类似如下的搜索过滤器：

```
(&(<filter>)(<attribute>=<username>))
```

例如，可考虑如下 URL：

```
ldap://ldap.example.com/o=Acme?cn?sub?(enabled=true)
```

当客户端尝试使用用户名 **bob** 连接时，生成的搜索过滤器将为 **(&(enabled=true)(cn=bob))**。

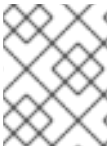
如果 LDAP 目录需要身份验证才能搜索，请指定用于执行条目搜索的 **bindDN** 和 **bindPassword**。

使用 LDAPPasswordIdentityProvider 的 master 配置

```
oauthConfig:
  ...
  identityProviders:
  - name: "my_ldap_provider" 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: LDAPPasswordIdentityProvider
      attributes:
        id: 5
        - dn
        email: 6
        - mail
        name: 7
        - cn
        preferredUsername: 8
```

```
  - uid
bindDN: "" 9
bindPassword: "" 10
ca: my-ldap-ca-bundle.crt 11
insecure: false 12
url: "ldap://ldap.example.com/ou=users,dc=acme,dc=com?uid" 13
```

1. 此提供程序名称作为前缀放在返回的用户 ID 前，以此组成身份名称。
2. 为 **true** 时，来自非 Web 客户端（如 CLI）的未经身份验证的令牌请求会为此提供程序发送 **WWW-Authenticate** 质询标头。
3. 为 **true** 时，来自 Web 客户端（如 Web 控制台）的未经身份验证的令牌请求会重定向到由该供应商支持的登录页面。
4. 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。
5. 用作身份的属性列表。使用第一个非空属性。至少需要一个属性。如果列出的属性都没有值，身份验证会失败。
6. 用作电子邮件地址的属的列表。使用第一个非空属性。
7. 用作显示名称的属性列表。使用第一个非空属性。
8. 为此身份置备用户时用作首选用户名的属性列表。使用第一个非空属性。
9. 在搜索阶段用来绑定的可选 DN。
10. 在搜索阶段用来绑定的可选密码。此值也可在 环境变量、外部文件或加密文件 中提供。
11. 用于验证所配置 URL 的服务器证书的证书捆绑包。此文件的内容复制到 */etc/origin/master/<identity_provider_name>_ldap_ca.crt* 文件中。身份提供程序名称是 **openshift_master_identity_providers** 参数的值。如果您没有指定 CA 文本或本地 CA 文件的路径，您必须将 CA 证书放在 **/etc/origin/master/** 目录中。如果指定多个身份提供程序，您必须手动将每个供应商的 CA 证书放在 **/etc/origin/master/** 目录中。您无法更改此位置。只有在清单文件中设置了 **insecure: false** 时，才会定义证书捆绑包。
12. 为 **true** 时，不会对服务器进行 TLS 连接。为 **false** 时，**ldaps://** URL 使用 TLS 进行连接，并且 **ldap://** URL 升级到 TLS。
13. RFC 2255 URL，指定要使用的 LDAP 主机和搜索参数，如上所述。



注意

要将用户列在 LDAP 集成的白名单中，请使用 **lookup** 映射方法。在允许从 LDAP 登录前，集群管理员必须为每个 LDAP 用户创建身份和用户对象。

13.3.8. 基本身份验证（远程）

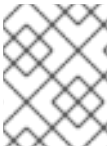
基本身份验证是一种通用后端集成机制，用户可以使用针对远程身份提供程序验证的凭证来登录 OpenShift Container Platform。

由于基本身份验证是通用的，因此您可以在高级身份验证配置中使用此身份提供程序。您可以配置 LDAP 故障转移，或使用 容器化基本身份验证 存储库作为另一个高级远程基本身份验证配置的起点。

小心

基本身份验证必须使用 HTTPS 连接到远程服务器，以防止遭受用户 ID 和密码嗅探以及中间人攻击。

配置了 **BasicAuthPasswordIdentityProvider** 后，用户将其用户名和密码发送到 OpenShift Container Platform，然后通过发出服务器对服务器请求来传递凭证作为 Basic Auth 标头来针对远程服务器验证这些凭证。这要求用户在登录期间向 OpenShift Container Platform 发送凭证。



注意

这只适用于用户名/密码登录机制，并且 OpenShift Container Platform 必须能够向远程身份验证服务器发出网络请求。

在 **identityProviders** 小节中设置 **BasicAuthPasswordIdentityProvider**，以使用 server-to-server 基本身份验证请求对远程服务器验证用户名和密码。针对受基本身份验证保护并返回 JSON 的远程 URL 验证用户名和密码。

401 响应表示身份验证失败。

非 **200** 状态或出现非空"error"键表示出现错误：

```
{"error":"Error message"}
```

200 状态并带有 **sub**（subject）键则表示成功：

```
{"sub":"userid"} 1
```

1 主体必须是经过身份验证的用户所特有的，而且必须不可修改。

成功响应可能会（可选）提供额外的数据，例如：

- 使用 **name** 键的显示名称。例如：

  ```
  {"sub":"userid", "name": "User Name", ...}
  ```

- 使用 **email** 键的电子邮件地址。例如：

  ```
  {"sub":"userid", "email":"user@example.com", ...}
  ```

- 使用 **preferred_username** 键的首选用户名。这可用在唯一不可改主体是数据库密钥或 UID 且存在更易读名称的情形中。为经过身份验证的身份置备 OpenShift Container Platform 用户时，这可用作提示。例如：

  ```
  {"sub":"014fbff9a07c", "preferred_username":"bob", ...}
  ```

13.3.8.1. 在 master 上配置身份验证

1. 如果您有：

   - 已完成 Openshift 的安装，然后将 */etc/origin/master/master-config.yaml* 文件复制到新目录中，例如：

```
$ mkdir basicauthconfig; cp master-config.yaml basicauthconfig
```

- 尚未安装 OpenShift Container Platform，然后启动 OpenShift Container Platform API 服务器，指定(future)OpenShift Container Platform master 的主机名，以及一个用于存储由 start 命令创建的配置文件的目录：

```
$ openshift start master --public-master=<apiserver> --write-config=<directory>
```

例如：

```
$ openshift start master --public-master=https://myapiserver.com:8443 --write-config=basicauthconfig
```

注意

如果要使用 Ansible 安装，您必须将 **identityProvider** 配置添加到 Ansible playbook 中。如果在使用 Ansible 安装后使用以下步骤手动修改配置，那么每当您重新运行安装工具或升级时，您都会丢失任何修改。

2. 编辑新的 *master-config.yaml* 文件的 **identityProviders** 小节，并复制 示例 **BasicAuthPasswordIdentityProvider** 配置 并粘贴它来替换现有的小节：

```
oauthConfig:
 ...
 identityProviders:
 - name: my_remote_basic_auth_provider (1)
  challenge: true (2)
  login: true (3)
  mappingMethod: claim (4)
  provider:
    apiVersion: v1
    kind: BasicAuthPasswordIdentityProvider
    url: https://www.example.com/remote-idp (5)
    ca: /path/to/ca.file (6)
    certFile: /path/to/client.crt (7)
    keyFile: /path/to/client.key (8)
```

(1) 此提供程序名称作为前缀放在返回的用户 ID 前，以此组成身份名称。

(2) 为 **true** 时，来自非 Web 客户端（如 CLI）的未经身份验证的令牌请求会为此提供程序发送 **WWW-Authenticate** 质询标头。

(3) 为 **true** 时，来自 Web 客户端（如 Web 控制台）的未经身份验证的令牌请求会重定向到由该供应商支持的登录页面。

(4) 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。

(5) 接受基本身份验证标头中凭证的 URL。

(6) 可选：用于验证所配置 URL 的服务器证书的证书捆绑包。

(7) 可选：向配置的 URL 发出请求时要出现的客户端证书。

8 客户端证书的密钥。如果指定了 **certFile**，则需要此项。

对 **identityProviders** 小节进行以下修改：

a. 将供应商**名称**设置为与您的部署的唯一且相关内容。此名称作为前缀放在返回的用户 ID 前，以此组成身份名称。

b. 如果需要，设置 **mappingMethod** 来控制如何在提供程序的身份和用户对象之间建立映射。

c. 指定 **HTTPS** URL 用于连接到接受基本身份验证标头中凭证的服务器。

d. （可选）将 **ca** 设置为证书捆绑包，以使用 以验证所配置 URL 的服务器证书，或将其留空，以使用系统信任的根证书。

e. （可选）删除或将 **certFile** 设置为客户端证书，以便在向配置的 URL 发出请求时显示。

f. 如果指定了 **certFile**，您必须将 **keyFile** 设置为客户端证书的密钥。

3. 保存您的更改并关闭该文件。

4. 启动 OpenShift Container Platform API 服务器，指定您刚才修改的配置文件：

```
$ openshift start master --config=<path/to/modified/config>/master-config.yaml
```

配置后，系统会提示您使用其基本身份验证凭证登录 OpenShift Container Platform Web 控制台。

13.3.8.2. 故障排除

最常见的问题与后端服务器网络连接相关。要进行简单调试，请在 master 上运行 **curl** 命令。要测试成功的登录，请将以下示例命令中的 **<user>** 和 **<password>** 替换为有效的凭证。要测试无效的登录，请将它们替换为错误的凭证。

```
curl --cacert /path/to/ca.crt --cert /path/to/client.crt --key /path/to/client.key -u <user>:<password> -v
https://www.example.com/remote-idp
```

成功响应

200 状态并带有 **sub**（subject）键则表示成功：

```
{"sub":"userid"}
```

subject 必须是经过身份验证的用户所特有的，而且必须不可修改。

成功响应可能会（可选）提供额外的数据，例如：

- 使用 **name** 键的显示名称：

```
{"sub":"userid", "name": "User Name", ...}
```

- 使用 **email** 键的电子邮件地址：

```
{"sub":"userid", "email":"user@example.com", ...}
```

- 使用 **preferred_username** 键的首选用户名：

```
{"sub":"014fbff9a07c", "preferred_username":"bob", ...}
```

preferred_username 键可用在唯一不可改主体是数据库密钥或 UID 且存在更易读名称的情形中。为经过身份验证的身份置备 OpenShift Container Platform 用户时，这可用作提示。

失败的响应

- **401** 响应表示身份验证失败。
- 非 **200** 状态或带有非空"error"键表示错误：**{"error":"Error message"}**

13.3.9. 请求标头（Request header）

在 **identityProviders** 小节中设置 **RequestHeaderIdentityProvider**，以标识请求标头值中的用户，如 **X-Remote-User**。它通常与设定请求标头值的身份验证代理一起使用。这与 OpenShift Enterprise 2 中的远程用户插件 是，管理员可以提供 Kerberos、LDAP 和许多其他形式的企业身份验证。

注意

您还可以将请求标头身份提供程序用于高级配置，如由社区支持的 SAML 身份验证。请注意，红帽不支持 SAML 验证。

要使用户使用这个身份提供程序进行身份验证，必须通过身份验证代理访问 **https://<master>/oauth/authorize**（及子路径）。要达到此目的，请将 OAuth 服务器配置为将 OAuth 令牌的未经身份验证的请求重定向到代理到 **https://<master>/oauth/authorize 的代理端点**。

重定向来自希望基于浏览器型登录流的客户端的未经身份验证请求：

1. 将 **login** 参数设置为 **true**。
2. 将 **provider.loginURL** 参数设置为身份验证代理 URL，该代理将验证交互式客户端并将其请求代理到 **https://<master>/oauth/authorize**。

重定向来自希望 **WWW-Authenticate** 质询的客户端的未经身份验证请求：

1. 将 **challenge** 参数设置为 **true**。
2. 将 **provider.challengeURL** 参数设置为身份验证代理 URL，该代理将验证希望 **WWW-Authenticate** 质询的客户端，然后将请求代理到 **https://<master>/oauth/authorize**。

provider.challengeURL 和 **provider.loginURL** 参数可以在 URL 的查询部分中包含以下令牌：

- **${url}** 替换为当前的 URL，进行转义以在查询参数中安全使用。
  例如：**https://www.example.com/sso-login?then=${url}**
- **${query}** 替换为当前的查询字符串，不进行转义。
  例如：**https://www.example.com/auth-proxy/oauth/authorize?${query}**

警告

如果您期望未经身份验证的请求可以访问 OAuth 服务器，则需要为此此身份提供程序设置 **clientCA** 参数，以便在检查请求标头检查用户名的标头前检查传入的请求。否则，任何向 OAuth 服务器直接请求都可通过设置请求标头来模拟该提供程序的任何身份。

使用 RequestHeaderIdentityProvider 的 master 配置

```
oauthConfig:
  ...
  identityProviders:
  - name: my_request_header_provider (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: RequestHeaderIdentityProvider
      challengeURL: "https://www.example.com/challenging-proxy/oauth/authorize?${query}" (5)
      loginURL: "https://www.example.com/login-proxy/oauth/authorize?${query}" (6)
      clientCA: /path/to/client-ca.file (7)
      clientCommonNames: (8)
      - my-auth-proxy
      headers: (9)
      - X-Remote-User
      - SSO-User
      emailHeaders: (10)
      - X-Remote-User-Email
      nameHeaders: (11)
      - X-Remote-User-Display-Name
      preferredUsernameHeaders: (12)
      - X-Remote-User-Login
```

(1) 此提供程序名称作为前缀放在请求标头中的用户名前，以此组成身份名称。

(2) **RequestHeaderIdentityProvider** 只能通过重定向到配置的 **challengeURL** 来响应请求 **WWW-Authenticate** 质询的客户端。配置的 URL 应以 **WWW-Authenticate** 质询进行响应。

(3) **RequestHeaderIdentityProvider** 只能通过重定向到配置的 **loginURL**来响应请求登录流的客户端。配置的 URL 应通过登录流做出响应。

(4) 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。

(5) 可选：将未经身份验证的 **/oauth/authorize** 请求重定向到的 URL，该请求将验证希望 **WWW-Authenticate** 质询的客户端，然后将它们代理到 **https://<master>/oauth/authorize**. **${url}** 替换为当前的 URL，进行转义以在查询参数中安全使用。**${query}** 替换为当前的查询字符串。

(6) 可选：将未经身份验证的 **/oauth/authorize** 请求重定向到的 URL，它将验证基于浏览器的客户端，然后将请求代理到 **https://<master>/oauth/authorize**。代理到 **https://<master>/oauth /authorize**

的 URL 必须以 /authorize 结尾（不含尾部斜杠），并代理子路径，以便 OAuth 批准流正常工作。**${url}** 替换为当前的 URL，进行转义以在查询参数中安全使用。**${query}** 替换为当前的查询字符串。

7 可选：PEM 编码的证书捆绑包。如果设置，则必须根据指定文件中的证书颁发机构显示并验证有效的客户端证书，然后才能检查请求标头中的用户名。

8 可选：通用名称 (**cn**) 的列表。如果设定，则必须出示带有指定列表中通用名称 (**cn**) 的有效客户端证书，然后才能检查请求标头中的用户名。如果为空，则允许任何通用名称。只能与 **clientCA** 结合使用。

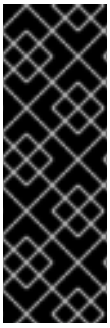
9 按顺序查找用户身份的标头名称。第一个包含值的标头被用作身份。必需，不区分大小写。

10 按顺序查找电子邮件地址的标头名称。第一个包含值的标头被用作电子邮件地址。可选，不区分大小写。

11 按顺序查找显示名称的标头名称。第一个包含值的标头被用作显示名称。可选，不区分大小写。

12 按顺序查找首选用户名的标头名称（如果与通过 **headers** 中指定的标头确定的不可变身份不同）。在置备时，第一个包含值的标头用作首选用户名。可选，不区分大小写。

Microsoft Windows 上的 SSPI 连接支持



重要

使用 Microsoft Windows 上的 SSPI 连接支持是技术预览功能。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

如需红帽技术预览功能支持范围的更多信息，请参阅 https://access.redhat.com/support/offerings/techpreview/。

从版本 3.11 开始，

oc 支持安全支持提供程序接口 (SSPI)，以允许 Microsft Windows 上的 SSO 流。如果您使用请求标头身份提供程序与支持 GSSAPI 的代理将 Active Directory 服务器连接到 OpenShift Container Platform，用户可以通过加入了域的 Microsoft Windows 计算机使用 **oc** 命令行界面来自动进行 OpenShift Container Platform 身份验证。

使用 Request 标头的 Apache 身份验证
本例在与 master 位于同一主机上配置身份验证代理。在同一主机上让代理和 master 提供方便方便的，可能不适用于您的环境。例如，如果您已在 master 上运行 路由器，端口 443 不可用。

另请务必注意，尽管此参考配置使用 Apache 的 **mod_auth_gssapi**，但这种引用配置并不是必需的，且如果满足以下要求，您可以轻松地使用其他代理：

1. 阻断来自客户端请求的 **X-Remote-User** 标头以防止欺骗。
2. 在 **RequestHeaderIdentityProvider** 配置中强制进行客户端证书验证 。
3. 使用质询流为所有身份验证请求设置 **X-Csrf-Token** 标头。
4. 只有 **/oauth/authorize** 端点及其子路径应代理，并且不应重写重定向，从而使后端服务器可以将客户端发送到正确的位置。

5. 代理到 **https://<master>/oauth/authorize** 的 URL 必须以 **/authorize** 结尾（不含尾部斜杠）。例如：

   - **https://proxy.example.com/login-proxy/authorize?… → https://<master>/oauth/authorize?…**

6. 代理到 **https://<master>/oauth/authorize** 的 URL 的子路径必须代理到 **https://<master>/oauth/authorize 的** 子路径。例如：

   - **https://proxy.example.com/login-proxy/authorize/approve?… → https://<master>/oauth/authorize/approve?…**

安装先决条件

1. 通过 Optional channel 获得 **mod_auth_gssapi** 模块。安装以下软件包：

```
# yum install -y httpd mod_ssl mod_session apr-util-openssl mod_auth_gssapi
```

2. 生成用于验证提交可信标头的请求的 CA。此 CA 应在 master 身份提供程序配置中用作 **clientCA** 的文件名。

```
# oc adm ca create-signer-cert \
  --cert='/etc/origin/master/proxyca.crt' \
  --key='/etc/origin/master/proxyca.key' \
  --name='openshift-proxy-signer@1432232228' \
  --serial='/etc/origin/master/proxyca.serial.txt'
```

注意

oc adm ca create-signer-cert 命令生成有效 5 年的证书。这可以通过 **--expire-days** 选项进行修改，但出于安全原因，建议不要超过这个值。

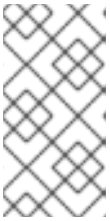
仅从 Ansible 主机清单文件中列出的第一个 master 运行 **oc adm** 命令，默认为 */etc/ansible/hosts*。

3. 示例：为代理生成客户端证书这可以通过任何 x509 证书工具完成。为方便起见，可以使用 **oc adm** CLI：

```
# oc adm create-api-client-config \
  --certificate-authority='/etc/origin/master/proxyca.crt' \
  --client-dir='/etc/origin/master/proxy' \
  --signer-cert='/etc/origin/master/proxyca.crt' \
  --signer-key='/etc/origin/master/proxyca.key' \
  --signer-serial='/etc/origin/master/proxyca.serial.txt' \
  --user='system:proxy' (1)

# pushd /etc/origin/master
# cp master.server.crt /etc/pki/tls/certs/localhost.crt (2)
# cp master.server.key /etc/pki/tls/private/localhost.key
# cp ca.crt /etc/pki/CA/certs/ca.crt
# cat proxy/system\:proxy.crt \
  proxy/system\:proxy.key > \
  /etc/pki/tls/certs/authproxy.pem
# popd
```

1. 用户名可以是任何内容，但为它赋予一个描述性名称，因为它将在日志中出现时很有用。

2. 在与 master 不同的主机名中运行身份验证代理时，生成与主机名匹配的证书非常重要，而不必使用如上所示的默认 master 证书。*/etc/origin/master/master-config.yaml* 文件中的 **masterPublicURL** 的值需要包括在为 **SSLCertificateFile** 指定的证书的 **X509v3 Subject Alternative Name** 中。如果需要创建新证书，可以使用 **oc adm ca create-server-cert** 命令。



注意

oc adm create-api-client-config 命令生成有效期为两年的证书。这可以通过 **--expire-days** 选项进行修改，但出于安全原因，建议不要超过这个值。仅从 Ansible 主机清单文件中列出的第一个 master 运行 **oc adm** 命令，默认为 */etc/ansible/hosts*。

配置 Apache

此代理不需要驻留在与 master 相同的主机上。它使用客户端证书连接到 master，该 master 配置为信任 **X-Remote-User** 标头。

1. 为 Apache 配置创建证书。您通过 **SSLProxyMachineCertificateFile** 参数值指定的证书是用于向服务器验证代理的代理的客户端证书。它必须使用 **TLS Web 客户端身份验证**作为扩展密钥类型。

2. 创建 Apache 配置文件。使用以下模板来提供所需设置和值：

重要

仔细检查模板的内容，并根据您的环境自定义其相应的内容。

```
LoadModule request_module modules/mod_request.so
LoadModule auth_gssapi_module modules/mod_auth_gssapi.so
# Some Apache configurations might require these modules.
# LoadModule auth_form_module modules/mod_auth_form.so
# LoadModule session_module modules/mod_session.so

# Nothing needs to be served over HTTP.  This virtual host simply redirects to
# HTTPS.
<VirtualHost *:80>
  DocumentRoot /var/www/html
  RewriteEngine              On
  RewriteRule     ^(.*)$     https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
  # This needs to match the certificates you generated.  See the CN and X509v3
  # Subject Alternative Name in the output of:
  # openssl x509 -text -in /etc/pki/tls/certs/localhost.crt
  ServerName www.example.com

  DocumentRoot /var/www/html
  SSLEngine on
  SSLCertificateFile /etc/pki/tls/certs/localhost.crt
  SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
```

```
  SSLCACertificateFile /etc/pki/CA/certs/ca.crt

  SSLProxyEngine on
  SSLProxyCACertificateFile /etc/pki/CA/certs/ca.crt
  # It's critical to enforce client certificates on the Master.  Otherwise
  # requests could spoof the X-Remote-User header by accessing the Master's
  # /oauth/authorize endpoint directly.
  SSLProxyMachineCertificateFile /etc/pki/tls/certs/authproxy.pem

  # Send all requests to the console
  RewriteEngine              On
  RewriteRule     ^/console(.*)$     https://%{HTTP_HOST}:8443/console$1 [R,L]

  # In order to using the challenging-proxy an X-Csrf-Token must be present.
  RewriteCond %{REQUEST_URI} ^/challenging-proxy
  RewriteCond %{HTTP:X-Csrf-Token} ^$ [NC]
  RewriteRule ^.* - [F,L]

  <Location /challenging-proxy/oauth/authorize>
     # Insert your backend server name/ip here.
     ProxyPass https://[MASTER]:8443/oauth/authorize
     AuthName "SSO Login"
     # For Kerberos
     AuthType GSSAPI
     Require valid-user
     RequestHeader set X-Remote-User %{REMOTE_USER}s

     GssapiCredStore keytab:/etc/httpd/protected/auth-proxy.keytab
     # Enable the following if you want to allow users to fallback
     # to password based authntication when they do not have a client
     # configured to perform kerberos authentication
     GssapiBasicAuth On

     # For ldap:
     # AuthBasicProvider ldap
     # AuthLDAPURL "ldap://ldap.example.com:389/ou=People,dc=my-domain,dc=com?uid?
sub?(objectClass=*)"

     # It's possible to remove the mod_auth_gssapi usage and replace it with
     # something like mod_auth_mellon, which only supports the login flow.
   </Location>

   <Location /login-proxy/oauth/authorize>
   # Insert your backend server name/ip here.
   ProxyPass https://[MASTER]:8443/oauth/authorize

     AuthName "SSO Login"
     AuthType GSSAPI
     Require valid-user
     RequestHeader set X-Remote-User %{REMOTE_USER}s env=REMOTE_USER

     GssapiCredStore keytab:/etc/httpd/protected/auth-proxy.keytab
     # Enable the following if you want to allow users to fallback
     # to password based authntication when they do not have a client
     # configured to perform kerberos authentication
     GssapiBasicAuth On
```

```
    ErrorDocument 401 /login.html
  </Location>

</VirtualHost>

RequestHeader unset X-Remote-User
```

配置 master
/etc/origin/master/master-config.yaml 文件中的 **identityProviders** 小节必须更新：

```
identityProviders:
- name: requestheader
  challenge: true
  login: true
  provider:
    apiVersion: v1
    kind: RequestHeaderIdentityProvider
    challengeURL: "https://[MASTER]/challenging-proxy/oauth/authorize?${query}"
    loginURL: "https://[MASTER]/login-proxy/oauth/authorize?${query}"
    clientCA: /etc/origin/master/proxyca.crt
    headers:
    - X-Remote-User
```

重启服务
最后，重启以下服务：

```
# systemctl restart httpd
# master-restart api
# master-restart controllers
```

验证配置

1. 通过绕过代理来测试。如果您提供正确的客户端证书和标头，则可以请求令牌：

```
# curl -L -k -H "X-Remote-User: joe" \
  --cert /etc/pki/tls/certs/authproxy.pem \
  https://[MASTER]:8443/oauth/token/request
```

2. 如果没有提供客户端证书，请求应该被拒绝：

```
# curl -L -k -H "X-Remote-User: joe" \
  https://[MASTER]:8443/oauth/token/request
```

3. 这应该会显示指向配置的 **challengeURL**（带有额外查询参数）的重定向：

```
# curl -k -v -H 'X-Csrf-Token: 1' \
  '<masterPublicURL>/oauth/authorize?client_id=openshift-challenging-
client&response_type=token'
```

4. 这会显示带有 **WWW-Authenticate** 基本质询、协商质询或两个质询都有的 401 响应：

```
#  curl -k -v -H 'X-Csrf-Token: 1' \
   '<redirected challengeURL from step 3 +query>'
```

5. 测试在使用 Kerberos ticket 并不使用 Kerberos ticket 的情况下登录到 **oc** 命令行：

   a. 如果您使用 **kinit** 生成了 Kerberos ticket，请将其销毁：

   ```
   # kdestroy -c cache_name 1
   ```

   1 提供 Kerberos 缓存的名称。

   b. 使用您的 Kerberos 凭证登录到 **oc** ：

   ```
   # oc login
   ```

   在提示符后输入您的 Kerberos 用户名和密码。

   c. 从 **oc** 命令行注销：

   ```
   # oc logout
   ```

   d. 使用您的 Kerberos 凭证获得一个 ticket：

   ```
   # kinit
   ```

   在提示符后输入您的 Kerberos 用户名和密码。

   e. 确认您可以登录到 **oc** 命令行：

   ```
   # oc login
   ```

   如果配置正确，您会在不需要单独输入凭证的情况下成功登录。

13.3.10. GitHub 和 GitHub Enterprise

GitHub 使用 OAuth，您可以集成 OpenShift Container Platform 集群以使用该 OAuth 身份验证。OAuth 可以协助 OpenShift Container Platform 和 GitHub 或 GitHub Enterprise 之间的令牌交换流。

您可以使用 GitHub 集成来连接 GitHub 或 GitHub Enterprise。对于 GitHub Enterprise 集成，您必须提供实例的 **hostname**，并可选择提供要在服务器请求中使用的 **ca** 证书捆绑包。

注意

除非有所注明，否则以下步骤同时适用于 GitHub 和 GitHub Enterprise。

配置 GitHub 身份验证后，用户可使用 GitHub 凭证登录 OpenShift Container Platform。为防止具有任何 GitHub 用户 ID 的任何人登录 OpenShift Container Platform 集群，您可以将访问权利限制给仅属于特定 GitHub 组织的用户。

13.3.10.1. 在 GitHub 上注册应用程序

1. 注册应用程序：

   - 对于 GitHub，点击 Settings → Developer settings → Register a new OAuth application。

- 对于 GitHub Enterprise，前往 GitHub Enterprise 主页，然后点击 **Settings → Developer settings → Register a new application**。

2. 输入应用程序名称，如 **My OpenShift Install**。
3. 输入主页 URL，如 **https://myapiserver.com:8443**。
4. （可选）输入应用程序描述。
5. 输入授权回调 URL，其中 URL 的末尾包含身份提供程序 **名称**，该名称在 master 配置文件（该配置文件在下一节中配置）的 **identityProviders** 小节中定义：

```
<apiserver>/oauth2callback/<identityProviderName>
```

例如：

```
https://myapiserver.com:8443/oauth2callback/github/
```

6. 点击 **Register application**。Github 会提供客户端 ID 和客户端 Secret。保持此窗口打开，以便您可以复制这些值并将其粘贴到主配置文件中。

13.3.10.2. 在 master 上配置身份验证

1. 如果您有：

- 已安装 OpenShift Container Platform，然后将 */etc/origin/master/master-config.yaml* 文件复制到新目录中，例如：

```
$ cd /etc/origin/master
$ mkdir githubconfig; cp master-config.yaml githubconfig
```

- 尚未安装 OpenShift Container Platform，然后启动 OpenShift Container Platform API 服务器，指定(future)OpenShift Container Platform master 的主机名，以及一个用于存储由 start 命令创建的配置文件的目录：

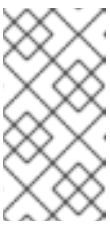
```
$ openshift start master --public-master=<apiserver> --write-config=<directory>
```

例如：

```
$ openshift start master --public-master=https://myapiserver.com:8443 --write-config=githubconfig
```

注意

如果要使用 Ansible 安装，您必须将 **identityProvider** 配置添加到 Ansible playbook 中。如果在使用 Ansible 安装后使用以下步骤手动修改配置，那么每当您重新运行安装工具或升级时，您都会丢失任何修改。



注意

使用 **openshift 自行启动 master** 将自动检测主机名，但 GitHub 必须能够重定向到您在注册应用程序时指定的准确主机名。因此，您无法自动检测 ID，因为它可能会重定向到错误的地址。反之，您必须指定 Web 浏览器用来与 OpenShift Container Platform 集群交互的主机名。

2. 编辑新的 *master-config.yaml* 文件的 **identityProviders** 小节，并复制示例 **GitHubIdentityProvider** 配置并粘贴它来替换现有的小节：

```
oauthConfig:
 ...
 identityProviders:
 - name: github (1)
  challenge: false (2)
  login: true (3)
  mappingMethod: claim (4)
  provider:
   apiVersion: v1
   kind: GitHubIdentityProvider
   ca: ... (5)
   clientID: ... (6)
   clientSecret: ... (7)
   hostname: ... (8)
   organizations: (9)
   - myorganization1
   - myorganization2
   teams: (10)
   - myorganization1/team-a
   - myorganization2/team-b
```

(1) 此提供程序名称作为前缀放在 GitHub 数字用户 ID 前，以此组成身份名称。它还可用来构建回调 URL。

(2) **GitHubIdentityProvider** 无法用于发送 **WWW-Authenticate** 质询。

(3) 为 **true** 时，来自 Web 客户端（如 Web 控制台）的未经身份验证的令牌请求将重定向到 GitHub 以进行登录。

(4) 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。

(5) 对于 GitHub Enterprise，CA 是向服务器发出请求时使用的可选的可信证书颁发机构捆绑包。省略此参数以使用默认系统 root 证书。对于 GitHub，请省略此参数。

(6) 注册的 GitHub OAuth 应用程序的客户端 ID。应用程序必须配置有 <**master>/oauth2callback/<identityProviderName&gt**; 的回调 URL。

(7) GitHub 发布的客户端 secret。此值也可在 环境变量、外部文件或加密文件 中提供。

(8) 对于 GitHub Enterprise，您必须提供实例的主机名，如 **example.com**。这个值必须与 */setup/settings* 文件中的 GitHub Enterprise **hostname** 值匹配，且不可包括端口号。对于 GitHub，请省略此参数。

(9) 可选的组织列表。如果指定，只有至少是一个所列组织成员的 GitHub 用户才能登录。如果在 **clientID** 中配置的 GitHub OAuth 应用程序不归该组织所有，则组织所有者必须授予第三方访问权限才能使用此选项。这可以在组织管理员第一次登录 GitHub 时完成，也可以在 GitHub 组织设置中完成。不可与 **teams** 字段结合使用。

(10) 可选的团队列表。如果指定，只有是至少一个列出团队的成员的 GitHub 用户才能登录。如果在 **clientID** 中配置的 GitHub OAuth 应用程序不归该团队的组织所有，则组织所有者必须授予第三方访问权限才能使用此选项。这可以在组织管理员第一次登录 GitHub 时完成，也可以在 GitHub 组织设置中完成。不可与 **organizations** 字段结合使用。

3. 对 **identityProviders** 小节进行以下修改：

   a. 更改供应商**名称**，以匹配您在 GitHub 上配置回调 URL。
   例如，如果您将回调 URL 定义为
   https://myapiserver.com:8443/oauth2callback/github/，则 **name** 必须是 **github**。

   b. 将 **clientID** 更改为您之前注册的 GitHub 中的客户端 ID。

   c. 将 **clientSecret** 改为 之前注册的 GitHub 中的 Client Secret。

   d. 更改 **organizations** 或 **teams**，使其包含一个或多个 GitHub 机构或团队的列表，用户必须具有成员资格才能进行身份验证。如果指定，只有至少是一个列出的机构或团队的成员的 GitHub 用户才能登录。如果未指定，则具有有效 GitHub 帐户的任何人都可以登录。

4. 保存您的更改并关闭该文件。

5. 启动 OpenShift Container Platform API 服务器，指定您刚才修改的配置文件：

```
$ openshift start master --config=<path/to/modified/config>/master-config.yaml
```

配置后，任何登录 OpenShift Container Platform Web 控制台的用户都将提示使用其 GitHub 凭证登录。首次登录时，用户必须点击 **授权应用程序** 来允许 GitHub 使用其用户名、密码和机构成员资格。然后，用户会被重新重定向到 Web 控制台。

13.3.10.3. 使用 GitHub 身份验证创建用户

当与外部身份验证供应商（如 GitHub）集成时，不会在 OpenShift Container Platform 中创建用户。GitHub 或 GitHub Enterprise 是记录系统，即用户由 GitHub 定义，以及属于指定机构的任何用户可以登录。

要将用户添加到 OpenShift Container Platform，您必须将该用户添加到 GitHub 或 GitHub Enterprise 上的批准机构中，如果需要为用户创建新的 GitHub 帐户。

13.3.10.4. 验证用户

登录一个或多个用户后，您可以运行 **oc get users** 来查看用户列表并验证用户是否已成功创建：

例 13.6. oc get users 命令的输出

```
$ oc get users
NAME       UID                                    FULL NAME   IDENTITIES
bobsmith   433b5641-066f-11e6-a6d8-acfc32c1ca87   Bob Smith   github:873654 (1)
```

(1) OpenShift Container Platform 中的身份由身份提供程序名称和 GitHub 的内部数字用户 ID 组成。这样，如果用户更改 GitHub 用户名或电子邮件，他们仍然可以登录到 OpenShift Container Platform，而不依赖于附加到 GitHub 帐户的凭证。这会创建一个稳定的登录。

在这里，您可能需要了解如何 控制用户角色。

13.3.11. GitLab

在 **identityProviders** 小节中设置 **GitLabIdentityProvider**，以使用 GitLab.com 或任何其他 GitLab 实例作为身份提供程序。如果使用 GitLab 版本 7.7.0 到 11.0，您可以使用 OAuth 集成进行连接。如果使用 GitLab 版本 11.1 或更高版本，您可以使用 OpenID Connect (OIDC) 进行连接，而不使用 OAuth。

例 13.7. 使用 GitLabIdentityProvider的 master 配置

```
oauthConfig:
  ...
  identityProviders:
  - name: gitlab 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: GitLabIdentityProvider
      legacy: 5
      url: ... 6
      clientID: ... 7
      clientSecret: ... 8
      ca: ... 9
```

1. 此提供程序名称作为前缀放在 GitLab 数字用户 ID 前，以此组成身份名称。它还可用来构建回调 URL。
2. 为 **true** 时，来自非 Web 客户端（如 CLI）的未经身份验证的令牌请求会为此提供程序发送 **WWW-Authenticate** 质询标头。这将使用 Resource Owner Password Credentials 授权流从 GitLab 获取访问令牌。
3. 为 **true** 时，来自 Web 客户端（如 Web 控制台）的未经身份验证的令牌请求将重定向到 GitLab 以进行登录。
4. 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。
5. 决定是否使用 OAuth 或 OIDC 作为身份验证供应商。设置为 **true**，以使用 OAuth 和 **false** 来使用 OIDC。您必须使用 GitLab.com 或 GitLab 版本 11.1 或更高版本使用 OIDC。如果没有提供值，OAuth 用于连接 GitLab 实例，并使用 OIDC 连接到 GitLab.com。
6. GitLab 提供程序的主机 URL。这可以是 **https://gitlab.com/** 或其他自托管 GitLab 实例。
7. 注册的 GitLab OAuth 应用程序的客户端 ID 。应用程序必须配置有 < **master>/oauth2callback/<identityProviderName&gt**; 的回调 URL。
8. GitLab 发布的客户端 secret。此值也可在 环境变量、外部文件或加密文件 中提供。
9. CA 是一个可选的可信证书颁发机构捆绑包，用于在向 GitLab 实例发出请求时使用。若为空，则使用默认的系统根证书。

13.3.12. Google

在 **identityProviders** 小节中将 **GoogleIdentityProvider** 设置为将 Google 用作身份提供程序，使用 Google 的 OpenID Connect 集成。

注意

使用 Google 作为身份提供程序要求用户使用 **<master>/oauth/token/request** 来获取令牌，以便用于命令行工具。

警告

使用 Google 作为身份提供程序时，任何 Google 用户都能与您的服务器进行身份验证。您可以使用 **hostedDomain** 配置属性将身份验证限制到特定托管域的成员，如下所示。

例 13.8. 使用 GoogleIdentityProvider进行 master 配置

```
oauthConfig:
  ...
  identityProviders:
  - name: google (1)
    challenge: false (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: GoogleIdentityProvider
      clientID: ... (5)
      clientSecret: ... (6)
      hostedDomain: "" (7)
```

1. 此提供程序名称作为前缀放在 Google 数字用户 ID 前，以此组成身份名称。它还可用来构建的重定向 URL。
2. **GoogleIdentityProvider** 无法用于发送 **WWW-Authenticate** 质询。
3. 当为 **true** 时，来自 web 客户端（如 Web 控制台）的未经身份验证的令牌请求会被重定向到 Google 进行登录。
4. 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。
5. 注册的 Google 项目 的客户端 ID 。项目必须配置有重定向 URI < **master>/oauth2callback/<identityProviderName>**。
6. Google 发布的客户端 secret。此值也可在 环境变量、外部文件或加密文件 中提供。
7. 可选的 hosted domain 以限制登录帐户。如果为空，任何 Google 帐户都可进行身份验证。

13.3.13. OpenID 连接

在 **identityProviders** 小节中设置 **OpenIDIdentityProvider**，以使用授权代码流与 OpenID Connect 身份提供程序 集成。

您可以将 Red Hat Single Sign-On 配置 为 OpenShift Container Platform 的 OpenID Connect 身份提供程序。

注意

不支持 **ID Token** 和 **UserInfo** 解密。

默认情况下，需要 **openid** 范围。如果必要，可在 **extraScopes** 字段中指定额外的范围。

声明可读取自从 OpenID 身份提供程序返回的 JWT **id_token**；若有指定，也可读取自从 **UserInfo** URL 返回的 JSON。

必须至少配置一个声明，以用作用户的身份。标准的身份声明是 **sub**。

您还可以指定将哪些声明用作用户的首选用户名、显示名称和电子邮件地址。如果指定了多个声明，则使用第一个带有非空值的声明。标准的声明是：

sub	“subject identifier”的缩写。 用户在签发者处的远程身份。
preferred_username	置备用户时的首选用户名。用户希望使用的简写名称，如 **janedoe**。通常，与身份验证系统中用户的登录或用户名对应的值，如用户名或电子邮件。
email	电子邮件地址。
name	显示名称。

如需更多信息，请参阅 OpenID 声明文档。

注意

使用 OpenID Connect 身份提供程序要求用户使用 **<master>/oauth/token/request** 来获取令牌，以便用于命令行工具。

使用 OpenIDIdentityProvider的标准 master 配置

```
oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: OpenIDIdentityProvider
      clientID: ... (5)
      clientSecret: ... (6)
      claims:
```

```
      id: 7
      - sub
      preferredUsername:
      - preferred_username
      name:
      - name
      email:
      - email
    urls:
      authorize: https://myidp.example.com/oauth2/authorize 8
      token: https://myidp.example.com/oauth2/token 9
```

1. 此提供程序名称作为前缀放在身份声明值前，以此组成身份名称。它还可用来构建的重定向 URL。
2. 为 **true** 时，来自非 Web 客户端（如 CLI）的未经身份验证的令牌请求会为此提供程序发送 **WWW-Authenticate** 质询标头。这要求 OpenID 供应商支持 Resource Owner Password Credentials 授权流。
3. 当为 **true** 时，来自 Web 客户端（如 Web 控制台）的未经身份验证的令牌请求会重定向到要登录的授权 URL。
4. 控制如何在此提供程序的身份和用户对象之间建立映射，如上 所述。
5. 在 OpenID 提供程序中注册的客户端的客户端 ID。该客户端必须能够重定向到 **<master>/oauth2callback/<identityProviderName>**。
6. 客户端机密。此值也可在 环境变量、外部文件或加密文件 中提供。
7. 用作身份的声明的列表。使用第一个非空声明。至少需要一个声明。如果列出的声明都没有值，身份验证会失败。例如，这使用返回的 **id_token** 中的 **sub** 声明的值作为用户的身份。
8. OpenID spec 中描述的 授权端点。必须使用 **https**。
9. OpenID spec 中描述的令牌端点。必须使用 **https**。

也可以指定自定义证书捆绑包、额外范围、额外授权请求参数和 **userInfo** URL：

例 13.9. 使用 OpenIDIdentityProvider的完整 Master 配置

```
oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect
    challenge: false
    login: true
    mappingMethod: claim
    provider:
      apiVersion: v1
      kind: OpenIDIdentityProvider
      clientID: ...
      clientSecret: ...
      ca: my-openid-ca-bundle.crt 1
      extraScopes: 2
      - email
      - profile
```

```
    extraAuthorizeParameters: (3)
      include_granted_scopes: "true"
    claims:
      id: (4)
      - custom_id_claim
      - sub
      preferredUsername: (5)
      - preferred_username
      - email
      name: (6)
      - nickname
      - given_name
      - name
      email: (7)
      - custom_email_claim
      - email
    urls:
      authorize: https://myidp.example.com/oauth2/authorize
      token: https://myidp.example.com/oauth2/token
      userInfo: https://myidp.example.com/oauth2/userinfo (8)
```

1. 用于验证所配置 URL 的服务器证书的证书捆绑包。如果为空，则使用系统可信根。
2. 除 **openid** 范围外的可选请求范围列表，在授权令牌请求期间使用。
3. 添加至授权令牌请求的附加参数的可选映射。
4. 用作身份的声明的列表。使用第一个非空声明。至少需要一个声明。如果列出的声明都没有值，身份验证会失败。
5. 为此身份置备用户时用作首选用户名的声明的列表。使用第一个非空声明。
6. 用作显示名称的声明列表。使用第一个非空声明。
7. 用作电子邮件地址的声明列表。使用第一个非空声明。
8. OpenID spec 中描述的 UserInfo 端点。必须使用 **https**。

13.4. 令牌选项

OAuth 服务器生成两种令牌：

访问令牌	存在时间较长的令牌，用于授权对 API 的访问。
授权代码	存在时间较短的令牌，仅用于交换访问令牌。

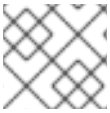
使用 **tokenConfig** 小节来设置令牌选项：

例 13.10. 主配置选项

```
oauthConfig:
```

```
...
tokenConfig:
  accessTokenMaxAgeSeconds: 86400 (1)
  authorizeTokenMaxAgeSeconds: 300 (2)
```

1. 设置 **accessTokenMaxAgeSeconds** 以控制访问令牌的生命周期。默认生命周期为 24 小时。
2. 设置 **authorizeTokenMaxAgeSeconds** 来控制授权代码的生命周期。默认生命周期为五分钟。



注意

您可以通过 **OAuthClient** 对象定义 覆盖 **accessTokenMaxAgeSeconds** 值。

13.5. 授权选项

当 OAuth 服务器收到用户之前没有授予权限的客户端的令牌请求时，OAuth 服务器采取的操作取决于 OAuth 客户端的授权策略。

当请求令牌的 OAuth 客户端不提供自己的授权策略时，会使用服务器端的默认策略。要配置默认策略，请在 **grantConfig** 部分中设置 **method** 值。**方法** 的有效值有：

auto	自动批准授权并重试请求。
prompt	提示用户批准或拒绝授权。
deny	自动拒绝授权并给客户端返回失败错误。

例 13.11. 主配置选项

```
oauthConfig:
  ...
  grantConfig:
    method: auto
```

13.6. 会话选项

OAuth 服务器在登录和重定向流程期间使用签名和加密的 cookie 会话。

使用 **sessionConfig** 小节来设置会话选项：

例 13.12. Master 配置会话选项

```
oauthConfig:
  ...
  sessionConfig:
```

```
sessionMaxAgeSeconds: 300 (1)
sessionName: ssn (2)
sessionSecretsFile: "..." (3)
```

1. 控制会话的最长期限；会话会在令牌请求完成后自动过期。如果没有启用 auto-grant，只要用户希望批准或拒绝客户端授权请求，就需要会话有效。
2. 用于存储会话的 Cookie 名称。
3. 包含序列化 **SessionSecrets** 对象的文件名。如果为空，则在每次服务器启动时都会生成随机签名和加密 secret。

如果没有指定 **sessionSecretsFile**，则会在每次主服务器启动时生成一个随机签名和加密 secret。这意味着，如果重启 master，任何正在进行中的登录都将在 master 重新启动时其会话无效。这也意味着它们将无法解码由其他其中一个 master 生成的会话。

要指定要使用的签名和加密 secret，请指定 **sessionSecretsFile**。这可让您将 secret 值与配置文件分开，并保留可分发的配置文件，例如用于调试目的。

可以在 **sessionSecretsFile** 中指定多个 secret 来启用轮转。使用列表中第一个 secret，对新会话进行签名和加密。现有会话由每个 secret 进行解密和验证，直到成功为止。

例 13.13. 会话 Secret 配置：

```
apiVersion: v1
kind: SessionSecrets
secrets: (1)
- authentication: "..." (2)
  encryption: "..." (3)
- authentication: "..."
  encryption: "..."
...
```

1. 用于进行身份验证和加密 cookie 会话的 secret 列表。必须至少指定一个 secret。每个 secret 必须设置身份验证和加密 secret。
2. 签名 secret，用于使用 HMAC 验证会话。建议您使用具有 32 或 64 字节的 secret。
3. 加密 secret，用于加密会话。必须长度为 16、24 或 32 个字符，才能选择 AES-128、AES-192 或 AES-256。

13.7. 防止 CLI 版本与用户代理不匹配

OpenShift Container Platform 实施了一个用户代理，可用于防止应用程序开发人员的 CLI 访问 OpenShift Container Platform API。

OpenShift Container Platform CLI 的用户代理由 OpenShift Container Platform 中的一组值组成：

```
<command>/<version>+<git_commit> (<platform>/<architecture>) <client>/<git_commit>
```

例如，在以下情况下：

- <command> = **oc**
- <version> = 客户端版本。例如： **v3.3.0**。对位于 **/api** 的 Kubernetes API 发出的请求会接收 Kubernetes 版本，对位于 **/oapi** 的 OpenShift Container Platform API 发出的请求则会接收 OpenShift Container Platform 版本（如 **oc version** 所指定）
- <platform> = **linux**
- <architecture> = **amd64**
- <client> = **openshift** 或 **kubernetes**，具体取决于请求的目标是位于 **/api** 的 Kubernetes API 还是位于 **/oapi** 的 OpenShift Container Platform API
- <git_commit> = 客户端版本的 Git 提交（例如 **f034127**）

用户代理将是：

```
oc/v3.3.0+f034127 (linux/amd64) openshift/f034127
```

您必须在 master 配置文件 ***/etc/origin/master/master-config.yaml*** 中配置用户代理。要应用配置，重启 API 服务器：

```
$ /usr/local/bin/master-restart api
```

作为 OpenShift Container Platform 管理员，您可以使用 master 配置中的 **userAgentMatching** 配置设置来防止客户端访问 API。因此，如果客户端使用特定的库或二进制代码，则阻止它们访问 API。

以下用户代理示例拒绝 Kubernetes 1.2 客户端二进制文件、OpenShift Origin 1.1.3 二进制文件，以及 POST 和 PUT **httpVerbs** ：

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old.  Go to https://example.org to update it."
    deniedClients:
    - regex: '\w+/v(?:(?:1\.1\.1)|(?:1\.0\.1)) \(.+/.+\) openshift/\w{7}'
    - regex: '\w+/v(?:1\.1\.3) \(.+/.+\) openshift/\w{7}'
      httpVerbs:
      - POST
      - PUT
    - regex: '\w+/v1\.2\.0 \(.+/.+\) kubernetes/\w{7}'
      httpVerbs:
      - POST
      - PUT
    requiredClients: null
```

管理员也可以拒绝与预期客户端不完全匹配的客户端：

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old.  Go to https://example.org to update it."
    deniedClients: []
    requiredClients:
    - regex: '\w+/v1\.1\.3 \(.+/.+\) openshift/\w{7}'
```

```
    - regex: '\w+/v1\.2\.0 \(.+/.+\) kubernetes/\w{7}'
      httpVerbs:
      - POST
      - PUT
```

要拒绝将原本包含在一组允许的客户端中的客户端，请结合使用 **deniedClients** 和 **requiredClients** 值。以下示例允许除 1.13 之外的所有 1.X 客户端二进制文件：

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old.  Go to https://example.org to update it."
    deniedClients:
    - regex: '\w+/v1\.13.0\+\w{7} \(.+/.+\) openshift/\w{7}'
    - regex: '\w+/v1\.13.0\+\w{7} \(.+/.+\) kubernetes/\w{7}'
    requiredClients:
    - regex: '\w+/v1\.[1-9][1-9].[0-9]\+\w{7} \(.+/.+\) openshift/\w{7}'
    - regex: '\w+/v1\.[1-9][1-9].[0-9]\+\w{7} \(.+/.+\) kubernetes/\w{7}'
```

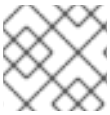
注意

当客户端的用户代理与配置不匹配时，会出现错误。要确保变异请求匹配，强制白名单。规则映射到特定的操作动词，因此您可以在允许非处理请求时对请求进行攻击。

第 14 章 使用 LDAP 同步组

14.1. 概述

作为 OpenShift Container Platform 管理员，您可以使用组来管理用户、更改其权限，并加强协作。您的组织可能已创建了用户组，并将其存储在 LDAP 服务器中。OpenShift Container Platform 可以将这些 LDAP 记录与 OpenShift Container Platform 内部记录同步，让您能够集中在一个位置管理您的组。OpenShift Container Platform 目前支持与使用以下三种通用模式定义组成员资格的 LDAP 服务器进行组同步：RFC 2307、Active Directory 和增强 Active Directory。



注意

您必须具有 **cluster-admin** 特权才能 同步组。

14.2. 配置 LDAP 同步

在运行 LDAP 同步 之前，您需要有一个同步配置文件。此文件包含 LDAP 客户端配置详情：

- 用于连接 LDAP 服务器的配置。
- 依赖于您的 LDAP 服务器中所用模式的同步配置选项。

同步配置文件也可以包含管理员定义的名称映射列表，用于将 OpenShift Container Platform 组名称映射到 LDAP 服务器中的组。

14.2.1. LDAP 客户端配置

LDAP 客户端配置

```
url: ldap://10.0.0.0:389 1
bindDN: cn=admin,dc=example,dc=com 2
bindPassword: password 3
insecure: false 4
ca: my-ldap-ca-bundle.crt 5
```

1. 连接协议、托管数据库的 LDAP 服务器的 IP 地址以及要连接的端口，格式为 **scheme://host:port**。
2. 可选的可分辨名称 (DN)，用作绑定 DN。如果需要升级特权才能检索同步操作的条目，OpenShift Container Platform 会使用此项。
3. 用于绑定的可选密码。如果需要升级特权才能检索同步操作的条目，OpenShift Container Platform 会使用此项。此值也可在 环境变量、外部文件或加密文件 中提供。
4. 为 **false** 时，安全 LDAP **ldaps://** URL 使用 TLS 进行连接，并且不安全 LDAP **ldap://** URL 会被升级到 TLS。为 **true** 时，不会对服务器进行 TLS 连接，除非您指定了 **ldaps://** URL，这时 URL 仍然尝试使用 TLS 进行连接。
5. 用于验证所配置 URL 的服务器证书的证书捆绑包。如果为空，OpenShift Container Platform 将使用系统信任的根证书。只有 **insecure** 设为 **false** 时才会应用此项。

14.2.2. LDAP 查询定义

同步配置由用于同步所需条目的 LDAP 查询定义组成。LDAP 查询的具体定义取决于用来在 LDAP 服务器中存储成员资格信息的模式。

LDAP 查询定义

```
baseDN: ou=users,dc=example,dc=com 1
scope: sub 2
derefAliases: never 3
timeout: 0 4
filter: (objectClass=inetOrgPerson) 5
pageSize: 0 6
```

1. 所有搜索都应从其中开始的目录分支的可分辨名称 (DN)。您需要指定目录树的顶端，但也可以指定目录中的子树。
2. 搜索的范围。有效值为 **base**、**one** 或 **sub**。如果未定义，则假定为 **sub** 范围。下表中可找到范围选项的描述。
3. 与 LDAP 树中别名相关的搜索行为。有效值是 **never**、**search**、**base** 或 **always**。如果未定义，则默认为 **always** 解引用别名。下表中可找到有关解引用行为的描述。
4. 客户端可进行搜索的时间限值（以秒为单位）。0 代表不实施客户端限制。
5. 有效的 LDAP 搜索过滤器。如果未定义，则默认为 **(objectClass=*)**。
6. 服务器响应页面大小的可选最大值，以 LDAP 条目数衡量。如果设置为 0，则不会对响应页面进行大小限制。当查询返回的条目数量多于客户端或服务器默认允许的数量时，需要设置分页大小。

表 14.1. LDAP 搜索范围选项

LDAP 搜索范围	描述
base	仅考虑通过为查询给定的基本 DN 指定的对象。
one	考虑作为查询的基本 DN 的树中同一级上的所有对象。
sub	考虑根部是为查询给定的基本 DN 的整个子树。

表 14.2. LDAP 解引用行为

dereferencing Behavior	描述
never	从不解引用 LDAP 树中找到的任何别名。
search	仅解引用搜索时找到的别名。
base	仅在查找基本对象时解引用别名。
always	始终解引用 LDAP 树中找到的所有别名。

14.2.3. 用户定义的名称映射

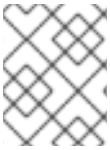
用户定义的名称映射明确将 OpenShift Container Platform 组的名称映射到可在 LDAP 服务器上找到组的唯一标识符。映射使用普通 YAML 语法。用户定义的映射可为 LDAP 服务器中每个组包含一个条目，或者仅包含这些组的一个子集。如果 LDAP 服务器上的组没有用户定义的名称映射，同步过程中的默认行为是使用指定为 OpenShift Container Platform 组名称的属性。

用户定义的名称映射

```
groupUIDNameMapping:
  "cn=group1,ou=groups,dc=example,dc=com": firstgroup
  "cn=group2,ou=groups,dc=example,dc=com": secondgroup
  "cn=group3,ou=groups,dc=example,dc=com": thirdgroup
```

14.3. 运行 LDAP 同步

创建 同步配置文件 后，同步可以开始。OpenShift Container Platform 允许管理员与同一服务器进行多种不同类型的同步。



注意

默认情况下，所有组同步或修剪操作都是空运行，因此您必须在 **sync-groups** 命令上设置 **--confirm** 标志，以便更改 OpenShift Container Platform 组记录。

将 LDAP 服务器上的所有组与 OpenShift Container Platform 同步：

```
$ oc adm groups sync --sync-config=config.yaml --confirm
```

同步所有已在 OpenShift Container Platform 中并与配置文件中指定的 LDAP 服务器中的组对应的组：

```
$ oc adm groups sync --type=openshift --sync-config=config.yaml --confirm
```

要将 LDAP 组的子集与 OpenShift Container Platform 同步，您可以使用白名单文件、黑名单文件或两者：

注意

您可以使用黑名单文件、白名单文件或白名单字面量的任意组合。白名单和黑名单文件必须每行包含一个唯一组标识符，您可以在该命令本身中直接包含白名单字面量。这些准则适用于在 LDAP 服务器上找到的组，以及 OpenShift Container Platform 中已存在的组。

```
$ oc adm groups sync --whitelist=<whitelist_file> \
                     --sync-config=config.yaml    \
                     --confirm
$ oc adm groups sync --blacklist=<blacklist_file> \
                     --sync-config=config.yaml    \
                     --confirm
$ oc adm groups sync <group_unique_identifier>    \
                     --sync-config=config.yaml    \
                     --confirm
$ oc adm groups sync <group_unique_identifier>    \
                     --whitelist=<whitelist_file> \
```

```
        --blacklist=<blacklist_file> \
        --sync-config=config.yaml    \
        --confirm
$ oc adm groups sync --type=openshift           \
        --whitelist=<whitelist_file> \
        --sync-config=config.yaml    \
        --confirm
```

14.4. 运行组修剪任务

如果创建组的 LDAP 服务器上的记录已不存在，管理员也可以选择从 OpenShift Container Platform 记录中移除这些组。修剪任务将接受与用于同步任务相同的同步配置文件和白名单。Pruning groups 部分提供了更多信息。

14.5. 同步示例

本节包含 RFC 2307、Active Directory 和增强 Active Directory 模式的示例。以下示例中两个成员同步了名为 **admins** 的组：**Jane** 和 **Jim**.每个示例都解释了：

- 如何将组和用户添加到 LDAP 服务器中。
- LDAP 同步配置文件的概貌。
- 同步之后 OpenShift Container Platform 中会生成什么组记录。

注意

这些示例假定所有用户都是其各自组的直接成员。具体而言，没有任何组的成员是其他组。如需有关如何 同步嵌套组 的信息，请参阅嵌套成员资格同步示例。

14.5.1. 使用 RFC 2307 模式同步组

在 RFC 2307 模式中，用户（Jane 和 Jim）和组都作为第一类条目存在于 LDAP 服务器上，组成员资格则存储在组的属性中。以下 **ldif** 片段定义了这个模式的用户和组：

使用 RFC 2307 模式的 LDAP 条目：*rfc2307.ldif*

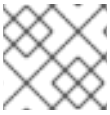
```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
```

```
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com (1)
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com (2)
member: cn=Jim,ou=users,dc=example,dc=com
```

1. 组是 LDAP 服务器中的第一类条目。
2. 组成员使用作为组属性的标识引用来列出。

要同步此组，您必须首先创建配置文件。RFC 2307 模式要求您提供用户和组条目的 LDAP 查询定义，以及在 OpenShift Container Platform 内部记录中代表它们的属性。

为明确起见，您在 OpenShift Container Platform 中创建的组应尽可能将可分辨名称以外的属性用于面向用户或管理员的字段。例如，通过电子邮件标识 OpenShift Container Platform 组的用户，并将该组的名称用作通用名称。以下配置文件创建了这些关系：



注意

如果使用用户定义的名称映射，您的配置文件 会有所不同。

使用 RFC 2307 模式的 LDAP 同步配置：*rfc2307_config.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389 (1)
insecure: false (2)
rfc2307:
    groupsQuery:
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    groupUIDAttribute: dn (3)
    groupNameAttributes: [ cn ] (4)
    groupMembershipAttributes: [ member ] (5)
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    userUIDAttribute: dn (6)
```

```
    userNameAttributes: [ uid ] 7
    tolerateMemberNotFoundErrors: false
    tolerateMemberOutOfScopeErrors: false
```

1. 存储该组记录的 LDAP 服务器的 IP 地址和主机。
2. 为 **false** 时，安全 LDAP **ldaps://** URL 使用 TLS 进行连接，并且不安全 LDAP **ldap://** URL 会被升级到 TLS。为 **true** 时，不会对服务器进行 TLS 连接，除非您指定了 **ldaps://** URL，这时 URL 仍然尝试使用 TLS 进行连接。
3. 唯一标识 LDAP 服务器上组的属性。将 DN 用于 groupUIDAttribute 时，您无法指定 **groupsQuery** 过滤器。要进行精细过滤，请使用 白名单/黑名单方法。
4. 要用作组名称的属性。
5. 存储成员资格信息的组属性。
6. 唯一标识 LDAP 服务器上用户的属性。将 DN 用于 userUIDAttribute 时，您无法指定**usersQuery** 过滤器。要进行精细过滤，请使用 白名单/黑名单方法。
7. OpenShift Container Platform 组记录中用作用户名称的属性。

使用 *rfc2307_config.yaml* 文件运行同步：

```
$ oc adm groups sync --sync-config=rfc2307_config.yaml --confirm
```

OpenShift Container Platform 创建以下组记录作为上述同步操作的结果：

使用 *rfc2307_config.yaml* 文件创建的 OpenShift Container Platform 组

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 1
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com 2
    openshift.io/ldap.url: LDAP_SERVER_IP:389 3
  creationTimestamp:
  name: admins 4
users: 5
- jane.smith@example.com
- jim.adams@example.com
```

1. 此 OpenShift Container Platform 组与 LDAP 服务器最后一次同步的时间，采用 ISO 6801 格式。
2. LDAP 服务器上组的唯一标识符。
3. 存储该组记录的 LDAP 服务器的 IP 地址和主机。
4. 根据同步文件指定的组的名称。
5. 属于组的成员的用户，名称由同步文件指定。

14.5.1.1. RFC2307，带有用户定义的名称映射

使用用户定义的名称映射同步组时，配置文件会更改为包含这些映射，如下所示。

使用 RFC 2307 模式及用户定义的名称映射的 LDAP 同步配置：
rfc2307_config_user_defined.yaml

```
kind: LDAPSyncConfig
apiVersion: v1
groupUIDNameMapping:
  "cn=admins,ou=groups,dc=example,dc=com": Administrators 1
rfc2307:
    groupsQuery:
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    groupUIDAttribute: dn 2
    groupNameAttributes: [ cn ] 3
    groupMembershipAttributes: [ member ]
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    userUIDAttribute: dn 4
    userNameAttributes: [ uid ]
    tolerateMemberNotFoundErrors: false
    tolerateMemberOutOfScopeErrors: false
```

1. 用户定义的名称映射。
2. 唯一标识符属性，用于用户定义的名称映射中的键。将 DN 用于 groupUIDAttribute 时，您无法指定 **groupsQuery** 过滤器。要进行精细过滤，请使用 白名单/黑名单方法。
3. 如果其唯一标识符不在用户定义的名称映射中，用于指定 OpenShift Container Platform 组的属性。
4. 唯一标识 LDAP 服务器上用户的属性。将 DN 用于 userUIDAttribute 时，您无法指定**usersQuery** 过滤器。要进行精细过滤，请使用 白名单/黑名单方法。

使用 ***rfc2307_config_user_defined.yaml*** 文件运行同步：

```
$ oc adm groups sync --sync-config=rfc2307_config_user_defined.yaml --confirm
```

OpenShift Container Platform 创建以下组记录作为上述同步操作的结果：

使用 *rfc2307_config_user_defined.yaml* 文件创建的 OpenShift Container Platform 组

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
```

```
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp:
  name: Administrators 1
users:
- jane.smith@example.com
- jim.adams@example.com
```

1 由用户定义的名称映射指定的组名称。

14.5.2. 使用 RFC 2307 及用户定义的容错来同步组

默认情况下，如果要同步的组包含其条目在成员查询中定义范围之外的成员，组同步会失败并显示以下错误：

```
Error determining LDAP group membership for "<group>": membership lookup for user "<user>" in group "<group>" failed because of "search for entry with dn="<user-dn>" would search outside of the base dn specified (dn="<base-dn>")".
```

这通常表示 **usersQuery** 字段中配置了 **baseDN**。不过，如果 **baseDN** 有意不含有组中的部分成员，那么设置 **tolerateMemberOutOfScopeErrors: true** 可以让组同步继续进行。范围之外的成员将被忽略。

同样，当组同步过程未能找到某个组的某一成员时，它会彻底失败并显示错误：

```
Error determining LDAP group membership for "<group>": membership lookup for user "<user>" in group "<group>" failed because of "search for entry with base dn="<user-dn>" refers to a non-existent entry".

Error determining LDAP group membership for "<group>": membership lookup for user "<user>" in group "<group>" failed because of "search for entry with base dn="<user-dn>" and filter "<filter>" did not return any results".
```

这通常表示错误配置的 **usersQuery** 字段。不过，如果组中包含已知缺失的成员条目，那么设置 **tolerateMemberNotFoundErrors: true** 可以让组同步继续进行。有问题的成员将被忽略。

警告

为 LDAP 组同步启用容错会导致同步过程忽略有问题的成员条目。如果 LDAP 组同步配置不正确，这可能会导致同步的 OpenShift Container Platform 组中缺少成员。

使用 RFC 2307 模式并且组成员资格有问题的 LDAP 条目：*rfc2307_problematic_users.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
```

```
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com
member: cn=INVALID,ou=users,dc=example,dc=com (1)
member: cn=Jim,ou=OUTOFSCOPE,dc=example,dc=com (2)
```

1. LDAP 服务器上不存在的成员。
2. 可能存在，但不在同步任务的用户查询的 **baseDN** 下的成员。

要容许以上示例中的错误，您必须在同步配置文件中添加以下内容：

使用 RFC 2307 模式且容许错误的 LDAP 同步配置：*rfc2307_config_tolerating.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
rfc2307:
    groupsQuery:
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
    groupUIDAttribute: dn
    groupNameAttributes: [ cn ]
    groupMembershipAttributes: [ member ]
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
    userUIDAttribute: dn (1)
```

```
    userNameAttributes: [ uid ]
    tolerateMemberNotFoundErrors: true 2
    tolerateMemberOutOfScopeErrors: true 3
```

2 为 **true** 时，同步任务容许找不到部分成员的组，并且找不到 LDAP 条目的成员将被忽略。如果找不到组成员，同步任务的默认行为将失败。

3 为 **true** 时，同步任务容许其部分成员在 **usersQuery** 基本 DN 中给定用户范围之外的组，并且不在成员查询范围的成员将被忽略。如果组中某个成员超出范围，则同步任务的默认行为将失败。

1 唯一标识 LDAP 服务器上用户的属性。将 DN 用于 userUIDAttribute 时，您无法指定**usersQuery** 过滤器。要进行精细过滤，请使用 白名单/黑名单方法。

使用 *rfc2307_config_tolerating.yaml* 文件运行同步：

```
$ oc adm groups sync --sync-config=rfc2307_config_tolerating.yaml --confirm
```

OpenShift Container Platform 创建以下组记录作为上述同步操作的结果：

使用 *rfc2307_config.yaml* 文件创建的 OpenShift Container Platform 组

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp:
  name: admins
users: 1
- jane.smith@example.com
- jim.adams@example.com
```

1 属于组的成员的用户，根据同步文件指定。缺少查询遇到容许错误的成员。

14.5.3. 使用 Active Directory 同步组

在 Active Directory 模式中，两个用户（Jane 和 Jim）都作为第一类条目存在于 LDAP 服务器中，组成员资格则存储在用户的属性中。以下 **ldif** 片段定义了这个模式的用户和组：

使用 Active Directory 模式的 LDAP 条目：*active_directory.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
```

```
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: admins (1)

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: admins
```

(1) 用户的组成员资格列为用户的属性，组没有作为条目存在于服务器中。**memberOf** 属性不一定是用户的字面量属性；在一些 LDAP 服务器中，它在搜索过程中创建并返回给客户端，但不提交给数据库。

要同步此组，您必须首先创建配置文件。Active Directory 模式要求您提供用户条目的 LDAP 查询定义，以及在内部 OpenShift Container Platform 组记录中代表它们的属性。

为明确起见，您在 OpenShift Container Platform 中创建的组应尽可能将可分辨名称以外的属性用于面向用户或管理员的字段。例如，通过电子邮件标识 OpenShift Container Platform 组的用户，但通过 LDAP 服务器上的组名称来定义组名称。以下配置文件创建了这些关系：

使用 Active Directory 模式的 LDAP 同步配置： ***active_directory_config.yaml***

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
activeDirectory:
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    filter: (objectclass=inetOrgPerson)
    pageSize: 0
  userNameAttributes: [ uid ] (1)
  groupMembershipAttributes: [ memberOf ] (2)
```

(1) OpenShift Container Platform 组记录中用作用户名称的属性。

(2) 存储成员资格信息的用户属性。

使用 ***active_directory_config.yaml*** 文件运行同步：

```
$ oc adm groups sync --sync-config=active_directory_config.yaml --confirm
```

OpenShift Container Platform 创建以下组记录作为上述同步操作的结果：

使用 *active_directory_config.yaml* 文件创建的 OpenShift Container Platform 组

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 (1)
    openshift.io/ldap.uid: admins (2)
    openshift.io/ldap.url: LDAP_SERVER_IP:389 (3)
  creationTimestamp:
  name: admins (4)
users: (5)
- jane.smith@example.com
- jim.adams@example.com
```

1. 此 OpenShift Container Platform 组与 LDAP 服务器最后一次同步的时间，采用 ISO 6801 格式。
2. LDAP 服务器上组的唯一标识符。
3. 存储该组记录的 LDAP 服务器的 IP 地址和主机。
4. LDAP 服务器中列出的组名称。
5. 属于组的成员的用户，名称由同步文件指定。

14.5.4. 使用增强 Active Directory 同步组

在增强 Active Directory 模式中，用户（Jane 和 Jim）和组都作为第一类条目存在于 LDAP 服务器中，组成员资格则存储在用户的属性中。以下 **ldif** 片段定义了这个模式的用户和组：

使用增强 Active Directory 模式的 LDAP 条目：*increaseded_active_directory.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com (1)

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
```

```
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com (2)
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com
```

1. 用户的组成员资格列为用户的属性。
2. 组是在 LDAP 服务器上的第一类条目。

要同步此组，您必须首先创建配置文件。增强 Active Directory（augmented Active Directory） 模式要求您提供用户条目和组条目的 LDAP 查询定义，以及在内部 OpenShift Container Platform 组记录中代表它们的属性。

为明确起见，您在 OpenShift Container Platform 中创建的组应尽可能将可分辨名称以外的属性用于面向用户或管理员的字段。例如，通过电子邮件标识 OpenShift Container Platform 组的用户，并将该组的名称用作通用名称。以下配置文件创建了这些关系。

使用增强 Active Directory 模式的 LDAP 同步配置：*increaseded_active_directory_config.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
augmentedActiveDirectory:
    groupsQuery:
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    groupUIDAttribute: dn (1)
    groupNameAttributes: [ cn ] (2)
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        filter: (objectclass=inetOrgPerson)
        pageSize: 0
    userNameAttributes: [ uid ] (3)
    groupMembershipAttributes: [ memberOf ] (4)
```

1. 唯一标识 LDAP 服务器上组的属性。将 DN 用于 groupUIDAttribute 时，您无法指定 **groupsQuery** 过滤器。要进行精细过滤，请使用 白名单/黑名单方法。

2 要用作组名称的属性。

3 OpenShift Container Platform 组记录中用作用户名称的属性。

4 存储成员资格信息的用户属性。

使用 *augmented_active_directory_config.yaml* 文件运行同步：

```
$ oc adm groups sync --sync-config=augmented_active_directory_config.yaml --confirm
```

OpenShift Container Platform 创建以下组记录作为上述同步操作的结果：

使用 *augmented_active_directory_config.yaml* 文件创建的 OpenShift 组

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 1
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com 2
    openshift.io/ldap.url: LDAP_SERVER_IP:389 3
  creationTimestamp:
  name: admins 4
users: 5
- jane.smith@example.com
- jim.adams@example.com
```

1 此 OpenShift Container Platform 组与 LDAP 服务器最后一次同步的时间，采用 ISO 6801 格式。

2 LDAP 服务器上组的唯一标识符。

3 存储该组记录的 LDAP 服务器的 IP 地址和主机。

4 根据同步文件指定的组的名称。

5 属于组的成员的用户，名称由同步文件指定。

14.6. 嵌套成员资格同步示例

OpenShift Container Platform 中的组不嵌套。在消耗数据之前，LDAP 服务器必须平展组成员资格。Microsoft 的 Active Directory Server 通过 **LDAP_MATCHING_RULE_IN_CHAIN** 规则支持这一功能，其 OID 为 **1.2.840.113556.1.4.1941**。另外，使用此匹配规则时只能同步明确列在白名单中 的组。

本节中的示例使用了增强 Active Directory 模式，它将同步一个名为 **admins** 的组，该组有一个用户 **Jane** 和一个组 **otheradmins**。**otheradmins** 组具有一个用户成员：**Jim**.这个示例阐述了：

- 如何将组和用户添加到 LDAP 服务器中。
- LDAP 同步配置文件的概貌。
- 同步之后 OpenShift Container Platform 中会生成什么组记录。

在增强 Active Directory 模式中，用户（**Jane** 和 **Jim**）和组都作为第一类条目存在于 LDAP 服务器中，组成员资格则存储在用户或组的属性中。以下 **ldif** 片段定义了这个模式的用户和组：

使用增强 Active Directory 模式和嵌套成员的 LDAP 条目：*augmented_active_directory_nested.ldif*

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com (1)

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: cn=otheradmins,ou=groups,dc=example,dc=com (2)

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com (3)
objectClass: group
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=otheradmins,ou=groups,dc=example,dc=com

dn: cn=otheradmins,ou=groups,dc=example,dc=com (4)
objectClass: group
cn: otheradmins
owner: cn=admin,dc=example,dc=com
description: Other System Administrators
memberOf: cn=admins,ou=groups,dc=example,dc=com (5) (6)
member: cn=Jim,ou=users,dc=example,dc=com
```

(1) (2) (5) 用户和组的成员资格列为对象的属性。

(3) (4) 组是在 LDAP 服务器上的第一类条目。

6 **otheradmins** 组是 **admins** 组的成员。

要将嵌套的组与 Active Directory 同步，您必须提供用户条目和组条目的 LDAP 查询定义，以及在内部 OpenShift Container Platform 组记录中代表它们的属性。另外，此配置也需要进行某些修改：

- **oc adm groups sync** 命令必须明确将组 列在白名单中。
- 用户的 **groupMembershipAttributes** 必须包含 **"memberOf:1.2.840.113556.1.4.1941:"**，以遵守 **LDAP_MATCHING_RULE_IN_CHAIN** 规则。
- **groupUIDAttribute** 必须设为 **dn**。
- **groupsQuery**：
  - 不得设置 **filter**。
  - 必须设置有效的 **derefAliases**。
  - 不应设置 **basedn**，因为此值将被忽略。
  - 不应设置 **scope**，因为此值将被忽略。

为明确起见，您在 OpenShift Container Platform 中创建的组应尽可能将可分辨名称以外的属性用于面向用户或管理员的字段。例如，通过电子邮件标识 OpenShift Container Platform 组的用户，并将该组的名称用作通用名称。以下配置文件创建了这些关系：

使用增强 Active Directory 模式和嵌套成员的 LDAP 同步配置：*increaseded_active_directory_config_nested.yaml*

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
augmentedActiveDirectory:
  groupsQuery: 1
    derefAliases: never
    pageSize: 0
  groupUIDAttribute: dn 2
  groupNameAttributes: [ cn ] 3
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    filter: (objectclass=inetOrgPerson)
    pageSize: 0
  userNameAttributes: [ uid ] 4
  groupMembershipAttributes: [ "memberOf:1.2.840.113556.1.4.1941:" ] 5
```

1 无法指定 **groupsQuery** 过滤器。**groupsQuery** 基本 DN 和范围值将被忽略。**groupsQuery** 必须设置有效的 **derefAliases**。

2 唯一标识 LDAP 服务器上组的属性。必须设为 **dn**。

3 要用作组名称的属性。

4

用作 OpenShift Container Platform 组记录中用户名称的属性。大多数安装中首选使用 **uid** 或 **sAMAccountName**。

5 存储成员资格信息的用户属性。注意 **LDAP_MATCHING_RULE_IN_CHAIN** 的使用。

使用 *augmented_active_directory_config_nested.yaml* 文件运行同步：

```
$ oc adm groups sync \
    'cn=admins,ou=groups,dc=example,dc=com' \
    --sync-config=augmented_active_directory_config_nested.yaml \
    --confirm
```

注意

您必须明确将 **cn=admins,ou=groups,dc=example,dc=com** 组 列在白名单中。

OpenShift Container Platform 创建以下组记录作为上述同步操作的结果：

使用 *augmented_active_directory_config_nested.yaml* 文件创建的 OpenShift 组

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 1
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com 2
    openshift.io/ldap.url: LDAP_SERVER_IP:389 3
  creationTimestamp:
  name: admins 4
users: 5
- jane.smith@example.com
- jim.adams@example.com
```

1 此 OpenShift Container Platform 组与 LDAP 服务器最后一次同步的时间，采用 ISO 6801 格式。

2 LDAP 服务器上组的唯一标识符。

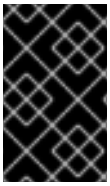
3 存储该组记录的 LDAP 服务器的 IP 地址和主机。

4 根据同步文件指定的组的名称。

5 属于组的成员的用户，名称由同步文件指定。请注意，嵌套组成员包含在内，因为 Microsoft Active Directory Server 已经平展了组成员关系。

14.7. LDAP 同步配置规格

配置文件的对象规格如下。请注意，不同的模式对象有不同的字段。例如，v1.ActiveDirectoryConfig 没有 **groupsQuery** 字段，而 v1.RFC2307Config 和 v1.AugmentedActiveDirectoryConfig 都有这个字段。



重要

不支持二进制属性。所有来自 LDAP 服务器的属性数据都必须采用 UTF-8 编码字符串的格式。例如，切勿将 **objectGUID** 等二进制属性用作 ID 属性。您必须改为使用字符串属性，如 **sAMAccountName** 或 **userPrincipalName**。

14.7.1. v1.LDAPSyncConfig

LDAPSyncConfig 包含定义 LDAP 组同步所需的配置选项。

名称	描述	模式
kind	代表此对象所代表的 REST 资源的字符串值。服务器可以从客户端向其提交请求的端点推断。无法更新。采用驼峰拼写法 (CamelCase)。更多信息：https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md#types-kinds	字符串
apiVersion	定义对象的此表示法的版本控制模式。服务器应该将识别的模式转换为最新的内部值，并可拒绝未识别的值。更多信息：https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md#resources	字符串
url	主机是要连接到的 LDAP 服务器的方案、主机和端口：**scheme://host:port**	字符串
bindDN	要绑定到 LDAP 服务器的可选 DN。	字符串
bindPassword	在搜索阶段要绑定的可选密码。	v1.StringSource
insecure	若为 **true**，则表示连接不应使用 TLS。若为 **false**，则 **ldaps://** URL 使用 TLS 进行连接，并且使用 https://tools.ietf.org/html/rfc2830 中指定的 StartTLS 将 **ldap://** URL 升级为 TLS 连接。如果将 **insecure** 设为 **true** 并且使用 **ldaps://** URL 方案，则 URL 仍然会尝试使用指定的 **ca** 进行 TLS 连接。	布尔值

名称	描述	模式
ca	在向服务器发出请求时使用的可选的可信证书颁发机构捆绑包。若为空，则使用默认的系统根证书。	字符串
groupUIDNameMapping	LDAP 组 UID 与 OpenShift Container Platform 组名称的可选直接映射。	对象
rfc2307	包含用于从设置的 LDAP 服务器提取数据的配置，其格式类似于 RFC2307：第一类组和用户条目，以及由列出其成员的组条目的多值属性决定的组成员资格。	v1.RFC2307Config
activeDirectory	包含用于从设置的 LDAP 服务器提取数据的配置，其格式与 Active Directory 中使用的类似：第一类用户条目，以及由列出所在组的成员的多值属性决定的组成员资格。	v1.ActiveDirectoryConfig
augmentedActiveDirectory	包含用于从设置的 LDAP 服务器提取数据的配置，其格式与上面描述的 Active Directory 中使用的类似，再另加一项：有第一类组条目，它们用来保存元数据而非组成员资格。	v1.AugmentedActiveDirectoryConfig

14.7.2. v1.StringSource

StringSource 允许指定内联字符串，或通过环境变量或文件从外部指定。当它只包含一个字符串值时，它会编列为一个简单 JSON 字符串。

名称	描述	模式
value	指定明文值，或指定加密值（如果指定了 **keyFile**）。	字符串
env	指定包含明文值或加密值（如果指定了 **keyFile**）的环境变量。	字符串
file	引用含有明文值或加密值（如果指定了 **keyFile**）的文件。	字符串
keyFile	引用包含用于解密值的密钥的文件。	字符串

14.7.3. v1.LDAPQuery

LDAPQuery 包含构建 LDAP 查询时所需的选项。

名称	描述	模式
baseDN	所有搜索都应从中开始的目录分支的 DN。	字符串
scope	（可选）搜索的范围。可以是 **base**（仅基本对象）、**one**（基本级别上的所有对象）、**sub**（整个子树）。若未设置，则默认为 **sub**。	字符串
derefAliases	（可选）搜索的行为与别名相关。可以是 **never**（不要 dereference 别名）、**search**（仅在搜索中 dereference）、**base**（仅在查找基本对象时 dereference）、**always**（始终 dereference）。若未设置，则默认为 **always**。	字符串
timeout	包含所有对服务器的请求在放弃等待响应前应保持待定的时间限制，以秒为单位。如果是 **0**，则不会实施客户端一侧的限制。	整数
filter	使用基本 DN 从 LDAP 服务器检索所有相关条目的有效 LDAP 搜索过滤器。	字符串
pageSize	最大首选页面大小，以 LDAP 条目数衡量。页面大小为 **0** 表示不进行分页。	整数

14.7.4. v1.RFC2307Config

RFC2307Config 包含必要的配置选项，用于定义 LDAP 组同步如何使用 RFC2307 模式与 LDAP 服务器交互。

名称	描述	模式
groupsQuery	包含用于返回组条目的 LDAP 查询模板。	v1.LDAPQuery
groupUIDAttribute	定义 LDAP 组条目上的哪个属性将解释为其唯一标识符。(**ldapGroupUID**)	字符串

名称	描述	模式
groupNameAttributes	定义 LDAP 组条目上的哪些属性将解释为用于 OpenShift Container Platform 组的名称。	字符串数组
groupMembershipAttributes	定义 LDAP 组条目上哪些属性将解释为其成员。这些属性中包含的值必须可由 **UserUIDAttribute** 查询。	字符串数组
usersQuery	包含用于返回用户条目的 LDAP 查询模板。	v1.LDAPQuery
userUIDAttribute	定义 LDAP 用户条目上的哪个属性将解释为其唯一标识符。它必须与 **GroupMembershipAttributes** 中找到的值对应。	字符串
userNameAttributes	定义要使用 LDAP 用户条目上的哪些属性，以用作其 OpenShift Container Platform 用户名。使用第一个带有非空值的属性。这应该与您的 **LDAPPasswordIdentityProvider** 的 **PreferredUsername** 设置匹配。用作 OpenShift Container Platform 组记录中用户名称的属性。大多数安装中首选 **mail** 或 **sAMAccountName**。	字符串数组
tolerateMemberNotFoundErrors	决定在遇到缺失的用户条目时 LDAP 同步任务的行为。若为 **true**，则容许找不到任何匹配项的 LDAP 用户查询，并且只记录错误。若为 **false**，则 LDAP 同步任务在用户查询找不到匹配项时将失败。默认值为 'false'。如果此标志设为 'true'，则配置错误的 LDAP 同步任务可能会导致组成员资格被移除，因此建议谨慎使用此标志。	布尔值

名称	描述	模式
tolerateMemberOutOfScopeErrors	决定在遇到超出范围的用户条目时 LDAP 同步任务的行为。如果为 **true**，则容许超出为所有用户查询给定的基本 DN 范围的 LDAP 用户查询，并且仅记录错误。若为 **false**，则当用户查询在所有用户查询指定的基本 DN 范围外搜索时，LDAP 同步任务将失败。如果此标志设为 **true**，则配置错误的 LDAP 同步任务可导致组中缺少用户，因此建议谨慎使用此标志。	布尔值

14.7.5. v1.ActiveDirectoryConfig

ActiveDirectoryConfig 包含必要的配置选项，用于定义 LDAP 组同步如何使用 Active Directory 模式与 LDAP 服务器交互。

名称	描述	模式
usersQuery	包含用于返回用户条目的 LDAP 查询模板。	v1.LDAPQuery
userNameAttributes	定义 LDAP 用户条目上的哪些属性将解释为其 OpenShift Container Platform 用户名。用作 OpenShift Container Platform 组记录中用户名称的属性。大多数安装中首选 **mail** 或 **sAMAccountName**。	字符串数组
groupMembershipAttributes	定义 LDAP 用户条目上的哪些属性将解释为它所属的组。	字符串数组

14.7.6. v1.AugmentedActiveDirectoryConfig

AugmentedActiveDirectoryConfig 包含必要的配置选项，用于定义 LDAP 组同步如何使用增强 Active Directory 模式与 LDAP 服务器交互。

名称	描述	模式
usersQuery	包含用于返回用户条目的 LDAP 查询模板。	v1.LDAPQuery

名称	描述	模式
userNameAttributes	定义 LDAP 用户条目上的哪些属性将解释为其 OpenShift Container Platform 用户名。用作 OpenShift Container Platform 组记录中用户名称的属性。大多数安装中首选 **mail** 或 **sAMAccountName**。	字符串数组
groupMembershipAttributes	定义 LDAP 用户条目上的哪些属性将解释为它所属的组。	字符串数组
groupsQuery	包含用于返回组条目的 LDAP 查询模板。	v1.LDAPQuery
groupUIDAttribute	定义 LDAP 组条目上的哪个属性将解释为其唯一标识符。(**ldapGroupUID**)	字符串
groupNameAttributes	定义 LDAP 组条目上的哪些属性将解释为用于 OpenShift Container Platform 组的名称。	字符串数组

第 15 章 配置 LDAP 故障切换

OpenShift Container Platform 提供了一个 身份验证供应商，用于轻量级目录访问协议(LDAP)设置，但它只能连接到单个 LDAP 服务器。在 OpenShift Container Platform 安装过程中，您可以为 LDAP 故障切换配置系统安全服务守护进程(SSSD)，以确保在一个 LDAP 服务器失败时访问集群。

此配置的设置是高级的，需要单独的身份验证服务器（也称为 **远程基本身份验证服务器** ）用于 OpenShift Container Platform 进行通信。您可以将这个服务器配置为将额外属性（如电子邮件地址）传递给 OpenShift Container Platform，以便它可以在 web 控制台中显示它们。

本小节论述了如何在专用物理或虚拟机(VM)上完成此设置，但您也可以在容器中配置 SSSD。

重要

您必须完成本主题的所有部分。

15.1. 配置基本身份验证的先决条件

- 在开始设置前，您需要了解以下有关 LDAP 服务器的以下信息：
  - 目录服务器是否由 FreeIPA、Active Directory 或其他 LDAP 解决方案提供驱动。
  - LDAP 服务器的统一资源标识符 (URI)，如 **ldap.example.com**。
  - LDAP 服务器的 CA 证书的位置。
  - LDAP 服务器是否为 RFC 2307 还是 RFC2307bis 供用户组使用。
- 准备服务器：
  - ***remote-basic.example.com*** ：用作远程基本身份验证服务器的虚拟机。
    - 选择包括此服务器的 SSSD 版本 1.12.0 的操作系统，如 Red Hat Enterprise Linux 7.0 或更高版本。
  - ***openshift.example.com*** ：OpenShift Container Platform 的新安装。
    - 没有为这个集群配置验证方法。
    - 不要在这个集群上启动 OpenShift Container Platform。

15.2. 使用远程基本身份验证服务器生成和共享证书

在 Ansible 主机清单文件（默认为 ***/etc/ansible/hosts***）中列出的第一个 master 主机上完成以下步骤。

1. 为确保远程基本身份验证服务器和 OpenShift Container Platform 间的通信需要信任，在这组其他阶段创建一组传输层安全(TLS)证书。运行以下命令：

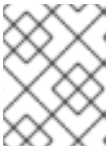
```
# openshift start \
    --public-master=https://openshift.example.com:8443 \
    --write-config=/etc/origin/
```

输出中包含 ***/etc/origin/master/ca.crt*** 和 ***/etc/origin/master/ca.key*** 签名证书。

2. 使用签名证书生成要在远程基本身份验证服务器中使用的密钥：
   -

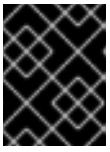
```
# mkdir -p /etc/origin/remote-basic/
# oc adm ca create-server-cert \
    --cert='/etc/origin/remote-basic/remote-basic.example.com.crt' \
    --key='/etc/origin/remote-basic/remote-basic.example.com.key' \
    --hostnames=remote-basic.example.com \ 1
    --signer-cert='/etc/origin/master/ca.crt' \
    --signer-key='/etc/origin/master/ca.key' \
    --signer-serial='/etc/origin/master/ca.serial.txt'
```

1 以逗号分隔的所有主机名和接口 IP 地址列表，它们需要访问远程基本身份验证服务器。



注意

您生成的证书文件有效期为两年。您可以通过更改 **--expire-days** 和 **--signer-expiredays** 值来更改这个周期，但出于安全原因，不要使它们大于 730。



重要

如果您没有列出需要访问远程基本身份验证服务器的所有主机名和接口 IP 地址，HTTPS 连接将失败。

3. 将必要的证书和密钥复制到远程基本身份验证服务器：

```
# scp /etc/origin/master/ca.crt \
      root@remote-basic.example.com:/etc/pki/CA/certs/

# scp /etc/origin/remote-basic/remote-basic.example.com.crt \
      root@remote-basic.example.com:/etc/pki/tls/certs/

# scp /etc/origin/remote-basic/remote-basic.example.com.key \
      root@remote-basic.example.com:/etc/pki/tls/private/
```

15.3. 为 LDAP 故障切换配置 SSSD

在远程基本身份验证服务器上完成这些步骤。

您可以将 SSSD 配置为检索属性，如电子邮件地址和显示名称，并将它们传递给 OpenShift Container Platform 以便在 web 界面中显示。在以下步骤中，您要将 SSSD 配置为向 OpenShift Container Platform 提供电子邮件地址：

1. 安装所需的 SSSD 和 Web 服务器组件：

```
# yum install -y sssd \
                 sssd-dbus \
                 realmd \
                 httpd \
                 mod_session \
                 mod_ssl \
                 mod_lookup_identity \
                 mod_authnz_pam \
                 php \
                 mod_php
```

2. 设置 SSSD 以针对 LDAP 服务器验证此虚拟机。如果 LDAP 服务器是 FreeIPA 或 Active Directory 环境，则使用 **realmd** 将这个计算机加入到域中。

```
# realm join ldap.example.com
```

有关更高级的情况，请参阅 系统级验证指南

3. 要使用 SSSD 管理 LDAP 的故障切换情况，请在 **ldap_uri** 行上的 */etc/sssd/sssd.conf* 文件中添加更多条目。使用 FreeIPA 注册的系统可以使用 DNS SRV 记录自动处理故障转移。

4. 修改 */etc/sssd/sssd.conf* 文件的 **[domain/DOMAINNAME]** 部分并添加此属性：

```
[domain/example.com]
...
ldap_user_extra_attrs = mail 1
```

1 指定检索 LDAP 解决方案的电子邮件地址的正确属性。对于 IPA，请指定 **邮件**。其他 LDAP 解决方案可能使用其他属性，如**电子邮件**。

5. 确认 */etc/sssd/sssd.conf* 文件中的 *domain* 参数仅包含 **[domain/DOMAINNAME]** 部分中列出的域名。

```
domains = example.com
```

6. 授予 Apache 权限以检索电子邮件属性。将以下行添加到 */etc/sssd/sssd.conf* 文件的 **[ifp]** 部分：

```
[ifp]
user_attributes = +mail
allowed_uids = apache, root
```

7. 要确定正确应用所有更改，请重启 SSSD：

```
$ systemctl restart sssd.service
```

8. 测试用户信息是否可以正确检索：

```
$ getent passwd <username>
username:*:12345:12345:Example User:/home/username:/usr/bin/bash
```

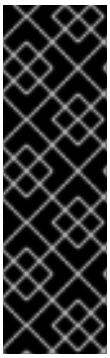
9. 确认您指定的 mail 属性返回您的域的电子邮件地址：

```
# dbus-send --print-reply --system --dest=org.freedesktop.sssd.infopipe \
    /org/freedesktop/sssd/infopipe org.freedesktop.sssd.infopipe.GetUserAttr \
    string:username \ 1
    array:string:mail 2

method return time=1528091855.672691 sender=:1.2787 -> destination=:1.2795 serial=13
reply_serial=2
   array [
      dict entry(
         string "mail"
         variant             array [
```

```
            string "username@example.com"
        ]
    )
]
```

1. 在 LDAP 解决方案中提供用户名。
2. 指定您配置的属性。

10. 尝试以 LDAP 用户身份登录虚拟机，并确认您可以使用 LDAP 凭证登录。您可以使用本地控制台或 SSH 等远程服务登录。



重要

默认情况下，所有用户都可以使用其 LDAP 凭据登录远程基本身份验证服务器。您可以更改此行为：

- 如果您使用 IPA 加入的系统，请配置基于主机的访问控制。
- 如果您使用 Active Directory 加入系统，请使用组策略对象。
- 有关其他情况，请参阅 SSSD 配置 文档。

15.4. 配置 APACHE 使用 SSSD

1. 创建一个包含以下内容的 */etc/pam.d/openshift* 文件：

```
auth required pam_sss.so
account required pam_sss.so
```

此配置可让 PAM（可插拔验证模块）使用 **pam_ss.so** 确定为 **openshift** 堆栈发出身份验证请求的身份验证和访问控制。

2. 编辑 */etc/httpd/conf.modules.d/55-authnz_pam.conf* 文件并取消注释以下行：

```
LoadModule authnz_pam_module modules/mod_authnz_pam.so
```

3. 要为远程基本身份验证配置 Apache ***httpd.conf*** 文件，请在 */etc/httpd/conf.d* 目录中创建 *openshift-remote-basic-auth.conf* 文件。使用以下模板来提供所需设置和值：

重要

仔细检查模板的内容，并根据您的环境自定义其相应的内容。

```
LoadModule request_module modules/mod_request.so
LoadModule php7_module modules/libphp7.so

# Nothing needs to be served over HTTP.  This virtual host simply redirects to
# HTTPS.
<VirtualHost *:80>
  DocumentRoot /var/www/html
  RewriteEngine              On
  RewriteRule     ^(.*)$     https://%{HTTP_HOST}$1 [R,L]
```

```
</VirtualHost>

<VirtualHost *:443>
  # This needs to match the certificates you generated.  See the CN and X509v3
  # Subject Alternative Name in the output of:
  # openssl x509 -text -in /etc/pki/tls/certs/remote-basic.example.com.crt
  ServerName remote-basic.example.com

  DocumentRoot /var/www/html

  # Secure all connections with TLS
  SSLEngine on
  SSLCertificateFile /etc/pki/tls/certs/remote-basic.example.com.crt
  SSLCertificateKeyFile /etc/pki/tls/private/remote-basic.example.com.key
  SSLCACertificateFile /etc/pki/CA/certs/ca.crt

  # Require that TLS clients provide a valid certificate
  SSLVerifyClient require
  SSLVerifyDepth 10

  # Other SSL options that may be useful
  # SSLCertificateChainFile ...
  # SSLCARevocationFile ...

  # Send logs to a specific location to make them easier to find
  ErrorLog logs/remote_basic_error_log
  TransferLog logs/remote_basic_access_log
  LogLevel warn

  # PHP script that turns the Apache REMOTE_USER env var
  # into a JSON formatted response that OpenShift understands
  <Location /check_user.php>
    # all requests not using SSL are denied
    SSLRequireSSL
    # denies access when SSLRequireSSL is applied
    SSLOptions +StrictRequire
    # Require both a valid basic auth user (so REMOTE_USER is always set)
    # and that the CN of the TLS client matches that of the OpenShift master
    <RequireAll>
      Require valid-user
      Require expr %{SSL_CLIENT_S_DN_CN} == 'system:openshift-master'
    </RequireAll>
    # Use basic auth since OpenShift will call this endpoint with a basic challenge
    AuthType Basic
    AuthName openshift
    AuthBasicProvider PAM
    AuthPAMService openshift

    # Store attributes in environment variables. Specify the email attribute that
    # you confirmed.
    LookupOutput Env
    LookupUserAttr mail REMOTE_USER_MAIL
    LookupUserGECOS REMOTE_USER_DISPLAY_NAME

    # Other options that might be useful
```

```
    # While REMOTE_USER is used as the sub field and serves as the immutable ID,
    # REMOTE_USER_PREFERRED_USERNAME could be used to have a different
username
    # LookupUserAttr <attr_name> REMOTE_USER_PREFERRED_USERNAME

    # Group support may be added in a future release
    # LookupUserGroupsIter REMOTE_USER_GROUP
  </Location>

  # Deny everything else
  <Location ~ "^((?!\/check_user\.php).)*$">
     Deny from all
  </Location>
</VirtualHost>
```

4. 在 */var/www/html* 目录中创建 *check_user.php* 脚本。包含以下代码：

```
<?php
// Get the user based on the Apache var, this should always be
// set because we 'Require valid-user' in the configuration
$user = apache_getenv('REMOTE_USER');

// However, we assume it may not be set and
// build an error response by default
$data = array(
    'error' => 'remote PAM authentication failed'
);

// Build a success response if we have a user
if (!empty($user)) {
    $data = array(
        'sub' => $user
    );
    // Map of optional environment variables to optional JSON fields
    $env_map = array(
        'REMOTE_USER_MAIL' => 'email',
        'REMOTE_USER_DISPLAY_NAME' => 'name',
        'REMOTE_USER_PREFERRED_USERNAME' => 'preferred_username'
    );

    // Add all non-empty environment variables to JSON data
    foreach ($env_map as $env_name => $json_name) {
        $env_data = apache_getenv($env_name);
        if (!empty($env_data)) {
            $data[$json_name] = $env_data;
        }
    }
}

// We always output JSON from this script
header('Content-Type: application/json', true);

// Write the response as JSON
echo json_encode($data);
?>
```

5. 启用 Apache 加载模块。修改 */etc/httpd/conf.modules.d/55-lookup_identity.conf* 文件并取消注释以下行：

```
LoadModule lookup_identity_module modules/mod_lookup_identity.so
```

6. 设置 SELinux 布尔值，以便 SElinux 允许 Apache 通过 D-BUS 连接到 SSSD：

```
# setsebool -P httpd_dbus_sssd on
```

7. 将布尔值设置为 告知 SELinux 可以接受 Apache 联络 PAM 子系统：

```
# setsebool -P allow_httpd_mod_auth_pam on
```

8. 启动 Apache：

```
# systemctl start httpd.service
```

15.5. 将 OPENSHIFT CONTAINER PLATFORM 配置为使用 SSSD 作为基本远程身份验证服务器

修改集群的默认配置，以使用您创建的新身份提供程序。在 Ansible 主机清单文件中列出的第一个 master 主机上完成以下步骤。

1. 打开 */etc/origin/master/master-config.yaml* 文件。

2. 找到 **identityProviders** 部分，并将其替换为以下代码：

```
identityProviders:
- name: sssd
  challenge: true
  login: true
  mappingMethod: claim
  provider:
    apiVersion: v1
    kind: BasicAuthPasswordIdentityProvider
    url: https://remote-basic.example.com/check_user.php
    ca: /etc/origin/master/ca.crt
    certFile: /etc/origin/master/openshift-master.crt
    keyFile: /etc/origin/master/openshift-master.key
```

3. 使用更新的配置重启 OpenShift Container Platform：

```
# /usr/local/bin/master-restart api api

# /usr/local/bin/master-restart controllers controllers
```

4. 使用 **oc** CLI 测试登录：

```
$ oc login https://openshift.example.com:8443
```

您只能使用有效的 LDAP 凭证登录。

5. 列出身份，并确认为每个用户名显示一个电子邮件地址。运行以下命令：

```
$ oc get identity -o yaml
```

第 16 章 配置 SDN

16.1. 概述

OpenShift SDN 启用 OpenShift Container Platform 集群内 pod 之间的通信，并建立 *pod 网络*。目前提供了三个 SDN 插件 （**ovs-subnet**、**ovs-multitenant** 和 **ovs-networkpolicy**），提供不同的方法来配置 pod 网络。

16.2. 可用的 SDN 供应商

上游 Kubernetes 项目不附带默认的网络解决方案。相反，Kubernetes 已开发了一个 Container Network Interface(CNI)，以便网络供应商与自己的 SDN 解决方案集成。

有几个 OpenShift SDN 插件可用于红帽及第三方插件。

红帽已与多个 SDN 供应商合作，通过 Kubernetes CNI 接口在 OpenShift Container Platform 上认证其 SDN 网络解决方案，其中包括通过其产品授权流程对 SDN 插件的支持流程。如果您通过 OpenShift 创建支持问题单，红帽可促进交易流程，以便这两个公司都参与满足您的需求。

以下 SDN 解决方案由第三方供应商直接在 OpenShift Container Platform 上验证和支持：

- Cisco ACI (™)
- Juniper Contrail(™)
- Nokia Nuage(™)
- Tigera Calico(™)
- VMware NSX-T (™)

在 OpenShift Container Platform 上安装 VMware NSX-T(™)
VMware NSX-T(™)提供了一个 SDN 和安全基础架构来构建云原生应用程序环境。除了 vSphere 管理程序(ESX)外，这些环境还包括 KVM 和原生公共云。

当前集成需要同时具有 NSX-T 和 OpenShift Container Platform 的*新*安装。目前，支持 NSX-T 版本 2.4，当前只支持使用 ESXi 和 KVM 虚拟机监控程序。

如需更多信息，请参阅 OpenShift 的 NSX-T Container Plug-in 。

16.3. 使用 ANSIBLE 配置 POD 网络

对于初始集群安装，则默认安装和配置 **ovs-subnet** 插件，尽管可以在安装期间使用 **os_sdn_network_plugin_name** 参数覆盖，可在 Ansible 清单文件中配置。

例如，要覆盖标准 **ovs-subnet** 插件，并使用 **ovs-multitenant** 插件：

```
# Configure the multi-tenant SDN plugin (default is 'redhat/openshift-ovs-subnet')
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
```

如需了解可在清单文件中设置的与网络相关的 Ansible 变量的描述，请参阅配置集群变量。

16.4. 在 MASTER 中配置 POD 网络

集群管理员可以通过修改 master 配置文件的 **networkConfig** 部分中的参数（默认位于 */etc/origin/master/master-config.yaml* ）来控制 master 主机上的 pod 网络设置：

为单个 CIDR 配置 pod 网络

```
networkConfig:
  clusterNetworks:
  - cidr: 10.128.0.0/14 (1)
    hostSubnetLength: 9 (2)
  networkPluginName: "redhat/openshift-ovs-subnet" (3)
  serviceNetworkCIDR: 172.30.0.0/16 (4)
```

(1) 用于节点 IP 分配的集群网络

(2) 节点内 pod IP 分配的位数

(3) 为 **ovs-subnet** 插件设置为 **redhat/openshift-ovs-subnet**，为 **ovs-multitenant** 插件设置为 **redhat/openshift-ovs-multitenant**，为 **ovs-networkpolicy** 插件设置为 **redhat/openshift-ovs-networkpolicy**

(4) 集群的服务 IP 分配

另外，您可以通过将单独的范围添加到带有范围和 **hostSubnetLength** 的 **clusterNetworks** 字段中，创建具有多个 CIDR 范围的 pod 网络。

可将多个范围用于一次，而且该范围可以被扩展或合同。通过撤离节点，可以把节点从一个范围移到另一个范围，然后删除并重新创建节点。如需更多信息，请参阅 管理节点 部分。列出顺序出现节点分配，然后在范围已满后移至列表上的下一个操作。

为多个 CIDR 配置 pod 网络

```
networkConfig:
  clusterNetworks:
  - cidr: 10.128.0.0/14 (1)
    hostSubnetLength: 9 (2)
  - cidr: 10.132.0.0/14
    hostSubnetLength: 9
  externalIPNetworkCIDRs: null
  hostSubnetLength: 9
  ingressIPNetworkCIDR: 172.29.0.0/16
  networkPluginName: redhat/openshift-ovs-multitenant (3)
  serviceNetworkCIDR: 172.30.0.0/16
```

(1) 用于节点 IP 分配的集群网络。

(2) 节点内 pod IP 分配的位数。

(3) 为 **ovs-subnet** 插件设置为 **redhat/openshift-ovs-subnet**，为 **ovs-multitenant** 插件设置为 **redhat/openshift-ovs-multitenant**，为 **ovs-networkpolicy** 插件设置为 **redhat/openshift-ovs-networkpolicy**。

您可以将元素添加到 **clusterNetworks** 值，如果没有节点正在使用这个 CIDR 范围，则删除它们。

重要

在集群首次创建后无法更改 **hostSubnetLength** 值，**cidr** 项只能更改为一个更大的网络，它需要仍然包含原始网络（如果节点在它的范围内分配），且只能扩展 **serviceNetworkCIDR**。例如，对于 10.128.0.0/14 的典型值，您可以将 **cidr** 改为 10.128.0.0/9 （例如，net 10 的整个上半个）而不是 10.64.0.0/16，因为这不会重叠原始值。

您可以将 **serviceNetworkCIDR** 从 172.30.0.0/16 改为 172.30.0.0/15，但不改为 172.28.0.0/14，因为原始范围完全位于 CIDR 的开头。如需更多信息，请参阅扩展服务网络。

确保重启 API 和 master 服务以使任何更改生效：

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

重要

节点上的 pod 网络设置必须与 master 上的 networkConfig.clusterNetworks 参数配置的 pod 网络设置匹配。这可以通过修改相应节点 配置映射的 **networkConfig** 部分中的参数来实现 ：

```
proxyArguments:
  cluster-cidr:
  - 10.128.0.0/12 1
```

1 CIDR 值必须包含在 master 级别上定义的所有集群网络 CIDR 范围，但不与其他 IP 范围冲突，如用于节点和服务。

在重启 master 服务后，必须将配置传播到节点。在每个节点上，必须重启 atomic-openshift-node 服务和 ovs pod。为了避免停机遵循 管理节点 中定义的步骤，并一次为每个节点或一组节点执行以下步骤：

1. 将节点标记为不可调度。

```
# oc adm manage-node <node1> <node2> --schedulable=false
```

2. 排空节点：

```
# oc adm drain <node1> <node2>
```

3. 重启节点：

```
# reboot
```

4. 将节点重新标记为可调度：

```
#  oc adm manage-node <node1> <node2> --schedulable
```

16.5. 更改集群网络的 VXLAN PORT

作为集群管理员，您可以更改系统使用的 VXLAN 端口。

由于您无法更改正在运行的 **clusternetwork** 对象的 VXLAN 端口，所以您必须通过编辑 master 配置文件中的 **vxlanPort** 变量来删除任何现有的网络配置。

1. 删除现有的 **clusternetwork** ：

```
# oc delete clusternetwork default
```

2. 编辑 master 配置文件（默认位于 */etc/origin/master/master-config.yaml*）来创建新的 **clusternetwork**：

```
networkConfig:
  clusterNetworks:
  - cidr: 10.128.0.0/14
    hostSubnetLength: 9
  - cidr: 10.132.0.0/14
    hostSubnetLength: 9
  externalIPNetworkCIDRs: null
  hostSubnetLength: 9
  ingressIPNetworkCIDR: 172.29.0.0/16
  networkPluginName: redhat/openshift-ovs-multitenant
  serviceNetworkCIDR: 172.30.0.0/16
  vxlanPort: 4889 (1)
```

(1) 设置为节点用于 VXLAN 端口的值。它可以是 1-65535 之间的一个整数。默认值为 **4789**。

3. 在每个集群节点上的 iptables 规则添加新端口：

```
# iptables -A OS_FIREWALL_ALLOW -p udp -m state --state NEW -m udp --dport 4889 -j
ACCEPT (1)
```

(1) **4889** 是您在主配置文件中设置的 **vxlanPort** 值。

4. 重启 master 服务：

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

5. 删除所有旧的 SDN pod，以使用新更改传播新 pod：

```
# oc delete pod -l app=sdn -n openshift-sdn
```

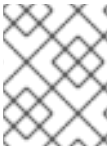
16.6. 在节点上配置 POD 网络

集群管理员可以通过修改相应节点配置映射的 **networkConfig** 部分中的参数来控制 节点上的 pod 网络设置 ：

```
networkConfig:
  mtu: 1450 (1)
  networkPluginName: "redhat/openshift-ovs-subnet" (2)
```

(1) pod 覆盖网络的最大传输单元(MTU)

2 为 **ovs-subnet** 插件设置为 **redhat/openshift-ovs-subnet**，为 **ovs-multitenant** 插件设置为 **redhat/openshift-ovs-multitenant**，为 **ovs-networkpolicy** 插件设置为 **redhat/openshift-ovs-**

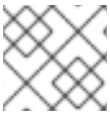


注意

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。

16.7. 扩展服务网络

如果您在服务网络中的有效地址的数量较少，只要您确定当前范围位于新范围的开头，就可以扩展该范围。



注意

服务网络只能扩展，无法更改或合同。

1. 对所有的 master，修改配置文件中的 **serviceNetworkCIDR** 和 **servicesSubnet** 参数（默认为 */etc/origin/master/master-config.yaml*）。仅将 / 后面的数字更改为较小的数字。
2. 删除 **clusterNetwork 默认** 对象：

```
$ oc delete clusternetwork default
```

3. 在所有 master 上重启控制器组件：

```
# master-restart controllers
```

4. 将 Ansible 清单文件中的 **openshift_portal_net** 变量的值更新为新的 CIDR：

```
# Configure SDN cluster network and kubernetes service CIDR blocks. These
# network blocks should be private and should not conflict with network blocks
# in your infrastructure that pods may require access to. Can not be changed
# after deployment.
openshift_portal_net=172.30.0.0/<new_CIDR_range>
```

对于集群中的每个节点，请完成以下步骤：

1. 将节点标记为不可调度。
2. 撤离节点 中的 pod。
3. 重新引导节点。
4. 节点再次可用后，将节点标记为可调度。

16.8. 在 SDN 插件之间迁移

如果您已经使用一个 SDN 插件，并希望切换到另外一个 SDN 插件：

1. 在所有 master 和 节点上 更改 **networkPluginName** 参数。

2. 在所有 master 上重启 API 和 master 服务：

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

3. 停止所有 master 和节点上的节点服务：

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

4. 如果您要在 OpenShift SDN 插件间进行切换，请在所有 master 和节点上重启 OpenShift SDN。

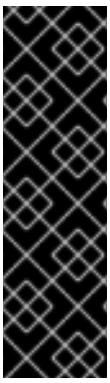
```
oc delete pod --all -n openshift-sdn
```

5. 在所有 master 和节点上重启节点服务：

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

6. 如果您要从 OpenShift SDN 插件切换到第三方插件，请清理特定于 OpenShift SDN 的工件：

```
$ oc delete clusternetwork --all
$ oc delete hostsubnets --all
$ oc delete netnamespaces --all
```

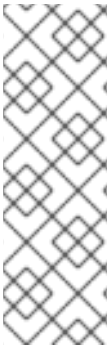


重要

另外，在切换到 **ovs-multitenant** 后，用户无法使用服务目录置备服务。对 **openshift-monitoring** 也是如此。要更正此问题，使这些项目成为全局项目：

```
$ oc adm pod-network make-projects-global kube-service-catalog
$ oc adm pod-network make-projects-global openshift-monitoring
```

如果集群最初使用 **ovs-multitenant** 安装，则不会出现此问题，因为这些命令是作为 Ansible playbook 的一部分执行的。



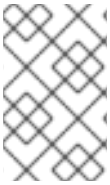
注意

从 **ovs-subnet** 切换到 **ovs-multitenant** OpenShift SDN 插件时，集群中的所有现有项目都将被完全隔离（作为唯一的 VNID）。集群管理员可以选择使用管理员 CLI 修改项目网络。

运行以下命令来检查 VNIDs：

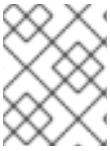
```
$ oc get netnamespace
```

16.8.1. 从 ovs-multitenant 迁移到 ovs-networkpolicy



注意

v1 NetworkPolicy 功能仅适用于 OpenShift Container Platform。这意味着 OpenShift Container Platform 不提供出口策略类型、IPBlock 和组合 **podSelector** 和 **namespaceSelector**。



注意

不要在默认的 OpenShift Container Platform 项目中应用 **NetworkPolicy** 功能，因为它们可能会破坏与集群的通信。

除了在 SDN 插件间迁移以上通用的插件迁移步骤外，从 **ovs-multitenant** 插件迁移到 **ovs-networkpolicy** 插件时，会有一个额外的步骤。您需要确定每个命名空间有一个唯一的 **NetID**这意味着，如果您之前已将项目接合在一起或将项目设置为全局项目，则需要在切换到 **ovs-networkpolicy** 插件前撤销该项目，否则 NetworkPolicy 对象无法正常工作。

提供了一个帮助脚本，可修复 **NetID 的**，创建 NetworkPolicy 对象来隔离之前隔离命名空间，并启用之前加入的命名空间之间的连接。

使用以下步骤将这个帮助程序脚本迁移到 **ovs-networkpolicy** 插件，同时仍然运行 **ovs-multitenant** 插件：

1. 下载脚本并添加执行文件权限：

```
$ curl -O https://raw.githubusercontent.com/openshift/origin/release-
3.11/contrib/migration/migrate-network-policy.sh
$ chmod a+x migrate-network-policy.sh
```

2. 运行脚本（需要集群管理员角色）。

```
$ ./migrate-network-policy.sh
```

运行此脚本后，每个命名空间都完全与其它命名空间隔离，因此不同命名空间中的 pod 间的连接尝试会失败，直到完成到 **ovs-networkpolicy** 插件为止。

如果您希望新创建的命名空间同时具有相同的策略，您可以将默认 NetworkPolicy 对象设置为与 **default-deny** 匹配，以及由迁移脚本创建的 **allow-from-global-namespaces** 策略。

注意

如果脚本失败或其他错误，或者稍后决定恢复到 **ovs-multitenant** 插件，您可以使用 un-migration 脚本。此脚本会撤消迁移脚本所做的更改，并重新加入之前加入的命名空间。

16.9. 外部访问集群网络

如果一个 OpenShift Container Platform 以外的主机需要访问集群网络，则有两个选项：

1. 将主机配置为 OpenShift Container Platform 节点，但将其 不可调度，以便 master 不会在其上调度容器。

2. 在主机和位于集群网络上的主机间创建一个隧道。

这两个选项都作为实际用例的一部分阐述，在文档中用于配置 从边缘负载均衡器到 OpenShift SDN 中的容器的路由。

16.10. 使用 FLANNEL

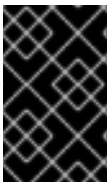
作为默认 SDN 的替代方案，OpenShift Container Platform 还提供用于安装基于 **flannel**的网络的 Ansible playbook。这在两个平台中也依赖于 SDN 的云供应商平台（如 Red Hat OpenStack Platform）中运行 OpenShift Container Platform 非常有用。

Flannel 使用单个 IP 网络命名空间用于向每个实例分配连续空间子集。因此，容器无法尝试联系同一网络空间中的任何 IP 地址。这种障碍的多租户是多租户，因为网络无法用于将一个应用中的容器与另一个应用隔离。

根据您首选的租户隔离还是性能，您应该在决定 OpenShift SDN（多租户）和用于内部网络的 flannel（多租户）之间决定适当的选择。

重要

Flannel 仅支持 Red Hat OpenStack Platform 上的 OpenShift Container Platform。



重要

当前版本的 Neutron 默认在端口上强制实施端口安全性。这可防止端口发送或接收使用与端口本身不同的 MAC 地址的数据包。Flannel 创建虚拟 MAC 和 IP 地址，且必须在端口上发送和接收数据包，因此在执行 flannel 流量的端口上必须禁用端口安全性。

在 OpenShift Container Platform 集群中启用 flannel：

1. Neutron 端口安全控制必须配置为与 Flannel 兼容。Red Hat OpenStack Platform 的默认配置会禁用用户对 **port_security** 的控制。配置 Neutron，以允许用户控制各个端口上的 **port_security** 设置。

   a. 在 Neutron 服务器上，将以下内容添加到 ***/etc/neutron/plugins/ml2/ml2_conf.ini*** 文件：

   ```
   [ml2]
   ...
   extension_drivers = port_security
   ```

   b. 然后重启 Neutron 服务：

   ```
   service neutron-dhcp-agent restart
   service neutron-ovs-cleanup restart
   service neutron-metadata-agentrestart
   service neutron-l3-agent restart
   service neutron-plugin-openvswitch-agent restart
   service neutron-vpn-agent restart
   service neutron-server  restart
   ```

2. 在 Red Hat OpenStack Platform 上创建 OpenShift Container Platform 实例时，在容器网络 flannel 接口的端口中禁用端口安全性和安全组：

   ```
   neutron port-update $port --no-security-groups --port-security-enabled=False
   ```

   注意

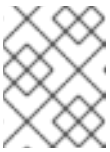
   Flannel 从 etcd 收集信息来配置和分配节点的子网。因此，附加到 etcd 主机的安全组应该允许从节点访问端口 2379/tcp，节点安全组应允许到 etcd 主机上该端口的出口通信。

   a. 在运行安装前，在 Ansible 清单文件中设置以下变量：

```
openshift_use_openshift_sdn=false (1)
openshift_use_flannel=true (2)
flannel_interface=eth0
```

1. 将 **openshift_use_openshift_sdn** 设置为 **false** 以禁用默认的 SDN。
2. 将 **openshift_use_flannel** 设置为 **true** 以启用 **flannel**。

b. 另外，您还可以使用 **flannel_interface** 变量来指定要用于主机间通信的接口。如果没有此变量，OpenShift Container Platform 安装将使用默认接口。



注意

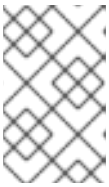
以后的发行版本中将支持使用 flannel 的 pod 和服务的自定义网络 CIDR。BZ#1473858

3. 在 OpenShift Container Platform 安装后，在每个 OpenShift Container Platform 节点上添加一组 iptables 规则：

```
iptables -A DOCKER -p all -j ACCEPT
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

要在 */etc/sysconfig/iptables* 中保留这些更改，在每个节点上使用以下命令：

```
cp /etc/sysconfig/iptables{,.orig}
sh -c "tac /etc/sysconfig/iptables.orig | sed -e '0,/:DOCKER -/ s/:DOCKER -/:DOCKER
ACCEPT/' | awk '"\!"p && /POSTROUTING/{print \"-A POSTROUTING -o eth1 -j
MASQUERADE\"; p=1} 1' | tac > /etc/sysconfig/iptables"
```



注意

iptables-save 命令保存在 *内存 iptables 规则中的所有* 当前。但是，由于 Docker，Kubernetes 和 OpenShift Container Platform 会创建大量 iptables 规则（服务等），因此无法保留这些规则，保存这些规则可能会变得存在问题。

要从 OpenShift Container Platform 流量隔离容器流量，红帽建议创建隔离的租户网络并将所有节点附加到其中。如果您使用不同的网络接口(eth1)，请记得通过 */etc/sysconfig/network-scripts/ifcfg-eth1* 文件将接口配置为在引导时启动：

```
DEVICE=eth1
TYPE=Ethernet
BOOTPROTO=dhcp
ONBOOT=yes
DEFTROUTE=no
PEERDNS=no
```

第 17 章 配置 NUAGE SDN

17.1. NUAGE SDN 和 OPENSHIFT CONTAINER PLATFORM

Nuage Networks Virtualized Services Platform (VSP) 为容器环境提供虚拟网络和软件定义网络 (SDN) 基础架构，从而简化 IT 操作并扩展 OpenShift Container Platform 的原生网络功能。

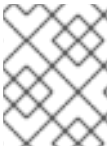
Nuage Networks VSP 支持在 OpenShift Container Platform 上运行的基于 Docker 的应用程序，以加快 pod 和传统工作负载之间的虚拟网络配置，并跨整个云基础架构启用安全策略。VSP 允许自动化安全设备，为容器应用包含精细安全性和微分段策略。

将 VSP 与 OpenShift Container Platform 应用程序工作流集成，可通过删除 DevOps 团队所面临的网络布局快速打开和更新应用程序。VSP 通过 OpenShift Container Platform 支持不同的工作流，以适应用户在使用基于策略的自动化时或完全控制的场景。

如需有关 VSP 与 OpenShift Container Platform 集成的更多信息，请参阅 网络。

17.2. 开发人员工作流

此工作流用于开发环境，需要开发人员进行很少的输入来设置网络。在此工作流中，**nuage-openshift-monitor** 负责创建为 OpenShift Container Platform 项目中创建的 pod 提供适当策略和网络所需的 VSP 结构（Zone、子网等）。创建项目时，由 **nuage-openshift-monitor** 创建该项目的默认区域和默认子网。当为给定项目创建的默认子网被省略时，nu **age-openshift-monitor** 会动态创建额外的子网。



注意

为每个 OpenShift Container Platform 项目创建一个单独的 VSP Zone，可确保在项目之间隔离。

17.3. 操作工作流

运维团队会使用此工作流来推出应用程序。在此工作流中，首先在 VSD 上配置网络和安全策略，以根据组织设置的规则来部署应用。管理用户可能会创建多个区域和子网，并使用标签将它们映射到同一项目。在启动 pod 时，用户可以使用 Nuage Labels 指定 pod 需要附加到的网络，以及需要将其应用哪些网络策略。这允许以精细的方式控制项目内和内部流量的部署。例如，项目基础上启用了项目间的通信。这可用于将项目连接到部署在共享项目中的常用服务。

17.4. 安装

VSP 与 OpenShift Container Platform 集成可用于虚拟机(VM)和裸机 OpenShift Container Platform 安装。

具有高可用性(HA)的环境可以配置多个 master 和多个节点。

在多 master 模式中的 Nuage VSP 集成只支持本节中介绍的原生 HA 配置方法。这可以和任何负载平衡解决方案结合使用，这是 HAProxy 的默认设置。清单文件包含三个 master 主机、节点、etcd 服务器和一个 HAProxy，以在所有 master 主机上平衡主 API。HAProxy 主机在清单文件的 [lb] 部分中定义，使 Ansible 能够自动安装和配置 HAProxy 作为负载平衡解决方案。

在 Ansible 节点文件中，需要指定以下参数，才能将 Nuage VSP 设置为网络插件：

```
# Create and OSEv3 group that contains masters, nodes, load-balancers, and etcd hosts
masters
```

```
nodes
etcd
lb

# Nuage specific parameters
openshift_use_openshift_sdn=False
openshift_use_nuage=True
os_sdn_network_plugin_name='nuage/vsp-openshift'
openshift_node_proxy_mode='userspace'

# VSP related parameters
vsd_api_url=https://192.168.103.200:8443
vsp_version=v4_0
enterprise=nuage
domain=openshift
vsc_active_ip=192.168.103.201
vsc_standby_ip=192.168.103.202
uplink_interface=eth0

# rpm locations
nuage_openshift_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/nuage-openshift-
monitor-4.0.X.1830.el7.centos.x86_64.rpm
vrs_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/nuage-openvswitch-
4.0.X.225.el7.x86_64.rpm
plugin_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/vsp-openshift-
4.0.X1830.el7.centos.x86_64.rpm

# Required for Nuage Monitor REST server and HA
openshift_master_cluster_method=native
openshift_master_cluster_hostname=lb.nuageopenshift.com
openshift_master_cluster_public_hostname=lb.nuageopenshift.com
nuage_openshift_monitor_rest_server_port=9443

# Optional parameters
nuage_interface_mtu=1460
nuage_master_adminusername='admin's user-name'
nuage_master_adminuserpasswd='admin's password'
nuage_master_cspadminpasswd='csp admin password'
nuage_openshift_monitor_log_dir=/var/log/nuage-openshift-monitor

# Required for brownfield install (where a {product-title} cluster exists without Nuage as the
networking plugin)
nuage_dockker_bridge=lbr0

# Specify master hosts
[masters]
fqdn_of_master_1
fqdn_of_master_2
fqdn_of_master_3

# Specify load balancer host
[lb]
fqdn_of_load_balancer
```

第 18 章 配置 NSX-T SDN

18.1. NSX-T SDN 和 OPENSHIFT CONTAINER PLATFORM

VMware NSX-T Data Center ™ 为简化 IT 操作并扩展了原生 OpenShift Container Platform 网络功能提供高级软件定义网络(SDN)、安全性和可见性。

NSX-T Data Center 支持跨多个集群的虚拟机、裸机和容器工作负载。这使得组织可以在整个环境中使用单个 SDN 来完成可见性。

有关如何与 OpenShift Container Platform 集成 NSX-T 的更多信息，请参阅 Available SDN 插件中的 NSX-T SDN。

18.2. TOPOLOGY 示例

一个典型的用例是有一个 Tier-0 (T0) 路由器将物理系统与虚拟环境连接，以及一个 Tier-1 (T1) 路由器充当 OpenShift Container Platform 虚拟机的默认网关。

每个虚拟机有两个 vNIC：一个 vNIC 连接至管理逻辑交换机，以访问虚拟机。其他 vNIC 连接到 Dump Logical Switch，并由 **nsx-node-agent** 用来连接 Pod 网络。详情请参阅 OpenShift 的 NSX Container Plug-in。

用于在 OpenShift Container Platform 安装过程中自动创建用于配置 OpenShift Container Platform 路由和所有项目 T1 路由器和逻辑交换机的 LoadBalancer。

在此拓扑中，默认的 OpenShift Container Platform HAProxy 路由器用于所有基础架构组件，如 Grafana、Prometheus、控制台、服务目录等。确保基础架构组件的 DNS 记录指向基础架构节点 IP 地址，因为 HAProxy 使用主机网络命名空间。这适用于基础架构路由，但为了避免将基础架构节点管理 IP 公开给外部世界，请将特定于应用程序的路由部署到 NSX-T LoadBalancer 中。

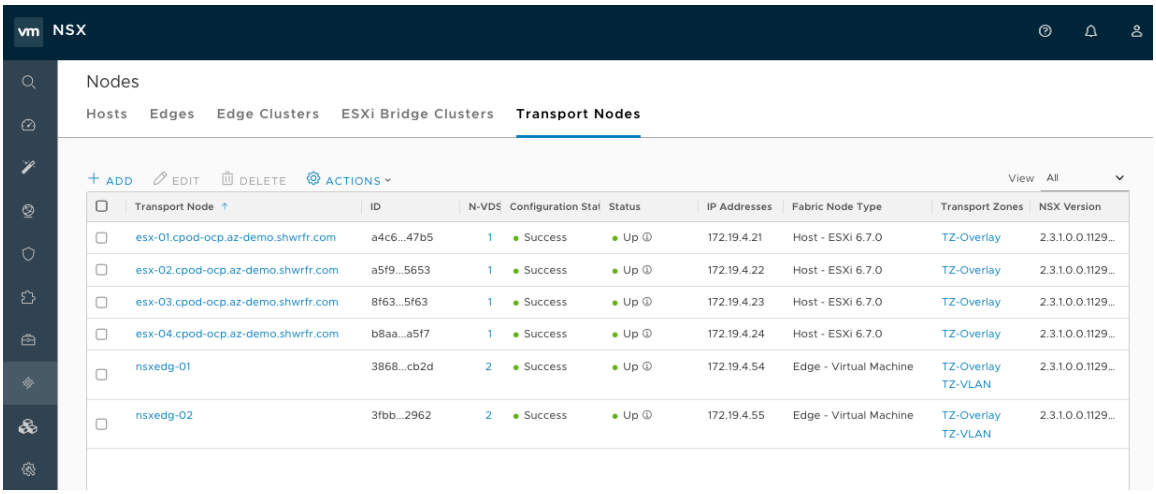
此示例拓扑假设您使用三个 OpenShift Container Platform master 虚拟机，以及四个 OpenShift Container Platform worker 虚拟机（用于基础架构，两个用于 compute）。

18.3. 安装 VMWARE NSX-T

先决条件

- ESXi 主机要求：
  - 托管 OpenShift Container Platform 节点虚拟机的 ESXi 服务器必须是 NSX-T 传输节点。

图 18.1. NSX UI 为典型的高可用性环境分离传输节点：

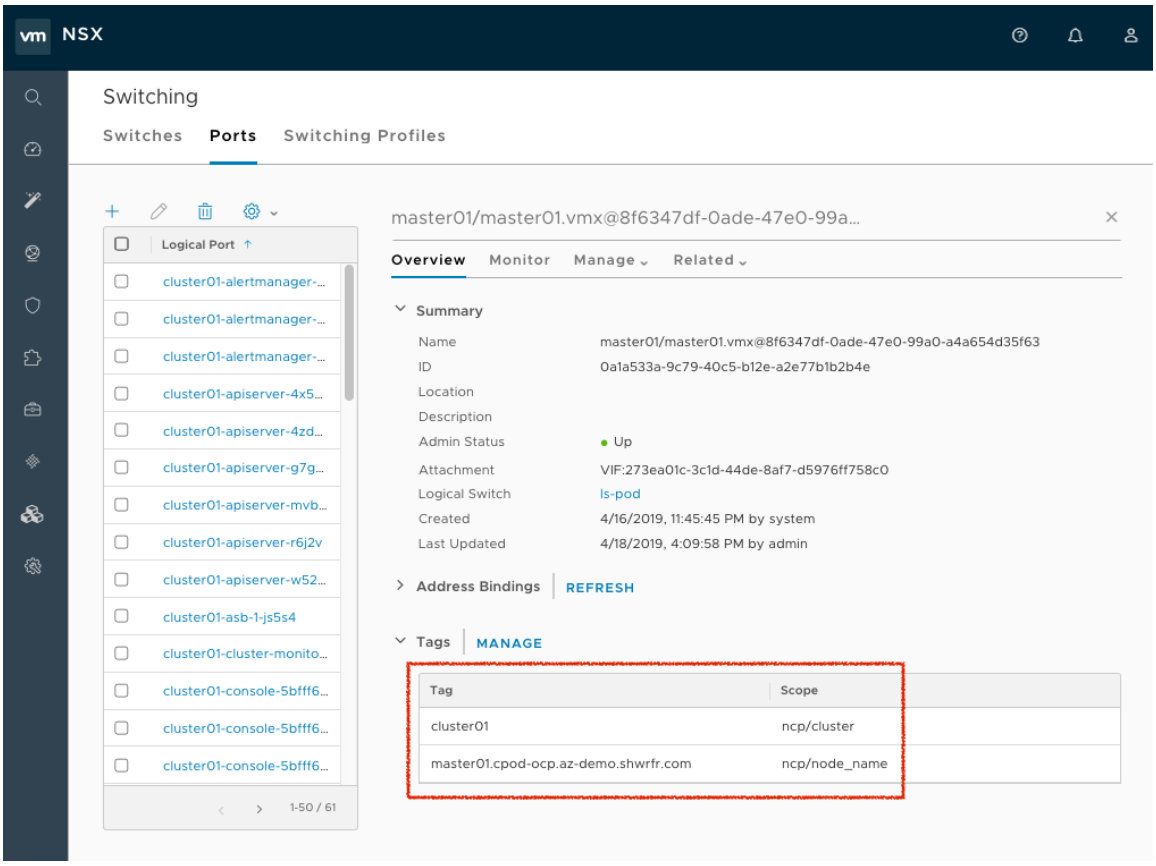


- DNS 要求：
  - 您必须在 DNS 服务器中向基础架构节点使用通配符添加新条目。这允许 NSX-T 或其他第三方 LoadBalancer 进行负载均衡。在下面的 **hosts** 文件中，条目由 **openshift_master_default_subdomain** 变量定义。
  - 您必须使用 **openshift_master_cluster_hostname** 和 **openshift_master_cluster_public_hostname** 变量更新您的 DNS 服务器。
- 虚拟机要求：
  - OpenShift Container Platform 节点虚拟机必须有两个 vNIC：
  - 管理 vNIC 必须连接到连接到管理 T1 路由器的逻辑交换机。
  - 所有虚拟机上的第二个 vNIC 必须标记 NSX-T，以便 NSX Container Plug-in(NCP)知道哪个端口需要用作特定 OpenShift Container Platform 节点上运行的所有 Pod 的父 VIF。标签必须如下：

    ```
    {'ncp/node_name': 'node_name'}
    {'ncp/cluster': 'cluster_name'}
    ```

    下图显示了所有节点的 NSX UI 中的标签。对于大规模集群，您可以使用 API 调用或使用 Ansible 自动标记。

图 18.2. NSX UI 显示节点标签



NSX UI 中的标签顺序与 API 相反。节点名称必须与 kubelet 预期相同，集群名称必须与 Ansible 主机文件中的 **nsx_openshift_cluster_name** 相同，如下所示。确保每个节点上的第二个 vNIC 中应用正确的标签。

- NSX-T 要求：
  在 NSX 中需要满足以下先决条件：
  - Tier-0 路由器.
  - Overlay Transport Zone.
  - POD 网络的 IP 块。
  - （可选）用于路由(NoNAT) POD 网络的 IP 块。
  - SNAT 的 IP 池。默认情况下，每个项目从 Pod 网络 IP Block 指定的子网只能在 NSX-T 中路由。NCP 使用这个 IP 池提供与外部的连接。
  - （可选）dFW（分发防火墙）中的 top 和 Bottom 防火墙部分。NCP 在这两个部分之间放置 Kubernetes 网络策略规则。
  - Open vSwitch 和 CNI 插件 RPM 需要托管在 HTTP 服务器中，可从 OpenShift Container Platform 节点虚拟机访问（在这个示例中为 **http://websrv.example.com**）。这些文件包含在 NCP Tar 文件中，您可以在 Download NSX Container Plug-in 2.4.0 从 VMware 下载。
- OpenShift Container Platform 要求：
  - 运行以下命令，为 OpenShift Container Platform 安装所需的软件包（若有）：

```
$ ansible-playbook -i hosts openshift-ansible/playbooks/prerequisites.yml
```

- 确保所有节点上本地下载 NCP 容器镜像
- 在 **prerequisites.yml** playbook 成功执行后，在所有节点上运行以下命令，将 **xxx** 替换为 NCP 构建版本：

```
$ docker load -i nsx-ncp-rhel-xxx.tar
```

例如：

```
$ docker load -i nsx-ncp-rhel-2.4.0.12511604.tar
```

- 获取镜像名称并重新标记它：

```
$ docker images
$ docker image tag registry.local/xxxxx/nsx-ncp-rhel nsx-ncp (1)
```

(1) 将 **xxx** 替换为 NCP 构建版本。例如：

```
docker image tag registry.local/2.4.0.12511604/nsx-ncp-rhel nsx-ncp
```

- 在 OpenShift Container Platform Ansible 主机文件中，指定以下参数，将 NSX-T 设置为网络插件：

```
[OSEv3:children]
masters
nodes
etcd

[OSEv3:vars]
ansible_ssh_user=root
openshift_deployment_type=origin
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge':
'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]
openshift_master_htpasswd_users={"admin" :
"$apr1$H0QeP6oX$HHdscz5gqMdtTcT5eoCJ20"}
openshift_master_default_subdomain=demo.example.com
openshift_use_nsx=true
os_sdn_network_plugin_name=cni
openshift_use_openshift_sdn=false
openshift_node_sdn_mtu=1500
openshift_master_cluster_method=native
openshift_master_cluster_hostname=master01.example.com
openshift_master_cluster_public_hostname=master01.example.com
openshift_hosted_manage_registry=true
openshift_hosted_manage_router=true
openshift_enable_service_catalog=true
openshift_cluster_monitoring_operator_install=true
openshift_web_console_install=true
openshift_console_install=true

# NSX-T specific configuration
#nsx_use_loadbalancer=false
nsx_openshift_cluster_name='cluster01'
```

```
nsx_api_managers='nsxmgr.example.com'
nsx_api_user='nsx_admin'
nsx_api_password='nsx_api_password_example'
nsx_tier0_router='LR-Tier-0'
nsx_overlay_transport_zone='TZ-Overlay'
nsx_container_ip_block='pod-networking'
nsx_no_snat_ip_block='pod-nonat'
nsx_external_ip_pool='pod-external'
nsx_top_fw_section='containers-top'
nsx_bottom_fw_section='containers-bottom'
nsx_ovs_uplink_port='ens224'
nsx_cni_url='http://websrv.example.com/nsx-cni-buildversion.x86_64.rpm'
nsx_ovs_url='http://websrv.example.com/openvswitch-buildversion.rhel75-1.x86_64.rpm'
nsx_kmod_ovs_url='http://websrv.example.com/kmod-openvswitch-buildversion.rhel75-
1.el7.x86_64.rpm'
nsx_insecure_ssl=true
# vSphere Cloud Provider
#openshift_cloudprovider_kind=vsphere
#openshift_cloudprovider_vsphere_username='administrator@example.com'
#openshift_cloudprovider_vsphere_password='viadmin_password'
#openshift_cloudprovider_vsphere_host='vcsa.example.com'
#openshift_cloudprovider_vsphere_datacenter='Example-Datacenter'
#openshift_cloudprovider_vsphere_cluster='example-Cluster'
#openshift_cloudprovider_vsphere_resource_pool='ocp'
#openshift_cloudprovider_vsphere_datastore='example-Datastore-name'
#openshift_cloudprovider_vsphere_folder='ocp'

[masters]
master01.example.com
master02.example.com
master03.example.com

[etcd]
master01.example.com
master02.example.com
master03.example.com

[nodes]
master01.example.com ansible_ssh_host=192.168.220.2
openshift_node_group_name='node-config-master'
master02.example.com ansible_ssh_host=192.168.220.3
openshift_node_group_name='node-config-master'
master03.example.com ansible_ssh_host=192.168.220.4
openshift_node_group_name='node-config-master'
node01.example.com ansible_ssh_host=192.168.220.5
openshift_node_group_name='node-config-infra'
node02.example.com ansible_ssh_host=192.168.220.6
openshift_node_group_name='node-config-infra'
node03.example.com ansible_ssh_host=192.168.220.7
openshift_node_group_name='node-config-compute'
node04.example.com ansible_ssh_host=192.168.220.8
openshift_node_group_name='node-config-compute'
```

如需有关 OpenShift Container Platform 安装参数的信息，请参阅 配置清单文件。

流程

满足所有先决条件后，您可以部署 NSX Data Center 和 OpenShift Container Platform。

1. 部署 OpenShift Container Platform 集群：

```
$ ansible-playbook -i hosts openshift-ansible/playbooks/deploy_cluster.yml
```

如需有关 OpenShift Container Platform 安装的更多信息，请参阅安装 OpenShift Container Platform。

2. 安装完成后，验证 NCP 和 nsx-node-agent Pod 是否正在运行：

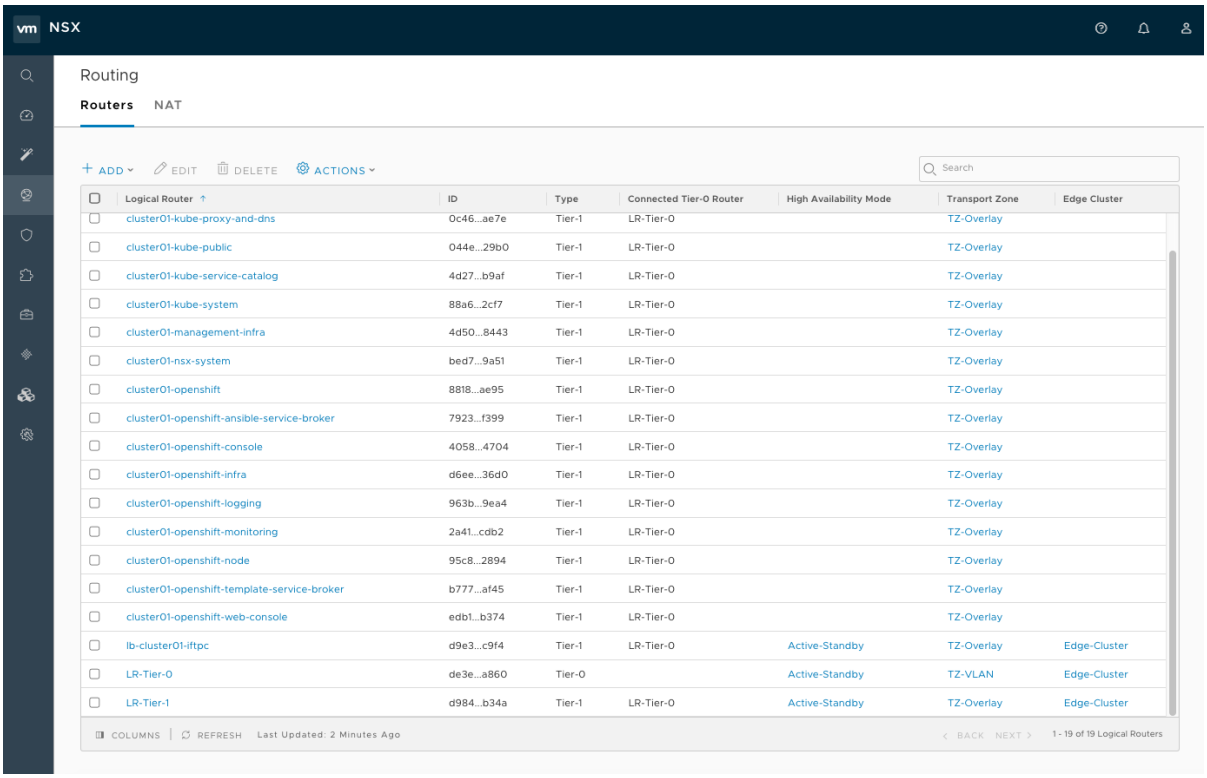
```
$ oc get pods -o wide -n nsx-system
NAME                READY     STATUS    RESTARTS   AGE       IP              NODE
NOMINATED NODE
nsx-ncp-5sggt       1/1       Running   0          1h        192.168.220.8   node04.example.com
<none>
nsx-node-agent-b8nkm   2/2       Running   0          1h        192.168.220.5
node01.example.com     <none>
nsx-node-agent-cldks   2/2       Running   0          2h        192.168.220.8
node04.example.com     <none>
nsx-node-agent-m2p5l   2/2       Running   28         3h        192.168.220.4
master03.example.com   <none>
nsx-node-agent-pcfd5   2/2       Running   0          1h        192.168.220.7
node03.example.com     <none>
nsx-node-agent-ptwnq   2/2       Running   26         3h        192.168.220.2
master01.example.com   <none>
nsx-node-agent-xgh5q   2/2       Running   26         3h        192.168.220.3
master02.example.com   <none>
```

18.4. 在 OPENSHIFT CONTAINER PLATFORM 部署后检查 NSX-T

安装 OpenShift Container Platform 并验证 NCP 和 **nsx-node-agent-*** Pod:

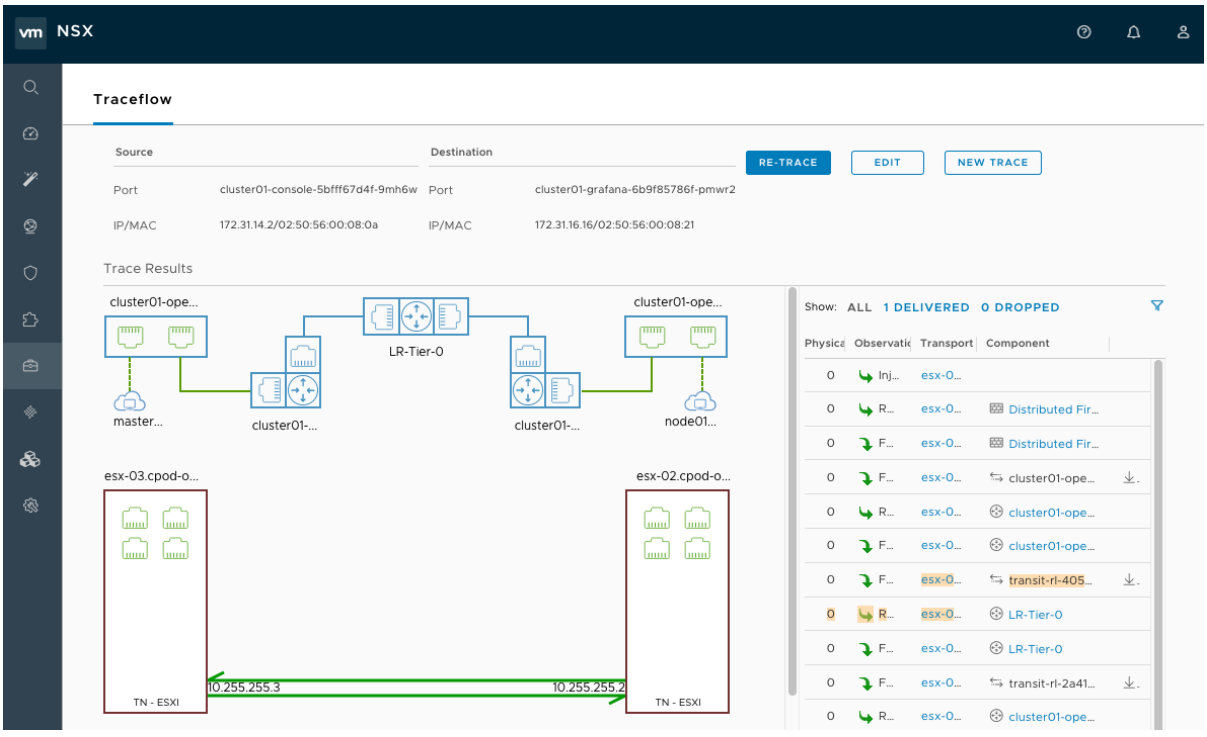
- 检查路由。确保安装期间创建了 Tier-1 路由器，并链接到 Tier-0 路由器：

图 18.3. NSX UI dislaying 显示 T1 路由器



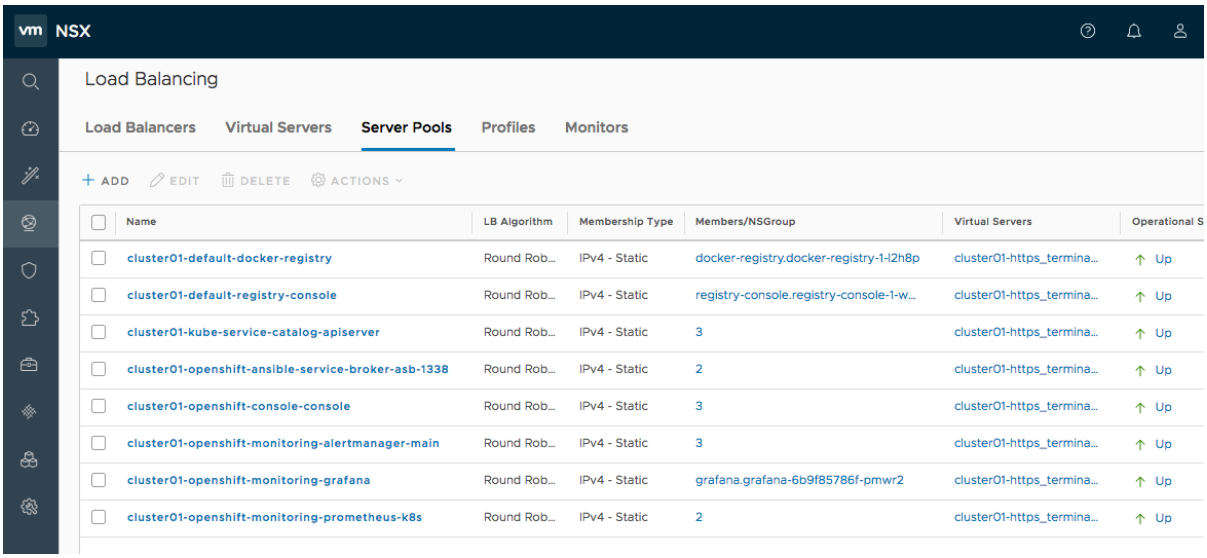
- 观察网络 traceflow 和 visibility。例如，检查 'console' 和 'grafana' 之间的连接。
如需有关保护 Pod、项目、虚拟机和外部服务间的通信的更多信息，请参见以下示例：

图 18.4. NSX UI dislaying 显示网络 traceflow



- 检查负载平衡。NSX-T 数据中心提供 Load Balancer 和 Ingress Controller 功能，如下例所示：

图 18.5. NSX UI 显示展示负载均衡器



有关其他配置和选项，请参阅 VMware NSX-T v2.4 OpenShift Plug-In 文档。

第 19 章 配置 KURYR SDN

19.1. KURYR SDN 和 OPENSHIFT CONTAINER PLATFORM

Kuryr （或更具体为 Kuryr-Kubernetes）是一个使用 CNI 和 OpenStack Neutron 构建的 SDN 解决方案。其优点包括能够使用广泛的 Neutron SDN 后端，并在 Kubernetes pod 和 OpenStack 虚拟机(VM)之间提供连接。

Kuryr-Kubernetes 和 OpenShift Container Platform 集成主要针对在 OpenStack 虚拟机上运行的 OpenShift Container Platform 集群设计。Kuryr-Kubernetes 组件作为 pod 在 **kuryr** 命名空间中的 OpenShift Container Platform 上安装：

- kuryr-controller - 在一个 **infra** 节点上安装的单个服务实例。在 OpenShift Container Platform 中建模为一个 **部署**。
- kuryr-cni - 在每个 OpenShift Container Platform 节点上安装和配置 Kuryr 作为 CNI 驱动程序。在 OpenShift Container Platform 中建模为一个 **DaemonSet**。

Kuryr 控制器监控 OpenShift API 服务器中的 pod、服务和命名空间创建、更新和删除事件。它将 OpenShift Container Platform API 调用映射到 Neutron 和 Octavia 中的对应对象。这意味着，实现了 Neutron 中继端口功能的每个网络解决方案都可以通过 Kuryr 支持 OpenShift Container Platform。这包括开源解决方案，如 OVS 和 OVN，以及 Neutron 兼容的商业 SDN。

19.2. 安装 KURYR SDN

对于 OpenStack 云上的 Kuryr SDN 安装，您必须遵循 OpenStack 配置文档 中所述的步骤。

19.3. 验证

安装 OpenShift Container Platform 后，您可以检查 Kuryr pod 是否已成功部署：

```
$ oc -n kuryr get pods -o wide
NAME                          READY     STATUS    RESTARTS   AGE       IP              NODE
kuryr-cni-ds-66kt2            2/2       Running   0          3d        192.168.99.14   infra-node-
0.openshift.example.com
kuryr-cni-ds-ggcpz            2/2       Running   0          3d        192.168.99.16   master-
0.openshift.example.com
kuryr-cni-ds-mhzjt            2/2       Running   0          3d        192.168.99.6    app-node-
1.openshift.example.com
kuryr-cni-ds-njctb            2/2       Running   0          3d        192.168.99.12   app-node-
0.openshift.example.com
kuryr-cni-ds-v8hp8            2/2       Running   0          3d        192.168.99.5    infra-node-
1.openshift.example.com
kuryr-controller-59fc7f478b-qwk4k   1/1       Running   0          3d        192.168.99.5    infra-node-
1.openshift.example.com
```

Kuryr-cni pod 在每个 OpenShift Container Platform 节点上运行。单一 kuryr-controller 实例可以在任何 **infra** 节点上运行。

注意

启用 Kuryr SDN 时不支持网络策略和节点端口服务。

第 20 章 为 AMAZON WEB SERVICES(AWS)配置

20.1. 概述

OpenShift Container Platform 可以配置为访问 AWS EC2 基础架构，包括使用 AWS 卷作为应用程序数据的持久性存储。配置 AWS 后，必须在 OpenShift Container Platform 主机上完成一些额外的配置。

20.1.1. 为 Amazon Web Services(AWS)配置授权

权限 AWS 实例需要使用在创建时分配给实例的 access 和 secret 密钥或 IAM 角色来请求和管理 OpenShift Container Platform 中的负载均衡器和存储的 IAM 角色。

IAM 帐户或 IAM 角色必须具有以下策略权限才能拥有完整的云供应商功能。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Action": [
            "ec2:DescribeVolume*",
            "ec2:CreateVolume",
            "ec2:CreateTags",
            "ec2:DescribeInstances",
            "ec2:AttachVolume",
            "ec2:DetachVolume",
            "ec2:DeleteVolume",
            "ec2:DescribeSubnets",
            "ec2:CreateSecurityGroup",
            "ec2:DescribeSecurityGroups",
            "ec2:DeleteSecurityGroup",
            "ec2:DescribeRouteTables",
            "ec2:AuthorizeSecurityGroupIngress",
            "ec2:RevokeSecurityGroupIngress",
            "elasticloadbalancing:DescribeTags",
            "elasticloadbalancing:CreateLoadBalancerListeners",
            "elasticloadbalancing:ConfigureHealthCheck",
            "elasticloadbalancing:DeleteLoadBalancerListeners",
            "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
            "elasticloadbalancing:DescribeLoadBalancers",
            "elasticloadbalancing:CreateLoadBalancer",
            "elasticloadbalancing:DeleteLoadBalancer",
            "elasticloadbalancing:ModifyLoadBalancerAttributes",
            "elasticloadbalancing:DescribeLoadBalancerAttributes"
         ],
         "Resource": "*",
         "Effect": "Allow",
         "Sid": "1"
      }
   ]
}
```

```
aws iam put-role-policy \
  --role-name openshift-role \
  --policy-name openshift-admin \
```

```
  --policy-document file: //openshift_iam_policy
```

```
aws iam put-user-policy \
  --user-name openshift-admin \
  --policy-name openshift-admin \
  --policy-document file: //openshift_iam_policy
```

注意

OpenShift 节点实例只需要 **ec2:DescribeInstance** 权限，但安装程序仅允许定义单个 AWS 访问密钥和 secret。这可绕过使用 IAM 角色，并将上面的权限分配给 master 实例，并将 **ec2:DescribeInstance** 分配给节点。

20.1.1.1. 在安装时配置 OpenShift Container Platform 云供应商

流程

要使用具有 access 和 secret 键的 IAM 帐户配置 Amazon Web Services 云供应商，请将以下值添加到清单中：

```
[OSEv3:vars]
openshift_cloudprovider_kind=aws
openshift_clusterid=openshift 1
openshift_cloudprovider_aws_access_key=AKIAJ6VLBLISADPBUA 2
openshift_cloudprovider_aws_secret_key=g/8PmDNYHVSQn0BQE+xtsHzbaZaGYjGNzhbdgwjH 3
```

1. 分配给所有资源（实例、负载平衡器、vpc 等）的标签（用于 OpenShift）。
2. IAM 帐户使用的 AWS 访问密钥。
3. IAM 帐户使用的 AWS secret 密钥。

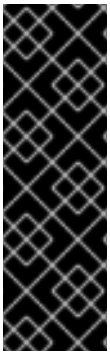
要使用 IAM 角色配置 Amazon Web Services 云供应商，请在清单中添加以下值：

```
[source,yaml]
----
[OSEv3:vars]
openshift_cloudprovider_kind=aws
openshift_clusterid=openshift 1
----
<1> A tag assigned to all resources (instances, load balancers, vpc, etc) used for OpenShift.
```

```
NOTE: The IAM role takes the place of needing an access and secret key.
```

20.1.1.2. 安装后配置 OpenShift Container Platform 云供应商

如果安装时没有提供 Amazon Web Services 云供应商值，则可以在安装后定义和创建配置。按照以下步骤配置配置文件，并为 AWS 手动配置 master 和节点。



重要

- 每个 master 主机、节点主机和子网必须具有 **kubernetes.io/cluster/<clusterid>,Value=(owned|shared)** 标签。
- 一个安全组（最好链接到节点）必须具有 **kubernetes.io/cluster/<clusterid>,Value=(owned|shared)** 标签。
  - 不要使用 **kubernetes.io/cluster/<clusterid>,Value=(owned|shared)** 标签标记所有安全组，否则 Elastic Load Balancing(ELB)将无法创建负载均衡器。

20.2. 配置安全组

在 AWS 上安装 OpenShift Container Platform 时，请确保设置适当的安全组。

这些是在安全组中必须具有的一些端口，而安装会失败。根据您要安装的集群配置，您可能需要更多。如需更多信息，并相应地调整您的安全组，请参阅所需端口以了解更多信息。

所有 OpenShift Container Platform 主机	• 来自运行安装程序/Ansible 的主机的 TCP/22
etcd 安全组	• 来自 master 的 TCP/2379 • 来自 etcd 主机的 TCP/2380
Master 安全组	• 来自 0.0.0.0/0 的 tcp/8443 • 从所有安装的环境到 3.2 的 OpenShift Container Platform 主机中的 TCP/53 • 来自安装环境的所有 OpenShift Container Platform 主机到 3.2 的 UDP/53 • 对于使用 3.2 安装的新环境，来自所有 OpenShift Container Platform 主机中的 TCP/8053 • 对于使用 3.2 安装的新环境，所有 OpenShift Container Platform 主机中的 UDP/8053
节点安全组	• 来自 master 的 TCP/10250 • 来自节点的 UDP/4789
基础架构节点（一个可以托管 OpenShift Container Platform 路由器）	• 来自 0.0.0.0/0 的 tcp/443 • 来自 0.0.0.0/0 的 tcp/80
CRI-O	如果使用 CRIO，则必须打开 tcp/10010 以允许 **oc exec** 和 **oc rsh** 操作。

如果为 master 和/或路由器配置外部负载均衡器(ELB)，您还需要为 ELB 相应地配置入口和 Egress 安全组。

20.2.1. 覆盖检测到的 IP 地址和主机名

在 AWS 中，需要覆盖变量的情况包括：

变量	使用方法
hostname	用户安装在 VPC 中，它没有为 **DNS 主机名**和**DNS 解析**进行配置。
ip	您已经配置了多个网络接口，并希望使用除默认接口以外的其他网络接口。
public_hostname	• 一个 master 实例，其中没有为 **Auto-assign Public IP** 配置 VPC 子网。对于此 master 的外部访问权限，您需要配置一个 ELB 或其他负载均衡器来提供所需的外部访问，或者您需要通过 VPN 连接主机的内部名称进行连接。 • 禁用元数据的 master 实例。 • 这个值实际上没有被节点使用。
public_ip	• 一个 master 实例，其中没有为 **Auto-assign Public IP** 配置 VPC 子网。 • 禁用元数据的 master 实例。 • 这个值实际上没有被节点使用。

对于 EC2 主机，必须部署到启用了 DNS **主机名和 DNS** 解析的 VPC 中。

20.2.1.1. 为 Amazon Web Services(AWS)配置 OpenShift Container Platform registry

Amazon Web Services (AWS) 提供对象存储，OpenShift Container Platform 可以使用它们使用 OpenShift Container Platform 容器 registry 存储容器镜像。

如需更多信息，请参阅 Amazon S3。

先决条件

OpenShift Container Platform 使用 S3 作为镜像存储。应创建 S3 存储桶、IAM 策略和带有 **Programmatic Access** 的 IAM 用户，以允许安装程序配置 registry。

以下示例使用 awscli 在 **us-east-1** 区域中创建一个名为 **openshift-registry-storage** 的存储桶。

```
# aws s3api create-bucket \
    --bucket openshift-registry-storage \
    --region us-east-1
```

默认策略

-

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation",
        "s3:ListBucketMultipartUploads"
      ],
      "Resource": "arn:aws:s3:::S3_BUCKET_NAME"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject",
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload"
      ],
      "Resource": "arn:aws:s3:::S3_BUCKET_NAME/*"
    }
  ]
}
```

20.2.1.1.1. 将 OpenShift Container Platform 清单配置为使用 S3

流程

将 registry 的 Ansible 清单配置为使用 S3 存储桶和 IAM 用户：

```
[OSEv3:vars]
# AWS Registry Configuration
openshift_hosted_manage_registry=true
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_provider=s3
openshift_hosted_registry_storage_s3_accesskey=AKIAJ6VLREDHATSPBUA (1)
openshift_hosted_registry_storage_s3_secretkey=g/8PmTYDQVGssFWWFvfawHpDbZyGkjGNZhbW
QpjH (2)
openshift_hosted_registry_storage_s3_bucket=openshift-registry-storage (3)
openshift_hosted_registry_storage_s3_region=us-east-1 (4)
openshift_hosted_registry_storage_s3_chunksize=26214400
openshift_hosted_registry_storage_s3_rootdirectory=/registry
openshift_hosted_registry_storage_s3_encrypt=false
openshift_hosted_registry_storage_s3_kmskeyid=aws_kms_key_id (5)
openshift_hosted_registry_pullthrough=true
openshift_hosted_registry_acceptschema2=true
openshift_hosted_registry_enforcequota=true
openshift_hosted_registry_replicas=3
```

(1) (1) IAM 用户的访问密钥。（当前不需要 IAM 角色）

(2) IAM 用户的 secret 密钥。（当前不需要 IAM 角色）

3 S3 存储桶名称。

4 bucket 所在的区域。

5 用于加密集群中数据的加密密钥的 AWS Key Management Service(AWS KMS)密钥 ID。

20.2.1.1.2. 手动将 OpenShift Container Platform registry 配置为使用 S3

要使用 Amazon Web Services(AWS)S3 对象存储，请编辑 registry 的配置文件并挂载到 registry pod。

流程

1. 导出当前的 ***config.yml*** ：

```
$ oc get secret registry-config \
    -o jsonpath='{.data.config\.yml}' -n default | base64 -d \
  >> config.yml.old
```

2. 从旧 ***config.yml*** 创建新配置文件：

```
$ cp config.yml.old config.yml
```

3. 编辑该文件，使其包含 S3 参数。在 registry 配置文件的 **storage** 部分指定 accountname、accountkey、container 和 realm。

```
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  s3:
    accesskey: AKIAJ6VLREDHATSPBUA 1
    secretkey: g/8PmTYDQVGssFWWFvfawHpDbZyGkjGNZhbWQpjH 2
    region: us-east-1 3
    bucket: openshift-registry-storage 4
    encrypt: False
    secure: true
    v4auth: true
    rootdirectory: /registry 5
    chunksize: "26214400"
```

1 使用有权访问 S3 存储桶的 AWS 访问密钥替换。

2 与定义的 AWS 访问密钥对应的 secret 键。

3 用作 registry 的 S3 存储桶的名称。

4 registry 将存储镜像和元数据的位置。（默认为 /registry）

4. 删除 **registry-config** secret：

```
$ oc delete secret registry-config -n default
```

5. 重新创建 secret 来引用更新的配置文件：

```
$ oc create secret generic registry-config \
    --from-file=config.yml -n default
```

6. 重新部署 registry 以读取更新的配置：

```
$ oc rollout latest docker-registry -n default
```

20.2.1.1.3. 验证 registry 是否使用 S3 存储

验证 registry 是否使用 Amazon S3 存储：

流程

1. 在成功部署 registry 后，registry **deploymentconfig** 会将 registry-storage 描述为 **emptydir** 而不是 AWS S3，但 AWS S3 存储桶的配置位于 secret **docker-config** 中。**docker-config** 机密挂载到 **REGISTRY_CONFIGURATION_PATH**，后者在将 AWS S3 用于注册表对象存储时提供所有 paramaters。

```
$ oc describe dc docker-registry -n default
...
    Environment:
      REGISTRY_HTTP_ADDR:     :5000
      REGISTRY_HTTP_NET:     tcp
      REGISTRY_HTTP_SECRET:
SPLR83SDsPaGbGuwSMDfnDwrDRvGf6YXl4h9JQrToQU=
      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_OPENSHIFT_SERVER_ADDR:    docker-registry.default.svc:5000
      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) 1
    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
....
```

1 共享 pod 生命周期的临时目录。

2. 确定 */registry* 挂载点为空：

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}')  -i -t -- ls -l /registry
total 0
```

如果为空，则代表 S3 配置在 **registry-config** secret 中定义：

```
$ oc describe secret registry-config
Name:         registry-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
config.yml:  398 bytes
```

3. 安装程序会使用扩展的 registry 功能创建带有所需配置的 ***config.yml*** 文件，如安装文档所述。要查看配置文件，包括存储存储桶配置的 **storage** 部分：

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
      -o=jsonpath='{.items[0].metadata.name}') \
  cat /etc/registry/config.yml

  version: 0.1
  log:
    level: debug
  http:
    addr: :5000
  storage:
    delete:
      enabled: true
    cache:
      blobdescriptor: inmemory
    s3:
      accesskey: AKIAJ6VLREDHATSPBUA
      secretkey: g/8PmTYDQVGssFWWFvfawHpDbZyGkjGNZhbWQpjH
      region: us-east-1
      bucket: openshift-registry-storage
      encrypt: False
      secure: true
      v4auth: true
      rootdirectory: /registry
      chunksize: "26214400"
  auth:
    openshift:
      realm: openshift
  middleware:
    registry:
    - name: openshift
```

```
    repository:
    - name: openshift
      options:
        pullthrough: true
        acceptschema2: true
        enforcequota: true
    storage:
    - name: openshift
```

另外，您可以查看 secret：

```
$ oc get secret registry-config -o jsonpath='{.data.config\.yml}' | base64 -d
version: 0.1
log:
  level: debug
http:
  addr: :5000
  storage:
    delete:
      enabled: true
    cache:
      blobdescriptor: inmemory
    s3:
      accesskey: AKIAJ6VLREDHATSPBUA
      secretkey: g/8PmTYDQVGssFWWFvfawHpDbZyGkjGNZhbWQpjH
      region: us-east-1
      bucket: openshift-registry-storage
      encrypt: False
      secure: true
      v4auth: true
      rootdirectory: /registry
      chunksize: "26214400"
auth:
  openshift:
    realm: openshift
middleware:
  registry:
  - name: openshift
  repository:
  - name: openshift
  options:
    pullthrough: true
    acceptschema2: true
    enforcequota: true
  storage:
  - name: openshift
```

如果使用 **emptyDir** 卷，**/registry** 挂载点类似如下：

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}')  -i -t -- df -h /registry
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc        100G  226M   30G   1% /registry
```

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}')  -i -t -- ls -l /registry
total 0
drwxr-sr-x. 3 1000000000 1000000000 22 Jun 19 12:24 docker
```

20.3. 配置 AWS 变量

要设置所需的 AWS 变量，请在所有 OpenShift Container Platform 主机上（master 和 节点）创建一个 */etc/origin/cloudprovider/aws.conf* 文件：

```
[Global]
Zone = us-east-1c (1)
```

(1) 这是 AWS 实例的 Availability Zone，且您的 EBS 卷所在的位置 ; 这些信息可从 AWS 管理控制台获取。

20.4. 为 AWS 配置 OPENSHIFT CONTAINER PLATFORM

您可以通过两种方式在 OpenShift Container Platform 上设置 AWS 配置：

- 使用 Ansible 或
- 手动。

20.4.1. 使用 Ansible 为 AWS 配置 OpenShift Container Platform

在集群安装过程中，可以使用 **openshift_cloudprovider_aws_access_key**, **openshift_cloudprovider_aws_secret_key**, **openshift_cloudprovider_kind**, **openshift_clusterid** 参数配置 AWS，这些参数可以在 inventory 文件 中进行配置。

使用 Ansible 的 AWS 配置示例

```
# Cloud Provider Configuration
#
# Note: You may make use of environment variables rather than store
# sensitive configuration within the ansible inventory.
# For example:
#openshift_cloudprovider_aws_access_key="{{ lookup('env','AWS_ACCESS_KEY_ID') }}"
#openshift_cloudprovider_aws_secret_key="{{ lookup('env','AWS_SECRET_ACCESS_KEY') }}"
#
#openshift_clusterid=unique_identifier_per_availablility_zone
#
# AWS (Using API Credentials)
#openshift_cloudprovider_kind=aws
#openshift_cloudprovider_aws_access_key=aws_access_key_id
#openshift_cloudprovider_aws_secret_key=aws_secret_access_key
#
# AWS (Using IAM Profiles)
#openshift_cloudprovider_kind=aws
# Note: IAM roles must exist before launching the instances.
```

注意

当 Ansible 配置 AWS 时，它会自动对以下文件进行必要的更改：

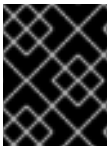
- */etc/origin/cloudprovider/aws.conf*
- */etc/origin/master/master-config.yaml*
- */etc/origin/node/node-config.yaml*

20.4.2. 为 AWS 手动配置 OpenShift Container Platform Master

在所有 master 上编辑 或创建 master 配置文件（默认为*/etc/origin/master/master-config.yaml* ）并更新 **apiServerArguments** 和 **controllerArguments** 部分的内容：

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "aws"
    cloud-config:
      - "/etc/origin/cloudprovider/aws.conf"
  controllerArguments:
    cloud-provider:
      - "aws"
    cloud-config:
      - "/etc/origin/cloudprovider/aws.conf"
```

目前，**nodeName 必须**与 AWS 中的实例名称匹配，以便云供应商集成正常工作。名称也必须与 RFC1123 兼容。



重要

在触发容器化安装时，只有 */etc/origin* 和 */var/lib/origin* 的目录被挂载到 master 和节点容器。因此，*aws.conf* 应该位于 */etc/origin/* 而不是 */etc/* 中。

20.4.3. 为 AWS 手动配置 OpenShift Container Platform 节点

编辑 适当的节点配置映射 并更新 **kubeletArguments** 部分的内容：

```
kubeletArguments:
  cloud-provider:
    - "aws"
  cloud-config:
    - "/etc/origin/cloudprovider/aws.conf"
```

重要

在触发容器化安装时，只有 */etc/origin* 和 */var/lib/origin* 的目录被挂载到 master 和节点容器。因此，*aws.conf* 应该位于 */etc/origin/* 而不是 */etc/* 中。

20.4.4. 手动设置键-值访问对

确保在 master 上的 */etc/origin/master/master.env* 文件中设置以下环境变量，以及节点上的 */etc/sysconfig/atomic-openshift-node* 文件中：

```
AWS_ACCESS_KEY_ID=<key_ID>
AWS_SECRET_ACCESS_KEY=<secret_key>
```

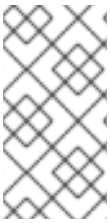
注意

在设置 AWS IAM 用户时获取访问密钥。

20.5. 应用配置更改

在所有 master 和节点主机上启动或重启 OpenShift Container Platform 服务以应用您的配置更改，请参阅重启 OpenShift Container Platform 服务 ：

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



注意

Kubernetes 架构需要来自云提供商的可靠端点。当云提供商停机时，kubelet 会防止 OpenShift Container Platform 重启。如果底层云供应商端点不可靠，请不要安装使用云供应商集成的集群。如在裸机环境中一样安装集群。不建议在已安装的集群中打开或关闭云提供商集成。但是，如果该情境不可避免，请完成以下过程。

从不使用云供应商切换到使用云提供商会产生错误消息。添加云供应商会尝试删除节点，因为从其切换的节点使用 **hostname** 作为 **externalID** （当没有云供应商使用时）使用云供应商的 **instance-id** （由云提供商指定）。要解决这个问题：

1. 以集群管理员身份登录到 CLI。
2. 检查和备份现有节点标签：

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?=Taints)'
```

3. 删除节点：

```
$ oc delete node <node_name>
```

4. 在每个节点主机上，重启 OpenShift Container Platform 服务。

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

5. 在每个主机上重新添加回您以前具有的标记。

20.6. 为 AWS 标记集群

如果配置 AWS 供应商凭证，还必须确保所有主机都被标记为。

要正确识别与集群关联的资源，请使用键 **kubernetes.io/cluster/<clusterid>** 标签资源，其中：

- **<clusterid>** 是集群的唯一名称。

如果节点专门用于集群，将相关值设为 **owned**；如果资源与其他系统共享，将相关值设置为 **shared**。

使用 **kubernetes.io/cluster/<clusterid>,Value=(owned|shared)** 标签标记所有资源，可以避免多个区或多个集群中的潜在问题。

请参阅 Pod 和服务，了解更多有关在 OpenShift Container Platform 中标记和标记的信息。

20.6.1. 需要标签的资源

需要标记四种资源：

- 实例
- 安全组
- Load Balancers
- EBS 卷

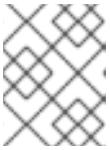
20.6.2. 标记现有集群

集群使用 **kubernetes.io/cluster/<clusterid>,Value=(owned|shared)** 标签的值来决定 AWS 集群的资源。这意味着所有相关资源必须使用与键相同的值的 **kubernetes.io/cluster/<clusterid>,Value=(owned|shared)** 标签进行标记。这些资源包括：

- 所有主机。
- 要在 AWS 实例中使用的所有相关负载均衡器。
- 所有 EBS 卷。需要标记的 EBS 卷可使用以下地址找到：

```
$ oc get pv -o json|jq '.items[].spec.awsElasticBlockStore.volumeID'
```

- 与 AWS 实例一起使用的所有相关安全组。



注意

不要使用 **kubernetes.io/cluster/<name>,Value=<clusterid>** 标签标记所有现有的安全组，否则 Elastic Load Balancing（ELB）将无法创建负载均衡器。

在标记任何资源后，重启 master 上的 master 服务，并在所有节点上重启节点服务。请参阅 应用配置部分。

20.6.3. 关于 Red Hat OpenShift Container Storage

Red Hat OpenShift Container Storage(RHOCS)是 OpenShift Container Platform 内部或混合云中与无关的持久性存储供应商。作为红帽存储解决方案，RHOCS 与 OpenShift Container Platform 完全集成，用于部署、管理和监控，无论它是否安装在 OpenShift Container Platform（融合）或与 OpenShift Container Platform（独立）。OpenShift Container Storage 不仅限于单个可用区或节点，这使得它可能会停机。您可以在 RHOCS 3.11 部署指南中找到使用 RHOCS 的完整说明。

第 21 章 为 RED HAT VIRTUALIZATION 配置

您可以通过创建一个堡垒虚拟机并使用它来安装 OpenShift Container Platform，为 Red Hat Virtualization 配置 OpenShift Container Platform。

21.1. 创建堡垒（BASTION）虚拟机

在 Red Hat Virtualization 中创建堡垒主机来安装 OpenShift Container Platform。

流程

1. 使用 SSH 登录到 Manager 机器。
2. 为安装文件创建一个临时 bastion 安装目录，如 */bastion_installation*。
3. 使用 **ansible-vault** 创建加密的 */bastion_installation/secure_vars.yaml* 文件并记录密码：

```
# ansible-vault create secure_vars.yaml
```

4. 在 *secure_vars.yaml* 文件中添加以下参数值：

```
engine_password: <Manager_password> 1
bastion_root_password: <bastion_root_password> 2
rhsub_user: <Red_Hat_Subscription_Manager_username> 3
rhsub_pass: <Red_Hat_Subscription_Manager_password>
rhsub_pool: <Red_Hat_Subscription_Manager_pool_id> 4
root_password: <OpenShift_node_root_password> 5
engine_cafile: <RHVM_CA_certificate> 6
oreg_auth_user: <image_registry_authentication_username> 7
oreg_auth_password: <image_registry_authentication_password>
```

1. 登录管理门户的密码。
2. 堡垒虚拟机的 root 密码。
3. Red Hat Subscription Manager 凭证。
4. Red Hat Virtualization Manager 订阅池的池 ID。
5. OpenShift Container Platform root 密码。
6. Red Hat Virtualization Manager CA 证书。如果您不从 Manager 机器运行 playbook，则需要 **engine_cafile** 值。Manager CA 证书的默认位置为 */etc/pki/ovirt-engine/ca.pem*。
7. 如果您使用需要身份验证的镜像 registry，请添加凭证。

5. 保存该文件。
6. 获取 ***Red Hat Enterprise Linux KVM 客户机镜像***下载链接：
   a. 访问 Red Hat Customer Portal:下载 Red Hat Enterprise Linux 。
   b. 在**产品软件** 选项卡中，找到 ***Red Hat Enterprise Linux KVM 客户机镜像***

c. 右键单击 **Download Now**，复制 链接并保存。
该链接对时间敏感，且必须先复制，然后才能创建 bastion 虚拟机。

7. 使用以下内容创建 */bastion_installation/create-bastion-machine-playbook.yaml* 文件并更新其参数值：

```
---
- name: Create a bastion machine
  hosts: localhost
  connection: local
  gather_facts: false
  no_log: true

  roles:
    - oVirt.image-template
    - oVirt.vm-infra
  no_log: true

  vars:
    engine_url: https://_Manager_FQDN_/ovirt-engine/api (1)
    engine_user: <admin@internal>
    engine_password: "{{ engine_password }}"
    engine_cafile: /etc/pki/ovirt-engine/ca.pem

    qcow_url: <RHEL_KVM_guest_image_download_link> (2)
    template_cluster: Default
    template_name: rhelguest7
    template_memory: 4GiB
    template_cpu: 2
    wait_for_ip: true
    debug_vm_create: false

    vms:
    - name: rhel-bastion
      cluster: "{{ template_cluster }}"
      profile:
        cores: 2
        template: "{{ template_name }}"
        root_password: "{{ root_password }}"
        ssh_key: "{{ lookup('file', '/root/.ssh/id_rsa_ssh_ocp_admin.pub') }}"
        state: running
      cloud_init:
        custom_script: |
          rh_subscription:
            username: "{{ rhsub_user }}"
            password: "{{ rhsub_pass }}"
            auto-attach: true
            disable-repo: ['*']
            # 'rhel-7-server-rhv-4.2-manager-rpms' supports RHV 4.2 and 4.3
            enable-repo: ['rhel-7-server-rpms', 'rhel-7-server-extras-rpms', 'rhel-7-server-ansible-2.7-rpms', 'rhel-7-server-ose-3.11-rpms', 'rhel-7-server-supplementary-rpms', 'rhel-7-server-rhv-4.2-manager-rpms']
          packages:
            - ansible
            - ovirt-ansible-roles
            - openshift-ansible
```

```
        - python-ovirt-engine-sdk4
  pre_tasks:
    - name: Create an ssh key-pair for OpenShift admin
      user:
        name: root
        generate_ssh_key: yes
        ssh_key_file: .ssh/id_rsa_ssh_ocp_admin

  roles:
    - oVirt.image-template
    - oVirt.vm-infra

- name: post installation tasks on the bastion machine
  hosts: rhel-bastion
  tasks:
    - name: create ovirt-engine PKI dir
      file:
        state: directory
        dest: /etc/pki/ovirt-engine/
    - name: Copy the engine ca cert to the bastion machine
      copy:
        src: "{{ engine_cafile }}"
        dest: "{{ engine_cafile }}"
    - name: Copy the secured vars to the bastion machine
      copy:
        src: secure_vars.yaml
        dest: secure_vars.yaml
        decrypt: false
    - file:
        state: directory
        path: /root/.ssh
    - name: copy the OpenShift_admin keypair to the bastion machine
      copy:
        src: "{{ item }}"
        dest: "{{ item }}"
        mode: 0600
      with_items:
        - /root/.ssh/id_rsa_ssh_ocp_admin
        - /root/.ssh/id_rsa_ssh_ocp_admin.pub
```

1. Manager 机器的 FQDN。
2. **<qcow_url>** 是 ***Red Hat Enterprise Linux KVM 客户机镜像***的下载链接。***Red Hat Enterprise Linux KVM 客户机镜像*** 包含 **cloud-init** 软件包，此 playbook 需要该软件包。如果没有使用 Red Hat Enterprise Linux，请下载 **cloud-init** 软件包，并在运行此 playbook 前手动安装它。

8. 创建堡垒虚拟机：

```
# ansible-playbook -i localhost create-bastion-machine-playbook.yaml -e @secure_vars.yaml
--ask-vault-pass
```

9. 登录管理门户。
10. 点击 **Compute → Virtual Machines** 来验证 ***rhel-bastion*** 虚拟机是否已成功创建。

21.2. 使用堡垒虚拟机安装 OPENSHIFT CONTAINER PLATFORM

使用 Red Hat Virtualization 中的 bastion 虚拟机安装 OpenShift Container Platform。

流程

1. 登录 ***rhel-bastion***。
2. 创建一个包含以下内容的 ***install_ocp.yaml*** 文件：

```
---
- name: Openshift on RHV
  hosts: localhost
  connection: local
  gather_facts: false

  vars_files:
    - vars.yaml
    - secure_vars.yaml

  pre_tasks:
    - ovirt_auth:
        url:      "{{ engine_url }}"
        username: "{{ engine_user }}"
        password: "{{ engine_password }}"
        insecure: "{{ engine_insecure }}"
        ca_file:  "{{ engine_cafile | default(omit) }}"

  roles:
    - role: openshift_ovirt

- import_playbook: setup_dns.yaml
- import_playbook: /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
- import_playbook: /usr/share/ansible/openshift-ansible/playbooks/openshift-
node/network_manager.yml
- import_playbook: /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

3. 创建一个包含以下内容的 ***setup_dns.yaml*** 文件：

```
- hosts: masters
  strategy: free
  tasks:
    - shell: "echo {{ ansible_default_ipv4.address }} {{ inventory_hostname }} etcd.{{
inventory_hostname.split('.', 1)[1] }} openshift-master.{{ inventory_hostname.split('.', 1)[1] }}
openshift-public-master.{{ inventory_hostname.split('.', 1)[1] }} docker-registry-default.apps.{{
inventory_hostname.split('.', 1)[1] }} webconsole.openshift-web-console.svc registry-console-
default.apps.{{ inventory_hostname.split('.', 1)[1] }} >> /etc/hosts"
      when: openshift_ovirt_all_in_one is defined | ternary((openshift_ovirt_all_in_one | bool),
false)
```

4. 创建一个包含以下内容的 ***/etc/ansible/openshift_3_11.hosts*** Ansible 清单文件：

```
[workstation]
localhost ansible_connection=local
```

```
[all:vars]
openshift_ovirt_dns_zone="{{ public_hosted_zone }}"
openshift_web_console_install=true
openshift_master_overwrite_named_certificates=true
openshift_master_cluster_hostname="openshift-master.{{ public_hosted_zone }}"
openshift_master_cluster_public_hostname="openshift-public-master.{{ public_hosted_zone
}}"
openshift_master_default_subdomain="{{ public_hosted_zone }}"
openshift_public_hostname="{{openshift_master_cluster_public_hostname}}"
openshift_deployment_type=openshift-enterprise
openshift_service_catalog_image_version="{{ openshift_image_tag }}"

[OSEv3:vars]
# General variables
debug_level=1
containerized=false
ansible_ssh_user=root
os_firewall_use_firewalld=true
openshift_enable_excluders=false
openshift_install_examples=false
openshift_clock_enabled=true
openshift_debug_level="{{ debug_level }}"
openshift_node_debug_level="{{ node_debug_level | default(debug_level,true) }}"
osn_storage_plugin_deps=[]
openshift_master_bootstrap_auto_approve=true
openshift_master_bootstrap_auto_approver_node_selector={"node-
role.kubernetes.io/master":"true"}
osm_controller_args={"experimental-cluster-signing-duration": ["20m"]}
osm_default_node_selector="node-role.kubernetes.io/compute=true"
openshift_enable_service_catalog=false

# Docker
container_runtime_docker_storage_type=overlay2
openshift_docker_use_system_container=false

[OSEv3:children]
nodes
masters
etcd
lb

[masters]
[nodes]
[etcd]
[lb]
```

5. 获取 ***Red Hat Enterprise Linux KVM 客户机镜像***下载链接：

   a. 访问 Red Hat Customer Portal:下载 Red Hat Enterprise Linux 。

   b. 在**产品软件** 选项卡中，找到 ***Red Hat Enterprise Linux KVM 客户机镜像***

   c. 右键单击 **Download Now**，复制 链接并保存。
   不要使用您在创建堡垒虚拟机时复制的链接。下载链接区分大小写，且必须先复制后才能运行安装 playbook。

6. 使用以下内容创建 *vars.yaml* 文件并更新其参数值：

```
---
# For detailed documentation of variables, see
# openshift_ovirt: https://github.com/openshift/openshift-
ansible/tree/master/roles/openshift_ovirt#role-variables
# openshift installation: https://github.com/openshift/openshift-ansible/tree/master/inventory
engine_url: https://<Manager_FQDN>/ovirt-engine/api (1)
engine_user: admin@internal
engine_password: "{{ engine_password }}"
engine_insecure: false
engine_cafile: /etc/pki/ovirt-engine/ca.pem

openshift_ovirt_vm_manifest:
  - name: 'master'
    count: 1
    profile: 'master_vm'
  - name: 'compute'
    count: 0
    profile: 'node_vm'
  - name: 'lb'
    count: 0
    profile: 'node_vm'
  - name: 'etcd'
    count: 0
    profile: 'node_vm'
  - name: infra
    count: 0
    profile: node_vm

# Currently, only all-in-one installation (`openshift_ovirt_all_in_one: true`) is supported.
# Multi-node installation (master and node VMs installed separately) will be supported in a
future release.
openshift_ovirt_all_in_one: true
openshift_ovirt_cluster: Default
openshift_ovirt_data_store: data
openshift_ovirt_ssh_key: "{{ lookup('file', '/root/.ssh/id_rsa_ssh_ocp_admin.pub') }}"

public_hosted_zone:
# Uncomment to disable install-time checks, for smaller scale installations
#openshift_disable_check: memory_availability,disk_availability,docker_image_availability

qcow_url: <RHEL_KVM_guest_image_download_link> (2)
image_path: /var/tmp
template_name: rhelguest7
template_cluster: "{{ openshift_ovirt_cluster }}"
template_memory: 4GiB
template_cpu: 1
template_disk_storage: "{{ openshift_ovirt_data_store }}"
template_disk_size: 100GiB
template_nics:
  - name: nic1
    profile_name: ovirtmgmt
    interface: virtio

debug_vm_create: false
```

```
wait_for_ip: true
vm_infra_wait_for_ip_retries: 30
vm_infra_wait_for_ip_delay: 20

node_item: &node_item
  cluster: "{{ openshift_ovirt_cluster }}"
  template: "{{ template_name }}"
  memory: "8GiB"
  cores: "2"
  high_availability: true
  disks:
    - name: docker
      size: 15GiB
      interface: virtio
      storage_domain: "{{ openshift_ovirt_data_store }}"
    - name: openshift
      size: 30GiB
      interface: virtio
      storage_domain: "{{ openshift_ovirt_data_store }}"
  state: running
  cloud_init:
    root_password: "{{ root_password }}"
    authorized_ssh_keys: "{{ openshift_ovirt_ssh_key }}"
    custom_script: "{{ cloud_init_script_node | to_nice_yaml }}"

openshift_ovirt_vm_profile:
  master_vm:
    <<: *node_item
    memory: 16GiB
    cores: "{{ vm_cores | default(4) }}"
    disks:
      - name: docker
        size: 15GiB
        interface: virtio
        storage_domain: "{{ openshift_ovirt_data_store }}"
      - name: openshift_local
        size: 30GiB
        interface: virtio
        storage_domain: "{{ openshift_ovirt_data_store }}"
      - name: etcd
        size: 25GiB
        interface: virtio
        storage_domain: "{{ openshift_ovirt_data_store }}"
    cloud_init:
      root_password: "{{ root_password }}"
      authorized_ssh_keys: "{{ openshift_ovirt_ssh_key }}"
      custom_script: "{{ cloud_init_script_master | to_nice_yaml }}"
  node_vm:
    <<: *node_item
  etcd_vm:
    <<: *node_item
  lb_vm:
    <<: *node_item

cloud_init_script_node: &cloud_init_script_node
  packages:
```

```
    - ovirt-guest-agent
  runcmd:
    - sed -i 's/# ignored_nics =.*/ignored_nics = docker0 tun0 /' /etc/ovirt-guest-agent.conf
    - systemctl enable ovirt-guest-agent
    - systemctl start ovirt-guest-agent
    - mkdir -p /var/lib/docker
    - mkdir -p /var/lib/origin/openshift.local.volumes
    - /usr/sbin/mkfs.xfs -L dockerlv /dev/vdb
    - /usr/sbin/mkfs.xfs -L ocplv /dev/vdc
  mounts:
    - [ '/dev/vdb', '/var/lib/docker', 'xfs', 'defaults,gquota' ]
    - [ '/dev/vdc', '/var/lib/origin/openshift.local.volumes', 'xfs', 'defaults,gquota' ]
  power_state:
    mode: reboot
    message: cloud init finished - boot and install openshift
    condition: True
cloud_init_script_master:
  <<: *cloud_init_script_node
  runcmd:
    - sed -i 's/# ignored_nics =.*/ignored_nics = docker0 tun0 /' /etc/ovirt-guest-agent.conf
    - systemctl enable ovirt-guest-agent
    - systemctl start ovirt-guest-agent
    - mkdir -p /var/lib/docker
    - mkdir -p /var/lib/origin/openshift.local.volumes
    - mkdir -p /var/lib/etcd
    - /usr/sbin/mkfs.xfs -L dockerlv /dev/vdb
    - /usr/sbin/mkfs.xfs -L ocplv /dev/vdc
    - /usr/sbin/mkfs.xfs -L etcdlv /dev/vdd
  mounts:
    - [ '/dev/vdb', '/var/lib/docker', 'xfs', 'defaults,gquota' ]
    - [ '/dev/vdc', '/var/lib/origin/openshift.local.volumes', 'xfs', 'defaults,gquota' ]
    - [ '/dev/vdd', '/var/lib/etcd', 'xfs', 'defaults,gquota' ]
```

1. Manager 机器的 FQDN。
2. **<qcow_url>** 是 ***Red Hat Enterprise Linux KVM 客户机镜像***的下载链接。***Red Hat Enterprise Linux KVM 客户机镜像*** 包含 **cloud-init** 软件包，此 playbook 需要该软件包。如果没有使用 Red Hat Enterprise Linux，请下载 **cloud-init** 软件包，并在运行此 playbook 前手动安装它。

7. 安装 OpenShift Container Platform：

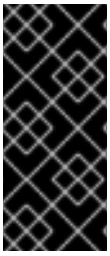
```
# export ANSIBLE_ROLES_PATH="/usr/share/ansible/roles/:/usr/share/ansible/openshift-ansible/roles"
# export ANSIBLE_JINJA2_EXTENSIONS="jinja2.ext.do"
# ansible-playbook -i /etc/ansible/openshift_3_11.hosts install_ocp.yaml -e @vars.yaml -e @secure_vars.yaml --ask-vault-pass
```

8. 为每个基础架构实例创建路由器的 DNS 条目。
9. 配置循环路由，以便路由器可以将流量传递到应用。
10. 为 OpenShift Container Platform Web 控制台创建 DNS 条目。
11. 指定负载均衡器节点的 IP 地址。

第 22 章 为 OPENSTACK 配置

22.1. 概述

在 OpenStack 上部署时，OpenShift Container Platform 可以配置为访问 OpenStack 基础架构， 包括使用 OpenStack Cinder 卷作为应用程序数据的持久性存储。



重要

OpenShift Container Platform 3.11 支持与 Red Hat OpenStack Platform 13 搭配使用。

最新的 OpenShift Container Platform 版本支持最新的 Red Hat OpenStack Platform 长生命版本和中间版本。OpenShift Container Platform 和 Red Hat OpenStack Platform 的发行周期不同，今后测试的版本会根据两个产品的发行日期的不同而有所不同。

22.2. 开始前

22.2.1. OpenShift Container Platform SDN

默认的 OpenShift Container Platform SDN 是 OpenShiftSDN。还有另一个选项： 使用 Kuryr SDN。

22.2.2. Kuryr SDN

Kuryr 是一个 CNI 插件，它使用 Neutron 和 Octavia 为 pod 和服务提供网络。它主要针对在 OpenStack 虚拟机上运行的 OpenShift Container Platform 集群设计。Kuryr 通过将 OpenShift Container Platform pod 插入 OpenStack SDN 来提高网络性能。另外，它还提供 OpenShift Container Platform pod 和 OpenStack 虚拟实例间的互联性。

建议在封装的 OpenStack 租户网络上部署 OpenShift Container Platform 时使用 Kuryr，以避免出现重复封装，例如通过 OpenStack 网络运行封装的 OpenShift SDN。每当需要 VXLAN、GRE 或 GENEVE 时，都建议使用 Kuryr。

因此，在以下情况下使用 Kuryr 并不有意义：

- 您可以使用提供商网络、租户 VLAN 或第三方商业 SDN，如 Cisco ACI 或 Juniper Contrail。
- 部署将在几个虚拟机监控程序或 OpenShift Container Platform 虚拟机节点上使用多个服务。每个 OpenShift Container Platform 服务会在 OpenStack 中创建一个 Octavia Amphora 虚拟机，它托管所需的负载均衡器。

要启用 Kuryr SDN，您的环境必须满足以下要求：

- 运行 OpenStack 13 或更高版本
- overcloud 带有 Octavia
- 启用 Neutron Trunk 端口扩展
- 如果使用 ML2/OVS Neutron 驱动程序，则必须使用 OpenvSwitch 防火墙驱动程序，而不是 ovs-hybrid。

重要

要将 Kuryr 与 OpenStack 13.0.13 搭配使用，Kuryr 容器镜像必须是版本 3.11.306 或更高版本。

22.2.3. OpenShift Container Platform 先决条件

成功部署 OpenShift Container Platform 需要满足很多前提条件。其中包括一组基础架构和主机配置步骤，然后再使用 Ansible 进行 OpenShift Container Platform 的实际安装。在后续小节中，详细介绍了 OpenStack 环境中 OpenShift Container Platform 所需的前提条件和配置更改。

注意

本参考环境中的所有 OpenStack CLI 命令都使用 CLI **openstack** 命令从 director 节点的不同节点执行。在另一节点中执行命令，以避免与 Ansible 版本 2.6 及更高版本产生软件包冲突。务必在指定的软件仓库中安装以下软件包。

例如：

从 Set Up Repositories 启用 rhel-7-server-openstack-13-tools-rpms 和所需的 OpenShift Container Platform 存储库。

```
$ sudo subscription-manager repos \
--enable rhel-7-server-openstack-{rhosp_version}-tools-rpms \
--enable rhel-7-server-openstack-14-tools-rpms
$ sudo subscription-manager repo-override --repo=rhel-7-server-openstack-14-tools-rpms --
add=includepkgs:"python2-openstacksdk.* python2-keystoneauth1.* python2-os-service-types.*"
$ sudo yum install -y python2-openstackclient python2-heatclient python2-octaviaclient ansible
```

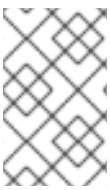
验证软件包是否至少包含以下版本（使用 **rpm -q <package_name>**）：

- **python2-openstackclient - 3.14.1.-1**
- **python2-heatclient 1.14.0-1**
- **python2-octaviaclient 1.4.0-1**
- **python2-openstacksdk 0.17.2**

22.2.3.1. 启用 Octavia：OpenStack 负载平衡即服务(LBaaS)

Octavia 是一个支持的负载均衡器解决方案，建议与 OpenShift Container Platform 一起使用，以便对外部传入的流量进行负载平衡，并为应用程序提供 OpenShift Container Platform master 服务的单一视图。

要启用 Octavia，必须包含在安装 OpenStack overcloud 的过程中，如果 overcloud 已存在则需要升级 Octavia 服务。以下步骤提供启用 Octavia 的基本非自定义步骤，并适用于 overcloud 的干净安装或 overcloud 更新。



注意

以下步骤只包括在部署 OpenStack 时需要处理 Octavia 部分的信息。有关更多信息，请参阅安装 OpenStack 文档。请注意对于节点 registry 方法会有所不同。有关更多信息，请参阅 Registry 方法 文档。这个示例使用了本地 registry 方法。

如果使用本地 registry，请创建一个模板来将镜像上传到 registry。如下例所示。

```
(undercloud) $ openstack overcloud container image prepare \
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/octavia.yaml \
--namespace=registry.access.redhat.com/rhosp13 \
--push-destination=<local-ip-from-undercloud.conf>:8787 \
--prefix=openstack- \
--tag-from-label {version}-{release} \
--output-env-file=/home/stack/templates/overcloud_images.yaml \
--output-images-file /home/stack/local_registry_images.yaml
```

验证创建的 ***local_registry_images.yaml*** 是否包含 Octavia 镜像。

本地 registry 文件中的 Octavia 镜像

```
...
- imagename: registry.access.redhat.com/rhosp13/openstack-octavia-api:13.0-43
  push_destination: <local-ip-from-undercloud.conf>:8787
- imagename: registry.access.redhat.com/rhosp13/openstack-octavia-health-manager:13.0-45
  push_destination: <local-ip-from-undercloud.conf>:8787
- imagename: registry.access.redhat.com/rhosp13/openstack-octavia-housekeeping:13.0-45
  push_destination: <local-ip-from-undercloud.conf>:8787
- imagename: registry.access.redhat.com/rhosp13/openstack-octavia-worker:13.0-44
  push_destination: <local-ip-from-undercloud.conf>:8787
```

注意

Octavia 容器的版本会根据安装的特定 Red Hat OpenStack Platform 版本而有所不同。

以下步骤从 **registry.redhat.io** 拉取容器镜像到 undercloud 节点。这个过程可能需要一些时间，具体要看网络和 undercloud 磁盘的速度。

```
(undercloud) $ sudo openstack overcloud container image upload \
  --config-file  /home/stack/local_registry_images.yaml \
  --verbose
```

当 Octavia Load Balancer 用于访问 OpenShift API 时，需要增加它们的监听程序默认超时时间。默认超时为 50 秒。通过将以下文件传递给 overcloud deploy 命令，将超时时间增加到 20 分钟：

```
(undercloud) $ cat octavia_timeouts.yaml
parameter_defaults:
  OctaviaTimeoutClientData: 1200000
  OctaviaTimeoutMemberData: 1200000
```

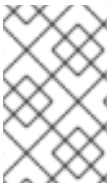
注意

在 Red Hat OpenStack Platform 14 起，不需要这样做。

使用 Octavia 安装或更新 overcloud 环境：

```
openstack overcloud deploy --templates \
.
```

```
.
.
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/octavia.yaml \
  -e octavia_timeouts.yaml
.
.
.
```



注意

以上命令只包含与 Octavia 相关的文件。此命令将根据您具体的 OpenStack 安装而有所不同。如需更多信息，请参阅官方 OpenStack 文档 。有关自定义 Octavia 安装的详情请参考 使用 Director 安装 Octavia。

注意

如果使用 Kuryr SDN，overcloud 安装需要在 Neutron 中启用"中继"扩展。这在 Director 部署中默认启用。当 Neutron 后端是 ML2/OVS 时，使用 openvswitch 防火墙而不是默认的 ovs-hybrid。如果后端为 ML2/OVN，则不需要修改。

22.2.3.2. 创建 OpenStack 用户帐户、项目和角色

在安装 OpenShift Container Platform 之前，Red Hat OpenStack Platform (RHOSP) 环境需要一个项目（通常称为 *租户*），后者存储了要安装的 OpenShift Container Platform 的 OpenStack 实例。此项目要求用户的所有权，并且该用户的角色设置为 **_member_**。

以下步骤演示了如何完成上述操作。

作为 OpenStack overcloud 管理员，

1. 创建一个项目（租户），用于存储 RHOSP 实例

```
$ openstack project create <project>
```

2. 创建拥有之前创建的项目的 RHOSP 用户：

```
$ openstack user create --password <password> <username>
```

3. 设置用户的角色：

```
$ openstack role add --user <username> --project <project> _member_
```

分配给新的 RH OSP 项目的默认配额不足以用于 OpenShift Container Platform 安装。将配额增加到至少 30 个安全组、200 个安全组规则和 200 端口。

```
$ openstack quota set --secgroups 30 --secgroup-rules 200 --ports 200 <project>
1
```

1 对于 **<project >**，请指定要修改的项目名称

22.2.3.3. Kuryr SDN 的额外步骤

如果启用了 Kuryr SDN，特别是在使用命名空间隔离时，增加项目的配额以满足这些最低要求：

- 300 个安全组 - 每个命名空间对应一个命名空间，每个负载均衡器都对应一个
- 150 网络 - 每个命名空间有一个
- 150 个子网 - 每个命名空间都有一个
- 500 个安全组规则
- 500 端口 - 每个 Pod 一个端口，以及池的额外端口用于加快 Pod 创建

注意

这不是全球推荐。调整您的配额以满足您的要求。

如果使用命名空间隔离，则每个命名空间都会获得一个新的网络和子网。另外，还会创建一个安全组来启用命名空间中的 Pod 间的流量。

```
$ openstack quota set --networks 150 --subnets 150 --secgroups 300 --secgroup-rules 500 --ports
500 <project>
1
```

1 对于 **<project >**，请指定要修改的项目名称

如果启用了命名空间隔离，您必须在创建项目后将项目 ID 添加到 **octavia.conf** 配置文件中。此步骤可确保所需的 LoadBalancer 安全组属于该项目，并可将其更新为在命名空间间强制实施服务隔离。

1. 获取项目 ID

```
$ openstack project show *<project>*
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | default                          |
| enabled     | True                             |
| id          | PROJECT_ID                       |
| is_domain   | False                            |
| name        | *<project>*                      |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

2. 将项目 ID 添加到控制器上的 [filename]octavia.conf 中，然后重新启动 octavia worker。

```
$ source stackrc  # undercloud credentials
$ openstack server list
+--------------------------------------+-------------+--------+----------------------+----------------+-----
-------+
|
| ID                                   | Name        | Status | Networks
| Image          | Flavor     |
|
```

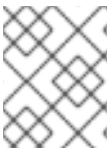
```
+--------------------------------------+-------------+--------+----------------------+---------------+-----
-------+
|
| 6bef8e73-2ba5-4860-a0b1-3937f8ca7e01 | controller-0 | ACTIVE |
ctlplane=192.168.24.8 | overcloud-full | controller |
|
| dda3173a-ab26-47f8-a2dc-8473b4a67ab9 | compute-0    | ACTIVE |
ctlplane=192.168.24.6 | overcloud-full | compute    |
|
+--------------------------------------+-------------+--------+----------------------+---------------+-----
-------+

$ ssh heat-admin@192.168.24.8  # ssh into the controller(s)

controller-0$ vi /var/lib/config-data/puppet-generated/octavia/etc/octavia/octavia.conf
[controller_worker]
# List of project ids that are allowed to have Load balancer security groups
# belonging to them.
amp_secgroup_allowed_projects = PROJECT_ID

controller-0$ sudo docker restart octavia_worker
```

22.2.3.4. 配置 RC 文件

配置完项目后，OpenStack 管理员可以创建对实施 OpenShift Container Platform 环境的用户所需的信息的 RC 文件。

一个 RC 文件示例：

```
$ cat path/to/examplerc
# Clear any old environment that may conflict.
for key in $( set | awk '{FS="="}  /^OS_/ {print $1}' ); do unset $key ; done
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=<project-name>
export OS_USERNAME=<username>
export OS_PASSWORD=<password>
export OS_AUTH_URL=http://<ip>:5000//v3
export OS_CLOUDNAME=<cloud-name>
export OS_IDENTITY_API_VERSION=3

# Add OS_CLOUDNAME to PS1
if [ -z "${CLOUDPROMPT_ENABLED:-}" ]; then
 export PS1=${PS1:-""}
 export PS1=\${OS_CLOUDNAME:+"(\$OS_CLOUDNAME)"}\ $PS1
 export CLOUDPROMPT_ENABLED=1
fi
```



注意

只要引用同一域，就支持从默认值更改 _OS_PROJECT_DOMAIN_NAME 和 _OS_USER_DOMAIN_NAME。

作为在 OpenStack director 节点或工作站中实施 OpenShift Container Platform 环境的用户，请确保如下所示 **source** 凭证：

```
$ source path/to/examplerc
```

22.2.3.5. 创建 OpenStack 类别

在 OpenStack 中，类别通过定义 **nova** 计算实例的计算、内存和存储容量来定义虚拟服务器的大小。由于此参考架构中的基础镜像是 Red Hat Enterprise Linux 7.5，因此使用以下规格创建 **m1.node** 和 **m1.master** 大小，如 表 22.1 “OpenShift 的最低系统要求” 所示。

重要

虽然最小系统要求足以运行集群，以提高性能，但建议在 master 节点上增加 vCPU。另外，如果 etcd 位于 master 节点上，则建议更多内存。

表 22.1. OpenShift 的最低系统要求

节点类型	CPU	RAM	根磁盘	Flavor
Master	4	16 GB	45 GB	**m1.master**
节点	1	8 GB	20 GB	**m1.node**

作为 OpenStack 管理员，

```
$ openstack flavor create <flavor_name> \
    --id auto \
    --ram <ram_in_MB> \
    --disk <disk_in_GB> \
    --vcpus <num_vcpus>
```

以下示例演示了在本参考环境中创建类别。

```
$ openstack flavor create m1.master \
    --id auto \
    --ram 16384 \
    --disk 45 \
    --vcpus 4
$ openstack flavor create m1.node \
    --id auto \
    --ram 8192 \
    --disk 20 \
    --vcpus 1
```

注意

如果访问 OpenStack 管理员权限来创建新类别不可用，请在 OpenStack 环境中使用现有类别来满足 表 22.1 “OpenShift 的最低系统要求” 中的要求。

通过以下方法验证 OpenStack 类别：

```
$ openstack flavor list
```

22.2.3.6. 创建 OpenStack 密钥对

Red Hat OpenStack Platform 使用 **cloud-init** 将 **ssh** 公钥放在每个实例上，允许 **ssh** 访问该实例。Red Hat OpenStack Platform 期望用户存放私钥。

警告

丢失私钥将导致无法访问实例。

要生成密钥对，请使用以下命令：

```
$ openstack keypair create <keypair-name> > /path/to/<keypair-name>.pem
```

对密钥对创建进行验证可通过以下方法完成：

```
$ openstack keypair list
```

创建了密钥对后，将权限设置为 **600**，从而仅允许文件所有者来读取和写入该文件。

```
$ chmod 600 /path/to/<keypair-name>.pem
```

22.2.3.7. 为 OpenShift Container Platform 设置 DNS

DNS 服务是 OpenShift Container Platform 环境中的一个重要组件。无论 DNS 的供应商如何，机构都需要有特定记录才能为各种 OpenShift Container Platform 组件提供服务。

警告

使用 **/etc/hosts** 无效，必须存在正确的 DNS 服务。

通过使用 DNS 的关键 secret，您可以为 OpenShift Ansible 安装程序提供信息，它将自动为目标实例和各种 OpenShift Container Platform 组件添加记录。稍后将在配置 OpenShift Ansible 安装程序时描述此过程设置。

预期可以访问 DNS 服务器。您可以使用 Red Hat Labs DNS 帮助程序 获得访问权限的帮助。

应用程序 DNS

OpenShift 提供的应用可通过端口 80/TCP 和 443/TCP 上的路由器访问。路由器使用*通配符*记录将特定子域下的所有主机名映射到同一 IP 地址，而无需为每个名称单独记录。

这允许 OpenShift Container Platform 添加带有任意名称的应用程序，只要它们位于那个子域下。

例如，*. **apps.example.com** 的通配符记录会导致 DNS 名称查找 **tax.apps.example.com** 和 **home-goods.apps.example.com**，两者返回相同的 IP 地址：**10.19.x.y**.所有流量都转发到 OpenShift 路由器。路由器检查查询的 HTTP 标头并将其转发到正确的目的地。

使用负载均衡器（如 Octavia、主机地址 10.19.x.y）时，可以添加通配符 DNS 记录，如下所示：

表 22.2. 负载均衡器 DNS 记录

IP 地址	Hostname	用途
10.19.x.y	***.apps.example.com**	对应用程序 Web 服务的用户访问

22.2.3.8. 通过 OpenStack 创建 OpenShift Container Platform 网络

在 Red Hat OpenStack Platform 上部署 OpenShift Container Platform 时，该片段中描述的要求有两个网络是 *public* -public 和 *内部网络*。

公共网络

公共网络 是包含外部访问的网络，可以被外部世界访问。*公共*网络创建只能由 OpenStack 管理员执行。

以下命令提供了为 *公共网络* 访问创建 OpenStack 提供商网络的示例。

作为 OpenStack 管理员（overcloudrc 访问），

```
$ source /path/to/examplerc

$ openstack network create <public-net-name> \
  --external \
  --provider-network-type flat \
  --provider-physical-network datacentre

$ openstack subnet create <public-subnet-name> \
  --network <public-net-name> \
  --dhcp \
  --allocation-pool start=<float_start_ip>,end=<float_end_ip> \
  --gateway <ip> \
  --subnet-range <CIDR>
```

创建网络和子网后，通过以下方法验证：

```
$ openstack network list
$ openstack subnet list
```

注意

<float_start_ip & **gt; 和 <float_end_ip** > 是提供给标记为 *公共网络* 的浮动 IP 池。无类别域间路由(CIDR)采用 **<ip>/<routing_prefix>** 格式，如 10.0.0.1/24。

内部网络

在网络设置过程中，*内部网络*通过路由器连接到*公共网络*。这样，每个附加到*内部*网络的 Red Hat OpenStack Platform 实例都可以从*公共*网络请求浮动 IP 进行公共访问。通过设置 **openshift_openstack_private_network_name** 来自动由 OpenShift Ansible 安装程序自动创建 *内部网络*。稍后介绍了有关 OpenShift Ansible 安装程序所需更改的更多信息。

22.2.3.9. 创建 OpenStack 部署主机安全组

OpenStack 网络允许用户定义入站和出站流量过滤器，可以应用到网络上的每个实例。这允许用户根据实例服务的功能限制每个实例的网络流量，而不依赖于基于主机的过滤。OpenShift Ansible 安装程序处理除部署主机外，为 OpenShift Container Platform 集群一部分的每种主机需要的所有端口和服务正确创建。

以下命令创建一个空安全组，没有为部署主机设置规则。

```
$ source path/to/examplerc
$ openstack security group create <deployment-sg-name>
```

验证安全组的创建：

```
$ openstack security group list
```

部署主机安全组

部署实例只需要允许入站 **ssh**。此实例存在，供操作员提供一个稳定的基础，以部署、监控和管理 OpenShift Container Platform 环境。

表 22.3. 部署主机安全组 TCP 端口

端口/协议	服务	远程源	用途
ICMP	ICMP	任意	允许 ping、traceroute 等。
22/TCP	SSH	任意	安全 shell 登录

创建上述安全组规则如下：

```
$ source /path/to/examplerc
$ openstack security group rule create \
    --ingress \
    --protocol icmp \
    <deployment-sg-name>
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 22 \
    <deployment-sg-name>
```

安全组规则验证如下：

```
$ openstack security group rule list <deployment-sg-name>
+--------------------------------------+-------------+-----------+-----------+----------------------+
| ID                                   | IP Protocol | IP Range  | Port Range | Remote Security Group |
```

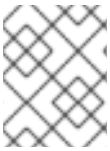
```
+--------------------------------------+------------+----------+-----------+----------------------+
| 7971fc03-4bfe-4153-8bde-5ae0f93e94a8 | icmp       | 0.0.0.0/0 |           | None                 |
| b8508884-e82b-4ee3-9f36-f57e1803e4a4 | None       | None      |           | None                 |
| cb914caf-3e84-48e2-8a01-c23e61855bf6 | tcp        | 0.0.0.0/0 | 22:22     | None                 |
| e8764c02-526e-453f-b978-c5ea757c3ac5 | None       | None      |           | None                 |
+--------------------------------------+------------+----------+-----------+----------------------+
```

22.2.3.10. OpenStack Cinder 卷

OpenStack Block Storage 通过 **cinder** 服务提供持久的块存储管理。块存储使得 OpenStack 用户能够创建可以连接到不同 OpenStack 实例的卷。

22.2.3.10.1. Docker 卷

master 和节点实例包含用于存储 **docker** 镜像的卷。卷的目的是确保大型镜像或容器不破坏现有节点的节点性能或能力。



注意

运行容器至少需要 15GB 的 docker 卷。这可能需要根据每个节点运行的大小和容器数量进行调整。

docker 卷由 OpenShift Ansible 安装程序通过变量 **openshift_openstack_docker_volume_size** 创建。稍后介绍了有关 OpenShift Ansible 安装程序所需更改的更多信息。

22.2.3.10.2. registry 卷

OpenShift 镜像注册表需要一个 **cinder** 卷，以确保在 registry 需要迁移到另一个节点时保存镜像。以下步骤演示了如何通过 OpenStack 创建镜像 registry。创建卷后，卷 ID 将包含在 OpenShift Installer *OSEv3.yml* 文件中，该文件通过参数 **openshift_hosted_registry_storage_openstack_volumeID** 如下所述。

```
$ source /path/to/examplerc
$ openstack volume create --size <volume-size-in-GB> <registry-name>
```

注意

registry 卷大小应该至少有 30GB。

验证卷的创建。

```
$ openstack volume list
--------------------------------------+-----------------------------------------------------+
| ID                                   | Name          | Status    | Size | Attached to  |
+--------------------------------------+-----------------------------------------------------+
| d65209f0-9061-4cd8-8827-ae6e2253a18d | <registry-name>| available |   30 |              |
+--------------------------------------+-----------------------------------------------------+
```

22.2.3.11. 创建并配置部署实例

部署实例的角色是充当 OpenShift Container Platform 部署和管理的 utility 主机。

创建部署主机网络和路由器

在创建实例之前，必须创建一个内部网络和路由器来与部署主机通信。以下命令创建了该网络和路由器。

```
$ source path/to/examplerc

$ openstack network create <deployment-net-name>

$ openstack subnet create --network <deployment-net-name> \
  --subnet-range <subnet_range> \
  --dns-nameserver <dns-ip> \
  <deployment-subnet-name>

$ openstack router create <deployment-router-name>

$ openstack router set --external-gateway <public-net-name> <deployment-router-name>

$ openstack router add subnet <deployment-router-name> <deployment-subnet-name>
```

部署部署实例

创建网络和安全组时，部署该实例。

```
$ domain=<domain>
$ netid1=$(openstack network show <deployment-net-name> -f value -c id)
$ openstack server create \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-group <deployment-sg-name> \
    deployment.$domain
```

注意

默认情况下，如果 **m1.small** 类别不存在，则使用满足 1 vCPU 和 2GB RAM 要求的现有类别。

创建并将浮动 IP 添加到部署实例

创建部署实例后，必须创建一个浮动 IP，然后分配给实例。以下示例显示了示例。

```
$ source /path/to/examplerc
$ openstack floating ip create <public-network-name>
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2017-08-24T22:44:03Z                 |
| description         |                                      |
| fixed_ip_address    | None                                 |
| floating_ip_address | 10.20.120.150                        |
| floating_network_id | 084884f9-d9d2-477a-bae7-26dbb4ff1873 |
| headers             |                                      |
| id                  | 2bc06e39-1efb-453e-8642-39f910ac8fd1 |
```

```
| port_id             | None                                 |
| project_id          | ca304dfee9a04597b16d253efd0e2332     |
| project_id          | ca304dfee9a04597b16d253efd0e2332     |
| revision_number     | 1                                    |
| router_id           | None                                 |
| status              | DOWN                                 |
| updated_at          | 2017-08-24T22:44:03Z                 |
+---------------------+--------------------------------------+
```

在上面的输出中，**floating_ip_address** 字段显示创建了浮动 IP **10.20.120.150**。要将这个 IP 分配给部署实例，请运行以下命令：

```
$ source /path/to/examplerc
$ openstack server add floating ip <deployment-instance-name> <ip>
```

例如，如果实例 **deployment.example.com** 被分配 IP **10.20.120.150**，则命令将为：

```
$ source /path/to/examplerc
$ openstack server add floating ip deployment.example.com 10.20.120.150
```

在部署主机中添加 RC 文件

部署主机存在后，通过 **scp** 将之前创建的 RC 文件复制到部署主机上，如下所示

```
scp <rc-file-deployment-host> cloud-user@<ip>:/home/cloud-user/
```

22.2.3.12. OpenShift Container Platform 的部署配置

以下小节描述了正确配置部署实例所需的所有步骤。

配置 ~/.ssh/config 以使用 Deployment Host 作为 Jumphost

要轻松连接到 OpenShift Container Platform 环境，请按照以下步骤操作。

在 OpenStack director 节点上，或使用私钥 <keypair-name>.pem 的本地工作站：

```
$ exec ssh-agent bash

$ ssh-add /path/to/<keypair-name>.pem
Identity added: /path/to/<keypair-name>.pem (/path/to/<keypair-name>.pem)
```

添加到 ~/**.ssh**/**config** 文件中：

```
Host deployment
   HostName       <deployment_fqdn_hostname OR IP address>
   User           cloud-user
   IdentityFile   /path/to/<keypair-name>.pem
   ForwardAgent   yes
```

使用带有 **-A** 选项的 **ssh** 连接到部署主机，这可以启用转发身份验证代理连接。

确保只针对 *~/.ssh/config* 文件所有者的读取写入权限：

```
$ chmod 600 ~/.ssh/config
```

```
$ ssh -A cloud-user@deployment
```

登录部署主机后，请通过检查 **SSH_AUTH_SOCK** 来验证 ssh 代理转发是否正常工作

```
$ echo "$SSH_AUTH_SOCK"
/tmp/ssh-NDFDQD02qB/agent.1387
```

订阅管理器并启用 OpenShift Container Platform 软件仓库

在部署实例中，在 Red Hat Subscription Manager 中注册。这可以通过使用凭证来实现：

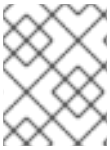
```
$ sudo subscription-manager register --username <user> --password '<password>'
```

另外，您可以使用激活码：

```
$ sudo subscription-manager register --org="<org_id>" --activationkey=<keyname>
```

注册后，启用以下软件仓库，如下所示：

```
$ sudo subscription-manager repos \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ose-3.11-rpms" \
    --enable="rhel-7-server-ansible-2.6-rpms" \
    --enable="rhel-7-server-openstack-13-rpms" \
    --enable="rhel-7-server-openstack-13-tools-rpms"
```



注意

请参阅设置存储库，以确认要启用的 OpenShift Container Platform 存储库和 Ansible 版本。以上文件只是一个示例。

部署主机上所需的软件包

部署主机上需要安装以下软件包。

安装以下软件包：

- **openshift-ansible**
- **python-openstackclient**
- **python2-heatclient**
- **python2-octaviaclient**
- **python2-shade**
- **python-dns**
- **git**

- **ansible**

```
$ sudo yum -y install openshift-ansible python-openstackclient python2-heatclient python2-
octaviaclient python2-shade python-dns git ansible
```

配置 Ansible

ansible 安装在部署实例上，以执行注册、安装软件包，并在 master 和节点实例上部署 OpenShift Container Platform 环境。

在运行 playbook 前，务必要创建一个 ***ansible.cfg*** 文件来反映您要部署的环境：

```
$ cat ~/ansible.cfg

[defaults]
forks = 20
host_key_checking = False
remote_user = openshift
gathering = smart
fact_caching = jsonfile
fact_caching_connection = $HOME/ansible/facts
fact_caching_timeout = 600
log_path = $HOME/ansible.log
nocows = 1
callback_whitelist = profile_tasks
inventory = /usr/share/ansible/openshift-ansible/playbooks/openstack/inventory.py,/home/cloud-
user/inventory

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=600s -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=false
control_path = %(directory)s/%%h-%%r
pipelining = True
timeout = 10

[persistent_connection]
connect_timeout = 30
connect_retries = 30
connect_interval = 1
```

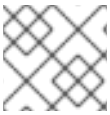
警告

以下参数值对于 ***ansible.cfg*** 文件很重要。

- **remote_user** 必须保留为用户 *openshift*。
- inventory 参数确保两个清单之间没有空格。

示例： *inventory = path/to/inventory1,path/to/inventory2*

以上代码块可覆盖文件中的默认值。确保使用复制到部署实例的密钥对填充 <keypair-name>。



注意

inventory 目录在 第 22.3.1 节 “为置备准备清单” 中创建。

OpenShift Authentication

OpenShift Container Platform 提供了使用许多不同的身份验证平台。如需身份验证选项列表，请参阅 配置身份验证和用户代理。

配置默认身份提供程序非常重要，因为默认配置是 Deny All。

22.3. 使用 OPENSHIFT ANSIBLE PLAYBOOK 置备 OPENSHIFT CONTAINER PLATFORM 实例

完成部署配置部署主机后，我们将使用 Ansible 为部署 OpenShift Container Platform 准备环境。在以下小节中，配置了 Ansible，并修改某些 YAML 文件，以实现 OpenStack 部署成功的 OpenShift Container Platform。

22.3.1. 为置备准备清单

通过前面的步骤安装 **openshift-ansible** 软件包后，有一个 **sample-inventory** 目录，我们将复制到部署主机的 **cloud-user** 主目录。

在部署主机上，

```
$ cp -r /usr/share/ansible/openshift-ansible/playbooks/openstack/sample-inventory/ ~/inventory
```

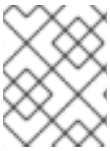
在这个清单目录中，*all.yml* 文件包含所有必须设置的不同参数，才能成功置备 RHOCP 实例。*OSEv3.yml* 文件包含 *all.yml* 文件和可自定义的所有可用 OpenShift Container Platform 集群参数的一些引用。

22.3.1.1. OpenShiftSDN 所有 YAML 文件

all.yml 文件有许多选项，可以修改以符合您的特定需求。此文件中收集的信息适用于成功部署 OpenShift Container Platform 所需的实例的调配部分。务必要仔细审阅。本文档将提供所有 YAML 文件的精简版本，并专注于为成功部署而需要设置的最关键参数。

```
$ cat ~/inventory/group_vars/all.yml
---
openshift_openstack_clusterid: "openshift"
openshift_openstack_public_dns_domain: *"example.com"*
openshift_openstack_dns_nameservers: *["10.19.115.228"]*
openshift_openstack_public_hostname_suffix: "-public"
openshift_openstack_nsupdate_zone: "{{ openshift_openstack_public_dns_domain }}"

openshift_openstack_keypair_name: *"openshift"*
openshift_openstack_external_network_name: *"public"*

openshift_openstack_default_image_name: *"rhel75"*

## Optional (Recommended) - This removes the need for floating IPs
## on the OpenShift Cluster nodes
```

```
openshift_openstack_node_subnet_name: *<deployment-subnet-name>*
openshift_openstack_router_name: *<deployment-router-name>*
openshift_openstack_master_floating_ip: *false*
openshift_openstack_infra_floating_ip: *false*
openshift_openstack_compute_floating_ip: *false*
## End of Optional Floating IP section

openshift_openstack_num_masters: *3*
openshift_openstack_num_infra: *3*
openshift_openstack_num_cns: *0*
openshift_openstack_num_nodes: *2*

openshift_openstack_master_flavor: *"m1.master"*
openshift_openstack_default_flavor: *"m1.node"*

openshift_openstack_use_lbaas_load_balancer: *true*

openshift_openstack_docker_volume_size: "15"

# # Roll-your-own DNS
*openshift_openstack_external_nsupdate_keys:*
  public:
    *key_secret: '/alb8h0EAFWvb4i+CMA12w=='*
    *key_name: "update-key"*
    *key_algorithm: 'hmac-md5'*
    *server: '<ip-of-DNS>'*
  private:
    *key_secret: '/alb8h0EAFWvb4i+CMA12w=='*
    *key_name: "update-key"*
    *key_algorithm: 'hmac-md5'*
    *server: '<ip-of-DNS>'*

ansible_user: openshift

## cloud config
openshift_openstack_disable_root: true
openshift_openstack_user: openshift
```



注意

由于使用外部 DNS 服务器，私有和公共部分使用 DNS 服务器的公共 IP 地址，因为 DNS 服务器不驻留在 OpenStack 环境中。

以上以星号(*)括起的值需要根据您的 OpenStack 环境和 DNS 服务器进行修改。

要正确修改 All YAML 文件的 DNS 部分，登录到 DNS 服务器并执行以下命令捕获密钥名称、密钥算法和密钥 secret：

```
$ ssh <ip-of-DNS>
$ sudo -i
# cat /etc/named/<key-name.key>
key "update-key" {
 algorithm hmac-md5;
 secret "/alb8h0EAFWvb4i+CMA02w==";
};
```

注意

密钥名称可能有所不同，上面的只是一个示例。

22.3.1.2. KuryrSDN 所有 YAML 文件

以下 *all.yml* 文件启用 Kuryr SDN 而不是默认的 OpenShiftSDN。请注意，以下示例是密度的版本，务必要仔细检查默认模板。

```
$ cat ~/inventory/group_vars/all.yml
---
openshift_openstack_clusterid: "openshift"
openshift_openstack_public_dns_domain: *"example.com"*
openshift_openstack_dns_nameservers: *["10.19.115.228"]*
openshift_openstack_public_hostname_suffix: "-public"
openshift_openstack_nsupdate_zone: "{{ openshift_openstack_public_dns_domain }}"

openshift_openstack_keypair_name: *"openshift"*
openshift_openstack_external_network_name: *"public"*

openshift_openstack_default_image_name: *"rhel75"*

## Optional (Recommended) - This removes the need for floating IPs
## on the OpenShift Cluster nodes
openshift_openstack_node_subnet_name: *<deployment-subnet-name>*
openshift_openstack_router_name: *<deployment-router-name>*
openshift_openstack_master_floating_ip: *false*
openshift_openstack_infra_floating_ip: *false*
openshift_openstack_compute_floating_ip: *false*
## End of Optional Floating IP section

openshift_openstack_num_masters: *3*
openshift_openstack_num_infra: *3*
openshift_openstack_num_cns: *0*
openshift_openstack_num_nodes: *2*

openshift_openstack_master_flavor: *"m1.master"*
openshift_openstack_default_flavor: *"m1.node"*

## Kuryr configuration
openshift_use_kuryr: True
openshift_use_openshift_sdn: False
use_trunk_ports: True
os_sdn_network_plugin_name: cni
openshift_node_proxy_mode: userspace
kuryr_openstack_pool_driver: nested
openshift_kuryr_precreate_subports: 5

kuryr_openstack_public_net_id: *<public_ID>*

# To disable namespace isolation, comment out the next 2 lines
openshift_kuryr_subnet_driver: namespace
openshift_kuryr_sg_driver: namespace
# If you enable namespace isolation, `default` and `openshift-monitoring` become the
# global namespaces. Global namespaces can access all namespaces. All
# namespaces can access global namespaces.
```

```
# To make other namespaces global, include them here:
kuryr_openstack_global_namespaces: default,openshift-monitoring

# If OpenStack cloud endpoints are accessible over HTTPS, provide the CA certificate
kuryr_openstack_ca: *<path-to-ca-certificate>*

openshift_master_open_ports:
- service: dns tcp
  port: 53/tcp
- service: dns udp
  port: 53/udp
openshift_node_open_ports:
- service: dns tcp
  port: 53/tcp
- service: dns udp
  port: 53/udp

# To set the pod network CIDR range, uncomment the following property and set its value:
#
# openshift_openstack_kuryr_pod_subnet_prefixlen: 24
#
# The subnet prefix length value must be smaller than the CIDR value that is
# set in the inventory file as openshift_openstack_kuryr_pod_subnet_cidr.
# By default, this value is /24.

# openshift_portal_net is the range that OpenShift services and their associated Octavia
# load balancer VIPs use. Amphora VMs use Neutron ports in the range that is defined by
# openshift_openstack_kuryr_service_pool_start and openshift_openstack_kuryr_service_pool_end.
#
# The value of openshift_portal_net in the OSEv3.yml file must be within the range that is
# defined by openshift_openstack_kuryr_service_subnet_cidr. This range must be half
# of openshift_openstack_kuryr_service_subnet_cidr's range. This practice ensures that
# openshift_portal_net does not overlap with the range that load balancers' VMs use, which is
# defined by openshift_openstack_kuryr_service_pool_start and
openshift_openstack_kuryr_service_pool_end.
#
# For reference only, copy the value in the next line from OSEv3.yml:
# openshift_portal_net: *"172.30.0.0/16"*

openshift_openstack_kuryr_service_subnet_cidr: *"172.30.0.0/15"*
openshift_openstack_kuryr_service_pool_start: *"172.31.0.1"*
openshift_openstack_kuryr_service_pool_end: *"172.31.255.253"*

# End of Kuryr configuration

openshift_openstack_use_lbaas_load_balancer: *true*

openshift_openstack_docker_volume_size: "15"

# # Roll-your-own DNS
*openshift_openstack_external_nsupdate_keys:*
  public:
    *key_secret: '/alb8h0EAFWvb4i+CMA12w==*'
    *key_name: "update-key"*
    *key_algorithm: 'hmac-md5'*
    *server: '<ip-of-DNS>'*
```

```
  private:
    *key_secret: '/alb8h0EAFWvb4i+CMA12w==*
    *key_name: "update-key"*
    *key_algorithm: 'hmac-md5'*
    *server: '<ip-of-DNS>'*

ansible_user: openshift

## cloud config
openshift_openstack_disable_root: true
openshift_openstack_user: openshift
```

注意

如果使用命名空间隔离，Kuryr-controller 会为每个命名空间创建一个新的 Neutron 网络和子网。

注意

启用 Kuryr SDN 时不支持网络策略和节点端口服务。

注意

如果启用了 Kuryr，OpenShift Container Platform 服务会通过 OpenStack Octavia Amphora 虚拟机实现。

Octavia 不支持 UDP 负载均衡。不支持公开 UDP 端口的服务。

22.3.1.2.1. 配置全局命名空间访问权限

kuryr_openstack_global_namespace 参数包含定义全局命名空间的列表。默认情况下，只有 **默认** 和 **openshift-monitoring** 命名空间包含在此列表中。

如果您要从 OpenShift Container Platform 3.11 的以前的 z-release 升级，请注意，从全局命名空间中对其他命名空间的访问由安全组 ***-allow_from_default** 控制。

虽然 **remote_group_id 规则** 可以从全局命名空间中控制对其他命名空间的访问，但使用该脚本可能会导致缩放和连接问题。要避免这些问题，请从在 ***_allow_from_default** 到 **remote_ip_prefix** 处使用 **remote_group_id**:

1. 在命令行中检索您的网络 **subnetCIDR** 值：

```
$ oc get kuryrnets ns-default -o yaml | grep subnetCIDR
  subnetCIDR: 10.11.13.0/24
```

2. 为这个范围创建 TCP 和 UDP 规则：

```
$ openstack security group rule create --remote-ip 10.11.13.0/24 --protocol tcp openshift-ansible-openshift.example.com-allow_from_default
$ openstack security group rule create --remote-ip 10.11.13.0/24 --protocol udp openshift-ansible-openshift.example.com-allow_from_default
```

3. 删除使用 **remote_group_id** 的安全组规则：

```
$ openstack security group show *-allow_from_default | grep remote_group_id
$ openstack security group rule delete REMOTE_GROUP_ID
```

表 22.4. All YAML 文件中的变量描述

变量	描述
openshift_openstack_clusterid	集群识别名称
openshift_openstack_public_dns_domain	公共 DNS 域名
openshift_openstack_dns_nameservers	DNS 名称服务器的 IP
openshift_openstack_public_hostname_suffix	在公共和私有的 DNS 记录中的节点主机名中添加后缀
openshift_openstack_nsupdate_zone	要使用 OCP 实例 IP 更新的区域
openshift_openstack_keypair_name	用于登录 OCP 实例的密钥对名称
openshift_openstack_external_network_name	OpenStack 公共网络名称
openshift_openstack_default_image_name	用于 OCP 实例的 OpenStack 镜像
openshift_openstack_num_masters	要部署的 master 节点数量
openshift_openstack_num_infra	要部署的基础架构节点数量
openshift_openstack_num_cns	要部署的容器原生存储节点数量
openshift_openstack_num_nodes	要部署的应用程序节点数量
openshift_openstack_master_flavor	用于 master 实例的 OpenStack 类别的名称
openshift_openstack_default_flavor	如果未指定的具体类别，则 Openstack 类别的名称用于所有实例。
openshift_openstack_use_lbaas_load_balancer	启用 Octavia 负载均衡器的布尔值（必须安装 Octavia）
openshift_openstack_docker_volume_size	Docker 卷的最小大小（必需变量）
openshift_openstack_external_nsupdate_keys	使用实例 IP 地址更新 DNS
ansible_user	用于部署 OpenShift Container Platform 的 Ansible 用户。"openshift"是需要的名称，且不得更改。
openshift_openstack_disable_root	禁用 root 访问权限的布尔值

变量	描述
openshift_openstack_user	使用此用户创建的 OCP 实例
openshift_openstack_node_subnet_name	用于部署的现有 OpenShift 子网的名称。这应该与用于您的部署主机的子网名称相同。
openshift_openstack_router_name	用于部署的现有 OpenShift 路由器的名称。这应该与用于部署主机的路由器名称相同。
openshift_openstack_master_floating_ip	默认为 **true**。如果您不希望分配给 master 节点的浮动 IP，则必须设置为 **false**。
openshift_openstack_infra_floating_ip	默认为 **true**。如果您不希望浮动 IP 分配给基础架构节点，则必须设置为 **false**。
openshift_openstack_compute_floating_ip	默认为 **true**。如果您不希望分配给计算节点的浮动 IP，则必须设置为 **false**。
openshift_use_openshift_sdn	如果要禁用 openshift-sdn，则必须设置为 **false**
openshift_use_kuryr	如果要启用 kuryr sdn，则必须设置为 **true**
use_trunk_ports	必须设置为 **true**，才能创建带有中继端口的 OpenStack 虚拟机（kuryr 必需）
os_sdn_network_plugin_name	选择 SDN 行为。为 kuryr 设置为 **cni**
openshift_node_proxy_mode	必须将 Kuryr 设置为 **用户空间**
openshift_master_open_ports	使用 Kuryr 时会在虚拟机上打开的端口
kuryr_openstack_public_net_id	由 Kuryr 的需求。从中获取 FIP 的公共 OpenStack 网络的 ID
openshift_kuryr_subnet_driver	Kuryr Subnet 驱动程序。必须是 **命名空间**，用于为每个命名空间创建一个子网
openshift_kuryr_sg_driver	Kuryr 安全组驱动程序。必须是 **命名空间** 才能隔离命名空间
kuryr_openstack_global_namespaces	用于命名空间隔离的全局命名空间。默认值为 **default**, **openshift-monitoring**。
kuryr_openstack_ca	到云的 CA 证书的路径。如果 OpenStack 云端点可以通过 HTTPS 访问，则需要此项。

22.3.1.3. OSEv3 YAML 文件

OSEv3 YAML 文件指定与 OpenShift 安装相关的所有不同参数和自定义。

以下是文件的一个精简版本，其中包含成功部署所需的所有变量。根据特定 OpenShift Container Platform 部署所需的自定义，可能需要额外的变量。

```
$ cat ~/inventory/group_vars/OSEv3.yml
---

openshift_deployment_type: openshift-enterprise
openshift_release: v3.11
oreg_url: registry.access.redhat.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams: true
oreg_auth_user: <oreg_auth_user>
oreg_auth_password: <oreg_auth_pw>
# The following is required if you want to deploy the Operator Lifecycle Manager (OLM)
openshift_additional_registry_credentials:
[{'host':'registry.connect.redhat.com','user':'REGISTRYCONNECTUSER','password':'REGISTRYCONN
ECTPASSWORD','test_image':'mongodb/enterprise-operator:0.3.2'}]

openshift_master_default_subdomain: "apps.{{ (openshift_openstack_clusterid|trim == '') |
ternary(openshift_openstack_public_dns_domain, openshift_openstack_clusterid + '.' +
openshift_openstack_public_dns_domain) }}"

openshift_master_cluster_public_hostname: "console.{{ (openshift_openstack_clusterid|trim == '') |
ternary(openshift_openstack_public_dns_domain, openshift_openstack_clusterid + '.' +
openshift_openstack_public_dns_domain) }}"

#OpenStack Credentials:
openshift_cloudprovider_kind: openstack
openshift_cloudprovider_openstack_auth_url: "{{ lookup('env','OS_AUTH_URL') }}"
openshift_cloudprovider_openstack_username: "{{ lookup('env','OS_USERNAME') }}"
openshift_cloudprovider_openstack_password: "{{ lookup('env','OS_PASSWORD') }}"
openshift_cloudprovider_openstack_tenant_name: "{{ lookup('env','OS_PROJECT_NAME') }}"
openshift_cloudprovider_openstack_blockstorage_version: v2
openshift_cloudprovider_openstack_domain_name: "{{ lookup('env','OS_USER_DOMAIN_NAME') }}"
openshift_cloudprovider_openstack_conf_file: <path_to_local_openstack_configuration_file>

#Use Cinder volume for Openshift registry:
openshift_hosted_registry_storage_kind: openstack
openshift_hosted_registry_storage_access_modes: ['ReadWriteOnce']
openshift_hosted_registry_storage_openstack_filesystem: xfs
openshift_hosted_registry_storage_volume_size: 30Gi


openshift_hosted_registry_storage_openstack_volumeID: d65209f0-9061-4cd8-8827-ae6e2253a18d
openshift_hostname_check: false
ansible_become: true

#Setting SDN (defaults to ovs-networkpolicy) not part of OSEv3.yml
#For more info, on which to choose, visit:
#https://docs.openshift.com/container-platform/3.11/architecture/networking/sdn.html#overview
networkPluginName: redhat/ovs-networkpolicy
#networkPluginName: redhat/ovs-multitenant
```

```
#Configuring identity providers with Ansible
#For initial cluster installations, the Deny All identity provider is configured
#by default. It is recommended to be configured with either htpasswd
#authentication, LDAP authentication, or Allowing all authentication (not recommended)
#For more info, visit:
#https://docs.openshift.com/container-
platform/3.10/install_config/configuring_authentication.html#identity-providers-ansible
#Example of Allowing All
#openshift_master_identity_providers: [{'name': 'allow_all', 'login': 'true', 'challenge': 'true', 'kind':
'AllowAllPasswordIdentityProvider'}]


#Optional Metrics (uncomment below lines for installation)

#openshift_metrics_install_metrics: true
#openshift_metrics_cassandra_storage_type: dynamic
#openshift_metrics_storage_volume_size: 25Gi
#openshift_metrics_cassandra_nodeselector: {"node-role.kubernetes.io/infra":"true"}
#openshift_metrics_hawkular_nodeselector: {"node-role.kubernetes.io/infra":"true"}
#openshift_metrics_heapster_nodeselector: {"node-role.kubernetes.io/infra":"true"}

#Optional Aggregated Logging (uncomment below lines for installation)

#openshift_logging_install_logging: true
#openshift_logging_es_pvc_dynamic: true
#openshift_logging_es_pvc_size: 30Gi
#openshift_logging_es_cluster_size: 3
#openshift_logging_es_number_of_replicas: 1
#openshift_logging_es_nodeselector: {"node-role.kubernetes.io/infra":"true"}
#openshift_logging_kibana_nodeselector: {"node-role.kubernetes.io/infra":"true"}
#openshift_logging_curator_nodeselector: {"node-role.kubernetes.io/infra":"true"}
```

有关列出的任何变量的更多详情，请参阅 OpenShift-Ansible 主机清单示例。

22.3.2. OpenStack 先决条件 Playbook

OpenShift Container Platform Ansible 安装程序提供了一个 playbook，以确保满足 OpenStack 实例的所有置备步骤。

在运行 playbook 之前，请确保 source RC 文件

```
$ source path/to/examplerc
```

通过部署主机上的 **ansible-playbook** 命令，使用 **prerequisites.yml** playbook 确保满足所有先决条件：

```
$  ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openstack/openshift-
cluster/prerequisites.yml
```

当先决条件 playbook 成功完成后，运行置备 playbook，如下所示：

```
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openstack/openshift-
cluster/provision.yml
```

重要

如果 *provision.yml* prematurely 错误，请检查 OpenStack 堆栈的状态并等待它完成

```
$ watch openstack stack list
+--------------------------------------+------------------+--------------------+----------------------
+-------------+
| ID                                   | Stack Name       | Stack Status       | Creation Time        |
Updated Time |
+--------------------------------------+------------------+--------------------+----------------------
+-------------+
| 87cb6d1c-8516-40fc-892b-49ad5cb87fac | openshift-cluster |
CREATE_IN_PROGRESS | 2018-08-20T23:44:46Z | None         |
+--------------------------------------+------------------+--------------------+----------------------
+-------------+
```

如果堆栈显示 **CREATE_IN_PROGRESS**，请等待堆栈以最终结果（如 **CREATE_COMPLETE**）完成。如果堆栈成功完成，请重新运行 *provision.yml* playbook，以完成所有其他必要的步骤。

如果堆栈显示 **CREATE_FAILED**，请确保运行以下命令来查看导致错误的原因：

```
$ openstack stack failures list openshift-cluster
```

22.3.3. 堆栈名称配置

默认情况下，OpenStack 为 OpenShift Container Platform 集群创建的 Heat 堆栈名为 **openshift-cluster**。如果要使用其他名称，必须在运行 playbook 前设置 **OPENSHIFT_CLUSTER** 环境变量：

```
$ export OPENSHIFT_CLUSTER=openshift.example.com
```

如果使用非默认堆栈名称并运行 openshift-ansible playbook 来更新部署，您必须将 **OPENSHIFT_CLUSTER** 设置为您的堆栈名称以避免错误。

22.4. 使用 OPENSHIFT CONTAINER PLATFORM 实例注册 SUBSCRIPTION MANAGER

成功置备节点后，下一步是确保通过 **subscription-manager** 成功注册所有节点，以安装成功 OpenShift Container Platform 安装的所有必要软件包。为简单起见，已创建了一个并提供了 ***repos.yml*** 文件。

```
$ cat ~/repos.yml
---
- name: Enable the proper repositories for OpenShift installation
  hosts: OSEv3
  become: yes
  tasks:
  - name: Register with activationkey and consume subscriptions matching Red Hat Cloud Suite or
Red Hat OpenShift Container Platform
    redhat_subscription:
      state: present
      activationkey: <key-name>
      org_id: <orig_id>
      pool: '^(Red Hat Cloud Suite|Red Hat OpenShift Container Platform)$'
```

```
- name: Disable all current repositories
  rhsm_repository:
    name: '*'
    state: disabled

- name: Enable Repositories
  rhsm_repository:
    name: "{{ item }}"
    state: enabled
  with_items:
    - rhel-7-server-rpms
    - rhel-7-server-extras-rpms
    - rhel-7-server-ansible-2.6-rpms
    - rhel-7-server-ose-3.11-rpms
```

注意

请参阅 Set Up Repositories 来确认要启用的软件仓库和版本。以上文件只是一个示例。

使用 *repos.yml*，运行 **ansible-playbook** 命令：

```
$ ansible-playbook repos.yml
```

上例使用 Ansible 的 **redhat_subscription** 和 **rhsm_repository** 模块来进行所有注册，禁用和启用存储库。本特定示例使用了使用红帽激活码。如果您没有激活码，请确保访问 Ansible **redhat_subscription** 模块以使用用户名和密码进行修改，如下例所示：
https://docs.ansible.com/ansible/2.6/modules/redhat_subscription_module.html

注意

有时候，**redhat_subscription** 模块可能会在某些节点上失败。如果出现这个问题，请使用 **subscription-manager** 手动注册 OpenShift Container Platform 实例。

22.5. 使用 ANSIBLE PLAYBOOK 安装 OPENSHIFT CONTAINER PLATFORM

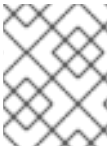
在调配的 OpenStack 实例后，将重点转移到安装 OpenShift Container Platform。安装和配置通过 OpenShift RPM 软件包提供的一系列 Ansible playbook 和角色来进行。查看之前配置好的 OSEv3.yml 文件，以确保所有选项都已正确设置。

在运行安装程序 playbook 之前，请确保通过以下方法满足所有 {rhocp} 先决条件：

```
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
```

运行安装程序 playbook 以安装 Red Hat OpenShift Container Platform：

```
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openstack/openshift-
cluster/install.yml
```



注意

OpenShift Container Platform 版本 3.11 支持 RH OSP 14 和 RH OSP 13。OpenShift Container Platform 版本 3.10 需要运行在 RH OSP 13 上。

22.6. 将配置更改应用到现有 OPENSHIFT CONTAINER PLATFORM 环境

在所有 master 和节点主机上启动或重启 OpenShift Container Platform 服务以应用您的配置更改，请参阅重启 OpenShift Container Platform 服务 ：

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

注意

Kubernetes 架构需要来自云提供商的可靠端点。当云提供商停机时，kubelet 会防止 OpenShift Container Platform 重启。如果底层云供应商端点不可靠，请不要安装使用云供应商集成的集群。如在裸机环境中一样安装集群。不建议在已安装的集群中打开或关闭云提供商集成。但是，如果该情境不可避免，请完成以下过程。

从不使用云供应商切换到使用云提供商会产生错误消息。添加云供应商会尝试删除节点，因为从其切换的节点使用 **hostname** 作为 **externalID** （当没有云供应商使用时）使用云供应商的 **instance-id** （由云提供商指定）。要解决这个问题：

1. 以集群管理员身份登录到 CLI。
2. 检查和备份现有节点标签：

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?=Taints)'
```

3. 删除节点：

```
$ oc delete node <node_name>
```

4. 在每个节点主机上，重启 OpenShift Container Platform 服务。

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

5. 在每个主机上重新添加回您以前具有的标记。

22.6.1. 在现有 OpenShift 环境中配置 OpenStack 变量

要设置所需的 OpenStack 变量，请修改所有 OpenShift Container Platform 主机上以下内容的 */etc/origin/cloudprovider/openstack.conf* 文件，包括 master 和节点：

```
[Global]
auth-url = <OS_AUTH_URL>
username = <OS_USERNAME>
password = <password>
domain-id = <OS_USER_DOMAIN_ID>
tenant-id = <OS_TENANT_ID>
```

```
region = <OS_REGION_NAME>

[LoadBalancer]
subnet-id = <UUID of the load balancer subnet>
```

请参考您的 OpenStack 管理员获取 **OS_** 变量的值，它们通常用于 OpenStack 配置。

22.6.2. 为动态创建的 OpenStack PV 配置区域标签

管理员可以为动态创建的 OpenStack PV 配置区域标签。如果 OpenStack Cinder 区域名称与计算区域名称不匹配，则此选项很有用，例如，如果只有一个 Cinder 区域，并且有多个计算区域。管理员可以动态创建 Cinder 卷，然后检查标签。

查看 PV 的区标签：

```
# oc get pv --show-labels
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM              STORAGECLASS   REASON   AGE       LABELS
pvc-1faa6f93-64ac-11e8-930c-fa163e3c373c   1Gi        RWO            Delete           Bound     openshift-
node/pvc1   standard                12s       failure-domain.beta.kubernetes.io/zone=nova
```

默认设置是 enabled。使用 **oc get pv --show-labels** 命令返回 **failure-domain.beta.kubernetes.io/zone=nova** 标签。

要禁用区标签，请通过添加以下内容更新 ***openstack.conf*** 文件：

```
[BlockStorage]
ignore-volume-az = yes
```

重启 master 服务后创建的 PV 将不会具有 zone 标签。

第 23 章 为 GOOGLE COMPUTE ENGINE 配置

您可以配置 OpenShift Container Platform 以访问现有的 Google Compute Engine(GCE)基础架构，包括使用 GCE 卷作为应用程序数据的持久性存储。

23.1. 开始前

23.1.1. 为 Google Cloud Platform 配置授权

角色

为 OpenShift Container Platform 配置 GCP 需要以下 GCP 角色：

roles/owner	创建服务帐户、云存储、实例、镜像、模板、云 DNS 条目以及部署负载均衡器和健康检查所需要的。

如果用户应在测试阶段重新部署环境，则需要**删除**权限。

您还可以创建服务帐户以避免在部署 GCP 对象时使用个人用户。

如需更多信息，请参阅 GCP 文档中的了解角色部分，包括如何配置角色的步骤。

范围和服务帐户

GCP 使用范围来确定经过身份验证的身份是否有权在资源内执行操作。例如，如果具有只读范围访问令牌的应用程序 A 只能读取，而具有读写范围访问令牌的应用程序 B 可能会读取和修改数据。

范围在 GCP API 级别中定义为 **https://www.googleapis.com/auth/compute.readonly**。

您可以使用 **--scopes=[SCOPE,... 指定范围。]** 选项在创建实例时，或者您可以使用 **--no-scopes** 选项来创建实例，如果没有考虑实例访问 GCP API，则可以使用 --no-scopes 选项来创建实例。

如需更多信息，请参阅 GCP 文档中的 Scopes 部分。

所有 GCP 项目都包括一个默认的 **[PROJECT_NUMBER]-compute@developer.gserviceaccount.com** 服务帐户，具有项目编辑器权限。

默认情况下，新创建的实例会自动启用为具有以下访问范围的默认服务帐户运行：

- https://www.googleapis.com/auth/devstorage.read_only
- https://www.googleapis.com/auth/logging.write
- https://www.googleapis.com/auth/monitoring.write
- https://www.googleapis.com/auth/pubsub
- https://www.googleapis.com/auth/service.management.readonly
- https://www.googleapis.com/auth/servicecontrol
- https://www.googleapis.com/auth/trace.append
- https://www.googleapis.com/auth/bigquery

- https://www.googleapis.com/auth/cloud-platform
- https://www.googleapis.com/auth/compute.readonly
- https://www.googleapis.com/auth/compute
- https://www.googleapis.com/auth/datastore
- https://www.googleapis.com/auth/logging.write
- https://www.googleapis.com/auth/monitoring
- https://www.googleapis.com/auth/monitoring.write
- https://www.googleapis.com/auth/servicecontrol
- https://www.googleapis.com/auth/service.management.readonly
- https://www.googleapis.com/auth/sqlservice.admin
- https://www.googleapis.com/auth/devstorage.full_control
- https://www.googleapis.com/auth/devstorage.read_only
- https://www.googleapis.com/auth/devstorage.read_write
- https://www.googleapis.com/auth/taskqueue
- https://www.googleapis.com/auth/userinfo.email

在创建实例时，您可以使用 **--service-account=SERVICE_ACCOUNT** 选项指定另一个服务帐户，或使用 **gcloud** CLI 明确禁用实例的 **服务帐户**。

如需更多信息，请参阅 GCP 文档中的创建新服务帐户部分。

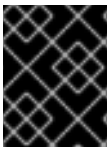
23.1.2. Google Compute Engine 对象

将 OpenShift Container Platform 与 Google Compute Engine(GCE)集成需要以下组件或服务。

GCP 项目

GCP 项目是组织基本级别的实体，它组成了创建、启用和使用所有 GCP 服务的基础。这包括管理 API、启用账单、添加和删除协作器以及管理权限。

如需更多信息，请参阅 GCP 文档中的项目资源部分。



重要

项目 ID 是唯一标识符，项目 ID 必须在所有 Google Cloud Engine 之间唯一。这意味着，如果其他人使用该 ID 创建了一个项目 ID，则您无法使用 **myproject** 作为项目 ID。

账单

除非将账单附加到帐户，否则您无法创建新的资源。新项目可以链接到现有项目，也可以输入新的信息。

如需更多信息，请参阅 GCP 文档中的创建、修改或关闭您的账单帐户。

云身份和访问管理

部署 OpenShift Container Platform 需要正确权限。用户必须能够创建服务帐户、云存储、实例、镜像、模板、云 DNS 条目，以及部署负载均衡器和健康检查。删除权限也会有帮助，以便在测试过程中重新部署环境。

您可以使用特定权限创建服务帐户，然后使用它们部署基础架构组件，而不是常规用户。您还可以创建角色来限制对不同用户或服务帐户的访问。

GCP 实例使用服务帐户来允许应用程序调用 GCP API。例如，OpenShift Container Platform 节点主机可以调用 GCP 磁盘 API，为应用程序提供持久性卷。

IAM 服务可使用对各种基础架构、服务资源和细粒度角色的访问控制。如需更多信息，请参阅 GCP 文档中的访问云概述 部分。

SSH 密钥

GCP 将 SSH 公钥作为授权密钥注入，以便您可以在创建的实例中使用 SSH 登录。您可以为每个实例或每个项目配置 SSH 密钥。

您可以使用现有的 SSH 密钥。GCP 元数据可帮助存储实例中引导时注入的 SSH 密钥，以允许 SSH 访问。

如需更多信息，请参阅 GCP 文档中的 Metadata 部分。

GCP 区域和区

GCP 有一个覆盖区域和可用区的全局基础架构。在不同区的 GCP 中部署 OpenShift Container Platform 有助于避免出现单点故障，但存储有一些注意事项。

GCP 磁盘在区内创建。因此，如果 OpenShift Container Platform 节点主机在区域 "A" 中停机，并且 pod 移到 zone "B"，则持久性存储无法附加到这些 pod，因为磁盘位于不同的区。

在安装 OpenShift Container Platform 前，部署多区 OpenShift Container Platform 环境是一个重要决定。如果部署多区环境，建议的设置是在单一区域中使用三个不同的区。

如需更多信息，请参阅 GCP 文档中有关 区域和 区的 Kubernetes 文档。

外部 IP 地址

因此，GCP 实例可以与互联网通信，您必须将外部 IP 地址附加到实例。另外，还需要外部 IP 地址与从 Virtual Private Cloud(VPC)网络外部署的实例进行通信。

警告

需要外部 **IP 地址** 进行互联网访问 是该提供程序的限制。如果需要，您可以将防火墙规则配置为阻止实例中进入的外部流量。

如需更多信息，请参阅外部 IP 地址上的 GCP 文档。

Cloud DNS

GCP Cloud DNS 是一个 DNS 服务，用于使用 GCP DNS 服务器将域名发布到全局 DNS。

公共云 DNS 区域需要一个通过 Google 的"Domains"服务或第三方供应商购买的域名。在创建区时，必须将 Google 提供的名称服务器添加到注册商中。

如需更多信息，请参阅 Cloud DNS 的 GCP 文档。

注意

GCP VPC 网络有一个内部 DNS 服务，可自动解析内部主机名。

实例的内部完全限定域名(FQDN)遵循 **[HOST_NAME].c.[PROJECT_ID].internal** 格式。

如需更多信息，请参阅内部 DNS 上的 GCP 文档。

负载平衡

GCP 负载均衡服务启用在 GCP 云中的多个实例间流量分布。

负载均衡有三种：

- 内部
- 网络负载均衡
- HTTP(S)负载均衡
- SSL 代理负载均衡
- TCP 代理负载均衡

注意

HTTPS 和 TCP 代理负载平衡是 master 节点使用 HTTPS 健康检查的唯一选项，用于检查 ***/healthz*** 的状态。

由于 HTTPS 负载均衡需要自定义证书，这种实现使用 TCP 代理负载平衡来简化过程。

如需更多信息，请参阅 GCP 文档。

实例大小

成功的 OpenShift Container Platform 环境需要一些最低硬件要求：

表 23.1. 实例大小

角色	大小
Master	**n1-standard-8**
节点	**n1-standard-4**

GCP 允许您创建自定义实例大小来满足不同的要求。如需更多信息，请参阅使用自定义 Machine Type 创建实例，或参阅 Machine type 和 OpenShift Container Platform Minimum 硬件要求 来获得有关实例大小的更多信息。

存储选项

默认情况下，每个 GCP 实例都有一个包含操作系统的小根磁盘。当实例中运行的应用程序需要更多存储空间时，您可以为实例添加额外的存储选项：

- 标准持久性磁盘
- SSD 持久性磁盘
- 本地 SSD
- 云存储存储桶

如需更多信息，请参阅有关存储选项的 GCP 文档。

23.2. 为 GCE 配置 OPENSHIFT CONTAINER PLATFORM

您可以通过两种方式为 GCE 配置 OpenShift Container Platform：

- 使用 Ansible
- 通过修改 *master-config.yaml* 文件手动进行

23.2.1. 选项 1：使用 Ansible 为 GCP 配置 OpenShift Container Platform

您可以在安装时或安装后修改 Ansible 清单文件，为 Google Compute Platform(GCP)配置 OpenShift Container Platform。

流程

1. 您至少必须定义 **openshift_cloudprovider_kind**、**openshift_gcp_project** 和 **openshift_gcp_prefix** 参数，以及对于 multizone 部署的可选 **openshift_gcp_multizone**，则为 **openshift_gcp_network_name** 定义。
在安装时将以下部分添加到 Ansible 清单文件中，以便为 GCP 配置 OpenShift Container Platform 环境：

```
[OSEv3:vars]
openshift_cloudprovider_kind=gce
openshift_gcp_project=<projectid> 1
openshift_gcp_prefix=<uid> 2
openshift_gcp_multizone=False 3
openshift_gcp_network_name=<network name> 4
```

1. 提供运行现有实例的 GCP 项目 ID。在 Google Cloud Platform 控制台中创建项目时，会生成此 ID。
2. 提供唯一字符串来标识每个 OpenShift Container Platform 集群。这在 GCP 之间必须是唯一的。
3. 另外，还可设置为 **True** 以在 GCP 上触发多 zone 部署。默认设置为 **False**。
4. 另外，如果没有使用**默认**网络，则提供网络名称。

使用 Ansible 安装也会创建并配置以下文件以适合您的 GCP 环境：

- */etc/origin/cloudprovider/gce.conf*
- */etc/origin/master/master-config.yaml*
- */etc/origin/node/node-config.yaml*

2. 如果您使用 GCP 运行负载均衡器服务，Compute Engine 虚拟机实例需要 **ocp** 后缀。例如，如果 **openshift_gcp_prefix** 参数的值设置为 **mycluster**，则必须标记带有 **myclusterocp** 的节点。有关如何在 Compute Engine 虚拟机实例中添加网络标签的更多信息，请参阅添加和删除网络标签。

3. 另外，您可以配置多区支持。
集群安装过程默认配置单区支持，但您可以配置多个区以避免出现单一故障点。

   因为 GCP 磁盘是在区中创建的，在不同的区上的 GCP 中部署 OpenShift Container Platform 可能会导致存储出现问题。如果 OpenShift Container Platform 节点主机在区域 "A" 中停机，并且 pod 移到 zone "B"，则持久性存储无法附加到这些 pod，因为磁盘现在位于不同的区。如需更多信息，请参阅 Kubernetes 文档中的多个区限制。

   要使用 Ansible 清单文件启用多区支持，请添加以下参数：

   ```
   [OSEv3:vars]
   openshift_gcp_multizone=true
   ```

   要返回单区支持，将 **openshift_gcp_multizone** 值设置为 **false**，然后重新运行 Ansible 清单文件。

23.2.2. 选项 2：为 GCE 手动配置 OpenShift Container Platform

23.2.2.1. 为 GCE 手动配置 master 主机

在所有 master 主机上执行以下步骤。

流程

1. 默认情况下，将 GCE 参数添加到 **/etc/origin/master/master-config.yaml** 中的 master 配置文件的 **apiServerArguments** 和 **controllerArguments** 部分：

   ```
   apiServerArguments:
     cloud-provider:
       - "gce"
     cloud-config:
       - "/etc/origin/cloudprovider/gce.conf"
   controllerArguments:
     cloud-provider:
       - "gce"
     cloud-config:
       - "/etc/origin/cloudprovider/gce.conf"
   ```

2. 使用 Ansible 为 GCP 配置 OpenShift Container Platform 时，会自动创建 */etc/origin/cloudprovider/gce.conf* 文件。由于您要手动为 GCP 配置 OpenShift Container Platform，所以您必须创建该文件并输入以下命令：

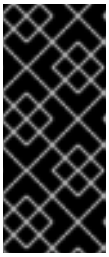
   ```
   [Global]
   ```

```
project-id = <project-id> 1
network-name = <network-name> 2
node-tags = <node-tags> 3
node-instance-prefix = <instance-prefix> 4
multizone = true 5
```

1. 提供运行现有实例的 GCP 项目 ID。
2. 如果没有使用默认值，则提供网络名称。
3. 为 GCP 节点提供标签。必须包含 **ocp** 作为后缀。例如，如果 **node-instance-prefix** 参数的值设置为 **mycluster**，则节点必须标记为 **myclusterocp**。
4. 提供唯一字符串来标识 OpenShift Container Platform 集群。
5. 设置为 **true** 以在 GCP 上触发多 zone 部署。默认设置为 **False**。

集群安装过程默认配置单区支持。

在不同区的 GCP 中部署 OpenShift Container Platform 有助于避免出现单一故障点，但可能会导致存储出现问题。这是因为 GCP 磁盘是在区中创建。如果 OpenShift Container Platform 节点主机在区域 "A" 中停机，并且 pod 应该移到 zone "B" 中，则持久性存储无法附加到这些 pod，因为磁盘现在位于不同的区中。如需更多信息，请参阅 Kubernetes 文档中的多个区限制。



重要

要使用 GCP 运行负载均衡器服务，Compute Engine 虚拟机节点实例需要 **ocp** 后缀：<**openshift_gcp_prefix>ocp**。例如，如果 **openshift_gcp_prefix** 参数的值设置为 **mycluster**，则必须标记带有 **myclusterocp** 的节点。有关如何在 Compute Engine 虚拟机实例中添加网络标签的更多信息，请参阅添加和删除网络标签。

3. 重启 OpenShift Container Platform 主机服务：

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

要返回单区支持，将 multizone 值设置为 **false**，然后重新启动 master 和节点主机服务。

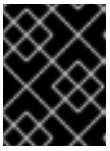
23.2.2.2. 为 GCE 手动配置节点主机

在所有节点主机上执行以下内容。

流程

1. 编辑 适当的节点配置映射 并更新 **kubeletArguments** 部分的内容：

```
kubeletArguments:
  cloud-provider:
    - "gce"
  cloud-config:
    - "/etc/origin/cloudprovider/gce.conf"
```



重要

nodeName 必须与 GCP 中的实例名称匹配，才能使云供应商集成正常工作。名称也必须与 RFC1123 兼容。

2. 重启所有节点上的 OpenShift Container Platform 服务。

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

23.2.3. 为 GCP 配置 OpenShift Container Platform registry

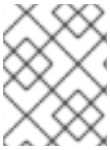
Google Cloud Platform(GCP)提供对象存储，OpenShift Container Platform 可以使用 OpenShift Container Platform 使用 OpenShift Container Platform 容器镜像 registry 存储容器镜像。

如需更多信息，请参阅 GCP 文档中的 Cloud Storage。

先决条件

您必须在安装前创建存储桶来托管 registry 镜像。以下命令使用配置的服务帐户创建区域存储桶：

```
gsutil mb -c regional -l <region> gs://ocp-registry-bucket
cat <<EOF > labels.json
{
  "ocp-cluster": "mycluster"
}
EOF
gsutil label set labels.json gs://ocp-registry-bucket
rm -f labels.json
```



注意

默认情况下，存储桶的数据会使用 Google 管理的密钥自动加密。要指定不同的密钥来加密数据，请参阅 GCP 中的数据加密选项。

如需更多信息，请参阅创建存储存储桶文档。

流程

配置 registry 的 Ansible 清单文件 以使用 Google Cloud Storage (GCS) 存储桶：

```
[OSEv3:vars]
# GCP Provider Configuration
openshift_hosted_registry_storage_provider=gcs
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_replicas=1 (1)
openshift_hosted_registry_storage_gcs_bucket=<bucket_name> (2)
openshift_hosted_registry_storage_gcs_keyfile=<bucket_keyfile> (3)
openshift_hosted_registry_storage_gcs_rootdirectory=<registry_directory> (4)
```

(1) 用于配置的副本数。

(2) registry 存储的存储桶名称。

3. 使用自定义密钥文件加密数据的安装程序主机上的路径。
4. 用于存储数据的目录。默认情况下 **/registry**

如需更多信息，请参阅 GCP 文档中的 Cloud Storage。

23.2.3.1. 为 GCP 手动配置 OpenShift Container Platform registry

要使用 GCP 对象存储，请编辑 registry 的配置文件并挂载到 registry pod。

有关存储驱动程序配置文件的更多信息，请参阅 Google Cloud Storage Driver 文档 。

流程

1. 导出当前的 */etc/registry/config.yml* 文件：

```
$ oc get secret registry-config \
    -o jsonpath='{.data.config\.yml}' -n default | base64 -d \
  >> config.yml.old
```

2. 从旧的 */etc/registry/config.yml* 文件创建新配置文件：

```
$ cp config.yml.old config.yml
```

3. 编辑该文件以包含 GCP 参数。在 registry 配置文件的 **storage** 部分指定存储桶和密钥文件：

```
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  gcs:
    bucket: ocp-registry 1
    keyfile: mykeyfile 2
```

1. 使用 GCP 存储桶名称替换。
2. 私有服务帐户密钥文件，采用 JSON 格式。如果使用 Google Application Default Credentials，请不要指定 **keyfile** 参数。

4. 删除 **registry-config** secret：

```
$ oc delete secret registry-config -n default
```

5. 重新创建 secret 来引用更新的配置文件：

```
$ oc create secret generic registry-config \
    --from-file=config.yml -n default
```

6. 重新部署 registry 以读取更新的配置：

```
$ oc rollout latest docker-registry -n default
```

23.2.3.1.1. 验证 registry 是否使用 GCP 对象存储

验证 registry 是否使用 GCP 存储桶存储：

流程

1. 使用 GCP 存储成功部署 registry 后，如果 registry 部署使用 **emptydir** 而不是 GCP 存储桶存储，registry **deploymentconfig** 不会显示任何信息：

```
$ oc describe dc docker-registry -n default
...
Mounts:
  ...
  /registry from registry-storage (rw)
Volumes:
registry-storage:
Type:       EmptyDir 1
...
```

1 共享 pod 生命周期的临时目录。

2. 检查 */registry* 挂载点是否为空。这是将使用卷 GCP 存储：

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}')  -i -t -- ls -l /registry
total 0
```

3. 如果为空，它会因为 GCP 存储桶配置在 **registry-config** secret 中执行：

```
$ oc describe secret registry-config
Name:         registry-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
config.yml:  398 bytes
```

4. 安装程序会使用扩展的 registry 功能创建带有所需配置的 *config.yml* 文件，如安装文档所述。要查看配置文件，包括存储存储桶配置的 **storage** 部分：

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
      -o=jsonpath='{.items[0].metadata.name}') \
  cat /etc/registry/config.yml

version: 0.1
log:
  level: debug
http:
```

```
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  gcs:
    bucket: ocp-registry
auth:
  openshift:
    realm: openshift
middleware:
  registry:
  - name: openshift
  repository:
  - name: openshift
    options:
      pullthrough: True
      acceptschema2: True
      enforcequota: False
  storage:
  - name: openshift
```

或者，您可以查看 secret：

```
$ oc get secret registry-config -o jsonpath='{.data.config\.yml}' | base64 -d
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  gcs:
    bucket: ocp-registry
auth:
  openshift:
    realm: openshift
middleware:
  registry:
  - name: openshift
  repository:
  - name: openshift
    options:
      pullthrough: True
      acceptschema2: True
      enforcequota: False
  storage:
  - name: openshift
```

您可以通过在 GCP 控制台中查看 **Storage**，然后点 **Browser** 并选择存储桶，或运行 **gsutil** 命令验证镜像推送是否成功：

```
$ gsutil ls gs://ocp-registry/
gs://ocp-registry/docker/

$ gsutil du gs://ocp-registry/
7660385     gs://ocp-
registry/docker/registry/v2/blobs/sha256/03/033565e6892e5cc6dd03187d00a4575720a928db1
11274e0fbf31b410a093c10/data
7660385     gs://ocp-
registry/docker/registry/v2/blobs/sha256/03/033565e6892e5cc6dd03187d00a4575720a928db1
11274e0fbf31b410a093c10/
7660385     gs://ocp-registry/docker/registry/v2/blobs/sha256/03/
...
```

如果使用 **emptyDir** 卷，**/registry** 挂载点类似如下：

```
$ oc exec \
   $(oc get pod -l deploymentconfig=docker-registry \
   -o=jsonpath='{.items[0].metadata.name}')  -i -t -- df -h /registry
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc         30G  226M   30G   1% /registry


$ oc exec \
   $(oc get pod -l deploymentconfig=docker-registry \
   -o=jsonpath='{.items[0].metadata.name}')  -i -t -- ls -l /registry
total 0
drwxr-sr-x. 3 1000000000 1000000000 22 Jun 19 12:24 docker
```

23.2.4. 配置 OpenShift Container Platform 使用 GCP 存储

OpenShift Container Platform 可以使用持久性卷机制使用 GCP 存储。OpenShift Container Platform 在 GCP 中创建磁盘，并将磁盘附加到正确的实例。

GCP 磁盘是 **ReadWriteOnce** 访问模式，这意味着该卷可以被单一节点以读写模式挂载。如需更多信息，请参阅架构指南中的访问模式部分。

流程

1. 当使用 **gce-pd** 置备程序时，OpenShift Container Platform 会创建以下 **storageclass**，并且如果您使用 **openshift_cloudprovider_kind=gce** 和 **openshift_gcp_*** 变量。否则，如果您在没有使用 Ansible 的情况下配置 OpenShift Container Platform，且在安装过程中还没有创建 **storageclass**，则可以手动创建它：

```
$ oc get --export storageclass standard -o yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 annotations:
   storageclass.kubernetes.io/is-default-class: "true"
 creationTimestamp: null
 name: standard
 selfLink: /apis/storage.k8s.io/v1/storageclasses/standard
parameters:
```

```
 type: pd-standard
provisioner: kubernetes.io/gce-pd
reclaimPolicy: Delete
```

在请求一个 PV 并使用上一步中显示的 storageclass 后，OpenShift Container Platform 在 GCP 基础架构中创建磁盘。验证磁盘是否已创建：

```
$ gcloud compute disks list | grep kubernetes
kubernetes-dynamic-pvc-10ded514-7625-11e8-8c52-42010af00003  us-west1-b  10      pd-standard  READY
```

23.2.5. 关于 Red Hat OpenShift Container Storage

Red Hat OpenShift Container Storage(RHOCS)是 OpenShift Container Platform 内部或混合云中与无关的持久性存储供应商。作为红帽存储解决方案，RHOCS 与 OpenShift Container Platform 完全集成，用于部署、管理和监控，无论它是否安装在 OpenShift Container Platform（融合）或与 OpenShift Container Platform（独立）。OpenShift Container Storage 不仅限于单个可用区或节点，这使得它可能会停机。您可以在 RHOCS 3.11 部署指南中找到使用 RHOCS 的完整说明。

23.3. 使用 GCP 外部负载均衡器作为服务

您可以通过使用 **LoadBalancer** 服务在外部公开服务，将 OpenShift Container Platform 配置为使用 GCP 负载均衡器。OpenShift Container Platform 在 GCP 中创建负载均衡器，并创建必要的防火墙规则。

流程

1. 创建新应用程序：

```
$ oc new-app openshift/hello-openshift
```

2. 公开负载均衡器服务：

```
$ oc expose dc hello-openshift --name='hello-openshift-external' --type='LoadBalancer'
```

这个命令会创建类似以下示例的 **LoadBalancer** 服务：

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hello-openshift
  name: hello-openshift-external
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: port-1
    nodePort: 30714
    port: 8080
    protocol: TCP
    targetPort: 8080
  - name: port-2
    nodePort: 30122
```

```
    port: 8888
    protocol: TCP
    targetPort: 8888
  selector:
    app: hello-openshift
    deploymentconfig: hello-openshift
  sessionAffinity: None
  type: LoadBalancer
```

3. 验证服务是否已创建：

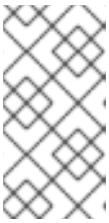
```
$ oc get svc
NAME                    TYPE          CLUSTER-IP      EXTERNAL-IP     PORT(S)
AGE
hello-openshift         ClusterIP     172.30.62.10    <none>          8080/TCP,8888/TCP
20m
hello-openshift-external   LoadBalancer   172.30.147.214   35.230.97.224
8080:31521/TCP,8888:30843/TCP   19m
```

LoadBalancer 类型和外部 IP 值表示服务使用 GCP 负载均衡器公开应用程序。

OpenShift Container Platform 在 GCP 基础架构中创建所需的对象，例如：

- 防火墙规则：

```
$ gcloud compute firewall-rules list | grep k8s
k8s-4612931a3a47c204-node-http-hc       my-net  INGRESS    1000      tcp:10256
k8s-fw-a1a8afaa7762811e88c5242010af0000  my-net  INGRESS    1000
tcp:8080,tcp:8888
```



注意

这些防火墙规则应用到使用 **<openshift_gcp_prefix>ocp** 标记的实例。例如，如果 **openshift_gcp_prefix** 参数的值设置为 **mycluster**，则必须标记带有 **myclusterocp** 的节点。有关如何在 Compute Engine 虚拟机实例中添加网络标签的更多信息，请参阅添加和删除网络标签。

- 健康检查：

```
$ gcloud compute http-health-checks list | grep k8s
k8s-4612931a3a47c204-node       10256  /healthz
```

- 一个负载均衡器：

```
$ gcloud compute target-pools list | grep k8s
a1a8afaa7762811e88c5242010af0000  us-west1  NONE                  k8s-
4612931a3a47c204-node
$ gcloud compute forwarding-rules list | grep a1a8afaa7762811e88c5242010af0000
a1a8afaa7762811e88c5242010af0000  us-west1  35.230.97.224  TCP         us-
west1/targetPools/a1a8afaa7762811e88c5242010af0000
```

要验证负载均衡器是否已正确配置，请从外部主机运行以下命令：

```
$ curl 35.230.97.224:8080
Hello OpenShift!
```

第 24 章 为 AZURE 配置

您可以将 OpenShift Container Platform 配置为使用 Microsoft Azure 负载均衡器 和 磁盘作为持久性应用程序数据。

24.1. 开始前

24.1.1. 为 Microsoft Azure 配置授权

Azure 角色

为 OpenShift Container Platform 配置 Microsoft Azure 需要以下 Microsoft Azure 角色：

贡献者	创建和管理所有类型的 Microsoft Azure 资源。

请参阅 Classic 订阅管理员角色对比Azure RBAC 角色与.Azure AD 管理员角色文档。

权限

为 OpenShift Container Platform 配置 Microsoft Azure 需要服务主体，它允许创建和管理 Kubernetes 服务负载均衡器和磁盘以用于持久性存储。服务主体值在安装时定义并部署到 Azure 配置文件，该文件位于 OpenShift Container Platform master 和节点主机上的 **/etc/origin/cloudprovider/azure.conf** 中。

流程

1. 使用 Azure CLI 获取帐户订阅 ID：

```
# az account list
[
{
  "cloudName": "AzureCloud",
  "id": "<subscription>", (1)
  "isDefault": false,
  "name": "Pay-As-You-Go",
  "state": "Enabled",
  "tenantId": "<tenant-id>",
  "user": {
    "name": "admin@example.com",
    "type": "user"
  }
]
```

(1) 用于创建新权限的订阅 ID。

2. 使用 Microsoft Azure 角色及 Microsoft Azure 订阅和资源组的范围创建服务主体。记录这些值的输出，以便在定义清单时使用。使用上一步中的 **<subscription** > 值替换以下值：

```
# az ad sp create-for-rbac --name openshiftcloudprovider \
    --password <secret> --role contributor \
    --scopes /subscriptions/<subscription>/resourceGroups/<resource-group>

Retrying role assignment creation: 1/36
```

```
Retrying role assignment creation: 2/36
{
  "appId": "<app-id>",
  "displayName": "ocpcloudprovider",
  "name": "http://ocpcloudprovider",
  "password": "<secret>",
  "tenant": "<tenant-id>"
}
```

24.1.2. 配置 Microsoft Azure 对象

将 OpenShift Container Platform 与 Microsoft Azure 集成需要以下组件或服务，以创建高度可用的、功能完整的环境。

重要

为确保可以启动适当数量的实例，请在创建实例前请求增加 Microsoft 的 CPU 配额。

资源组

资源组包含用于部署的所有 Microsoft Azure 组件，包括网络、负载均衡器、虚拟机和 DNS。配额和权限可应用到资源组，以控制和管理在 Microsoft Azure 上部署的资源。资源组按照区域区域创建并定义。为 OpenShift Container Platform 环境创建的所有资源都应位于同一区域内，以及同一资源组内。

如需更多信息，请参阅 Azure Resource Manager 概述。

Azure Virtual Networks

Azure Virtual Networks 用于将 Azure 云网络相互隔离。实例和负载均衡器使用虚拟网络来允许相互通信和与互联网的通信。虚拟网络允许创建一个或多个子网，供资源组内的组件使用。您还可以将虚拟网络连接到不同的 VPN 服务，允许与内部服务通信。

如需更多信息，请参阅什么是 Azure 虚拟网络？

Azure DNS

Azure 提供了一个托管的 DNS 服务，它提供内部和外部的主机名和负载均衡器解析。请参考环境使用 DNS 区域来托管三个 DNS A 记录，允许将公共 IP 映射到 OpenShift Container Platform 资源和 bastion。

如需更多信息，请参阅 Azure DNS?

负载平衡

Azure 负载均衡器允许网络连接在 Azure 环境中虚拟机上运行的服务扩展和高可用性。

请参阅什么是 Azure Load Balancer?

存储帐户

存储帐户允许虚拟机等资源访问 Microsoft Azure 提供的不同类型的存储组件。在安装过程中，存储帐户定义用于 OpenShift Container Platform registry 的基于对象的 **blob** 存储的位置。

如需更多信息，请参阅 Azure Storage 简介，或为 Microsoft Azure 配置 OpenShift Container Platform registry 部分，以了解有关为 registry 创建存储帐户的步骤。

Service Principal

Azure 提供了创建访问、管理或创建 Azure 组件的服务帐户的功能。服务帐户授予对特定服务的 API 访问权限。例如，服务主体允许 Kubernetes 或 OpenShift Container Platform 实例请求持久性存储和负载均衡器。服务主体允许精细访问实例或特定功能的用户。

如需更多信息，请参阅 Azure Active Directory 中的应用程序和服务主体对象。

可用性集

可用性集可确保部署的虚拟机在集群中的多个隔离硬件节点之间分布。该分布有助于确保在云提供商硬件上维护时，实例不会在一个特定节点上运行。

您应该根据角色将实例划分为不同的可用性集。例如，一个可用性集含有三个 master 主机，一个可用性集包含基础架构主机，以及一个包含应用主机的可用性集。这允许分段并在 OpenShift Container Platform 中使用外部负载均衡器。

如需更多信息，请参阅管理 Linux 虚拟机可用性。

网络安全组

网络安全组(NSG)提供允许或拒绝在 Azure Virtual Network 中部署的资源的流量的规则列表。NSG 使用数字优先级值和规则来定义允许哪些项目相互通信。您可以对允许通信发生的情况施加限制，比如仅在负载均衡器内部或从任何位置处理。

管理员可通过优先级值授予允许或不允许进行端口通信的顺序的粒度值。

如需更多信息，请参阅 计划虚拟网络。

实例大小

成功的 OpenShift Container Platform 环境需要满足最低硬件要求。

如需更多信息，请参阅 OpenShift Container Platform 文档或云服务 大小中的 Minimum Had ware 要求部分。

24.2. AZURE 配置文件

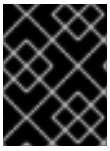
为 Azure 配置 OpenShift Container Platform 需要每个节点主机上的 */etc/azure/azure.conf* 文件。

如果文件不存在，您可以创建该文件。

```
tenantId: <> 1
subscriptionId: <> 2
aadClientId: <> 3
aadClientSecret: <> 4
aadTenantId: <> 5
resourceGroup: <> 6
cloud: <> 7
location: <> 8
vnetName: <> 9
securityGroupName: <> 10
primaryAvailabilitySetName: <> 11
```

1. 集群在其中部署订阅的 AAD 租户 ID。
2. 集群部署到的 Azure 订阅 ID。

3. 带有 RBAC 访问权限的 AAD 应用程序的客户端 ID 与 Azure RM API 对话。
4. AAD 应用程序的客户端 secret，具有 RBAC 访问权限，与 Azure RM API 对话。
5. 确保这与租户 ID 相同（可选）。
6. Azure 虚拟机所属的 Azure Resource Group 名称。
7. 特定云区域。例如，**AzurePublicCloud**。
8. 紧凑风格 Azure 区域。例如，**southeastasia** （可选）。
9. 包含实例并在创建负载均衡器时使用的虚拟网络。
10. 与实例和负载均衡器关联的安全组名称。
11. 在创建负载均衡器（可选）资源时要使用的可用性集。



重要

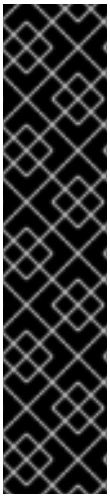
用于访问实例的 NIC 必须设置 **internal-dns-name**，否则节点将无法重新加入集群，显示构建日志，并会导致 **oc rsh** 无法正常工作。

24.3. MICROSOFT AZURE 上的 OPENSHIFT CONTAINER PLATFORM 示例

以下示例清单假设创建了以下项目：

- 资源组
- Azure 虚拟网络
- 包含所需 OpenShift Container Platform 端口的一个或多个网络安全组
- 存储帐户
- 服务主体
- 两个负载均衡器
- 路由器和 OpenShift Container Platform Web 控制台的两个或多个 DNS 条目
- 三个可用性集
- 三个 master 实例
- 三个基础架构实例
- 一个或多个应用程序实例

以下清单使用默认的 **storageclass** 创建持久性卷，供指标、日志记录和服务目录组件由服务主体管理。registry 使用 Microsoft Azure blob 存储。



重要

如果 Microsoft Azure 实例使用受管磁盘，请在清单中提供以下变量：

openshift_storageclass_parameters={'kind': 'managed', 'storageaccounttype':'Premium_LRS'}

或者

openshift_storageclass_parameters={'kind': 'managed', 'storageaccounttype':'Standard_LRS'}

这样可确保 **storageclass** 为 **PV** 创建正确的磁盘类型，因为它与部署的实例相关。如果使用非受管磁盘，则 **存储类将使用 shared** 参数，允许为 **PV** 创建不受管理的磁盘。

```
[OSEv3:children]
masters
etcd
nodes

[OSEv3:vars]
ansible_ssh_user=cloud-user
ansible_become=true
openshift_cloudprovider_kind=azure

#cloudprovider
openshift_cloudprovider_kind=azure
openshift_cloudprovider_azure_client_id=v9c97ead-1v7E-4175-93e3-623211bed834
openshift_cloudprovider_azure_client_secret=s3r3tR3gistryN0special
openshift_cloudprovider_azure_tenant_id=422r3f91-21fe-4esb-vad5-d96dfeooee5d
openshift_cloudprovider_azure_subscription_id=6003c1c9-d10d-4366-86cc-e3ddddcooe2d
openshift_cloudprovider_azure_resource_group=openshift
openshift_cloudprovider_azure_location=eastus
#endcloudprovider

oreg_auth_user=service_account (1)
oreg_auth_password=service_account_token (2)
openshift_master_api_port=443
openshift_master_console_port=443
openshift_hosted_router_replicas=3
openshift_hosted_registry_replicas=1
openshift_master_cluster_method=native
openshift_master_cluster_hostname=openshift-master.example.com
openshift_master_cluster_public_hostname=openshift-master.example.com
openshift_master_default_subdomain=apps.openshift.example.com
openshift_deployment_type=openshift-enterprise
openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true', 'login': 'true', 'kind':
'LDAPPasswordIdentityProvider', 'attributes': {'id': ['dn'], 'email': ['mail'], 'name': ['cn'],
'preferredUsername': ['uid']}, 'bindDN': 'uid=admin,cn=users,cn=accounts,dc=example,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt', 'insecure': 'false', 'url':
'ldap://ldap.example.com/cn=users,cn=accounts,dc=example,dc=com?uid?sub?(memberOf=cn=ose-
user,cn=groups,cn=accounts,dc=example,dc=com)'}]
networkPluginName=redhat/ovs-networkpolicy
openshift_examples_modify_imagestreams=true
```

```
# Storage Class change to use managed storage
openshift_storageclass_parameters={'kind': 'managed', 'storageaccounttype': 'Standard_LRS'}

# service catalog
openshift_enable_service_catalog=true
openshift_hosted_etcd_storage_kind=dynamic
openshift_hosted_etcd_storage_volume_name=etcd-vol
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_etcd_storage_volume_size=SC_STORAGE
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}

# metrics
openshift_metrics_install_metrics=true
openshift_metrics_cassandra_storage_type=dynamic
openshift_metrics_storage_volume_size=20Gi
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra": "true"}

# logging
openshift_logging_install_logging=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_storage_volume_size=50Gi
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"}

# Setup azure blob registry storage
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_azure_blob_accountkey=uZdkVlbca6xzwBqK8VDz15/loLUoc8I6cPfI
31ZS+QOSxL6ylWT6CLrcadSqvtNTMgztxH4CGjYfVnRNUhvMiA==
openshift_hosted_registry_storage_provider=azure_blob
openshift_hosted_registry_storage_azure_blob_accountname=registry
openshift_hosted_registry_storage_azure_blob_container=registry
openshift_hosted_registry_storage_azure_blob_realm=core.windows.net

[masters]
ocp-master-1
ocp-master-2
ocp-master-3

[etcd]
ocp-master-1
ocp-master-2
ocp-master-3

[nodes]
ocp-master-1 openshift_node_group_name="node-config-master"
ocp-master-2 openshift_node_group_name="node-config-master"
ocp-master-3 openshift_node_group_name="node-config-master"
ocp-infra-1 openshift_node_group_name="node-config-infra"
ocp-infra-2 openshift_node_group_name="node-config-infra"
ocp-infra-3 openshift_node_group_name="node-config-infra"
ocp-app-1 openshift_node_group_name="node-config-compute"
```

1 2 如果您使用需要身份验证的容器 registry，如默认容器镜像 registry，请指定该帐户的凭证。请参阅访问并配置 Red Hat Registry 。

24.4. 为 MICROSOFT AZURE 配置 OPENSHIFT CONTAINER PLATFORM

您可以通过两种方式为 Microsoft Azure 配置 OpenShift Container Platform：

- 使用 Ansible
- 通过修改 *master-config.yaml* 文件手动进行

24.4.1. 使用 Ansible 为 Azure 配置 OpenShift Container Platform

您可以在安装时为 Azure 配置 OpenShift Container Platform。

将以下内容添加到位于 */etc/ansible/hosts* 的 Ansible 清单文件中，以配置 Microsoft Azure 的 OpenShift Container Platform 环境：

```
[OSEv3:vars]
openshift_cloudprovider_kind=azure
openshift_cloudprovider_azure_client_id=<app_ID> 1
openshift_cloudprovider_azure_client_secret=<secret> 2
openshift_cloudprovider_azure_tenant_id=<tenant_ID> 3
openshift_cloudprovider_azure_subscription_id=<subscription> 4
openshift_cloudprovider_azure_resource_group=<resource_group> 5
openshift_cloudprovider_azure_location=<location> 6
```

1 服务主体的 app ID 值。

2 包含服务主体密码的 secret。

3 服务主体所在的租户。

4 服务主体使用的订阅。

5 服务帐户所在的资源组。

6 资源组所在的 Microsoft Azure 位置。

使用 Ansible 安装也会创建并配置以下文件以适合您的 Microsoft Azure 环境：

- */etc/origin/cloudprovider/azure.conf*
- */etc/origin/master/master-config.yaml*
- */etc/origin/node/node-config.yaml*

24.4.2. 为 Microsoft Azure 手动配置 OpenShift Container Platform

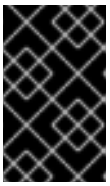
24.4.2.1. 为 Microsoft Azure 手动配置 master 主机

在所有 master 主机上执行以下操作。

流程

1. 在所有 master 上编辑位于 **/etc/origin/master/master-config.yaml** 的 master 配置文件，并更新 **apiServerArguments** 和 **controllerArguments** 部分的内容：

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "azure"
    cloud-config:
      - "/etc/origin/cloudprovider/azure.conf"
  controllerArguments:
    cloud-provider:
      - "azure"
    cloud-config:
      - "/etc/origin/cloudprovider/azure.conf"
```



重要

在触发容器化安装时，只有 */etc/origin* 和 */var/lib/origin* 目录被挂载到 master 和节点容器中。因此，确保 *master-config.yaml* 位于 */etc/origin/master* 目录中，而不是 */etc/*。

2. 使用 Ansible 为 Microsoft Azure 配置 OpenShift Container Platform 时，会自动创建 **/etc/origin/cloudprovider/azure.conf** 文件。由于要为 Microsoft Azure 手动配置 OpenShift Container Platform，所以您必须在所有节点实例中创建该文件，并包括以下内容：

```
tenantId: <tenant_ID> (1)
subscriptionId: <subscription> (2)
aadClientId: <app_ID> (3)
aadClientSecret: <secret> (4)
aadTenantId: <tenant_ID> (5)
resourceGroup: <resource_group> (6)
location: <location> (7)
```

(1) 服务主体所在的租户。

(2) 服务主体使用的订阅。

(3) 服务主体的 appID 值。

(4) 包含服务主体密码的 secret。

(5) 服务主体所在的租户。

(6) 服务帐户所在的资源组。

(7) 资源组所在的 Microsoft Azure 位置。

3. 重启 OpenShift Container Platform master 服务：

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

24.4.2.2. 为 Microsoft Azure 手动配置节点主机

在所有节点主机上执行以下内容。

流程

1. 编辑 适当的节点配置映射 并更新 **kubeletArguments** 部分的内容：

```
kubeletArguments:
  cloud-provider:
    - "azure"
  cloud-config:
    - "/etc/origin/cloudprovider/azure.conf"
```

重要

用于访问实例的 NIC 必须设置内部 DNS 名称，节点将无法重新加入集群，显示构建日志到控制台，并会导致 **oc rsh** 无法正常工作。

2. 重启所有节点上的 OpenShift Container Platform 服务：

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

24.4.3. 为 Microsoft Azure 配置 OpenShift Container Platform registry

Microsoft Azure 提供 OpenShift Container Platform 可以用来使用 OpenShift Container Platform 容器镜像 registry 存储容器镜像的对象云存储。

如需更多信息，请参阅 Azure 文档中的 Cloud Storage。

您可以使用 Ansible 配置 registry，或者通过配置 registry 配置文件来手动配置 registry。

先决条件

在安装前，您必须创建一个存储帐户来托管 registry 镜像。以下命令会创建一个在安装过程中用于镜像存储的存储帐户：

您可以使用 Microsoft Azure blob 存储来存储容器镜像。OpenShift Container Platform registry 使用 blob 存储来允许 registry 大小动态增长，而无需管理员进行干预。

1. 创建 Azure 存储帐户：

```
az storage account create
--name <account_name> \
--resource-group <resource_group> \
--location <location> \
--sku Standard_LRS
```

这会创建一个帐户密钥。查看帐户密钥：

```
az storage account keys list \
   --account-name <account-name> \
   --resource-group <resource-group> \
   --output table

KeyName    Permissions    Value
key1       Full           <account-key>
key2       Full           <extra-account-key>
```

OpenShift Container Platform registry 配置只需要一个帐户键值。

选项 1：使用 Ansible 为 Azure 配置 OpenShift Container Platform registry

流程

1. 将 registry 的 Ansible 清单配置为使用存储帐户：

```
[OSEv3:vars]
# Azure Registry Configuration
openshift_hosted_registry_replicas=1 (1)
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_azure_blob_accountkey=<account_key> (2)
openshift_hosted_registry_storage_provider=azure_blob
openshift_hosted_registry_storage_azure_blob_accountname=<account_name> (3)
openshift_hosted_registry_storage_azure_blob_container=<registry> (4)
openshift_hosted_registry_storage_azure_blob_realm=core.windows.net
```

(1) 用于配置的副本数。

(2) 与 <account-name> 关联的帐户密钥。

(3) 存储帐户名称。

(4) 用于存储数据的目录。默认是 **registry**

选项 2：为 Microsoft Azure 手动配置 OpenShift Container Platform registry

要使用 Microsoft Azure 对象存储，请编辑 registry 的配置文件并挂载到 registry pod。

流程

1. 导出当前的 ***config.yml*** ：

```
$ oc get secret registry-config \
    -o jsonpath='{.data.config\.yml}' -n default | base64 -d \
  >> config.yml.old
```

2. 从旧 ***config.yml*** 创建新配置文件：

```
$ cp config.yml.old config.yml
```

3. 编辑该文件使其包含 Azure 参数：

```
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  azure:
    accountname: <account-name> 1
    accountkey: <account-key> 2
    container: registry 3
    realm: core.windows.net 4
```

1. 使用存储帐户名称替换。
2. 与 <account-name> 关联的帐户密钥。
3. 用于存储数据的目录。默认是 **registry**
4. 默认情况下，存储 realm **core.windows.net**

4. 删除 **registry-config** secret：

```
$ oc delete secret registry-config -n default
```

5. 重新创建 secret 来引用更新的配置文件：

```
$ oc create secret generic registry-config \
    --from-file=config.yml -n default
```

6. 重新部署 registry 以读取更新的配置：

```
$ oc rollout latest docker-registry -n default
```

验证 registry 正在使用 blob 对象存储

验证 registry 是否使用 Microsoft Azure blob 存储：

流程

1. 在成功部署 registry 后，registry **deploymentconfig** 将始终显示 registry 使用 **emptydir** 而不是 Microsoft Azure blob 存储：

```
$ oc describe dc docker-registry -n default
...
Mounts:
  ...
  /registry from registry-storage (rw)
Volumes:
registry-storage:
Type:       EmptyDir 1
...
```

1. 共享 pod 生命周期的临时目录。

2. 检查 */registry* 挂载点是否为空。这是 Microsoft Azure 存储将使用的卷：

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}')  -i -t -- ls -l /registry
total 0
```

3. 如果为空，则会因为 Microsoft Azure blob 配置在 **registry-config** secret 中执行：

```
$ oc describe secret registry-config
Name:         registry-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
config.yml:  398 bytes
```

4. 安装程序会使用扩展的 registry 功能创建带有所需配置的 *config.yml* 文件，如安装文档所述。要查看配置文件，包括存储存储桶配置的 **storage** 部分：

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
      -o=jsonpath='{.items[0].metadata.name}') \
  cat /etc/registry/config.yml

  version: 0.1
  log:
    level: debug
  http:
    addr: :5000
  storage:
    delete:
      enabled: true
    cache:
      blobdescriptor: inmemory
    azure:
      accountname: registry
      accountkey:
uZekVBJBa6xzwAqK8EDz15/hoHUoc8I6cPfP31ZS+QOSxLfo7WT7CLrVPKaqvtNTMgztxH7C
GjYfpFRNUhvMiA==
      container: registry
      realm: core.windows.net
  auth:
    openshift:
      realm: openshift
  middleware:
    registry:
    - name: openshift
    repository:
    - name: openshift
      options:
```

```
        pullthrough: True
        acceptschema2: True
        enforcequota: False
    storage:
    - name: openshift
```

或者，您可以查看 secret：

```
$ oc get secret registry-config -o jsonpath='{.data.config\.yml}' | base64 -d
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  azure:
    accountname: registry
    accountkey:
uZekVBJBa6xzwAqK8EDz15/hoHUoc8I6cPfP31ZS+QOSxLfo7WT7CLrVPKaqvtNTMgztxH7C
GjYfpFRNUhvMiA==
    container: registry
    realm: core.windows.net
auth:
  openshift:
    realm: openshift
middleware:
  registry:
  - name: openshift
  repository:
  - name: openshift
    options:
      pullthrough: True
      acceptschema2: True
      enforcequota: False
  storage:
  - name: openshift
```

如果使用 **emptyDir** 卷，**/registry** 挂载点类似如下：

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}')  -i -t -- df -h /registry
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc         30G  226M   30G   1% /registry


$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}')  -i -t -- ls -l /registry
total 0
drwxr-sr-x. 3 1000000000 1000000000 22 Jun 19 12:24 docker
```

24.4.4. 配置 OpenShift Container Platform 使用 Microsoft Azure 存储

OpenShift Container Platform 可以使用持久性卷机制使用 Microsoft Azure 存储。OpenShift Container Platform 在资源组中创建磁盘，并将磁盘附加到正确的实例。

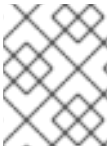
流程

1. 当在 Ansible 清单中使用 **openshift_cloudprovider_kind=azure** 和 **openshift_cloud_provider_azure** 配置 Azure 云供应商时，会创建以下 **storageclass** ：

```
$ oc get --export storageclass azure-standard -o yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  creationTimestamp: null
  name: azure-standard
parameters:
  kind: Shared
  storageaccounttype: Standard_LRS
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

   如果不使用 Ansible 启用 OpenShift Container Platform 和 Microsoft Azure 集成，您可以手动创建 **storageclass**。如需更多信息，请参阅 动态置备和创建存储类部分。

2. 目前，默认的 **storageclass** kind 是**共享的**，这意味着 Microsoft Azure 实例必须使用非受管磁盘。您可以选择通过提供 **openshift_storageclass_parameters={'kind' 来修改实例来使用受管磁盘：'managed', 'storageaccounttype':'Premium_LRS'}** 或 **openshift_storageclass_parameters={'kind':'managed', 'storageaccounttype':安装 Ansible 清单文件中 'Standard_LRS'}** 变量。



注意

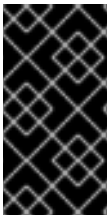
Microsoft Azure 磁盘是 **ReadWriteOnce** 访问模式，这意味着该卷可以被单一节点以读写模式挂载。如需更多信息，请参阅架构指南中的访问模式部分。

24.4.5. 关于 Red Hat OpenShift Container Storage

Red Hat OpenShift Container Storage(RHOCS)是 OpenShift Container Platform 内部或混合云中与无关的持久性存储供应商。作为红帽存储解决方案，RHOCS 与 OpenShift Container Platform 完全集成，用于部署、管理和监控，无论它是否安装在 OpenShift Container Platform（融合）或与 OpenShift Container Platform（独立）。OpenShift Container Storage 不仅限于单个可用区或节点，这使得它可能会停机。您可以在 RHOCS 3.11 部署指南中找到使用 RHOCS 的完整说明。

24.5. 使用 MICROSOFT AZURE 外部负载均衡器作为服务

OpenShift Container Platform 可以通过使用 **LoadBalancer** 服务向外部公开服务来利用 Microsoft Azure 负载均衡器。OpenShift Container Platform 在 Microsoft Azure 中创建负载均衡器，并创建正确的防火墙规则。



重要

目前，当将额外变量用作云供应商时，以及将其用作外部负载均衡器时，会导致在 Microsoft Azure 基础架构中包含额外的变量。如需更多信息，请参阅以下内容：

- https://bugzilla.redhat.com/show_bug.cgi?id=1613546

先决条件

确保位于 */etc/origin/cloudprovider/azure.conf* 的 Azure 配置文件 使用适当的对象正确配置。如需示例 */etc/origin/cloudprovider/azure.conf* 文件，请参阅 为 Microsoft Azure 手动配置 OpenShift Container Platform 部分。

添加值后，重启所有主机上的 OpenShift Container Platform 服务：

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

24.5.1. 使用负载均衡器部署示例应用程序

流程

1. 创建新应用程序：

```
$ oc new-app openshift/hello-openshift
```

2. 公开负载均衡器服务：

```
$ oc expose dc hello-openshift --name='hello-openshift-external' --type='LoadBalancer'
```

这会创建类似如下的 **Loadbalancer** 服务：

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hello-openshift
  name: hello-openshift-external
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: port-1
    nodePort: 30714
    port: 8080
    protocol: TCP
    targetPort: 8080
  - name: port-2
    nodePort: 30122
    port: 8888
    protocol: TCP
    targetPort: 8888
  selector:
    app: hello-openshift
```

```
    deploymentconfig: hello-openshift
  sessionAffinity: None
  type: LoadBalancer
```

3. 验证服务是否已创建：

```
$ oc get svc
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
AGE
hello-openshift         ClusterIP      172.30.223.255  <none>          8080/TCP,8888/TCP
1m
hello-openshift-external  LoadBalancer  172.30.99.54    40.121.42.180
8080:30714/TCP,8888:30122/TCP  4m
```

LoadBalancer 类型和 **External-IP** 字段表示该服务正在使用 Microsoft Azure 负载均衡器来公开应用程序。

这会在 Azure 基础架构中创建以下所需的对象：

- 一个负载均衡器：

```
az network lb list -o table
Location    Name         ProvisioningState    ResourceGroup    ResourceGuid
----------  -----------  -------------------  ---------------  ------------------------------------
eastus      kubernetes   Succeeded            refarch-azr      30ec1980-b7f5-407e-aa4f-
e570f06f168d
eastus      OcpMasterLB  Succeeded            refarch-azr      acb537b2-8a1a-45d2-aae1-
ea9eabfaea4a
eastus      OcpRouterLB  Succeeded            refarch-azr      39087c4c-a5dc-457e-a5e6-
b25359244422
```

要验证负载均衡器是否已正确配置，请从外部主机运行以下命令：

```
$ curl 40.121.42.180:8080 (1)
Hello OpenShift!
```

(1) 使用上面的 **EXTERNAL-IP** 验证步骤中的值以及端口号替换。

第 25 章 为 VMWARE VSPHERE 配置

您可以将 OpenShift Container Platform 配置为使用 VMware vSphere VMDK 作为 PersistentVolume。此配置可以包括 使用 VMware vSphere VMDK 作为应用程序数据的持久性存储。

vSphere Cloud Provider 允许在 OpenShift Container Platform 中使用 vSphere 管理的存储，并支持 Kubernetes 使用的每个存储：

- PersistentVolume(PV)
- PersistentVolumesClaim (PVC)
- StorageClass

有状态容器化应用程序请求的 PersistentVolume 可以在 VMware vSAN、VVOL、VMFS 或 NFS 数据存储上置备。

Kubernetes PV 在 Pod 规格中定义。如果您使用静态置备，当使用静态置备时，可以直接引用 VMDK 文件，或者首选使用 Dynamic Provisioning。

对 vSphere Cloud Provider 的最新更新位于 vSphere Storage for Kubernetes。

25.1. 开始前

25.1.1. 要求

VMware vSphere

重要

不支持独立 ESXi。

- 如果您希望支持完整的 VMware Validate Design，则需要 vSphere 版本 6.0.x 最少使用 6.7 U1b 版本。
- 支持 vSAN、VMFS 和 NFS。
  - VSAN 支持仅限于一个 vCenter 中的集群。

注意

OpenShift Container Platform 3.11 支持并部署到 vSphere 7 集群上。如果您使用 vSphere in-tree 存储驱动程序、vSAN、VMFS 和 NFS 存储选项。

先决条件

您必须在每个 Node 虚拟机上安装 VMware 工具。如需更多信息，请参阅安装 VMware 工具。

您可以使用开源 VMware **govmomi** CLI 工具进行额外的配置和故障排除。例如，请参阅以下 **govc** CLI 配置：

```
export GOVC_URL='vCenter IP OR FQDN'
export GOVC_USERNAME='vCenter User'
export GOVC_PASSWORD='vCenter Password'
```

```
export GOVC_INSECURE=1
```

25.1.1.1. 权限

创建角色并分配给 vSphere Cloud Provider。需要具有所需权限集的 vCenter 用户。

通常，分配给 vSphere Cloud Provider 的 vSphere 用户必须具有以下权限：

- 对节点虚拟机的父实体，如 **文件夹**、**主机**、数据中心、数据存储文件夹、数据存储群集等的 **Read**权限。
- **VirtualMachine.Inventory.Create/Delete** permissions on the **vsphere.conf** 定义的资源池 - 用于创建和删除测试虚拟机。

有关创建自定义角色、用户和角色分配的步骤，请参阅 vSphere 文档中心。

vSphere Cloud Provider 支持跨越多个 vCenter 的 OpenShift Container Platform 集群。请确定为所有 vCenter 正确设置了所有以上权限。

动态置备权限

建议采用动态持久性卷创建。

角色	权限	实体	传播到子对象
manage-k8s-node-vms	Resource.AssignVMToPool, VirtualMachine.Config.AddExistingDisk, VirtualMachine.Config.AddNewDisk, VirtualMachine.Config.AddRemoveDevice, VirtualMachine.Config.RemoveDisk, VirtualMachine.Inventory.Create, VirtualMachine.Inventory.Delete, VirtualMachine.Config.Settings	Cluster, Hosts, VM Folder	是
manage-k8s-volumes	Datastore.AllocateSpace, Datastore.FileManagement（低级别文件操作）	数据存储	否
k8s-system-read-and-spbm-profile-view	StorageProfile.View（Profile 驱动的存储视图）	vCenter	否

角色	权限	实体	传播到子对象
只读（不存在的默认角色）	system.Anonymous、system.Read、System.View	Data, Datastore Cluster, Datastore Storage Folder	否

静态置备权限

如果创建 PVC 与静态置备的 PV 绑定，并将重新声明策略设置为删除，则只有 manage-k8s-volumes role 需要 Datastore.FileManagement。删除 PVC 时，关联的静态置备的 PV 也会被删除。

角色	权限	实体	传播到 Children
manage-k8s-node-vms	VirtualMachine.Config.AddExistingDisk, VirtualMachine.Config.AddNewDisk, VirtualMachine.Config.AddRemoveDevice, VirtualMachine.Config.RemoveDisk	VM Folder	是
manage-k8s-volumes	Datastore.FileManagement（低级别文件操作）	数据存储	否
只读（不存在的默认角色）	system.Anonymous、system.Read、System.View	vCenter, Datacenter, Datastore Cluster, Datastore Storage Folder, Cluster, Hosts	no ...

流程

1. 创建虚拟机 文件夹 并将 OpenShift Container Platform 节点虚拟机移到这个文件夹。
2. 为每个 Node 虚拟机将 **disk.EnableUUID** 参数设置为 **true**。这个设置可确保 VMware vSphere 的 Virtual Machine Disk (VMDK) 始终为虚拟机显示一致的 UUID，允许正确挂载磁盘。
   每个要加入集群的 VM 节点都必须将 **disk.EnableUUID** 参数设置为 **true**。要设置这个值，请按照 vSphere 控制台或 **govc** CLI 工具的步骤进行操作：
   a. 在 vSphere HTML Client 中，导航到 **VM properties → VM Options → Advanced → Configuration Parameters → disk.enableUUID=TRUE**
   b. 或使用 govc CLI 来查找 Node VM 路径：

```
$govc ls /datacenter/vm/<vm-folder-name>
```

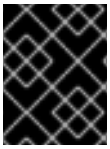
   i. 将所有虚拟机的 **disk.EnableUUID** 设置为 **true** ：

```
$govc vm.change -e="disk.enableUUID=1" -vm='VM Path'
```

注意

如果从虚拟机模板创建了 OpenShift Container Platform 节点虚拟机，您可以在模板虚拟机上设置 **disk.EnableUUID=1**。从该模板克隆的虚拟机将继承此属性。

25.1.1.2. 将 OpenShift Container Platform 与 vMotion 搭配使用



重要

OpenShift Container Platform 通常支持仅计算 vMotion。使用 Storage vMotion 可能会导致问题且不受支持。

如果您在 pod 中使用 vSphere 卷，请手动或通过 Storage vMotion 在数据存储间迁移虚拟机，这会导致 OpenShift Container Platform 持久性卷(PV)对象中的无效引用。这些引用可防止受影响的 pod 启动，并可能导致数据丢失。

同样，OpenShift Container Platform 不支持在数据存储间有选择地迁移 VMDK，使用数据存储集群进行虚拟机置备或动态或静态置备 PV，或使用作为数据存储集群一部分的数据存储来动态或静态置备 PV。

25.2. 为 VSPHERE 配置 OPENSHIFT CONTAINER PLATFORM

您可以通过两种方式为 vSphere 配置 OpenShift Container Platform：

- 使用 Ansible
- 通过修改 *master-config.yaml* 文件手动进行

25.2.1. 选项 1：使用 Ansible 为 vSphere 配置 OpenShift Container Platform

您可以通过修改 Ansible 清单文件，为 VMware vSphere(VCP)配置 OpenShift Container Platform。这些更改可在安装后或现有集群进行。

流程

1. 将以下内容添加到 Ansible 清单文件中：

```
[OSEv3:vars]
openshift_cloudprovider_kind=vsphere
openshift_cloudprovider_vsphere_username=administrator@vsphere.local (1)
openshift_cloudprovider_vsphere_password=<password>
openshift_cloudprovider_vsphere_host=10.x.y.32 (2)
openshift_cloudprovider_vsphere_datacenter=<Datacenter> (3)
openshift_cloudprovider_vsphere_datastore=<Datastore> (4)
```

(1) 在 vSphere 中创建和附加磁盘的适当权限的用户名。

(2) vCenter 服务器地址。

(3) OpenShift Container Platform 虚拟机所在的 vCenter Datacenter 名称。

(4) 用于创建 VMDK 的数据存储。

2. 运行 **deploy_cluster.yml** playbook。

```
$ ansible-playbook -i <inventory_file> \
  playbooks/deploy_cluster.yml
```

使用 Ansible 安装也会创建并配置以下文件以适合您的 vSphere 环境：

- */etc/origin/cloudprovider/vsphere.conf*
- */etc/origin/master/master-config.yaml*
- */etc/origin/node/node-config.yaml*

作为参考，显示完整清单，如下所示：

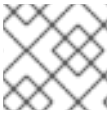
OpenShift Container Platform 需要 **openshift_cloudprovider_vsphere_** 值，以便能够在持久性卷的数据存储上创建 **vSphere** 资源。

```
$ cat /etc/ansible/hosts

[OSEv3:children]
ansible
masters
infras
apps
etcd
nodes
lb

[OSEv3:vars]
become=yes
ansible_become=yes
ansible_user=root
oreg_auth_user=service_account (1)
oreg_auth_password=service_account_token (2)
openshift_deployment_type=openshift-enterprise
# Required per https://access.redhat.com/solutions/3480921
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

# vSphere Cloud provider
openshift_cloudprovider_kind=vsphere
openshift_cloudprovider_vsphere_username="administrator@vsphere.local"
openshift_cloudprovider_vsphere_password="password"
openshift_cloudprovider_vsphere_host="vcsa65-dc1.example.com"
openshift_cloudprovider_vsphere_datacenter=Datacenter
openshift_cloudprovider_vsphere_cluster=Cluster
openshift_cloudprovider_vsphere_resource_pool=ResourcePool
openshift_cloudprovider_vsphere_datastore="datastore"
openshift_cloudprovider_vsphere_folder="folder"

# Service catalog
openshift_hosted_etcd_storage_kind=dynamic
openshift_hosted_etcd_storage_volume_name=etcd-vol
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
```

```
openshift_hosted_etcd_storage_volume_size=1G
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}

openshift_master_ldap_ca_file=/home/cloud-user/mycert.crt
openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true', 'login': 'true', 'kind':
'LDAPPasswordIdentityProvider', 'attributes': {'id': ['dn'], 'email': ['mail'], 'name': ['cn'],
'preferredUsername': ['uid']}, 'bindDN': 'uid=admin,cn=users,cn=accounts,dc=example,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt', 'insecure': 'false', 'url':
'ldap://ldap.example.com/cn=users,cn=accounts,dc=example,dc=com?uid?sub?(memberOf=cn=ose-
user,cn=groups,cn=accounts,dc=openshift,dc=com)'}]

# Setup vsphere registry storage
openshift_hosted_registry_storage_kind=vsphere
openshift_hosted_registry_storage_access_modes=['ReadWriteOnce']
openshift_hosted_registry_storage_annotations=['volume.beta.kubernetes.io/storage-provisioner:
kubernetes.io/vsphere-volume']
openshift_hosted_registry_replicas=1

openshift_hosted_router_replicas=3
openshift_master_cluster_method=native
openshift_node_local_quota_per_fsgroup=512Mi

default_subdomain=example.com
openshift_master_cluster_hostname=openshift.example.com
openshift_master_cluster_public_hostname=openshift.example.com
openshift_master_default_subdomain=apps.example.com

os_sdn_network_plugin_name='redhat/openshift-ovs-networkpolicy'
osm_use_cockpit=true

# Red Hat subscription name and password
rhsub_user=username
rhsub_pass=password
rhsub_pool=8a85f9815e9b371b015e9b501d081d4b

# metrics
openshift_metrics_install_metrics=true
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=25Gi

# logging
openshift_logging_install_logging=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_es_pvc_size=30Gi
openshift_logging_elasticsearch_storage_type=pvc
openshift_logging_es_cluster_size=1
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_fluentd_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_storage_kind=dynamic

#registry
openshift_public_hostname=openshift.example.com

[ansible]
```

```
localhost

[masters]
master-0.example.com vm_name=master-0 ipv4addr=10.x.y.103
master-1.example.com vm_name=master-1 ipv4addr=10.x.y.104
master-2.example.com vm_name=master-2 ipv4addr=10.x.y.105

[infras]
infra-0.example.com vm_name=infra-0 ipv4addr=10.x.y.100
infra-1.example.com vm_name=infra-1 ipv4addr=10.x.y.101
infra-2.example.com vm_name=infra-2 ipv4addr=10.x.y.102

[apps]
app-0.example.com vm_name=app-0 ipv4addr=10.x.y.106
app-1.example.com vm_name=app-1 ipv4addr=10.x.y.107
app-2.example.com vm_name=app-2 ipv4addr=10.x.y.108

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

[lb]
haproxy-0.example.com vm_name=haproxy-0 ipv4addr=10.x.y.200

[nodes]
master-0.example.com openshift_node_group_name="node-config-master"
openshift_schedulable=true
master-1.example.com openshift_node_group_name="node-config-master"
openshift_schedulable=true
master-2.example.com openshift_node_group_name="node-config-master"
openshift_schedulable=true
infra-0.example.com openshift_node_group_name="node-config-infra"
infra-1.example.com openshift_node_group_name="node-config-infra"
infra-2.example.com openshift_node_group_name="node-config-infra"
app-0.example.com openshift_node_group_name="node-config-compute"
app-1.example.com openshift_node_group_name="node-config-compute"
app-2.example.com openshift_node_group_name="node-config-compute"
```

1 2 如果您使用需要身份验证的容器 registry，如默认容器镜像 registry，请指定该帐户的凭证。请参阅访问并配置 Red Hat Registry 。



注意

红帽不正式支持部署 vSphere 虚拟机环境，但可以配置它。

25.2.2. 选项 2：为 vSphere 手动配置 OpenShift Container Platform

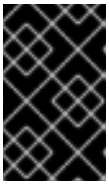
25.2.2.1. 为 vSphere 手动配置 master 主机

在所有 master 主机上执行以下操作。

流程

1. 在所有 master 上编辑 */etc/origin/master/master-config.yaml* 中的 master 配置文件，并更新 **apiServerArguments** 和 **controllerArguments** 部分的内容：

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "vsphere"
    cloud-config:
      - "/etc/origin/cloudprovider/vsphere.conf"
  controllerArguments:
    cloud-provider:
      - "vsphere"
    cloud-config:
      - "/etc/origin/cloudprovider/vsphere.conf"
```



重要

在触发容器化安装时，只有 */etc/origin* 和 */var/lib/origin* 目录被挂载到 master 和节点容器中。因此，*master-config.yaml* 必须位于 */etc/origin/master* 中，而不是 */etc/*。

2. 当使用 Ansible 为 vSphere 配置 OpenShift Container Platform 时，*/etc/origin/cloudprovider/vsphere.conf* 文件会被自动创建。由于您要手动为 vSphere 配置 OpenShift Container Platform，所以您必须创建该文件。在创建该文件前，请确定您想多个 vCenter 区域。
集群安装过程默认配置单区或单个 vCenter。但是，在不同区的 vSphere 中部署 OpenShift Container Platform 有助于避免出现单一故障点，但跨区创建共享存储需要。如果 OpenShift Container Platform 节点主机在区域 "A" 中停机，pod 应该移到 zone "B"。如需更多信息，请参阅 Kubernetes 文档中的 Kubernetes 文档中的在多个区中运行。

   - 要配置单个 vCenter 服务器，在 */etc/origin/cloudprovider/vsphere.conf* 文件中使用以下格式：

```
[Global] 1
        user = "myusername" 2
        password = "mypassword" 3
        port = "443" 4
        insecure-flag = "1" 5
        datacenters = "mydatacenter" 6

[VirtualCenter "10.10.0.2"] 7
        user = "myvCenterusername"
        password = "password"

[Workspace] 8
        server = "10.10.0.2" 9
        datacenter = "mydatacenter"
        folder = "path/to/vms" 10
        default-datastore = "shared-datastore" 11
        resourcepool-path = "myresourcepoolpath" 12

[Disk]
```

```
        scsicontrollertype = pvscsi 13

[Network]
        public-network = "VM Network" 14
```

1. **[Global]** 部分中设置的任何属性都用于指定的 vcenters，除非单独 **[VirtualCenter]** 部分中的设置覆盖。
2. vSphere 云供应商的 vCenter 用户名。
3. 指定用户的 vCenter 密码。
4. 可选。vCenter 服务器的端口号。默认为端口 **443**。
5. 如果 vCenter 使用自签名证书，则设置为 **1**。
6. 部署节点虚拟机的数据中心的名称。
7. 覆盖这个虚拟中心的特定 **[Global]** 属性。可能的设置扫描为 **[Port]**, **[user]**, **[insecure-flag]**, **[datacenters]**.所有未指定设置都会从 **[Global]** 部分中拉取。
8. 设置用于各种 vSphere Cloud Provider 功能的任何属性。例如，动态置备、基于存储配置文件的卷置备等。
9. vCenter 服务器的 IP 地址或 FQDN。
10. 节点虚拟机虚拟机目录的路径。
11. 使用存储类或动态置备，设置为用于置备卷的数据存储名称。在 OpenShift Container Platform 3.9 之前，如果数据存储位于存储目录中，或者是数据存储集群的成员，则需要完整路径。
12. 可选。设置为资源池的路径，其中必须创建 dummy 虚拟机用于 Storage Profile 基于卷置备。
13. SCSI 控制器的类型将作为.
14. 设置为 vSphere 的网络端口组以访问节点，该节点默认称为 VM Network。这是在 Kubernetes 中注册的节点主机的 ExternalIP。

- 要配置多个 vCenter 服务器，在 */etc/origin/cloudprovider/vsphere.conf* 文件中使用以下格式：

```
[Global] 1
        user = "myusername" 2
        password = "mypassword" 3
        port = "443" 4
        insecure-flag = "1" 5
        datacenters = "us-east, us-west" 6

[VirtualCenter "10.10.0.2"] 7
        user = "myvCenterusername"
        password = "password"

[VirtualCenter "10.10.0.3"]
```

```
        port = "448"
        insecure-flag = "0"

[Workspace] 8
        server = "10.10.0.2" 9
        datacenter = "mydatacenter"
        folder = "path/to/vms" 10
        default-datastore = "shared-datastore" 11
        resourcepool-path = "myresourcepoolpath" 12

[Disk]
        scsicontrollertype = pvscsi 13

[Network]
        public-network = "VM Network" 14
```

1. **[Global]** 部分中设置的任何属性都用于指定的 vcenters，除非单独 **[VirtualCenter]** 部分中的设置覆盖。
2. vSphere 云供应商的 vCenter 用户名。
3. 指定用户的 vCenter 密码。
4. 可选。vCenter 服务器的端口号。默认为端口 **443**。
5. 如果 vCenter 使用自签名证书，则设置为 **1**。
6. 部署节点虚拟机的数据中心的名称。
7. 覆盖这个虚拟中心的特定 **[Global]** 属性。可能的设置扫描为 **[Port]**, **[user]**, **[insecure-flag]**, **[datacenters]**.所有未指定设置都会从 **[Global]** 部分中拉取。
8. 设置用于各种 vSphere Cloud Provider 功能的任何属性。例如，动态置备、基于存储配置文件的卷置备等。
9. 云供应商通信的 vCenter 服务器的 IP 地址或 FQDN。
10. 节点虚拟机虚拟机目录的路径。
11. 使用存储类或动态置备，设置为用于置备卷的数据存储名称。在 OpenShift Container Platform 3.9 之前，如果数据存储位于存储目录中，或者是数据存储集群的成员，则需要完整路径。
12. 可选。设置为资源池的路径，其中必须创建 dummy 虚拟机用于 Storage Profile 基于卷置备。
13. SCSI 控制器的类型将作为.
14. 设置为 vSphere 的网络端口组以访问节点，该节点默认称为 VM Network。这是在 Kubernetes 中注册的节点主机的 ExternalIP。

3. 重启 OpenShift Container Platform 主机服务：

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

25.2.2.2. 为 vSphere 手动配置节点主机

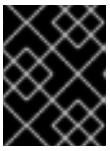
在所有节点主机上执行以下内容。

流程

为 vSphere 配置 OpenShift Container Platform 节点：

1. 编辑 适当的节点配置映射 并更新 **kubeletArguments** 部分的内容：

```
kubeletArguments:
  cloud-provider:
    - "vsphere"
  cloud-config:
    - "/etc/origin/cloudprovider/vsphere.conf"
```



重要

nodeName 必须与 vSphere 中的虚拟机名称匹配，才能使云供应商集成正常工作。名称也必须与 RFC1123 兼容。

2. 重启所有节点上的 OpenShift Container Platform 服务。

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

25.2.2.3. 应用配置更改

在所有 master 和节点主机上启动或重启 OpenShift Container Platform 服务以应用您的配置更改，请参阅重启 OpenShift Container Platform 服务 ：

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

注意

Kubernetes 架构需要来自云提供商的可靠端点。当云提供商停机时，kubelet 会防止 OpenShift Container Platform 重启。如果底层云供应商端点不可靠，请不要安装使用云供应商集成的集群。如在裸机环境中一样安装集群。不建议在已安装的集群中打开或关闭云提供商集成。但是，如果该情境不可避免，请完成以下过程。

从不使用云供应商切换到使用云提供商会产生错误消息。添加云供应商会尝试删除节点，因为从其切换的节点使用 **hostname** 作为 **externalID** （当没有云供应商使用时）使用云供应商的 **instance-id** （由云提供商指定）。要解决这个问题：

1. 以集群管理员身份登录到 CLI。

2. 检查和备份现有节点标签：

-

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?=Taints)'
```

3. 删除节点：

```
$ oc delete node <node_name>
```

4. 在每个节点主机上，重启 OpenShift Container Platform 服务。

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

5. 在每个主机上重新添加回您以前具有的标记。

25.3. 配置 OPENSHIFT CONTAINER PLATFORM 以使用 VSPHERE 存储

OpenShift Container Platform 支持 VMware vSphere 的虚拟机磁盘(VMDK)卷。您可以使用 VMware vSphere 为 OpenShift Container Platform 集群置备持久性存储。我们假设您对 Kubernetes 和 VMware vSphere 已有一定了解。

OpenShift Container Platform 在 vSphere 中创建磁盘，并将磁盘附加到正确的实例。

OpenShift Container Platform 持久性卷(PV) 框架允许管理员置备带有持久性存储的集群，并为用户提供在不了解底层基础架构的情况下请求这些资源的方法。vSphere VMDK 卷 可以动态置备。

PV 不绑定到单个项目或命名空间，它们可以在 OpenShift Container Platform 集群间共享。但是，PV 声明 是特定于项目或命名空间的，用户可请求它。

重要

存储的高可用性功能由底层的存储架构提供。

先决条件

在使用 vSphere 创建 PV 前，请确保 OpenShift Container Platform 集群满足以下要求：

- OpenShift Container Platform 必须首先 为 vSphere 配置。
- 基础架构中的每个节点主机必须与 vSphere 虚拟机名称匹配。
- 每个节点主机必须位于同一资源组中。

25.3.1. 动态置备 VMware vSphere 卷

动态置备 VMware vSphere 卷是首选的置备方法。

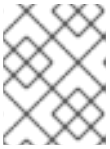
1. 如果您在置备集群时没有指定 Ansible 清单文件中的 **openshift_cloudprovider_kind=vsphere** 和 **openshift_vsphere_*** 变量，您必须手动创建以下 **StorageClass** 来使用 **vsphere-volume** 置备程序：

```
$ oc get --export storageclass vsphere-standard -o yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: "vsphere-standard" (1)
provisioner: kubernetes.io/vsphere-volume (2)
```

```
parameters:
  diskformat: thin (3)
  datastore: "YourvSphereDatastoreName" (4)
reclaimPolicy: Delete
```

1. StorageClass 的名称。
2. 存储置备程序类型。指定 **vsphere-volume**。
3. 磁盘类型。指定零个 **或 精简**。
4. 创建磁盘的源数据存储。

2. 在使用上一步中显示的 StorageClass 请求 PV 后，OpenShift Container Platform 会在 vSphere 基础架构中创建 VMDK 磁盘。要验证磁盘是否已创建，请在 vSphere 中使用 Datastore 浏览器。



注意

vSphere-volume 磁盘是 **ReadWriteOnce** 访问模式，这意味着该卷可以被单一节点以读写模式挂载。如需更多信息，请参阅架构指南中的访问模式部分。

25.3.2. 静态置备 VMware vSphere 卷

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。确保为 vSphere 配置 OpenShift Container Platform 后，OpenShift Container Platform 和 vSphere 都需要一个虚拟机文件夹路径、文件系统类型和 **PersistentVolume** API。

25.3.2.1. 创建 PersistentVolume

1. 定义 PV 对象定义，如 ***vsphere-pv.yaml*** ：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 (1)
spec:
  capacity:
    storage: 2Gi (2)
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  vsphereVolume: (3)
    volumePath: "[datastore1] volumes/myDisk" (4)
    fsType: ext4 (5)
```

1. 卷的名称。这必须是如何通过 PV 声明或从 pod 识别它。
2. 为这个卷分配的存储量。
3. 使用的卷类型。这个示例使用 **vsphereVolume**。此标签用于将 vSphere VMDK 卷挂载到 Pod 中。卸载卷时会保留卷内容。卷类型支持 VMFS 和 VSAN 数据存储。
4. 要使用的现有 VMDK 卷。在卷定义中，数据存储名称必须放在方括号([])中，如下所示。

5 要挂载的文件系统类型。例如： **ext** 4、**xfs** 或者其它文件系统。

重要

在格式化并置备卷后更改 **fsType** 参数的值可能会导致数据丢失和 pod 失败。

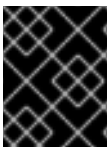
2. 创建 PV：

```
$ oc create -f vsphere-pv.yaml
  persistentvolume "pv0001" created
```

3. 确定创建了 PV：

```
$ oc get pv
NAME    LABELS  CAPACITY  ACCESSMODES  STATUS     CLAIM   REASON  AGE
pv0001  <none>  2Gi       RWO          Available          2s
```

现在，您可以使用 PV 声明来请求存储，该声明现在可以使用 PV。



重要

PV 声明仅存在于用户的命名空间中，且只能被同一命名空间中的 pod 引用。尝试从不同命名空间中访问 PV 会导致 pod 失败。

25.3.2.2. 格式化 VMware vSphere 卷

在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查卷是否包含由 PV 定义中 **fsType** 参数指定的文件系统。如果没有使用文件系统格式化该设备，该设备中的所有数据都会被删除，并使用指定的文件系统自动格式化该设备。

因为 OpenShift Container Platform 在首次使用卷前会进行格式化，所以您可以使用未格式化的 vSphere 卷作为 PV。

25.4. 为 VSPHERE 配置 OPENSHIFT CONTAINER PLATFORM REGISTRY

25.4.1. 使用 Ansible 为 vSphere 配置 OpenShift Container Platform registry

流程

将 registry 的 Ansible 清单配置为使用 vSphere 卷：

```
[OSEv3:vars]
# vSphere Provider Configuration
openshift_hosted_registry_storage_kind=vsphere 1
openshift_hosted_registry_storage_access_modes=['ReadWriteOnce'] 2
openshift_hosted_registry_storage_annotations=['volume.beta.kubernetes.io/storage-provisioner:
kubernetes.io/vsphere-volume'] 3
openshift_hosted_registry_replicas=1 4
```

1. 存储类型。
2. vSphere 卷只支持 **RWO**。
3. 卷的注解。
4. 用于配置的副本数。

注意

上面配置文件中的括号是必需的。

25.4.2. 为 OpenShift Container Platform registry 动态置备存储

要使用 vSphere 卷存储，请编辑 registry 的配置文件并挂载到 registry pod。

流程

1. 从 vSphere 卷创建新配置文件：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: vsphere-registry-storage
  annotations:
    volume.beta.kubernetes.io/storage-class: vsphere-standard
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 30Gi
```

2. 在 OpenShift Container Platform 中创建该文件：

```
$ oc create -f pvc-registry.yaml
```

3. 更新卷配置以使用新的 PVC：

```
$ oc set volume dc docker-registry --add --name=registry-storage -t \
pvc --claim-name=vsphere-registry-storage --overwrite
```

4. 重新部署 registry 以读取更新的配置：

```
$ oc rollout latest docker-registry -n default
```

5. 验证卷是否已分配：

```
$ oc set volume dc docker-registry -n default
```

25.4.3. 为 OpenShift Container Platform registry 手动置备存储

运行以下命令手动创建存储，该存储用于在 **StorageClass** 不可用时为 registry 创建存储。

```
# VMFS
cd /vmfs/volumes/datastore1/
mkdir kubevols # Not needed but good hygiene

# VSAN
cd /vmfs/volumes/vsanDatastore/
/usr/lib/vmware/osfs/bin/osfs-mkdir kubevols # Needed

cd kubevols

vmkfstools -c 25G registry.vmdk
```

25.4.4. 关于 Red Hat OpenShift Container Storage

Red Hat OpenShift Container Storage(RHOCS)是 OpenShift Container Platform 内部或混合云中与无关的持久性存储供应商。作为红帽存储解决方案，RHOCS 与 OpenShift Container Platform 完全集成，用于部署、管理和监控，无论它是否安装在 OpenShift Container Platform（融合）或与 OpenShift Container Platform（独立）。OpenShift Container Storage 不仅限于单个可用区或节点，这使得它可能会停机。您可以在 RHOCS 3.11 部署指南中找到使用 RHOCS 的完整说明。

25.5. 持久性卷备份

OpenShift Container Platform 将新卷置备为 **独立持久性磁盘**，以便在集群中的任何节点上自由附加和分离卷。因此，无法备份使用快照的卷。

创建 PV 的备份：

1. 使用 PV 停止应用程序。
2. 克隆持久磁盘。
3. 重新启动应用程序。
4. 创建克隆的磁盘的备份。
5. 删除克隆的磁盘。

第 26 章 配置本地卷

26.1. 概述

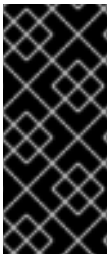
OpenShift Container Platform 可以配置为访问应用程序数据的本地卷。

本地卷是代表本地挂载的文件系统（包括原始块设备）的持久性卷(PV)。原始设备提供了一个更直接路由到物理设备的直接路由，并允许应用程序对 I/O 操作到该物理设备的时间进行更多控制。这使得原始设备更适合于复杂的应用程序，如通常执行自己的缓存的数据库管理系统。本地卷具有一些独一无二的功能。所有使用本地卷 PV 的 pod 都会调度到挂载本地卷的节点。

另外，本地卷包括一个置备程序，它可自动为本地挂载的设备创建 PV。此置备程序目前只扫描预先配置的目录。此置备程序无法动态置备卷，但这个功能可能会在以后的版本中实现。

本地卷置备程序允许在 OpenShift Container Platform 中使用本地存储并提供支持：

- 卷
- PV



重要

本地卷只是一个技术预览功能。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。如需红帽技术预览功能支持范围的更多信息，请参阅 https://access.redhat.com/support/offerings/techpreview/。

26.2. 挂载本地卷

注意

所有本地卷都必须手动挂载，然后 OpenShift Container Platform 可以被 OpenShift Container Platform 使用为 PV。

挂载本地卷：

1. 将所有卷挂载到 */mnt/local-storage/<storage-class-name>/<volume>* 路径。管理员需要使用任何方法（如磁盘分区或 LVM）创建本地设备，并在这些设备中创建适当的文件系统，并使用脚本或 **/etc/fstab** 条目挂载这些设备，例如：

```
# device name   # mount point              # FS    # options # extra
/dev/sdb1       /mnt/local-storage/ssd/disk1 ext4    defaults 1 2
/dev/sdb2       /mnt/local-storage/ssd/disk2 ext4    defaults 1 2
/dev/sdb3       /mnt/local-storage/ssd/disk3 ext4    defaults 1 2
/dev/sdc1       /mnt/local-storage/hdd/disk1 ext4    defaults 1 2
/dev/sdc2       /mnt/local-storage/hdd/disk2 ext4    defaults 1 2
```

2. 使所有卷可供容器内运行的进程访问。您可以更改挂载的文件系统的标签以允许这样，例如：

```
---
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-storage/
---
```

26.3. 配置本地置备程序

OpenShift Container Platform 依赖于外部置备程序来为本地设备创建 PV，并在不使用它时清除 PV 以启用重复使用。

注意

- 本地卷置备程序与大多数置备程序不同，且不支持动态置备。
- 本地卷置备程序要求管理员在每个节点上预配置本地卷并将其挂载到发现目录中。然后，置备程序通过为每个卷创建并清理 PV 来管理卷。

配置本地置备程序：

1. 使用 ConfigMap 配置外部置备程序，以将目录与存储类相关。此配置必须在部署置备程序前创建，例如：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-volume-config
data:
    storageClassMap: |
        local-ssd: (1)
            hostDir:  /mnt/local-storage/ssd (2)
            mountDir: /mnt/local-storage/ssd (3)
        local-hdd:
            hostDir: /mnt/local-storage/hdd
            mountDir: /mnt/local-storage/hdd
```

(1) 存储类的名称。

(2) 到主机上的目录的路径。它必须是 */mnt/local-storage* 的子目录。

(3) 指向 provisioner pod 中的目录的路径。我们建议您使用与在主机上相同的目录结构，本例中可以省略 **mountDir**。

2. *（可选）*为本地卷置备程序及其配置创建一个独立命名空间，例如：**oc new-project local-storage**。

使用这个配置，置备程序会创建：

- 一个带有存储类 **本地的** PV，每个子目录都挂载到 */mnt/local-storage/ssd* 目录中
- 一个带有存储类 **local-hdd** 的 PV，用于挂载在 */mnt/local-storage/hdd* 目录中的每个子目录

26.4. 部署本地置备程序

注意

在开始置备程序前，挂载所有本地设备并使用存储类及其目录创建 ConfigMap。

部署本地置备程序：

1. 从 *local-storage-provisioner-template.yaml* 文件安装本地置备程序。

2. 创建一个服务帐户，允许以 root 用户身份运行 pod，使用 hostPath 卷，并使用任何 SELinux 上下文监控、管理和清理本地卷：

```
$ oc create serviceaccount local-storage-admin
$ oc adm policy add-scc-to-user privileged -z local-storage-admin
```

要允许 provisioner pod 删除任何 pod 创建的本地卷中的内容，需要 root 权限和任何 SELinux 上下文。hostPath 需要访问主机上的 */mnt/local-storage* 路径。

3. 安装模板：

```
$ oc create -f https://raw.githubusercontent.com/openshift/origin/release-
3.11/examples/storage-examples/local-examples/local-storage-provisioner-template.yaml
```

4. 通过为 **CONFIGMAP**、**SERVICE_ACCOUNT**、**NAMESPACE** 和 **PROVISIONER_IMAGE** 参数指定值来实例化模板：

```
$ oc new-app -p CONFIGMAP=local-volume-config \
  -p SERVICE_ACCOUNT=local-storage-admin \
  -p NAMESPACE=local-storage \
  -p PROVISIONER_IMAGE=registry.redhat.io/openshift3/local-storage-provisioner:v3.11 \
1
  local-storage-provisioner
```

1 提供 OpenShift Container Platform 版本号，如 **v3.11**。

5. 添加所需的存储类：

```
$ oc create -f ./storage-class-ssd.yaml
$ oc create -f ./storage-class-hdd.yaml
```

例如：

storage-class-ssd.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: local-ssd
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

storage-class-hdd.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: local-hdd
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

其他可配置选项请查看 本地存储置备程序模板。此模板创建在每个节点上运行 pod 的 DaemonSet。pod 监视 ConfigMap 中指定的目录，并为它们自动创建 PV。

置备程序以 root 权限运行，因为它会在 PV 被释放时从修改的目录中删除所有数据。

26.5. 添加新设备

添加新设备是半自动的。置备程序会定期检查配置目录中的新挂载。管理员必须创建新的子目录，挂载设备并允许 pod 通过应用 SELinux 标签来使用该设备，例如：

```
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-storage/
```

重要

省略这些步骤可能会导致创建错误的 PV。

26.6. 配置原始块设备

可以使用本地卷置备程序静态置备原始块设备。此功能默认为禁用，需要其他配置。

配置原始块设备：

1. 在所有 master 上启用 **BlockVolume** 功能门。在所有 master（默认为*/etc/origin/master/master-config.yaml* ）上编辑或创建 master 配置文件，并在 **apiServerArguments** 和 **controllerArguments** 部分中添加 **BlockVolume=true** ：

```
apiServerArguments:
  feature-gates:
  - BlockVolume=true
...

controllerArguments:
  feature-gates:
  - BlockVolume=true
...
```

2. 通过编辑节点配置 ConfigMap，在所有节点中启用功能门：

```
$ oc edit configmap node-config-compute --namespace openshift-node
$ oc edit configmap node-config-master --namespace openshift-node
$ oc edit configmap node-config-infra --namespace openshift-node
```

3. 确保所有 ConfigMap 都包含 **kubeletArguments** 的功能门数组中的 **BlockVolume=true**，例如：

节点 configmap 功能门设置

```
kubeletArguments:
  feature-gates:
  -
RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true,BlockVolume=true
```

4. 重启 master。节点在配置更改后自动重启。这可能需要几分钟时间。

26.6.1. 准备原始块设备

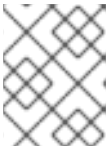
在启动置备程序前，请链接 pod 可用于 */mnt/local-storage/<storage class>* 目录结构的所有原始块设备。例如，使目录 */dev/dm-36* 可用：

1. 在 */mnt/local-storage* 中为设备的存储类创建一个目录：

```
$ mkdir -p /mnt/local-storage/block-devices
```

2. 创建指向该设备的符号链接：

```
$ ln -s /dev/dm-36 dm-uuid-LVM-1234
```



注意

为了避免可能的名称冲突，请将相同的名称用于符号链接，以及 */dev/disk/by-uuid* 或 */dev/disk/by-id* 目录的链接。

3. 创建或更新配置置备程序的 ConfigMap：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-volume-config
data:
    storageClassMap: |
        block-devices: 1
            hostDir:  /mnt/local-storage/block-devices 2
            mountDir: /mnt/local-storage/block-devices 3
```

1. 存储类的名称。
2. 到主机上的目录的路径。它必须是 */mnt/local-storage* 的子目录。
3. 指向 provisioner pod 中的目录的路径。如果使用主机使用的目录结构，建议使用它，省略 **mountDir** 参数。

4. 更改设备的 **SELinux** 标签和 */mnt/local-storage/* ：

```
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-storage/
$ chcon unconfined_u:object_r:svirt_sandbox_file_t:s0 /dev/dm-36
```

5. 为原始块设备创建存储类：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: block-devices
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

块设备 */dev/dm-36* 现在可供置备程序使用，并置备为 PV。

26.6.2. 部署原始块设备置备程序

为原始块设备部署置备程序与在本地卷中部署置备程序类似。有两个区别：

1. 置备程序必须在特权容器中运行。
2. 置备程序必须能够从主机访问 */dev* 文件系统。

为原始块设备部署置备程序：

1. 从 *local-storage-provisioner-template.yaml* 文件下载模板。
2. 编辑模板：
   a. 将容器规格的 **securityContext** 的 **privileged** 属性设置为 **true** ：

```
...
  containers:
...
    name: provisioner
...
      securityContext:
        privileged: true
...
```

   b. 使用 **hostPath** 将主机 */dev/* 文件系统挂载到容器：

```
...
  containers:
...
    name: provisioner
...
    volumeMounts:
    - mountPath: /dev
      name: dev
...
  volumes:
    - hostPath:
        path: /dev
      name: dev
...
```

3. 从修改后的 YAML 文件创建模板：

```
$ oc create -f local-storage-provisioner-template.yaml
```

4. 启动置备程序：

```
$ oc new-app -p CONFIGMAP=local-volume-config \
  -p SERVICE_ACCOUNT=local-storage-admin \
  -p NAMESPACE=local-storage \
  -p
  PROVISIONER_IMAGE=registry.redhat.io/openshift3/local-storage-provisioner:v3.11 \
  local-storage-provisioner
```

26.6.3. 使用原始块设备持久性卷

要使用 pod 中的原始块设备，创建一个带有 **volumeMode:** 设置为 **Block** 的持久性卷声明 (PVC)，**storageClassName** 设置为 **block-devices**，例如：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  storageClassName: block-devices
  accessModes:
    - ReadWriteOnce
  volumeMode: Block
  resources:
    requests:
      storage: 1Gi
```

使用原始块设备 PVC 的 Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox-test
  labels:
    name: busybox-test
spec:
  restartPolicy: Never
  containers:
    - resources:
        limits :
          cpu: 0.5
      image: gcr.io/google_containers/busybox
      command:
        - "/bin/sh"
        - "-c"
        - "while true; do date; sleep 1; done"
      name: busybox
      volumeDevices:
        - name: vol
          devicePath: /dev/xvda
  volumes:
    - name: vol
      persistentVolumeClaim:
        claimName: block-pvc
```

注意

卷没有挂载到 pod 中，而是作为 */dev/xvda* 原始块设备公开。

第 27 章 配置持久性存储

27.1. 概述

Kubernetes 持久性卷 框架允许您使用环境中可用的联网存储来置备带有持久性存储的 OpenShift Container Platform 集群。这可在根据应用程序需要完成初始 OpenShift Container Platform 安装后完成，为用户提供在不了解底层基础架构的情况下请求这些资源的方法。

这些主题演示了如何使用以下支持的卷插件在 OpenShift Container Platform 中配置持久性卷：

- NFS
- GlusterFS
- OpenStack Cinder
- Ceph RBD
- AWS Elastic Block Store (EBS)
- GCE Persistent Disk
- iSCSI
- Fibre Channel
- Azure Disk
- Azure File
- FlexVolume
- VMware vSphere
- Container Storage Interface (CSI)
- 动态置备和创建存储类
- 卷安全性
- selector-Label Volume Binding

27.2. 使用 NFS 的持久性存储

27.2.1. 概述

OpenShift Container Platform 集群可以使用 NFS 来 置备持久性存储。持久性卷 (PV) 和持久性卷声明 (PVC) 提供了在项目间共享卷的方法。虽然 PV 定义中包含的与 NFS 相关的信息也可以直接在 pod 定义中定义，但这样做不会将卷创建为一个特定的集群资源，从而可能会导致卷冲突。

本节涵盖使用 NFS 持久性存储类型的具体内容。对 OpenShift Container Platform 和 NFS 有一定的了解会有所帮助。如需了解 OpenShift Container Platform 持久性卷(PV)框架的详细信息，请参阅持久性存储概念主题。

27.2.2. 置备

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。要置备 NFS 卷，则需要一个 NFS 服务器和导出路径列表。

您必须首先为 PV 创建对象定义：

例 27.1. 使用 NFS 的 PV 对象定义

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 5Gi 2
  accessModes:
  - ReadWriteOnce 3
  nfs: 4
    path: /tmp 5
    server: 172.17.0.2 6
  persistentVolumeReclaimPolicy: Retain 7
```

1. 卷的名称。这是各个 **oc <command> pod** 命令中的 PV 标识。
2. 为这个卷分配的存储量。
3. 虽然这看上去象是设置对卷的访问控制，但它实际上被用作标签并用来将 PVC 与 PV 匹配。目前，没有根据 **accessModes** 强制执行访问规则。
4. 使用的卷类型，在这个示例里是 **nfs** 插件。
5. NFS 服务器导出的路径。
6. NFS 服务器的主机名或 IP 地址。
7. PV 的 reclaim 策略。它决定了从声明中释放卷时会发生什么。请参阅回收资源。

注意

每个 NFS 卷都必须由集群中的所有可调度节点挂载。

将定义保存到文件中，如 ***nfs-pv.yaml*** 并创建 PV：

```
$ oc create -f nfs-pv.yaml
persistentvolume "pv0001" created
```

确定创建了 PV：

```
# oc get pv
NAME                LABELS    CAPACITY    ACCESSMODES   STATUS      CLAIM     REASON
AGE
pv0001              <none>    5368709120  RWO           Available               31s
```

下一步是创建一个 PVC，它绑定到新 PV：

例 27.2. PVC 对象定义

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
    - ReadWriteOnce 1
  resources:
    requests:
      storage: 1Gi 2
```

1 如前面对 PV 提供的一样， **accessModes** 不会强制实现安全控制，而是作为标签来把一个 PV 和一个 PVC 进行匹配。

2 这个声明会寻找带有 **1Gi** 或更高容量的 PV。

将定义保存到文件中，如 ***nfs-claim.yaml*** 并创建 PVC：

```
# oc create -f nfs-claim.yaml
```

27.2.3. 强制磁盘配额

使用磁盘分区强制磁盘配额和大小限制。每个分区都可以有自己的导出。每个导出都是一个 PV。OpenShift Container Platform 会保证每个 PV 都使用不同的名称，但 NFS 卷服务器和路径的唯一性是由管理员实现的。

以这种方式强制配额可让开发人员以特定数量（例如 10Gi）请求持久性存储，并与相等或更大容量的卷匹配。

27.2.4. NFS 卷安全性

这部分论述了 NFS 卷安全性，其中包括匹配的权限和 SELinux 考虑。用户需要了解 POSIX 权限、进程 UID 、supplemental 组和 SELinux 的基本知识。

注意

在实施 NFS 卷前，请查看完整的卷安全性 主题。

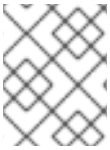
开发人员在其 Pod 定义的 **volumes** 部分（按名称或 NFS 卷插件）中引用 NFS 存储来请求 NFS 存储。

NFS 服务器中的 ***/etc/exports*** 文件包含可访问的 NFS 目录。目标 NFS 目录有 POSIX 拥有者和组群 ID。OpenShift Container Platform NFS 插件使用相同的 POSIX 所有者权限及在导出的 NFS 目录中找到的权限挂载容器的 NFS 目录。然而，容器实际运行时所使用的 UID 与 NFS 挂载的所有者的 UID 不同。这是所需的行为。

例如，目标 NFS 目录在 NFS 服务器中，如下所示：

```
# ls -lZ /opt/nfs -d
drwxrws---. nfsnobody 5555 unconfined_u:object_r:usr_t:s0   /opt/nfs

# id nfsnobody
uid=65534(nfsnobody) gid=65534(nfsnobody) groups=65534(nfsnobody)
```

然后，容器必须与 SELinux 标签匹配，并使用 UID **65534**（**nfsnobody** 的所有者）或其 supplemental 组的 **5555** 运行，才能访问该目录。

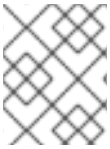


注意

所有者 ID 65534 只是一个示例。虽然 NFS 的 **root_squash** 将 **root** (0)映射到 **nfsnobody** (65534))，但 NFS 导出可以具有任意所有者 ID。NFS 导出的所有者不需要 65534。

27.2.4.1. 组 ID

处理 NFS 访问（假设这不是更改 NFS 导出权限的选项）是使用补充组。OpenShift Container Platform 中的附件组的功能是用于共享存储（NFS 是一个共享存储）。与之相反，块存储（如 Ceph RBD 或 iSCSI）使用 **fsGroup** SCC 策略和在 pod 的 **securityContext** 中的 **fsGroup** 值。



注意

通常情况下，最好使用附件组群 ID 而不是用户 ID 来获得对持久性存储的访问。在完整的卷安全主题中会进一步阐述 补充组。

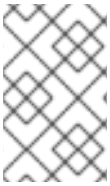
因为上面显示的示例目标 NFS 目录上的组 ID 是 5555，所以 pod 可以使用 pod 级 **securityContext** 定义下的 **supplementalGroups** 来定义组 ID。例如：

```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    supplementalGroups: [5555] 2
```

1. **securityContext** 必须在 pod 一级定义，而不是在某个特定容器中定义。
2. 为 pod 定义的 GID 数组。在这种情况下，阵列中有一个元素；额外的 GID 会用逗号分开。

假设没有可能满足 Pod 要求的自定义 SCC，Pod 可能与**受限** SCC 匹配。此 SCC 将 **supplementalGroups** 策略设置为 **RunAsAny**，表示任何提供的组 ID 都被接受而无需范围检查。

因此，上面的 pod 可以通过，并被启动。但是，如果需要进行组 ID 范围检查，一个自定义 SCC，如 Pod 安全性和自定义 SCC 中所述，这是首选的解决方案。可创建一个定义了最小和最大组群 ID 的自定义 SCC，这样就会强制进行组 ID 范围检查，组 ID 5555 将被允许。



注意

要使用自定义 SCC，需要首先将其添加到适当的服务帐户（service account）中。例如，在一个特定项目中使用 **default** 服务账户（除非在 pod 规格中指定了另外一个账户）。详情请参阅 向用户、组或项目添加 SCC。

27.2.4.2. 用户 ID

用户 ID 可以在容器镜像或容器集定义中定义。完整 卷安全 主题涵盖基于用户 ID 控制存储访问，应在设置 NFS 持久性存储前读取。

注意

与使用用户 ID 相比，通常会首选使用 supplemental 组 ID 来获得对持久性存储的访问。

在上面显示的目标 NFS 目录示例中，容器需要将其 UID 设定为 65534，忽略组 ID。因此可以把以下内容添加到 Pod 定义中：

```
spec:
  containers: 1
  - name:
  ...
    securityContext:
      runAsUser: 65534 2
```

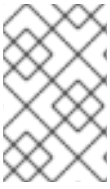
1 Pods 包括一个特定于每一个容器的 **securityContext** （如此处所示），以及一个适用于 pod 中定义的所有容器的 pod 级别 **securityContext**。

2 65534 是 **nfsnobody** 用户。

假设 **default** 项目和 **restricted** SCC，pod 请求的用户 ID 65534 不被允许，因此 pod 失败。pod 因以下原因失败：

- 它要求 65534 作为其用户 ID。
- 检查 Pod 可用的所有 SCC 都会检查哪个 SCC 允许用户 ID 65534 （实际上，会检查 SCC 的所有策略，但这里着重关注用户 ID）。
- 因为所有可用的 SCC 都使用 **MustRunAsRange** 作为其 **runAsUser** 策略，所以需要 UID 范围检查。
- 65534 不包含在 SCC 或项目的用户 ID 范围内。

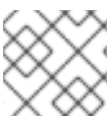
一般情况下，作为一个最佳实践方案，最好不要修改预定义的 SCC。解决这个问题的首选方法是创建一个自定义 SCC，如完整的 卷安全 主题中所述。可以创建一个自定义 SCC，以便定义最小和最大用户 ID，仍强制进行 UID 范围检查，UID 65534 会被允许。



注意

要使用自定义 SCC，需要首先将其添加到适当的服务帐户（service account）中。例如，在一个特定项目中使用 **default** 服务账户（除非在 pod 规格中指定了另外一个账户）。详情请参阅 向用户、组或项目添加 SCC。

27.2.4.3. SELinux



注意

有关使用 SELinux 控制存储访问的详情，请查看完整的 卷安全性 主题。

默认情况下，SELinux 不允许从 pod 写入远程 NFS 服务器。NFS 卷会被正确挂载，但只读。

要在每个节点中使用 SELinux 强制写入 NFS 卷，请运行：

```
# setsebool -P virt_use_nfs 1
```

上面的 **-P** 选项会使重启之间保持 bool。

virt_use_nfs 布尔值由 ***docker-selinux*** 软件包提供。如果看到一个错误，表示没有定义此 bool，请确保已安装此软件包。

27.2.4.4. 导出设置

为了使任意容器用户都可以读取和写入卷， NFS 服务器中的每个导出的卷都应该满足以下条件：

- 每个导出都必须：

  ```
  /<example_fs> *(rw,root_squash)
  ```

- 必须将防火墙配置为允许到挂载点的流量。
  - 对于 NFSv4，配置默认端口 **2049** （**nfs**）。

    NFSv4

    ```
    # iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
    ```

  - 对于 NFSv3，需要配置三个端口：**2049** (**nfs**), **20048** (**mountd**), 和 **111** (**portmapper**).

    NFSv3

    ```
    # iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
    # iptables -I INPUT 1 -p tcp --dport 20048 -j ACCEPT
    # iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
    ```

- 必须设置 NFS 导出和目录，以便目标 pod 可以对其进行访问。将导出设定为由容器的主 UID 拥有，或使用 **supplementalGroups** 来允许 pod 组进行访问（如上面的与 Group ID 相关的章节所示）。如需了解更多 pod 安全性信息，请参阅完整的卷 安全性主题。

27.2.5. 重新声明资源

NFS 实现了 OpenShift Container Platform **Recyclable** 插件接口。自动进程根据在每个持久性卷上设定的策略处理重新声明的任务。

默认情况下，PV 被设置为 **Retain**。

当一个 PV 的声明被释放（即 PVC 被删除），此 PV 对象不应该被重新使用。反之，应该创建一个新的 PV，其基本的卷详情与原始卷相同。

例如：管理员创建一个名为 **nfs1** 的 PV：

```
apiVersion: v1
kind: PersistentVolume
metadata:
```

```
  name: nfs1
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

用户创建 **PVC1**，它绑定到 **nfs1**。然后用户删除了 **PVC1**，将声明发布到 **nfs1**，这会导致 **nfs1** 被**释放**。如果管理员希望使同一 NFS 共享可用，则应该创建一个具有相同 NFS 服务器详情的新 PV，但有一个不同的 PV 名称：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs2
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

删除原来的 PV。不建议使用相同名称重新创建。尝试手工把一个 PV 的状态从 **Released** 改为 **Available** 会导致错误并可能造成数据丢失。

27.2.6. 自动化

可使用 NFS 置备持久性存储集群：

- 使用磁盘分区强制存储配额。
- 通过将卷限制给具有声明它的项目，以强制实现安全性。
- 为每个 PV 配置 丢弃资源的重新声明。

您可以使用许多方法自动执行上述任务。您可以使用与 OpenShift Container Platform 3.11 发行版本关联的 示例 Ansible playbook 来帮助您入门。

27.2.7. 其他配置和故障排除

根据所使用的 NFS 版本以及配置，可能还需要额外的配置步骤来进行正确的导出和安全映射。以下是一些可能适用的信息：

NFSv4 挂载错误地显示所有文件的所有者为 **nobody:nobody**	• 可以归结至 NFS 上的 ID 映射设置(/etc/idmapd.conf) • 请参考红帽解决方案。

在 NFSv4 上禁用 ID 映射	• 在 NFS 服务器中运行：
	`# echo 'Y' > /sys/module/nfsd/parameters/nfs4_disable_idmapping`
	• 在 NFS 客户端中运行：
	`# echo 'Y' > /sys/module/nfs/parameters/nfs4_disable_idmapping`

27.3. 使用 RED HAT GLUSTER STORAGE 的持久性存储

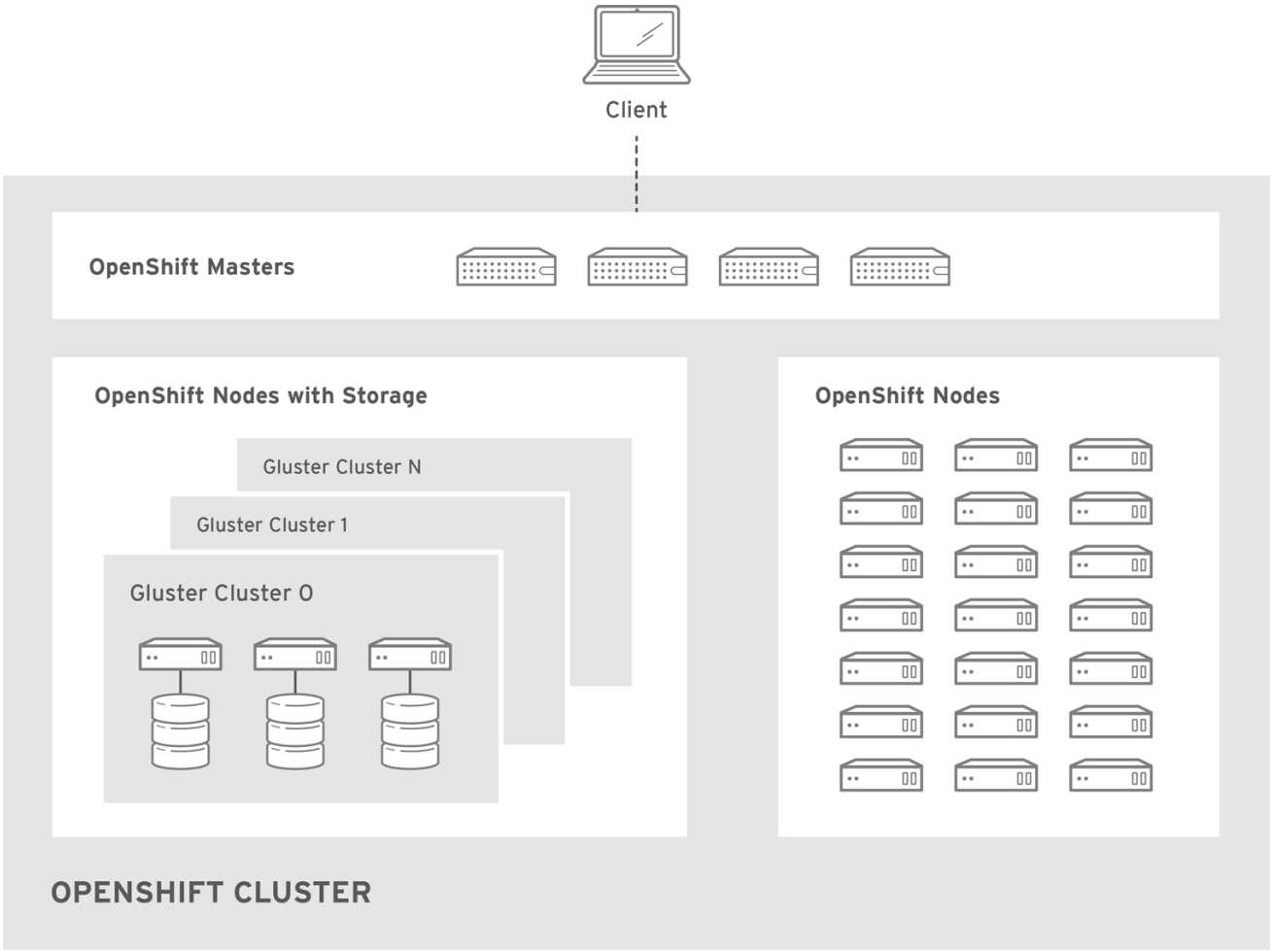
27.3.1. 概述

Red Hat Gluster Storage 可以配置为为 OpenShift Container Platform 提供持久性存储和动态置备。它可用于 OpenShift Container Platform 中容器化（**聚合模式**）和在其自身节点上的非容器化（**独立模式**）。

27.3.1.1. 聚合模式

使用聚合模式时，Red Hat Gluster Storage 直接在 OpenShift Container Platform 节点上运行容器化。这允许调度计算和存储实例，并从同一组硬件上运行。

图 27.1. 架构 - 聚合模式



红帽 Gluster 存储 3.4 中提供了聚合模式。如需了解更多文档，请参阅 OpenShift Container Platform 聚合模式。

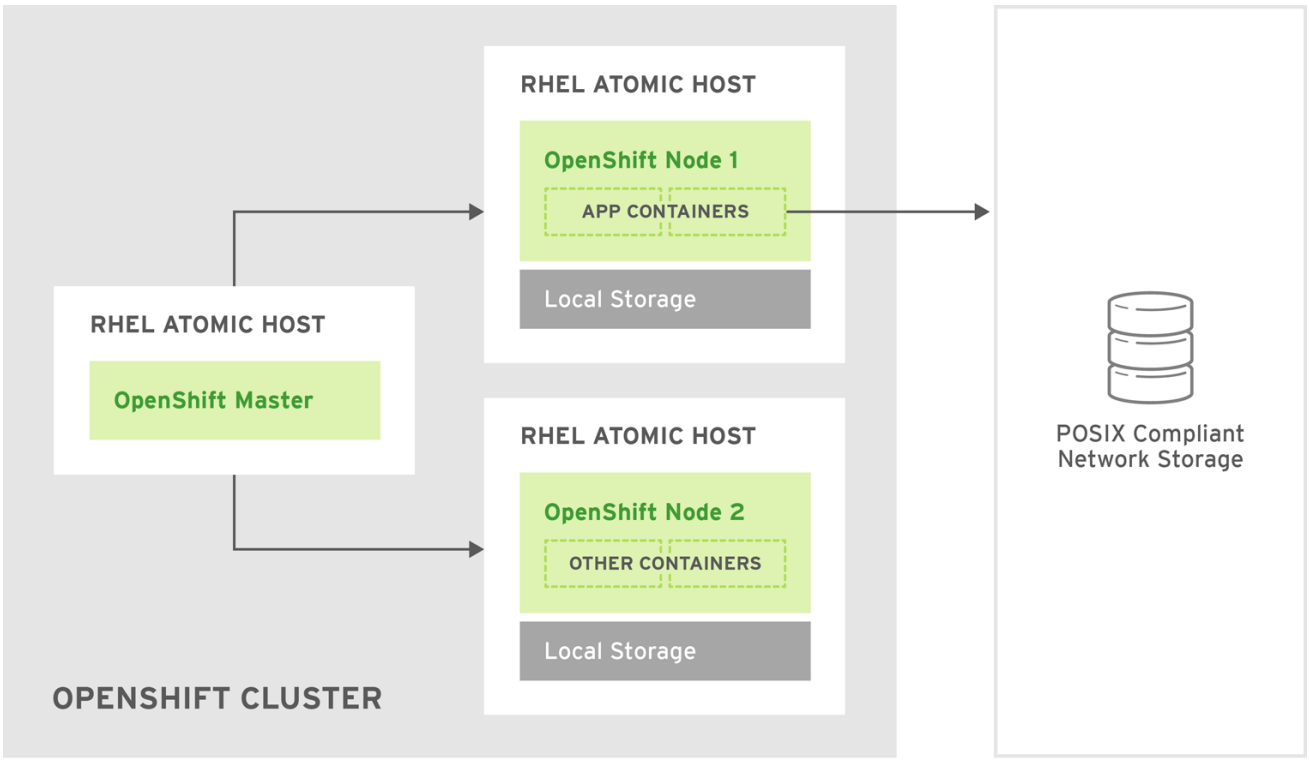
27.3.1.2. independent mode

使用独立模式，Red Hat Gluster Storage 在各自的专用节点上运行，并由 heketi 实例（GlusterFS 卷管理 REST 服务）的实例进行管理。这个 heketi 服务必须以容器化的形式运行，而不是作为单机运行。容器化允许简单的机制为服务提供高可用性。本文档重点介绍容器化 heketi 配置。

27.3.1.3. 独立 Red Hat Gluster Storage

如果您的环境中有独立 Red Hat Gluster Storage 集群，您可以使用 OpenShift Container Platform 的 GlusterFS 卷插件在该集群中使用卷。此解决方案是传统部署，其中应用程序在专用计算节点、OpenShift Container Platform 集群以及存储从自己的专用节点上运行。

图 27.2. 架构 - 使用 OpenShift Container Platform 的 GlusterFS 卷插件的独立红帽 Gluster 存储集群



有关红帽 Gluster 存储的更多信息，请参阅 Red Hat Gluster Storage 安装指南 和 Red Hat Gluster Storage 管理指南。

重要

存储的高可用性功能由底层的存储架构提供。

27.3.1.4. GlusterFS 卷

GlusterFS 卷提供兼容 POSIX 的文件系统，由集群中的一个或多个节点上组成一个或多个"bricks"。brick 只是给定存储节点上的目录，通常是块存储设备的挂载点。GlusterFS 根据卷的配置，处理在给定卷的 brick 之间文件的分发和复制。

建议对大多数常见卷管理操作（如 create、delete 和 resize）使用 heketi。在使用 GlusterFS 置备程序时，OpenShift Container Platform 预期存在 heketi。默认情况下，创建属于三个ray 副本的卷，即每个文件在三个不同节点间有三个副本的卷。因此，建议任何由 heketi 使用的 Red Hat Gluster Storage 集群都至少提供三个节点。

有许多适用于 GlusterFS 卷的功能，但这些内容已超出本文档的范围。

27.3.1.5. Gluster-block 卷

Gluster 块卷是可通过 iSCSI 挂载的卷。这可以通过在现有 GlusterFS 卷上创建文件，然后通过 iSCSI 目标将该文件作为块设备进行。这种 GlusterFS 卷称为块托管卷。

Gluster-block 卷呈现一种利弊。被用作 iSCSI 目标，gluster-block 卷一次只能被一个节点/客户端挂载，它与 GlusterFS 卷相比，可由多个节点/客户端挂载。但是，在后端文件中，允许在 GlusterFS 卷上通常昂贵的操作（例如元数据查找）转换为 GlusterFS 卷上速度要快得多的操作（如读写）。这为某些工作负载带来潜在的显著性能改进。

重要

如需有关 OpenShift Container Storage 和 OpenShift Container Platform 互操作性的更多信息，请参阅链接：OpenShift Container Storage 和 OpenShift Container Platform 互操作性列表。

27.3.1.6. Gluster S3 存储

Gluster S3 服务允许用户应用程序通过 S3 接口访问 GlusterFS 存储。服务绑定到两个 GlusterFS 卷，一个用于对象数据，一个用于对象元数据，并将传入的 S3 REST 请求转换为卷上的文件系统操作。建议您在 OpenShift Container Platform 中作为 pod 运行该服务。

重要

目前，使用和安装 Gluster S3 服务处于技术预览状态。

27.3.2. 注意事项

本节介绍了在 OpenShift Container Platform 中使用 Red Hat Gluster Storage 时需要考虑的一些主题。

27.3.2.1. 软件先决条件

要访问 GlusterFS 卷，必须在所有可调度节点上使用 **mount.glusterfs** 命令。对于基于 RPM 的系统，必须安装 **glusterfs-fuse** 软件包：

```
# yum install glusterfs-fuse
```

这个软件包会在每个 RHEL 系统上安装。但是，如果您的服务器使用 x86_64 架构，则建议您将 Red Hat Gluster Storage 升级到最新的可用版本。要做到这一点，必须启用以下 RPM 存储库：

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-server-rpms
```

如果已在节点上安装了 **glusterfs-fuse**，请确保安装了最新版本：

```
# yum update glusterfs-fuse
```

27.3.2.2. 硬件要求

任何在聚合模式或独立模式集群中使用的节点都被视为存储节点。虽然单个节点不能在多个组中，但存储节点可以被分到不同的集群组。对于每个存储节点组：

- 根据存储 gluster volumetype 选项，每个组最少需要一个或多个存储节点。
- 每个存储节点必须至少有 8 GB RAM。这可允许运行 Red Hat Gluster Storage Pod，以及其他应用程序和底层操作系统。
  - 每个 GlusterFS 卷也在其存储集群中的每个存储节点上消耗内存，大约是 30MB。RAM 总量应该根据计划或预期的并发卷数量来决定。
- 每个存储节点必须至少有一个没有当前数据或元数据的原始块设备。这些块设备将完全用于 GlusterFS 存储。确保不存在以下内容：
  - 分区表（GPT 或 MSDOS）

- 文件系统或者文件系统签名
- 前卷组和逻辑卷的 LVM2 签名
- LVM2 物理卷的 LVM2 元数据

如果有疑问，使用 **wipefs -a <device>** 命令清除以上数据。

重要

建议规划两个集群：一个用于存储基础架构应用程序（如 OpenShift Container Registry）的专用存储集群，和一个用于常规应用程序的专用存储集群。这需要总共 6 个存储节点。此项建议是为了避免在创建 I/O 和卷时对性能造成潜在影响。

27.3.2.3. 存储大小

每个 GlusterFS 集群都必须根据预期应用程序使用其存储的需求进行大小。例如，OpenShift Logging 和 OpenShift 指标 都有大小指南。

需要考虑的一些额外事项有：

- 对于每个聚合模式或独立模式集群，默认行为是使用三路复制来创建 GlusterFS 卷。因此，计划的总存储应是所需的容量。
  - 例如，每个 heti 实例都会创建一个大小为 2 GB 的 **hetidbstorage** 卷，它至少需要 6 GB 的原始存储。此容量始终是必需的，在计算大小时始终需要考虑它。
  - 集成的 OpenShift Container Registry 等应用程序在应用程序的多个实例间共享一个 GlusterFS 卷。
- Gluster 块卷需要存在 GlusterFS 块托管卷，并且有足够的容量来容纳任何给定块卷的容量。
  - 默认情况下，如果没有这样的块托管卷，则会在集合大小时自动创建一个。这个大小为 100 GB。如果集群中没有足够的空间来创建新块托管卷，则块卷的创建将失败。auto-create 行为和自动创建的卷大小都可配置。
  - 具有多个使用 gluster-block 卷（如 OpenShift Logging 和 OpenShift 指标）的实例的应用程序将每个实例使用一个卷。
- Gluster S3 服务绑定到两个 GlusterFS 卷。在 默认的集群安装中，每个卷都是 1 GB，占用总共 6 GB 的原始存储。

27.3.2.4. 卷操作行为

卷操作（如创建和删除）可能会受到各种环境环境的影响，还可影响应用程序。

- 如果应用程序 pod 请求动态置备的 GlusterFS 持久性卷声明 (PVC)，那么可能需要考虑额外的时间以便创建并绑定到对应的 PVC。这影响了应用容器集的启动时间。

注意

GlusterFS 卷的创建时间根据卷数量进行线性扩展。例如，使用推荐的硬件规格的集群中有 100 个卷，每个卷需要大约 6 秒来创建、分配和绑定到 pod。

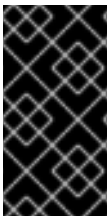
- 删除 PVC 时，该操作会触发删除基础 GlusterFS 卷。虽然 PVC 会立即从 **oc get pvc** 输出中消失，但这并不表示该卷已被完全删除。只有在 **heketi-cli 卷列表和 gluster volume list 和 gluster volume list** 没有出现在命令行输出中时，才能考虑 GlusterFS 卷。

注意

删除 GlusterFS 卷并回收其存储的时间取决于和线性扩展，使用活跃的 GlusterFS 卷数量进行线性扩展。虽然待处理卷删除不会影响正在运行的应用程序，但存储管理员应该了解并可以估算它们需要的时间，特别是大规模调整资源消耗的时间。

27.3.2.5. 卷安全性

本节涵盖 Red Hat Gluster Storage 卷安全性，包括可移植操作系统接口 [对于 Unix](POSIX)权限和 SELinux 的注意事项。了解 卷安全性、POSIX 权限和 SELinux 的基础知识。



重要

在 OpenShift Container Storage 3.11 中，您必须启用 SSL 加密以确保对持久性卷的安全访问控制。

如需更多信息，请参阅 Red Hat OpenShift Container Storage 3.11 操作指南 。

27.3.2.5.1. POSIX 权限

Red Hat Gluster 存储卷提供兼容 POSIX 的文件系统。因此，可以使用 **chmod** 和 **chown** 等标准命令行工具来管理访问权限。

对于聚合模式和独立模式，还可指定在卷创建时拥有卷根的组 ID。对于静态置备，这被指定为 **heketi-cli** 卷创建命令的一部分：

```
$ heketi-cli volume create --size=100 --gid=10001000
```

警告

与此卷关联的 PersistentVolume 必须使用组 ID 标注，以便消耗 PersistentVolume 的 pod 能够访问该文件系统。此注解采用以下格式：

pv.beta.kubernetes.io/gid: "<GID>" ---

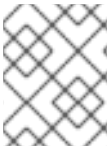
对于动态置备，置备程序会自动生成并应用组 ID。可以使用 **gidMin** 和 **gidMax** StorageClass 参数控制从中选择此组 ID 的范围（请参阅 Dynamic Provisioning）。置备程序还负责使用组 ID 为生成的 PersistentVolume 标注。

27.3.2.5.2. SELinux

默认情况下，SELinux 不允许从 pod 写入远程 Red Hat Gluster 存储服务器。要启用使用 SELinux 对 Red Hat Gluster 存储卷写入，请在运行 GlusterFS 的每个节点中运行以下命令：

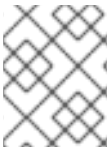
```
$ sudo setsebool -P virt_sandbox_use_fusefs on 1
$ sudo setsebool -P virt_use_fusefs on
```

1 使用 **-P** 选项可使布尔值在系统重启后持久保留。



注意

virt_sandbox_use_fusefs 布尔值由 **docker-selinux** 软件包提供。如果您收到一个错误，说明它没有被定义，请确保已安装此软件包。



注意

如果您使用 Atomic Host，升级 Atomic 主机时将清除 SELinux 布尔值。升级 Atomic Host 时，您必须再次设置这些布尔值。

27.3.3. 支持要求

创建受支持的 Red Hat Gluster Storage 和 OpenShift Container Platform 集成需要满足以下要求。

对于独立模式或独立红帽 Gluster 存储：

- 最低版本：Red Hat Gluster Storage 3.4
- 所有 Red Hat Gluster Storage 节点都必须具有红帽网络频道和 Subscription Manager 仓库的有效订阅。
- Red Hat Gluster Storage 节点必须遵循 规划 Red Hat Gluster Storage 安装 中指定的要求。
- Red Hat Gluster Storage 节点必须完全使用最新的补丁和升级。请参阅 Red Hat Gluster Storage 安装指南以升级到最新版本。
- 必须为每个 Red Hat Gluster Storage 节点设置完全限定域名(FQDN)。确保存在正确的 DNS 记录，并且 FQDN 可通过正向和反向 DNS 查找解析。

27.3.4. 安装

对于一个独立的 Red Hat Gluster Storage，则不需要在 OpenShift Container Platform 中使用组件安装。OpenShift Container Platform 附带一个内置的 GlusterFS 卷驱动程序，允许它在现有集群中使用现有卷。有关如何使用现有卷的更多信息，请参阅 置备。

对于聚合模式和独立模式，建议使用集群安装过程来安装所需组件。

27.3.4.1. independent mode:安装 Red Hat Gluster Storage 节点

对于独立模式，每个 Red Hat Gluster Storage 节点都必须运行适当的系统配置（如防火墙端口、内核模块、Red Hat Gluster Storage 服务）。这些服务不应进一步配置，并且不应形成一个受信存储池。

Red Hat Gluster Storage 节点的安装已超出本文档的范围。如需更多信息，请参阅设置独立模式。

27.3.4.2. 使用安装程序

重要

将单独的节点用于 **glusterfs** 和 **glusterfs_registry** 节点组。每一实例必须是单独的 **gluster** 实例，因为它们需要独立管理。将同一节点用于 **glusterfs** 和 **glusterfs_registry** 节点组会导致部署失败。

集群安装过程 可以用来安装两个 GlusterFS 节点组群中的一个或两个：

- **glusterfs**:供用户应用程序使用的一般存储集群。
- **glusterfs_registry**:专用存储集群，供基础架构应用程序使用，如集成的 OpenShift Container Registry。

建议您部署这两个组以避免对 I/O 和卷创建的性能造成潜在的影响。它们都在清单文件中定义。

要定义存储集群，请在 **[OSEv3: Child]** 组中包括相关的名称，创建相似的命名组。然后，使用节点信息填充组。

在 **[OSEv3: Child]** 组中，添加 **master**、**节点**、**etcd** 和 **glusterfs** 和 **glusterfs_registry** 存储集群。

创建并填充组后，您可以通过在 **[OSEv3:vars]** 组中定义更多参数值来配置集群。变量与 GlusterFS 集群交互，存储在清单文件中，如下例所示。

- **GlusterFS** 变量以 **openshift_storage_glusterfs_** 开头。
- **glusterfs_registry** 变量以 **openshift_storage_glusterfs_registry_** 开头。

以下清单文件示例演示了在部署两个 GlusterFS 节点组时使用变量：

```
`[OSEv3:children]
masters
nodes
etcd
glusterfs
glusterfs_registry`

[OSEv3:vars]
install_method=rpm
os_update=false
install_update_docker=true
docker_storage_driver=devicemapper
ansible_ssh_user=root
openshift_release=v3.11
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:v3.11
#openshift_cockpit_deployer_image='registry.redhat.io/openshift3/registry-console:v3.11'
openshift_docker_insecure_registries=registry.access.redhat.com
openshift_deployment_type=openshift-enterprise
openshift_web_console_install=true
openshift_enable_service_catalog=false
osm_use_cockpit=false
osm_cockpit_plugins=['cockpit-kubernetes']
debug_level=5
openshift_set_hostname=true
openshift_override_hostname_check=true
openshift_disable_check=docker_image_availability
openshift_check_min_host_disk_gb=2
```

```
openshift_check_min_host_memory_gb=1
openshift_portal_net=172.31.0.0/16
openshift_master_cluster_method=native
openshift_clock_enabled=true
openshift_use_openshift_sdn=true

openshift_master_dynamic_provisioning_enabled=true


# logging
openshift_logging_install_logging=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_es_pvc_size=20Gi
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-block"


# metrics
openshift_metrics_install_metrics=true
openshift_metrics_storage_kind=dynamic
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_metrics_storage_volume_size=20Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-registry-block"


# glusterfs
openshift_storage_glusterfs_timeout=900
openshift_storage_glusterfs_namespace=glusterfs
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_create=true
openshift_storage_glusterfs_block_host_vol_size=100


# glusterfs_registry
openshift_storage_glusterfs_registry_namespace=glusterfs-registry
openshift_storage_glusterfs_registry_storageclass=true
openshift_storage_glusterfs_registry_storageclass_default=false
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=false
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_create=true
openshift_storage_glusterfs_registry_block_host_vol_size=100

# glusterfs_registry_storage
openshift_hosted_registry_storage_kind=glusterfs
openshift_hosted_registry_storage_volume_size=20Gi
openshift_hosted_registry_selector="node-role.kubernetes.io/infra=true"
```

```
openshift_storage_glusterfs_heketi_admin_key='adminkey'
openshift_storage_glusterfs_heketi_user_key='heketiuserkey'


openshift_storage_glusterfs_image='registry.access.redhat.com/rhgs3/rhgs-server-rhel7:v3.11'

openshift_storage_glusterfs_heketi_image='registry.access.redhat.com/rhgs3/rhgs-volmanager-
rhel7:v3.11'

openshift_storage_glusterfs_block_image='registry.access.redhat.com/rhgs3/rhgs-gluster-block-prov-
rhel7:v3.11'


openshift_master_cluster_hostname=node101.redhat.com
openshift_master_cluster_public_hostname=node101.redhat.com

[masters]
node101.redhat.com

[etcd]
node101.redhat.com

[nodes]
node101.redhat.com openshift_node_group_name="node-config-master"
node102.redhat.com openshift_node_group_name="node-config-infra"
node103.redhat.com openshift_node_group_name="node-config-compute"
node104.redhat.com openshift_node_group_name="node-config-compute"
node105.redhat.com openshift_node_group_name="node-config-compute"
node106.redhat.com openshift_node_group_name="node-config-compute"
node107.redhat.com openshift_node_group_name="node-config-compute"
node108.redhat.com openshift_node_group_name="node-config-compute"

[glusterfs]
node103.redhat.com glusterfs_zone=1 glusterfs_devices='["/dev/sdd"]'
node104.redhat.com glusterfs_zone=2 glusterfs_devices='["/dev/sdd"]'
node105.redhat.com glusterfs_zone=3 glusterfs_devices='["/dev/sdd"]'

[glusterfs_registry]
node106.redhat.com glusterfs_zone=1 glusterfs_devices='["/dev/sdd"]'
node107.redhat.com glusterfs_zone=2 glusterfs_devices='["/dev/sdd"]'
node108.redhat.com glusterfs_zone=3 glusterfs_devices='["/dev/sdd"]'
```

27.3.4.2.1. 主机变量

glusterfs 和 **glusterfs_registry** 组中的每个主机都必须定义 **glusterfs_devices** 变量。此变量定义作为 GlusterFS 集群一部分管理的块设备列表。您必须至少有一个设备（必须是裸机），没有分区或 LVM PV。

您还可以为每个主机定义以下变量。如果定义这些变量，则这些变量可以进一步控制主机配置作为 GlusterFS 节点：

- **glusterfs_cluster**:此节点所属的集群的 ID。
- **glusterfs_hostname**:用于内部 GlusterFS 通信的主机名或 IP 地址。

- **glusterfs_ip**:pod 用于与 GlusterFS 节点通信的 IP 地址。
- **glusterfs_zone** ：节点的区号。在集群内，zone 决定如何分发 GlusterFS 卷的 brick。

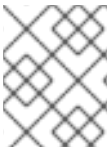
27.3.4.2.2. 角色变量

要控制 GlusterFS 集群整合到新的或现有的 OpenShift Container Platform 集群，您还可以定义存储在清单文件中的多个角色变量。每个角色变量也有一个对应的变量，用于选择性地配置单独的 GlusterFS 集群，用作集成 Docker registry 的存储。

27.3.4.2.3. 镜像名称和版本标签变量

为了防止 OpenShift Container Platform pod 在导致使用不同 OpenShift Container Platform 版本的集群中断后进行升级，建议您为所有容器化组件指定镜像名称和版本标签。这些变量是：

- **openshift_storage_glusterfs_image**
- **openshift_storage_glusterfs_block_image**
- **openshift_storage_glusterfs_s3_image**
- **openshift_storage_glusterfs_heketi_image**



注意

只有在对应的部署变量（以 **_block_deploy** 和 **_s3_deploy**结尾）的相应部署变量，才需要 *gluster-block* 和 *gluster-s3* 的镜像变量。

为了使部署成功，需要有效的镜像标签。将 **<tag>** 替换为与 OpenShift Container Platform 3.11 兼容的 Red Hat Gluster Storage 版本，如清单文件中以下变量互操作性表格 中所述：

- **openshift_storage_glusterfs_image=registry.redhat.io/rhgs3/rhgs-server-rhel7:<tag>**
- **openshift_storage_glusterfs_block_image=registry.redhat.io/rhgs3/rhgs-gluster-block-prov-rhel7:<tag>**
- **openshift_storage_glusterfs_s3_image=registry.redhat.io/rhgs3/rhgs-s3-server-rhel7:<tag>**
- **openshift_storage_glusterfs_heketi_image=registry.redhat.io/rhgs3/rhgs-volmanager-rhel7:<tag>**
- **openshift_storage_glusterfs_registry_image=registry.redhat.io/rhgs3/rhgs-server-rhel7:<tag>**
- **openshift_storage_glusterfs_block_registry_image=registry.redhat.io/rhgs3/rhgs-gluster-block-prov-rhel7:<tag>**
- **openshift_storage_glusterfs_s3_registry_image=registry.redhat.io/rhgs3/rhgs-s3-server-rhel7:<tag>**
- **openshift_storage_glusterfs_heketi_registry_image=registry.redhat.io/rhgs3/rhgs-volmanager-rhel7:<tag>**

有关变量的完整列表，请参阅 GitHub 上的 GlusterFS 角色 README。

配置了变量后，会根据安装情况有几个可用的 playbook：

- 集群安装的主要 playbook 可用于在 OpenShift Container Platform 初始安装的情况下部署 GlusterFS 集群。
  - 这包括部署使用 GlusterFS 存储的集成 OpenShift Container Registry。
  - 这不包括 OpenShift Logging 或 OpenShift Metrics，因为当前仍然是一个单独的步骤。如需更多信息，请参阅 OpenShift Logging 和 Metrics 的聚合模式 。
- **playbooks/openshift-glusterfs/config.yml** 可用于将集群部署到现有的 OpenShift Container Platform 安装中。
- **playbooks/openshift-glusterfs/registry.yml** 可用于将集群部署到现有的 OpenShift Container Platform 安装中。另外，这将部署使用 GlusterFS 存储的集成的 OpenShift 容器 Registry。

重要

在 OpenShift Container Platform 集群中不能有已存在的 registry。

- **playbooks/openshift-glusterfs/uninstall.yml** 可用于删除与清单主机文件中配置匹配的现有集群。这在因为配置错误而部署失败时清理 OpenShift Container Platform 环境会很有用。

注意

GlusterFS playbook 无法保证具有幂等性。

注意

目前，不支持在不删除整个 GlusterFS 安装（包括磁盘数据）并启动整个 GlusterFS 安装的情况下，多次运行 playbook。

27.3.4.2.4. 例如：基本聚合模式安装

1. 在清单文件的 **[OSEv3:vars]** 部分中包含以下变量，并根据您的配置需要调整它们：

```
[OSEv3:vars]
...
openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
```

2. 在 **[OSEv3: Child]** 部分添加 **glusterfs** 来启用 **[glusterfs]** 组：

```
[OSEv3:children]
masters
nodes
glusterfs
```

3. 添加 **[glusterfs]** 部分，其中包含托管 GlusterFS 存储的每个存储节点的条目。对于每个节点，将 **glusterfs_devices** 设置为作为 GlusterFS 集群一部分完全管理的原始块设备列表。必须至少列出一个设备。每个设备都必须是空的，没有分区或 LVM PV。以以下形式指定变量：

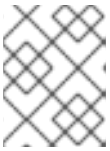
```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "</path/to/device2>", ... ]'
```

例如：

```
[glusterfs]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. 将 **[glusterfs]** 下列出的主机添加到 **[nodes]** 组中：

```
[nodes]
...
node11.example.com openshift_node_group_name="node-config-compute"
node12.example.com openshift_node_group_name="node-config-compute"
node13.example.com openshift_node_group_name="node-config-compute"
```



注意

前面的步骤仅提供一些必须添加到清单文件中的选项。使用完整清单文件来部署红帽 Gluster 存储。

5. 切换到 playbook 目录并运行安装 playbook。将清单文件的相对路径作为选项提供。

- 对于新的 OpenShift Container Platform 安装：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
$ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
```

- 对于现有 OpenShift Container Platform 集群安装：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
```

27.3.4.2.5. 例如：基本独立模式安装

1. 在清单文件的 **[OSEv3:vars]** 部分中包含以下变量，并根据您的配置需要调整它们：

```
[OSEv3:vars]
...
openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
openshift_storage_glusterfs_is_native=false
```

```
openshift_storage_glusterfs_heketi_is_native=true
openshift_storage_glusterfs_heketi_executor=ssh
openshift_storage_glusterfs_heketi_ssh_port=22
openshift_storage_glusterfs_heketi_ssh_user=root
openshift_storage_glusterfs_heketi_ssh_sudo=false
openshift_storage_glusterfs_heketi_ssh_keyfile="/root/.ssh/id_rsa"
```

2. 在 **[OSEv3: Child]** 部分添加 **glusterfs** 来启用 **[glusterfs]** 组：

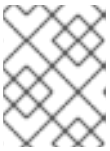
```
[OSEv3:children]
masters
nodes
glusterfs
```

3. 添加 **[glusterfs]** 部分，其中包含托管 GlusterFS 存储的每个存储节点的条目。对于每个节点，将 **glusterfs_devices** 设置为作为 GlusterFS 集群一部分完全管理的原始块设备列表。必须至少列出一个设备。每个设备都必须是空的，没有分区或 LVM PV。另外，将 **glusterfs_ip** 设置为节点的 IP 地址。以以下形式指定变量：

```
<hostname_or_ip> glusterfs_ip=<ip_address> glusterfs_devices='[ "</path/to/device1/>", "
</path/to/device2>", ... ]'
```

例如：

```
[glusterfs]
gluster1.example.com glusterfs_ip=192.168.10.11 glusterfs_devices='[ "/dev/xvdc",
"/dev/xvdd" ]'
gluster2.example.com glusterfs_ip=192.168.10.12 glusterfs_devices='[ "/dev/xvdc",
"/dev/xvdd" ]'
gluster3.example.com glusterfs_ip=192.168.10.13 glusterfs_devices='[ "/dev/xvdc",
"/dev/xvdd" ]'
```



注意

前面的步骤仅提供一些必须添加到清单文件中的选项。使用完整清单文件来部署红帽 Gluster 存储。

4. 切换到 playbook 目录并运行安装 playbook。将清单文件的相对路径作为选项提供。

   - 对于新的 OpenShift Container Platform 安装：

     ```
     $ cd /usr/share/ansible/openshift-ansible
     $ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
     $ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
     ```

   - 对于现有 OpenShift Container Platform 集群安装：

     ```
     $ cd /usr/share/ansible/openshift-ansible
     $ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
     ```

27.3.4.2.6. 示例：带有集成的 OpenShift Container Registry 的聚合模式

1. 在清单文件的 **[OSEv3:vars]** 部分中设置以下变量，并根据您的配置需要调整它们：

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs 1
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-role.kubernetes.io/infra=true'
```

1 建议在基础架构节点上运行集成的 OpenShift Container Registry。基础架构节点是专用于运行管理员部署的应用程序的节点,用于为 OpenShift Container Platform 集群提供服务。

2. 在 **[OSEv3: Child]** 部分添加 **glusterfs_registry** 来启用 **[glusterfs_registry]** 组：

```
[OSEv3:children]
masters
nodes
glusterfs_registry
```

3. 添加 **[glusterfs_registry]** 部分，其中包含托管 GlusterFS 存储的每个存储节点的条目。对于每个节点，将 **glusterfs_devices** 设置为作为 GlusterFS 集群一部分完全管理的原始块设备列表。必须至少列出一个设备。每个设备都必须是空的，没有分区或 LVM PV。以以下形式指定变量：

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "</path/to/device2>", ... ]'
```

例如：

```
[glusterfs_registry]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. 将 **[glusterfs_registry]** 下列出的主机添加到 **[nodes]** 组中：

```
[nodes]
...
node11.example.com openshift_node_group_name="node-config-infra"
node12.example.com openshift_node_group_name="node-config-infra"
node13.example.com openshift_node_group_name="node-config-infra"
```

注意

前面的步骤仅提供一些必须添加到清单文件中的选项。使用完整清单文件来部署红帽 Gluster 存储。

5. 切换到 playbook 目录并运行安装 playbook。将清单文件的相对路径作为选项提供。

- 对于新的 OpenShift Container Platform 安装：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
$ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
```

- 对于现有 OpenShift Container Platform 集群安装：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
```

27.3.4.2.7. 示例：OpenShift Logging 和 Metrics 聚合模式

1. 在清单文件中，在 **[OSEv3:vars]** 部分中设置以下变量，并根据您的配置需要调整它们：

```
[OSEv3:vars]
...

openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra": "true"} 1
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra": "true"} 2
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra": "true"} 3
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-registry-block" 4

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"} 5
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"} 6
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"} 7
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi 8
openshift_logging_elasticsearch_storage_type=pvc 9
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-block" 10

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=false
```

1 2 3 5 6 7 建议您在专用于"infrastructure"应用程序的节点上运行集成的 OpenShift Container Registry、日志记录和指标，它们是管理员为 OpenShift Container Platform 集群提供服务的应用程序。

4 10 指定用于日志记录和指标的 StorageClass。这个名称由目标 GlusterFS 集群的名称生成（如 **glusterfs-<name>-block**）。在本例中，默认为 **registry**。

8 OpenShift Logging 要求指定 PVC 大小。提供的值只是一个示例，而不是推荐。

9 如果使用 Persistent Elasticsearch Storage，请将存储类型设置为 **pvc**。

注意

如需了解有关它们和其他变量的详细信息，请参阅 GlusterFS 角色 README。

2. 在 **[OSEv3: Child]** 部分添加 **glusterfs_registry** 来启用 **[glusterfs_registry]** 组：

```
[OSEv3:children]
masters
```

```
nodes
glusterfs_registry
```

3. 添加 **[glusterfs_registry]** 部分，其中包含托管 GlusterFS 存储的每个存储节点的条目。对于每个节点，将 **glusterfs_devices** 设置为作为 GlusterFS 集群一部分完全管理的原始块设备列表。必须至少列出一个设备。每个设备都必须是空的，没有分区或 LVM PV。以以下形式指定变量：

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "</path/to/device2>", ... ]'
```

例如：

```
[glusterfs_registry]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. 将 **[glusterfs_registry]** 下列出的主机添加到 **[nodes]** 组中：

```
[nodes]
...
node11.example.com openshift_node_group_name="node-config-infra"
node12.example.com openshift_node_group_name="node-config-infra"
node13.example.com openshift_node_group_name="node-config-infra"
```

注意

前面的步骤仅提供一些必须添加到清单文件中的选项。使用完整清单文件来部署红帽 Gluster 存储。

5. 切换到 playbook 目录并运行安装 playbook。将清单文件的相对路径作为选项提供。

- 对于新的 OpenShift Container Platform 安装：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
$ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
```

- 对于现有 OpenShift Container Platform 集群安装：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
```

27.3.4.2.8. 示例：用于应用程序、Registry、日志记录和指标的聚合模式

1. 在清单文件中，在 **[OSEv3:vars]** 部分中设置以下变量，并根据您的配置需要调整它们：

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs (1)
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-role.kubernetes.io/infra=true'

openshift_metrics_install_metrics=true
```

```
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra": "true"} 2
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra": "true"} 3
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra": "true"} 4
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-registry-block" 5

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"} 6
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"} 7
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"} 8
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi 9
openshift_logging_elasticsearch_storage_type=pvc 10
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-block" 11

openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100 12
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=false
```

1 2 3 4 6 7 8 建议在基础架构节点上运行集成的 OpenShift Container Registry、日志记录和指标。基础架构节点是专用于运行管理员部署的应用程序的节点,用于为 OpenShift Container Platform 集群提供服务。

5 11 指定用于日志记录和指标的 StorageClass。这个名称由目标 GlusterFS 集群的名称生成，如 **glusterfs-<name>-block**。在本例中，&lt ;**name&gt**; 默认为 **registry**。

9 OpenShift Logging 需要指定 PVC 大小。提供的值只是一个示例，而不是推荐。

10 如果使用 Persistent Elasticsearch Storage，请将存储类型设置为 **pvc**。

12 将自动创建的 GlusterFS 卷的大小（以 GB 为单位）来托管 glusterblock 卷。只有在 glusterblock 卷创建请求没有足够空间时才使用此变量。这个值代表 glusterblock 卷大小上限，除非您手动创建更大的 GlusterFS 块托管卷。

2. 在 **[OSEv3: Child]** 部分添加 **glusterfs** 和 **glusterfs_registry** 来启用 **[glusterfs]** 和 **[glusterfs_registry]** 组：

```
[OSEv3:children]
...
glusterfs
glusterfs_registry
```

3. 添加 **[glusterfs]** 和 **[glusterfs_registry]** 部分，其中包含托管 GlusterFS 存储的每个存储节点的条目。对于每个节点，将 **glusterfs_devices** 设置为作为 GlusterFS 集群一部分完全管理的原始块设备列表。必须至少列出一个设备。每个设备都必须是空的，没有分区或 LVM PV。以以下形式指定变量：

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "</path/to/device2>", ... ]'
```

例如：

```
[glusterfs]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'

[glusterfs_registry]
node14.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node15.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node16.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. 将 **[glusterfs]** 和 **[glusterfs_registry]** 下列出的主机添加到 **[nodes]** 组中：

```
[nodes]
...
node11.example.com openshift_node_group_name='node-config-compute'  1
node12.example.com openshift_node_group_name='node-config-compute'  2
node13.example.com openshift_node_group_name='node-config-compute'  3
node14.example.com openshift_node_group_name='node-config-infra'"  4
node15.example.com openshift_node_group_name='node-config-infra'"  5
node16.example.com openshift_node_group_name='node-config-infra'"  6
```

1 2 3 4 5 6 节点被标记为表示是否允许将常规应用程序或基础架构应用程序调度到它们上。管理员是配置应用的约束方式。

注意

前面的步骤仅提供一些必须添加到清单文件中的选项。使用完整清单文件来部署红帽 Gluster 存储。

5. 切换到 playbook 目录并运行安装 playbook。将清单文件的相对路径作为选项提供。

- 对于新的 OpenShift Container Platform 安装：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
$ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
```

- 对于现有 OpenShift Container Platform 集群安装：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
```

27.3.4.2.9. 示例：用于应用程序、Registry、日志记录和指标的独立模式

1. 在清单文件中，在 **[OSEv3:vars]** 部分中设置以下变量，并根据您的配置需要调整它们：

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs (1)
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-role.kubernetes.io/infra=true'

openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra": "true"} (2)
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra": "true"} (3)
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra": "true"} (4)
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-registry-block" (5)

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"} (6)
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"} (7)
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"} (8)
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi (9)
openshift_logging_elasticsearch_storage_type (10)
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-block" (11)

openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100 (12)
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
openshift_storage_glusterfs_is_native=false
openshift_storage_glusterfs_heketi_is_native=true
openshift_storage_glusterfs_heketi_executor=ssh
openshift_storage_glusterfs_heketi_ssh_port=22
openshift_storage_glusterfs_heketi_ssh_user=root
openshift_storage_glusterfs_heketi_ssh_sudo=false
openshift_storage_glusterfs_heketi_ssh_keyfile="/root/.ssh/id_rsa"

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=false
openshift_storage_glusterfs_registry_is_native=false
openshift_storage_glusterfs_registry_heketi_is_native=true
openshift_storage_glusterfs_registry_heketi_executor=ssh
openshift_storage_glusterfs_registry_heketi_ssh_port=22
openshift_storage_glusterfs_registry_heketi_ssh_user=root
openshift_storage_glusterfs_registry_heketi_ssh_sudo=false
openshift_storage_glusterfs_registry_heketi_ssh_keyfile="/root/.ssh/id_rsa"
```

1 2 3 4 6 7 8 建议您在专用于"infrastructure"应用程序的节点上运行集成的 OpenShift Container Registry，它们是管理员为 OpenShift Container Platform 集群提供服务的应用程序。管理员最多可为基础架构应用程序选择和标记节点。

5 11 指定用于日志记录和指标的 StorageClass。这个名称由目标 GlusterFS 集群的名称生成（如 **glusterfs-<name>-block**）。在本例中，默认为 **registry**。

9 OpenShift Logging 要求指定 PVC 大小。提供的值只是一个示例，而不是推荐。

10 如果使用 Persistent Elasticsearch Storage，请将存储类型设置为 **pvc**。

12 将自动创建的 GlusterFS 卷的大小（以 GB 为单位）来托管 glusterblock 卷。只有在 glusterblock 卷创建请求没有足够空间时才使用此变量。这个值代表 glusterblock 卷大小上限，除非您手动创建更大的 GlusterFS 块托管卷。

2. 在 **[OSEv3: Child]** 部分添加 **glusterfs** 和 **glusterfs_registry** 来启用 **[glusterfs]** 和 **[glusterfs_registry]** 组：

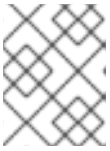
```
[OSEv3:children]
...
glusterfs
glusterfs_registry
```

3. 添加 **[glusterfs]** 和 **[glusterfs_registry]** 部分，其中包含托管 GlusterFS 存储的每个存储节点的条目。对于每个节点，将 **glusterfs_devices** 设置为作为 GlusterFS 集群一部分完全管理的原始块设备列表。必须至少列出一个设备。每个设备都必须是空的，没有分区或 LVM PV。另外，将 **glusterfs_ip** 设置为节点的 IP 地址。以以下形式指定变量：

```
<hostname_or_ip> glusterfs_ip=<ip_address> glusterfs_devices='[ "</path/to/device1/>", "
</path/to/device2>", ... ]'
```

例如：

```
[glusterfs]
gluster1.example.com glusterfs_ip=192.168.10.11 glusterfs_devices='[ "/dev/xvdc",
"/dev/xvdd" ]'
gluster2.example.com glusterfs_ip=192.168.10.12 glusterfs_devices='[ "/dev/xvdc",
"/dev/xvdd" ]'
gluster3.example.com glusterfs_ip=192.168.10.13 glusterfs_devices='[ "/dev/xvdc",
"/dev/xvdd" ]'

[glusterfs_registry]
gluster4.example.com glusterfs_ip=192.168.10.14 glusterfs_devices='[ "/dev/xvdc",
"/dev/xvdd" ]'
gluster5.example.com glusterfs_ip=192.168.10.15 glusterfs_devices='[ "/dev/xvdc",
"/dev/xvdd" ]'
gluster6.example.com glusterfs_ip=192.168.10.16 glusterfs_devices='[ "/dev/xvdc",
"/dev/xvdd" ]'
```



注意

前面的步骤仅提供一些必须添加到清单文件中的选项。使用完整清单文件来部署红帽 Gluster 存储。

4. 切换到 playbook 目录并运行安装 playbook。将清单文件的相对路径作为选项提供。

   - 对于新的 OpenShift Container Platform 安装：

     ```
     $ cd /usr/share/ansible/openshift-ansible
     $ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
     $ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
     ```

   - 对于现有 OpenShift Container Platform 集群安装：

     ```
     $ cd /usr/share/ansible/openshift-ansible
     $ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
     ```

27.3.5. 卸载聚合模式

对于聚合模式，OpenShift Container Platform 安装会附带一个 playbook 来卸载集群中的所有资源和工件。要使用 playbook，请提供用于安装聚合模式目标实例的原始清单文件，更改为 playbook 目录，并运行以下 playbook：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/uninstall.yml
```

另外，playbook 支持使用名为 **openshift_storage_glusterfs_wipe** 的变量，在启用时，销毁用于 Red Hat Gluster Storage 后端存储的块设备上的任何数据。要使用 **openshift_storage_glusterfs_wipe** 变量，请切换到 playbook 目录并运行以下 playbook:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> -e \
  "openshift_storage_glusterfs_wipe=true" \
  playbooks/openshift-glusterfs/uninstall.yml
```

警告

这个过程销毁数据。请小心操作。

27.3.6. 置备

GlusterFS 卷可以静态或动态置备。静态置备适用于所有配置。只有聚合模式和独立模式支持动态置备。

27.3.6.1. 静态置备

1. 要启用静态置备，首先请创建一个 GlusterFS 卷。请参阅 Red Hat Gluster Storage Administration Guide 了解如何使用 **gluster** 命令行的信息；请参阅 heketi 项目网站来了解如何使用 **heketi-cli** 的信息。在本例中，卷将命名为 **myVol1**。

2. 在 **gluster-endpoints.yaml** 中定义以下服务和端点：

   ```
   ---
   ```

```
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster (1)
spec:
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster (2)
subsets:
  - addresses:
      - ip: 192.168.122.221 (3)
    ports:
      - port: 1 (4)
  - addresses:
      - ip: 192.168.122.222 (5)
    ports:
      - port: 1 (6)
  - addresses:
      - ip: 192.168.122.223 (7)
    ports:
      - port: 1 (8)
```

(1)(2) 这些名称必须匹配。

(3)(5)(7) **ip** 值必须是 Red Hat Gluster Storage 服务器的实际 IP 地址，而不是主机名。

(4)(6)(8) 端口号被忽略。

3. 在 OpenShift Container Platform master 主机上创建服务和端点：

```
$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created
```

4. 验证服务和端点是否已创建：

```
$ oc get services
NAME                CLUSTER_IP      EXTERNAL_IP   PORT(S)    SELECTOR        AGE
glusterfs-cluster   172.30.205.34   <none>        1/TCP      <none>          44s

$ oc get endpoints
NAME                ENDPOINTS                                         AGE
docker-registry     10.1.0.3:5000                                     4h
glusterfs-cluster   192.168.122.221:1,192.168.122.222:1,192.168.122.223:1   11s
kubernetes          172.16.35.3:8443                                  4d
```

注意

端点每个项目都是唯一的。访问 GlusterFS 卷的每个项目都需要自己的端点。

5. 若要访问卷，容器必须使用用户 ID(UID)或组 ID(GID)运行，该容器有权访问卷上的文件系统。这些信息可以通过以下方法发现：

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0    myVol1 1 2
```

1. UID 为 592。
2. GID 是 590。

6. 在 **gluster-pv.yaml** 中定义以下 PersistentVolume(PV)：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume 1
  annotations:
    pv.beta.kubernetes.io/gid: "590" 2
spec:
  capacity:
    storage: 2Gi 3
  accessModes: 4
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster 5
    path: myVol1 6
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

1. 卷的名称。
2. GlusterFS 卷根上的 GID。
3. 为这个卷分配的存储量。
4. **accessModes** 用作标签，以匹配 PV 和 PVC。它们目前没有定义任何形式的访问控制。
5. 之前创建的 Endpoints 资源。
6. 将要访问的 GlusterFS 卷。

7. 在 OpenShift Container Platform master 主机上创建 PV：

```
$ oc create -f gluster-pv.yaml
```

8. 确定创建了 PV：

```
$ oc get pv
NAME                LABELS    CAPACITY     ACCESSMODES   STATUS      CLAIM
```

```
REASON    AGE
gluster-default-volume   <none>   2147483648   RWX          Available                  2s
```

9. 创建一个 PersistentVolumeClaim(PVC)，它将绑定到 **gluster-claim.yaml** 中的新 PV：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim (1)
spec:
  accessModes:
  - ReadWriteMany (2)
  resources:
    requests:
      storage: 1Gi (3)
```

(1) 声明名称由 pod 在其 **volumes** 部分下引用。

(2) 必须与 PV 的 **accessModes** 匹配。

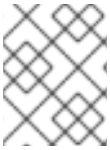
(3) 这个声明会查找提供 **1Gi** 或更高容量的 PV。

10. 在 OpenShift Container Platform master 主机上创建 PVC：

```
$ oc create -f gluster-claim.yaml
```

11. 验证 PV 和 PVC 是否已绑定：

```
$ oc get pv
NAME        LABELS    CAPACITY   ACCESSMODES   STATUS     CLAIM          REASON
AGE
gluster-pv  <none>    1Gi        RWX           Available  gluster-claim          37s

$ oc get pvc
NAME           LABELS    STATUS    VOLUME      CAPACITY   ACCESSMODES   AGE
gluster-claim  <none>    Bound     gluster-pv  1Gi        RWX           24s
```



注意

PVC 每个项目都是唯一的。访问 GlusterFS 卷的每个项目都需要自己的 PVC。PV 不绑定到单个项目，因此多个项目的 PVC 可能会引用同一 PV。

27.3.6.2. 动态置备

1. 要启用动态置备，首先请创建一个 **StorageClass** 对象定义。以下定义基于本示例与 OpenShift Container Platform 搭配使用所需的最低要求。如需了解更多参数和规格定义，请参阅动态置备和创建存储类。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: glusterfs
provisioner: kubernetes.io/glusterfs
```

```
parameters:
  resturl: "http://10.42.0.0:8080" 1
  restauthenabled: "false" 2
```

1 heketi 服务器 URL。

2 由于本例中未打开身份验证，因此设置为 **false**。

2. 在 OpenShift Container Platform master 主机上创建 StorageClass：

```
# oc create -f gluster-storage-class.yaml
storageclass "glusterfs" created
```

3. 使用新创建的 StorageClass 创建 PVC。例如：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster1
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 30Gi
  storageClassName: glusterfs
```

4. 在 OpenShift Container Platform master 主机上创建 PVC：

```
# oc create -f glusterfs-dyn-pvc.yaml
persistentvolumeclaim "gluster1" created
```

5. 查看 PVC，查看卷是否动态创建并绑定到 PVC：

```
# oc get pvc
NAME      STATUS   VOLUME                                     CAPACITY   ACCESSMODES
STORAGECLASS   AGE
gluster1   Bound    pvc-78852230-d8e2-11e6-a3fa-0800279cf26f   30Gi       RWX
glusterfs      42s
```

27.4. 使用 OPENSTACK CINDER 的持久性存储

27.4.1. 概述

您可以使用 OpenStack Cinder 为 OpenShift Container Platform 集群置备持久性存储。我们假设您对 Kubernetes 和 OpenStack 有一定的了解。

重要

在使用 Cinder 创建持久性卷(PV)之前，先 为 OpenStack 配置 OpenShift Container Platform。

Kubernetes 持久性卷框架 允许管理员置备具有持久性存储的集群，并为用户提供在不了解底层基础架构的情况下请求这些资源的方法。您可以 动态置备 OpenStack Cinder 卷。

持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。但是，持久性卷声明 是针对某个项目或命名空间的，用户可请求它。

重要

存储的高可用性功能由底层存储供应商实现。

27.4.2. 置备 Cinder PV

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。确保为 OpenStack 配置 OpenShift Container Platform 后，Cinder 所需的所有内容都是 Cinder 卷 ID 和 **PersistentVolume** API。

27.4.2.1. 创建持久性卷

您必须在对象定义中定义 PV，然后才能在 OpenShift Container Platform 中创建它：

1. 将对象定义保存到文件中，如 ***cinder-pv.yaml*** ：

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" (1)
spec:
  capacity:
    storage: "5Gi" (2)
  accessModes:
    - "ReadWriteOnce"
  cinder: (3)
    fsType: "ext3" (4)
    volumeID: "f37a03aa-6212-4c62-a805-9ce139fab180" (5)
```

(1) 持久性卷声明或 Pod 使用的卷名称。

(2) 为这个卷分配的存储量。

(3) 卷类型，本例中为 **cinder**。

(4) 要挂载的文件系统类型。

(5) 要使用的 Cinder 卷。

重要

不要在卷被格式化和置备后更改 **fstype** 参数值。更改此值可能会导致数据丢失和 pod 失败。

2. 创建持久性卷：

```
# oc create -f cinder-pv.yaml

persistentvolume "pv0001" created
```

3. 验证持久性卷是否存在：

```
# oc get pv

NAME    LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM     REASON
AGE
pv0001  <none>    5Gi        RWO           Available               2s
```

然后，用户可以使用持久性卷声明请求存储，该声明现在可以使用您的新持久性卷。

重要

持久性卷声明只在用户的命名空间中存在，可以被同一命名空间中的 pod 引用。尝试从其他命名空间访问持久性卷声明会导致 pod 失败。

27.4.2.2. Cinder PV 格式

在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查它是否包含由持久性卷定义中的 **fstype** 参数指定的文件系统。如果没有使用文件系统格式化该设备，该设备中的所有数据都会被删除，并使用指定的文件系统自动格式化该设备。

这将可以使用未格式化的 Cinder 卷作为持久性卷，因为 OpenShift Container Platform 在第一次使用前会对其进行格式化。

27.4.2.3. Cinder 卷安全

如果在应用程序中使用 Cinder PV，请在其部署配置中配置安全性。

注意

在实施 Cinder 卷前，检查卷安全性 信息。

1. 创建一个使用适当 **fsGroup** 策略的 SCC。
2. 创建一个服务帐户并将其添加到 SCC：

```
[source,bash]
$ oc create serviceaccount <service_account>
$ oc adm policy add-scc-to-user <new_scc> -z <service_account> -n <project>
```

3. 在应用程序的部署配置中，提供服务帐户名称和 **securityContext**：

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1 (1)
  selector: (2)
```

```
    name: frontend
  template:  3
    metadata:
      labels:  4
        name: frontend  5
    spec:
      containers:
      - image: openshift/hello-openshift
        name: helloworld
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
      serviceAccountName: <service_account>  6
      securityContext:
        fsGroup: 7777  7
```

1. 要运行的 pod 的副本数。
2. 要运行的 pod 的标签选择器。
3. 控制器创建的 pod 模板。
4. pod 上的标签必须包含标签选择器中的标签。
5. 扩展任何参数后的最大名称长度为 63 个字符。
6. 指定您创建的服务帐户。
7. 为 pod 指定 **fsGroup**。

27.4.2.4. Cinder 卷限制

默认情况下，最大 256 Cinder 卷可以附加到集群中的每个节点。要更改这个限制：

1. 将 **KUBE_MAX_PD_VOLS** 环境变量设置为整数。例如，在 **/etc/origin/master/master.env** 中：

```
KUBE_MAX_PD_VOLS=26
```

2. 在命令行中重启 API 服务：

```
# master-restart api
```

3. 在命令行中重启控制器服务：

```
# master-restart controllers
```

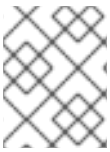
27.5. 使用 CEPH RADOS 块设备(RBD)的持久性存储

27.5.1. 概述

OpenShift Container Platform 集群可以使用 Ceph RBD 来置备持久性存储。

持久性卷(PV)和 持久性卷声明(PVC) 可以在单个项目间共享卷。虽然 PV 定义中包含的与 Ceph RBD 相关的信息也可以直接在 pod 定义中定义，但这样做不会将卷创建为一个特定的集群资源，从而可能会导致卷冲突。

本主题假定您对 OpenShift Container Platform 和 Ceph RBD 有一定的了解。如需了解 OpenShift Container Platform 持久性卷(PV)框架的详细信息，请参阅持久性存储概念主题。



注意

*项目*和*命名空间*在整个文档中可能会互换使用。有关关系的详细信息，请参阅 项目和用户。

重要

存储的高可用性功能由底层存储供应商实现。

27.5.2. 置备

要置备 Ceph 卷，需要以下内容：

- 您的底层基础架构中现有的存储设备。
- 要在 OpenShift Container Platform secret 对象中使用的 Ceph 密钥。
- Ceph 镜像名称。
- 块存储顶部的文件系统类型（例如 ext4）。
- **Ceph-common** 在集群中的每个可调度 OpenShift Container Platform 节点上安装：

```
# yum install ceph-common
```

27.5.2.1. 创建 Ceph Secret

在 secret 配置中定义授权密钥，然后将其转换为 base64，供 OpenShift Container Platform 使用。

注意

要使用 Ceph 存储来备份持久性卷，必须在与 PVC 和 pod 相同的命名空间中创建 secret。secret 不能只是位于 default 项目中。

1. 在 Ceph MON 节点上运行 **ceph auth get-key**，以显示 **client.admin** 用户的键值：

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key: QVFBOFF2SlZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ==
type: kubernetes.io/rbd
```

2. 将 secret 定义保存到文件中，如 ***ceph-secret.yaml***，然后创建 secret：

```
$ oc create -f ceph-secret.yaml
```

3. 验证是否已创建 secret：

```
# oc get secret ceph-secret
NAME          TYPE                DATA      AGE
ceph-secret   kubernetes.io/rbd   1         23d
```

27.5.2.2. 创建持久性卷

开发人员通过引用 PVC 或容器集规格的 **volumes** 部分中的 Gluster 卷插件来请求 Ceph RBD 存储。PVC 只在用户的命名空间中存在，且只能被同一命名空间中的 pod 引用。尝试从不同命名空间中访问 PV 会导致 pod 失败。

1. 在 OpenShift Container Platform 中创建前，在对象定义中定义 PV：

例 27.3. 使用 Ceph RBD 的持久性卷对象定义

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-pv (1)
spec:
  capacity:
    storage: 2Gi (2)
  accessModes:
    - ReadWriteOnce (3)
  rbd: (4)
    monitors: (5)
      - 192.168.122.133:6789
    pool: rbd
    image: ceph-image
    user: admin
    secretRef:
      name: ceph-secret (6)
    fsType: ext4 (7)
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

(1) 在 pod 定义中引用的 PV 名称，或者在各种 **oc** volume 命令中显示。

(2) 为这个卷分配的存储量。

(3) **accessModes** 用作标签，以匹配 PV 和 PVC。它们目前没有定义任何形式的访问控制。所有块存储都是定义为单个用户（非共享存储）。

(4) 使用的卷类型，本例中为 **rbd** 插件。

(5) Ceph 监视器 IP 地址和端口的数组。

(6) 用于创建从 OpenShift Container Platform 到 Ceph 服务器的安全连接的 Ceph secret。

(7) 挂载到 Ceph RBD 块设备上的文件系统类型。

-

重要

在卷被格式化并置备后，修改 **fstype** 参数的值会导致数据丢失和 pod 失败。

2. 将定义保存到文件中，如 ***ceph-pv.yaml*** 并创建 PV：

```
# oc create -f ceph-pv.yaml
```

3. 验证持久性卷是否已创建：

```
# oc get pv
NAME                LABELS    CAPACITY     ACCESSMODES   STATUS      CLAIM
REASON    AGE
ceph-pv             <none>    2147483648   RWO           Available               2s
```

4. 创建一个 PVC，它将绑定到新 PV：

例 27.4. PVC 对象定义

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim
spec:
  accessModes: 1
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi 2
```

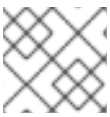
1 **accessModes** 没有被强制访问，而是作为标签来把一个 PV 与 PVC 匹配。

2 这个声明会寻找提供 **2Gi** 或更高容量的 PV。

5. 将定义保存到文件中，如 ***ceph-claim.yaml*** 并创建 PVC：

```
# oc create -f ceph-claim.yaml
```

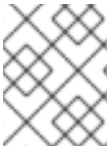
27.5.3. Ceph 卷安全性



注意

在实施 Ceph RBD 卷前，请参阅完整的卷安全性。

共享卷（NFS 和 GlusterFS）与块卷（Ceph RBD、iSCSI 和大多数云存储）之间的显著区别是，容器集定义或容器镜像中定义的用户和组 ID 应用到目标物理存储。这称为管理块设备的所有权。例如，如果 Ceph RBD 挂载的所有者设为 **123**，其组 ID 设为 **567**，如果 pod 定义了 **runAsUser** 设为 **222**，并且其 **fsGroup** 设为 **7777**，则 Ceph RBD 物理挂载的所有权将更改为 **222:7777**。



注意

即使 pod 规格中没有定义用户和组群 ID，生成的 pod 可以根据匹配的 SCC 或其项目为这些 ID 定义默认值。请参阅完整的 卷安全性 主题，它们涵盖了 SCC 的存储方面和默认值。

pod 使用 pod 的 **securityContext** 定义下的 **fsGroup** 小节来定义 Ceph RBD 卷的组所有权：

```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    fsGroup: 7777 2
```

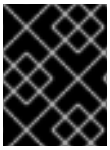
1 **securityContext** 必须在 pod 一级定义，而不是在特定容器中定义。

2 pod 中的所有容器将具有相同的 fsGroup ID。

27.6. 使用 AWS ELASTIC BLOCK STORE 的持久性存储

27.6.1. 概述

OpenShift Container Platform 支持 AWS Elastic Block Store 卷 (EBS) 。您可以使用 AWS EC2 为 OpenShift Container Platform 集群置备持久性存储。我们假设您对 Kubernetes 和 AWS 有一定的了解。



重要

在使用 AWS 创建持久性卷前，必须首先 为 AWS ElasticBlockStore 正确配置 OpenShift Container Platform。

Kubernetes 持久性卷框架 允许管理员置备具有持久性存储的集群，并为用户提供在不了解底层基础架构的情况下请求这些资源的方法。AWS Elastic Block Store 卷可以动态置备。持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。但是，持久性卷声明 是针对某个项目或命名空间的，用户可请求它。

重要

存储的高可用性功能由底层存储供应商实现。

27.6.2. 置备

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。确保为 AWS Elastic Block Store 配置 OpenShift 后，OpenShift 和 AWS 所需的所有内容都是 AWS EBS 卷 ID 和 **PersistentVolume** API。

27.6.2.1. 创建持久性卷

您必须在对象定义中定义持久性卷，然后才能在 OpenShift Container Platform 中创建它：

例 27.5. 使用 AWS 的持久性卷对象定义

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
    storage: "5Gi" 2
  accessModes:
    - "ReadWriteOnce"
  awsElasticBlockStore: 3
    fsType: "ext4" 4
    volumeID: "vol-f37a03aa" 5
```

1. 卷的名称。这将通过 持久性卷声明或 从 pod 识别它。
2. 为这个卷分配的存储量。
3. 这将定义要使用的卷类型，本例中为 **awsElasticBlockStore** 插件。
4. 要挂载的文件系统类型。
5. 这是要使用的 AWS 卷。

重要

在卷被格式化并置备后，修改 **fstype** 参数的值会导致数据丢失和 pod 失败。

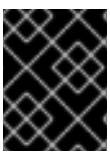
将定义保存到文件中，如 ***aws-pv.yaml*** 并创建持久性卷：

```
# oc create -f aws-pv.yaml
persistentvolume "pv0001" created
```

验证持久性卷是否已创建：

```
# oc get pv
NAME     LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM     REASON    AGE
pv0001   <none>    5Gi        RWO           Available                       2s
```

然后，用户可以使用持久性卷声明请求存储，该声明现在可以使用您的新持久性卷。



重要

持久性卷声明只在用户的命名空间中存在，且只能被同一命名空间中的 pod 引用。任何尝试从其他命名空间中访问持久性卷都会导致 pod 失败。

27.6.2.2. 卷格式

在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查它是否包含由持久性卷定义中的 **fstype** 参数指定的文件系统。如果没有使用文件系统格式化该设备，该设备中的所有数据都会被删除，并使用指定的文件系统自动格式化该设备。

这将可以使用未格式化的 AWS 卷作为持久性卷，因为 OpenShift Container Platform 在第一次使用前会对其进行格式化。

27.6.2.3. 一个节点上的 EBS 卷的最大数目

默认情况下，OpenShift Container Platform 最多支持把 39 个 EBS 卷附加到一个节点。这个限制与 AWS Volume Limits 一致。

通过设置环境变量 **KUBE_MAX_PD_VOLS**，可将 OpenShift Container Platform 配置为具有更高的限制。但是 AWS 需要对附加设备有一个特定的命名方案（AWS Device Naming），它最多只支持 52 个卷。这将把通过 OpenShift Container Platform 附加到节点中的卷数量限制为 52。

27.7. 使用 GCE PERSISTENT DISK 的持久性存储

27.7.1. 概述

OpenShift Container Platform 支持 GCE Persistent Disk 卷 (gcePD)。您可以使用 GCE 为 OpenShift Container Platform 集群置备持久性存储。我们假设您对 Kubernetes 和 GCE 有一定的了解。

重要

在使用 GCE 创建持久性卷之前，必须首先 为 GCE Persistent Disk 正确配置 OpenShift Container Platform。

Kubernetes 持久性卷框架 允许管理员置备具有持久性存储的集群，并为用户提供在不了解底层基础架构的情况下请求这些资源的方法。GCE Persistent Disk 卷 可以动态置备。持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。但是，持久性卷声明 是针对某个项目或命名空间的，用户可请求它。

重要

存储的高可用性功能由底层存储供应商实现。

27.7.2. 置备

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。确保为 GCE PersistentDisk 配置 OpenShift Container Platform 后，OpenShift Container Platform 和 GCE 都需要一个 GCE Persistent Disk 卷 ID 和 **PersistentVolume** API。

27.7.2.1. 创建持久性卷

您必须在对象定义中定义持久性卷，然后才能在 OpenShift Container Platform 中创建它：

例 27.6. 使用 GCE 的持久性卷对象定义

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
    storage: "5Gi" 2
```

```
  accessModes:
    - "ReadWriteOnce"
  gcePersistentDisk: 3
    fsType: "ext4" 4
    pdName: "pd-disk-1" 5
```

1. 卷的名称。这将通过 持久性卷声明或 从 pod 识别它。
2. 为这个卷分配的存储量。
3. 这将定义正在使用的卷类型，本例中为 **gcePersistentDisk** 插件。
4. 要挂载的文件系统类型。
5. 这是将使用的 GCE Persistent Disk 卷。

重要

在卷被格式化并置备后，修改 **fstype** 参数的值会导致数据丢失和 pod 失败。

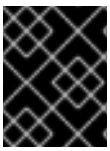
将定义保存到文件中，如 ***gce-pv.yaml*** 并创建持久性卷：

```
# oc create -f gce-pv.yaml
persistentvolume "pv0001" created
```

验证持久性卷是否已创建：

```
# oc get pv
NAME     LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM     REASON    AGE
pv0001   <none>    5Gi        RWO           Available                       2s
```

然后，用户可以使用持久性卷声明请求存储，该声明现在可以使用您的新持久性卷。



重要

持久性卷声明只在用户的命名空间中存在，且只能被同一命名空间中的 pod 引用。任何尝试从其他命名空间中访问持久性卷都会导致 pod 失败。

27.7.2.2. 卷格式

在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查它是否包含由持久性卷定义中的 **fstype** 参数指定的文件系统。如果没有使用文件系统格式化该设备，该设备中的所有数据都会被删除，并使用指定的文件系统自动格式化该设备。

这将可以使用未格式化的 GCE 卷作为持久性卷，因为 OpenShift Container Platform 在第一次使用前会对其进行格式化。

27.8. 使用 ISCSI 的持久性存储

27.8.1. 概述

您可以使用 iSCSI 为 OpenShift Container Platform 集群置备 持久性存储。我们假设您对 Kubernetes 和 iSCSI 有一定的了解。

Kubernetes 持久性卷框架 允许管理员置备具有持久性存储的集群，并为用户提供在不了解底层基础架构的情况下请求这些资源的方法。

重要

存储的高可用性功能由底层存储供应商实现。

27.8.2. 置备

在将存储作为卷挂载到 OpenShift Container Platform 之前，请确认它已存在于底层的基础架构中。iSCSI 需要的所有是 iSCSI 目标门户、有效的 iSCSI 限定名称(IQN)、一个有效的 LUN 号码、文件系统类型和 **PersistentVolume** API。

另外，还可提供多路径门户和 Challenge Handshake Authentication Protocol(CHAP)配置。

例 27.7. 持久性卷对象定义

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.16.154.81:3260
    portals: ['10.16.154.82:3260', '10.16.154.83:3260']
    iqn: iqn.2014-12.example.server:storage.target00
    lun: 0
    fsType: 'ext4'
    readOnly: false
    chapAuthDiscovery: true
    chapAuthSession: true
    secretRef:
      name: chap-secret
```

27.8.2.1. 强制磁盘配额

使用 LUN 分区强制磁盘配额和大小限制。每个 LUN 都是一个持久性卷。kubernetes 为持久性卷强制使用唯一的名称。

以这种方式强制配额可让最终通过指定一个数量（例如，10Gi）来请求持久性存储，并与相等或更大容量的对应卷匹配。

27.8.2.2. iSCSI 卷安全

用户使用 **PersistentVolumeClaim** 来请求存储。这个声明只在用户的命名空间中有效，且只能被在同一命名空间中的 pod 调用。任何尝试访问命名空间中的持久性卷都会导致 pod 失败。

每个 iSCSI LUN 都需要可以被集群中的所有节点访问。

27.8.2.3. iSCSI 多路径

对于基于 iSCSI 的存储，您可以使用相同的 IQN 为多个目标入口 IP 地址配置多路径。通过多路径，当路径中的一个或者多个组件失败时，仍可保证对持久性卷的访问。

使用 portals 字段在 pod 规格中指定 **多路径**。例如：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260'] 1
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    fsType: ext4
    readOnly: false
```

1 使用 **portals** 字段添加额外的目标门户。

27.8.2.4. iSCSI 自定义 Initiator IQN

如果 iSCSI 目标仅限于特定的 IQN，则配置自定义 initiator iSCSI 限定名称 (IQN)，但不会保证 iSCSI PV 附加到的节点具有这些 IQN。

要指定自定义 initiator IQN，请使用 **initiatorName** 字段。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260']
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    initiatorName: iqn.2016-04.test.com:custom.iqn 1
    fsType: ext4
    readOnly: false
```

1 要添加额外的自定义 initiator IQN，请使用 **initiatorName** 字段。

27.9. 使用 FIBRE CHANNEL 的持久性存储

27.9.1. 概述

您可以置备使用 Fibre Channel (FC) 的带有 持久性存储的 OpenShift Container Platform 集群。我们假设您对 Kubernetes 和 FC 有一定的了解。

Kubernetes 持久性卷框架 允许管理员置备具有持久性存储的集群，并为用户提供在不了解底层基础架构的情况下请求这些资源的方法。

重要

存储的高可用性功能由底层存储供应商实现。

27.9.2. 置备

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。FC 持久性存储需要的所有是 **PersistentVolume** API、**wwids** 或 **targetWWNs** （有效的 **lun** 号码）以及 **fsType**。持久性卷和 LUN 之间有一个一对一的映射。

持久性卷对象定义

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  fc:
    wwids: [scsi-3600508b400105e210000900000490000] 1
    targetWWNs: ['500a0981891b8dc5', '500a0981991b8dc5'] 2
    lun: 2 3
    fsType: ext4
```

1 可选：全局广泛的标识符(WWID)。FC **wwids** 或 FC 目标 **WWN** 和 **lun** 的组合必须设置，但不能同时设置。建议在 WWN 目标中使用 FC WWID 标识符，因为它可以保证每个存储设备独有，并且独立于用于访问该设备的路径。通过发出 SCSI Indentification Vital Product Data（**page 0x83** ）或单元 Serial Number（**page 0x80**）来获得 WWID 标识符。FC WWID 被标识为 **/dev/disk/by-id/** 来引用磁盘上的数据，即使设备的路径发生了变化，即使从不同系统访问该设备也是如此。

2 3 可选：World wide name(WWNs)。FC **wwids** 或 FC 目标 **WWN** 和 **lun** 的组合必须设置，但不能同时设置。建议在 WWN 目标中使用 FC WWID 标识符，因为它可以保证每个存储设备独有，并且独立于用于访问该设备的路径。FC WWN 被识别为 **/dev/disk/by-path/pci-<identifier>-fc-0x<wn>-lun-<lun_#** >，但您不需要提供导致 & **lt;wwn** > 的路径的任何部分，包括 **0x** 以及任何包括 **-** (hyphen)。

重要

在卷被格式化并置备后，修改 **fstype** 参数的值会导致数据丢失和 pod 失败。

27.9.2.1. 强制磁盘配额

使用 LUN 分区强制磁盘配额和大小限制。每个 LUN 都是一个持久性卷。kubernetes 为持久性卷强制使用唯一的名称。

以这种方式强制配额可让最终用户以特定数量（如 10 Gi）请求持久性存储，并与相等或更大容量的对应卷匹配。

27.9.2.2. Fibre Channel 卷安全性

用户使用 **PersistentVolumeClaim** 来请求存储。这个声明只在用户的命名空间中有效，且只能被在同一命名空间中的 pod 调用。尝试使用其他命名空间中的持久性卷声明会导致 pod 失败。

每个 FC LUN 必须可以被集群中的所有节点访问。

27.10. 使用 AZURE DISK 的持久性存储

27.10.1. 概述

OpenShift Container Platform 支持 Microsoft Azure Disk 卷。您可以使用 Azure 为 OpenShift Container Platform 集群置备持久性存储。我们假设您对 Kubernetes 和 Azure 有一定的了解。

Kubernetes 持久性卷框架 允许管理员置备具有持久性存储的集群，并为用户提供在不了解底层基础架构的情况下请求这些资源的方法。

Azure 磁盘卷可以动态置备。持久性卷不与某个特定项目或命名空间相关联，它们可以在 OpenShift Container Platform 集群间共享。但是，持久性卷声明 是针对某个项目或命名空间的，用户可请求它。

重要

存储的高可用性功能由底层的存储架构提供。

27.10.2. 先决条件

在使用 Azure 创建持久性卷前，请确保 OpenShift Container Platform 集群满足以下要求：

- OpenShift Container Platform 必须首先 为 Azure Disk 配置。
- 基础架构中的每个节点主机必须与 Azure 虚拟机名称匹配。
- 每个节点主机必须位于同一资源组中。

27.10.3. 置备

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。确保为 Azure Disk 配置 OpenShift Container Platform 后，OpenShift Container Platform 和 Azure 需要所有这些都是 Azure Disk Name 和 Disk URI 和 **PersistentVolume** API。

27.10.4. 为区域云配置 Azure 磁盘

Azure 有多个要部署实例的区域。要指定所需的区域，请在 *azure.conf* 文件中添加以下内容：

```
cloud: <region>
```

区域可以是以下任意一种：

- German cloud:**AZUREGERMANCLOUD**
- China cloud:**AZURECHINACLOUD**
- Public cloud:**AZUREPUBLICCLOUD**
- US cloud:**AZUREUSGOVERNMENTCLOUD**

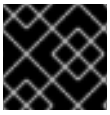
27.10.4.1. 创建持久性卷

您必须在对象定义中定义持久性卷，然后才能在 OpenShift Container Platform 中创建它：

例 27.8. 使用 Azure 的持久性卷对象定义

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
    storage: "5Gi" 2
  accessModes:
    - "ReadWriteOnce"
  azureDisk: 3
    diskName: test2.vhd 4
    diskURI: https://someacount.blob.core.windows.net/vhds/test2.vhd 5
    cachingMode: ReadWrite 6
    fsType: ext4 7
    readOnly: false 8
```

1. 卷的名称。这将通过 持久性卷声明或 从 pod 识别它。
2. 为这个卷分配的存储量。
3. 这将定义正在使用的卷类型（本例中为**azureDisk** 插件）。
4. blob 存储中的数据磁盘的名称。
5. blob 存储中的数据磁盘的 URI。
6. 主机缓存模式：none、ReadOnly 或 ReadWrite.
7. 要挂载的文件系统类型（例如 **ext** 4、**xfs** 等等）。
8. 默认为 **false** （读/写）。在此处阅读将强制使用 **VolumeMounts** 中的 **ReadOnly** 设置。



重要

在格式化并置备卷后更改 **fsType** 参数的值可能会导致数据丢失和 pod 失败。

1. 将定义保存到文件中，如 ***azure-pv.yaml*** 并创建持久性卷：

```
# oc create -f azure-pv.yaml
persistentvolume "pv0001" created
```

2. 验证持久性卷是否已创建：

```
# oc get pv
NAME      LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM     REASON
AGE
pv0001    <none>    5Gi        RWO           Available                 2s
```

现在，您可以使用持久性卷声明请求存储，该声明现在可以使用您的新持久性卷。

重要

对于通过 Azure 磁盘 PVC 挂载卷的 pod，将 pod 调度到新节点需要几分钟。等待两分钟完成 *Disk Detach* 操作，然后启动新的部署。如果在完成 *Disk Detach* 操作前启动新的 pod 创建请求，则 pod 创建启动的 *Disk Attach* 操作会失败，从而导致 pod 创建失败。

重要

持久性卷声明只在用户的命名空间中存在，且只能被同一命名空间中的 pod 引用。任何尝试从其他命名空间中访问持久性卷都会导致 pod 失败。

27.10.4.2. 卷格式

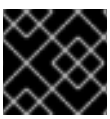
在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查它是否包含由持久性卷定义中的 **fstype** 参数指定的文件系统。如果没有使用文件系统格式化该设备，该设备中的所有数据都会被删除，并使用指定的文件系统自动格式化该设备。

这允许未格式化的 Azure 卷用作持久性卷，因为 OpenShift Container Platform 在第一次使用前会对其进行格式化。

27.11. 使用 AZURE FILE 的持久性存储

27.11.1. 概述

OpenShift Container Platform 支持 Microsoft Azure File 卷。您可以使用 Azure 为 OpenShift Container Platform 集群置备持久性存储。我们假设您对 Kubernetes 和 Azure 有一定的了解。



重要

存储的高可用性功能由底层的存储架构提供。

27.11.2. 开始前

1. 在所有节点上安装 **samba-client**、**samba-common** 和 **cifs-utils** ：

```
$ sudo yum install samba-client samba-common cifs-utils
```

2. 在所有节点上启用 SELinux 布尔值：

```
$ /usr/sbin/setsebool -P virt_use_samba on
$ /usr/sbin/setsebool -P virt_sandbox_use_samba on
```

3. 运行 **mount** 命令检查 **dir_mode** 和 **file_mode** 权限，例如：

```
$ mount
```

如果 **dir_mode** 和 **file_mode** 权限被设置为 **0755**，则将默认值 **0755** 更改为 **0777** 或 **0775**。需要此手动步骤，因为 OpenShift Container Platform 3.9 中默认的 **dir_mode** 和 **file_mode** 权限从 **0777** 更改为 **0755**。以下示例显示了带有更改值的配置文件。

使用 Azure File 时的注意事项

Azure File 不支持以下文件系统功能：

- 符号链接
- 硬链接
- 扩展属性
- 稀疏文件
- 命名管道

另外，Azure File 挂载目录的所有者用户标识符 (UID) 与容器的进程 UID 不同。

小心

如果您使用任何使用不支持的文件系统功能的容器镜像，则环境中的不稳定。PostgreSQL 和 MySQL 的容器已知在与 Azure File 搭配使用时出现问题。

在 Azure File 中使用 MySQL 的方法

如果使用 MySQL 容器，您必须修改 PV 配置，作为在挂载的目录 UID 和容器进程 UID 间的文件所有权不匹配的一个临时解决方案。对 PV 配置文件进行以下更改：

1. 在 PV 配置文件的 **runAsUser** 变量中指定 Azure File 挂载的目录 UID：

```
spec:
  containers:
    ...
  securityContext:
    runAsUser: <mounted_dir_uid>
```

2. 在 PV 配置文件中，指定 **mountOptions** 下的容器进程 UID：

```
mountOptions:
  - dir_mode=0700
  - file_mode=0600
```

```
- uid=<container_process_uid>
- gid=0
```

27.11.3. 配置文件示例

以下示例配置文件显示使用 Azure File 的 PV 配置：

PV 配置文件示例

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "azpv"
spec:
  capacity:
    storage: "1Gi"
  accessModes:
    - "ReadWriteMany"
  azureFile:
    secretName: azure-secret
    shareName: azftest
    readOnly: false
  mountOptions:
    - dir_mode=0777
    - file_mode=0777
```

以下示例配置文件显示使用 Azure File 的存储类：

存储类配置文件示例

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azurefile
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
parameters:
  storageAccount: ocp39str
  location: centralus
```

27.11.4. 为区域云配置 Azure File

虽然 Azure Disk 与多个区域云兼容，但 Azure File 只支持 Azure 公共云，因为端点是硬编码的。

27.11.5. 创建 Azure Storage Account secret

在 secret 配置中定义 Azure Storage Account 名称和密钥，然后将其转换为 base64，供 OpenShift Container Platform 使用。

1. 获取 Azure Storage Account 名称和密钥，并编码到 base64：

```
apiVersion: v1
kind: Secret
metadata:
  name: azure-secret
type: Opaque
data:
  azurestorageaccountname: azhzdGVzdA==
  azurestorageaccountkey:
eElGMXpKYm5ub2pGTE1Ta0JwNTBteDAyckhzTUsyc2pVN21GdDRMMTNob0l3ZHJBYUo4a
kQ2K0E0NDNqSm9nVjd5MkZVT2hRQ1dQbU02WWFOSHk3cWc9PQ==
```

2. 将 secret 定义保存到文件中，如 ***azure-secret.yaml***，然后创建 secret：

```
$ oc create -f azure-secret.yaml
```

3. 验证是否已创建 secret：

```
$ oc get secret azure-secret
NAME         TYPE     DATA     AGE
azure-secret   Opaque    1        23d
```

4. 在 OpenShift Container Platform 中创建前，在对象定义中定义 PV：

使用 Azure File 示例的 PV 对象定义

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" (1)
spec:
  capacity:
    storage: "5Gi" (2)
  accessModes:
    - "ReadWriteMany"
  azureFile: (3)
    secretName: azure-secret (4)
    shareName: example (5)
    readOnly: false (6)
```

(1) 卷的名称。这是如何通过 PV 声明 或 pod 识别它。

(2) 为这个卷分配的存储量。

(3) 这将定义正在使用的卷类型： **azureFile** 插件。

(4) 使用的 secret 的名称。

(5) 文件共享的名称。

(6) 默认为 **false** （读/写）。此处的 **ReadOnly** 用于强制 **VolumeMount** 中的 **ReadOnly** 设置。

5. 将定义保存到文件中，如 ***azure-file-pv.yaml*** 并创建 PV：

```
$ oc create -f azure-file-pv.yaml
persistentvolume "pv0001" created
```

6. 确定创建了 PV：

```
$ oc get pv
NAME     LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM     REASON
AGE
pv0001   <none>    5Gi        RWM           Available                 2s
```

现在，您可以使用 PV 声明来请求存储，该声明现在可以使用您的新 PV。

重要

PV 声明仅存在于用户的命名空间中，且只能被同一命名空间中的 pod 引用。尝试从不同命名空间中访问 PV 会导致 pod 失败。

27.12. 使用 FLEXVOLUME 插件的持久性存储

27.12.1. 概述

OpenShift Container Platform 内置卷插件 来使用不同的存储技术。要从没有内置插件的后端使用存储，您可以通过 FlexVolume 驱动程序来扩展 OpenShift Container Platform，并为应用程序 提供持久性存储。

27.12.2. FlexVolume 驱动程序

FlexVolume 驱动程序是一个可执行文件，它位于集群中的所有机器上（master 和节点）上明确定义的目录中。OpenShift Container Platform 会在需要附加、分离、挂载或卸载由带有 **flexVolume** 的 **PersistentVolume** 代表的卷时调用 FlexVolume 驱动程序。

驱动程序的第一个命令行参数始终是一个操作名称。其他参数都针对于每个操作。大多数操作都使用 JSON 字符串作为参数。这个参数是一个完整的 JSON 字符串，而不是包括 JSON 数据的文件名称。

FlexVolume 驱动程序包含：

- 所有 **flexVolume.options**。
- **flexVolume** 的一些选项带有 **kubernetes.io/**前缀，如 **fsType** 和 **readwrite**。
- 如果使用 secret，secret 的内容带有 **kubernetes.io/secret/** 前缀。

FlexVolume 驱动程序 JSON 输入示例

```
{
 "fooServer": "192.168.0.1:1234", 1
       "fooVolumeName": "bar",
 "kubernetes.io/fsType": "ext4", 2
 "kubernetes.io/readwrite": "ro", 3
 "kubernetes.io/secret/<key name>": "<key value>", 4
 "kubernetes.io/secret/<another key name>": "<another key value>",
}
```

1. **flexVolume.options** 中的所有选项。
2. **flexVolume.fsType** 的值。
3. 基于 **flexVolume.readOnly** 的 **ro**/**rw**。
4. 由 **flexVolume.secretRef**引用的 secret 的所有键及其值。

OpenShift Container Platform 需要有关驱动程序标准输出的 JSON 数据。如果没有指定，输出会描述操作的结果。

FlexVolume 驱动程序默认输出

```
{
 "status": "<Success/Failure/Not supported>",
 "message": "<Reason for success/failure>"
}
```

驱动程序的退出代码应该为 **0**（成功），或 **1**（失败）。

操作应该是幂等的，这意味着附加已附加的卷或已挂载的卷的挂载应该可以成功操作。

FlexVolume 驱动程序以两种模式工作：

- 具有 master-initated attach/detach 操作，或者
- 没有 master 启动的 attach/detach 操作。

OpenShift Container Platform master 使用 **attach**/**detach** 操作将卷附加到节点，并从节点分离它。当节点因任何原因而变得无响应时，这很有用。然后，master 可以终止节点上的所有 pod，将所有卷从它分离，并将卷附加到其他节点来恢复应用程序，同时仍可访问原始节点。

重要

并非所有存储后端都支持从另一台机器执行卷的主发起断开。

27.12.2.1. 带有 master-initiated attach/detach 的 FlexVolume 驱动程序

支持由 master 控制的 attach/detach 的 FlexVolume 驱动程序必须实现以下操作：

init

初始化驱动程序。它在 master 和节点初始化过程中被调用。

- 参数: 无
- 执行于：master、node
- 预期输出：默认 JSON

getvolumename

返回卷的唯一名称。此名称在所有 master 和节点上都一致，因为它在后续的 **detach** 调用中使用为 <**volume-name**>。< **volume-name** > 中的任何 / 字符将自动替换为 ~。

- 参数：**<json>**

- 执行于：master、node
- 预期输出： 默认 JSON + **volumeName** ：

```
{
 "status": "Success",
 "message": "",
 "volumeName": "foo-volume-bar" 1
}
```

1 存储后端 **foo** 中卷的唯一名称。

attach

将 JSON 代表的卷附加到给定节点。此操作应返回节点上设备名称（如果已知），即，如果在运行之前由存储后端分配了该设备。如果设备未知，则必须通过后续 **waitforattach** 操作在节点上找到该设备。

- 参数： &lt **;json> <node-name>**
- 执行于：master
- 预期输出：默认 JSON + 设备 （如果已知）：

```
{
 "status": "Success",
 "message": "",
 "device": "/dev/xvda" 1
}
```

1 如果已知，则节点上的设备名称。

waitforattach

等待卷完全附加到节点及其设备发生。如果前面的 **附加** 操作返回了 &lt **;device-name** >，它将作为输入参数提供。否则，**<device-name** > 为空，操作必须查找节点上的设备。

- 参数： &lt **;device-name> &lt ;json>**
- 执行于：节点
- 预期输出：默认 JSON + 设备

```
{
 "status": "Success",
 "message": "",
 "device": "/dev/xvda" 1
}
```

1 节点上的设备名称。

detach

将给定卷从节点分离。**<volume-name>** 是 **getvolumename** 操作返回的设备名称。< **volume-name** > 中的任何 / 字符将自动替换为 ~。

- 参数： &lt **;volume-name> & lt;node-name>**
- 执行于：master
- 预期输出：默认 JSON

isattached

检查卷是否已附加到节点。

- 参数： &lt **;json> <node-name>**
- 执行于：master
- 预期输出： **附加**默认 JSON +

```
{
 "status": "Success",
 "message": "",
 "attached": true (1)
}
```

1 将卷的状态附加到节点。

mountdevice

将卷的设备挂载到目录。**<device-name>** 是上一个 **waitforattach** 操作返回的设备名称。

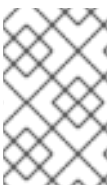
- 参数： &lt **;mount-dir> &lt ;device-name> &lt ;json>**
- 执行于：节点
- 预期输出：默认 JSON

unmountdevice

从目录中卸载卷的设备。

- 参数: **<mount-dir>**
- 执行于：节点

所有其他操作都应该返回带有 **{"status"的 JSON：**"不支持"} 和退出代码 **1**。



注意

master 启动的 attach/detach 操作会被默认启用。如果没有启用，则附加/组操作由卷应该附加到或从中分离的节点启动。在这两种情况下，Flex 驱动程序调用的语法和所有参数都相同。

27.12.2.2. 不使用 master-initiated attach/detach 的 FlexVolume 驱动程序

不支持 master 控制的 attach/detach 的 FlexVolume 驱动程序仅在节点上执行，且必须实现这些操作：

init

初始化驱动程序。它会在初始化所有节点的过程中被调用。

- 参数: 无
- 执行于：节点
- 预期输出：默认 JSON

mount

挂载一个卷到目录。这可包括挂载卷所需的任何内容，包括将卷附加到节点，查找其设备，然后挂载该设备。

- 参数: **<mount-dir> <json>**
- 执行于：节点
- 预期输出：默认 JSON

unmount

从目录中卸载卷。这可包括在卸载后清除卷所必需的任何内容，比如将卷从节点分离。

- 参数: **<mount-dir>**
- 执行于：节点
- 预期输出：默认 JSON

所有其他操作都应该返回带有 **{"status"的 JSON："不支持"}** 和退出代码 **1**。

27.12.3. 安装 FlexVolume 驱动程序

安装 FlexVolume 驱动程序：

1. 确保可执行文件存在于集群中的所有 master 和节点上。
2. 将可执行文件放在卷插件路径： */usr/libexec/kubernetes/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>*.

例如，要为存储 **foo** 安装 FlexVolume 驱动程序，请将可执行文件放在：*/usr/libexec/kubernetes/kubelet-plugins/volume/exec/openshift.com~foo/foo*。

在 OpenShift Container Platform 3.11 中，因为 **controller-manager** 作为静态 pod 运行，执行 attach 和 detach 操作的 FlexVolume 二进制文件必须是自包含的可执行文件，且没有外部依赖项。

在 Atomic 主机上，FlexVolume 插件目录的默认位置为 */etc/origin/kubelet-plugins/*。您必须将 FlexVolume 可执行文件放在集群中所有 master 和节点上的 */etc/origin/kubelet-plugins/volume/exec/<vendor* > 目录中。

27.12.4. 使用 FlexVolume 驱动程序消耗存储

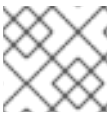
使用 **PersistentVolume** 对象来引用已安装的存储。OpenShift Container Platform 中的每个 **PersistentVolume** 都代表一个存储资产，通常是存储后端中的一个卷。

使用 FlexVolume 驱动程序示例定义持久性卷对象

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 1Gi 2
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: openshift.com/foo 3
    fsType: "ext4" 4
    secretRef: foo-secret 5
    readOnly: true 6
    options: 7
      fooServer: 192.168.0.1:1234
      fooVolumeName: bar
```

1. 卷的名称。这是如何通过持久性卷声明或 从 pod 识别它。这个名称可以与后端存储中的卷的名称不同。
2. 为这个卷分配的存储量。
3. 驱动程序的名称。这个字段是必须的。
4. 卷中的文件系统。这个字段是可选的。
5. 对 secret 的引用。此 secret 中的键和值在调用时会提供给 FlexVolume 驱动程序。这个字段是可选的。
6. read-only 标记。这个字段是可选的。
7. FlexVolume 驱动程序的额外选项。除了用户在 **options** 字段中指定的标记外，以下标记还会传递给可执行文件：

```
"fsType":"<FS type>",
"readwrite":"<rw>",
"secret/key1":"<secret1>"
...
"secret/keyN":"<secretN>"
```



注意

secret 只会传递给 mount/mount call-outs。

27.13. 使用 VMWARE VSPHERE 卷进行持久性存储

27.13.1. 概述

OpenShift Container Platform 支持 VMware vSphere 的虚拟机磁盘(VMDK)卷。您可以使用 VMware vSphere 为 OpenShift Container Platform 集群置备持久性存储。我们假设您对 Kubernetes 和 VMware vSphere 已有一定了解。

OpenShift Container Platform 在 vSphere 中创建磁盘，并将磁盘附加到正确的实例。

OpenShift Container Platform 持久性卷(PV) 框架允许管理员置备带有持久性存储的集群，并为用户提供在不了解底层基础架构的情况下请求这些资源的方法。vSphere VMDK 卷 可以动态置备。

PV 不绑定到单个项目或命名空间，它们可以在 OpenShift Container Platform 集群间共享。但是，PV 声明 是特定于项目或命名空间的，用户可请求它。

重要

存储的高可用性功能由底层的存储架构提供。

先决条件

在使用 vSphere 创建 PV 前，请确保 OpenShift Container Platform 集群满足以下要求：

- OpenShift Container Platform 必须首先 为 vSphere 配置。
- 基础架构中的每个节点主机必须与 vSphere 虚拟机名称匹配。
- 每个节点主机必须位于同一资源组中。

27.13.2. 动态置备 VMware vSphere 卷

动态置备 VMware vSphere 卷是首选的置备方法。

1. 如果您在置备集群时没有指定 Ansible 清单文件中的 **openshift_cloudprovider_kind=vsphere** 和 **openshift_vsphere_*** 变量，您必须手动创建以下 **StorageClass** 来使用 **vsphere-volume** 置备程序：

```
$ oc get --export storageclass vsphere-standard -o yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: "vsphere-standard" (1)
provisioner: kubernetes.io/vsphere-volume (2)
parameters:
    diskformat: thin (3)
    datastore: "YourvSphereDatastoreName" (4)
reclaimPolicy: Delete
```

(1) StorageClass 的名称。

(2) 存储置备程序类型。指定 **vsphere-volume**。

(3) 磁盘类型。指定零个 **或 精简**。

(4) 创建磁盘的源数据存储。

2. 在使用上一步中显示的 StorageClass 请求 PV 后，OpenShift Container Platform 会在 vSphere 基础架构中创建 VMDK 磁盘。要验证磁盘是否已创建，请在 vSphere 中使用 Datastore 浏览器。

注意

vSphere-volume 磁盘是 **ReadWriteOnce** 访问模式，这意味着该卷可以被单一节点以读写模式挂载。如需更多信息，请参阅架构指南中的访问模式部分。

27.13.3. 静态置备 VMware vSphere 卷

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。确保为 vSphere 配置 OpenShift Container Platform 后，OpenShift Container Platform 和 vSphere 都需要一个虚拟机文件夹路径、文件系统类型和 **PersistentVolume** API。

27.13.3.1. 创建 VMDK

重要

使用以下任一方法创建 VMDK。

- 使用 **vmkfstools** 创建：
  通过 Secure Shel (SSH) 访问 ESX，然后使用以下命令创建 VMDK 卷：

```
vmkfstools -c 40G /vmfs/volumes/DatastoreName/volumes/myDisk.vmdk
```

- 使用 **vmware-vdiskmanager** 创建：

```
shell vmware-vdiskmanager -c -t 0 -s 40GB -a lsilogic myDisk.vmdk
```

27.13.3.2. 创建 PersistentVolume

1. 定义 PV 对象定义，如 *vsphere-pv.yaml* ：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 (1)
spec:
  capacity:
    storage: 2Gi (2)
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  vsphereVolume: (3)
    volumePath: "[datastore1] volumes/myDisk" (4)
    fsType: ext4 (5)
```

(1) 卷的名称。这必须是如何通过 PV 声明或从 pod 识别它。

(2) 为这个卷分配的存储量。

(3) 使用的卷类型。这个示例使用 **vsphereVolume**。此标签用于将 vSphere VMDK 卷挂载到 Pod 中。卸载卷时会保留卷内容。卷类型支持 VMFS 和 VSAN 数据存储。

(4) 要使用的现有 VMDK 卷。在卷定义中，数据存储名称必须放在方括号([])中，如下所示。

5 要挂载的文件系统类型。例如： **ext** 4、**xfs** 或者其它文件系统。

重要

在格式化并置备卷后更改 **fsType** 参数的值可能会导致数据丢失和 pod 失败。

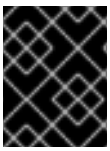
2. 创建 PV：

```
$ oc create -f vsphere-pv.yaml
  persistentvolume "pv0001" created
```

3. 确定创建了 PV：

```
$ oc get pv
NAME    LABELS  CAPACITY  ACCESSMODES  STATUS     CLAIM   REASON  AGE
pv0001  <none>  2Gi       RWO          Available          2s
```

现在，您可以使用 PV 声明来请求存储，该声明现在可以使用 PV。



重要

PV 声明仅存在于用户的命名空间中，且只能被同一命名空间中的 pod 引用。尝试从不同命名空间中访问 PV 会导致 pod 失败。

27.13.3.3. 格式化 VMware vSphere 卷

在 OpenShift Container Platform 挂载卷并将其传递给容器之前，它会检查卷是否包含由 PV 定义中 **fsType** 参数指定的文件系统。如果没有使用文件系统格式化该设备，该设备中的所有数据都会被删除，并使用指定的文件系统自动格式化该设备。

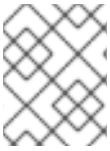
因为 OpenShift Container Platform 在首次使用卷前会进行格式化，所以您可以使用未格式化的 vSphere 卷作为 PV。

27.14. 使用本地卷的持久性存储

27.14.1. 概述

OpenShift Container Platform 集群可以使用本地卷来置备持久性存储。本地持久性卷允许您使用标准 PVC 接口访问本地存储设备，如磁盘、分区或目录。

无需手动将 pod 调度到节点即可使用本地卷，因为系统了解卷的节点限制。但是，本地卷仍会受到底层节点可用性的影响，而且并不适用于所有应用程序。



注意

本地卷是一个 alpha 功能，可能会在以后的 OpenShift Container Platform 发行版本中改变。有关已知问题和临时解决方案的详情，请参阅 功能状态（本地卷） 部分。

警告

本地卷只能用作静态创建的持久性卷。

27.14.2. 置备

当存储可以被挂载为 OpenShift Container Platform 中的卷之前，它必须已存在于底层的存储系统中。在使用 **PersistentVolume** API 前，请确保为 本地卷 配置了 OpenShift Container Platform。

27.14.3. 创建本地持久性卷

在对象定义中定义持久性卷。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-local-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /mnt/disks/ssd1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - my-node
```

27.14.4. 创建本地持久性卷声明

在对象定义中定义持久性卷声明。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-local-claim
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi (1)
  storageClassName: local-storage (2)
```

■

1 存储卷所需的大小。

2 存储类的名称，用于本地 PV。

27.14.5. 功能状态

what Works：

- 通过指定具有节点关联性的目录来创建 PV。
- 使用与前面提到的 PV 绑定的 PVC 的 Pod 始终会被调度到该节点。
- 用于发现本地目录、创建、清理和删除 PV 的外部静态置备程序 daemonset。

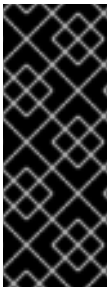
无法正常工作：

- 单个 pod 中的多个本地 PVC。
- PVC 绑定不考虑 pod 调度要求，并可能会做出子优化或不正确的决定。
  - 临时解决方案：
    - 首先运行这些 pod，这需要本地卷。
    - 为 pod 授予高优先级。
    - 运行一个临时解决方案控制器，为卡住的 pod 取消绑定 PVC。
- 如果在外部置备程序启动后添加了挂载，则外部置备程序无法检测到挂载的正确功能。
  - 临时解决方案：
    - 在添加新挂载点之前，首先停止 daemonset，添加新挂载点，然后启动 daemonset。
- 如果多个使用相同 PVC 的 pod 指定不同的 **fsgroup** 的，则 FSGroup 冲突。

27.15. 使用容器存储接口 (CSI) 的持久性存储

27.15.1. 概述

容器存储接口(CSI)允许 OpenShift Container Platform 使用支持 CSI 接口的存储后端提供 的持久性存储。



重要

CSI 卷当前还处于技术预览阶段，不适用于生产环境中的工作负载。CSI 卷可能会在以后的版本中更改。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

请参阅 link:https://access.redhat.com/support/offerings/techpreview/[Red Hat

注意

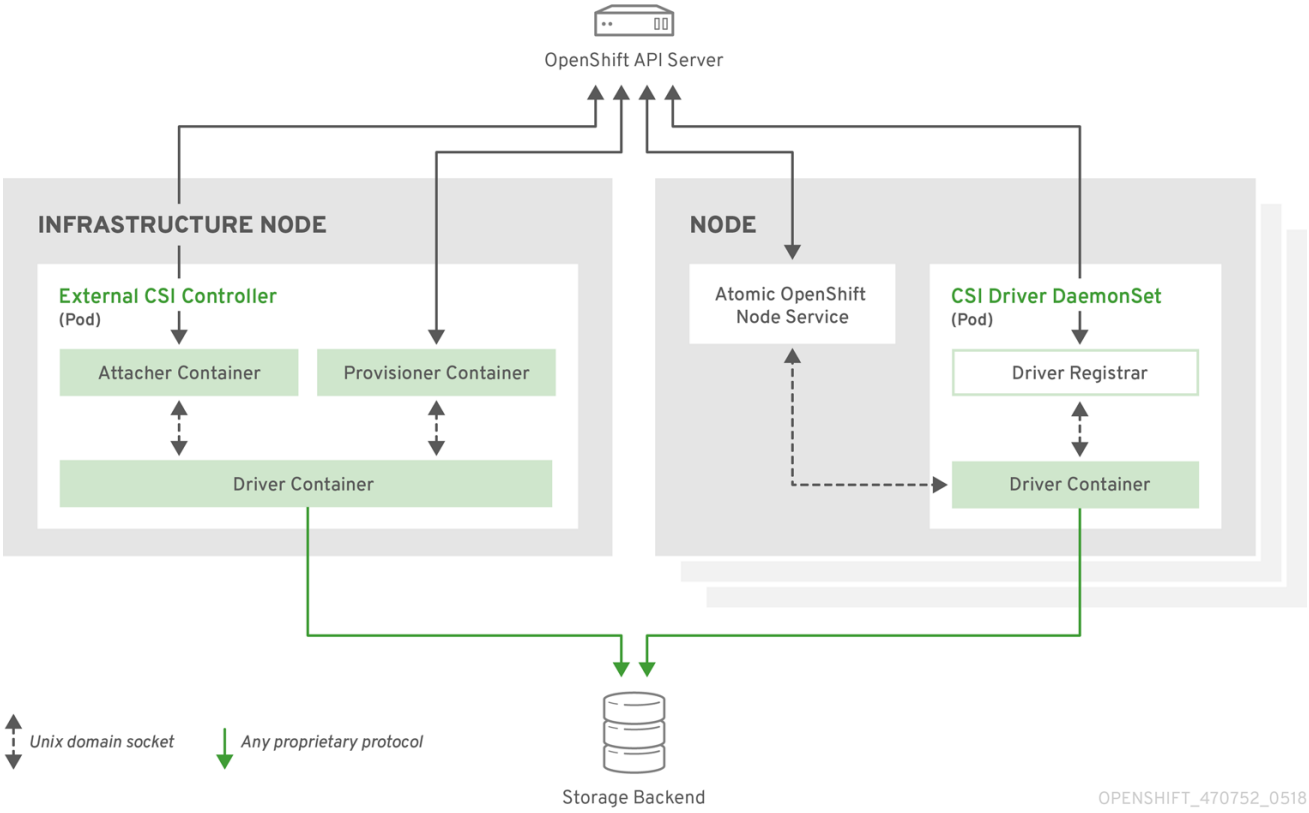
OpenShift Container Platform 不附带任何 CSI 驱动程序。建议您使用由开源社区或存储供应商提供的 CSI 驱动程序。

OpenShift Container Platform 3.11 支持 CSI 规范 版本 0.2.0。

27.15.2. 架构

CSI 驱动程序通常由容器镜像提供。这些容器不了解其运行的 OpenShift Container Platform。要在 OpenShift Container Platform 中使用与 CSI 兼容的存储后端，集群管理员必须部署几个组件，作为 OpenShift Container Platform 和存储驱动程序间的桥接。

下图提供了在 OpenShift Container Platform 集群中以 pod 运行的组件的概述。



对于不同的存储后端，可以运行多个 CSI 驱动程序。每个驱动程序需要其自身的外部控制器部署，以及带驱动程序和 CSI 注册器的 DaemonSet。

27.15.2.1. 外部 CSI Controller

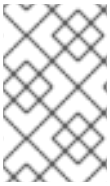
外部 CSI 控制器是一个部署，它部署带有以下三个容器的一个或多个 pod：

- 将 OpenShift Container Platform 的 **attach** 和 **detach** 调用转换为相应的 **ControllerPublish** 和 **ControllerUnpublish** 调用 CSI 驱动程序的外部 CSI attacher 容器
- 将 OpenShift Container Platform 的 **provision** 和 **delete** 调用转换为 CSI 驱动程序的 **CreateVolume** 和 **DeleteVolume** 调用的外部 CSI 置备程序容器
- CSI 驱动程序容器

CSI attacher 和 CSI provisioner 容器使用 UNIX 域套接字与 CSI 驱动程序容器对话，确保没有 CSI 通信。从 pod 以外无法访问 CSI 驱动程序。

注意

attach、 **detach**、 **provision**和 **delete** 操作通常需要 CSI 驱动程序在存储后端使用凭证。在 infrastructure 节点上运行 CSI controller pod，因此即使在一个计算节点上发生严重的安全破坏时，凭据也不会暴露给用户进程。



注意

对于不支持第三方的 attach/detach 操作的 CSI 驱动程序，还必须运行外部附加器。外部附加器不会向 CSI 驱动程序发出任何 **ControllerPublish** 或 **ControllerUnpublish** 操作。然而，它仍必须运行方可实现所需的 OpenShift Container Platform attachment API。

27.15.2.2. CSI Driver DaemonSet

最后，CSI 驱动程序 DaemonSet 在每个节点上运行一个 pod，它允许 OpenShift Container Platform 挂载 CSI 驱动程序提供的存储，并使用它作为持久性卷 (PV) 的用户负载 (pod) 。安装了 CSI 驱动程序的 pod 包含以下容器：

- CSI 驱动程序注册器，它会在节点上运行的 **openshift-node** 服务中注册 CSI 驱动程序。在节点上运行的 **openshift-node** 进程然后使用节点上可用的 UNIX 域套接字直接连接到 CSI 驱动程序。
- CSI 驱动程序。

在节点上部署的 CSI 驱动程序应该在存储后端上有尽量少的凭证。OpenShift Container Platform 只使用节点插件的 CSI 调用集合，如 **NodePublish/NodeUnpublish** 和 **NodeStage/NodeUnstage** （如果已实施）。

27.15.3. Deployment 示例

由于 OpenShift Container Platform 不附带任何 CSI 驱动程序，本例演示了如何在 OpenShift Container Platform 中部署 OpenStack Cinder 的社区驱动程序。

1. 创建一个新项目，运行 CSI 组件以及运行组件的新服务帐户。显式节点选择器使用 CSI 驱动程序在 master 节点上运行 Daemonset。

```
# oc adm new-project csi --node-selector=""
Now using project "csi" on server "https://example.com:8443".

# oc create serviceaccount cinder-csi
serviceaccount "cinder-csi" created

# oc adm policy add-scc-to-user privileged system:serviceaccount:csi:cinder-csi
scc "privileged" added to: ["system:serviceaccount:csi:cinder-csi"]
```

2. 应用此 YAML 文件，以使用外部 CSI attacher 和 provisioner 和 DaemonSet 使用 CSI 驱动程序来创建部署。

```
# This YAML file contains all API objects that are necessary to run Cinder CSI
# driver.
#
# In production, this needs to be in separate files, e.g. service account and
# role and role binding needs to be created once.
```

```
#
# It serves as an example of how to use external attacher and external provisioner
# images that are shipped with OpenShift Container Platform with a community CSI driver.

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cinder-csi-role
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["create", "delete", "get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["volumeattachments"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]

---

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cinder-csi-role
subjects:
  - kind: ServiceAccount
    name: cinder-csi
    namespace: csi
roleRef:
  kind: ClusterRole
  name: cinder-csi-role
  apiGroup: rbac.authorization.k8s.io

---
apiVersion: v1
data:
  cloud.conf:
W0dsb2JhbF0KYXV0aC11cmwgPSBodHRwczovL2V4YW1wbGUuY29tOjEzMDAwL3YyLjAvC
nVzZXJuYW1lID0gYWxhZGRpbgpwYXNzd29yZCA9IG9wZW5zZXNhbWUKdGVuYW50LWlkI
D0gZTBmYTg1YjZhMDY0NDM5NTlkMmQzYjQ5NzE3NGJlZDYKcmVnaW9uID0gcmVnaW9u
T25lCg== 1
kind: Secret
metadata:
```

```
  creationTimestamp: null
  name: cloudconfig
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: cinder-csi-controller
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cinder-csi-controllers
  template:
    metadata:
      labels:
        app: cinder-csi-controllers
    spec:
      serviceAccount: cinder-csi
      containers:
        - name: csi-attacher
          image: registry.redhat.io/openshift3/csi-attacher:v3.11
          args:
            - "--v=5"
            - "--csi-address=$(ADDRESS)"
            - "--leader-election"
            - "--leader-election-namespace=$(MY_NAMESPACE)"
            - "--leader-election-identity=$(MY_NAME)"
          env:
            - name: MY_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: MY_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: ADDRESS
              value: /csi/csi.sock
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: csi-provisioner
          image: registry.redhat.io/openshift3/csi-provisioner:v3.11
          args:
            - "--v=5"
            - "--provisioner=csi-cinderplugin"
            - "--csi-address=$(ADDRESS)"
          env:
            - name: ADDRESS
              value: /csi/csi.sock
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: cinder-driver
          image: quay.io/jsafrane/cinder-csi-plugin
          command: [ "/bin/cinder-csi-plugin" ]
```

```
          args:
            - "--nodeid=$(NODEID)"
            - "--endpoint=unix://$(ADDRESS)"
            - "--cloud-config=/etc/cloudconfig/cloud.conf"
          env:
            - name: NODEID
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: ADDRESS
              value: /csi/csi.sock
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
            - name: cloudconfig
              mountPath: /etc/cloudconfig
      volumes:
        - name: socket-dir
          emptyDir:
        - name: cloudconfig
          secret:
            secretName: cloudconfig

---

kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: cinder-csi-ds
spec:
  selector:
    matchLabels:
      app: cinder-csi-driver
  template:
    metadata:
      labels:
        app: cinder-csi-driver
    spec:
      2
      serviceAccount: cinder-csi
      containers:
        - name: csi-driver-registrar
          image: registry.redhat.io/openshift3/csi-driver-registrar:v3.11
          securityContext:
            privileged: true
          args:
            - "--v=5"
            - "--csi-address=$(ADDRESS)"
          env:
            - name: ADDRESS
              value: /csi/csi.sock
            - name: KUBE_NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
          volumeMounts:
```

```
        - name: socket-dir
          mountPath: /csi
      - name: cinder-driver
        securityContext:
          privileged: true
          capabilities:
            add: ["SYS_ADMIN"]
          allowPrivilegeEscalation: true
        image: quay.io/jsafrane/cinder-csi-plugin
        command: [ "/bin/cinder-csi-plugin" ]
        args:
          - "--nodeid=$(NODEID)"
          - "--endpoint=unix://$(ADDRESS)"
          - "--cloud-config=/etc/cloudconfig/cloud.conf"
        env:
          - name: NODEID
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: ADDRESS
            value: /csi/csi.sock
        volumeMounts:
          - name: socket-dir
            mountPath: /csi
          - name: cloudconfig
            mountPath: /etc/cloudconfig
          - name: mountpoint-dir
            mountPath: /var/lib/origin/openshift.local.volumes/pods/
            mountPropagation: "Bidirectional"
          - name: cloud-metadata
            mountPath: /var/lib/cloud/data/
          - name: dev
            mountPath: /dev
      volumes:
        - name: cloud-metadata
          hostPath:
            path: /var/lib/cloud/data/
        - name: socket-dir
          hostPath:
            path: /var/lib/kubelet/plugins/csi-cinderplugin
            type: DirectoryOrCreate
        - name: mountpoint-dir
          hostPath:
            path: /var/lib/origin/openshift.local.volumes/pods/
            type: Directory
        - name: cloudconfig
          secret:
            secretName: cloudconfig
        - name: dev
          hostPath:
            path: /dev
```

1 使用 **cloud.conf** 作为 OpenStack 部署，如 OpenStack 配置 中所述。例如，可以使用 **oc create secret generic cloudconfig --from-file cloud.conf --dry-run -o yaml** 生成 Secret。

2 另外，还可在 CSI 驱动程序 pod 模板中添加 **nodeSelector** 以配置 CSI 驱动程序启动的节点。只有与选择器匹配的节点运行使用 CSI 驱动程序提供的卷的 pod。如果没有

27.15.4. 动态置备

动态置备持久性存储取决于 CSI 驱动程序和底层存储后端的功能。CSI 驱动的供应商应该提供了在 OpenShift Container Platform 中创建 StorageClass 及进行配置的参数文档。

如 OpenStack Cinder 示例中所示，您可以部署此 StorageClass 以启用动态置备。以下示例会创建一个新的默认存储类，以确保所有不需要特殊存储类的 PVC 都由已安装的 CSI 驱动程序置备：

```
# oc create -f - << EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cinder
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi-cinderplugin
parameters:
EOF
```

27.15.5. 使用方法

部署 CSI 驱动程序后，会创建动态置备的 StorageClass，OpenShift Container Platform 就可以使用 CSI。以下示例在没有对模板进行任何更改的情况下安装一个默认的 MySQL 模板：

```
# oc new-app mysql-persistent
--> Deploying template "openshift/mysql-persistent" to project default
...

# oc get pvc
NAME            STATUS    VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
mysql           Bound     kubernetes-dynamic-pv-3271ffcb4e1811e8   1Gi        RWO           cinder
3s
```

27.16. 使用 OPENSTACK MANILA 的持久性存储

27.16.1. 概述

重要

使用 OpenStack Manila 置备持久性卷(PV)只是一个技术预览功能。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

如需红帽技术预览功能支持范围的更多信息，请参阅 https://access.redhat.com/support/offerings/techpreview/。

OpenShift Container Platform 可以使用 OpenStack Manila 共享文件系统服务置备的持久性卷（PV）。

它假定 OpenStack Manila 服务已被正确设置，并可从 OpenShift Container Platform 集群访问。只有 NFS 共享类型才能被置备。

建议您熟悉 PV、持久性卷声明(PVC)、动态置备 和 RBAC 授权的概念。

27.16.2. 安装及设置

该功能由外部置备程序提供。您必须在 OpenShift Container Platform 集群中安装和配置它。

27.16.2.1. 启动外部调配器

外部置备程序服务以容器镜像形式发布，并可像平常一样在 OpenShift Container Platform 集群中运行。

要允许容器管理 API 对象，请以集群管理员身份配置所需的基于角色的访问控制(RBAC)规则：

1. 创建一个 **ServiceAccount** ：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: manila-provisioner-runner
```

2. 创建 **ClusterRole** ：

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: manila-provisioner-role
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["list", "watch", "create", "update", "patch"]
```

3. 通过 **ClusterRoleBinding** 绑定规则：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: manila-provisioner
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: manila-provisioner-role
subjects:
```

```
- kind: ServiceAccount
  name: manila-provisioner-runner
  namespace: default
```

4. 创建新的 **StorageClass** ：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: "manila-share"
provisioner: "externalstorage.k8s.io/manila"
parameters:
  type: "default" (1)
  zones: "nova" (2)
```

(1) 置备程序将为卷创建 Manila 共享类型。

(2) 在其中创建卷的 Manila 可用区集。

使用环境变量将置备程序配置为连接、验证并授权给 Manila servic。从以下列表中为您的安装选择适当的组合：

```
OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_DOMAIN_NAME
OS_TENANT_NAME
```

```
OS_USERID
OS_PASSWORD
OS_AUTH_URL
OS_TENANT_ID
```

```
OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_DOMAIN_ID
OS_TENANT_NAME
```

```
OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_DOMAIN_ID
OS_TENANT_ID
```

要将变量传递给置备程序，请使用 **Secret**。以下示例显示了为第一个变量组合配置的 **Secret**

```
apiVersion: v1
kind: Secret
metadata:
  name: manila-provisioner-env
type: Opaque
```

```
data:
  os_username: <base64 encoded Manila username>
  os_password: <base64 encoded password>
  os_auth_url: <base64 encoded OpenStack Keystone URL>
  os_domain_name: <base64 encoded Manila service Domain>
  os_tenant_name: <base64 encoded Manila service Tenant/Project name>
```

注意

较新的 OpenStack 版本使用 "project" 而不是 "tenant"。 但是，置备程序使用的环境变量必须在其名称中使用 **TENANT**。

最后一步是启动 provisioner 本身，例如使用部署：

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: manila-provisioner
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: manila-provisioner
    spec:
      serviceAccountName: manila-provisioner-runner
      containers:
        - image: "registry.redhat.io/openshift3/manila-provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          name: manila-provisioner
          env:
            - name: "OS_USERNAME"
              valueFrom:
                secretKeyRef:
                  name: manila-provisioner-env
                  key: os_username
            - name: "OS_PASSWORD"
              valueFrom:
                secretKeyRef:
                  name: manila-provisioner-env
                  key: os_password
            - name: "OS_AUTH_URL"
              valueFrom:
                secretKeyRef:
                  name: manila-provisioner-env
                  key: os_auth_url
            - name: "OS_DOMAIN_NAME"
              valueFrom:
                secretKeyRef:
                  name: manila-provisioner-env
                  key: os_domain_name
            - name: "OS_TENANT_NAME"
              valueFrom:
```

```
        secretKeyRef:
          name: manila-provisioner-env
          key: os_tenant_name
```

27.16.3. 使用方法

在置备程序运行后，您可以使用 PVC 和相应的 StorageClass 来置备 PV：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: manila-nfs-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2G
  storageClassName: manila-share
```

然后，**PersistentVolumeClaim** 被绑定到新置备的 Manila 共享支持的 **PersistentVolume**。当 **PersistentVolumeClaim** 及其后 **PersistentVolume** 被删除时，置备程序会删除 Manila 共享并取消导出。

27.17. 动态置备和创建存储类

27.17.1. 概述

StorageClass 资源对象描述并分类了可请求的存储，并提供了根据需要为 **动态置备存储** 传递参数的方法。*StorageClass* 也可以作为控制不同级别的存储和访问存储的管理机制。集群管理员（**cluster-admin**）或者存储管理员（**storage-admin**）可以在无需了解底层存储卷资源的情况下，定义并创建用户可以请求的 *StorageClass* 对象。

OpenShift Container Platform 持久性卷框架 启用了此功能，并允许管理员使用持久性存储置备集群。该框架还可让用户在不了解底层存储架构的情况下请求这些资源。

很多存储类型都可用于 OpenShift Container Platform 中的持久性卷。虽然它们都可以由管理员静态置备，但有些类型的存储是使用内置供应商和插件 API 动态创建的。

注意

要启用动态置备，将 **openshift_master_dynamic_provisioning_enabled** 变量添加到 Ansible 清单文件的 **[OSEv3:vars]** 部分，并将其值设为 **True**。

```
[OSEv3:vars]

openshift_master_dynamic_provisioning_enabled=True
```

27.17.2. 可用的动态置备插件

OpenShift Container Platform *提供了以下* 置备程序插件，用于使用集群配置的供应商 API 创建新存储资源的动态置备：

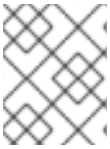
存储类型	provisioner 插件名称	所需配置	备注
OpenStack Cinder	**kubernetes.io/cinder**	为 OpenStack 配置	
AWS Elastic Block Store (EBS)	**kubernetes.io/aws-ebs**	为 AWS 配置	要在不同区中使用多个集群进行动态置备，使用 **Key=kubernetes.io/cluster/xxxx,Value=clusterid** 标记每个节点，其中 **xxxx** 和 **clusterid** 是每个集群唯一。
GCE 持久性磁盘 (gcePD)	**kubernetes.io/gce-pd**	为 GCE 配置	在多区(multi-zone)配置中，建议在每个 GCE 项目中运行一个 Openshift 集群，以避免在没有当前集群的区域中创建的 PV。
GlusterFS	**kubernetes.io/glusterfs**	配置 GlusterFS	
Ceph RBD	**kubernetes.io/rbd**	配置 Ceph RBD	
NetApp 中的 Trident	**netapp.io/trident**	为 Trident 配置	NetApp ONTAP、SolidFire 和 E-Series 存储的存储编排器。
VMWare vSphere	**kubernetes.io/vsphere-volume**	vSphere 和 Kubernetes 入门	
Azure Disk	**kubernetes.io/azure-disk**	为 Azure 配置	

重要

任何选择的置备程序插件还需要根据相关文档为相关的云、主机或者第三方供应商配置。

27.17.3. 定义 StorageClass

StorageClass 对象目前是一个全局范围的对象，需要由 **cluster-admin** 或 **storage-admin** 用户创建。



注意

对于 GCE 和 AWS，在 OpenShift Container Platform 安装过程中创建一个默认 StorageClass。您可以 更改默认 StorageClass 或删除它。

目前支持六个插件。以下小节描述了 *StorageClass* 以及每个支持的插件类型的具体示例的基本对象定义。

27.17.3.1. 基本 StorageClass 对象定义

StorageClass 基本对象定义

```
kind: StorageClass 1
apiVersion: storage.k8s.io/v1 2
metadata:
  name: foo 3
  annotations: 4
    ...
provisioner: kubernetes.io/plug-in-type 5
parameters: 6
  param1: value
  ...
  paramN: value
```

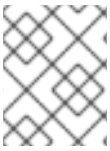
1. （必需）API 对象类型。
2. （必需）当前的 apiVersion。
3. （必需）StorageClass 的名称。
4. （可选）StorageClass 的注释
5. （必需）与这个存储类关联的置备程序类型。
6. （可选）特定置备程序所需的参数，这将根据插件的不同而有所不同。

27.17.3.2. StorageClass 注解（annotations）

将 *StorageClass* 设置为集群范围的默认值：

```
storageclass.kubernetes.io/is-default-class: "true"
```

这可让任何没有指定特定卷的 PVC 通过 *default* StorageClass 自动置备



注意

beta 注解 **storageclass.beta.kubernetes.io/is-default-class** 仍在工作。但是，它将在以后的发行版本中被删除。

设置 *StorageClass* 描述：

```
kubernetes.io/description: My StorageClass Description
```

27.17.3.3. OpenStack Cinder 对象定义

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
```

```
metadata:
  name: gold
provisioner: kubernetes.io/cinder
parameters:
  type: fast (1)
  availability: nova (2)
  fsType: ext4 (3)
```

1. 在 Cinder 中创建的卷类型。默认为空。
2. 可用区。如果没有指定可用区，则通常会在所有 OpenShift Container Platform 集群有节点的所有活跃区域间轮换选择。
3. 在动态部署卷中创建的文件系统。这个值被复制到动态配置的持久性卷的 **fstype** 字段中，并在第一个挂载卷时创建文件系统。默认值为 **ext4**。

27.17.3.4. AWS ElasticBlockStore(EBS)对象定义

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 (1)
  zone: us-east-1d (2)
  iopsPerGB: "10" (3)
  encrypted: "true" (4)
  kmsKeyId: keyvalue (5)
  fsType: ext4 (6)
```

1. 从 **io1**, **gp2**, **sc1**, **st1** 中选择。默认为 **gp2**。如需有效的 Amazon 资源名称(ARN)值，请参阅 AWS 文档。
2. AWS 区域。如果没有指定任何区，则通常会在所有 OpenShift Container Platform 集群有节点的活跃区域间轮换选择。zone 和 zones 参数不能同时使用。
3. 只适用于 **io1** 卷。每个 GiB 每秒一次 I/O 操作。AWS 卷插件乘以这个值，再乘以请求卷的大小以计算卷的 IOPS。数值上限为 20,000 IOPS，这是 AWS 支持的最大值。详情请查看 AWS 文档。
4. 表示是否加密 EBS 卷。有效值为 **true** 或者 **false**。
5. 可选。加密卷时要使用的密钥的完整 ARN。如果没有提供任何信息，但 **encypted** 被设置为 **true**，则 AWS 会生成一个密钥。有效 ARN 值请查看 AWS 文档。
6. 在动态部署卷中创建的文件系统。这个值被复制到动态配置的持久性卷的 **fstype** 字段中，并在第一个挂载卷时创建文件系统。默认值为 **ext4**。

27.17.3.5. GCE PersistentDisk (gcePD) 对象定义

gce-pd-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard (1)
  zone: us-central1-a (2)
  zones: us-central1-a, us-central1-b, us-east1-b (3)
  fsType: ext4 (4)
```

1. 选择 **pd-standard** 或 **pd-ssd**。默认为 **pd-ssd**。
2. GCE 区域。如果没有指定任何区，则通常会在所有 OpenShift Container Platform 集群有节点的活跃区域间轮换选择。zone 和 zones 参数不能同时使用。
3. GCE 区的逗号分隔列表。如果没有指定任何区，则通常会在所有 OpenShift Container Platform 集群有节点的活跃区域间轮换选择。zone 和 zones 参数不能同时使用。
4. 在动态部署卷中创建的文件系统。这个值被复制到动态配置的持久性卷的 **fstype** 字段中，并在第一个挂载卷时创建文件系统。默认值为 **ext4**。

27.17.3.6. GlusterFS 对象定义

glusterfs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/glusterfs
parameters: (1)
  resturl: http://127.0.0.1:8081 (2)
  restuser: admin (3)
  secretName: heketi-secret (4)
  secretNamespace: default (5)
  gidMin: "40000" (6)
  gidMax: "50000" (7)
  volumeoptions: group metadata-cache, nl-cache on (8)
  volumetype: replicate:3 (9)
  volumenameprefix: custom (10)
```

1. 列出是必须的，以及几个可选参数。如需了解更多参数，请参阅 注册 存储类。
2. heketi （用于 Gluster 的卷管理 REST 服务）URL，可按需置备 GlusterFS 卷。常规格式应为 **{http/https}://{IPaddress}:{Port}**。这是 GlusterFS 动态置备程序强制参数。如果 heketi 服务作为可路由的服务在 OpenShift Container Platform 中公开，它将具有可解析的完全限定域名 (FQDN) 和 heketi 服务 URL。
3. heketi 用户有权访问创建卷。通常为 "admin"。

4 识别包含与 heketi 对话时使用的用户密码的 Secret。可选；当省略 **secretNamespace** 和 **secretName** 时，将使用空密码。提供的 secret 必须是 **"kubernetes.io/glusterfs"** 类型。

5 上述 **secretName** 的命名空间。可选；当省略 **secretNamespace** 和 **secretName** 时，将使用空密码。提供的 secret 必须是 **"kubernetes.io/glusterfs"** 类型。

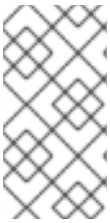
6 可选。这个 StorageClass 的卷的 GID 范围最小值。

7 可选。这个 StorageClass 的卷的 GID 范围的最大值。

8 可选。新创建的卷的选项。它允许性能调整。如需了解更多 GlusterFS 卷选项，请参阅调整 卷选项。

9 可选。要使用的卷类型。

10 可选。启用自定义卷名称支持，其格式如下：**<volumenameprefix>_<namespace>_<claimname>_UUID**。如果您使用这个 storageClass 在项目 **project1** 中创建一个名为 **myclaim** 的新 PVC，则卷名称是 **custom-project1-myclaim-UUID**。



注意

如果没有指定 **gidMin** 和 **gidMax** 值，其默认值分别为 2000 和 2147483647。每个动态置备的卷都将获得这个范围中的 GID(**gidMin-gidMax**)。当相应的卷被删除时，池中会发布这个 GID。GID 池为每个 StorageClass 使用。如果两个或多个存储类具有 GID 范围，则置备程序可能会分配重复的 GID。

当使用 heketi 身份验证时，包含 admin 键的 Secret 也应存在：

heketi-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  key: bXlwYXNzd29yZA== 1
type: kubernetes.io/glusterfs
```

1 base64 编码的密码，例如 **echo -n "mypassword" | base64**

注意

当 PV 被动态置备时，GlusterFS 插件会自动创建一个 Endpoints 和一个无头服务，名为 **gluster-dynamic-<claimname>**。删除 PVC 时，这些动态资源会被自动删除。

27.17.3.7. Ceph RBD 对象定义

ceph-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
```

```
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/rbd
parameters:
  monitors: 10.16.153.105:6789 (1)
  adminId: admin (2)
  adminSecretName: ceph-secret (3)
  adminSecretNamespace: kube-system (4)
  pool: kube (5)
  userId: kube (6)
  userSecretName: ceph-secret-user (7)
  fsType: ext4 (8)
```

1. Ceph 监视器，以逗号分隔.它是必需的。
2. 能够在池中创建镜像的 Ceph 客户端 ID。默认为 "admin"。
3. **adminId** 的 secret 名称。它是必需的。提供的 secret 必须具有类型 "kubernetes.io/rbd"。
4. **adminSecret** 的命名空间。默认为 "default"。
5. Ceph RBD 池。默认为 "rbd"。
6. 用于映射 Ceph RBD 镜像的 Ceph 客户端 ID。默认值与 **adminId** 相同。
7. 对于 **userId**，用于映射 Ceph RBD 镜像的 Ceph Secret 名称。它必须与 PVC 位于同一个命名空间中。它是必需的。
8. 在动态部署卷中创建的文件系统。这个值被复制到动态配置的持久性卷的 **fstype** 字段中，并在第一个挂载卷时创建文件系统。默认值为 **ext4**。

27.17.3.8. Trident 对象定义

trident.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gold
provisioner: netapp.io/trident (1)
parameters: (2)
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
```

Trident 使用参数作为注册的不同存储池的选择标准。Trident 本身配置单独配置。

1. 如需有关在 OpenShift Container Platform 中安装 Trident 的更多信息，请参阅 Tr ident 文档。
2. 有关支持参数的更多信息，请参阅 Trident 文档 的存储属性 部分。

27.17.3.9. VMware vSphere 对象定义

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: slow
provisioner: kubernetes.io/vsphere-volume (1)
parameters:
  diskformat: thin (2)
```

(1) 有关在 OpenShift Container Platform 中使用 VMWare vSphere 的详情，请参考 VMWare vSphere 文档。

(2) **diskformat** ：**thin**、**zeroedthick** 和 **eagerzeroedthick**。详情请查看 vSphere 文档。默认： **thin**

27.17.3.10. Azure File 对象定义

配置 Azure 文件动态置备：

1. 在用户的项目中创建角色：

```
$ cat azf-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: system:controller:persistent-volume-binder
  namespace: <user's project name>
rules:
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "get", "delete"]
```

2. 在 **kube-system** 项目中创建到 **persistent-volume-binder** 服务帐户的角色绑定：

```
$ cat azf-rolebind.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: system:controller:persistent-volume-binder
  namespace: <user's project>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: system:controller:persistent-volume-binder
subjects:
- kind: ServiceAccount
  name: persistent-volume-binder
namespace: kube-system
```

3. 将服务帐户作为 **admin** 添加到用户的项目中：

```
$ oc policy add-role-to-user admin system:serviceaccount:kube-system:persistent-volume-
binder -n <user's project>
```

4. 为 Azure 文件创建存储类：

```
$ cat azfsc.yaml | oc create -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azfsc
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
```

用户现在可以创建一个使用此存储类的 PVC。

27.17.3.11. Azure Disk 对象定义

azure-advanced-disk-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/azure-disk
parameters:
  storageAccount: azure_storage_account_name (1)
  storageaccounttype: Standard_LRS (2)
  kind: Dedicated (3)
```

(1) Azure 存储帐户名称。这必须与集群位于同一个资源组中。如果指定了存储帐户，则忽略 **location**。如果没有指定存储帐户，则在与集群相同的资源组中创建一个新的存储帐户。如果您要指定 **StorageAccount**，则 **kind** 的值必须是 **Dedicated**。

(2) Azure 存储帐户 SKU 层。默认为空。**备注：**高级虚拟机可以同时附加 *Standard_LRS* 和 *Premium_LRS* 磁盘，标准虚拟机只能附加 *Standard_LRS* 磁盘，受管虚拟机只能附加受管磁盘，非受管虚拟机只能附加非受管磁盘。

(3) 可能的值有 **Shared**（默认）、**Dedicated** 和 **Managed**。

a. 如果 **kind** 设为 **Shared**，Azure 会在与集群相同的资源组中的几个共享存储帐户下创建所有未受管磁盘。

b. 如果 **kind** 设为 **Managed**，Azure 会创建新的受管磁盘。

c. 如果 **kind** 设为 **Dedicated**，并且指定了 **StorageAccount**，Azure 会将指定的存储帐户用于与集群相同的资源组中新的非受管磁盘。为此，请确保：

- 指定的存储帐户必须位于同一区域。
- Azure Cloud Provider 必须对存储帐户有写入权限。

d. 如果 **kind** 设为 **Dedicated**，并且未指定 **StorageAccount**，Azure 会在与集群相同的资源组中为新的非受管磁盘创建一个新的专用存储帐户。

重要

OpenShift Container Platform 版本 3.7 中修改了 Azure StorageClass。如果您从以前的版本升级，则以下任一操作：

- 指定属性 **kind:** 以便继续使用升级前创建的 Azure StorageClass。或者，
- 在 ***azure.conf*** 文件中添加 location 参数（如 **"location": "southcentralus",**）以使用默认属性 **kind: shared**。这样做会创建新的存储帐户供以后使用。

27.17.4. 修改默认的 StorageClass

如果使用 GCE 和 AWS，请使用以下步骤更改默认 StorageClass：

1. 列出 StorageClass:

```
$ oc get storageclass

NAME                 TYPE
gp2 (default)        kubernetes.io/aws-ebs (1)
standard             kubernetes.io/gce-pd
```

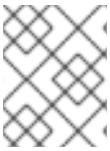
(1) **（默认）** 指定默认 StorageClass。

2. 为默认 StorageClass 将注解 **storageclass.kubernetes.io/is-default-class** 的值改为 **false** ：

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations": \
  {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. 通过添加或修改注解 **storageclass.kubernetes.io/is-default-class=true** 来使另外一个 StorageClass 作为默认。

```
$ oc patch storageclass standard -p '{"metadata": {"annotations": \
  {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```



注意

如果一个以上的 StorageClass 被标记为默认，则只能在 **storageClassName** 被显式指定时才能创建 PVC。因此，应只将一个 StorageClass 设置为默认值。

1. 确认更改：

```
$ oc get storageclass

NAME                 TYPE
gp2                  kubernetes.io/aws-ebs
standard (default)   kubernetes.io/gce-pd
```

27.17.5. 更多信息和示例

- 动态置备的 StorageClass 的示例和使用
- 没有动态置备的 StorageClass 的示例和使用

27.18. 卷安全性

27.18.1. 概述

本主题为 pod 安全性提供了常规指南，因为它与卷安全性相关。有关 pod 级别安全性的信息，请参阅管理安全性上下文约束(SCC)和安全性上下文约束(SCC) 和 安全性上下文约束 概念。有关 OpenShift Container Platform 持久性卷(PV)框架的信息，请参阅持久性存储概念主题。

访问持久性存储需要集群和/或存储管理员与最终开发人员之间的协调。集群管理员创建 PV，提取底层物理存储。开发人员会创建 pod，以及可选的 PVC，它根据匹配标准（如容量）绑定到 PV。

同一项目中的多个持久性卷声明 (PVC) 可以绑定到同一 PV。但是，当一个 PVC 绑定到 PV 后，这个 PV 不能被第一个声明的项目之外的声明绑定。如果需要多个项目访问底层存储，则每个项目需要自己的 PV，这可能指向同一物理存储。因此，绑定 PV 已绑定到项目。如需详细的 PV 和 PVC 示例，请参阅 使用持久性卷 部署 WordPress 和 MySQL 的示例。

对于集群管理员，授予对 PV 的 pod 访问权限涉及：

- 知道分配给实际存储的组 ID 和/或用户 ID，
- 了解 SELinux 的注意事项，以及
- 确保在为项目定义的法律 ID 范围内允许这些 ID，并且/或与 Pod 要求匹配的 SCC。

组 ID、用户 ID 和 SELinux 值在 Pod 定义中的 **SecurityContext** 部分中定义。组 ID 是 pod 的全局性，适用于 pod 中定义的所有容器。用户 ID 也可以全局，或者特定于每个容器。四个部分控制对卷的访问：

- **supplementalGroups**
- **fsGroup**
- **RunAsUser**
- **seLinuxOptions**

27.18.2. SCC、defaults 和 Allowed Ranges

SCC 会影响 pod 是否被授予默认用户 ID、**fsGroup** ID、补充组 ID 和 SELinux 标签。它们也会影响 Pod 定义（或镜像中）提供的 ID 是否针对一系列允许 ID 进行验证。如果需要验证并失败，pod 也会失败。

SCC 定义策略，如 **runAsUser**、**supplementalGroups** 和 **fsGroup**。这些策略帮助决定 pod 是否被授权。设为 **RunAsAny** 的策略值实质上表示该 pod 能执行与该策略相关的内容。会为该策略跳过授权，且不会基于此策略生成 OpenShift Container Platform 默认授权。因此，生成的容器中的 ID 和 SELinux 标签基于容器默认值，而不是 OpenShift Container Platform 策略。

有关 **RunAsAny** 的快速摘要：

- 允许容器集定义（或镜像）中定义的任何 ID。
- 如果容器集定义（和镜像中）中没有 ID，则容器会分配 ID，这是 Docker 的 **root** (0)。

- 没有定义 SELinux 标签，因此 Docker 将分配唯一标签。

因此，应对 ID 相关策略的带有 **RunAsAny** 的 SCC 进行保护，以便普通开发人员无法访问 SCC。另一方面，SCC 策略设置为 **MustRunAs** 或 **MustRunAsRange** 触发器 ID 验证（用于 ID 相关的策略），并在 Pod 定义或镜像中未提供这些值时向容器提供默认值。

小心

允许访问具有 **RunAsAny FSGroup** 策略的 SCC 也可以阻止用户访问其块设备。Pod 需要指定 **fsGroup** 才能接管其块设备。通常，这在 SCC **FSGroup** 策略被设置为 **MustRunAs** 时完成。如果用户的 Pod 分配有 **RunAsAny FSGroup** 策略的 SCC，则用户可能会面临 **权限被拒绝** 的错误，直到他们发现需要指定 **fsGroup** 本身。

SCC 可以定义允许的 ID（用户或组）的范围。如果需要进行范围检查（例如，使用 **MustRunAs**，且 SCC 中没有定义允许范围），则项目决定 ID 范围。因此，项目支持允许的 ID 范围。但是，与 SCC 不同，项目不定义策略，如 **runAsUser**。

允许的范围不仅有用，因为它们为容器 ID 定义界限，而且因为该范围内的最小值成为问题 ID 的默认值。例如，如果 SCC ID 策略值是 **MustRunAs**，则 ID 范围的最小值为 **100**，并且容器集定义中没有该 ID，则将 100 作为此 ID 的默认值。

作为 pod 准入的一部分，会检查 Pod 可用的 SCC（大约在优先级顺序中，其次序为限制）来最好与 pod 请求匹配。将 SCC 的策略类型设置为 **RunAsAny** 的限制性较低，而 **MustRunAs** 类型则更严格的。所有这些策略都被评估。要查看哪些 SCC 分配给 pod，请使用 **oc get pod** 命令：

```
# oc get pod <pod_name> -o yaml
...
metadata:
  annotations:
    openshift.io/scc: nfs-scc 1
  name: nfs-pod1 2
  namespace: default 3
...
```

1. 使用 Pod 的 SCC 的名称（本例中为自定义 SCC）。
2. pod 的名称。
3. 项目的名称。"namespace"在 OpenShift Container Platform 中与 "project" 进行交换。详情请参阅项目和用户。

可能不会直接识别哪些 SCC 与 pod 匹配，因此上述命令对于了解 UID、补充组和 SELinux 在实时容器中重新标记非常有用。

任何策略设置为 **RunAsAny** 的 SCC 允许在容器集定义（和/或镜像）中定义该策略的特定值。当这应用到用户 ID (**runAsUser**) 时，需要谨慎来限制对 SCC 的访问，以防止容器以 root 用户身份运行。

由于 Pod 通常与 **受限** SCC 匹配，因此值得了解这一要求的安全性。**受限** SCC 具有以下特征：

- 用户 ID 被限制，因为 **runAsUser** 策略被设置为 **MustRunAsRange**。这会强制用户 ID 验证。
- 因为 SCC 中没有定义允许的用户 ID 范围（请参阅 oc get -o yaml --export scc restricted' 以获取更多详情），项目的 **openshift.io/sa.scc.uid-range** 范围将用于范围检查和默认 ID（如果需要）。

- 当 Pod 定义中没有指定用户 ID 且匹配的 SCC 的 **runAsUser** 设置为 **MustRunAsRange** 时，会生成一个默认用户 ID。
- 需要 SELinux 标签（**seLinuxContext** 设置为 **MustRunAs**），它使用项目的默认 MCS 标签。
- **fsGroup** ID 仅限于单个值，因为 **FSGroup** 策略被设置为 **MustRunAs**，这表示要使用的值是指定第一个范围的最小值。
- 因为 SCC 中没有定义允许 **fsGroup** ID 范围，因此项目的 **openshift.io/sa.scc.supplemental-groups** 范围（或用于用户 ID 的相同范围）将用于验证和默认 ID（如果需要）。
- 当 pod 中没有指定 **fsGroup** ID 且匹配的 SCC 的 **FSGroup** 设为 **MustRunAs** 时，会生成一个默认的 **fsGroup** ID。
- 允许使用任意补充组 ID，因为不需要范围检查。这是将 **supplementalGroups** 策略设置为 **RunAsAny** 的结果。
- 由于 **RunAsAny** 对于上述两个组策略，因此不会为正在运行的 Pod 生成默认的补充组。因此，如果在 Pod 定义中（或镜像中没有定义任何组），则容器不会预定义任何补充组。

以下显示了 **default** 项目和自定义 SCC(**my-custom-scc**)，它总结了 SCC 和项目的交互：

```
$ oc get project default -o yaml (1)
...
metadata:
  annotations: (2)
    openshift.io/sa.scc.mcs: s0:c1,c0 (3)
    openshift.io/sa.scc.supplemental-groups: 1000000000/10000 (4)
    openshift.io/sa.scc.uid-range: 1000000000/10000 (5)

$ oc get scc my-custom-scc -o yaml
...
fsGroup:
  type: MustRunAs (6)
  ranges:
  - min: 5000
    max: 6000
runAsUser:
  type: MustRunAsRange (7)
  uidRangeMin: 1000100000
  uidRangeMax: 1000100999
seLinuxContext: (8)
  type: MustRunAs
  SELinuxOptions: (9)
    user: <selinux-user-name>
    role: ...
    type: ...
    level: ...
supplementalGroups:
  type: MustRunAs (10)
  ranges:
  - min: 5000
    max: 6000
```

(1) **default** 是项目的名称。

2 仅当对应的 SCC 策略不是 **RunAsAny** 时，才会生成默认值。

3 在 Pod 定义或 SCC 中定义时，SELinux 默认。

4 允许的组 ID 范围。只有 SCC 策略是 **RunAsAny** 时才会进行 ID 验证。可以指定多个范围，用逗号分开。有关 支持的格式，请参阅以下。

5 与 <4> 相同，但适用于用户 ID。另外，只支持一个用户 ID。

6 10 **MustRunAs** 强制实施组 ID 范围检查，提供容器的组默认值。根据此 SCC 定义，默认值为 5000（最小 ID 值）。如果 SCC 省略了范围，则默认值为 1000000000（从项目获得）。其他支持的类型 **RunAsAny** 不执行范围检查，因此允许任何组 ID，且不生成默认组。

7 **MustRunAsRange** 强制实施用户 ID 范围检查并提供 UID 默认。根据此 SCC，默认的 UID 为 1000100000（最小值）。如果在 SCC 中省略了最小和最大范围，则默认用户 ID 为 1000000000（从项目派生出）。**MustRunAsNonRoot** 和 **RunAsAny** 是其他受支持的类型。可以定义允许 ID 的范围，使其包含目标存储所需的任何用户 ID。

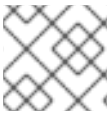
8 当设置为 **MustRunAs** 时，会使用 SCC 的 SELinux 选项或项目中定义的 MCS 默认创建容器。类型为 **RunAsAny** 表示不需要 SELinux 上下文，如果 Pod 中未定义，则不会在容器中设置 SELinux 上下文。

9 此处可以定义 SELinux 用户名、角色名称、类型和标签。

允许的范围支持两种格式：

1. **M/N**，其中 **M** 是起始 ID，**N** 是 count，因此范围变为 **M** （包括） **M+N-1**。
2. **M-N**，其中 **M** 再次是起始 ID，**N** 是结束的 ID。默认组 ID 是第一个范围内的起始 ID，本例中为 **1000000000**。如果 SCC 没有定义最小组 ID，则应用项目的默认 ID。

27.18.3. 补充组



注意

在使用补充组之前，请阅读 SCC、defaults 和 Allowed Ranges。

提示

与使用 用户 ID 相比，通常最好使用组 ID（supplemental 或 fsGroup）来获得对持久性存储的访问。

补充组是常规的 Linux 组。当进程在 Linux 中运行时，它具有一个 UID、一个 GID，以及一个或多个补充组。可以为容器的主进程设置这些属性。**supplementalGroups** ID 通常用于控制对共享存储的访问，如 NFS 和 GlusterFS，而 fsGroup 则用于控制对块存储的访问，如 Ceph RBD 和 iSCSI。

OpenShift Container Platform 共享存储插件挂载卷，以便挂载上的 POSIX 权限与目标存储上的权限匹配。例如，如果目标存储的所有者 ID 是 1234，其组 ID 为 5678，则主机节点上和容器中的挂载将具有同样的 ID。因此，容器的主进程必须匹配其中一个或两个 ID 才能访问该卷。

例如，请考虑以下 NFS 导出：

在 OpenShift Container Platform 节点上：

注意

showmount 需要访问由 **rpcbind** 和 **rpc.mount** 在 NFS 服务器上使用的端口

```
# showmount -e <nfs-server-ip-or-hostname>
Export list for f21-nfs.vm:
/opt/nfs  *
```

在 NFS 服务器中：

```
# cat /etc/exports
/opt/nfs *(rw,sync,root_squash)
...

# ls -lZ /opt/nfs -d
drwx------. 1000100001 5555 unconfined_u:object_r:usr_t:s0   /opt/nfs
```

/opt/nfs/ 导出可以被 UID **1000100001** 和组 **5555** 访问。通常，容器不应以 root 用户身份运行。因此，在这个 NFS 示例中，没有以 UID **1000100001** 运行且不是组 **5555** 的容器将无法访问 NFS 导出。

通常，与 pod 匹配的 SCC 不允许指定特定用户 ID，因此使用补充组可以更加灵活地向 pod 授予存储访问权限。例如，要向导出授予 NFS 访问权限，可在 Pod 定义中定义组 **5555** ：

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - name: ...
    volumeMounts:
    - name: nfs 1
      mountPath: /usr/share/... 2
  securityContext: 3
    supplementalGroups: [5555] 4
  volumes:
  - name: nfs 5
    nfs:
      server: <nfs_server_ip_or_host>
      path: /opt/nfs 6
```

1. 卷挂载的名称。必须与 **volumes** 部分中的名称匹配。
2. 与容器中看到的 NFS 导出路径。
3. Pod 全局安全上下文。适用于 pod 中的所有容器。每个容器也可以定义其 **securityContext**，但组 ID 对 pod 全局，且无法为单个容器定义。
4. 补充组（这是 ID 数组）被设置为 5555。这样可授予对导出的组访问权限。
5. 卷的名称。必须与 **volumeMounts** 部分中的名称匹配。
6. NFS 服务器上的实际 NFS 导出路径。

以上 pod 中的所有容器（假设匹配的 SCC 或项目允许组 **5555**）成为组 **5555** 的成员，并且无论容器的用户 ID 都有权访问该卷。但是，上述假设至关重要。有时，SCC 并不定义允许的组 ID 范围，而是需要组 ID 验证（s **supplementalGroups** 的结果设置为 **MustRunAs**）。请注意，这**不是 restricted** SCC 的情况。该项目不太可能允许组 ID **5555**，除非项目自定义了用于访问此 NFS 导出。因此，在这种情况下，上面的 Pod 将失败，因为它组 ID **5555** 不在 SCC 或项目的允许组 ID 中。

补充组和自定义 SCC

要补救上一个示例中的情况，可以创建一个自定义 SCC，以便：

- 已定义最小和最大组 ID，
- 已强制检查 ID 范围检查，
- 允许组 ID **5555**。

创建新 SCC 而非修改预定义的 SCC，或更改预定义项目中允许的 ID 范围，通常是创建新的 SCC。

创建新 SCC 的最简单方法是导出现有的 SCC，并自定义 YAML 文件以满足新 SCC 的要求。例如：

1. 使用 **restricted** SCC 作为新 SCC 的模板：

```
$ oc get -o yaml --export scc restricted > new-scc.yaml
```

2. 将 ***new-scc.yaml*** 文件编辑到您需要的规格。
3. 创建新 SCC：

```
$ oc create -f new-scc.yaml
```

注意

oc edit scc 命令可用于修改实例化的 SCC。

以下是名为 **nfs-scc** 的新 SCC 片段：

```
$ oc get -o yaml --export scc nfs-scc

allowHostDirVolumePlugin: false (1)
...
kind: SecurityContextConstraints
metadata:
  ...
  name: nfs-scc (2)
priority: 9 (3)
...
supplementalGroups:
  type: MustRunAs (4)
  ranges:
  - min: 5000 (5)
    max: 6000
...
```

(1) **allow** 布尔值与 **restricted** SCC 相同。

2 新 SCC 的名称。

3 数字较大的数值具有更高的优先级。nil 或 omitted 是最低优先级。在优先级较低的 SCC 前排列较高的优先级 SCC，因此可以更好地匹配新 pod。

4 **supplementalGroups** 是一个策略，它被设置为 **MustRunAs**，这意味着需要进行组 ID 检查。

5 支持多个范围。这里允许的组 ID 范围为 5000 到 5999，默认的补充组为 5000。

如果前面显示的相同 pod 针对这个新 SCC 运行（假设，pod 与新 SCC 匹配时），它将会启动，因为 Pod 定义中提供的组 **5555** 可以被自定义 SCC 支持。

27.18.4. fsGroup

注意

在使用补充组之前，请阅读 SCC、defaults 和 Allowed Ranges。

提示

与使用 用户 ID 相比，通常最好使用组 ID（supplemental 或 **fsGroup**）来获得对持久性存储的访问。

fsGroup 定义 pod 的"文件系统组"ID，该 ID 已添加到容器的补充组中。 **supplementalGroups** ID 适用于共享存储，而 **fsGroup** ID 用于块存储。

块存储（如 Ceph RBD、iSCSI 和各种云存储）通常专用于单一 pod 请求块存储卷，可以是直接或使用 PVC。与共享存储不同，块存储被 pod 接管；也就是说，容器集定义（或镜像）中提供的用户和组 ID 被应用到实际的物理块设备。通常，块存储不被共享。

在以下 pod 定义片段中显示了 **fsGroup** 定义：

```
kind: Pod
...
spec:
  containers:
  - name: ...
  securityContext: 1
    fsGroup: 5555 2
  ...
```

1 与 **supplementalGroups** 一样，**fsGroup** 必须全局定义到 pod，而不是每个容器。

2 5555 将成为卷组组权限以及卷中创建的所有新文件的组 ID。

与 **supplementalGroups** 一样，以上 Pod 中的所有容器（假设匹配的 SCC 或项目允许组 **5555**）将成为组 **5555** 的成员，并且有权访问块卷，而不管容器的用户 ID 是什么。如果 pod 与 **restricted** SCC 匹配，它的 **fsGroup** 策略是 **MustRunAs**，则 pod 将运行失败。但是，如果 SCC 的 **fsGroup** 策略设为 **RunAsAny**，则将接受任何 **fsGroup** ID（包括 5555）。请注意，如果 SCC 的 **fsGroup** 策略设置为 **RunAsAny**，并且未指定 **fsGroup** ID，则不会发生块存储的"接管"，则将拒绝该 Pod 的权限。

fsGroups 和 Custom SCCs

要补救上例中的情况，可以创建自定义 SCC，以便：

- 定义最小和最大组 ID，
- 已强制检查 ID 范围检查，
- 允许组 ID **5555**。

最好创建新 SCC，而不修改预定义的 SCC，或更改预定义项目中允许的 ID 范围。

请考虑以下片段的新 SCC 定义：

```
# oc get -o yaml --export scc new-scc
...
kind: SecurityContextConstraints
...
fsGroup:
  type: MustRunAs (1)
  ranges: (2)
  - max: 6000
    min: 5000 (3)
...
```

1. **MustRunAs** 触发组 ID 范围检查，而 **RunAsAny** 不需要范围检查。
2. 允许的组 ID 范围是 5000 到，包括 5999。支持多个范围，但不支持使用。这里允许的组 ID 范围为 5000 到 5999，默认的 **fsGroup** 为 5000。
3. 最小值（或整个范围）可以从 SCC 省略，因此范围检查和生成默认值将推迟到项目的 **openshift.io/sa.scc.supplemental-groups** 范围。**fsGroup** 和 **supplementalGroups** 在项目中使用相同的 group 字段，因此没有 **fsGroup** 的范围。

当上方所示的 pod 针对这个新 SCC 运行（假定为 course），pod 与新 SCC 匹配时，它将会启动，因为 Pod 定义中提供的组 **5555** 受到自定义 SCC 的限制。另外，pod 会"接管"块设备，因此当 pod 之外的进程查看块存储时，它实际上将具有 **5555** 作为其组 ID。

支持块所有权的卷列表包括：

- AWS Elastic Block Store
- OpenStack Cinder
- Ceph RBD
- GCE Persistent Disk
- iSCSI
- emptyDir

注意

此列表可能不完整。

27.18.5. 用户 ID

注意

在使用补充组之前，请阅读 SCC、defaults 和 Allowed Ranges。

提示

与使用用户 ID 相比，通常最好使用组 ID（supplemental 或 fsGroup）来获得对持久性存储的访问。

用户 ID 可以在容器镜像或容器集定义中定义。在 pod 定义中，单个用户 ID 可以全局定义到所有容器，或特定于单个容器（或两者）。用户 ID 包括在 pod 定义片段中：

```
spec:
  containers:
  - name: ...
    securityContext:
      runAsUser: 1000100001
```

上述中的 ID 1000100001 是特定于容器的，与导出上的所有者 ID 匹配。如果 NFS 导出的所有者 ID 是 **54321**，则该数字将在 pod 定义中使用。指定容器定义之外的 **securityContext** 会将 ID 全局提供给 pod 中的所有容器。

与组 ID 类似，用户 ID 可能会根据 SCC 和/或项目中设置的策略来验证。如果 SCC 的 **runAsUser** 策略设为 **RunAsAny**，则允许容器集定义或镜像中定义的任何用户 ID。

警告

这意味着允许 UID **0** (root)。

如果设为 MustRunAsRange，则 **runAsUser** 策略被设置为 **MustRunAsRange**，则提供的用户 ID 将会根据允许 ID 的范围进行验证。如果 pod 没有提供用户 ID，则默认 ID 被设置为允许的用户 ID 范围的最小值。

返回先前的 NFS 示例，容器需要将其 UID 设定为 **1000100001**，这显示在上面的 pod 片段中。假设 **默认项目** 和 **restricted** SCC，则不允许 Pod 请求的用户 ID 1000100001，因此 pod 将失败。pod 失败，因为：

- 它将请求 **1000100001** 作为其用户 ID，
- 所有可用的 SCC 使用 **MustRunAsRange** 作为其 **runAsUser** 策略，因此需要 UID 范围检查，
- **1000100001** 不包含在 SCC 或项目用户 ID 范围内。

要避免这种情况，可以使用适当的用户 ID 范围创建新的 SCC。还可以通过定义适当的用户 ID 范围创建新项目。另外还有其他首选选项：

- 可将 **restricted** SCC 修改为在最小和最大用户 ID 范围内包含 **1000100001**。不建议这样做，否则应避免在可能的情况下修改预定义的 SCC。
- **restricted** SCC 可以修改为使用 **RunAsAny** 作为 **runAsUser** 值，从而消除 ID 范围检查。*强烈建议不要*这样做，因为容器可以以 root 用户身份运行。

- **默认** 项目的 UID 范围可以更改，以允许用户 ID **1000100001**。这通常不建议，因为只能指定单个用户 ID，因此如果更改了范围，则其他 pod 可能无法运行。

用户 ID 和自定义 SCC

最好避免在可能的情况下修改预定义的 SCC。首选方法是创建一个最适合组织安全需求的自定义 SCC，或 创建一个 支持所需用户 ID 的新项目。

要补救上例中的情况，可以创建自定义 SCC，以便：

- 已定义最小和最大用户 ID，
- UID 范围检查仍会被强制使用，
- 允许使用 **1000100001** 的 UID。

例如：

```
$ oc get -o yaml --export scc nfs-scc

allowHostDirVolumePlugin: false (1)
...
kind: SecurityContextConstraints
metadata:
  ...
  name: nfs-scc (2)
priority: 9 (3)
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsRange (4)
  uidRangeMax: 1000100001 (5)
  uidRangeMin: 1000100001
...
```

1. **allowXX** bools 与 **受限** SCC 相同。
2. 此新 SCC 的名称为 **nfs-scc**。
3. 数字较大的数值具有更高的优先级。nil 或 omitted 是最低优先级。在优先级较低的 SCC 前排列较高的优先级 SCC，因此可以更好地匹配新 pod。
4. **runAsUser** 策略被设置为 **MustRunAsRange**，这意味着强制进行 UID 范围检查。
5. UID 范围为 1000100001 到 1000100001（一个值的范围）。

现在，使用 **runAsUser：上一个 pod 定义片段中显示的 1000100001**，pod 与新的 **nfs-scc** 匹配，且 UID 为 1000100001。

27.18.6. SELinux 选项

除 **privileged** SCC 外，所有预定义的 SCC 都会将 **seLinuxContext** 设置为 **MustRunAs**。因此，最有可能与 Pod 要求匹配的 SCC 将强制 Pod 使用 SELinux 策略。Pod 使用的 SELinux 策略可以在 Pod 本身、在镜像、SCC 或项目中定义（提供默认值）。

SELinux 标签可以在 pod 的 **securityContext.seLinuxOptions** 部分中定义，并支持 **user**, **role**, **type**, 和 **level**:

注意

这一主题中可互换使用 level 和 MCS 标签。

```
...
 securityContext: 1
    seLinuxOptions:
      level: "s0:c123,c456" 2
...
```

1 可以为整个 pod 全局定义 **级别**，也可以针对每个容器单独定义。

2 SELinux 级别标签。

以下是来自 SCC 和 **default** 项目的片段：

```
$ oc get -o yaml --export scc scc-name
...
seLinuxContext:
  type: MustRunAs 1

# oc get -o yaml --export namespace default
...
metadata:
  annotations:
    openshift.io/sa.scc.mcs: s0:c1,c0 2
...
```

1 **MustRunAs** 会导致卷重新标记。

2 如果 Pod 或 SCC 中没有提供该标签，则默认来自项目。

除 **privileged** SCC 外，所有预定义的 SCC 都会将 **seLinuxContext** 设置为 **MustRunAs**。这会强制 pod 使用 MCS 标签，这些标签可以在容器集定义、镜像或作为默认定义中定义。

SCC 决定是否需要 SELinux 标签，并提供默认标签。如果 **seLinuxContext** 策略被设置为 **MustRunAs**，并且 pod（或镜像）没有定义标签，OpenShift Container Platform 默认为从 SCC 本身或从项目中选择的标签。

如果 **seLinuxContext** 设置为 **RunAsAny**，则不提供默认标签，容器决定了最终的标签。如果是 Docker，容器将使用唯一的 MCS 标签，该标签可能与现有存储挂载上的标签匹配。支持 SELinux 管理的卷将被重新标记，以便它们可以被指定标签访问，这取决于标签如何排除，只有该标签才可以被该标签访问。

这意味着，对于非特权容器的两个操作：

- 卷被授予可由非特权容器访问的类型。这个类型通常是 Red Hat Enterprise Linux(RHEL)版本 7.5 及更新的版本中的 **container_file_t**。这个类型将卷视为容器内容。在以前的 RHEL 版本、RHEL 7.4、7.3 等中，为卷提供 **svirt_sandbox_file_t** 类型。

- 如果指定了 **级别**，则卷使用给定的 MCS 标签标记。

要使卷可以被 pod 访问，pod 必须具有这两个卷类别。因此，使用 **s0:c1,c2** 的 pod 可以通过 **s0:c1,c2** 访问卷。所有 pod 都可以访问具有 **s0** 的卷。

如果 pod 出现故障，或者存储挂载因为权限错误而失败，则可能会 SELinux 强制中断。要检查这一点的一种方法是运行：

```
# ausearch -m avc --start recent
```

这会检查 AVC(Access Vector Cache)错误的日志文件。

27.19. SELECTOR-LABEL VOLUME BINDING

27.19.1. 概述

本指南提供了通过 **selector** 和 **label** 属性将持久性卷声明(PVC)绑定到持久性卷(PV)所需的步骤。通过实施选择器和标签，常规用户可以根据集群管理员定义 的标识符来目标调配的存储。

27.19.2. 动机

在静态置备存储的情况下，需要寻找持久性存储的开发人员需要了解一些 PV 属性才能部署和绑定 PVC。这会产生几个有问题的情况。常规用户可能需要联系集群管理员来部署 PVC 或提供 PV 值。独立 PV 属性不会使存储卷的预期使用，也不能提供可以被对卷进行分组的方法。

选择器和标签属性可用于从用户提取 PV 详情，同时为集群管理员提供一个描述性和可自定义的标签来识别卷的方法。通过 selector-label 方法绑定，用户只需要知道管理员定义了哪些标签。

注意

selector-label 功能目前仅适用于 *静态置备* 的存储，目前没有为动态置备的存储实施。

27.19.3. Deployment

本节描述了如何定义和部署 PVC。

27.19.3.1. 先决条件

1. 正在运行的 OpenShift Container Platform 3.3+ 集群
2. 由受支持的存储供应商提供的卷
3. 具有 cluster-admin 角色绑定的用户

27.19.3.2. 定义持久性卷声明

1. 以 **cluster-admin** 用户身份，定义 PV。在本例中，我们将使用 GlusterFS 卷。如需了解 您的供应商配置，请参阅相应的存储供应商。

例 27.9. 带有标签的持久性卷

```
apiVersion: v1
kind: PersistentVolume
metadata:
```

```
  name: gluster-volume
  labels: 1
    volume-type: ssd
    aws-availability-zone: us-east-1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster
    path: myVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

1 选择器 *与所有* PV 标签匹配的 PVC 将会被绑定，假设一个 PV 可用。

2. 定义 PVC：

例 27.10. 使用 Selectors 的持久性卷声明

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector: 1
    matchLabels: 2
      volume-type: ssd
      aws-availability-zone: us-east-1
```

1 开始 **selectors** 部分。

2 列出用户请求存储的所有标签。*必须*与目标 PV *的所有* 标签匹配。

27.19.3.3. 可选：将 PVC 绑定到特定 PV

未指定 PV 名称的 PVC 或选择器将与任何 PV 匹配。

以集群管理员身份将 PVC 绑定到特定 PV:

- 如果您知道 PV 名称，请使用 **pvc.spec.volumeName**。
- 如果您知道 PV 标签，请使用 **pvc.spec.selector**。
  通过指定选择器，PVC 需要 PV 有特定的标签。

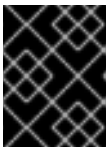
27.19.3.4. 可选：为特定的 PVC 保留 PV

要为特定的任务保留 PV，有两个选项：创建特定的存储类，或者预先将 PV 绑定到 PVC。

1. 通过指定存储类的名称，为 PV 请求特定的存储类。
   以下资源显示了用来配置 StorageClass 所需的值。这个示例使用 AWS ElasticBlockStore (EBS) 对象定义。

例 27.11. EBS 的 StorageClass 定义

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: kafka
provisioner: kubernetes.io/aws-ebs
...
```



重要

如果需要在多租户环境中，使用配额定义来保留存储类和 PV，只保留特定命名空间。

2. 使用 PVC 命名空间和名称预先将 PV 绑定到 PVC。定义的 PV 只会绑定到指定的 PVC，而不绑定到其他 PVC，如下例所示：

例 27.12. PV 定义中的 claimRef

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mktg-ops--kafka--kafka-broker01
spec:
  capacity:
    storage: 15Gi
  accessModes:
    - ReadWriteOnce
  claimRef:
      apiVersion: v1
      kind: PersistentVolumeClaim
      name: kafka-broker01
      namespace: default
...
```

27.19.3.5. 部署 PVC

以 **cluster-admin** 用户身份，创建持久性卷：

例 27.13. 创建持久性卷

```
# oc create -f gluster-pv.yaml
persistentVolume "gluster-volume" created
```

```
# oc get pv
NAME                LABELS    CAPACITY     ACCESSMODES   STATUS      CLAIM     REASON
AGE
gluster-volume      map[]     2147483648   RWX           Available               2s
```

创建 PV 后，任何与*所有*标签匹配的用户都可以创建其 PVC。

例 27.14. 创建持久性卷声明

```
# oc create -f gluster-pvc.yaml
persistentVolumeClaim "gluster-claim" created
# oc get pvc
NAME            LABELS    STATUS    VOLUME
gluster-claim             Bound     gluster-volume
```

27.20. 启用控制器管理的附加和分离

27.20.1. 概述

默认情况下，在集群 master 上运行的控制器代表一组节点管理卷附加和分离操作，而不是让它们管理自己的卷附加和分离操作。

控制器管理的附加和分离有以下优点：

- 如果节点丢失，则附加到它的卷可以被控制器分离，并在其它位置重新附加。
- 用于附加和分离的凭证不需要在每个节点上存在，从而提高安全性。

27.20.2. 确定管理附加和分离的内容

如果节点自己设置了注解 **volumes.kubernetes.io/controller-managed-attach-detach**，则它的 attach 和 detach 操作由控制器管理。控制器将自动检查此注解的所有节点，并根据它是否存在。因此，您可以检查此注解的节点，以确定它是否启用了控制器管理的附加和分离。

要进一步确保节点选择控制器管理的附加和分离，可以搜索其日志以找到以下行：

```
Setting node annotation to enable volume controller attach/detach
```

如果没有找到以上行，日志应包含：

```
Controller attach/detach is disabled for this node; Kubelet will attach and detach volumes
```

要在控制器结束时进行检查，它正在管理特定节点的附加和分离操作，日志级别必须首先设置为至少 **4**。然后，应该找到以下行：

```
processVolumesInUse for node <node_hostname>
```

有关如何查看日志和配置日志级别的详情，请参考 配置日志记录级别。

27.20.3. 配置节点以启用控制器管理的附加和分离

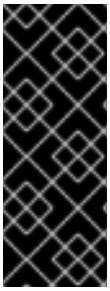
启用控制器管理的附加和分离通过将各个节点配置为选择并禁用自己的节点级附加和分离管理。如需了解要编辑并添加以下节点配置文件的信息，请参阅节点配置文件：

```
kubeletArguments:
  enable-controller-attach-detach:
  - "true"
```

配置节点后，必须重启它才能使设置生效。

27.21. 持久性卷快照

27.21.1. 概述



重要

持久性卷快照是一个技术预览功能。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

如需红帽技术预览功能支持范围的更多信息，请参阅 https://access.redhat.com/support/offerings/techpreview/。

许多存储系统提供创建持久性卷(PV)的"快照"以防数据丢失的能力。外部快照控制器和置备程序提供了使用 OpenShift Container Platform 集群中的功能，并通过 OpenShift Container Platform API 处理卷快照。

本文档描述了 OpenShift Container Platform 中卷快照支持的当前状态。建议您熟悉 PV、持久性卷声明 (PVC) 和 动态置备。

27.21.2. 功能

- 为绑定到 **PersistentVolumeClaim**的 **PersistentVolume** 创建快照
- 列出现有的 **VolumeSnapshot**
- 删除现有的 **VolumeSnapshot**
- 从现有 **VolumeSnapshot** 创建新的 **PersistentVolume**
- 支持的 **PersistentVolume** 类型 ：
  - AWS Elastic Block Store (EBS)
  - Google Compute Engine (GCE) Persistent Disk (PD)

27.21.3. 安装及设置

外部控制器和置备程序是提供卷快照的外部组件。这些外部组件在集群中运行。控制器负责创建、删除和报告卷快照的事件。置备程序从卷快照创建新 **PersistentVolume**。如需更多信息，请参阅创建快照和恢复快照。

27.21.3.1. 启动外部控制器和 Provisioner

外部控制器和置备程序服务以容器镜像形式分发，并可像平常一样在 OpenShift Container Platform 集群中运行。另外，控制器和置备程序也有 RPM 版本。

要允许容器管理 API 对象，管理员需要配置必要的基于角色的访问控制(RBAC)规则：

1. 创建 **ServiceAccount** 和 **ClusterRole** ：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: snapshot-controller-runner
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: snapshot-controller-role
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["list", "watch", "create", "update", "patch"]
  - apiGroups: ["apiextensions.k8s.io"]
    resources: ["customresourcedefinitions"]
    verbs: ["create", "list", "watch", "delete"]
  - apiGroups: ["volumesnapshot.external-storage.k8s.io"]
    resources: ["volumesnapshots"]
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
  - apiGroups: ["volumesnapshot.external-storage.k8s.io"]
    resources: ["volumesnapshotdatas"]
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

2. 通过 **ClusterRoleBinding** 绑定规则：

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: snapshot-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: snapshot-controller-role
subjects:
- kind: ServiceAccount
  name: snapshot-controller-runner
  namespace: default
```

如果外部控制器和置备程序部署在 Amazon Web Services(AWS)中，则它们必须能够使用访问密钥进行身份验证。要为 pod 提供凭证，管理员创建一个新的 secret：

```
apiVersion: v1
kind: Secret
metadata:
  name: awskeys
type: Opaque
data:
  access-key-id: <base64 encoded AWS_ACCESS_KEY_ID>
  secret-access-key: <base64 encoded AWS_SECRET_ACCESS_KEY>
```

外部控制器和置备程序容器的 AWS 部署（注意两个 pod 容器都使用 secret 访问 AWS 云供应商 API）：

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: snapshot-controller
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: snapshot-controller
    spec:
      serviceAccountName: snapshot-controller-runner
      containers:
        - name: snapshot-controller
          image: "registry.redhat.io/openshift3/snapshot-controller:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "aws"]
          env:
            - name: AWS_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: awskeys
                  key: access-key-id
            - name: AWS_SECRET_ACCESS_KEY
              valueFrom:
                secretKeyRef:
                  name: awskeys
                  key: secret-access-key
        - name: snapshot-provisioner
          image: "registry.redhat.io/openshift3/snapshot-provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "aws"]
          env:
            - name: AWS_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: awskeys
                  key: access-key-id
            - name: AWS_SECRET_ACCESS_KEY
              valueFrom:
```

```
            secretKeyRef:
              name: awskeys
              key: secret-access-key
```

对于 GCE，不需要使用 secret 来访问 GCE 云供应商 API。管理员可以继续部署：

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: snapshot-controller
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: snapshot-controller
    spec:
      serviceAccountName: snapshot-controller-runner
      containers:
        - name: snapshot-controller
          image: "registry.redhat.io/openshift3/snapshot-controller:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "gce"]
        - name: snapshot-provisioner
          image: "registry.redhat.io/openshift3/snapshot-provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "gce"]
```

27.21.3.2. 管理快照用户

根据集群配置，可能需要允许非管理员用户操作 API 服务器上的 **VolumeSnapshot** 对象。这可以通过创建一个绑定到特定用户或组的 **ClusterRole** 来实现。

例如，假设用户"alice"需要使用集群中的快照。集群管理员完成下列步骤：

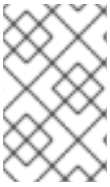
1. 定义一个新的 **ClusterRole** ：

```
apiVersion: v1
kind: ClusterRole
metadata:
  name: volumesnapshot-admin
rules:
- apiGroups:
  - "volumesnapshot.external-storage.k8s.io"
  attributeRestrictions: null
  resources:
  - volumesnapshots
  verbs:
  - create
  - delete
  - deletecollection
  - get
  - list
```

```
  - patch
  - update
  - watch
```

2. 通过创建一个 **ClusterRoleBinding** 对象，将集群角色绑定到用户的"alice"：

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: volumesnapshot-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: volumesnapshot-admin
subjects:
- kind: User
  name: alice
```



注意

这只是 API 访问配置的一个示例。**VolumeSnapshot** 对象的行为与其他 OpenShift Container Platform API 对象类似。有关管理 API RBAC 的详情，请参阅 API 访问控制文档。

27.21.4. 卷快照和卷快照数据的生命周期

27.21.4.1. 持久性卷声明和持久性卷

PersistentVolumeClaim 绑定到 **PersistentVolume**。**PersistentVolume** 类型必须是支持的快照类型之一。

27.21.4.1.1. snapshot Promoter

创建 **StorageClass** ：

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: snapshot-promoter
provisioner: volumesnapshot.external-storage.k8s.io/snapshot-promoter
```

这个 **StorageClass** 需要从之前创建的 **VolumeSnapshot** 恢复 **PersistentVolume**。

27.21.4.2. 创建快照

要生成 PV 的快照，请创建一个新的 **VolumeSnapshot** 对象：

```
apiVersion: volumesnapshot.external-storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: snapshot-demo
spec:
  persistentVolumeClaimName: ebs-pvc
```

persistentVolumeClaimName 是绑定到 **PersistentVolume** 的 **PersistentVolumeClaim** 的名称。已为此特定 PV 创建了快照。

然后会根据 **VolumeSnapshot** 自动创建 **VolumeSnapshotData** 对象。**VolumeSnapshot** 和 **VolumeSnapshotData** 之间的关系与 **PersistentVolumeClaim** 和 **PersistentVolume** 之间的关系类似。

根据 PV 类型，操作可能会经历几个阶段，由 **VolumeSnapshot** 状态来反映：

1. 新的 **VolumeSnapshot** 对象已创建。
2. 控制器启动快照操作。可能需要冻结快照的 **PersistentVolume**，并暂停应用程序。
3. 存储系统完成创建快照（快照为"cut"），且快照的 **PersistentVolume** 可能会返回正常操作。快照本身尚未就绪。最后的状态条件是 **Pending** 类型，状态为 **True**。创建一个新的 **VolumeSnapshotData** 对象来代表实际快照。
4. 新创建的快照已完成并可使用。最后的状态条件是 **Ready** 类型，状态为 **True**。

重要

用户负责确保数据一致性（停止 Pod/应用程序，清除缓存，以及冻结文件系统等）。

注意

如果出现错误，**VolumeSnapshot** 状态会附加一个 **Error** 条件。

显示 **VolumeSnapshot** 状态：

```
$ oc get volumesnapshot -o yaml
```

此时会显示状态。

```
apiVersion: volumesnapshot.external-storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  clusterName: ""
  creationTimestamp: 2017-09-19T13:58:28Z
  generation: 0
  labels:
    Timestamp: "1505829508178510973"
  name: snapshot-demo
  namespace: default
  resourceVersion: "780"
  selfLink: /apis/volumesnapshot.external-
storage.k8s.io/v1/namespaces/default/volumesnapshots/snapshot-demo
  uid: 9cc5da57-9d42-11e7-9b25-90b11c132b3f
spec:
  persistentVolumeClaimName: ebs-pvc
  snapshotDataName: k8s-volume-snapshot-9cc8813e-9d42-11e7-8bed-90b11c132b3f
status:
  conditions:
  - lastTransitionTime: null
    message: Snapshot created successfully
    reason: ""
```

```
    status: "True"
    type: Ready
  creationTimestamp: null
```

27.21.4.3. 恢复快照

要从 **VolumeSnapshot** 恢复 PV，请创建一个 PVC：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: snapshot-pv-provisioning-demo
  annotations:
    snapshot.alpha.kubernetes.io/snapshot: snapshot-demo
spec:
  storageClassName: snapshot-promoter
```

annotations:**snapshot.alpha.kubernetes.io/snapshot** 是要恢复的 **VolumeSnapshot** 的名称。**storageClassName**:**StorageClass** 由管理员创建，用于恢复 **VolumeSnapshot**。

创建一个新的 **PersistentVolume** 并绑定到 **PersistentVolumeClaim**。根据 PV 类型，这个过程可能需要几分钟时间。

27.21.4.4. 删除快照

删除 **snapshot-demo** ：

```
$ oc delete volumesnapshot/snapshot-demo
```

自动删除绑定到 **VolumeSnapshot** 的 **VolumeSnapshotData**。

27.22. 使用 HOSTPATH

OpenShift Container Platform 集群中的 **hostPath** 卷将主机节点的文件系统中的文件或目录挂载到 pod 中。大多数 pod 不需要 **hostPath** 卷，但是如果应用程序需要它，它会提供一个快速的测试选项。

重要

集群管理员必须将 pod 配置为以特权方式运行。这样可访问同一节点上的 pod。

27.22.1. 概述

OpenShift Container Platform 支持在单节点集群中使用 **hostPath** 挂载进行开发和测试。

在生产环境集群中，不要使用 **hostPath**。集群管理员置备网络资源，如 GCE Persistent Disk 卷或 Amazon EBS 卷。网络资源支持使用存储类设置动态置备。

hostPath 卷必须静态置备。

27.22.2. 在 Pod 规格中配置 hostPath 卷

您可以使用 **hostPath** 卷访问节点上的 **读写** 文件。这对可以从内部配置和监控主机的 pod 很有用。您还可以使用 **hostPath** 卷使用 **mountPropagation** 在主机上挂载卷。

警告

使用 **hostPath** 卷可能会具有危险性，因为它们允许 pod 读取和写入主机上的任何文件。请小心操作。

建议您直接在 **Pod** 规格中指定 **hostPath** 卷，而不是在 **PersistentVolume** 对象中指定。这很有用，因为 pod 已经知道它在配置节点时访问的路径。

流程

1. 创建特权 pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name
spec:
  containers:
  ...
    securityContext:
      privileged: true
  volumeMounts:
  - mountPath: /host/etc/motd.confg 1
    name: hostpath-privileged
...
volumes:
  - name: hostpath-privileged
    hostPath:
      path: /etc/motd.confg 2
```

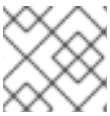
1 用于在特权 pod 中挂载 **hostPath** 共享的路径。

2 用于共享特权 pod 的主机上的路径。

在本例中，pod 可以看到 **/etc/motd.confg** 中作为 **/host/etc/motd.confg** 的主机路径。因此，可以在不直接访问主机的情况下配置 **motd**。

27.22.3. 静态置备 hostPath 卷

使用 **hostPath** 卷的 pod 必须通过手动或静态置备来引用。



注意

只有在没有持久性存储可用时，才应使用 **hostPath** 的持久性卷。

流程

1. 定义持久性卷（PV）的名称。使用 **PersistentVolume** 对象定义创建一个 **pv.yaml** 文件：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume (1)
  labels:
    type: local
spec:
  storageClassName: manual (2)
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce (3)
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/mnt/data" (4)
```

1. 卷的名称。持久性卷声明或 pod 识别它的名称。
2. 用于将持久性卷声明请求绑定到这个持久性卷。
3. 这个卷可以被一个单一的节点以 **read-write** 的形式挂载。
4. 配置文件指定卷在集群节点的 **/mnt/data** 中。

2. 从该文件创建 PV：

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

3. 定义持久性卷声明（PVC）。使用 **PersistentVolumeClaim** 对象定义创建一个 **pvc.yaml** 文件：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pvc-volume
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual
```

4. 从文件创建 PVC：

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

27.22.4. 在特权 pod 中挂载 hostPath 共享

创建持久性卷声明后，应用程序就可以在 pod 中使用它。以下示例演示了在 pod 中挂载此共享。

先决条件

- 已存在一个映射到底层 **hostPath** 共享的持久性卷声明。

流程

- 创建可挂载现有持久性卷声明的特权 pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name 1
spec:
  containers:
    ...
    securityContext:
      privileged: true 2
    volumeMounts:
    - mountPath: /data 3
      name: hostpath-privileged
  ...
  securityContext: {}
  volumes:
    - name: hostpath-privileged
      persistentVolumeClaim:
        claimName: task-pvc-volume 4
```

1. pod 的名称。
2. pod 必须以特权运行，才能访问节点的存储。
3. 在特权 pod 中挂载 hostPath 共享的路径。
4. 之前创建的 **PersistentVolumeClaim** 对象的名称。

27.22.5. 其它资源

- 挂载传播

第 28 章 持久性存储示例

28.1. 概述

以下小节提供了设置和配置常见存储用例的详细信息、全面的说明。这些示例涵盖持久性卷及其安全性的管理，以及如何以系统用户身份对卷进行声明。

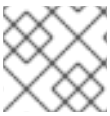
- 在两个 Pod 中共享 NFS PV
- ceph-RBD 块存储卷
- 使用 GlusterFS 卷的共享存储
- 使用 GlusterFS 动态置备存储
- 将 PV 挂载到特权 Pod
- 使用 GlusterFS 存储支持容器镜像 registry
- 根据标签绑定持久性卷
- 为动态置备使用 StorageClass
- 为现有旧存储使用 StorageClass
- 为集成 Container Image Registry 配置 Azure Blob 存储

28.2. 在两个持久性卷声明间共享 NFS 挂载

28.2.1. 概述

以下用例描述了如何使用共享存储以供两个独立的容器配置解决方案。此示例重点介绍了 NFS 的使用，但可以轻松适应其他共享存储类型，如 GlusterFS。另外，本例中会显示 Pod 安全性的配置，因为它与共享存储相关。

使用 NFS 的持久性存储 提供了持久性卷 (PV)、持久性卷声明(PVC)以及使用 NFS 作为持久性存储的信息。本主题显示了使用现有 NFS 集群和 OpenShift Container Platform 持久性存储的端到端示例，并假定 OpenShift Container Platform 基础架构中存在现有的 NFS 服务器和导出。



注意

所有 **oc** 命令在 OpenShift Container Platform master 主机上执行。

28.2.2. 创建持久性卷

在 OpenShift Container Platform 中创建 PV 对象前，定义了持久性卷(PV)文件：

例 28.1. 使用 NFS 的持久性卷对象定义

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv 1
```

```
spec:
  capacity:
    storage: 1Gi (2)
  accessModes:
    - ReadWriteMany (3)
  persistentVolumeReclaimPolicy: Retain (4)
  nfs: (5)
    path: /opt/nfs (6)
    server: nfs.f22 (7)
    readOnly: false
```

1. PV 的名称，它在 pod 定义中引用或在各种 **oc** volume 命令中显示。
2. 为这个卷分配的存储量。
3. **accessModes** 用作标签，以匹配 PV 和 PVC。它们目前没有定义任何形式的访问控制。
4. 卷重新声明(reclaim)策略 **Retain** 表示该卷在 pod 访问终止后会保留。
5. 这将定义正在使用的卷类型，在这个示例中是 **NFS** 插件。
6. 这是 NFS 挂载路径。
7. 这是 NFS 服务器。这也可以通过 IP 地址指定。

将 PV 定义保存到文件中，如 ***nfs-pv.yaml*** 并创建持久性卷：

```
# oc create -f nfs-pv.yaml
persistentvolume "nfs-pv" created
```

验证持久性卷是否已创建：

```
# oc get pv
NAME      LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM     REASON    AGE
nfs-pv    <none>    1Gi        RWX           Available                       37s
```

28.2.3. 创建持久性卷声明

持久性卷声明(PVC)指定所需的访问模式和存储容量。目前，仅基于这两个属性，PVC 绑定到一个 PV。当 PV 与 PVC 绑定后，PV 基本上与 PVC 的项目绑定，且无法与另一个 PVC 绑定。PV 和 PVC 有一对一的映射。但是，同一项目中的多个 pod 可以使用相同的 PVC。这是我们在此示例中突出显示的用例。

例 28.2. PVC 对象定义

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc (1)
spec:
  accessModes:
  - ReadWriteMany (2)
```

```
  resources:
    requests:
      storage: 1Gi
```

1. 声明名称由 pod 在其 **volumes** 部分下引用。
2. 如前面对 PV 提供的一样，**accessModes** 不会强制访问访问，而是作为标签来把一个 PV 与 PVC 进行匹配。
3. 这个声明会查找提供 **1Gi** 或更高容量的 PV。

将 PVC 定义保存到文件中，如 ***nfs-pvc.yaml*** 并创建 PVC：

```
# oc create -f nfs-pvc.yaml
persistentvolumeclaim "nfs-pvc" created
```

验证 PVC 是否已创建并绑定到预期的 PV:

```
# oc get pvc
NAME         LABELS    STATUS    VOLUME      CAPACITY   ACCESSMODES   AGE
nfs-pvc      <none>    Bound     nfs-pv      1Gi        RWX           24s
                                   1
```

1. 声明 **nfs-pvc** 已绑定到 **nfs-pv** PV。

28.2.4. 确保 NFS 卷访问

访问是 NFS 服务器中节点所必需的。在这个节点中，检查 NFS 导出挂载：

```
[root@nfs nfs]# ls -lZ /opt/nfs/
total 8
-rw-r--r--. 1 root 100003  system_u:object_r:usr_t:s0     10 Oct 12 23:27 test2b
            1      2
```

1. 所有者的 ID 为 0。
2. 组具有 ID 100003。

若要访问 NFS 挂载，容器必须与 SELinux 标签匹配，并且运行 UID 为 0，或者在其补充组范围内使用 100003。通过匹配 NFS 挂载的组获取卷的访问权限，该组将在如下 Pod 定义中定义。

默认情况下，SELinux 不允许从 pod 写入远程 NFS 服务器。要在每个节点中使用 SELinux 强制写入 NFS 卷，请运行：

```
# setsebool -P virt_use_nfs on
```

28.2.5. 创建 Pod

pod 定义文件或模板文件可用于定义 pod。以下是创建单个容器并挂载 NFS 卷的 pod 规格，以便进行读写访问：

例 28.3. Pod 对象定义

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-nfs-pod 1
  labels:
    name: hello-openshift-nfs-pod
spec:
  containers:
    - name: hello-openshift-nfs-pod
      image: openshift/hello-openshift 2
      ports:
        - name: web
          containerPort: 80
      volumeMounts:
        - name: nfsvol 3
          mountPath: /usr/share/nginx/html 4
  securityContext:
      supplementalGroups: [100003] 5
      privileged: false
  volumes:
    - name: nfsvol
      persistentVolumeClaim:
        claimName: nfs-pvc 6
```

1. **oc get pod** 显示此 pod 的名称。
2. 此 pod 运行的镜像。
3. 卷的名称。在 **containers** 和 **volumes** 部分中，此名称必须相同。
4. 如容器所示的挂载路径。
5. 要分配给容器的组 ID。
6. 上一步中创建的 PVC。

将 pod 定义保存到文件中，如 ***nfs.yaml***，然后创建 pod：

```
# oc create -f nfs.yaml
pod "hello-openshift-nfs-pod" created
```

验证 pod 是否已创建：

```
# oc get pods
NAME                     READY     STATUS    RESTARTS   AGE
hello-openshift-nfs-pod      1/1       Running   0          4s
```

oc describe pod 命令中显示更多详细信息：

```
[root@ose70 nfs]# oc describe pod hello-openshift-nfs-pod
Name:    hello-openshift-nfs-pod
Namespace:   default ①
Image(s):   fedora/S3
Node:    ose70.rh7/192.168.234.148 ②
Start Time:   Mon, 21 Mar 2016 09:59:47 -0400
Labels:    name=hello-openshift-nfs-pod
Status:    Running
Reason:
Message:
IP:    10.1.0.4
Replication Controllers: <none>
Containers:
  hello-openshift-nfs-pod:
    Container ID:
docker://a3292104d6c28d9cf49f440b2967a0fc5583540fc3b062db598557b93893bc6f
    Image:  fedora/S3
    Image ID:
docker://403d268c640894cbd76d84a1de3995d2549a93af51c8e16e89842e4c3ed6a00a
    QoS Tier:
      cpu:  BestEffort
      memory:  BestEffort
    State:  Running
      Started:  Mon, 21 Mar 2016 09:59:49 -0400
    Ready:  True
    Restart Count: 0
    Environment Variables:
Conditions:
  Type  Status
  Ready  True
Volumes:
  nfsvol:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName: nfs-pvc ③
    ReadOnly: false
  default-token-a06zb:
    Type: Secret (a secret that should populate this volume)
    SecretName: default-token-a06zb
Events: ④
  FirstSeen LastSeen Count From   SubobjectPath                  Reason  Message
  ───────── ──────── ───── ────   ─────────────                  ──────  ───────
─────────
  4m  4m  1 {scheduler }                         Scheduled Successfully assigned hello-openshift-
nfs-pod to ose70.rh7
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD        Pulled  Container image
"openshift3/ose-pod:v3.1.0.4" already present on machine
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD        Created  Created with docker
id 866a37108041
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD        Started  Started with docker
id 866a37108041
  4m  4m  1 {kubelet ose70.rh7} spec.containers{hello-openshift-nfs-pod}  Pulled  Container image
"fedora/S3" already present on machine
  4m  4m  1 {kubelet ose70.rh7} spec.containers{hello-openshift-nfs-pod}  Created  Created with
```

```
docker id a3292104d6c2
  4m  4m  1 {kubelet ose70.rh7} spec.containers{hello-openshift-nfs-pod}  Started  Started with docker
id a3292104d6c2
```

1. 项目（命名空间）名称。
2. 运行 pod 的 OpenShift Container Platform 节点的 IP 地址。
3. pod 使用的 PVC 名称。
4. 生成 pod 启动的事件列表，以及挂载的 NFS 卷。如果卷无法挂载，则容器无法正确启动。

还有更多内部信息，包括用于授权 pod、pod 的用户和组 ID、SELinux 标签等，如 **oc get pod <name> -o yaml** 命令所示：

```
[root@ose70 nfs]# oc get pod hello-openshift-nfs-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    openshift.io/scc: restricted (1)
  creationTimestamp: 2016-03-21T13:59:47Z
  labels:
    name: hello-openshift-nfs-pod
  name: hello-openshift-nfs-pod
  namespace: default (2)
  resourceVersion: "2814411"
  selflink: /api/v1/namespaces/default/pods/hello-openshift-nfs-pod
  uid: 2c22d2ea-ef6d-11e5-adc7-000c2900f1e3
spec:
  containers:
  - image: fedora/S3
    imagePullPolicy: IfNotPresent
    name: hello-openshift-nfs-pod
    ports:
    - containerPort: 80
      name: web
      protocol: TCP
    resources: {}
    securityContext:
      privileged: false
    terminationMessagePath: /dev/termination-log
    volumeMounts:
    - mountPath: /usr/share/S3/html
      name: nfsvol
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-a06zb
      readOnly: true
  dnsPolicy: ClusterFirst
  host: ose70.rh7
  imagePullSecrets:
  - name: default-dockercfg-xvdew
  nodeName: ose70.rh7
  restartPolicy: Always
  securityContext:
```

```
    supplementalGroups:
    - 100003 3
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  volumes:
  - name: nfsvol
    persistentVolumeClaim:
      claimName: nfs-pvc 4
  - name: default-token-a06zb
    secret:
      secretName: default-token-a06zb
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: 2016-03-21T13:59:49Z
    status: "True"
    type: Ready
  containerStatuses:
  - containerID: docker://a3292104d6c28d9cf49f440b2967a0fc5583540fc3b062db598557b93893bc6f
    image: fedora/S3
    imageID: docker://403d268c640894cbd76d84a1de3995d2549a93af51c8e16e89842e4c3ed6a00a
    lastState: {}
    name: hello-openshift-nfs-pod
    ready: true
    restartCount: 0
    state:
      running:
        startedAt: 2016-03-21T13:59:49Z
  hostIP: 192.168.234.148
  phase: Running
  podIP: 10.1.0.4
  startTime: 2016-03-21T13:59:47Z
```

1 Pod 使用的 SCC。

2 项目（命名空间）名称。

3 pod 的补充组 ID（所有容器）。

4 pod 使用的 PVC 名称。

28.2.6. 创建额外 Pod 以引用 Same PVC

此 pod 定义在同一命名空间中创建，使用不同的容器。但是，您可以通过在下面的 volumes 部分中指定声明名称来使用相同的后备存储：

例 28.4. Pod 对象定义

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox-nfs-pod 1
  labels:
```

```
    name: busybox-nfs-pod
spec:
  containers:
  - name: busybox-nfs-pod
    image: busybox (2)
    command: ["sleep", "60000"]
    volumeMounts:
    - name: nfsvol-2 (3)
      mountPath: /usr/share/busybox (4)
      readOnly: false
  securityContext:
    supplementalGroups: [100003] (5)
    privileged: false
  volumes:
  - name: nfsvol-2
    persistentVolumeClaim:
      claimName: nfs-pvc (6)
```

1. **oc get pod** 显示此 pod 的名称。
2. 此 pod 运行的镜像。
3. 卷的名称。在 **containers** 和 **volumes** 部分中，此名称必须相同。
4. 如容器所示的挂载路径。
5. 要分配给容器的组 ID。
6. 之前创建的 PVC 也被不同的容器使用。

将 pod 定义保存到文件中，如 ***nfs-2.yaml***，并创建 pod：

```
# oc create -f nfs-2.yaml
pod "busybox-nfs-pod" created
```

验证 pod 是否已创建：

```
# oc get pods
NAME             READY     STATUS    RESTARTS   AGE
busybox-nfs-pod  1/1       Running   0          3s
```

oc describe pod 命令中显示更多详细信息：

```
[root@ose70 nfs]# oc describe pod busybox-nfs-pod
Name:    busybox-nfs-pod
Namespace:   default
Image(s):   busybox
Node:    ose70.rh7/192.168.234.148
Start Time:   Mon, 21 Mar 2016 10:19:46 -0400
Labels:    name=busybox-nfs-pod
Status:    Running
Reason:
```

```
Message:
IP:    10.1.0.5
Replication Controllers: <none>
Containers:
  busybox-nfs-pod:
    Container ID:
docker://346d432e5a4824ebf5a47fceb4247e0568ecc64eadcc160e9bab481aecfb0594
    Image:  busybox
    Image ID:  docker://17583c7dd0dae6244203b8029733bdb7d17fccbb2b5d93e2b24cf48b8bfd06e2
    QoS Tier:
      cpu:  BestEffort
      memory:  BestEffort
    State:  Running
      Started:  Mon, 21 Mar 2016 10:19:48 -0400
    Ready:  True
    Restart Count: 0
    Environment Variables:
Conditions:
  Type  Status
  Ready  True
Volumes:
  nfsvol-2:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName: nfs-pvc
    ReadOnly: false
  default-token-32d2z:
    Type: Secret (a secret that should populate this volume)
    SecretName: default-token-32d2z
Events:
  FirstSeen LastSeen Count From   SubobjectPath    Reason  Message
  ————————— ———————— ————— ————   ———————————————    ————————  ———————————
  4m  4m  1 {scheduler }       Scheduled Successfully assigned busybox-nfs-pod to ose70.rh7
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD Pulled  Container image
"openshift3/ose-pod:v3.1.0.4" already present on machine
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD Created  Created with docker id
249b7d7519b1
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD Started  Started with docker id
249b7d7519b1
  4m  4m  1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod} Pulled  Container image "busybox"
already present on machine
  4m  4m  1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod} Created  Created with docker id
346d432e5a48
  4m  4m  1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod} Started  Started with docker id
346d432e5a48
```

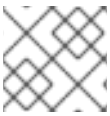
如您所见，两个容器都使用附加到后端上同一 NFS 挂载的相同存储声明。

28.3. 使用 CEPH RBD 完成示例

28.3.1. 概述

本主题提供了使用现有 Ceph 集群作为 OpenShift Container Platform 持久性存储的端到端示例。假设已经设置了可正常工作的 Ceph 集群。如果没有，请参阅 Red Hat Ceph Storage 的概述 。

使用 Ceph Rados 块设备的持久性存储 提供了持久性卷 (PV)、持久性卷声明 (PVC) 的持久性存储，并使用 Ceph RBD 作为持久性存储。

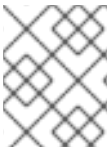


注意

所有 **oc ...** 在 OpenShift Container Platform master 主机上执行命令。

28.3.2. 安装 ceph-common 软件包

ceph-common 库必须安装在 **所有可调度的** OpenShift Container Platform 节点上：



注意

OpenShift Container Platform all-in-one 主机通常不会用于运行 pod 工作负载，因此不作为可调度节点包含。

```
# yum install -y ceph-common
```

28.3.3. 创建 Ceph Secret

ceph auth get-key 命令在 Ceph **MON** 节点上运行，以显示 **client.admin** 用户的键值：

例 28.5. Ceph Secret 定义

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key: QVFBOFF2SlZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ== 1
type: kubernetes.io/rbd 2
```

1. 此 base64 密钥在其中一个 Ceph MON 节点上生成，使用 **ceph auth get-key client.admin | base64** 命令，然后复制输出并将其复制为 secret 密钥的值。
2. Ceph RBD 需要此值才能用于动态置备。

将 secret 定义保存到文件中，如 ***ceph-secret.yaml***，然后创建 secret：

```
$ oc create -f ceph-secret.yaml
secret "ceph-secret" created
```

验证是否已创建 secret：

```
# oc get secret ceph-secret
NAME          TYPE              DATA      AGE
ceph-secret   kubernetes.io/rbd  1         23d
```

28.3.4. 创建持久性卷

接下来，在 OpenShift Container Platform 中创建 PV 对象前，定义持久性卷文件：

例 28.6. 使用 Ceph RBD 的持久性卷对象定义

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-pv 1
spec:
  capacity:
    storage: 2Gi 2
  accessModes:
    - ReadWriteOnce 3
  rbd: 4
    monitors: 5
      - 192.168.122.133:6789
    pool: rbd
    image: ceph-image
    user: admin
    secretRef:
      name: ceph-secret 6
    fsType: ext4 7
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

1. PV 的名称，它在 pod 定义中引用或在各种 **oc** volume 命令中显示。
2. 为这个卷分配的存储量。
3. **accessModes** 用作标签，以匹配 PV 和 PVC。它们目前没有定义任何形式的访问控制。所有块存储都是定义为单个用户（非共享存储）。
4. 这将定义正在使用的卷类型。在本例中，定义了 **rbd** 插件。
5. 这是 Ceph 监控 IP 地址和端口的数组。
6. 这是以上定义的 Ceph secret。它被用来创建从 OpenShift Container Platform 到 Ceph 服务器的安全连接。
7. 这是挂载到 Ceph RBD 块设备上的文件系统类型。

将 PV 定义保存到文件中，如 ***ceph-pv.yaml*** 并创建持久性卷：

```
# oc create -f ceph-pv.yaml
persistentvolume "ceph-pv" created
```

验证持久性卷是否已创建：

```
# oc get pv
NAME                LABELS    CAPACITY     ACCESSMODES   STATUS      CLAIM     REASON
AGE
ceph-pv             <none>    2147483648   RWO           Available                 2s
```

28.3.5. 创建持久性卷声明

持久性卷声明(PVC)指定所需的访问模式和存储容量。目前，仅基于这两个属性，PVC 绑定到一个 PV。当 PV 与 PVC 绑定后，PV 基本上与 PVC 的项目绑定，且无法与另一个 PVC 绑定。PV 和 PVC 有一对一的映射。但是，同一项目中的多个 pod 可以使用相同的 PVC。

例 28.7. PVC 对象定义

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim
spec:
  accessModes:     1
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi  2
```

1 如前面对 PV 提供的一样，**accessModes** 不会强制访问访问，而是作为标签来把一个 PV 与 PVC 进行匹配。

2 这个声明会查找提供 **2Gi** 或更高容量的 PV。

将 PVC 定义保存到文件中，如 ***ceph-claim.yaml*** 并创建 PVC：

```
# oc create -f ceph-claim.yaml
persistentvolumeclaim "ceph-claim" created

#and verify the PVC was created and bound to the expected PV:
# oc get pvc
NAME         LABELS    STATUS    VOLUME    CAPACITY   ACCESSMODES   AGE
ceph-claim   <none>    Bound     ceph-pv   1Gi        RWX           21s
                                             1
```

1 声明已绑定到 **ceph-pv** PV。

28.3.6. 创建 Pod

pod 定义文件或模板文件可用于定义 pod。以下是创建单个容器并挂载 Ceph RBD 卷的 pod 规格，以便进行读写访问：

例 28.8. Pod 对象定义

```
apiVersion: v1
kind: Pod
metadata:
  name: ceph-pod1          1
spec:
  containers:
  - name: ceph-busybox
```

```
    image: busybox          2
    command: ["sleep", "60000"]
    volumeMounts:
    - name: ceph-vol1       3
      mountPath: /usr/share/busybox 4
      readOnly: false
  volumes:
  - name: ceph-vol1         5
    persistentVolumeClaim:
      claimName: ceph-claim 6
```

1. **oc get pod** 显示此 pod 的名称。
2. 此 pod 运行的镜像。在这种情况下，我们把 **busybox** 命名为 sleep。
3. 5 卷的名称。在 **containers** 和 **volumes** 部分中，此名称必须相同。
4. 如容器所示的挂载路径。
6. 绑定到 Ceph RBD 集群的 PVC。

将 pod 定义保存到文件中，如 ***ceph-pod1.yaml*** 并创建 pod：

```
# oc create -f ceph-pod1.yaml
pod "ceph-pod1" created

#verify pod was created
# oc get pod
NAME       READY     STATUS    RESTARTS   AGE
ceph-pod1  1/1       Running   0          2m
                     1
```

1. 一两分钟，pod 将处于 **Running** 状态。

28.3.7. 定义组及所有者 ID（可选）

在使用块存储（如 Ceph RBD）时，物理块存储由 pod **管理**。在 pod 中订阅的组 ID 会成成为容器内 Ceph RBD 挂载的组 ID，**以及**实际的存储本身的组 ID。因此，在 pod 规范中定义组 ID 通常不需要。但是，如果需要使用 fsGroup 定义组 ID，可以使用 **fsGroup** 定义，如以下 pod 定义片段中所示：

例 28.9. 组 ID Pod 定义

```
...
spec:
  containers:
    - name:
    ...
  securityContext: 1
    fsGroup: 7777 2
...
```

1. **securityContext** 必须在 pod 一级定义，而不是在某个特定容器中定义。
2. pod 中的所有容器将具有相同的 **fsGroup** ID。

28.3.8. 将 ceph-user-secret 设置为项目的默认

如果您希望让永久存储可供您修改默认项目模板的每个项目使用。您可以阅读更多修改默认项目模板。了解有关 修改默认项目模板 的更多信息。将此功能添加到您的默认项目模板后，每个有权访问的用户都能够访问 Ceph 集群。

默认项目示例

```
...
apiVersion: v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects:
- apiVersion: v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
    creationTimestamp: null
    name: ${PROJECT_NAME}
  spec: {}
  status: {}
- apiVersion: v1
  kind: Secret
  metadata:
    name: ceph-user-secret
  data:
    key: yoursupersecretbase64keygoeshere (1)
  type:
    kubernetes.io/rbd
...
```

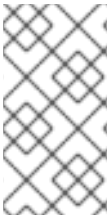
1. 将 Ceph 用户密钥放在 base64 格式。

28.4. 使用 CEPH RBD 进行动态置备

28.4.1. 概述

本主题提供了将现有 Ceph 集群用于 OpenShift Container Platform 持久性存储的完整示例。假设已经设置了可正常工作的 Ceph 集群。如果没有，请参阅 Red Hat Ceph Storage 的概述 。

使用 Ceph Rados 块设备的持久性存储 提供了持久性卷(PV)、持久性卷声明(PVC)的持久性存储，以及如何使用 Ceph Rados 块设备(RBD)作为持久性存储。



注意

- 在 OpenShift Container Platform master 主机上运行所有 **oc** 命令。
- OpenShift Container Platform all-in-one 主机通常不会用于运行 pod 工作负载，因此不作为可调度节点包含。

28.4.2. 为动态卷创建池

1. 安装最新的 ceph-common 软件包：

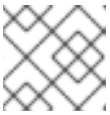
```
yum install -y ceph-common
```

注意

ceph-common 库必须安装在 **所有可调度的** OpenShift Container Platform 节点上。

2. 从管理员或 **MON** 节点，为动态卷创建新池，例如：

```
$ ceph osd pool create kube 1024
$ ceph auth get-or-create client.kube mon 'allow r, allow command "osd blacklist"' osd 'allow
class-read object_prefix rbd_children, allow rwx pool=kube' -o ceph.client.kube.keyring
```



注意

使用默认 RBD 池是 选项，但不推荐这样做。

28.4.3. 使用现有的 Ceph 集群进行动态持久性存储

使用现有的 Ceph 集群进行动态持久性存储：

1. 生成 client.admin base64 编码的密钥：

```
$ ceph auth get client.admin
```

Ceph secret 定义示例

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
  namespace: kube-system
data:
  key: QVFBOFF2SlZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ== (1)
type: kubernetes.io/rbd (2)
```

(1) 此 base64 密钥在其中一个 Ceph MON 节点上生成，使用 **ceph auth get-key client.admin | base64** 命令，然后复制输出并将其复制为 secret 密钥的值。

(2) Ceph RBD 需要此值才能用于动态置备。

2. 为 client.admin 创建 Ceph secret：

```
$ oc create -f ceph-secret.yaml
secret "ceph-secret" created
```

3. 验证是否已创建 secret：

```
$ oc get secret ceph-secret
NAME          TYPE                DATA      AGE
ceph-secret   kubernetes.io/rbd   1         5d
```

4. 创建存储类：

```
$ oc create -f ceph-storageclass.yaml
storageclass "dynamic" created
```

Ceph 存储类示例

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: dynamic
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/rbd
parameters:
  monitors: 192.168.1.11:6789,192.168.1.12:6789,192.168.1.13:6789 1
  adminId: admin 2
  adminSecretName: ceph-secret 3
  adminSecretNamespace: kube-system 4
  pool: kube 5
  userId: kube 6
  userSecretName: ceph-user-secret 7
```

1. 以逗号分隔的 IP 地址 Ceph 监视器列表。此值是必需的。
2. 能够在池中创建镜像的 Ceph 客户端 ID。默认值为 **admin**。
3. **adminId** 的机密名称。此值是必需的。您提供的 secret 必须具有 **kubernetes.io/rbd**。
4. **adminSecret** 的命名空间。默认为 **default**。
5. Ceph RBD 池。默认值为 **rbd**，但不推荐使用此值。
6. 用于映射 Ceph RBD 镜像的 Ceph 客户端 ID。默认值与 **adminId** 的 secret 名称相同。
7. 用于映射 Ceph RBD 镜像的 **userId** 的 Ceph secret 名称。它必须与 PVC 位于同一个命名空间中。除非在新项目中将 Ceph secret 设置为默认值，否则您必须提供此参数值。

5. 验证存储类是否已创建：

```
$ oc get storageclasses
NAME                 TYPE
dynamic (default)    kubernetes.io/rbd
```

6. 创建 PVC 对象定义：

PVC 对象定义示例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim-dynamic
spec:
  accessModes: 1
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi 2
```

1 **accessModes** 不强制访问权利，而是作为标签来把一个 PV 与 PVC 匹配。

2 这个声明会寻找提供 **2Gi** 或更高容量的 PV。

7. 创建 PVC：

```
$ oc create -f ceph-pvc.yaml
persistentvolumeclaim "ceph-claim-dynamic" created
```

8. 验证 PVC 是否已创建并绑定到预期的 PV:

```
$ oc get pvc
NAME         STATUS  VOLUME                                     CAPACITY ACCESSMODES  AGE
ceph-claim   Bound   pvc-f548d663-3cac-11e7-9937-0024e8650c7a 2Gi      RWO          1m
```

9. 创建 pod 对象定义：

Pod 对象定义示例

```
apiVersion: v1
kind: Pod
metadata:
  name: ceph-pod1 1
spec:
  containers:
  - name: ceph-busybox
    image: busybox 2
    command: ["sleep", "60000"]
    volumeMounts:
    - name: ceph-vol1 3
      mountPath: /usr/share/busybox 4
      readOnly: false
  volumes:
```

```
  - name: ceph-vol1
    persistentVolumeClaim:
      claimName: ceph-claim-dynamic 5
```

1. **oc get pod** 显示此 pod 的名称。
2. 此 pod 运行的镜像。在这种情况下，**busybox** 设置为 **sleep**。
3. 卷的名称。这个名称在 **containers** 和 **volumes** 部分中必须相同。
4. 容器中的挂载路径。
5. 绑定到 Ceph RBD 集群的 PVC。

10. 创建 pod：

```
$ oc create -f ceph-pod1.yaml
pod "ceph-pod1" created
```

11. 验证 pod 是否已创建：

```
$ oc get pod
NAME       READY     STATUS    RESTARTS   AGE
ceph-pod1  1/1       Running   0          2m
```

过一分钟后，pod 状态更改为 **Running**。

28.4.4. 将 ceph-user-secret 设置为项目的默认值

要使持久性存储可供每个项目使用，您必须修改默认的项目模板。将此功能添加到您的默认项目模板后，每个有权访问的用户都能够访问 Ceph 集群。如需更多信息，请参阅修改默认项目模板。

默认项目示例

```
...
apiVersion: v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects:
- apiVersion: v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
    creationTimestamp: null
    name: ${PROJECT_NAME}
  spec: {}
  status: {}
- apiVersion: v1
  kind: Secret
```

```
  metadata:
    name: ceph-user-secret
  data:
    key: QVFCbEV4OVpmaGJtQ0JBQW55d2Z0NHZtcS96cE42SW1JVUQvekE9PQ== (1)
  type:
    kubernetes.io/rbd
...
```

(1) 将 Ceph 用户密钥放在 base64 格式。

28.5. 使用 GLUSTERFS 的完整示例

28.5.1. 概述

本主题提供了一个端到端示例，有关如何使用现有的聚合模式、独立模式或独立 Red Hat Gluster Storage 集群作为 OpenShift Container Platform 的持久性存储。假设已经设置了一个有效的 Red Hat Gluster Storage 集群。有关安装聚合模式或独立模式的帮助，请参阅使用 Red Hat Gluster Storage 的持久性存储。有关独立红帽 Gluster 存储，请查阅《红帽 Gluster 存储管理指南》。

有关如何动态置备 GlusterFS 卷的端到端示例，请参阅使用 GlusterFS 进行动态置备的完整示例。

注意

所有 **oc** 命令在 OpenShift Container Platform master 主机上执行。

28.5.2. 先决条件

要访问 GlusterFS 卷，必须在所有可调度节点上使用 **mount.glusterfs** 命令。对于基于 RPM 的系统，必须安装 **glusterfs-fuse** 软件包：

```
# yum install glusterfs-fuse
```

这个软件包会在每个 RHEL 系统上安装。但是，如果您的服务器使用 x86_64 架构，则建议您将 Red Hat Gluster Storage 升级到最新的可用版本。要做到这一点，必须启用以下 RPM 存储库：

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-server-rpms
```

如果已在节点上安装了 **glusterfs-fuse**，请确保安装了最新版本：

```
# yum update glusterfs-fuse
```

默认情况下，SELinux 不允许从 pod 写入远程 Red Hat Gluster 存储服务器。要启用使用 SELinux 对 Red Hat Gluster 存储卷写入，请在运行 GlusterFS 的每个节点中运行以下命令：

```
$ sudo setsebool -P virt_sandbox_use_fusefs on (1)
$ sudo setsebool -P virt_use_fusefs on
```

(1) 使用 **-P** 选项可使布尔值在系统重启后持久保留。

注意

virt_sandbox_use_fusefs 布尔值由 **docker-selinux** 软件包提供。如果您收到一个错误，说明它没有被定义，请确保已安装此软件包。

注意

如果您使用 Atomic Host，升级 Atomic 主机时将清除 SELinux 布尔值。升级 Atomic Host 时，您必须再次设置这些布尔值。

28.5.3. 静态置备

1. 要启用静态置备，首先请创建一个 GlusterFS 卷。请参阅 Red Hat Gluster Storage Administration Guide 了解如何使用 **gluster** 命令行的信息；请参阅 heketi 项目网站来了解如何使用 **heketi-cli** 的信息。在本例中，卷将命名为 **myVol1**。

2. 在 **gluster-endpoints.yaml** 中定义以下服务和端点：

```
---
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster (1)
spec:
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster (2)
subsets:
  - addresses:
      - ip: 192.168.122.221 (3)
    ports:
      - port: 1 (4)
  - addresses:
      - ip: 192.168.122.222 (5)
    ports:
      - port: 1 (6)
  - addresses:
      - ip: 192.168.122.223 (7)
    ports:
      - port: 1 (8)
```

(1) (2) 这些名称必须匹配。

(3) (5) (7) **ip** 值必须是 Red Hat Gluster Storage 服务器的实际 IP 地址，而不是主机名。

(4) (6) (8) 端口号被忽略。

3. 在 OpenShift Container Platform master 主机上创建服务和端点：

```
$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created
```

4. 验证服务和端点是否已创建：

```
$ oc get services
NAME                CLUSTER_IP      EXTERNAL_IP   PORT(S)    SELECTOR        AGE
glusterfs-cluster   172.30.205.34   <none>        1/TCP      <none>          44s

$ oc get endpoints
NAME                ENDPOINTS                                           AGE
docker-registry     10.1.0.3:5000                                       4h
glusterfs-cluster   192.168.122.221:1,192.168.122.222:1,192.168.122.223:1   11s
kubernetes          172.16.35.3:8443                                    4d
```

注意

端点每个项目都是唯一的。访问 GlusterFS 卷的每个项目都需要自己的端点。

5. 若要访问卷，容器必须使用用户 ID(UID)或组 ID(GID)运行，该容器有权访问卷上的文件系统。这些信息可以通过以下方法发现：

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0    myVol1 (1) (2)
```

(1) UID 为 592。

(2) GID 是 590。

6. 在 **gluster-pv.yaml** 中定义以下 PersistentVolume(PV)：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume (1)
  annotations:
    pv.beta.kubernetes.io/gid: "590" (2)
spec:
  capacity:
    storage: 2Gi (3)
  accessModes: (4)
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster (5)
    path: myVol1 (6)
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

1. 卷的名称。
2. GlusterFS 卷根上的 GID。
3. 为这个卷分配的存储量。
4. **accessModes** 用作标签，以匹配 PV 和 PVC。它们目前没有定义任何形式的访问控制。
5. 之前创建的 Endpoints 资源。
6. 将要访问的 GlusterFS 卷。

7. 在 OpenShift Container Platform master 主机上创建 PV：

```
$ oc create -f gluster-pv.yaml
```

8. 确定创建了 PV：

```
$ oc get pv
NAME                LABELS    CAPACITY    ACCESSMODES   STATUS      CLAIM
REASON    AGE
gluster-default-volume   <none>    2147483648   RWX         Available                 2s
```

9. 创建一个 PersistentVolumeClaim(PVC)，它将绑定到 **gluster-claim.yaml** 中的新 PV：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim (1)
spec:
  accessModes:
  - ReadWriteMany (2)
  resources:
    requests:
      storage: 1Gi (3)
```

1. 声明名称由 pod 在其 **volumes** 部分下引用。
2. 必须与 PV 的 **accessModes** 匹配。
3. 这个声明会查找提供 **1Gi** 或更高容量的 PV。

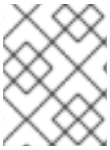
10. 在 OpenShift Container Platform master 主机上创建 PVC：

```
$ oc create -f gluster-claim.yaml
```

11. 验证 PV 和 PVC 是否已绑定：

```
$ oc get pv
NAME         LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM          REASON
AGE
gluster-pv   <none>    1Gi        RWX           Available   gluster-claim          37s
```

```
$ oc get pvc
NAME          LABELS    STATUS    VOLUME      CAPACITY   ACCESSMODES   AGE
gluster-claim   <none>    Bound     gluster-pv   1Gi        RWX           24s
```



注意

PVC 每个项目都是唯一的。访问 GlusterFS 卷的每个项目都需要自己的 PVC。PV 不绑定到单个项目，因此多个项目的 PVC 可能会引用同一 PV。

28.5.4. 使用存储

此时，您创建了一个动态创建的 GlusterFS 卷，它绑定到 PVC。现在，您可以在 pod 中使用这个 PVC。

1. 创建 pod 对象定义：

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-pod
  labels:
    name: hello-openshift-pod
spec:
  containers:
  - name: hello-openshift-pod
    image: openshift/hello-openshift
    ports:
    - name: web
      containerPort: 80
    volumeMounts:
    - name: gluster-vol1
      mountPath: /usr/share/nginx/html
      readOnly: false
  volumes:
  - name: gluster-vol1
    persistentVolumeClaim:
      claimName: gluster1 (1)
```

(1) 上一步中创建的 PVC 名称。

2. 在 OpenShift Container Platform master 主机上创建 pod：

```
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
```

3. 查看 pod。花几分钟时间，因为镜像（如果尚不存在）可能需要下载它：

```
# oc get pods -o wide
NAME                     READY     STATUS    RESTARTS   AGE       IP           NODE
hello-openshift-pod      1/1       Running   0          9m        10.38.0.0    node1
```

4. **oc exec** 进入容器并在 pod 的 **mountPath** 定义中创建 ***index.html*** 文件：

```
$ oc exec -ti hello-openshift-pod /bin/sh
```

```
$ cd /usr/share/nginx/html
$ echo 'Hello OpenShift!!!' > index.html
$ ls
index.html
$ exit
```

5. 现在，**curl** pod 的 URL:

```
# curl http://10.38.0.0
Hello OpenShift!!!
```

6. 删除 pod，重新创建它并等待它出现：

```
# oc delete pod hello-openshift-pod
pod "hello-openshift-pod" deleted
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
# oc get pods -o wide
NAME                      READY     STATUS    RESTARTS   AGE       IP            NODE
hello-openshift-pod       1/1       Running   0          9m        10.37.0.0     node1
```

7. 现在再次 **curl** pod，它应该仍然具有与以前相同的数据。请注意，其 IP 地址可能已更改：

```
# curl http://10.37.0.0
Hello OpenShift!!!
```

8. 通过在任何节点上执行以下操作，检查 ***index.html*** 文件是否已写入 GlusterFS 存储：

```
$ mount | grep heketi
/dev/mapper/VolGroup00-LogVol00 on /var/lib/heketi type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_1e730a5462c352835055018e1874e578 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_1e730a5462c35283505
5018e1874e578 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,swidth=512,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_d8c06e606ff4cc29ccb9d018c73ee292 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c06e606ff4cc29ccb9d
018c73ee292 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,swidth=512,noquota)

$ cd
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c06e606ff4cc29ccb9d
018c73ee292/brick
$ ls
index.html
$ cat index.html
Hello OpenShift!!!
```

28.6. 使用 GLUSTERFS 进行动态置备完成示例

28.6.1. 概述

本主题提供了有关如何使用现有聚合模式、独立模式或独立 Red Hat Gluster Storage 集群作为 OpenShift Container Platform 的动态持久性存储的端到端示例。假设已经设置了一个有效的 Red Hat Gluster Storage 集群。有关安装聚合模式或独立模式的帮助，请参阅使用 Red Hat Gluster Storage 的持久性存储。有关独立红帽 Gluster 存储，请查阅《红帽 Gluster 存储管理指南》。

注意

所有 **oc** 命令在 OpenShift Container Platform master 主机上执行。

28.6.2. 先决条件

要访问 GlusterFS 卷，必须在所有可调度节点上使用 **mount.glusterfs** 命令。对于基于 RPM 的系统，必须安装 **glusterfs-fuse** 软件包：

```
# yum install glusterfs-fuse
```

这个软件包会在每个 RHEL 系统上安装。但是，如果您的服务器使用 x86_64 架构，则建议您将 Red Hat Gluster Storage 升级到最新的可用版本。要做到这一点，必须启用以下 RPM 存储库：

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-server-rpms
```

如果已在节点上安装了 **glusterfs-fuse**，请确保安装了最新版本：

```
# yum update glusterfs-fuse
```

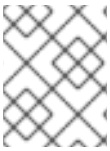
默认情况下，SELinux 不允许从 pod 写入远程 Red Hat Gluster 存储服务器。要启用使用 SELinux 对 Red Hat Gluster 存储卷写入，请在运行 GlusterFS 的每个节点中运行以下命令：

```
$ sudo setsebool -P virt_sandbox_use_fusefs on 1
$ sudo setsebool -P virt_use_fusefs on
```

1 使用 **-P** 选项可使布尔值在系统重启后持久保留。

注意

virt_sandbox_use_fusefs 布尔值由 **docker-selinux** 软件包提供。如果您收到一个错误，说明它没有被定义，请确保已安装此软件包。



注意

如果您使用 Atomic Host，升级 Atomic 主机时将清除 SELinux 布尔值。升级 Atomic Host 时，您必须再次设置这些布尔值。

28.6.3. 动态置备

1. 要启用动态置备，首先请创建一个 **StorageClass** 对象定义。以下定义基于本示例与 OpenShift Container Platform 搭配使用所需的最低要求。如需了解更多参数和规格定义，请参阅动态置备和创建存储类。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
```

```
metadata:
  name: glusterfs
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://10.42.0.0:8080" 1
  restauthenabled: "false" 2
```

1 heketi 服务器 URL。

2 由于本例中未打开身份验证，因此设置为 **false**。

2. 在 OpenShift Container Platform master 主机上创建 StorageClass：

```
# oc create -f gluster-storage-class.yaml
storageclass "glusterfs" created
```

3. 使用新创建的 StorageClass 创建 PVC。例如：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster1
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 30Gi
  storageClassName: glusterfs
```

4. 在 OpenShift Container Platform master 主机上创建 PVC：

```
# oc create -f glusterfs-dyn-pvc.yaml
persistentvolumeclaim "gluster1" created
```

5. 查看 PVC，查看卷是否动态创建并绑定到 PVC：

```
# oc get pvc
NAME      STATUS   VOLUME                                     CAPACITY   ACCESSMODES
STORAGECLASS   AGE
gluster1  Bound    pvc-78852230-d8e2-11e6-a3fa-0800279cf26f   30Gi       RWX
glusterfs      42s
```

28.6.4. 使用存储

此时，您创建了一个动态创建的 GlusterFS 卷，它绑定到 PVC。现在，您可以在 pod 中使用这个 PVC。

1. 创建 pod 对象定义：

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-pod
```

```
  labels:
    name: hello-openshift-pod
spec:
  containers:
  - name: hello-openshift-pod
    image: openshift/hello-openshift
    ports:
    - name: web
      containerPort: 80
    volumeMounts:
    - name: gluster-vol1
      mountPath: /usr/share/nginx/html
      readOnly: false
  volumes:
  - name: gluster-vol1
    persistentVolumeClaim:
      claimName: gluster1 (1)
```

(1) 上一步中创建的 PVC 名称。

2. 在 OpenShift Container Platform master 主机上创建 pod：

```
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
```

3. 查看 pod。花几分钟时间，因为镜像（如果尚不存在）可能需要下载它：

```
# oc get pods -o wide
NAME                     READY     STATUS    RESTARTS   AGE       IP           NODE
hello-openshift-pod      1/1       Running   0          9m        10.38.0.0    node1
```

4. **oc exec** 进入容器并在 pod 的 **mountPath** 定义中创建 ***index.html*** 文件：

```
$ oc exec -ti hello-openshift-pod /bin/sh
$ cd /usr/share/nginx/html
$ echo 'Hello OpenShift!!!' > index.html
$ ls
index.html
$ exit
```

5. 现在，**curl** pod 的 URL:

```
# curl http://10.38.0.0
Hello OpenShift!!!
```

6. 删除 pod，重新创建它并等待它出现：

```
# oc delete pod hello-openshift-pod
pod "hello-openshift-pod" deleted
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
```

```
# oc get pods -o wide
NAME                     READY     STATUS    RESTARTS   AGE       IP           NODE
hello-openshift-pod      1/1       Running   0          9m        10.37.0.0    node1
```

7. 现在再次 **curl** pod，它应该仍然具有与以前相同的数据。请注意，其 IP 地址可能已更改：

```
# curl http://10.37.0.0
Hello OpenShift!!!
```

8. 通过在任何节点上执行以下操作，检查 ***index.html*** 文件是否已写入 GlusterFS 存储：

```
$ mount | grep heketi
/dev/mapper/VolGroup00-LogVol00 on /var/lib/heketi type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_1e730a5462c352835055018e1874e578 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_1e730a5462c35283505
5018e1874e578 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,swidth=512,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_d8c06e606ff4cc29ccb9d018c73ee292 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c06e606ff4cc29ccb9d
018c73ee292 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,swidth=512,noquota)

$ cd
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c06e606ff4cc29ccb9d
018c73ee292/brick
$ ls
index.html
$ cat index.html
Hello OpenShift!!!
```

28.7. 在特权 POD 中挂载卷

28.7.1. 概述

持久性卷可以挂载到附加了**特权**安全上下文约束 (SCC)的 pod。

注意

虽然本主题使用 GlusterFS 作为将卷挂载到特权 pod 的示例用例，但可以使用 任何支持的存储插件。

28.7.2. 先决条件

- 现有 Gluster 卷。
- 所有主机上均安装 **glusterfs-fuse**。
- GlusterFS 定义：
  - 端点和服务:***gluster-endpoints-service.yaml*** 和 ***gluster-endpoints.yaml***

- 持久卷 ：*gluster-pv.yaml*
- 持久性卷声明 ：*gluster-pvc.yaml*
- 特权 pod:*gluster-S3-pod.yaml*

- 具有 cluster-admin 角色绑定的用户。对于本指南，该用户名为 **admin**。

28.7.3. 创建持久性卷

创建 **PersistentVolume** 可让用户访问存储，无论项目是什么。

1. 以 admin 身份，创建服务、端点对象和持久性卷：

```
$ oc create -f gluster-endpoints-service.yaml
$ oc create -f gluster-endpoints.yaml
$ oc create -f gluster-pv.yaml
```

2. 验证对象是否已创建：

```
$ oc get svc
NAME              CLUSTER_IP      EXTERNAL_IP   PORT(S)   SELECTOR   AGE
gluster-cluster   172.30.151.58   <none>        1/TCP     <none>     24s
```

```
$ oc get ep
NAME              ENDPOINTS                         AGE
gluster-cluster   192.168.59.102:1,192.168.59.103:1   2m
```

```
$ oc get pv
NAME                    LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
gluster-default-volume   <none>    2Gi        RWX           Available                 2d
```

28.7.4. 创建常规用户

将 常规用户 添加到 **特权** SCC（或添加到 SCC 给定访问权限的组中）允许他们运行 **特权** Pod：

1. 以 admin 身份，将用户添加到 SCC：

```
$ oc adm policy add-scc-to-user privileged <username>
```

2. 以普通用户身份登录：

```
$ oc login -u <username> -p <password>
```

3. 然后，创建一个新项目：

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

28.7.5. 创建持久性卷声明

1. 作为常规用户，创建 **PersistentVolumeClaim** 以访问卷：

```
$ oc create -f gluster-pvc.yaml -n <project_name>
```

2. 定义 pod 以访问声明：

例 28.10. Pod 定义

```
apiVersion: v1
id: gluster-S3-pvc
kind: Pod
metadata:
  name: gluster-nginx-priv
spec:
  containers:
    - name: gluster-nginx-priv
      image: fedora/nginx
      volumeMounts:
        - mountPath: /mnt/gluster  (1)
          name: gluster-volume-claim
      securityContext:
        privileged: true
  volumes:
    - name: gluster-volume-claim
      persistentVolumeClaim:
        claimName: gluster-claim  (2)
```

(1) pod 中的卷挂载。

(2) **gluster-claim** 必须反映 **PersistentVolume** 的名称。

3. 创建 pod 后，挂载目录会被创建，卷将连接到那个挂载点。
以普通用户身份，从定义中创建 pod：

```
$ oc create -f gluster-S3-pod.yaml
```

4. 验证 pod 是否已成功创建：

```
$ oc get pods
NAME             READY     STATUS    RESTARTS   AGE
gluster-S3-pod   1/1       Running   0          36m
```

创建 pod 可能需要几分钟时间。

28.7.6. 验证设置

28.7.6.1. 检查 Pod SCC

1. 导出 pod 配置：

```
$ oc get -o yaml --export pod <pod_name>
```

2. 检查输出。检查 **openshift.io/scc** 是否具有 **privileged** 的值：

例 28.11. 导出片段

```
metadata:
  annotations:
    openshift.io/scc: privileged
```

28.7.6.2. 验证挂载

1. 访问 pod，检查卷是否已挂载：

```
$ oc rsh <pod_name>
[root@gluster-S3-pvc /]# mount
```

2. 检查 Gluster 卷的输出：

例 28.12. 卷挂载

```
192.168.59.102:gv0 on /mnt/gluster type fuse.gluster
(rw,relatime,user_id=0,group_id=0,default_permissions,allow_other,max_read=131072)
```

28.8. 挂载传播

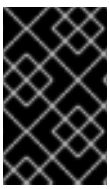
28.8.1. 概述

挂载传播允许将容器挂载的卷共享到同一 pod 中的其他容器，甚至同一节点上的其他 pod 共享。

28.8.2. 值

卷的挂载传播由 **Container.volumeMounts** 中的 **mountPropagation** 字段控制。其值为：

- **none** - 此卷挂载不会接收挂载到此卷的任何后续挂载，或者主机中的任何子目录。同样，主机上看不到容器创建的挂载。这是默认的模式，等同于 Linux 内核中的 **私有** 挂载传播。
- **HostToContainer** - 此卷挂载接收挂载到这个卷的所有后续挂载或其任何子目录。换句话说，如果主机在卷挂载中挂载任何内容，则容器会确认它已被挂载。这个模式等同于 Linux 内核中的 **rslave** mount propagation。
- **双向** - 此卷挂载的行为与 **HostToContainer** 挂载相同。另外，容器创建的所有卷挂载都会传播到主机以及所有使用相同卷的 pod 的容器。这个模式的典型用例是使用 FlexVolume 或 CSI 驱动程序的 Pod，或需要在使用 **hostPath** 卷在主机上挂载内容的 Pod。这个模式等同于 Linux 内核中的 **rshared** 挂载传播。



重要

双向 挂载传播可能会存在危险。它可以损坏主机操作系统，因此仅允许在特权容器中。强烈建议熟悉 Linux 内核的行为。另外，pod 中容器创建的任何卷挂载都必须被在终止时被销毁或卸载。

28.8.3. Configuration

在挂载传播前，可以在一些部署中正常工作，如 CoreOS、Red Hat Enterprise Linux/Centos 或 Ubuntu，挂载共享必须在 Docker 中正确配置。

流程

1. 编辑 Docker 的 systemd 服务文件。设置 **MountFlags**，如下所示：

```
MountFlags=shared
```

   或者，删除 **MountFlags=slave** （如果存在）。

2. 重启 Docker 守护进程：

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

28.9. 将集成的 OPENSHIFT CONTAINER REGISTRY 切换到 GLUSTERFS

28.9.1. 概述

本节介绍了如何将 GlusterFS 卷附加到集成的 OpenShift Container Registry。这可以通过任何聚合模式、独立模式或独立 Red Hat Gluster Storage 来完成。假设 registry 已启动并且创建了卷。

28.9.2. 先决条件

- **在不配置存储的情况下** 部署现有 registry。
- 现有的 GlusterFS 卷
- 在所有可调度节点上安装 GlusterFS **-fuse**。
- 具有 **cluster-admin** 角色绑定的用户。
  - 对于本指南，该用户是 **admin**。

注意

所有 **oc** 命令都是以 **admin** 用户身份在 master 节点上执行的。

28.9.3. 手动置备 GlusterFS PersistentVolumeClaim

1. 要启用静态置备，首先请创建一个 GlusterFS 卷。请参阅 Red Hat Gluster Storage Administration Guide 了解如何使用 **gluster** 命令行的信息；请参阅 heketi 项目网站来了解如何使用 **heketi-cli** 的信息。在本例中，卷将命名为 **myVol1**。

2. 在 **gluster-endpoints.yaml** 中定义以下服务和端点：

```
---
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster (1)
spec:
```

```
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster (2)
subsets:
  - addresses:
      - ip: 192.168.122.221 (3)
    ports:
      - port: 1 (4)
  - addresses:
      - ip: 192.168.122.222 (5)
    ports:
      - port: 1 (6)
  - addresses:
      - ip: 192.168.122.223 (7)
    ports:
      - port: 1 (8)
```

(1)(2)这些名称必须匹配。

(3)(5)(7)**ip** 值必须是 Red Hat Gluster Storage 服务器的实际 IP 地址，而不是主机名。

(4)(6)(8)端口号被忽略。

3. 在 OpenShift Container Platform master 主机上创建服务和端点：

```
$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created
```

4. 验证服务和端点是否已创建：

```
$ oc get services
NAME                CLUSTER_IP      EXTERNAL_IP   PORT(S)    SELECTOR        AGE
glusterfs-cluster   172.30.205.34   <none>        1/TCP      <none>          44s

$ oc get endpoints
NAME                ENDPOINTS                                         AGE
docker-registry     10.1.0.3:5000                                     4h
glusterfs-cluster   192.168.122.221:1,192.168.122.222:1,192.168.122.223:1   11s
kubernetes          172.16.35.3:8443                                  4d
```

注意

端点每个项目都是唯一的。访问 GlusterFS 卷的每个项目都需要自己的端点。

5. 若要访问卷，容器必须使用用户 ID(UID)或组 ID(GID)运行，该容器有权访问卷上的文件系统。这些信息可以通过以下方法发现：

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0    myVol1 (1) (2)
```

(1) UID 为 592。

(2) GID 是 590。

6. 在 **gluster-pv.yaml** 中定义以下 PersistentVolume(PV)：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume (1)
  annotations:
    pv.beta.kubernetes.io/gid: "590" (2)
spec:
  capacity:
    storage: 2Gi (3)
  accessModes: (4)
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster (5)
    path: myVol1 (6)
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

(1) 卷的名称。

(2) GlusterFS 卷根上的 GID。

(3) 为这个卷分配的存储量。

(4) **accessModes** 用作标签，以匹配 PV 和 PVC。它们目前没有定义任何形式的访问控制。

(5) 之前创建的 Endpoints 资源。

(6) 将要访问的 GlusterFS 卷。

7. 在 OpenShift Container Platform master 主机上创建 PV：

```
$ oc create -f gluster-pv.yaml
```

8. 确定创建了 PV：

```
$ oc get pv
NAME                LABELS    CAPACITY     ACCESSMODES   STATUS      CLAIM
REASON    AGE
gluster-default-volume   <none>    2147483648   RWX           Available                 2s
```

9. 创建一个 PersistentVolumeClaim(PVC)，它将绑定到 **gluster-claim.yaml** 中的新 PV：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim (1)
spec:
  accessModes:
  - ReadWriteMany (2)
  resources:
    requests:
      storage: 1Gi (3)
```

(1) 声明名称由 pod 在其 **volumes** 部分下引用。

(2) 必须与 PV 的 **accessModes** 匹配。

(3) 这个声明会查找提供 **1Gi** 或更高容量的 PV。

10. 在 OpenShift Container Platform master 主机上创建 PVC：

```
$ oc create -f gluster-claim.yaml
```

11. 验证 PV 和 PVC 是否已绑定：

```
$ oc get pv
NAME        LABELS    CAPACITY   ACCESSMODES   STATUS     CLAIM          REASON
AGE
gluster-pv  <none>    1Gi        RWX           Available  gluster-claim          37s

$ oc get pvc
NAME          LABELS    STATUS    VOLUME      CAPACITY   ACCESSMODES   AGE
gluster-claim  <none>    Bound     gluster-pv  1Gi        RWX           24s
```

注意

PVC 每个项目都是唯一的。访问 GlusterFS 卷的每个项目都需要自己的 PVC。PV 不绑定到单个项目，因此多个项目的 PVC 可能会引用同一 PV。

28.9.4. 将 PersistentVolumeClaim 附加到 Registry

在继续之前，请确保 **docker-registry** 服务正在运行。

```
$ oc get svc
NAME              CLUSTER_IP      EXTERNAL_IP   PORT(S)             SELECTOR                 AGE
docker-registry   172.30.167.194  <none>        5000/TCP            docker-registry=default  18m
```

注意

如果 **docker-registry** 服务或其相关 pod 没有运行，请参阅 registry 设置说明，以便在继续操作前返回用于故障排除的 registry 设置说明。

然后附加 PVC：

```
$ oc set volume deploymentconfigs/docker-registry --add --name=registry-storage -t pvc \
    --claim-name=gluster-claim --overwrite
```

设置 Registry 提供了有关使用 OpenShift Container Registry 的更多信息。

28.10. 根据标签绑定持久性卷

28.10.1. 概述

本主题通过定义 PVC 中的标签以及 PVC 中的匹配选择器，提供了一个端到端示例，用于将 持久性卷声明(PVC) 绑定到持久性卷(PV)。此功能可用于所有 存储选项。假设 OpenShift Container Platform 集群包含可供 PVC 绑定的持久性存储资源。

标签和选择器的备注

标签(label)是一个 OpenShift Container Platform 功能，它支持用户定义的标签（键值对），作为对象规格的一部分。它们的主要目的是通过在它们中定义相同的标签来启用任意对象分组。然后，这些标签可以通过选择器来匹配所有带有指定标签值的对象。我们可以利用此功能来允许我们的 PVC 绑定到 PV。如需更深入地查看标签，请参阅 Pod 和服务。

注意

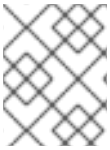
在本例中，我们将使用修改后的 GlusterFS PV 和 PVC 规格。但是，针对所有存储选项，选择器和标签的实施是通用的。请参阅卷供应商 的相关存储选项，以了解更多有关其唯一配置的信息。

28.10.1.1. 假设

假设您有：

- 至少有一个 **master** 和 **一个节点**的现有 OpenShift Container Platform 集群
- 至少一个支持 的存储卷
- 具有 **cluster-admin** 权限的用户

28.10.2. 定义规格



注意

这些规范是为 **GlusterFS** 量身定制的。请参阅卷提供程序 的相关存储选项，以了解更多有关其唯一配置的信息。

28.10.2.1. 带有标签的持久性卷

例 28.13. glusterfs-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-volume
```

```
  labels: (1)
    storage-tier: gold
    aws-availability-zone: us-east-1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster (2)
    path: myVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

(1) 使用标签来标识卷之间共享的通用属性或特征。在本例中，我们定义 Gluster 卷，使其具有名为 **storage-tier** 的自定义属性（密钥），其值为 **gold**。声明将能够选择具有 **storage-tier=gold** 的 PV 来匹配这个 PV。

(2) 端点定义 Gluster 可信池，如下所述。

28.10.2.2. 使用 Selectors 的持久性卷声明

具有 **selector** 小节的声明（参见以下示例）会尝试匹配现有的、未声明的 PV。PVC 选择器存在会忽略 PV 的容量。但是，**accessModes** 仍在匹配标准中考虑。

务必要注意，声明必须与其 **selector** 小节中包含的**所有**键值对匹配。如果没有 PV 与声明匹配，则 PVC 将保持未绑定(Pending)。随后可以创建 PV，声明将自动检查标签匹配。

例 28.14. glusterfs-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector: (1)
    matchLabels:
      storage-tier: gold
      aws-availability-zone: us-east-1
```

(1) **选择器** 小节定义了 PV 中所需的所有标签以匹配此声明。

28.10.2.3. 卷端点

若要将 PV 连接到 Gluster 卷，应当在创建对象之前配置端点。

例 28.15. glusterfs-ep.yaml

```
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster
subsets:
  - addresses:
      - ip: 192.168.122.221
    ports:
      - port: 1
  - addresses:
      - ip: 192.168.122.222
    ports:
      - port: 1
```

28.10.2.4. 部署 PV、PVC 和端点

在本例中，以 **cluster-admin** 特权用户身份运行 **oc** 命令。在生产环境中，集群客户端可能需要定义并创建 PVC。

```
# oc create -f glusterfs-ep.yaml
endpoints "glusterfs-cluster" created
# oc create -f glusterfs-pv.yaml
persistentvolume "gluster-volume" created
# oc create -f glusterfs-pvc.yaml
persistentvolumeclaim "gluster-claim" created
```

最后，确认 PV 和 PVC 是否已成功绑定。

```
# oc get pv,pvc
NAME             CAPACITY   ACCESSMODES     STATUS     CLAIM                  REASON    AGE
gluster-volume   2Gi        RWX             Bound      gfs-trial/gluster-claim          7s
NAME             STATUS     VOLUME          CAPACITY   ACCESSMODES            AGE
gluster-claim    Bound      gluster-volume  2Gi        RWX                    7s
```

注意

PVC 是项目的本地，而 PV 是集群范围的全局资源。开发人员和非管理员用户可能无法查看可用 PV 的所有（或任何）权限。

28.11. 使用存储类进行动态置备

28.11.1. 概述

在这些示例中，我们将执行几个使用 Google Cloud Platform Compute Engine (GCE) 的 StorageClass 配置和动态置备的情况。这些示例假定您对 Kubernetes、GCE 和 Persistent Disks 和 OpenShift Container Platform 进行了一些了解，并 被正确配置为使用 GCE。

- 基本动态置备

- 默认集群动态置备行为

28.11.2. 情况 1：使用两种 *StorageClass* 类型的基本动态置备

StorageClasses 可用于区分和划分存储级别和使用。在本例中，**cluster-admin** 或 **storage-admin** 在 GCE 中设置了两个不同的存储类别。

- **速度较慢** ：为后续数据操作低廉、高效和优化（读和写越）
- **fast**:优化的随机 IOPS 和持续吞吐量（读取和编写的速度）

通过创建这些 *StorageClasses*，**cluster-admin** 或 **storage-admin** 允许用户创建请求特定级别的或 *StorageClass* 服务的声明。

例 28.16. 在对象定义下 StorageClass

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow (1)
provisioner: kubernetes.io/gce-pd (2)
parameters:
  type: pd-standard (3)
  zone: us-east1-d (4)
```

1. *StorageClass* 的名称。
2. 要使用的 provisioner 插件。这是 *StorageClasses* 的必需字段。
3. PD 类型。本示例使用 **pd-standard**，其成本略有线、IOPS 的速度和吞吐量，与带 **pdssd** 的吞吐量相比，传输可持续的 IOPS 和吞吐量。
4. zone 是必需的。

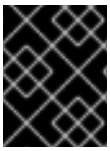
例 28.17. StorageClass 快速对象定义

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  zone: us-east1-d
```

以 **cluster-admin** 或 **storage-admin** 身份，将这两个定义保存为 YAML 文件。例如： **slow-gce.yaml** 和 **fast-gce.yaml**。然后，创建 *StorageClasses*。

```
# oc create -f slow-gce.yaml
storageclass "slow" created
```

```
# oc create -f fast-gce.yaml
storageclass "fast" created

# oc get storageclass
NAME       TYPE
fast       kubernetes.io/gce-pd
slow       kubernetes.io/gce-pd
```



重要

cluster-admin 或 **storage-admin** 用户负责将正确的 *StorageClass* 名称中继到正确的用户、组和项目。

作为常规用户，创建一个新项目：

```
# oc new-project rh-eng
```

创建声明 YAML 定义，将其保存到文件中(**pvc-fast.yaml**)：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-engineering
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 10Gi
 storageClassName: fast
```

使用 **oc create** 命令添加声明：

```
# oc create -f pvc-fast.yaml
persistentvolumeclaim "pvc-engineering" created
```

检查您的声明是否已绑定：

```
# oc get pvc
NAME             STATUS    VOLUME                                     CAPACITY   ACCESSMODES   AGE
pvc-engineering   Bound     pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX           2m
```

重要

由于此声明在 *rh-eng* 项目中创建并绑定，它可以由同一项目中的任何用户共享。

以 **cluster-admin** 或 **storage-admin** 用户身份，查看最新动态置备的持久性卷(PV)。

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM                 REASON    AGE
```

```
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi      RWX           Delete          Bound     rh-
eng/pvc-engineering              5m
```

重要

请注意，所有动态置备的卷 *默认为* RECLAIMPOLICY 被删除。这意味着，当系统中存在声明时，卷才会持续。如果您删除声明，该卷也会被删除，且该卷上的所有数据都会丢失。

最后，检查 GCE 控制台。新磁盘已创建，并可供使用。

```
kubernetes-dynamic-pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  SSD persistent disk  10 GB  us-
east1-d
```

Pod 现在可以引用持久性卷声明并开始使用卷。

28.11.3. 场景 2：如何为集群启用默认 *StorageClass* 行为

在这个示例中，**cluster-admin** 或 **storage-admin** 为所有没有在其声明中隐式指定了 *StorageClass* 的其他用户和项目启用一个*默认*存储类。这对 **cluster-admin** 或 **storage-admin** 有助于提供轻松管理存储卷，而无需在集群中设置或通信专用 *StorageClasses*。

这个示例基于 第 28.11.2 节 “情况 1：使用两种 *StorageClass* 类型的基本动态置备” 构建。**cluster-admin** 或 **storage-admin** 将创建另一个 *StorageClass* 以设计为 *默认的 StorageClass*。

例 28.18. 默认 StorageClass 对象定义

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: generic (1)
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" (2)
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  zone: us-east1-d
```

(1) *StorageClass* 的名称，该名称需要在集群中唯一。

(2) 将这个 *StorageClass* 标记为默认存储类的注解。在这个 API 版本中必须使用 **"true"** quoted。如果没有此注解，OpenShift Container Platform 会考虑它不是 *默认 StorageClass*。

作为 **cluster-admin** 或 **storage-admin** 将定义保存到 YAML 文件(**generic-gce.yaml**)，然后创建 *StorageClasses* ：

```
# oc create -f generic-gce.yaml
storageclass "generic" created

# oc get storageclass
NAME      TYPE
```

```
generic    kubernetes.io/gce-pd
fast       kubernetes.io/gce-pd
slow       kubernetes.io/gce-pd
```

作为常规用户，在不满足任何 *StorageClass* 要求的情况下创建一个新的声明定义，并将它保存到文件 (**generic-pvc.yaml**)。

例 28.19. *默认* 存储声明对象定义

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-engineering2
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 5Gi
```

执行它并检查声明是否绑定：

```
# oc create -f generic-pvc.yaml
persistentvolumeclaim "pvc-engineering2" created
                                                3s
# oc get pvc
NAME              STATUS    VOLUME                                     CAPACITY   ACCESSMODES   AGE
pvc-engineering   Bound     pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX           41m
pvc-engineering2  Bound     pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX           7s
1
```

1 **pvc-engineering2** *默认*绑定到动态置备的卷。

作为 **cluster-admin** 或 **storage-admin**，查看目前定义的持久性卷：

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM                  REASON    AGE
pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX           Delete          Bound     rh-
eng/pvc-engineering2             5m 1
pvc-ba4612ce-8b4d-11e6-9962-42010af00004   5Gi        RWO           Delete          Bound
mytest/gce-dyn-claim1              21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX           Delete          Bound     rh-
eng/pvc-engineering              46m 2
```

1 这个 PV 会绑定到来自 *default StorageClass* 的 *default* 动态卷。

2 这个 PV 使用 *快速 StorageClass* 从 第 28.11.2 节 “情况 1：使用两种 *StorageClass* 类型的基本动态置备” 绑定到第一个 PVC。

使用 GCE （未动态置备）创建手动置备的磁盘。然后创建一个 持久性卷，以连接到新的 GCE 磁盘(**pv-manual-gce.yaml**)。

例 28.20. 手动 PV 对象片段

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-manual-gce
spec:
 capacity:
   storage: 35Gi
 accessModes:
   - ReadWriteMany
 gcePersistentDisk:
   readOnly: false
   pdName: the-newly-created-gce-PD
   fsType: ext4
```

执行对象定义文件：

```
# oc create -f pv-manual-gce.yaml
```

现在再次查看 PV。请注意，**pv-manual-gce** 卷 *可用*。

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM                  REASON    AGE
pv-manual-gce                                35Gi       RWX           Retain          Available
4s
pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX           Delete          Bound       rh-
eng/pvc-engineering2             12m
pvc-ba4612ce-8b4d-11e6-9962-42010af00004   5Gi        RWO           Delete          Bound
mytest/gce-dyn-claim1            21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX           Delete          Bound       rh-
eng/pvc-engineering              53m
```

现在，创建与 **generic-pvc.yaml** PVC 定义相同的另一个声明，但更改名称且不设置存储类名称。

例 28.21. 声明对象定义

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-engineering3
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 15Gi
```

因为在这个实例中启用了 *默认 StorageClass*，所以手动创建的 PV 不满足声明请求。用户接收新的动态置备的持久性卷。

```
# oc get pvc
NAME              STATUS    VOLUME                                     CAPACITY   ACCESSMODES   AGE
pvc-engineering   Bound     pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX           1h
pvc-engineering2  Bound     pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX           19m
pvc-engineering3  Bound     pvc-6fa8e73b-8c00-11e6-9962-42010af00004   15Gi       RWX           6s
```

重要

由于在这个系统上启用了 *默认 StorageClass*，因此手动创建的持久性卷以通过上述声明绑定且没有绑定新的动态置备的卷，所以需要在*默认 StorageClass* 中创建 PV。

由于在这个系统中启用了 *默认 StorageClass*，您需要在手动创建的持久性卷的*默认 StorageClass* 中创建 PV，以便绑定到上述声明，且没有绑定到该声明的新动态置备卷。

要解决这个问题，**cluster-admin** 或 **storage-admin** 用户只需要创建另一个 GCE 磁盘或删除第一个手动 PV，并使用分配 *StorageClass* 名称(**pv-manual-gce2.yaml**)的 PV 对象定义：

例 28.22. 使用 *default* StorageClass 名称的手动 PV Spec

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-manual-gce2
spec:
 capacity:
   storage: 35Gi
 accessModes:
   - ReadWriteMany
 gcePersistentDisk:
   readOnly: false
   pdName: the-newly-created-gce-PD
   fsType: ext4
 storageClassName: generic (1)
```

(1) 之前创建的 *通用 StorageClass* 的名称。

执行对象定义文件：

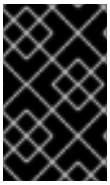
```
# oc create -f pv-manual-gce2.yaml
```

列出 PV：

```
# oc get pv
NAME                        CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM              REASON   AGE
pv-manual-gce               35Gi       RWX           Retain          Available
4s (1)
pv-manual-gce2              35Gi       RWX           Retain          Bound       rh-eng/pvc-
```

```
engineering3           4s  (2)
pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi       RWX           Delete          Bound       rh-
eng/pvc-engineering2           12m
pvc-ba4612ce-8b4d-11e6-9962-42010af00004   5Gi        RWO           Delete          Bound
mytest/gce-dyn-claim1              21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX           Delete          Bound       rh-
eng/pvc-engineering            53m
```

1. 原始的手动 PV 仍未绑定且可用。这是因为在 *默认 StorageClass* 中没有创建它。
2. 第二个 PVC（除名称以外）绑定到 Available 手动创建 PV **pv-manual-gce2**。



重要

请注意，所有动态部署的卷都默认为 *RECLAIMPOLICY Delete*。当 PVC 动态绑定到 PV 后，GCE 卷会被删除，并会丢失所有数据。但是，手动创建的 PV 的默认 *RECLAIMPOLICY* 为 *Retain*。

28.12. 使用现有旧存储的存储类

28.12.1. 概述

在本例中，旧数据卷存在，**cluster-admin** 或 **storage-admin** 需要它可用于特定项目。使用 *StorageClasses* 会从声明中减少对这个卷的访问权限的其他用户和项目的可能性，因为声明必须具有与 *StorageClass* 名称具有完全匹配的值。这个示例还禁用动态置备。这个示例假设：

- 熟悉 OpenShift Container Platform、GCE 和 Persistent Disks
- OpenShift Container Platform 正确配置为使用 GCE。

28.12.1.1. 情况 1：将 StorageClass 链接到带有旧数据的现有持久性卷

作为 **cluster-admin** 或 **storage-admin**，为历史财务数据定义并创建 *StorageClass*。

例 28.23. StorageClass finance-history 对象定义

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: finance-history  (1)
provisioner: no-provisioning  (2)
parameters:  (3)
```

1. StorageClass 的名称。
2. 这是必填字段，但由于没有动态置备，因此只要不是实际的置备程序插件类型，就必须在此处放置一个值。
3. 参数只需留空，因为它们仅用于动态置备程序。

将定义保存到 YAML 文件(**finance-history-storageclass.yaml**)并创建 *StorageClass*。

```
# oc create -f finance-history-storageclass.yaml
storageclass "finance-history" created


# oc get storageclass
NAME              TYPE
finance-history   no-provisioning
```

重要

cluster-admin 或 **storage-admin** 用户负责将正确的 *StorageClass* 名称中继到正确的用户、组和项目。

StorageClass 存在。**cluster-admin** 或 **storage-admin** 可以创建用于 *StorageClass* 的持久性卷 (PV)。使用 GCE （未动态置备）创建一个手动置备的磁盘，以及一个连接到新的 GCE 磁盘(**gce-pv.yaml**)的持久性卷。

例 28.24. 财务历史 PV 对象

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-finance-history
spec:
 capacity:
   storage: 35Gi
 accessModes:
   - ReadWriteMany
 gcePersistentDisk:
   readOnly: false
   pdName: the-existing-PD-volume-name-that-contains-the-valuable-data (1)
   fsType: ext4
 storageClassName: finance-history (2)
```

(2) *StorageClass* 名称，必须完全匹配。

(1) 已存在并包含旧数据的 GCE 磁盘名称。

作为 **cluster-admin** 或 **storage-admin**，创建并查看 PV。

```
# oc create -f gce-pv.yaml
persistentvolume "pv-finance-history" created

# oc get pv
NAME              CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS      CLAIM
REASON    AGE
pv-finance-history   35Gi       RWX           Retain          Available                                   2d
```

请注意，您有一个 **pv-finance-history** 可用并可供使用。

以用户身份，创建一个持久性卷声明(PVC)作为 YAML 文件并指定正确的 *StorageClass* 名称：

例 28.25. 对 finance-history 对象定义的声明

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-finance-history
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 20Gi
 storageClassName: finance-history (1)
```

1. 在创建了与名称匹配的 *StorageClass* 名称，必须完全匹配或声明不绑定，直到其被删除或另一个 *StorageClass* 被创建。

创建并查看 PVC 和 PV 以查看它是否已绑定。

```
# oc create -f pvc-finance-history.yaml
persistentvolumeclaim "pvc-finance-history" created

# oc get pvc
NAME                 STATUS    VOLUME              CAPACITY   ACCESSMODES   AGE
pvc-finance-history   Bound     pv-finance-history   35Gi       RWX           9m


# oc get pv  (cluster/storage-admin)
NAME                CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS      CLAIM
REASON    AGE
pv-finance-history   35Gi       RWX           Retain          Bound       default/pvc-finance-history
5m
```

重要

您可以将同一集群中的 *StorageClasses* 用于旧的数据（没有动态置备）以及 动态置备。

28.13. 为集成 CONTAINER IMAGE REGISTRY 配置 AZURE BLOB 存储

28.13.1. 概述

本节介绍了如何为 OpenShift 集成的容器镜像 registry 配置 Microsoft Azure Blob Storage。

28.13.2. 开始前

- 使用 Microsoft Azure Portal、Microsoft Azure CLI 或 Microsoft Azure Storage Explorer 创建存储容器。记录下 ***storage account name***, ***storage account key*** 和 ***container name***。
- 如果集成容器镜像 registry 没有被部署，则部署它。

28.13.3. 覆盖 Registry 配置

要创建新 registry pod 并自动替换旧的 pod：

1. 创建名为 ***registryconfig.yaml*** 的新 registry 配置文件并添加以下信息：

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
  azure: (1)
    accountname: azureblobacc
    accountkey:  azureblobacckey
    container: azureblobname
    realm: core.windows.net (2)
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
      options:
        acceptschema2: false
        pullthrough: true
        enforcequota: false
        projectcachettl: 1m
        blobrepositorycachettl: 10m
  storage:
    - name: openshift
```

(1) 将 **accountname**、**acountkey** 和 **container** 的值替换为 ***存储帐户名称***、***存储帐户密钥***，以及 ***存储容器名称***。

(2) 如果使用 Azure 区域云，请将 设置为所需的域。例如，德国区域的云为 **core.cloudapi.de**。

2. 创建新 registry 配置：

```
$ oc create secret generic registry-config --from-file=config.yaml=registryconfig.yaml
```

3. 添加 secret：

```
$ oc set volume dc/docker-registry --add --type=secret \
    --secret-name=registry-config -m /etc/docker/registry/
```

4. 设置 **REGISTRY_CONFIGURATION_PATH** 环境变量：

```
$ oc set env dc/docker-registry \
    REGISTRY_CONFIGURATION_PATH=/etc/docker/registry/config.yaml
```

5. 如果您已经创建了 registry 配置：

   a. 删除 secret：

   ```
   $ oc delete secret registry-config
   ```

   b. 创建新 registry 配置：

   ```
   $ oc create secret generic registry-config --from-file=config.yaml=registryconfig.yaml
   ```

   c. 启动新的推出部署来更新配置：

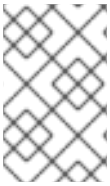
   ```
   $ oc rollout latest docker-registry
   ```

第 29 章 配置临时存储

29.1. 概述

OpenShift Container Platform 可以配置为管理 pod 和容器工作数据的临时存储。虽然容器可以使用可写层、日志目录和 EmptyDir 卷，但这种存储会受到很多限制，如此处所述。

临时存储管理允许管理员限制各个 pod 和容器消耗的资源，而 pod 和容器则指定它们使用的请求和限制。这是一个技术预览，默认是禁用的。



注意

此技术预览不会更改任何在 OpenShift Container Platform 中提供本地存储的机制，即现有的机制、根目录或运行时目录，仍然适用。此技术预览仅提供管理此资源的使用的机制。

29.2. 启用临时存储

启用临时存储：

1. 在所有 master 中编辑或创建 master 配置文件（默认为 */etc/origin/master/master-config.yaml*），在 **apiServerArguments** 和 **controllerArguments** 项中添加 **LocalStorageCapacityIsolation=true**：

```
apiServerArguments:
  feature-gates:
  - LocalStorageCapacityIsolation=true
...

 controllerArguments:
  feature-gates:
  - LocalStorageCapacityIsolation=true
...
```

2. 编辑所有节点的 ConfigMap，在命令行中启用 LocalStorageCapacityIsolation。您可以识别需要编辑的 ConfigMap，如下所示：

```
$ oc get cm -n openshift-node
NAME                 DATA      AGE
node-config-compute  1         52m
node-config-infra    1         52m
node-config-master   1         52m
```

对于每个映射，**node-config-compute**、**node-config-infra** 和 **node-config-master** 都需要添加功能门：

```
oc edit cm node-config-master -n openshift-node
```

如果已存在 **feature-gates:** 声明，请在功能门列表中添加以下文本：

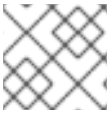
```
,LocalStorageCapacityIsolation=true
```

如果没有 **feature-gates:** 声明，请添加以下部分：

```
feature-gates:
- LocalStorageCapacityIsolation=true
```

3. 对 **node-config-compute**、**node-config-infra** 以及任何其他 ConfigMap 重复。

4. 重启 OpenShift Container Platform 并删除运行 apiserver 的容器。



注意

省略这些步骤可能会导致没有启用临时存储管理。

第 30 章 使用 HTTP PROXIES

30.1. 概述

生产环境可能会拒绝直接访问互联网，而是提供 HTTP 或 HTTPS 代理。将 OpenShift Container Platform 配置为使用这些代理可以像在配置或 JSON 文件中设置标准环境变量一样简单。这可在集群安装过程中完成，也可以在安装后配置。

在集群的每个主机上代理配置必须相同。因此，在设置代理或修改时，必须将每个 OpenShift Container Platform 主机上的文件更新为相同的值。然后，您必须在集群中的每个主机上重启 OpenShift Container Platform 服务。

NO_PROXY、**HTTP_PROXY** 和 **HTTPS_PROXY** 环境变量包括在每个主机的 */etc/origin/master/master.env* 和 */etc/sysconfig/atomic-openshift-node* 文件中。

30.2. 配置 NO_PROXY

NO_PROXY 环境变量列出了 OpenShift Container Platform 组件以及由 OpenShift Container Platform 管理的所有 IP 地址。

在 OpenShift 服务接受 CIDR 中，**NO_PROXY** 接受以 CIDR 格式的逗号分隔主机、IP 地址或 IP 范围列表：

对于 master 主机

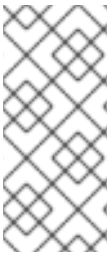
- 节点主机名
- Master IP 或主机名
- etcd 主机的 IP 地址

对于节点主机

- Master IP 或主机名

对于 Docker 服务

- registry 服务 IP 和主机名
- registry 服务 URL **docker-registry.default.svc.cluster.local**
- registry 路由主机名（如果已创建）



注意

在使用 Docker 时，Docker 接受由逗号分隔的主机、域扩展或 IP 地址列表，但不接受 CIDR 格式的 IP 范围，而只被 OpenShift 服务接受。'no_proxy' 变量应包含以逗号分隔的域扩展列表，它们*不应该*应用于代理。

例如，如果将 **no_proxy** 设为 **.school.edu**，则代理将不会从特定中部检索文档。

NO_PROXY 还包括 *master-config.yaml* 文件中找到的 SDN 网络和服务 IP 地址。

/etc/origin/master/master-config.yaml

```
networkConfig:
  clusterNetworks:
  - cidr: 10.1.0.0/16
    hostSubnetLength: 9
  serviceNetworkCIDR: 172.30.0.0/16
```

OpenShift Container Platform 不接受 * 作为附加到域后缀的通配符。例如，接受以下内容：

```
NO_PROXY=.example.com
```

但是，以下内容不是：

```
NO_PROXY=*.example.com
```

唯一通配符 **NO_PROXY** 接受是一个 * 字符，匹配所有主机并有效禁用代理。

此列表中的每个名称都与一个域匹配，该域包含主机名为后缀，或者主机名本身。

注意

在扩展节点时，使用域名而不是主机名列表。

例如，**example.com** 将匹配 **example.com**、**example.com:80** 和 **www.example.com**。

30.3. 为代理配置主机

1. 编辑 OpenShift Container Platform 控制文件中的代理环境变量。确保集群中的所有文件都正确。

```
HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>/
HTTPS_PROXY=https://<user>:<password>@<ip_addr>:<port>/
NO_PROXY=master.hostname.example.com,10.1.0.0/16,172.30.0.0/16 (1)
```

(1) 支持主机名和 CIDR。默认情况下，必须包括 SDN 网络和服务 IP 范围 **10.1.0.0/16,172.30.0.0/16**。

2. 重启 master 或节点主机：

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

30.4. 使用 ANSIBLE 为代理配置主机

在集群安装过程中，可以使用 **openshift_no_proxy**、**openshift_http_proxy** 和 **openshift_https_proxy** 参数配置 **NO_PROXY**、**HTTP_PROXY** 和 **HTTPS_PROXY** 环境变量，这些变量可在清单文件中进行配置。

使用 Ansible 的代理配置示例

```
# Global Proxy Configuration
# These options configure HTTP_PROXY, HTTPS_PROXY, and NOPROXY environment
# variables for docker and master services.
openshift_http_proxy=http://<user>:<password>@<ip_addr>:<port>
openshift_https_proxy=https://<user>:<password>@<ip_addr>:<port>
openshift_no_proxy='.hosts.example.com,some-host.com'
#
# Most environments do not require a proxy between OpenShift masters, nodes, and
# etcd hosts. So automatically add those host names to the openshift_no_proxy list.
# If all of your hosts share a common domain you may wish to disable this and
# specify that domain above.
# openshift_generate_no_proxy_hosts=True
```

注意

您可以使用 Ansible 参数 为构建 配置额外的 代理设置。例如：

openshift_builddefaults_git_http_proxy 和 **openshift_builddefaults_git_https_proxy** 参数 允许您使用代理进行 Git 克隆

openshift_builddefaults_http_proxy 和 **openshift_builddefaults_https_proxy** 参数可以将环境变量提供给 Docker 构建策略 和 Custom 构建策略 进程。

30.5. 代理 DOCKER PULL

OpenShift Container Platform 节点主机需要执行对 Docker registry 进行推送和拉取操作。如果您有一个不需要代理才能访问的 registry，请包括 **NO_PROXY** 参数：

- registry 的主机名，
- registry 服务的 IP 地址以及
- 服务名称。

此会将该 registry 列入黑名单，将外部 HTTP 代理保留为唯一选项。

1. 运行以下命令，检索 registry 服务的 IP 地址 **docker_registy_ip** ：

```
$ oc describe svc/docker-registry -n default

Name:   docker-registry
Namespace:  default
Labels:   docker-registry=default
Selector:  docker-registry=default
Type:   ClusterIP
IP:   172.30.163.183 (1)
Port:   5000-tcp 5000/TCP
Endpoints:  10.1.0.40:5000
Session Affinity: ClientIP
No events.
```

(1) Registry 服务 IP。

2. 编辑 */etc/sysconfig/docker* 文件并添加 shell 格式的 **NO_PROXY** 变量，将 &lt ;**docker_registry_ip** > 替换为上一步中的 IP 地址。

```
HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>/
HTTPS_PROXY=https://<user>:<password>@<ip_addr>:<port>/
NO_PROXY=master.hostname.example.com,<docker_registry_ip>,docker-
registry.default.svc.cluster.local
```

3. 重启 Docker 服务：

```
# systemctl restart docker
```

30.6. 使用 MAVEN 替换代理

将 Maven 与代理一起使用需要使用 **HTTP_PROXY_NONPROXYHOSTS** 变量。

如需有关 为 Red Hat JBoss Enterprise Application Platform 配置 OpenShift Container Platform 环境的信息，请参阅 Red Hat JBoss Enterprise Application Platform for OpenShift 文档，包括在代理后面设置 Maven 的步骤。

30.7. 为代理配置 S2I 构建

S2I 构建从各个位置获取依赖项。您可以使用 *.s2i/environment* 文件来指定 简单的 shell 变量，当看到构建镜像时，OpenShift Container Platform 会相应地做出反应。

以下是支持的代理环境变量，带有示例值：

```
HTTP_PROXY=http://USERNAME:PASSWORD@10.0.1.1:8080/
HTTPS_PROXY=https://USERNAME:PASSWORD@10.0.0.1:8080/
NO_PROXY=master.hostname.example.com
```

30.8. 为代理配置默认模板

默认情况下，OpenShift Container Platform 中提供的示例 模板 不包括 HTTP 代理的设置。对于基于这些模板的现有应用程序，修改应用程序构建配置 **的源** 部分并添加代理设置：

```
...
source:
  type: Git
  git:
    uri: https://github.com/openshift/ruby-hello-world
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
    noProxy: somedomain.com, otherdomain.com
...
```

这类似于使用代理进行 Git 克隆的过程。

30.9. 在 POD 中设置代理环境变量

您可以在部署配置中的 **templates.spec.containers** 部分中设置 **NO_PROXY**、**HTTP_PROXY** 和 **HTTPS_PROXY** 环境变量，以传递代理连接信息。在运行时配置 Pod 代理可以相同操作：

```
...
containers:
- env:
  - name: "HTTP_PROXY"
    value: "http://<user>:<password>@<ip_addr>:<port>"
...
```

您还可以使用 **oc set env** 命令使用新的环境变量更新现有部署配置：

```
$ oc set env dc/frontend HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>
```

如果您在 OpenShift Container Platform 实例中设置了 ConfigChange 触发器，则会自动进行更改。否则，请手动重新部署您的应用程序以使更改生效。

30.10. GIT 存储库访问

如果 Git 存储库需要使用代理才能访问，您可以在 **BuildConfig** 的 **source** 部分中定义要使用的代理。您可以配置要使用的 HTTP 和 HTTPS 代理。两个字段都是可选的。也可以通过 **NoProxy** 字段指定不应执行代理的域。

注意

源 URI 必须使用 HTTP 或 HTTPS 协议才可以正常工作。

```
source:
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
    noProxy: somedomain.com, otherdomain.com
```

第 31 章 配置全局构建默认值和覆盖

31.1. 概述

开发人员可以在其项目内的特定构建配置中定义设置，例如 为 Git 克隆配置代理。除了要求开发人员定义每个构建配置中的某些设置外，管理员可以使用准入插件来配置全局构建默认值，并覆盖在任何构建中自动使用这些设置。

这些插件中的设置仅在构建过程中使用，但不在构建配置或构建本身中设置。通过插件配置设置后，管理员可以随时更改全局配置，并且从现有构建配置或构建中重新运行的任何构建都会分配新设置。

- 借助 **BuildDefaults** 准入插件，管理员可以为 Git HTTP 和 HTTPS 代理等设置设置全局默认值，以及默认的环境变量。这些默认值不会覆盖为特定构建配置的值。但是，如果构建定义中没有这些值，则它们会设置为默认值。

- **BuildOverrides** 准入插件允许管理员覆盖构建中的设置，而不考虑构建中存储的值。
  插件目前支持覆盖构建策略上的 **forcePull** 标志，以便在构建期间从 registry 中强制刷新本地镜像。这意味着，每次构建启动时都会对镜像执行访问检查，确保用户只能使用允许拉取的镜像来构建。强制刷新为您的构建提供多租户。但是，您无法依赖构建节点上存储的镜像的本地缓存，您必须始终能够访问 registry。

  该插件也可以配置为将一组镜像标签应用到每个构建的镜像。

  有关配置 **BuildOverrides** 准入插件和您可以覆盖的值的详情，请参阅 手动设置全局 Build Overrides。

默认节点选择器和 **BuildDefaults** 或 **BuildOverrides** 准入插件可以正常工作，如下所示：

- master 配置文件的 **projectConfig.defaultNodeSelector** 字段中定义的默认项目节点选择器应用到在没有指定 **nodeSelector** 值的情况下在所有项目中创建的 pod。这些设置应用于在未设置 **BuildDefaults** 或 **BuildOverrides** nodeselector 的集群上使用 **nodeSelector="null"** 的构建。

- 只有在构建配置中设置了 nodeSelector="null" 参数，才会应用集群范围的默认选择器 **admissionConfig.pluginConfig.BuildDefaults.configuration.nodeSelector**。才会在构建配置中设置 **nodeSelector="null "** 参数。

- 使用默认项目或集群范围节点选择器时，默认设置会作为构建节点选择器的 AND 添加到构建节点选择器中，由 **BuildDefaults** 或 **BuildOverrides** 准入插件设置。这些设置表示构建将仅调度到满足 **BuildOverrides** 节点选择器和项目默认节点选择器的节点。

注意

您可以使用 RunOnceDuration 插件 来定义有关构建 pod 可运行的硬限制。

31.2. 设置全局构建默认值

您可以通过两种方式设置全局构建默认值：

- 使用 Ansible
- 通过修改 *master-config.yaml* 文件手动进行

31.2.1. 使用 Ansible 配置全局构建默认值

在集群安装过程中，可以使用以下参数来配置 **BuildDefaults** 插件，该插件可在清单文件中配置：

- **openshift_builddefaults_http_proxy**
- **openshift_builddefaults_https_proxy**
- **openshift_builddefaults_no_proxy**
- **openshift_builddefaults_git_http_proxy**
- **openshift_builddefaults_git_https_proxy**
- **openshift_builddefaults_git_no_proxy**
- **openshift_builddefaults_image_labels**
- **openshift_builddefaults_nodeselectors**
- **openshift_builddefaults_annotations**
- **openshift_builddefaults_resources_requests_cpu**
- **openshift_builddefaults_resources_requests_memory**
- **openshift_builddefaults_resources_limits_cpu**
- **openshift_builddefaults_resources_limits_memory**

例 31.1. 使用 Ansible 构建默认配置示例

```
# These options configure the BuildDefaults admission controller which injects
# configuration into Builds. Proxy related values will default to the global proxy
# config values. You only need to set these if they differ from the global proxy settings.
openshift_builddefaults_http_proxy=http://USER:PASSWORD@HOST:PORT
openshift_builddefaults_https_proxy=https://USER:PASSWORD@HOST:PORT
openshift_builddefaults_no_proxy=mycorp.com
openshift_builddefaults_git_http_proxy=http://USER:PASSWORD@HOST:PORT
openshift_builddefaults_git_https_proxy=https://USER:PASSWORD@HOST:PORT
openshift_builddefaults_git_no_proxy=mycorp.com
openshift_builddefaults_image_labels=[{'name':'imagelabelname1','value':'imagelabelvalue1'}]
openshift_builddefaults_nodeselectors={'nodelabel1':'nodelabelvalue1'}
openshift_builddefaults_annotations={'annotationkey1':'annotationvalue1'}
openshift_builddefaults_resources_requests_cpu=100m
openshift_builddefaults_resources_requests_memory=256Mi
openshift_builddefaults_resources_limits_cpu=1000m
openshift_builddefaults_resources_limits_memory=512Mi

# Or you may optionally define your own build defaults configuration serialized as json
#openshift_builddefaults_json='{"BuildDefaults":{"configuration":{"apiVersion":"v1","env":
[{"name":"HTTP_PROXY","value":"http://proxy.example.com.redhat.com:3128"},
{"name":"NO_PROXY","value":"ose3-
master.example.com"}],"gitHTTPProxy":"http://proxy.example.com:3128","gitNoProxy":"ose3-
master.example.com","kind":"BuildDefaultsConfig"}}}'
```

31.2.2. 手动设置全局构建默认值

配置 **BuildDefaults** 插件：

1. 在 master 节点上的 */etc/origin/master/master-config.yaml* 文件中添加一个配置：

```
admissionConfig:
  pluginConfig:
    BuildDefaults:
      configuration:
        apiVersion: v1
        kind: BuildDefaultsConfig
        gitHTTPProxy: http://my.proxy:8080 (1)
        gitHTTPSProxy: https://my.proxy:8443 (2)
        gitNoProxy: somedomain.com, otherdomain.com (3)
        env:
        - name: HTTP_PROXY (4)
          value: http://my.proxy:8080
        - name: HTTPS_PROXY (5)
          value: https://my.proxy:8443
        - name: BUILD_LOGLEVEL (6)
          value: 4
        - name: CUSTOM_VAR (7)
          value: custom_value
        imageLabels:
        - name: url (8)
          value: https://containers.example.org
        - name: vendor
          value: ExampleCorp Ltd.
        nodeSelector: (9)
          key1: value1
          key2: value2
        annotations: (10)
          key1: value1
          key2: value2
        resources: (11)
          requests:
            cpu: "100m"
            memory: "256Mi"
          limits:
            cpu: "100m"
            memory: "256Mi"
```

(1) 设置在从 Git 存储库克隆源代码时要使用的 HTTP 代理。

(2) 设置 HTTPS 代理，以在从 Git 存储库克隆源代码时使用。

(3) 设置不使用代理的域列表。

(4) 默认环境变量，用于设置在构建期间使用的 HTTP 代理。这可用于在 assemble 和构建阶段下载依赖项。

(5) 默认环境变量，用于设置在构建期间使用的 HTTP 代理。这可用于在 assemble 和构建阶段下载依赖项。

6. 在构建期间设置构建日志级别的默认环境变量。
7. 将添加到每个构建的额外默认环境变量。
8. 要应用到每个构建的镜像的标签。用户可以在 **BuildConfig** 中覆盖它们。
9. 构建 pod 仅在带有 **key1=value2** 和 **key2=value2** 标签的节点上运行。用户可以为其构建定义一组不同的 **nodeSelectors**，在这种情况下，这些值将被忽略。
10. 构建 pod 将将这些注解添加到其中。
11. 如果 **BuildConfig** 没有定义相关资源，请将默认资源设置为构建 pod。

2. 重启 master 服务以使更改生效：

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

31.3. 设置全局构建覆盖

您可以通过两种方式设置全局构建覆盖：

- 使用 Ansible
- 通过修改 *master-config.yaml* 文件手动进行

31.3.1. 使用 Ansible 配置全局构建覆盖

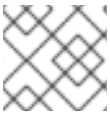
在集群安装过程中，**BuildOverrides** 插件可以使用以下参数来配置，该插件可在清单文件中配置：

- **openshift_buildoverrides_force_pull**
- **openshift_buildoverrides_image_labels**
- **openshift_buildoverrides_nodeselectors**
- **openshift_buildoverrides_annotations**
- **openshift_buildoverrides_tolerations**

例 31.2. 使用 Ansible 构建覆盖配置示例

```
# These options configure the BuildOverrides admission controller which injects
# configuration into Builds.
openshift_buildoverrides_force_pull=true
openshift_buildoverrides_image_labels=[{'name':'imagelabelname1','value':'imagelabelvalue1'}]
openshift_buildoverrides_nodeselectors={'nodelabel1':'nodelabelvalue1'}
openshift_buildoverrides_annotations={'annotationkey1':'annotationvalue1'}
openshift_buildoverrides_tolerations=
[{'key':'mykey1','value':'myvalue1','effect':'NoSchedule','operator':'Equal'}]

# Or you may optionally define your own build overrides configuration serialized as json
```

```
#openshift_buildoverrides_json='{"BuildOverrides":{"configuration":
{"apiVersion":"v1","kind":"BuildOverridesConfig","forcePull":"true","tolerations":
[{'key':'mykey1','value':'myvalue1','effect':'NoSchedule','operator':'Equal'}]}}}'
```



注意

您必须使用 **BuildOverrides** 节点选择器来使用 **BuildOverrides** 插件覆盖容限。

31.3.2. 手动设置全局构建覆盖

配置 **BuildOverrides** 插件：

1. 在 master 上的 */etc/origin/master/master-config.yaml* 文件中为它添加配置：

```
admissionConfig:
  pluginConfig:
    BuildOverrides:
      configuration:
        apiVersion: v1
        kind: BuildOverridesConfig
        forcePull: true (1)
        imageLabels:
        - name: distribution-scope (2)
          value: private
        nodeSelector: (3)
          key1: value1
          key2: value2
        annotations: (4)
          key1: value1
          key2: value2
        tolerations: (5)
        - key: mykey1
          value: myvalue1
          effect: NoSchedule
          operator: Equal
        - key: mykey2
          value: myvalue2
          effect: NoExecute
          operator: Equal
```

(1) 强制所有构建在启动构建前，拉取其构建器镜像和任何源镜像。

(2) 要应用到每个构建的镜像的额外标签。此处定义的标签优先于 **BuildConfig** 中定义的标签。

(3) 构建 pod 仅在带有 **key1=value2** 和 **key2=value2** 标签的节点上运行。用户可以定义额外的键/值标签，以进一步限制构建运行所在的节点集合，但 **该节点** 必须至少具有这些标签。

(4) 构建 pod 将将这些注解添加到其中。

(5) 构建 pod 将在这里列出的任何现有容限覆盖。

注意

您必须使用 **BuildOverrides** 节点选择器来使用 **BuildOverrides** 插件覆盖容限。

2. 重启 master 服务以使更改生效：

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

第 32 章 配置管道执行

32.1. 概述

用户第一次使用 Pipeline 构建策略创建构建配置，OpenShift Container Platform 会在 **openshift** 命名空间中查找名为 **jenkins-ephemeral** 的模板，并在用户项目中实例化它。OpenShift Container Platform 附带的 **jenkins-ephemeral** 模板会在实例化时创建：

- 使用官方 OpenShift Container Platform Jenkins 镜像的 Jenkins 部署配置
- 用于访问 Jenkins 部署的服务和路由
- 新的 Jenkins 服务帐户
- RoleBindings 为服务帐户授予项目编辑访问权限

集群管理员可以通过修改内置模板的内容或编辑集群配置来将集群定向到不同的模板位置来控制创建的内容。

修改默认模板的内容：

```
$ oc edit template jenkins-ephemeral -n openshift
```

要使用不同的模板，如 Jenkins 使用持久性存储的 **jenkins-persistent** 模板，请将以下内容添加到 master 配置文件中：

```
jenkinsPipelineConfig:
  autoProvisionEnabled: true (1)
  templateNamespace: openshift (2)
  templateName: jenkins-persistent (3)
  serviceName: jenkins-persistent-svc (4)
  parameters: (5)
    key1: value1
    key2: value2
```

1. 若未指定，则默认为 **true**。如果为 **false**，则没有模板实例化。
2. 包含要实例化的模板的命名空间。
3. 要实例化的模板的名称。
4. 实例化时由模板创建的服务的名称。
5. 实例化过程中要传递给模板的可选值。

创建 Pipeline 构建配置时，OpenShift Container Platform 会查找匹配 **serviceName** 的服务。这意味着必须选择 **serviceName**，以便它在项目中是唯一的。如果没有找到 Service，OpenShift Container Platform 会实例化 **jenkinsPipelineConfig** 模板。如果这不是必须的（例如，要使用 OpenShift Container Platform 外部的 Jenkins 服务器），您可以执行一些操作，具体取决于您是谁。

- 如果您是集群管理员，只需将 **autoProvisionEnabled** 设置为 **false**。这将在集群中禁用自动置备。

- 如果您是未激活的用户，则必须创建一个服务供 OpenShift Container Platform 使用。服务名称必须与 **jenkinsPipelineConfig** 中的 **serviceName** 的集群配置值匹配。默认值为 **jenkins**。如果您要禁用自动置备，因为您要在项目之外运行 Jenkins 服务器，建议您将这个新服务指向现有的 Jenkins 服务器。请参阅：集成外部服务

后一选项也可用于仅在选择的项目中禁用自动置备。

32.2. OPENSHIFT JENKINS 客户端插件

OpenShift Jenkins 客户端插件是一种 Jenkins 插件，旨在提供易读、简洁、全面且流畅的 Jenkins Pipeline 语法，以便与 OpenShift API 服务器进行丰富的交互。该插件利用了 OpenShift 命令行工具 (**oc**)，此工具必须在执行脚本的节点上可用。

有关安装和配置插件的更多信息，请使用下面参考官方文档的链接。

- 安装
- 配置 OpenShift 集群
- 设置凭证
- 设置 Jenkins 节点

注意

您是需要了解使用此插件信息的开发人员？如果是，请参阅 OpenShift Pipeline 概述。

32.3. OPENSHIFT JENKINS SYNC PLUGIN

此 Jenkins 插件使 OpenShift BuildConfig 和 Build 对象与 Jenkins 任务和构建保持同步。

OpenShift jenkins 同步插件提供以下：

- Jenkins 中动态创建任务/运行。
- 从 ImageStreams、ImageStreamTag 或 ConfigMap 动态创建 slave Pod 模板。
- 注入环境变量。
- OpenShift web 控制台中的管道视觉化。
- 与 Jenkins git 插件集成，后者将 OpenShift 构建中的提交信息传递给 Jenkins git 插件。

有关这个插件的更多信息，请参阅：

- OpenShift Jenkins Sync Plug-in
- OpenShift Container Platform 同步插件

第 33 章 配置路由超时

安装 OpenShift Container Platform 并部署路由器后，您可以在需要低超时的服务（满足服务级别可用性 (SLA)目的或高超时）的情况下，为现有路由配置默认超时。

使用 **oc annotate** 命令，为路由添加超时：

```
# oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit>
```

例如，将名为 **myroute** 的路由设置为 2 秒的超时：

```
# oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

支持的时间单位是微秒 (us)、毫秒 (ms)、秒钟 (s)、分钟 (m)、小时 (h)、或天 (d)。

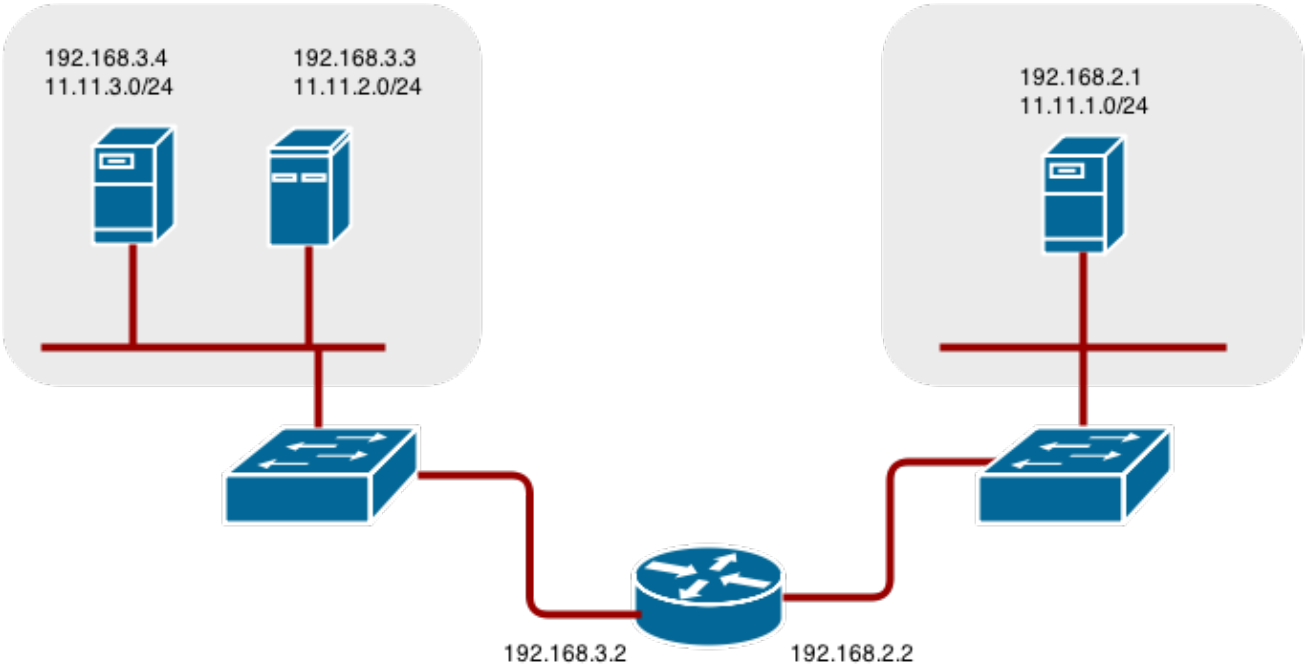
第 34 章 配置原生容器路由

34.1. 网络概述

下面描述了常规网络设置：

- 11.11.0.0/16 是容器网络。
- 11.11.x.0/24 子网为每个节点保留，并分配给 Docker Linux 网桥。
- 每个节点都有一个路由器，用于到达 11.11.0.0/16 范围内的任何内容，但本地子网除外。
- 路由器具有每个节点的路由，以便它能够定向到正确的节点。
- 在添加新节点时，现有节点不需要任何更改，除非修改了网络拓扑。
- 每个节点中都启用了 IP 转发。

下图显示了本主题中描述的容器网络设置。它使用两个网络接口卡的一个 Linux 节点作为路由器、两个交换机，以及连接到这些交换机的三个节点。



34.2. 配置原生容器路由

您可以使用现有交换机和路由器以及 Linux 中的内核网络堆栈来设置容器网络。

作为网络管理员，您必须修改或创建一个脚本来修改脚本，当新节点添加到集群中时，路由器或路由器。

您可以调整这个过程，以用于任何类型的路由器。

34.3. 为容器联网设置节点

1. 为节点上的 Linux 网桥分配一个未使用 11.11.x.0/24 子网 IP 地址：

```
# brctl addbr lbr0
# ip addr add 11.11.1.1/24 dev lbr0
# ip link set dev lbr0 up
```

2. 修改 Docker 启动脚本，以使用新网桥。默认情况下，启动脚本是 **/etc/sysconfig/docker** 文件：

```
# docker -d -b lbr0 --other-options
```

3. 向路由器添加 11.11.0.0/16 网络的路由：

```
# ip route add 11.11.0.0/16 via 192.168.2.2 dev p3p1
```

4. 在节点上启用 IP 转发：

```
# sysctl -w net.ipv4.ip_forward=1
```

34.4. 为容器网络设置路由器

以下流程假设有多个 NIC 的 Linux 方框被用作路由器。根据需要修改步骤，以使用特定路由器的语法：

1. 在路由器上启用 IP 转发：

```
# sysctl -w net.ipv4.ip_forward=1
```

2. 为添加到集群中的每个节点添加路由：

```
# ip route add <node_subnet> via <node_ip_address> dev <interface through which node is
L2 accessible>
# ip route add 11.11.1.0/24 via 192.168.2.1 dev p3p1
# ip route add 11.11.2.0/24 via 192.168.3.3 dev p3p2
# ip route add 11.11.3.0/24 via 192.168.3.4 dev p3p2
```

第 35 章 EDGE LOAD BALANCERS 的路由

35.1. 概述

OpenShift Container Platform 集群内部的 pod 只能通过集群网络上的 IP 地址访问。边缘负载均衡器可用于接受外部网络的流量，并将流量代理到 OpenShift Container Platform 集群内的 pod。如果负载均衡器不是集群网络的一部分，则路由变成 hurdle，因为边缘负载均衡器无法访问内部集群网络。

要解决这个问题，OpenShift Container Platform 集群使用 OpenShift Container Platform SDN 作为集群网络解决方案，可通过两种方式来实现 pod 的网络访问。

35.2. 在 SDN 中包含 LOAD BALANCER

若有可能，在使用 OpenShift SDN 作为网络插件的负载均衡器本身上运行 OpenShift Container Platform 节点实例。这样，边缘机器获取自己的 Open vSwitch 网桥，SDN 会自动配置为提供对集群中的 pod 和节点的访问。*路由表*由 SDN 动态配置，因为 pod 被创建和删除，因此路由软件可以访问 pod。

将负载均衡器机器标记为不可调度的节点，以确保没有 pod 被调度到负载均衡器本身：

```
$ oc adm manage-node <load_balancer_hostname> --schedulable=false
```

如果负载均衡器被打包为容器，则与 OpenShift Container Platform 集成也更容易：只需将负载均衡器作为公开主机端口的 pod 运行。OpenShift Container Platform 中预打包的 HAProxy 路由器 以精确的方式运行。

35.3. 使用示例节点建立 TUNNEL

在某些情况下，以前的解决方案是不可能的。例如，**F5 BIG-IP®** 主机无法运行 OpenShift Container Platform 节点实例或 OpenShift Container Platform SDN，因为 **F5®** 使用了自定义的、不兼容的 Linux 内核和分布。

相反，要使 **F5 BIG-IP®** 可访问 pod，您可以在集群网络中选择一个现有节点作为一个 *坡道节点（ramp node）*，并在 **F5 BIG-IP®** 主机和指定的管道之间建立隧道。由于这是常规的 OpenShift Container Platform 节点，可获取必要的配置来将流量路由到集群网络中的任何节点上的任何 pod。因此，使用节点来假定 **F5 BIG-IP®** 主机能够访问整个集群网络的网关角色。

以下是在 **F5 BIG-IP®** 主机和指定通道节点之间建立 **ipip** 隧道的示例。

在 F5 BIG-IP® 主机上：

1. 设置以下变量：

```
# F5_IP=10.3.89.66 (1)
# RAMP_IP=10.3.89.89 (2)
# TUNNEL_IP1=10.3.91.216 (3)
# CLUSTER_NETWORK=10.128.0.0/14 (4)
```

(1) (2) **F5_IP** 和 **RAMP_IP** 变量分别指 **F5 BIG-IP®** 主机的 IP 地址，以及节点 IP 地址（在共享内部网络上）。

(3) **F5®** 主机的 **ipip** 隧道结尾的一个任意的、非冲突的 IP 地址。

(4) OpenShift SDN 用来为 pod 分配地址的覆盖网络 CIDR 范围。

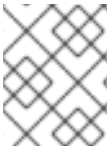
2. 删除所有旧的路由、自助、隧道和 SNAT 池：

```
# tmsh delete net route $CLUSTER_NETWORK || true
# tmsh delete net self SDN || true
# tmsh delete net tunnels tunnel SDN || true
# tmsh delete ltm snatpool SDN_snatpool || true
```

3. 创建新的隧道、自助、路由和 SNAT 池，并使用虚拟服务器中的 SNAT 池：

```
# tmsh create net tunnels tunnel SDN \
    \{ description "OpenShift SDN" local-address \
    $F5_IP profile ipip remote-address $RAMP_IP \}
# tmsh create net self SDN \{ address \
    ${TUNNEL_IP1}/24 allow-service all vlan SDN \}
# tmsh create net route $CLUSTER_NETWORK interface SDN
# tmsh create ltm snatpool SDN_snatpool members add { $TUNNEL_IP1 }
# tmsh modify ltm virtual  ose-vserver source-address-translation { type snat pool
SDN_snatpool }
# tmsh modify ltm virtual  https-ose-vserver source-address-translation { type snat pool
SDN_snatpool }
```

在通道节点上：



注意

以下会创建一个不是持久性的配置，这意味着当节点或 **openvswitch** 服务重启时，设置会消失。

1. 设置以下变量：

```
# F5_IP=10.3.89.66
# TUNNEL_IP1=10.3.91.216
# TUNNEL_IP2=10.3.91.217 (1)
# CLUSTER_NETWORK=10.128.0.0/14 (2)
```

(1) 第二个是 **ipip** 隧道的 ramp 节点端的任意 IP 地址。

(2) OpenShift SDN 用来为 pod 分配地址的覆盖网络 CIDR 范围。

2. 删除所有旧的隧道：

```
# ip tunnel del tun1 || true
```

3. 使用适当的 L2-connected 接口（如 **eth0**）在通道节点上创建 **ipip** 隧道：

```
# ip tunnel add tun1 mode ipip \
    remote $F5_IP dev eth0
# ip addr add $TUNNEL_IP2 dev tun1
# ip link set tun1 up
# ip route add $TUNNEL_IP1 dev tun1
# ping -c 5 $TUNNEL_IP1
```

4. SNAT，带有来自 SDN 子网的未使用 IP 的隧道 IP：

```
# source /run/openshift-sdn/config.env
# tap1=$(ip -o -4 addr list tun0 | awk '{print $4}' | cut -d/ -f1 | head -n 1)
# subaddr=$(echo ${OPENSHIFT_SDN_TAP1_ADDR:-"$tap1"} | cut -d "." -f 1,2,3)
# export RAMP_SDN_IP=${subaddr}.254
```

5. 将此 **RAMP_SDN_IP** 分配为 **tun0** 的额外地址（本地 SDN 的网关）：

```
# ip addr add ${RAMP_SDN_IP} dev tun0
```

6. 修改 SNAT 的 OVS 规则：

```
# ipflowopts="cookie=0x999,ip"
# arpflowopts="cookie=0x999, table=0, arp"
#
# ovs-ofctl -O OpenFlow13 add-flow br0 \

"${ipflowopts},nw_src=${TUNNEL_IP1},actions=mod_nw_src:${RAMP_SDN_IP},resubmit(,0)"

# ovs-ofctl -O OpenFlow13 add-flow br0 \

"${ipflowopts},nw_dst=${RAMP_SDN_IP},actions=mod_nw_dst:${TUNNEL_IP1},resubmit(,0)"

# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "${arpflowopts}, arp_tpa=${RAMP_SDN_IP}, actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "${arpflowopts}, priority=200, in_port=2, arp_spa=${RAMP_SDN_IP},
arp_tpa=${CLUSTER_NETWORK}, actions=goto_table:30"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "arp, table=5, priority=300, arp_tpa=${RAMP_SDN_IP}, actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "ip,table=5,priority=300,nw_dst=${RAMP_SDN_IP},actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0
"${ipflowopts},nw_dst=${TUNNEL_IP1},actions=output:2"
```

7. 另外，如果您不计划将路线图节点配置为具有高可用性，请将通道节点标记为不可调度。如果计划按照下一小节进行操作，请跳过这一步，并计划创建高度可用的节点。

```
$ oc adm manage-node <ramp_node_hostname> --schedulable=false
```

35.3.1. 配置高可用性 Ramp 节点

您可以使用 OpenShift Container Platform 的 **ipfailover** 功能（在内部使用 **keepalived** ）使通道节点从 **F5 BIG-IP®**的时间点可用。为此，首先要在相同的 L2 子网上启动两个节点，例如名为 **ramp-node-1** 和 **ramp-node-2**。

然后，从同一子网中选择一些未分配的 IP 地址，以用于您的虚拟 IP 或 *VIP*。这将设置为 **RAMP_IP** 变量，您要在 **F5 BIG-IP®** 上配置隧道。

例如，假设您使用 10.20.30.0/24 子网作为坡道节点，并且您将 10.20.30.2 分配给 **ramp-node-1**，将 10.20.30.3 分配给 **ramp-node-2**。对于您的 VIP，从同一 10.20.30.0/24 子网中选择一些未分配的地址，例如 10.20.30.4。然后，要配置 **ipfailover**，将两个节点标记为标签，如 **f5rampnode** ：

```
$ oc label node ramp-node-1 f5rampnode=true
$ oc label node ramp-node-2 f5rampnode=true
```

与 ipfailover 文档 的说明类似，您现在必须创建一个服务帐户并将其添加到 **特权** SCC 中。首先，创建 **f5ipfailover** 服务帐户：

```
$ oc create serviceaccount f5ipfailover -n default
```

接下来，您可以将 **f5ipfailover** 服务添加到 **特权** SCC。要将 **default** 命名空间中的 **f5ipfailover** 添加到 **特权** SCC，请运行：

```
$ oc adm policy add-scc-to-user privileged system:serviceaccount:default:f5ipfailover
```

最后，使用您选择的 VIP（**RAMP_IP** 变量）和 **f5 ipfailover** 服务帐户配置 ipfailover，使用您之前设置的 **f5rampnode** 标签将 VIP 分配给您的两个节点：

```
# RAMP_IP=10.20.30.4
# IFNAME=eth0 (1)
# oc adm ipfailover <name-tag> \
    --virtual-ips=$RAMP_IP \
    --interface=$IFNAME \
    --watch-port=0 \
    --replicas=2 \
    --service-account=f5ipfailover  \
    --selector='f5rampnode=true'
```

(1) 应该配置 **RAMP_IP** 的接口。

使用以上设置时，当当前分配了它的节点主机失败时，VIP（**RAMP_IP** 变量）会自动重新分配。

第 36 章 聚合容器日志

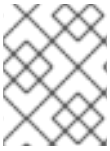
36.1. 概述

作为 OpenShift Container Platform 集群管理员，您可以部署 EFK 堆栈来聚合各种 OpenShift Container Platform 服务的日志。应用程序开发人员可以查看它们具有查看访问权限的项目的日志。EFK 堆栈聚合来自主机和应用的日志，无论是来自多个容器还是已删除的 pod。

EFK 堆栈是 ELK 堆栈 的修改版本，它由以下组成：

- Elasticsearch(ES) ：存储所有日志的对象存储。
- Fluentd ：从节点收集日志并将其提供给 Elasticsearch。
- Kibana ：Elasticsearch 的 Web UI。

在集群中部署后，堆栈将所有节点和项目的日志聚合到 Elasticsearch 中，并提供 Kibana UI 来查看任何日志。集群管理员可以查看所有日志，但应用程序开发人员只能查看他们有权查看的项目的日志。堆栈组件安全地通信。



注意

管理 Docker Container Logs 将讨论使用 **json-file** 日志记录驱动程序选项来管理容器日志并防止节点磁盘填满。

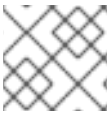
36.2. 预部署配置

1. Ansible playbook 可用于部署和升级聚合日志记录。您应该熟悉 安装集群 指南。这提供了准备使用 Ansible 的信息，并包含有关配置的信息。参数添加到 Ansible 清单文件中，以配置 EFK 堆栈的不同区域。
2. 查看 大小指南 以确定如何最佳配置部署。
3. 确保已为集群部署了路由器。
4. 确保具有 Elasticsearch 所需的存储。注意每个 Elasticsearch 节点都需要自己的存储卷。如需更多信息，请参阅 Elasticsearch。
5. 确定是否需要 高可用性 Elasticsearch。高可用性环境需要至少三个 Elasticsearch 节点，各自在不同的主机上。默认情况下，OpenShift Container Platform 会为这些分片创建每个索引和零副本的分片。高可用性还需要每个分片的多个副本。要创建高可用性，请将 **openshift_logging_es_number_of_replicas** Ansible 变量设置为 高于 **1** 的值。如需更多信息，请参阅 Elasticsearch。

36.3. 指定日志记录 ANSIBLE 变量

您可以通过在清单主机文件中为 EFK 部署指定参数来覆盖默认的参数值。

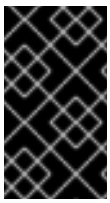
在选择参数前，读取 Elasticsearch 和 Fluentd 部分：



注意

默认情况下，Elasticsearch 服务使用端口 9300 进行集群中的节点间的 TCP 通信。

参数	描述
openshift_logging_install_logging	设置为 **true** 来安装日志记录。设置为 **false** 以卸载日志记录。当设置为 **true** 时，您必须使用 **openshift_logging_es_nodeselector** 指定节点选择器。
openshift_logging_use_ops	如果设置为 **true**，请为操作日志配置第二个 Elasticsearch 集群和 Kibana。Fluentd 在主集群和为操作日志保留的集群日志之间分割日志，其中包括项目 **默认值**、**openshift** 和 **openshift-infra**、Docker、OpenShift 和系统日志中的日志。这意味着部署了第二个 Elasticsearch 集群和 Kibana。部署可以通过其名称中包含的 **-ops** 后缀区分，并在下面列出并行部署选项，如 创建 Curator 配置 中所述。如果设置为 **true**，则必须**openshift_logging_es_ops_nodeselector**。
openshift_logging_master_url	Kubernetes 主机的 URL，这并不是面向公共的，但应该可以从集群内部访问。例如，https://<PRIVATE-MASTER-URL>:8443。
openshift_logging_purge_logging	常见的卸载会保留 PVC，以防止在重新安装过程中不需要的数据丢失。为确保 Ansible playbook 完全删除了所有日志持久性数据，包括 PVC，将 **openshift_logging_install_logging** 设置为 **false** 以触发卸载，将 **openshift_logging_purge_logging** 设置为 **true**。默认值为 **false**。
openshift_logging_install_eventrouter	与 **openshift_logging_install_logging** 相结合。当两者都设为 **true** 时，将安装eventrouter。当两者都是 **false** 时，eventrouter 将被卸载。
openshift_logging_eventrouter_image	Eventrouter 的镜像版本。例如： **registry.redhat.io/openshift3/ose-logging-eventrouter:v3.11**
openshift_logging_eventrouter_image_version	日志记录路由器的 **镜像版本**。
openshift_logging_eventrouter_sink	为 eventrouter,支持的 **stdout** 和 **glog** 选择一个 sink。默认值为 **stdout**。
openshift_logging_eventrouter_nodeselector	标签映射，如 **"node":"infra"**、**"region":"west"**，以选择 pod 将在其中加入的节点。
openshift_logging_eventrouter_replicas	默认值为 **1**。
openshift_logging_eventrouter_cpu_limit	分配给路由器的最小 CPU **量**。默认值为 **100m**。
openshift_logging_eventrouter_memory_limit	eventrouter pod 的内存限值。默认值为 **128Mi**。

参数	描述
openshift_logging_eventrouter_namespace	部署 source 的项目。默认设置为 **默认值**。  **重要** 不要将项目设置为 **default** 或 **openshift-*** 以外的任何其他对象。如果指定了不同的项目，则其他项目中的事件信息可能会泄漏到不仅限于操作用户的索引中。要使用非默认项目，请照常使用 **oc new-project** 创建项目。
openshift_logging_image_pull_secret	指定用于从经过身份验证的 registry 中拉取组件镜像的现有 pull secret 名称。
openshift_logging_curator_image	Curator 的镜像版本。例如： **registry.redhat.io/openshift3/ose-logging-curator5:v3.11**
openshift_logging_curator_default_days	Curator 用于删除日志记录的默认最短期限（以天为单位）。
openshift_logging_curator_run_hour	Curator 将运行的时间（天中的小时）。
openshift_logging_curator_run_minute	Curator 将运行的时间（小时中的的分钟数）。
openshift_logging_curator_run_timezone	Curator 使用判断运行时间的时区。以 tzselect(8)或 timedatectl(1)"Region/Locality" 格式提供时区，如 **America/New_York** 或 **UTC**。
openshift_logging_curator_script_log_level	Curator 的脚本日志级别。
openshift_logging_curator_log_level	Curator 进程的日志级别。
openshift_logging_curator_cpu_limit	要分配给 Curator 的 CPU 数量。
openshift_logging_curator_memory_limit	要分配给 Curator 的内存量。
openshift_logging_curator_nodeselector	节点选择器，指定哪些节点是部署 Curator 实例的合格目标。

参数	描述
openshift_logging_curator_ops_cpu_limit	当 **openshift_logging_use_ops** 设置为 **true** 时，相当于 Ops 集群的 **openshift_logging_curator_cpu_limit**。
openshift_logging_curator_ops_memory_limit	当 **openshift_logging_use_ops** 设置为 **true** 时，相当于 Ops 集群的 **openshift_logging_curator_memory_limit**。
openshift_logging_curator_replace_configmap	设置为 **no** 以防止升级替换 **logging-curator** ConfigMap。设置为 **yes**，以允许 ConfigMap 被覆盖。
openshift_logging_kibana_image	Kibana 的镜像版本。例如： **registry.redhat.io/openshift3/ose-logging-kibana5:v3.11**
openshift_logging_kibana_hostname	Web 客户端的外部主机名，用于访问 Kibana。
openshift_logging_kibana_cpu_limit	要分配给 Kibana 的 CPU 数量。
openshift_logging_kibana_memory_limit	要分配给 Kibana 的内存量。
openshift_logging_kibana_proxy_image	Kibana 代理的镜像版本。例如： **registry.redhat.io/openshift3/oauth-proxy:v3.11**
openshift_logging_kibana_proxy_debug	为 **true** 时，将 Kibana Proxy 日志级别设置为 **DEBUG**。
openshift_logging_kibana_proxy_cpu_limit	要分配给 Kibana 代理的 CPU 数量。
openshift_logging_kibana_proxy_memory_limit	要分配给 Kibana 代理的内存量。
openshift_logging_kibana_replica_count	Kibana 应扩展的节点数量。
openshift_logging_kibana_nodeselector	节点选择器，指定哪些节点有资格部署 Kibana 实例。
openshift_logging_kibana_env_vars	要添加到 Kibana 部署配置的环境变量映射。例如，{"ELASTICSEARCH_REQUESTTIMEOUT":"30000"}。
openshift_logging_kibana_key	创建 Kibana 路由时使用的面向公钥的公钥。

参数	描述
openshift_logging_kibana_cert	在创建 Kibana 路由时与该密钥匹配的证书。
openshift_logging_kibana_ca	可选。要用于创建 Kibana 路由时使用的密钥和证书。
openshift_logging_kibana_ops_hostname	当 **openshift_logging_use_ops** 设为 **true** 时，Daation cluster 等同于 **openshift_logging_kibana_hostname**。
openshift_logging_kibana_ops_cpu_limit	当 **openshift_logging_use_ops** 设置为 **true** 时，适用于 Ops 集群的 **openshift_logging_kibana_cpu_limit** 等同于 openshift_logging_kibana_limit。
openshift_logging_kibana_ops_memory_limit	当 **openshift_logging_use_ops** 设置为 **true** 时，适用于 Ops 集群的 **openshift_logging_kibana_memory_limit** 等同于 openshift_logging_kibana_limit。
openshift_logging_kibana_ops_proxy_debug	当 **openshift_logging_use_ops** 设置为 **true** 时，适用于 Ops 集群的 **openshift_logging_kibana_proxy_debug** 等同于 openshift_logging_kibana_debug。
openshift_logging_kibana_ops_proxy_cpu_limit	当 **openshift_logging_use_ops** 设置为 **true** 时，适用于 Ops 集群的 **openshift_logging_kibana_proxy_cpu_limit** 等同于 openshift_logging_kibana_cpu_limit。
openshift_logging_kibana_ops_proxy_memory_limit	当将 **openshift_logging_use_ops** 设置为 **true** 时，适用于 Ops 集群的 **openshift_logging_kibana_proxy_memory_limit** 等同于 openshift_logging_kibana_memory_limit。
openshift_logging_kibana_ops_replica_count	当 **openshift_logging_use_ops** 设置为 **true** 时，Opeross cluster 的运营集群等同于 **openshift_logging_kibana_replica_count**。
openshift_logging_es_allow_external	设置为 **true**，以将 Elasticsearch 公开为重新加密路由。默认设置为 **false**。
openshift_logging_es_hostname	用于路由和 TLS 服务器证书的面向外部主机名。默认值为 **es**。 例如，如果 **openshift_master_default_subdomain** 设为 **=example.test**，则 **openshift_logging_es_hostname** 的默认值将是 **es.example.test**。
openshift_logging_es_cert	Elasticsearch 用于外部 TLS 服务器证书的证书位置。默认为生成的证书。
openshift_logging_es_key	Elasticsearch 用于外部 TLS 服务器证书的密钥位置。默认为生成的密钥。

参数	描述
openshift_logging_es_ca_ext	用于外部 TLS 服务器证书的 CA 证书 Elasticsearch 的位置。默认为内部 CA。
openshift_logging_es_ops_allow_external	设置为 **true**，以将 Elasticsearch 公开为重新加密路由。默认设置为 **false**。
openshift_logging_es_ops_hostname	用于路由和 TLS 服务器证书的面向外部主机名。默认值为 **es-ops**。 例如，如果 **openshift_master_default_subdomain** 设为 **=example.test**，则 **openshift_logging_es_ops_hostname** 的默认值为 **es-ops.example.test**。
openshift_logging_es_ops_cert	Elasticsearch 用于外部 TLS 服务器证书的证书位置。默认为生成的证书。
openshift_logging_es_ops_key	Elasticsearch 用于外部 TLS 服务器证书的密钥位置。默认为生成的密钥。
openshift_logging_es_ops_ca_ext	用于外部 TLS 服务器证书的 CA 证书 Elasticsearch 的位置。默认为内部 CA。
openshift_logging_fluentd_image	Fluentd 的镜像版本。例如： **registry.redhat.io/openshift3/ose-logging-fluentd:v3.11**
openshift_logging_fluentd_nodeselector	一个节点选择器，用于指定部署 Fluentd 实例的具有哪些节点。任何 Fluentd 应该运行（通常情况下，所有）的节点都必须具有此标签，然后才能运行和收集日志。 在安装后扩展 Aggregated Logging 集群时，**openshift_logging_fluentd_hosts 提供的 openshift_logging** 角色标签节点使用此节点选择器。 作为安装的一部分，建议您将 Fluentd 节点选择器标签添加到持久的节点标签列表中。
openshift_logging_fluentd_cpu_limit	Fluentd Pod 的 CPU 限值。
openshift_logging_fluentd_memory_limit	Fluentd Pod 的内存限值。
openshift_logging_fluentd_journal_read_from_head	如果 Fluentd 首次启动时从 Journal 的 head 中读取，则设置为 **true**，则使用此操作可能会在 Elasticsearch 收到当前日志记录时造成延迟。
openshift_logging_fluentd_hosts	应该为要部署的 Fluentd 标记的节点列表。默认为使用 ['--all'] 标记所有节点。null 值是 **openshift_logging_fluentd_hosts={}**。启动 Fluentd pod 将 daemonset 的 **nodeSelector** 更新为有效的标签。例如，['host1.example.com', 'host2.example.com']。

参数	描述
openshift_logging_fluentd_audit_container_engine	当 **openshift_logging_fluentd_audit_container_engine** 设置为 **true** 时，容器引擎的审计日志会被收集并存储在 ES 中。启用此变量允许 EFK 监视指定的审计日志文件或默认的 **/var/log/audit.log** 文件，为平台收集容器引擎的审计信息，然后将其放入 Kibana。
openshift_logging_fluentd_audit_file	审计日志文件的位置。默认为 **/var/log/audit/audit.log**。启用此变量允许 EFK 监视指定的审计日志文件或默认的 **/var/log/audit.log** 文件，为平台收集容器引擎的审计信息，然后将其放入 Kibana。
openshift_logging_fluentd_audit_pos_file	审计日志文件的 Fluentd **in_tail** 位置文件的位置。默认为 **/var/log/audit/audit.log.pos**。启用此变量允许 EFK 监视指定的审计日志文件或默认的 **/var/log/audit.log** 文件，为平台收集容器引擎的审计信息，然后将其放入 Kibana。
openshift_logging_fluentd_merge_json_log	设置为 **true** 以启用在记录的 **log** 或 **MESSAGE** 字段中处理 JSON 日志的处理。默认值是 **true**。
openshift_logging_fluentd_extra_keep_fields	指定在处理使用 **openshift_logging_fluentd_merge_json_log** 时生成的额外字段时不想更改的字段列表。否则，Fluentd 根据下面的其他未定义字段设置处理字段。默认值为空。
openshift_logging_fluentd_keep_empty_fields	使用 **openshift_logging_fluentd_merge_json_log** 时指定以逗号分隔的字段列表，以保留为空白字段。默认情况下，除了 **message** 字段外，Fluentd 从记录中删除带有空值的字段。
openshift_logging_fluentd_replace_configmap	设置为 **no** 以防止升级替换 **logging-fluentd** ConfigMap。设置为 **yes**，以允许 ConfigMap 被覆盖。
openshift_logging_fluentd_use_undefined	设置为 **true**，将 **openshift_logging_fluentd_merge_json_log** 生成的字段移到由 **openshift_logging_fluentd_undefined_name** 参数命名的子字段。默认情况下，Fluentd 会保留这些记录的最顶层，这会导致 Elasticsearch 冲突和模式错误。
openshift_logging_fluentd_undefined_name	指定在使用 **openshift_logging_fluentd_use_undefined** 时将未定义字段的名称移到 中。默认值为 **undefined**。
openshift_logging_fluentd_undefined_to_string	设置为 **true**，在使用 **openshift_logging_fluentd_merge_json_log** 时，将所有未定义字段值转换为其 JSON 字符串表示形式。默认值为 **false**。
openshift_logging_fluentd_undefined_dot_replace_char	使用 **openshift_logging_fluentd_merge_json _ log** 指定一个字符来替换字段名称中的任何 **.** 字符，如 _。名称中带有 . **字符的** 未定义字段会导致 Elasticsearch 出现问题。默认为 **UNUSED**， **即在** 字段名称中保留。

参数	描述
openshift_logging_fluentd_undefined_max_num_fields	使用 **openshift_logging_fluentd_merge_json_log** 时，指定到未定义字段数量的限制。日志可以包含数百个未定义字段，这会导致 Elasticsearch 出现问题。如果有多个字段，这些字段将转换为 JSON 散列字符串，并存储在 **openshift_logging_fluentd_undefined_name** 字段中。默认值为 **-1**，表示无限数量的字段。
openshift_logging_fluentd_use_multiline_json	设置为 **true**，强制 Fluentd 在使用 **openshift_logging_fluentd_merge_json_log** 时，将任何分割日志行重新创建为一行。使用 **json-file** 驱动程序，Docker 会分割日志行大小为 16k 字节。默认值为 **false**。
openshift_logging_fluentd_use_multiline_journal	设置为 **true**，在使用 **openshift_logging_fluentd_merge_json_log** 时，强制 Fluentd 将分割行重新创建为一行。使用 **journald** 驱动程序时，Docker 会分割日志行大小为 16k 字节。默认值为 **false**。
openshift_logging_es_host	Fluentd 应该发送日志的 Elasticsearch 服务的名称。
openshift_logging_es_port	Fluentd 应该发送日志的 Elasticsearch 服务的端口。
openshift_logging_es_ca	CA Fluentd 用来与 **openshift_logging_es_host** 通信的位置。
openshift_logging_es_client_cert	客户端证书 Fluentd 用于 **openshift_logging_es_host** 的位置。
openshift_logging_es_client_key	客户端密钥 Fluentd 用于 **openshift_logging_es_host** 的位置。
openshift_logging_es_cluster_size	要部署的 Elasticsearch 节点。高可用性需要三个或更多.
openshift_logging_es_cpu_limit	Elasticsearch 集群的 CPU 限值量。
openshift_logging_es_memory_limit	每个 Elasticsearch 实例保留的 RAM 量。它必须至少 512M。可能的后缀为 G,g,M,m。
openshift_logging_es_number_of_replicas	每个新索引的每个主分片的副本数。默认值为 '0'。在生产环境中，推荐最少使用 **1**。对于高可用性环境，请将此值设置为 **1** 个或更多，并且至少有三个 Elasticsearch 节点，各自在不同的主机上。如果更改副本数，则新值只会应用到新索引。新号不适用于现有的索引。有关如何更改现有索引的副本数量的信息，请参阅更改 Elasticsearch 副本的数量。
openshift_logging_es_number_of_shards	在 ES 中创建的每个新索引的主分片数量。默认为 **1**。

参数	描述
openshift_logging_es_pv_selector	添加到 PVC 的键/值映射，以选择特定的 PV。
openshift_logging_es_pvc_dynamic	要动态置备后备存储，将参数值设置为 **true**。当设置为 **true** 时，PVC 定义中会忽略 storageClass 规格。当设置为 **false** 时，必须为 **openshift_logging_es_pvc_size** 参数指定一个值。 如果您为 **openshift_logging_es_pvc_storage_class_name** 参数设置一个值，其值会覆盖 **openshift_logging_es_pvc_dynamic** 参数的值。
openshift_logging_es_pvc_storage_class_name	要使用非默认存储类，请指定存储类名称，如 **glusterprovisioner** 或 **cephrbdprovisioner**。指定存储类名称后，无论 **openshift_logging_es_pvc_dynamic** 值是什么，动态卷置备都会活跃。
openshift_logging_es_pvc_size	为每个 Elasticsearch 实例创建的持久性卷声明的大小。例如：100G。如果省略，则不会创建 PVC，并且会改为使用临时卷。如果设置此参数，日志记录安装程序将 **openshift_logging_elasticsearch_storage_type** 设置为 **pvc**。 如果 **openshift_logging_es_pvc_dynamic** 参数设置为 **false**，则必须为此参数设置一个值。有关详细信息，请阅读 **openshift_logging_es_pvc_prefix** 的描述。
openshift_logging_elasticsearch_image	Elasticsearch 的镜像版本。例如： **registry.redhat.io/openshift3/ose-logging-elasticsearch5:v3.11**
openshift_logging_elasticsearch_storage_type	设置 Elasticsearch 存储类型。如果使用 Persistent Elasticsearch Storage，日志记录安装程序把它设置为 **pvc**。
openshift_logging_es_pvc_prefix	用作 Elasticsearch 节点的存储声明名称的前缀。每个节点附加一个数字，如 logging-es-1。如果不存在，则使用大小为 ***es-pvc-size*** 创建它们。 当设置了 **openshift_logging_es_pvc_prefix** 时，并： • **openshift_logging_es_pvc_dynamic=true**，**openshift_logging_es_pvc_size** 的值是可选的。 • **openshift_logging_es_pvc_dynamic=false**，必须设置 **openshift_logging_es_pvc_size** 的值。
openshift_logging_es_recover_after_time	Elasticsearch 在尝试恢复前等待的时间。支持的时间单位为秒(s)或分钟(m)。
openshift_logging_es_storage_group	用于访问 Elasticsearch 存储卷的补充组 ID 数量。后端卷应允许此组 ID 访问。

参数	描述
openshift_logging_es_nodeselector	节点选择器作为映射指定，决定哪些节点是部署 Elasticsearch 节点的合格目标。使用此映射将这些实例放在为运行它们保留或优化的节点上。例如，选择器可以是 **{"node-role.kubernetes.io/infra":"true"}**。至少一个活跃的节点必须具有此标签才能部署 Elasticsearch。在安装日志记录时，此参数是必须的。
openshift_logging_es_ops_host	当 **openshift_logging_use_ops** 设置为 **true** 时，相当于 Ops 集群的 **openshift_logging_es_host**。
openshift_logging_es_ops_port	当 **openshift_logging_use_ops** 设置为 **true** 时，相当于 Ops 集群的 **openshift_logging_es_port**。
openshift_logging_es_ops_ca	当 **openshift_logging_use_ops** 设置为 **true** 时，相当于 Ops 集群的 **openshift_logging_es_ca**。
openshift_logging_es_ops_client_cert	当 **openshift_logging_use_ops** 设置为 **true** 时，等同于 **openshift_logging_es_client_cert** for Ops 集群。
openshift_logging_es_ops_client_key	当 **openshift_logging_use_ops** 设置为 **true** 时，相当于 Ops 集群的 **openshift_logging_es_client_key**。
openshift_logging_es_ops_cluster_size	当 **openshift_logging_use_ops 设置为 true 时，适用于 Ops 集群的 openshift_logging_es_size**。
openshift_logging_es_ops_cpu_limit	当 **openshift_logging_use_ops** 设置为 **true** 时，相当于 Ops 集群的 **openshift_logging_es_cpu_limit**。
openshift_logging_es_ops_memory_limit	当 **openshift_logging_use_ops** 设为 **true** 时，Daation cluster 等同于 **openshift_logging_es_memory_limit**。
openshift_logging_es_ops_pv_selector	当 **openshift_logging_use_ops** 设置为 **true** 时，相当于 Ops 集群的 **openshift_logging_es_pv_selector**。
openshift_logging_es_ops_pvc_dynamic	当 **openshift_logging_use_ops** 设置为 **true** 时相当于 Ops 集群的 **openshift_logging_es_pvc_dynamic**。
openshift_logging_es_ops_pvc_size	当 **openshift_logging_use_ops** 设置为 **true** 时，适用于 Ops 集群的 **openshift_logging_es_pvc_size** 等同于 openshift_logging_use_size。
openshift_logging_es_ops_pvc_prefix	当 **openshift_logging_use_ops** 设为 **true** 时，操作集群等同于 **openshift_logging_es_pvc_prefix**。
openshift_logging_es_ops_storage_group	当 **openshift_logging_use_ops** 设置为 **true** 时，相当于 Ops 集群的 **openshift_logging_es_storage_group**。

参数	描述
openshift_logging_es_ops_nodeselector	节点选择器，指定哪些节点有资格部署 Elasticsearch 节点。这可用于将这些实例放在保留或优化的节点上以便运行它们。例如，选择器可以是 **node-type=infrastructure**。至少一个活跃的节点必须具有此标签才能部署 Elasticsearch。当 **openshift_logging_use_ops** 设置为 **true** 时，此参数是必需的。

openshift_logging_elasticsearch_kibana_index_mode	默认值（**唯一**）允许用户各自拥有自己的 Kibana 索引。在这个模式中，它们保存的查询、视觉化和仪表板不会被共享。
	
	您也可以设置值 **shared_ops**。在这个模式中，所有操作用户共享 Kibana 索引，允许每个操作用户查看相同的查询、视觉化和仪表板。要确定您是操作用户：
	
	`#oc auth can-i view pod/logs -n default`
	`yes`
	
	如果您没有适当的访问权限，请联络您的集群管理员。

openshift_logging_elasticsearch_poll_timeout_minutes	调整 Ansible playbook 在升级给定 Elasticsearch 节点后等待 Elasticsearch 集群进入绿色状态的时间。大型分片（50 GB 或更多）可以花费超过 60 分钟的时间来初始化，从而导致 Ansible playbook 中止升级过程。默认值为 **60**。
openshift_logging_kibana_ops_nodeselector	节点选择器，指定哪些节点有资格部署 Kibana 实例。
openshift_logging_curator_ops_nodeselector	节点选择器，指定哪些节点是部署 Curator 实例的合格目标。
openshift_logging_elasticsearch_replace_configmap	设置为 **true**，将 **logging-elasticsearch** ConfigMap 替换为当前默认值。您当前的 ConfigMap 保存到 **logging-elasticsearch.old** 中，您可以使用它们将自定义复制到新 ConfigMap。在某些情况下，使用旧的 ConfigMap 可能会导致升级失败。默认值为 **false**。

自定义证书

您可以使用以下清单变量来指定自定义证书，而不依赖于部署过程中生成的证书。这些证书用于加密和保护用户浏览器和 Kibana 之间的通信。如果没有提供与安全相关的文件，则会生成它们。

文件名	描述
openshift_logging_kibana_cert	Kibana 服务器的面向浏览器的证书。
openshift_logging_kibana_key	与面向浏览器的 Kibana 证书一起使用的密钥。
openshift_logging_kibana_ca	控制节点上的绝对路径到 CA 文件，用于将浏览器用于 Kibana 证书。

文件名	描述
openshift_logging_kibana_ops_cert	Ops Kibana 服务器的面向浏览器的证书。
openshift_logging_kibana_ops_key	要与面向浏览器的 Ops Kibana 证书一起使用的密钥。
openshift_logging_kibana_ops_ca	控制节点上到 CA 文件的绝对路径，用于面向 ops Kibana certs 的浏览器。

如果需要重新部署这些证书，请参阅重新部署 EFK 证书。

36.4. 部署 EFK 堆栈

EFK 堆栈使用 Ansible playbook 部署到 EFK 组件。使用默认清单文件，从默认 OpenShift Ansible 位置运行 playbook。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i </path/to/inventory>] \
    playbooks/openshift-logging/config.yml
```

运行 playbook 会部署支持堆栈所需的所有资源；如 Secrets、ServiceAccounts 和 DeploymentConfigs，部署到项目 **openshift-logging**。playbook 会等待部署组件 Pod，直到堆栈运行为止。如果等待步骤失败，则部署可能仍然成功；可以从 registry 检索组件镜像（最多需要几分钟）。您可以通过以下方法监视进程：

```
$ oc get pods -w

logging-curator-1541129400-l5h77          0/1       Running   0          11h  1
logging-es-data-master-ecu30lr4-1-deploy  0/1       Running   0          11h  2
logging-fluentd-2lgwn                     1/1       Running   0          11h  3
logging-fluentd-lmvms                     1/1       Running   0          11h
logging-fluentd-p9nd7                     1/1       Running   0          11h
logging-kibana-1-zk94k                    2/2       Running   0          11h  4
```

1. Curator Pod。Curator 只需要一个 pod。
2. 此主机上的 Elasticsearch pod。
3. Fliuentd pod。集群中的每个节点都有一个 pod。
4. Kibana Pod。

您可以使用 'oc get pods -o wide 命令查看部署了 Fluentd Pod 的节点：

```
$ oc get pods -o wide
NAME                                      READY     STATUS    RESTARTS   AGE       IP            NODE
NOMINATED NODE
logging-es-data-master-5av030lk-1-2x494   2/2       Running   0          38m       154.128.0.80  ip-153-
12-8-6.wef.internal   <none>
```

```
logging-fluentd-lqdxg                 1/1       Running   0          2m        154.128.0.85   ip-153-12-8-
6.wef.internal   <none>
logging-kibana-1-gj5kc                2/2       Running   0          39m       154.128.0.77   ip-153-12-8-
6.wef.internal   <none>
```

它们最终将进入 **Running** 状态。如需通过检索相关事件在部署过程中 pod 状态的更多信息：

```
$ oc describe pods/<pod_name>
```

检查 pod 没有成功运行的日志：

```
$ oc logs -f <pod_name>
```

36.5. 了解并调整部署

本节论述了对部署组件所做的调整。

36.5.1. ops 集群

注意

默认、openshift 和 openshift-infra 项目的日志会自动聚合并分组到 Kibana 界面中的 **.operations** 项。

您部署 EFK 堆栈的项目（**logging**，如此处记录） *未*聚合到 **.operations**，并在其 ID 下找到。

如果在清单文件中将 **openshift_logging_use_ops** 设置为 **true**，Fluentd 会被配置为将主要 Elasticsearch 集群和另一个用于操作日志的日志分离，这些日志被定义为节点系统日志以及项目 **default**、**openshift** 和 **openshift-infra**。因此，一个单独的 Elasticsearch 集群、一个单独的 Kibana 和一个单独的 Curator 部署到索引、访问和管理操作日志。这些部署与包含 **-ops** 的名称不同。如果启用这个选项，请记住这些独立的部署。如果存在，以下大部分讨论也适用于操作集群，只需将名称更改为 include **-ops**。

36.5.2. Elasticsearch

Elasticsearch(ES) 是一个存储所有日志的对象存储。

Elasticsearch 将日志数据整理到数据存储中，各自称为 *索引*。Elasticsearch 将每个索引分成多个部分，称为 *分片（shards）*，它将分散到集群中的一组 Elasticsearch 节点上。您可以配置 Elasticsearch 来为分片制作拷贝（称为*副本*）。Elasticsearch 还会将副本分散到 Elactisearch 节点。分片和副本的组合旨在提供冗余和故障恢复能力。例如，如果您为一个副本为索引配置三个分片，Elasticsearch 会为该索引生成六个分片：三个主分片，三个副本作为备份。

OpenShift Container Platform 日志记录安装程序确保每个 Elasticsearch 节点都使用带有自身存储卷的唯一部署配置进行部署。您可以为 您添加到日志记录系统的每个 Elasticsearch 节点创建额外的部署配置。在安装过程中，您可以使用 **openshift_logging_es_cluster_size** Ansible 变量来指定 Elasticsearch 节点的数量。

另外，您可以通过修改清单文件中的 **openshift_logging_es_cluster_size** 来扩展现有集群，并重新运行日志记录 playbook。可以修改其他集群参数，并在 指定日志记录 Ansible 变量 中所述。

如需有关选择存储和网络位置的注意事项，请参考 Elastic 文档，如下所示。

注意

高可用性 Elasticsearch 环境 需要至少三个 Elasticsearch 节点，每个都在不同的主机上，并将 **openshift_logging_es_number_of_replicas** Ansible 变量设置为 **1** 或更高版本来创建副本。

查看所有 Elasticsearch 部署

查看所有当前的 Elasticsearch 部署：

```
$ oc get dc --selector logging-infra=elasticsearch
```

配置 Elasticsearch 实现高可用性

高可用性 Elasticsearch 环境需要至少三个 Elasticsearch 节点，每个都在不同的主机上，并将 **openshift_logging_es_number_of_replicas** Ansible 变量设置为 **1** 或更高版本来创建副本。

使用以下场景作为具有三个 Elasticsearch 节点的 OpenShift Container Platform 集群指南：

- 将 **openshift_logging_es_number_of_replicas** 设置为 **1**，两个节点具有集群中所有 Elasticsearch 数据的副本。这样可确保如果带有 Elasticsearch 数据的节点停机，另一个节点具有集群中所有 Elasticsearch 数据的副本。

- 将 **openshift_logging_es_number_of_replicas** 设置为 **3**，四节点具有集群中所有 Elasticsearch 数据的副本。这样可确保如果三个带有 Elasticsearch 数据的节点停机，则一个节点会拥有集群中所有 Elasticsearch 数据的副本。
  在这种情况下，如果有多个 Elasticsearch 节点停机，Elasticsearch 状态为 RED，新的 Elasticsearch 分片不会被分配。但是，由于高可用性，您不会丢失 Elasticsearch 数据。

请注意，高可用性和性能之间有一个利弊。例如，使 **openshift_logging_es_number_of_replicas=2** 和 **openshift_logging_es_number_of_shards=3** 需要 Elasticsearch 消耗大量资源在集群中复制分片数据。另外，使用数量较高的副本需要在每个节点上进行碎片或开序数据存储要求，因此在为 Elasticsearch 计划持久性存储时，您必须考虑这一点。

配置分片数量时的注意事项

对于 **openshift_logging_es_number_of_shards** 参数，请考虑：

- 要获得更高的性能，请增加分片的数量。例如，在三个节点集群中，设置 **openshift_logging_es_number_of_shards=3**。这将导致每个索引分成三个部分（分片），处理索引的负载将分散到所有 3 个节点。

- 如果您有大量项目，如果集群中有超过几千个分片，则可能会看到性能下降。减少分片数量或减少策展时间。

- 如果您有少量非常大的索引，您可能需要配置 **openshift_logging_es_number_of_shards=3** 或更高版本。Elasticsearch 建议使用最多分片大小小于 50 GB。

Node Selector

由于 Elasticsearch 可以使用许多资源，集群的所有成员都应有相互低延迟网络连接和任何远程存储。使用节点选择器将实例定向到专用节点，或集群中专用区域来确保这一点。

要配置节点选择器，请在清单文件中指定 **openshift_logging_es_nodeselector** 配置选项。这适用于所有 Elasticsearch 部署；如果需要分隔节点选择器，则必须在部署后手动编辑每个部署配置。节点选择器以 python 兼容 dict 字典的形式指定。例如，**{"node-type":"infra", "region":"east"}**。

36.5.2.1. 持久性 Elasticsearch 存储

默认情况下，**openshift_logging** Ansible 角色会创建一个临时部署，pod 中的所有数据在 pod 重启后会丢失。

对于生产环境，每个 Elasticsearch 部署配置都需要一个持久性存储卷。您可以指定现有的 持久性卷声明，或允许 OpenShift Container Platform 创建。

- **使用现有的 PVC。**如果为部署创建自己的 PVC，OpenShift Container Platform 会使用这些 PVC。
  将 PVC 命名为与 **openshift_logging_es_pvc_prefix** 设置匹配，该设置默认为 **logging-es**。为每个 PVC 分配一个名称，其中添加了一个序列号：**logging-es-0**、**logging-es-1**、**logging-es-2** 等。

- **允许 OpenShift Container Platform 创建 PVC。**如果 Elsaticsearch 的 PVC 不存在，OpenShift Container Platform 会根据 Ansible 清单文件 中的参数创建 PVC。

参数	描述
openshift_logging_es_pvc_size	指定 PVC 请求的大小。
openshift_logging_elasticsearch_storage_type	将存储类型指定为 **pvc**。 **注意** 这是一个可选参数。如果将 **openshift_logging_es_pvc_size** 参数设置为大于 0 的值，则日志记录安装程序默认自动将此参数设置为 **pvc**。
openshift_logging_es_pvc_prefix	另外，还可为 PVC 指定自定义前缀。

例如：

```
openshift_logging_elasticsearch_storage_type=pvc
openshift_logging_es_pvc_size=104802308Ki
openshift_logging_es_pvc_prefix=es-logging
```

如果使用动态置备 PV，OpenShift Container Platform 日志记录安装程序会创建一个使用默认存储类或通过 **openshift_logging_elasticsearch_pvc_class_name** 参数指定的 PVC 的 PVC。

如果使用 NFS 存储，OpenShift Container Platform 安装程序会根据 **openshift_logging_storage_*** 参数和 OpenShift Container Platform 日志记录安装程序创建持久性卷，使用 **openshift_logging_es_pvc_*** 参数创建 PVC。务必指定正确的参数，以便将持久卷与 EFK 搭配使用。另外，在 Ansible 清单文件中设置 **openshift_enable_unsupported_configurations=true** 参数，因为日志记录安装程序默认阻止了带有核心基础架构的 NFS 安装。

警告

使用 NFS 存储作为卷或持久性卷，或者 Elasticsearch 存储（如 Gluster）不支持使用 NAS，因为 Lucene 依赖于 NFS 不提供的文件系统行为。数据崩溃和其他问题可能会发生。

如果您的环境需要 NFS 存储，请使用以下方法之一：

- NFS 作为持久性卷
- NFS 存储作为本地存储

36.5.2.1.1. 使用 NFS 作为持久性卷

您可以将 NFS 部署为 自动置备的持久性卷，也可以使用 预定义的 NFS 卷。

如需更多信息，请参阅 在两个持久性卷声明间共享 NFS 挂载，以利用共享存储来供两个独立的容器使用。

使用自动置备的 NFS

使用 NFS 作为自动置备 NFS 的持久性卷：

1. 将以下行添加到 Ansible 清单文件，以创建 NFS 自动配置的存储类并动态置备后备存储：

```
openshift_logging_es_pvc_storage_class_name=$nfsclass
openshift_logging_es_pvc_dynamic=true
```

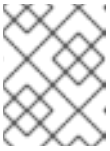
2. 使用以下命令，使用日志记录 playbook 部署 NFS 卷：

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openshift-logging/config.yml
```

3. 使用以下步骤创建 PVC：

   a. 编辑 Ansible 清单文件以设置 PVC 大小：

```
openshift_logging_es_pvc_size=50Gi
```



注意

日志记录 playbook 根据大小选择卷，如果其他持久性卷大小相同，则可能使用意外卷。

   b. 使用以下命令重新运行 Ansible ***deploy_cluster.yml*** playbook：

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

安装程序 playbook 根据 **openshift_logging_storage** 变量创建 NFS 卷。

使用预定义的 NFS 卷

使用现有 NFS 卷与 OpenShift Container Platform 集群部署日志：

1. 编辑 Ansible 清单文件以配置 NFS 卷并设置 PVC 大小：

```
openshift_logging_storage_kind=nfs
openshift_logging_storage_access_modes=['ReadWriteOnce']
openshift_logging_storage_nfs_directory=/share 1
openshift_logging_storage_nfs_options='*(rw,root_squash)' 2
openshift_logging_storage_labels={'storage': 'logging'}
openshift_logging_elasticsearch_storage_type=pvc
openshift_logging_es_pvc_size=10Gi
openshift_logging_es_pvc_storage_class_name=''
openshift_logging_es_pvc_dynamic=true
openshift_logging_es_pvc_prefix=logging
```

1 2 这些参数只适用于 **/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml** 安装 playbook。这些参数无法用于 **/usr/share/ansible/openshift-ansible/playbooks/openshift-logging/config.yml** playbook。

2. 使用以下命令重新部署 EFK 堆栈：

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

36.5.2.1.2. 使用 NFS 作为本地存储

您可以在 NFS 服务器上分配大型文件，并将该文件挂载到节点。然后，您可以使用该文件作为主机路径设备。

```
$ mount -F nfs nfserver:/nfs/storage/elasticsearch-1 /usr/local/es-storage
$ chown 1000:1000 /usr/local/es-storage
```

然后，使用 */usr/local/es-storage* 作为 host-mount，如下所述。使用不同的后备文件作为每个 Elasticsearch 节点的存储。

此回环必须在 OpenShift Container Platform 之外的节点上手动维护。您不能在容器中维护它。

每个节点主机上可以使用本地磁盘卷（如果可用）作为 Elasticsearch 副本的存储。这样做需要进行一些准备，如下所示：

1. 必须授予相关的服务帐户来挂载和编辑本地卷：

```
$ oc adm policy add-scc-to-user privileged  \
    system:serviceaccount:openshift-logging:aggregated-logging-elasticsearch
```

注意

如果您从早期版本的 OpenShift Container Platform 升级，集群日志记录可能已安装在 **logging** 项目中。您应该相应地调整服务帐户。

2. 每个 Elasticsearch 节点定义都必须补丁才能声明该权限，例如：

```
$ for dc in $(oc get deploymentconfig --selector component=es -o name); do
    oc scale $dc --replicas=0
    oc patch $dc \
      -p '{"spec":{"template":{"spec":{"containers":[{"name":"elasticsearch","securityContext":
{"privileged": true}}]}}}}'
  done
```

3. Elasticsearch 副本必须位于正确的节点上，才能使用本地存储，即使这些节点在一段时间内停机也是如此。这要求为每个 Elasticsearch 副本提供一个对管理员为其分配存储的节点的唯一节点选择器。要配置节点选择器，请编辑每个 Elasticsearch 部署配置，添加或编辑 **nodeSelector** 部分，以指定您为每个节点应用的唯一标签：

```
apiVersion: v1
kind: DeploymentConfig
spec:
  template:
    spec:
      nodeSelector:
        logging-es-node: "1" 1
```

1 此标签必须在本例中为 **logging-es-node=1** 的单个节点，并带有相应的节点。

1. 为每个所需节点创建节点选择器。
2. 根据需要，使用 **oc label** 命令将标签应用到多个节点。

例如，如果您的部署有三个基础架构节点，您可以按如下方式为这些节点添加标签：

```
$ oc label node <nodename1> logging-es-node=0
$ oc label node <nodename2> logging-es-node=1
$ oc label node <nodename3> logging-es-node=2
```

有关向节点添加标签的详情，请参考 更新节点上的标签。

要自动应用节点选择器，您可以使用 **oc patch** 命令：

```
$ oc patch dc/logging-es-<suffix> \
  -p '{"spec":{"template":{"spec":{"nodeSelector":{"logging-es-node":"0"}}}}}'
```

完成这些步骤后，您可以将本地主机挂载应用到每个副本。以下示例假设存储挂载到每个节点上的相同路径。

```
$ for dc in $(oc get deploymentconfig --selector component=es -o name); do
    oc set volume $dc \
          --add --overwrite --name=elasticsearch-storage \
          --type=hostPath --path=/usr/local/es-storage
    oc rollout latest $dc
    oc scale $dc --replicas=1
  done
```

36.5.2.1.3. 为 Elasticsearch 配置 hostPath 存储

您可以为 Elasticsearch 使用 hostPath 存储来置备 OpenShift Container Platform 集群。

使用每个节点主机上的本地磁盘卷作为 Elasticsearch 副本的存储：

1. 在每个基础架构节点上为本地 Elasticsearch 存储创建一个本地挂载点：

```
$ mkdir /usr/local/es-storage
```

2. 在 Elasticsearch 卷上创建文件系统：

```
$ mkfs.ext4 /dev/xxx
```

3. 挂载 elasticsearch 卷：

```
$ mount /dev/xxx /usr/local/es-storage
```

4. 在 **/etc/fstab** 中添加以下行：

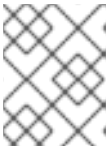
```
$ /dev/xxx /usr/local/es-storage ext4
```

5. 更改挂载点的所有权：

```
$ chown 1000:1000 /usr/local/es-storage
```

6. 赋予将本地卷挂载到相关服务帐户的权限：

```
$ oc adm policy add-scc-to-user privileged  \
      system:serviceaccount:openshift-logging:aggregated-logging-elasticsearch
```



注意

如果您从早期版本的 OpenShift Container Platform 升级，集群日志记录可能已安装在 **logging** 项目中。您应该相应地调整服务帐户。

7. 要声明该权限，请对每个 Elasticsearch 副本定义进行补丁，如下例所示，这为 Ops 集群指定 **--selector component=es-ops** ：

```
 $ for dc in $(oc get deploymentconfig --selector component=es -o name);
do
   oc scale $dc --replicas=0
   oc patch $dc \
      -p '{"spec":{"template":{"spec":{"containers":[{"name":"elasticsearch","securityContext":
{"privileged":
true}}]}}}}'
done
```

8. 在正确的节点上找到 Elasticsearch 副本以使用本地存储，也不会移动它们，即使这些节点在一段时间内停机。要指定节点位置，请为每个 Elasticsearch 副本赋予一个唯一一个管理员为其分配存储的节点的节点选择器。
要配置节点选择器，请编辑每个 Elasticsearch 部署配置，添加或编辑 **nodeSelector** 部分，以指定您所需的每个节点应用的唯一标签：

```
apiVersion: v1
kind: DeploymentConfig
```

```
spec:
  template:
    spec:
      nodeSelector:
        logging-es-node: "1"
```

该标签必须唯一地识别一个带有标签的一个节点的副本，在本例中为 **logging-es-node=1**。

9. 为每个所需节点创建节点选择器。根据需要，使用 **oc label** 命令将标签应用到多个节点。例如，如果您的部署有三个基础架构节点，您可以按如下方式为这些节点添加标签：

```
$ oc label node <nodename1> logging-es-node=0
$ oc label node <nodename2> logging-es-node=1
$ oc label node <nodename3> logging-es-node=2
```

要自动使用节点选择器的应用程序，请使用 **oc patch** 命令而不是 **oc label** 命令，如下所示：

```
$ oc patch dc/logging-es-<suffix> \
  -p '{"spec":{"template":{"spec":{"nodeSelector":{"logging-es-node":"1"}}}}}'
```

10. 完成这些步骤后，您可以将本地主机挂载应用到每个副本。以下示例假设存储挂载到每个节点的同一路径，并为 Ops 集群指定 **--selector component=es-ops**。

```
$ for dc in $(oc get deploymentconfig --selector component=es -o name);
do
   oc set volume $dc \
         --add --overwrite --name=elasticsearch-storage \
         --type=hostPath --path=/usr/local/es-storage
   oc rollout latest $dc
   oc scale $dc --replicas=1
done
```

36.5.2.1.4. 更改 Elasticsearch 的扩展

如果需要扩展集群中的 Elasticsearch 节点数量，可以为要添加的每个 Elasticsearch 节点创建部署配置。

由于持久性卷的性质以及 Elasticsearch 配置为存储数据并恢复集群的方式，您无法简单地在 Elasticsearch 部署配置中增加节点。

更改 Elasticsearch 扩展的最简单方法是修改清单主机文件，并重新运行日志记录 playbook，如前面所述。如果您为部署提供了持久性存储，则这不应具有破坏性。

注意

只有在新的 **openshift_logging_es_cluster_size** 值高于集群中当前 Elasticsearch 节点数量时，才可以使用日志记录 playbook 重新定义 Elasticsearch 集群大小。

36.5.2.1.5. 更改 Elasticsearch 副本的数量

您可以通过编辑清单主机文件中的 **openshift_logging_es_number_of_replicas** 值来更改 Elasticsearch 副本数量，并重新运行日志 playbook，如前面所述。

更改只适用于新索引。现有索引继续使用以前的副本数。例如，如果您将索引数量从 3 改为 2，您的集群会将 2 个副本用于新索引，并为现有索引使用 3 个副本。

您可以运行以下命令来修改现有索引的副本数：

```
$ oc exec -c elasticsearch $pod -- es_util --query=project.* -d '{"index":{"number_of_replicas":"2"}}'
```
1

1 指定现有索引的副本数。

36.5.2.1.6. 将 Elasticsearch 公开为路由

默认情况下，无法从日志记录集群外部访问部署了 OpenShift 聚合日志的 Elasticsearch。您可以为要访问 Elasticsearch 的工具启用对 Elasticsearch 的路由。

您可以使用 OpenShift 令牌访问 Elasticsearch，您可以在创建服务器证书时提供外部 Elasticsearch 和 Elasticsearch Ops 主机名（与 Kibana 相似）。

1. 要将 Elasticsearch 作为重新加密路由访问，请定义以下变量：

```
openshift_logging_es_allow_external=True
openshift_logging_es_hostname=elasticsearch.example.com
```

2. 进入 playbook 目录并运行以下 Ansible playbook：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i </path/to/inventory>] \
    playbooks/openshift-logging/config.yml
```

3. 要远程登录到 Elasticsearch，请求必须包含三个 HTTP 标头：

```
Authorization: Bearer $token
X-Proxy-Remote-User: $username
X-Forwarded-For: $ip_address
```

4. 您必须具有项目的访问权限，以便能访问其日志。例如：

```
$ oc login <user1>
$ oc new-project <user1project>
$ oc new-app <httpd-example>
```

5. 您需要获取这个 ServiceAccount 的令牌，以便在请求中使用：

```
$ token=$(oc whoami -t)
```

6. 使用之前配置的令牌，您应能够通过公开的路由访问 Elasticsearch：

```
$ curl -k -H "Authorization: Bearer $token" -H "X-Proxy-Remote-User: $(oc whoami)" -H "X-Forwarded-For: 127.0.0.1" https://es.example.test/project.my-project.*/_search?q=level:err | python -mjson.tool
```

36.5.3. fluentd

Fluentd 部署为 DaemonSet，它根据节点选择器部署节点，您可以使用 inventory 参数 **openshift_logging_fluentd_nodeselector** 指定，默认为 **logging-infra-fluentd**。作为 OpenShift 集群安装的一部分，建议您将 Fluentd 节点选择器添加到持久的 节点标签 列表中。

Fluentd 使用 **journald** 作为系统日志源。这些是来自操作系统、容器运行时和 OpenShift 的日志消息。

可用的容器运行时提供少许信息来标识日志消息的来源。在 pod 被删除后，日志收集和日志的规范化可能无法从 API 服务器检索其他元数据，如标签或注解。

如果在日志收集器完成处理日志前删除了具有指定名称和命名空间的 pod，则可能无法将日志消息与类似命名的 pod 和命名空间区分开来。这可能导致日志被索引，并将其注解为不是由部署 pod 的用户拥有的索引。

重要

可用的容器运行时提供少许信息来标识日志消息来源，无法确保唯一的个别日志消息，也不能保证可以追溯这些消息的来源。

OpenShift Container Platform 3.9 或更高版本的清理安装使用 **json-file** 作为默认日志驱动程序，但从 OpenShift Container Platform 3.7 升级的环境将维护其现有的 **journald** 日志驱动程序配置。建议您使用 **json-file** 日志驱动程序。如需了解将现有日志驱动程序配置改为 **json-file** 的步骤，请参阅更改 Aggregated Logging 驱动程序。

查看 Fluentd 日志

如何查看日志取决于 **LOGGING_FILE_PATH** 设置。

- 如果 **LOGGING_FILE_PATH** 指向一个文件，请使用 **logs** 程序打印 Fluentd 日志文件的内容：

```
oc exec <pod> -- logs 1
```

1 指定 Fluentd Pod 的名称。注意 **logs** 前面的空格。

例如：

```
oc exec logging-fluentd-lmvms -- logs
```

从最旧的日志开始，将打印出日志文件的内容。使用 **-f** 选项跟踪正在写入日志中的内容。

- 如果使用 **LOGGING_FILE_PATH=console**，Fluentd 会将日志记录到其默认位置 **/var/log/fluentd/fluentd.log**。您可以使用 **oc logs -f <pod_name>** 命令来检索日志。
例如：

```
oc logs -f fluentd.log
```

配置 Fluentd 日志位置

Fluentd 根据 **LOGGING_FILE_PATH** 环境变量，将日志写入指定的文件，默认为 **/var/log/fluentd/fluentd.log** 或控制台。

要更改 Fluentd 日志的默认输出位置，请使用默认清单文件中的 **LOGGING_FILE_PATH** 参数。您可以指定特定文件或使用 Fluentd 默认位置：

```
LOGGING_FILE_PATH=console (1)
LOGGING_FILE_PATH=<path-to-log/fluentd.log> (2)
```

1. 将日志输出发送到 Fluentd 默认位置。使用 **oc logs -f <pod_name>** 命令来检索日志。
2. 将日志输出发送到指定的文件。使用 **oc exec <pod_name> — logs** 命令获取日志数据。

更改这些参数后，重新运行 logging installer playbook ：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i </path/to/inventory>] \
    playbooks/openshift-logging/config.yml
```

配置 Fluentd 日志轮转

当当前的 Fluentd 日志文件达到指定大小时，OpenShift Container Platform 会自动重命名 **fluentd.log** 日志文件，以便收集新的日志数据。日志轮转会被默认启用。

以下示例显示了在一个集群中，最大日志大小为 1Mb，应保留四个日志。当 **fluentd.log** 到达 1Mb 时，OpenShift Container Platform 会删除当前的 **fluentd.log.4**，依次重命名每个 Fluentd 日志，并创建一个新的 **fluentd.log**。

```
fluentd.log    0b
fluentd.log.1  1Mb
fluentd.log.2  1Mb
fluentd.log.3  1Mb
fluentd.log.4  1Mb
```

您可以控制 Fluentd 日志文件的大小，以及 OpenShift Container Platform 使用环境变量保留的重命名文件的数量。

表 36.1. 用于配置 Fluentd 日志轮转的参数

参数	描述
LOGGING_FILE_SIZE	Bytes 中单个 Fluentd 日志文件的最大大小。如果 **flientd.log** 文件的大小超过这个值，OpenShift Container Platform 会重命名 **fluentd.log.*** 文件，并创建一个新的 **fluentd.log**。默认值为 1024000 (1MB)。
LOGGING_FILE_AGE	Fluentd 在删除前保留的日志数量。默认值为 **10**。

例如：

```
$ oc set env ds/logging-fluentd LOGGING_FILE_AGE=30 LOGGING_FILE_SIZE=1024000"
```

通过设置 **LOGGING_FILE_PATH=console** 来关闭日志轮转。这会导致 Fluentd 写入 Fluentd 默认位置 */var/log/fluentd/fluentd.log*，您可以在其中使用 **oc logs -f <pod_name>** 命令检索它们。

```
$ oc set env ds/fluentd LOGGING_FILE_PATH=console
```

使用 MERGE_JSON_LOG 禁用日志的 JSON 解析

默认情况下，Fluentd 会决定日志消息是否为 **JSON** 格式，并将消息合并到发布至 Elasticsearch 的 JSON 有效负载文档中。

使用 JSON 解析时，您可能会遇到：

- 由于类型映射不一致，Elasticsearch 会拒绝文档从而可能导致日志丢失。
- 拒绝消息循环导致的缓冲存储泄漏；
- 使用相同名称的字段覆盖数据。

有关如何缓解这些问题的信息，请参阅配置日志收集器规范日志的方式。

您可以禁用 JSON 解析以避免这些问题，或者不需要从日志解析 JSON。

禁用 JSON 解析：

1. 运行以下命令：

```
oc set env ds/logging-fluentd MERGE_JSON_LOG=false 1
```

1 把此项设为 **false** 会禁用此功能，设为 **true** 则启用此功能。

要确保此设置在每次运行 Ansible 时应用，请将 **openshift_logging_fluentd_merge_json_log="false"** 添加到您的 Ansible 清单中。

配置日志收集器规范日志的方式

集群日志记录使用特定的数据模型（例如数据库架构），将日志记录及其元数据存储在日志记录存储中。数据有一些限制：

- 必须有一个含有实际日志消息的 **"message"** 字段。
- 必须有一个包含 RFC 3339 格式的日志记录时间戳的 **"@timestamp"** 字段，最好是毫秒或更细的解析度。
- 必须有一个注明日志级别的 **"level"** 字段，如 **err**、**info** 和 **unknown** 等。

注意

如需有关数据模型的更多信息，请参阅导出字段。

由于这些要求的原因，从不同子系统收集的日志数据可能会发生冲突和不一致。

例如，如果您使用 **MERGE_JSON_LOG** 功能 (**MERGE_JSON_LOG=true**)，那么让您的应用程序以 JSON 格式记录其输出并让日志收集器自动在 Elasticsearch 中解析和索引数据会特别有用。不过，这会导致几个问题，具体包括：

- 字段名称可能为空，或包含 Elasticsearch 中非法的字符；
- 同一命名空间中的不同应用程序可能会输出具有不同值数据类型的相同字段名称；
- 应用程序可能会发出太多字段；
- 字段可能与集群日志记录内置字段冲突。

您可以编辑 Fluentd 日志收集器 DaemonSet 并设置下表中的环境变量，来配置集群日志记录如何对待来自不同来源的字段。

- **未定义字段。** ViaQ 数据模型未知的字段称为 *undefined*。来自不同系统的日志数据可以包含未定义字段。数据模型需要定义和描述所有顶级字段。
  使用参数来配置 OpenShift Container Platform 如何在名为 **undefined** 的顶级字段下移动任何未定义字段，以避免与*众所周知的*顶级字段冲突。您可以将未定义字段添加到顶级字段，并将其他字段移动到 **undefined** 容器中。

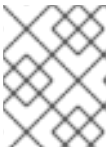
  您还可以替换未定义字段中的特殊字符，并将未定义字段转换为其 JSON 字符串表示形式。转换为 JSON 字符串可保留值的结构，以便您日后可以检索值并将其重新转换为映射或数组。

  - 简单的标量值（如数字和布尔值）将被变为带引号的字符串。例如：**10** 变为 **"10"**, **3.1415** 变为 **"3.1415"**, **false** 变为 **"false"**。

  - 映射/字典值和数组值将转换为对应的 JSON 字符串表示形式：**"mapfield":{"key":"value"}** 变成 **"mapfield":"{\"key\":\"value\"}"**，**"arrayfield":[1,2,"three"]** 则变成 **"arrayfield":"[1,2,\"three\"]"**。

- **已定义字段。** 已定义字段会出现在日志的顶级中。您可以配置哪些字段被视为已定义字段。
  默认的顶级字段通过 **CDM_DEFAULT_KEEP_FIELDS** 参数定义，它们包括 **CEE**、**time**、**@timestamp**、**aushape**、**ci_job**、**collectd**、**docker**、**fedora-ci**、**file**、**foreman**、**geoip**、**hostname**、**ipaddr4**、**ipaddr6**、**kubernetes**、**level**、**message**、**namespace_name**、**namespace_uuid**、**offset**、**openstack**、**ovirt**、**pid**、**pipeline_metadata**、**service**、**systemd**、**tags**、**testcase**、**tlog** 和 **viaq_msg_id**。

  如果 **CDM_USE_UNDEFINED** 为 **true**，所有未包含在 **${CDM_DEFAULT_KEEP_FIELDS}** 或 **${CDM_EXTRA_KEEP_FIELDS}** 中的字段将移到 **${CDM_UNDEFINED_NAME}**。有关这些参数的详情，请参考下表。

  注意

  CDM_DEFAULT_KEEP_FIELDS 参数仅供高级用户使用，或在红帽支持的指导下使用。

- **空白字段。** 空白字段没有数据。您可以确定要从日志中保留的空白字段。

表 36.2. 用于日志规范化的环境参数

参数	定义	示例
CDM_EXTRA_KEEP_FIELDS	除了 **CDM_DEFAULT_KEEP_FIELDS** 之外，还指定将一组额外的已定义字段保留在日志的顶级。默认值为 ""。	**CDM_EXTRA_KEEP_FIELDS="broker"**
CDM_KEEP_EMPTY_FIELDS	即使为空，也要以 CSV 格式指定要保留的字段。未指定的空白已定义字段将被丢弃。默认值为 "message"，即保留空消息。	**CDM_KEEP_EMPTY_FIELDS="message"**
CDM_USE_UNDEFINED	设为 **true** 可将未定义字段移到 **undefined** 顶级字段。默认值为 **false**。若为 **true**，则 **CDM_DEFAULT_KEEP_FIELDS** 和 **CDM_EXTRA_KEEP_FIELDS** 中的值不会移到 **undefined**。	**CDM_USE_UNDEFINED=true**

参数	定义	示例
CDM_UNDEFINED_NAME	如果使用 **CDM_USE_UNDEFINED**，则为未定义的顶级字段指定名称。默认值为 `undefined`。仅在 **CDM_USE_UNDEFINED** 为 **true** 时启用。	**CDM_UNDEFINED_NAME="undef"**
CDM_UNDEFINED_MAX_NUM_FIELDS	如果未定义字段的数量大于此数，则所有未定义字段都转换为其 JSON 字符串表示形式，并存储在 **CDM_UNDEFINED_NAME** 字段中。如果记录包含的未定义字段数大于此值，则不会对这些字段作进一步处理。相反，这些字段将转换为单个字符串 JSON 值，存储在顶级 **CDM_UNDEFINED_NAME** 字段中。保留默认值 **-1** 时允许无限数量的未定义字段，但建议不要这样做。 **注：**即使 **CDM_USE_UNDEFINED** 为 false，也遵从此参数。	**CDM_UNDEFINED_MAX_NUM_FIELDS=4**
CDM_UNDEFINED_TO_STRING	设为 **true** 时可将所有未定义字段转换为其 JSON 字符串表示形式。默认值为 **false**。	**CDM_UNDEFINED_TO_STRING=true**
CDM_UNDEFINED_DOT_REPLACE_CHAR	指定要在未定义字段中代替句点字符"."的字符。**MERGE_JSON_LOG** 必须为 **true**。默认值为 **UNUSED**。如果将 **MERGE_JSON_LOG** 参数设置为 **true**，请参见下面的注释部分。	**CDM_UNDEFINED_DOT_REPLACE_CHAR="_"**

注意

如果将 Fluentd 日志收集器 DaemonSet 中的 **MERGE_JSON_LOG** 参数和 **CDM_UNDEFINED_TO_STRING** 环境变量设置为 true，您可能会收到 Elasticsearch 400错误。当 **MERGE_JSON_LOG=true** 时，日志收集器会添加数据类型为字符串以外的字段。如果设置 **CDM_UNDEFINED_TO_STRING=true**，日志收集器会尝试将这些字段作为字符串值添加，从而导致 Elasticsearch 400 错误。日志收集器翻转下一日的日志的索引时会清除此错误。

当日志收集器翻转索引时，它将创建一个全新的索引。字段定义会更新，您也不会收到 400 错误。如需更多信息，请参阅设置 MERGE_JSON_LOG 和 CDM_UNDEFINED_TO_STRING。

要配置未定义和空字段处理，请编辑 **logging-fluentd** daemonset：

1. 根据需要配置如何处理字段：
   a. 使用 **CDM_EXTRA_KEEP_FIELDS** 指定要移动的字段。
   b. 以 CSV 格式在 **CDM_KEEP_EMPTY_FIELDS** 参数中指定要保留的空字段。
2. 根据需要配置如何处理未定义字段：
   a. 将 **CDM_USE_UNDEFINED** 设置为 **true** 可将未定义字段移到顶级 **undefined** 字段：
   b. 使用 **CDM_UNDEFINED_NAME** 参数为未定义字段指定名称。

c. 将 **CDM_UNDEFINED_MAX_NUM_FIELDS** 设置为默认值 **-1** 以外的值，以设置单个记录中未定义字段数的上限。

3. 指定 **CDM_UNDEFINED_DOT_REPLACE_CHAR**，将未定义字段名称中的句点字符 **.** 更改为其他字符。例如，如果 **CDM_UNDEFINED_DOT_REPLACE_CHAR=@@@** 并且有一个名为 **foo.bar.baz** 的字段，该字段会转变为 **foo@@@bar@@@baz**。

4. 将 **UNDEFINED_TO_STRING** 设置为 **true** 可将未定义字段转换为其 JSON 字符串表示形式。

注意

如果配置 **CDM_UNDEFINED_TO_STRING** 或 **CDM_UNDEFINED_MAX_NUM_FIELDS** 参数，您可以使用 **CDM_UNDEFINED_NAME** 更改未定义字段名称。此字段是必需的，因为 **CDM_UNDEFINED_TO_STRING** 或**CDM_UNDEFINED_MAX_NUM_FIELDS** 可能会更改未定义字段的值类型。当 **CDM_UNDEFINED_TO_STRING** 或 **CDM_UNDEFINED_MAX_NUM_FIELDS** 设为 true 并且日志中还有更多未定义字段时，值类型将变为 **string**。如果值类型改变（例如，从 JSON 变为 JSON 字符串），Elasticsearch 将停止接受记录。

例如，当 **CDM_UNDEFINED_TO_STRING** 为 **false** 或者 **CDM_UNDEFINED_MAX_NUM_FIELDS** 是默认值 **-1** 时，未定义字段的值类型将为 **json**。如果将 **CDM_UNDEFINED_MAX_NUM_FIELDS** 更改为默认值以外的值，且日志中还有更多未定义字段，则值类型将变为 **string** （JSON 字符串）。如果值类型改变，Elasticsearch 将停止接受记录。

设置 MERGE_JSON_LOG 和 CDM_UNDEFINED_TO_STRING

如果将 **MERGE_JSON_LOG** 和 **CDM_UNDEFINED_TO_STRING** 环境变量设置为 **true**，您可能会收到 Elasticsearch 400 错误。当 **MERGE_JSON_LOG=true** 时，日志收集器会添加数据类型为字符串以外的字段。如果设置了 **CDM_UNDEFINED_TO_STRING=true**，Fluentd 会尝试将这些字段作为**字符串**值来添加，从而导致 Elasticsearch 400 错误。下一日翻转索引时会清除此错误。

当 Fluentd 针对下一日的日志翻转索引时，它将创建一个全新的索引。字段定义会更新，您也不会收到 400 错误。

无法重试具有 **hard** 错误的记录，如架构违规和数据损坏等。日志收集器发送这些记录以进行错误处理。如果在 Fluentd 中添加了一个 **<label @ERROR>** 部分 作为最后一个 **<label>**，您可以根据需要处理这些记录。

例如：

```
data:
  fluent.conf:

....

    <label @ERROR>
      <match **>
        @type file
        path /var/log/fluent/dlq
        time_slice_format %Y%m%d
        time_slice_wait 10m
        time_format %Y%m%dT%H%M%S%z
        compress gzip
      </match>
    </label>
```

此部分将错误记录写入 Elasticsearch 死信队列 (DLQ) 文件。如需有关文件输出的更多信息，请参阅 fluentd 文档。

然后，您可以编辑该文件来手动清理记录，编辑该文件以与 Elasticsearch /_**bulk index** API 搭配使用，并且使用 cURL 来添加这些记录。如需有关 Elasticsearch Bulk API 的更多信息，请参阅 Elasticsearch 文档。

加入多行 Docker 日志

您可以将 Fluentd 配置为从 Docker 日志部分片段中重建整个日志记录。通过此功能活跃，Fluentd 会读取多行 Docker 日志，重新创建它们，并将日志作为 Elasticsearch 中的记录存储在没有缺失数据的 Elasticsearch 中。

但是，因为这个功能可能会导致性能回归，因此这个功能会默认关闭，必须手动启用。

以下 Fluentd 环境变量配置集群日志记录以处理多行 Docker 日志：

参数	描述
USE_MULTILINE_JSON	设置为 **true**，在使用 **json-file** 日志驱动程序时处理多行 Docker 日志。此参数默认设置为 **false**。
USE_MULTILINE_JOURNAL	设置为 **true**，在使用 **journald** 日志驱动程序时处理多行 Docker 日志，Fluentd 会重新构建 docker 日志片段中的整个日志记录。此参数默认设置为 **false**。

您可以使用以下命令来决定正在使用的日志驱动程序：

```
$ docker info | grep -i log
```

以下是输出结果之一：

```
Logging Driver: json-file
```

```
Logging Driver: journald
```

打开多行 Docker 日志处理：

1. 使用以下命令启用多行 Docker 日志：
   - 对于 **json-file** 日志驱动程序：

     ```
     oc set env daemonset/logging-fluentd USE_MULTILINE_JSON=true
     ```

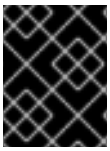
   - 对于 **journald** 日志驱动程序：

     ```
     oc set env daemonset/logging-fluentd USE_MULTILINE_JOURNAL=true
     ```

   集群中的 Fluentd Pod 重启。

配置 Fluentd 以将日志发送到外部日志聚合器

除了默认的 Elasticsearch 外，您还可以使用 **secure-forward** 插件，将 Fluentd 配置为将其日志的副本发送到外部日志聚合器。在本地托管的 Fluentd 处理日志记录之后，您可以从那里进一步处理日志记录。



重要

您不能使用客户端证书配置 **secure_foward** 插件。身份验证可以通过 SSL/TLS 协议运行，但要求使用 **secure_foward** 输入插件配置 **shared_key** 和目标 **Fluentd**。

日志记录部署操作在 Fluentd ConfigMap 中提供了一个 **secure-forward.conf** 部分，可用于配置外部聚合器：

```
<store>
@type secure_forward
self_hostname pod-${HOSTNAME}
shared_key thisisasharedkey
secure yes
enable_strict_verification yes
ca_cert_path /etc/fluent/keys/your_ca_cert
ca_private_key_path /etc/fluent/keys/your_private_key
ca_private_key_passphrase passphrase
<server>
  host ose1.example.com
  port 24284
</server>
<server>
  host ose2.example.com
  port 24284
  standby
</server>
<server>
  host ose3.example.com
  port 24284
  standby
</server>
</store>
```

这可以通过 **oc edit** 命令更新：

```
$ oc edit configmap/logging-fluentd
```

要在 **secure-forward.conf** 中使用的证书可以添加到 Fluentd Pod 上挂载的现有 secret 中。**your_ca_cert** 和 **your_private_key** 值必须与 **configmap/logging-fluentd** 中的 **secure-forward.conf** 中指定的值匹配：

```
$ oc patch secrets/logging-fluentd --type=json \
  --patch "[{'op':'add','path':'/data/your_ca_cert','value':'$(base64 -w 0 /path/to/your_ca_cert.pem)'}]"
$ oc patch secrets/logging-fluentd --type=json \
  --patch "[{'op':'add','path':'/data/your_private_key','value':'$(base64 -w 0
/path/to/your_private_key.pem)'}]"
```

注意

将 **your_private_key** 替换为一个通用名称。这个链接指向 JSON 路径，而不是主机系统上的路径。

配置外部聚合器时，它必须能够安全地接受来自 Fluentd 的消息。

如果外部聚合器是另一个 Fluentd 服务器，它必须安装 **fluent-plugin-secure-forward** 插件，并使用它提供的输入插件：

```
<source>
  @type secure_forward

  self_hostname ${HOSTNAME}
  bind 0.0.0.0
  port 24284

  shared_key thisisasharedkey

  secure yes
  cert_path       /path/for/certificate/cert.pem
  private_key_path /path/for/certificate/key.pem
  private_key_passphrase secret_foo_bar_baz
</source>
```

您可以参阅如何在 **fluent-plugin-secure-forward** repository 中设置**fluent-plugin-secure-forward** 插件。

将连接数量从 Fluentd 减小到 API 服务器

重要

MUX 只是一个技术预览功能。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

如需红帽技术预览功能支持范围的更多信息，请参阅 https://access.redhat.com/support/offerings/techpreview/。

MUX 是一个安全转发监听器服务。

参数	描述
openshift_logging_use_mux	默认值为 **False**。如果设置为 **True**，则会部署名为 **mux** 的服务。此服务充当集群中运行的节点代理 Fluentd daemonset 的 Fluentd **secure_forward** 聚合器。使用 **openshift_logging_use_mux** 来减少到 OpenShift API 服务器的连接数，并将 Fluentd 中的每个节点配置为发送原始日志到 **mux** 并关闭 Kubernetes 元数据插件。这需要使用 **openshift_logging_mux_client_mode**。

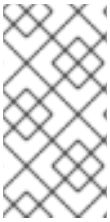
参数	描述
openshift_logging_mux_client_mode	**openshift_logging_mux_client_mode** 的值为 **minimal** 和 **maximal**，且没有默认值。**openshift_logging_mux_client_mode** 会导致 Fluentd 节点代理将日志发送到 mux 而不是直接发送到 Elasticsearch。值 **maximal** 表示，Fluentd 在将记录发送到 **mux** 前，可以在节点上进行尽可能多的处理。建议使用 **mux** 的 **maximal** 值。这个值 **最少** 表示 Fluentd 不会处理，并将原始日志发送到 **mux** 进行处理。不建议使用 **minimal** 值。
openshift_logging_mux_allow_external	默认值为 **False**。如果设置为 **True**，则 **mux** 服务已部署，并且被配置为允许集群外运行的 Fluentd 客户端使用 **secure_forward** 发送日志。这允许 OpenShift 日志记录用作 OpenShift 以外的客户端的中央日志记录服务，或其他 OpenShift 集群。
openshift_logging_mux_hostname	默认为 **mux** 加上 **openshift_master_default_subdomain**。这是主机名 **external_clients** 将用于连接到 **mux**，并用于 TLS 服务器证书主题。
openshift_logging_mux_port	24284
openshift_logging_mux_cpu_limit	500M
openshift_logging_mux_memory_limit	2Gi
openshift_logging_mux_default_namespaces	默认值为 **mux-undefined**。列表中的第一个值是用于未定义项目的 namespace，后面是默认创建的任何额外命名空间。通常，您不需要设置这个值。
openshift_logging_mux_namespaces	默认值为空，允许为外部 **mux** 客户端创建额外命名空间，以与其日志关联。您需要设置此值。

Fluentd 中的节流日志

对于特别冗长的项目，管理员可以减慢 Fluentd 中读取日志的速度，然后再处理。

警告

节流可能会导致日志聚合落后于配置的项目；如果在 Fluentd 赶上之前删除了 Pod，则日志条目可能会丢失。



注意

使用 systemd 系统日志作为日志源时，节流不起作用。节流的实施取决于各个项目中个别日志文件是否能够减慢读取速度。从系统日志中读取时，只有一个日志源，而没有日志文件，因此无法使用基于文件的节流。没有办法可以限制读取到 Fluentd 进程中的日志条目。

要告知 Fluentd 应该限制的项目，在部署后编辑 ConfigMap 中的节流配置：

```
$ oc edit configmap/logging-fluentd
```

throttle-config.yaml 键的格式是 YAML 文件，其包含项目名称以及各个节点上希望读取日志的速度。默认值为每个节点一次读取 1000 行。例如：

- 项目

```
project-name:
  read_lines_limit: 50

second-project-name:
  read_lines_limit: 100
```

- 日志记录

```
logging:
  read_lines_limit: 500

test-project:
  read_lines_limit: 10

.operations:
  read_lines_limit: 100
```

要更改 Fluentd，请更改配置并重启 Fluentd Pod 以应用更改。要对 Elasticsearch 进行更改，必须首先缩减 Fluentd，然后再缩减 Elasticsearch 为零。进行更改后，首先扩展 Elasticsearch，然后再将 Fluentd 扩展至其原始设置。

将 Elasticsearch 扩展为零：

```
$ oc scale --replicas=0 dc/<ELASTICSEARCH_DC>
```

更改 daemonset 配置中的 nodeSelector 以匹配零：

获取 Fluentd 节点选择器：

```
$ oc get ds logging-fluentd -o yaml |grep -A 1 Selector
    nodeSelector:
      logging-infra-fluentd: "true"
```

使用 oc patch 命令修改 daemonset nodeSelector：

```
$ oc patch ds logging-fluentd -p '{"spec":{"template":{"spec":{"nodeSelector":
{"nonexistlabel":"true"}}}}}'
```

获取 Fluentd 节点选择器：

```
$ oc get ds logging-fluentd -o yaml |grep -A 1 Selector
    nodeSelector:
      "nonexistlabel: "true"
```

从零扩展 Elasticsearch：

```
$ oc scale --replicas=# dc/<ELASTICSEARCH_DC>
```

将 daemonset 配置中的 nodeSelector 改为 logging-infra-fluentd: "true"。

使用 **oc patch** 命令修改 daemonset nodeSelector：

```
oc patch ds logging-fluentd -p '{"spec":{"template":{"spec":{"nodeSelector":{"logging-infra-fluentd":"true"}}}}}'
```

微调缓冲块限制

如果 Fluentd 日志记录器无法满足大量日志的需求，则需要切换到文件缓冲来降低内存用量并防止数据丢失。

Fluentd **buffer_chunk_limit** 由环境变量 **BUFFER_SIZE_LIMIT** 决定，其默认值为 **8m**。每个输出的文件缓冲区大小由环境变量 **FILE_BUFFER_LIMIT** 决定，其默认值为 **256Mi**。持久性卷大小必须大于 **FILE_BUFFER_LIMIT** 与输出相乘的结果。

在 Fluentd 和 Mux pod 上，持久性卷 **/var/lib/fluentd** 应该通过 PVC 或 hostmount 进行准备，例如：然后，将该区域用作文件缓冲区。

buffer_type 和 **buffer_path** 在 Fluentd 配置文件中进行配置，如下所示：

```
$ egrep "buffer_type|buffer_path" *.conf
output-es-config.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-output-es-config`
output-es-ops-config.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-output-es-ops-config`
filter-pre-mux-client.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-mux-client`
```

Fluentd **buffer_queue_limit** 是变量 **BUFFER_QUEUE_LIMIT** 的值。默认值为 **32**。

环境变量 **BUFFER_QUEUE_LIMIT** 计算为 **(FILE_BUFFER_LIMIT / (number_of_outputs * BUFFER_SIZE_LIMIT))**。

如果 **BUFFER_QUEUE_LIMIT** 变量具有默认值：

- **FILE_BUFFER_LIMIT = 256Mi**
- **number_of_outputs = 1**
- **BUFFER_SIZE_LIMIT = 8Mi**

buffer_queue_limit 的值为 **32**。要更改 **buffer_queue_limit**，您需要更改 **FILE_BUFFER_LIMIT** 的值。

在这个公式中，如果所有日志都发送到单个资源，则 **number_of_outputs** 为 **1**，否则每多一个资源就会递增 **1**。例如，**number_of_outputs** 的值为：

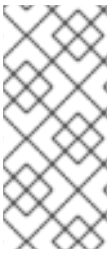
- **1** - 如果所有日志都发送到单个 ElasticSearch pod
- **2** - 如果应用程序日志发送到 ElasticSearch pod，并且 ops 日志发送到另一个 ElasticSearch pod
- **4** - 如果应用程序日志发送到一个 ElasticSearch pod，ops 日志发送到另一个 ElasticSearch pod，并且这两者都转发到其他 Fluentd 实例

36.5.4. Kibana

要从 OpenShift Container Platform web 控制台访问 Kibana 控制台，请使用 Kibana 控制台（**kibana-hostname** 参数）在 master webconsole-config configmap 文件中添加 **loggingPublicURL** 参数。该值必须是 HTTPS URL：

```
...
clusterInfo:
  ...
  loggingPublicURL: "https://kibana.example.com"
...
```

在 OpenShift Container Platform Web 控制台中设置 **loggingPublicURL** 参数会在 **Browse → Pods → <pod_name> → Logs** 选项卡中创建一个 **View Archive** 按钮。这会链接到 Kibana 控制台。



注意

当有效的登录 Cookie 过期时，您需要登录到 Kibana 控制台，例如：您需要登录：

- 首次使用
- 注销后

您可以像冗余方式扩展 Kibana 部署：

```
$ oc scale dc/logging-kibana --replicas=2
```

注意

为确保扩展在日志记录 playbook 的多个执行之间保留，请确保更新清单文件中的 **openshift_logging_kibana_replica_count**。

您可以通过访问 **openshift_logging_kibana_hostname** 变量指定的站点来查看用户界面。

有关 Kibana 的更多信息，请参阅 Kibana 文档。

Kibana Visualize

通过 Kibana Visualize，您可以创建用于监控容器和 pod 日志的视觉化和仪表板，管理员用户（**cluster-admin** 或 **cluster-reader**）可以按部署、命名空间、Pod 和容器来查看日志。

Kibana Visualize 存在于 Elasticsearch 和 ES-OPS pod 中，且必须在这些 pod 中运行。要加载仪表板和其他 Kibana UI 对象，您必须首先以您要添加仪表板的用户身份登录到 Kibana，然后注销。这将创建下一步所依赖的每个用户所需的配置。然后运行：

```
$ oc exec <$espod> -- es_load_kibana_ui_objects <user-name>
```

其中 **$espod** 是 Elasticsearch Pod 之一的名称。

在 Kibana Visualize 中添加自定义字段

如果您的 OpenShift Container Platform 集群以 JSON 格式生成日志，其中包含 Elasticsearch **.operations.*** 或 **项目.*** 索引中定义的自定义字段，则无法使用这些字段创建视觉化，因为自定义字段在 Kibana 中不可用。

但是，您可以将自定义字段添加到 Elasticsearch 索引中，该索引允许您将字段添加到 Kibana 索引模式中，以便在 Kibana Visualize 中使用。

注意

自定义字段仅应用于模板更新后创建的索引。

将自定义字段添加到 Kibana Visualize 中：

1. 在 Elasticsearch 索引模板中添加自定义字段：

   a. 确定要添加哪些 Elasticsearch 索引，可以是 **.operations.*** 或 **project.*** 索引。如果有具有自定义字段的特定项目，您可以将字段添加到项目的特定索引中，例如： **project.this-project-has-time-fields.***。

   b. 为自定义字段创建一个 JSON 文件，如下所示：
   例如：

```
{
 "order": 20,
 "mappings": {
  "_default_": {
   "properties": {
    "mytimefield1": { 1
     "doc_values": true,
     "format": "yyyy-MM-dd HH:mm:ss,SSSZ||yyyy-MM-dd'T'HH:mm:ss.SSSSSSZ||yyyy-
MM-dd'T'HH:mm:ssZ||dateOptionalTime",
     "index": "not_analyzed",
     "type": "date"
    },
    "mytimefield2": {
     "doc_values": true,
     "format": "yyyy-MM-dd HH:mm:ss,SSSZ||yyyy-MM-dd'T'HH:mm:ss.SSSSSSZ||yyyy-
MM-dd'T'HH:mm:ssZ||dateOptionalTime",
     "index": "not_analyzed",
     "type": "date"
    }
   }
  }
 }
```

```
 },
 "template": "project.<project-name>.*" 2
}
```

1 添加自定义字段和参数。

2 指定 **.operations.*** 或 **project.*** 索引。

c. 进入 **openshift-logging** 项目：

```
$ oc project openshift-logging
```

d. 获取一个 Elasticsearch Pod 的名称：

```
$ oc get -n logging pods -l component=es

NAME                                  READY   STATUS    RESTARTS   AGE     IP
NODE                    NOMINATED NODE
logging-es-data-master-5av030lk-1-2x494   2/2     Running   0          38m
154.128.0.80   ip-153-12-8-6.wef.internal   <none>
```

e. 将 JSON 文件加载到 Elasticsearch pod 中：

```
$ cat <json-file-name> | \ 1
oc exec -n logging -i -c elasticsearch <es-pod-name> -- \ 2
   curl -s -k --cert /etc/elasticsearch/secret/admin-cert \
   --key /etc/elasticsearch/secret/admin-key \
   https://localhost:9200/_template/<json-file-name> -XPUT -d@- | \ 3
python -mjson.tool
```

1 3 您创建的 JSON 文件的名称。

2 Elasticsearch Pod 的名称。

```
{
   "acknowledged": true
}
```

f. 如果您有单独的 OPS 集群，请获取其中一个 es-ops Elasticsearch pod 的名称：

```
$ oc get -n logging pods -l component=es-ops

NAME                                      READY   STATUS    RESTARTS   AGE     IP
NODE                    NOMINATED NODE
logging-es-ops-data-master-o7nhcbo4-5-b7stm   2/2     Running   0          38m
154.128.0.80   ip-153-12-8-6.wef.internal   <none>
```

g. 将 JSON 文件加载到 es-ops Elasticsearch pod 中：

```
$ cat <json-file-name> | \ 1
oc exec -n logging -i -c elasticsearch <esops-pod-name> -- \ 2
```

```
    curl -s -k --cert /etc/elasticsearch/secret/admin-cert \
    --key /etc/elasticsearch/secret/admin-key \
    https://localhost:9200/_template/<json-file-name> -XPUT -d@- | \ 3
python -mjson.tool
```

1 3 您创建的 JSON 文件的名称。

2 OPS 集群 Elasticsearch pod 的名称。

输出结果类似如下：

```
{
  "acknowledged": true
}
```

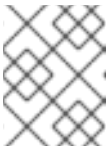
h. 验证索引是否已更新：

```
oc exec -n logging -i -c elasticsearch <es-pod-name> -- \ 1
    curl -s -k --cert /etc/elasticsearch/secret/admin-cert \
    --key /etc/elasticsearch/secret/admin-key \
    https://localhost:9200/project.*/_search?sort=<custom-field>:desc | \ 2
python -mjson.tool
```

1 Elasticsearch 或 OPS 集群 Elasticsearch pod 的名称。

2 您添加的自定义字段的名称。

命令输出自定义字段的索引记录，按降序排列。



注意

设置不适用于现有的索引。如果要将设置应用到现有索引中，请执行 re-index。

2. 将自定义字段添加到 Kibana：

a. 从 Elasticsearch 容器获取现有的索引模式文件：

```
$ mkdir index_patterns
$ cd index_patterns
$ oc project openshift-logging
$ for espod in $( oc get pods -l component=es -o jsonpath='{.items[*].metadata.name}' ) ;
do
>  for ff in $( oc exec -c elasticsearch <es-pod-name> -- ls
/usr/share/elasticsearch/index_patterns ) ; do
>    oc exec -c elasticsearch <es-pod-name> -- cat
/usr/share/elasticsearch/index_patterns/$ff > $ff
>  done
>  break
> done
```

索引模式文件下载到 */usr/share/elasticsearch/index_patterns* 目录中。

例如：

```
index_patterns $ ls

com.redhat.viaq-openshift.index-pattern.json
```

b. 编辑对应的索引模式文件，将每个自定义字段的定义添加到**字段**值中：
例如：

```
{\"count\": 0, \"name\": \"mytimefield2\", \"searchable\": true, \"aggregatable\": true,
\"readFromDocValues\": true, \"type\": \"date\", \"scripted\": false},
```

该定义必须包含 **\"searchable\": true,** 和 **\"aggregatable\": true,** 才能进行视觉化使用。数据类型必须与上方添加的 Elasticsearch 字段定义对应。例如，如果您在 Elasticsearch 中添加了一个 **myfield** 字段，它的类型为 **number**，则无法将 **myfield** 添加到 Kibana 中作为**字符串**类型。

c. 在索引模式文件中，将 Kibana 索引模式的名称添加到索引模式文件中：
例如，使用 **operations.*** 索引模式：

```
"title": "*operations.*"
```

要使用 **project.MYNAMESPACE.*** 索引模式：

```
"title": "project.MYNAMESPACE.*"
```

d. 识别用户名并获取用户名的 hash 值。索引模式使用用户名的 hash 来存储。按顺序运行以下两个命令：

```
$ get_hash() {
>     printf "%s" "$1" | sha1sum | awk '{print $1}'
> }
```

```
$ get_hash admin

d0aeb5660fc2140aec35850c4da997
```

e. 将索引模式文件应用到 Elasticsearch：

```
cat com.redhat.viaq-openshift.index-pattern.json | \ 1
  oc exec -i -c elasticsearch <espod-name> -- es_util \
    --query=".kibana.<user-hash>/index-pattern/<index>" -XPUT --data-binary @- | \ 2
  python -mjson.tool
```

1 索引模式文件的名称。

2 用户散列和索引，可以是 **.operations.*** 或 **project.***。

例如：

```
cat index-pattern.json | \
  oc exec -i -c elasticsearch mypod-23-gb9pl -- es_util \
```

```
    --query=".kibana.d0aeb5660fc2140aec35850c4da997/index-
pattern/project.MYNAMESPACE.*" -XPUT --data-binary @- | \
  python -mjson.tool
```

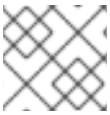
输出结果类似如下：

```
{
    "_id": ".operations.*",
    "_index": ".kibana.d0aeb5660fc2140aec35850c4da997",
    "_shards": {
        "failed": 0,
        "successful": 2,
        "total": 2
    },
    "_type": "index-pattern",
    "_version": 1,
    "created": true,
    "result": "created"
}
```

f. 退出并重启自定义字段的 Kibana 控制台，使其显示在 **Available Fields** 列表中，并在 **Management → Index Patterns** 页面上的字段列表中显示。

36.5.5. Curator

借助 Curator，管理员可以配置调度的 Elasticsearch 维护操作，从而基于每个项目自动执行。它被调度为每天根据其配置执行操作。建议每个 Elasticsearch 集群仅使用一个 Curator Pod。Curator Pod 仅在 cronjob 中声明的时间运行，然后 Pod 在完成后终止。Curator 通过 YAML 配置文件配置，结构如下：



注意

时区是根据运行 curator Pod 的主机节点设置的。

```
$PROJECT_NAME:
  $ACTION:
    $UNIT: $VALUE

$PROJECT_NAME:
  $ACTION:
    $UNIT: $VALUE
  ...
```

可用的参数如下：

变量名称	描述
PROJECT_NAME	项目的实际名称，例如 **myapp-devel**。对于 OpenShift Container Platform **operations** 日志，请使用 **.operations** 作为项目名称。
操作	当前只支持 **delete**。
UNIT	一整 **天**、**周** 或 **几个月**。

变量名称	描述
VALUE	单元数的整数。
.defaults	使用 **.defaults** 作为 **$PROJECT_NAME**，以设置未指定的项目的默认值。
.regex	与项目名称匹配的正则表达式列表。
pattern	有效且正确转义的正则表达式，用单引号括起。

例如，要将 Curator 配置为：

- 删除 **myapp-dev** 项目中存在时间超过 **1 天**的索引
- 删除 **myapp-qe** 项目中存在时间超过 **1 个星期**的索引
- 删除存在时间超过 **8 个星期**的**operations**日志
- 删除所有其他项目中存在时间超过 **31 天**的索引
- 删除与 '^project\..+\-dev.*$' regex 匹配的，早于 1 天的索引
- 删除与 '^project\..+\-test.*$' regex 匹配的，早于 2 天的索引

使用：

```
config.yaml: |
  myapp-dev:
    delete:
      days: 1

  myapp-qe:
    delete:
      weeks: 1

  .operations:
    delete:
      weeks: 8

  .defaults:
    delete:
      days: 31

  .regex:
    - pattern: '^project\..+\-dev\..*$'
      delete:
        days: 1
    - pattern: '^project\..+\-test\..*$'
      delete:
        days: 2
```

重要

当您将 **months** 用作操作的 **$UNIT** 时，Curator 会从当月的第一天开始计算，而不是当月的当天。例如，如果今天是 4 月 15 日，并且您想要删除今天超过 2 个月的索引（delete: months:2），Curator 不会删除日期超过 2 月 15 的索引，它会删除早于 2 月 1 的索引。也就是说，它会退回到当前月份的第一天，然后从该日期起返回两个整月。如果您想对 Curator 准确，最好使用 days（例如 **delete: days）：30**).

36.5.5.1. 使用 Curator Actions 文件

设置 OpenShift Container Platform 自定义配置文件格式可确保内部索引不会被错误地删除。

要使用 **操作文件**，请在 Curator 配置中添加 exclude 规则来保留这些索引。您必须手动添加所有所需模式。

```
actions.yaml: |
actions:

  action: delete_indices
  description: be careful!
  filters:
  - exclude: false
    kind: regex
    filtertype: pattern
    value: '^project\.myapp\..*$'
  - direction: older
    filtertype: age
    source: name
    timestring: '%Y.%m.%d'
    unit_count: 7
    unit: days
  options:
    continue_if_exception: false
    timeout_override: '300'
    ignore_empty_list: true

  action: delete_indices
  description: be careful!
  filters:
  - exclude: false
    kind: regex
    filtertype: pattern
    value: '^\.operations\..*$'
  - direction: older
    filtertype: age
    source: name
    timestring: '%Y.%m.%d'
    unit_count: 56
    unit: days
  options:
    continue_if_exception: false
    timeout_override: '300'
    ignore_empty_list: true

  action: delete_indices
  description: be careful!
```

```
filters:
- exclude: true
  kind: regex
  filtertype: pattern
  value: '^project\.myapp\..*$|^\.operations\..*$|^\.searchguard\..*$|^\.kibana$'
- direction: older
  filtertype: age
  source: name
  timestring: '%Y.%m.%d'
  unit_count: 30
  unit: days
options:
  continue_if_exception: false
  timeout_override: '300'
  ignore_empty_list: true
```

36.5.5.2. 创建 Curator 配置

openshift_logging Ansible 角色提供了一个 ConfigMap，Curator 从中读取其配置。您可以编辑或替换此 ConfigMap 来重新配置 Curator。目前，使用 **logging-curator** ConfigMap 来配置 ops 和 non-ops Curator 实例。任何 **.operations** 配置都与您的应用程序日志配置位于同一个位置。

1. 要创建 Curator 配置，请编辑部署的 ConfigMap 中的配置：

```
$ oc edit configmap/logging-curator
```

或者，从 cronjob 手动创建作业：

```
oc create job --from=cronjob/logging-curator <job_name>
```

- 对于脚本化部署，复制安装程序创建的配置文件并创建新的 OpenShift Container Platform 自定义配置：

```
$ oc extract configmap/logging-curator --keys=curator5.yaml,config.yaml --to=/my/config
  edit /my/config/curator5.yaml
  edit /my/config/config.yaml
$ oc delete configmap logging-curator ; sleep 1
$ oc create configmap logging-curator \
    --from-file=curator5.yaml=/my/config/curator5.yaml \
    --from-file=config.yaml=/my/config/config.yaml \
    ; sleep 1
```

- 或者，如果您使用 **操作文件** ：

```
$ oc extract configmap/logging-curator --keys=curator5.yaml,actions.yaml --
to=/my/config
  edit /my/config/curator5.yaml
  edit /my/config/actions.yaml
$ oc delete configmap logging-curator ; sleep 1
$ oc create configmap logging-curator \
    --from-file=curator5.yaml=/my/config/curator5.yaml \
    --from-file=actions.yaml=/my/config/actions.yaml \
    ; sleep 1
```

下一个调度的作业使用此配置。

您可以使用以下命令来控制 cronjob：

```
# suspend cronjob
oc patch cronjob logging-curator -p '{"spec":{"suspend":true}}'

# resume cronjob
oc patch cronjob logging-curator -p '{"spec":{"suspend":false}}

# change cronjob schedule
oc patch cronjob logging-curator -p '{"spec":{"schedule":"0 0 * * *"}}' 1
```

1 **schedule** 选项接受 cron 格式的调度。

36.6. CLEANUP

删除部署期间生成的所有内容。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i </path/to/inventory>] \
    playbooks/openshift-logging/config.yml \
    -e openshift_logging_install_logging=False
```

36.7. 将日志发送到外部 ELASTICSEARCH 实例

Fluentd 将日志发送到 Elasticsearch 部署配置的 **ES_HOST**、**ES_PORT**、**OPS_HOST** 和 **OPS_PORT** 环境变量的值。应用程序日志定向到 **ES_HOST** 目的地，操作日志则定向到 **OPS_HOST**。

注意

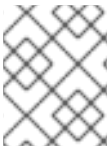
不支持将日志直接发送到 AWS Elasticsearch 实例。使用 Fluentd Secure Forward 将日志定向到您控制并使用 **fluent-plugin-aws-elasticsearch-service** 插件配置的 Fluentd 实例。

要将日志定向到特定的 Elasticsearch 实例，请编辑部署配置并将上述变量的值替换为所需的实例：

```
$ oc edit ds/<daemon_set>
```

对于包含应用程序和操作日志的外部 Elasticsearch 实例，您可以将 **ES_HOST** 和 **OPS_HOST** 设置为同一目的地，同时确保 **ES_PORT** 和 **OPS_PORT** 也具有相同的值。

仅支持 Mutual TLS 配置，因为提供的 Elasticsearch 实例也一样。使用客户端密钥、客户端证书和 CA 来修补或重新创建 **logging-fluentd** secret。



注意

如果不使用提供的 Kibana 和 Elasticsearch 镜像，您将没有同样的多租户功能，您的数据也不会由用户访问权限限制到特定的项目。

36.8. 将日志发送到外部 SYSLOG 服务器

在主机上使用 **fluent-plugin-remote-syslog** 插件，将日志发送到外部 syslog 服务器。

在 **logging-fluentd** 或 **logging-mux** daemonsets 中设置环境变量：

```
- name: REMOTE_SYSLOG_HOST 1
  value: host1
- name: REMOTE_SYSLOG_HOST_BACKUP
  value: host2
- name: REMOTE_SYSLOG_PORT_BACKUP
  value: 5555
```

1 所需的远程 syslog 主机。每个主机都需要。

这将建立两个目的地。**host1** 上的 syslog 服务器将在默认端口 **514** 上接收消息，**host2** 则在端口 **5555** 上接收相同的消息。

另外，您还可以在 **logging-fluentd** 或 **logging-mux** ConfigMap 中配置自己的自定义 ***fluent.conf***。

Fluentd 环境变量

参数	描述
USE_REMOTE_SYSLOG	默认值为 **false**。设置为 **true** 可启用**fluent-plugin-remote-syslog** gem
REMOTE_SYSLOG_HOST	（必需）远程 syslog 服务器的主机名或 IP 地址。
REMOTE_SYSLOG_PORT	要连接的端口号。默认值为 **514**。
REMOTE_SYSLOG_SEVERITY	设置 syslog 严重性级别。默认值为 **debug**。
REMOTE_SYSLOG_FACILITY	设置 syslog 工具。默认值为 **local0**。
REMOTE_SYSLOG_USE_RECORD	默认值为 **false**。设置为 **true** 可使用记录的严重性和工具字段对 syslog 消息进行设置。
REMOTE_SYSLOG_REMOVE_TAG_PREFIX	从标签中删除前缀，默认为 **''**（空白）。
REMOTE_SYSLOG_TAG_KEY	如果指定，则使用此字段作为要在记录上查看的键，以对 syslog 消息设置标签。
REMOTE_SYSLOG_PAYLOAD_KEY	如果指定，则使用此字段作为要在记录上查看的键，以对 syslog 消息设置有效负载。
REMOTE_SYSLOG_TYPE	设置传输层协议类型。默认为 **syslog_buffered**，它设定 TCP 协议。要切换到 UDP，请将其设定为 **syslog**。

警告

这种实施是不安全的，应当仅在能保证不嗅探连接的环境中使用。

Fluentd Logging Ansible 变量

参数	描述
openshift_logging_fluentd_remote_syslog	默认值为 **false**。设置为 **true** 以启用 fluent-plugin-remote-syslog gem。
openshift_logging_fluentd_remote_syslog_host	远程 syslog 服务器的主机名或 IP 地址，这是必需的。
openshift_logging_fluentd_remote_syslog_port	要连接的端口号，默认为 **514**。
openshift_logging_fluentd_remote_syslog_severity	设置 syslog 严重性级别，默认为 **debug**。
openshift_logging_fluentd_remote_syslog_facility	设置 syslog 工具，默认为 **local0**。
openshift_logging_fluentd_remote_syslog_use_record	默认值为 **false**。设置为 **true** 可使用记录的严重性和工具字段对 syslog 消息进行设置。
openshift_logging_fluentd_remote_syslog_remove_tag_prefix	从标签中删除前缀，默认为 ''（空白）。
openshift_logging_fluentd_remote_syslog_tag_key	如果指定了字符串，则使用此字段作为要在记录上查看的键，以对 syslog 消息设置标签。
openshift_logging_fluentd_remote_syslog_payload_key	如果指定了字符串，则使用此字段作为要在记录上查看的键，在 syslog 消息上设置有效负载。

MUX Logging Ansible 变量

参数	描述
openshift_logging_mux_remote_syslog	默认值为 **false**。设置为 **true** 以启用 fluent-plugin-remote-syslog gem。
openshift_logging_mux_remote_syslog_host	远程 syslog 服务器的主机名或 IP 地址，这是必需的。
openshift_logging_mux_remote_syslog_port	要连接的端口号，默认为 **514**。
openshift_logging_mux_remote_syslog_severity	设置 syslog 严重性级别，默认为 **debug**。
openshift_logging_mux_remote_syslog_facility	设置 syslog 工具，默认为 **local0**。
openshift_logging_mux_remote_syslog_use_record	默认值为 **false**。设置为 **true** 可使用记录的严重性和工具字段对 syslog 消息进行设置。
openshift_logging_mux_remote_syslog_remove_tag_prefix	从标签中删除前缀，默认为 **''**（空白）。
openshift_logging_mux_remote_syslog_tag_key	如果指定了字符串，则使用此字段作为要在记录上查看的键，以对 syslog 消息设置标签。
openshift_logging_mux_remote_syslog_payload_key	如果指定了字符串，则使用此字段作为要在记录上查看的键，在 syslog 消息上设置有效负载。

36.9. 执行管理 ELASTICSEARCH 操作

从日志记录版本 3.2.0 开始，一个管理员证书、密钥和 CA 可用于在 Elasticsearch 上通信并执行管理操作，会在 **logging-elasticsearch** secret 中提供。

注意

要确认您的 EFK 安装是否提供它们，请运行：

```
$ oc describe secret logging-elasticsearch
```

1. 连接到您试图执行维护的集群中的 Elasticsearch pod。
2. 要在集群中查找 pod，请使用：

```
$ oc get pods -l component=es -o name | head -1
$ oc get pods -l component=es-ops -o name | head -1
```

3. 连接到 pod:

```
$ oc rsh <your_Elasticsearch_pod>
```

4. 连接到 Elasticsearch 容器后，您可以使用从 secret 挂载的证书，根据其 Indices API 文档 与 Elasticsearch 通信。
Fluentd 使用索引格式 **project.{project_name}.{project_uuid}.YYYY.MM.DD** 将日志发送到 Elasticsearch，其中 YYYY.MM.DD 是日志记录的日期。

   例如，要删除来自 2016 年 6 月 15 日的 uuid 为 **3b3594fa-2ccd-11e6-acb7-0eb6b35eaee3** 的**openshift-logging** 项目的所有日志，您可以运行：

```
$ curl --key /etc/elasticsearch/secret/admin-key \
  --cert /etc/elasticsearch/secret/admin-cert \
  --cacert /etc/elasticsearch/secret/admin-ca -XDELETE \
  "https://localhost:9200/project.logging.3b3594fa-2ccd-11e6-acb7-
0eb6b35eaee3.2016.06.15"
```

36.10. 重新部署 EFK 证书

您可以使用 Ansible playbook 为 EFK 堆栈执行证书轮转，而无需运行 install/upgrade playbook。

此 playbook 删除当前证书文件、生成新的 EFK 证书、更新证书 secret 并重启 Kibana 和 Elasticsearch 以强制这些组件在更新证书中读取中。

要重新部署 EFK 证书：

1. 使用 Ansible playbook 重新部署 EFK 证书：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook playbooks/openshift-logging/redeploy-certificates.yml
```

36.11. 更改聚合日志记录驱动程序

对于聚合日志，建议使用 **json-file** 日志驱动程序。

重要

使用 **json-file** 驱动程序时，请确保使用 Docker 版本 **docker-1.12.6-55.gitc4618fb.el7_4 now** 或更高版本。

Fluentd 通过检查 */etc/docker/daemon.json* 和 */etc/sysconfig/docker* 文件来确定驱动程序 Docker 正在使用。

您可以通过 **docker info** 命令确定 Docker 正在使用哪些驱动程序：

```
# docker info | grep Logging

Logging Driver: journald
```

进入 **json-file** ：

1. 修改 */etc/sysconfig/docker* 或 */etc/docker/daemon.json* 文件。

例如：

```
# cat /etc/sysconfig/docker
OPTIONS=' --selinux-enabled --log-driver=json-file --log-opt max-size=1M --log-opt max-file=3 --signature-verification=False'

cat /etc/docker/daemon.json
{
"log-driver": "json-file",
"log-opts": {
"max-size": "1M",
"max-file": "1"
}
}
```

2. 重启 Docker 服务：

```
systemctl restart docker
```

3. 重启 Fluentd。

警告

同时，在超过 dozen 节点上重启 Fluentd 会在 Kubernetes 调度程序上创建一个大型负载。在使用以下指示重启 Fluentd 时要小心谨慎。

重启 Fluentd 的方法有两种。您可以在一个节点或一组节点或所有节点中重启 Fluentd。

a. 以下步骤演示了如何在一个节点或一组节点上重启 Fluentd。

i. 列出运行 Fluentd 的节点：

```
$ oc get nodes -l logging-infra-fluentd=true
```

ii. 对于每个节点，删除标签并关闭 Fluentd：

```
$ oc label node $node logging-infra-fluentd-
```

iii. 验证 Fluentd 已关闭：

```
$ oc get pods -l component=fluentd
```

iv. 对于每个节点，重启 Fluentd：

```
$ oc label node $node logging-infra-fluentd=true
```

b. 以下步骤演示了如何重启 Fluentd 所有节点。

i. 在所有节点上关闭 Fluentd：

```
$ oc label node -l logging-infra-fluentd=true --overwrite logging-infra-fluentd=false
```

ii. 验证 Fluentd 已关闭：

```
$ oc get pods -l component=fluentd
```

iii. 在所有节点上重启 Fluentd：

```
$ oc label node -l logging-infra-fluentd=false --overwrite logging-infra-fluentd=true
```

iv. 验证 Fluentd 是否存在：

```
$ oc get pods -l component=fluentd
```

36.12. 手动 ELASTICSEARCH ROLLOUTS

自 OpenShift Container Platform 3.7 起，Aggregated Logging 堆栈更新了 Elasticsearch Deployment Config 对象，使其不再有 Config Change Trigger，这意味着对 **dc** 的任何更改都不会造成自动推出部署。这是为了防止 Elasticsearch 集群中发生意外重启，这可能会在成员重启时造成过量分片重新平衡。

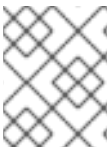
本节将介绍两个重启步骤： rolling-restart 和 full-restart。如果滚动重启会在没有停机时间的情况下对 Elasticsearch 集群进行适当的更改（配置了三个 master），并完全重启会安全地应用现有数据的主要更改。

36.12.1. 执行 Elasticsearch Rolling 集群重启

当进行了以下任何更改时，建议使用滚动重启：

- 运行 Elasticsearch Pod 的节点需要重启
- logging-elasticsearch configmap
- logging-es-* 部署配置
- 新镜像部署或升级

这是推荐的重启策略。



注意

如果 **openshift_logging_use_ops** 配置为 **True**，则需要对 ops 集群重复执行 Elasticsearch 集群的所有操作。

1. 防止在有意关闭节点时进行分片平衡：

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> -- \
    curl -s \
    --cacert /etc/elasticsearch/secret/admin-ca \
    --cert /etc/elasticsearch/secret/admin-cert \
    --key /etc/elasticsearch/secret/admin-key \
    -XPUT 'https://localhost:9200/_cluster/settings' \
    -d '{ "transient": { "cluster.routing.allocation.enable" : "none" } }'
```

2. 完成后，对于 Elasticsearch 集群都有的每个 **dc**，运行 **oc rollout latest** 来部署最新版本的 **dc** 对象：

```
$ oc rollout latest <dc_name>
```

您将看到部署了新的 pod。当 pod 有两个就绪的容器后，您可以继续到下一个 **dc**。

3. 推出集群的所有"dc"后，重新启用分片平衡：

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> -- \
        curl -s \
        --cacert /etc/elasticsearch/secret/admin-ca \
        --cert /etc/elasticsearch/secret/admin-cert \
        --key /etc/elasticsearch/secret/admin-key \
        -XPUT 'https://localhost:9200/_cluster/settings' \
        -d '{ "transient": { "cluster.routing.allocation.enable" : "all" } }'
```

36.12.2. 执行 Elasticsearch 完整集群重启

当更改 Elasticsearch 的主要版本或其他更改时，建议您完全重启，这可能会在更改过程中造成数据完整性。

注意

如果 **openshift_logging_use_ops** 配置为 **True**，则需要对 ops 集群重复执行 Elasticsearch 集群的所有操作。

注意

对 **logging-es-ops** 服务进行更改时，使用 "es-ops-blocked" 和 "es-ops" in patch

1. 禁用 Elasticsearch 集群的所有外部通信，同时禁用它。编辑非集群日志记录服务（如 **logging-es** 和**logging-ops**）不再与运行的 Elasticsearch pod 匹配：

```
$  oc patch svc/logging-es -p '{"spec":{"selector":{"component":"es-
blocked","provider":"openshift"}}}'
```

2. 执行分片同步刷新，确保在关机之前没有等待写入磁盘的待定操作：

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> -- \
        curl -s \
        --cacert /etc/elasticsearch/secret/admin-ca \
        --cert /etc/elasticsearch/secret/admin-cert \
        --key /etc/elasticsearch/secret/admin-key \
        -XPOST 'https://localhost:9200/_flush/synced'
```

3. 防止在有意关闭节点时进行分片平衡：

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> -- \
        curl -s \
        --cacert /etc/elasticsearch/secret/admin-ca \
        --cert /etc/elasticsearch/secret/admin-cert \
```

```
    --key /etc/elasticsearch/secret/admin-key \
    -XPUT 'https://localhost:9200/_cluster/settings' \
    -d '{ "transient": { "cluster.routing.allocation.enable" : "none" } }'
```

4. 完成后，对于 Elasticsearch 集群中的每个 **dc**，请缩减所有节点：

```
$ oc scale dc <dc_name> --replicas=0
```

5. 缩减完成后，对于 Elasticsearch 集群都有的每个 **dc**，运行 **oc rollout latest** 来部署最新版本的 **dc** 对象：

```
$ oc rollout latest <dc_name>
```

   您将看到部署了新的 pod。当 pod 有两个就绪的容器后，您可以继续到下一个 **dc**。

6. 部署完成后，对于 Elasticsearch 集群都有每个 **dc**，扩展节点：

```
$ oc scale dc <dc_name> --replicas=1
```

7. 扩展完成后，启用到 ES 集群的所有外部通信。编辑非集群日志记录服务（如 **logging-es** 和 **logging-ops**）来再次运行 Elasticsearch Pod：

```
$ oc patch svc/logging-es -p '{"spec":{"selector":{"component":"es","provider":"openshift"}}}'
```

36.13. EFK 故障排除

以下是对集群日志记录部署的多个常见问题的信息进行故障排除：

36.13.1. 与所有 EFK 组件相关的故障排除

以下故障排除问题一般适用于 EFK 堆栈。

部署失败，ReplicationControllers 扩展至 0

如果您执行在十分钟超时前无法成功启动实例的部署，OpenShift Container Platform 会将部署视为失败，并缩减到零实例。**oc get pods** 命令会显示一个 deployer pod，它带有非零退出代码，没有部署 pod。

在以下示例中，会显示 Elasticsearch 部署的 deployer pod 名称；这来自于 ReplicationController **logging-es-2e7ut0iq-1**，它是 DeploymentConfig **logging-es-2e7ut0iq** 的部署。

```
NAME                          READY     STATUS            RESTARTS   AGE
logging-es-2e7ut0iq-1-deploy   1/1       ExitCode:255      0          1m
```

由于多个过渡的原因，部署失败可能会发生，如镜像拉取用时过长，或者节点没有响应。

检查部署器 pod 日志，以了解可能的原因或尝试重新部署：

```
$ oc deploy --latest logging-es-2e7ut0iq
```

另外，尝试扩展现有部署：

```
$ oc scale --replicas=1 logging-es-2e7ut0iq-1
```

如果问题仍然存在，请检查 pod、事件和 systemd 单元日志以确定问题的来源。

无法解析 kubernetes.default.svc.cluster.local

此 master 的内部别名必须由主服务器上附带的 DNS 服务器解析。根据您的平台，您可以针对 master 运行 **dig** 命令（例如，容器）以检查这种情况：

```
$ dig kubernetes.default.svc.cluster.local @localhost
[...]
;; QUESTION SECTION:
;kubernetes.default.svc.cluster.local. IN A

;; ANSWER SECTION:
kubernetes.default.svc.cluster.local. 30 IN A   172.30.0.1
```

旧版本的集群日志记录不会自动为 master 定义此内部别名。您可能需要升级集群，以便使用聚合日志。如果您的集群为最新版本，则代表您的 pod 到达 master 上的 SkyDNS 解析器或者 pod 可能会被阻止。您必须在再次部署前解决这个问题。

无法连接到 master 或服务

如果 DNS 解析没有返回到所有地址，或者无法从 pod 中连接到的地址（如 **fluentd** pod），这可能代表系统防火墙/网络问题。您必须调试这个问题。

36.13.2. 与 ElasticSearch 的相关故障排除

以下故障排除问题适用于 EFK 堆栈的 ElasticSearch 组件。

Elasticsearch 部署永远不会成功，并回滚到以前的版本

此情形通常是在 OpenShift Container Platform 中部署在 AWS 上部署的集群日志记录。描述 Elasticsearch pod 通常会显示重新附加 pod 存储的问题：

```
$ oc describe pod <elasticsearch-pod>
```

考虑为每个 Elasticsearch 部署配置打补丁，以便 AWS 有更多时间使存储可用：

```
$ oc patch dc <elasticsearch-deployment-config> -p '{"spec":{"strategy":{"recreateParams":
{"timeoutSeconds":1800}}}}'
```

searchguard index 仍为红色

这是一个已知问题：升级并移到每个集群的单个 SearchGuard 索引，而不是每个部署配置一个索引。Elasticsearch Explain API 用于发现原因，并需要删除到节点分配的索引：

```
$ oc -c elasticsearch exec ${pod} -- es_util --query=".searchguard/_settings" -XPUT -d "
{\"index.routing.allocation.include._name\": \"\"}"
```

Elasticsearch Pod 永不就绪

当初始化和看到进程失败时，存在一个已知问题，这可以从红色的 **.searchguard** 索引中。

```
for p in $(oc get pods -l component=es -o jsonpath={.items[*].metadata.name}); do \
  oc exec -c elasticsearch $p -- touch /opt/app-root/src/init_failures;  \
done
```

36.13.3. Kibana

以下故障排除问题适用于 EFK 堆栈的 Kibana 组件。

在 Kibana 上循环登录

如果您启动 Kibana 控制台并成功登录，您会被错误地重定向到 Kibana，这会立即重定向到登录屏幕。

造成此问题的原因是，Kibana 前面的 OAuth2 代理必须与 master 的 OAuth2 服务器共享一个 secret，以便将其识别为有效的客户端。此问题可能表示 secret 不匹配。没有以公开方式报告此问题。

当您部署日志多次时，可能会发生这种情况。例如，如果您修复了初始部署，且由 Kibana 使用的 **secret** 会被替换，而匹配的 master **oauthclient** 条目不会被替换。

您可以执行以下操作：

```
$ oc delete oauthclient/kibana-proxy
```

按照 **openshift-ansible** 指令重新运行 **openshift_logging** 角色。这会替换 oauthclient，下一个成功登录不应循环。

```
*"error":"invalid\_request" on login*
```

Kibana 上的登录错误

当尝试访问 Kibana 控制台时，您可能会收到浏览器错误：

```
{"error":"invalid_request","error_description":"The request is missing a required parameter,
 includes an invalid parameter value, includes a parameter more than once, or is otherwise
malformed."}
```

此问题可能是由 OAuth2 客户端和服务器间的不匹配造成的。客户端的返回地址必须在白名单中，这样服务器才能在登录后安全地重定向回此地址。如果存在不匹配，则会显示错误消息。

原因可能是由来自以前部署的 **oauthclient** 条目导致，在这种情况下，您可以替换它：

```
$ oc delete oauthclient/kibana-proxy
```

按照 **openshift-ansible** 指令重新运行 **openshift_logging** 角色，该角色替换 **oauthclient** 条目。返回到 Kibana 控制台，然后再次登录。

如果问题仍然存在，请检查您是否通过 OAuth 客户端中列出的 URL 来访问 Kibana。通过转发端口（例如 1443）而非标准的 443 HTTPS 端口访问 URL 可能会造成此问题。

您可以通过编辑其 **oauthclient** 来调整服务器白名单：

```
$ oc edit oauthclient/kibana-proxy
```

编辑接受的重定向 URI 列表，使其包含您实际使用的地址。保存并退出后，应该解决这个错误。

Kibana 访问显示 503 错误

如果您在查看 Kibana 控制台时收到代理错误，则可能是由两个问题之一造成的。

- Kibana 可能无法识别 Pod。如果 ElasticSearch 启动缓慢，则 Kibana 可能会错误地尝试访问 ElasticSearch，而 Kibana 不会考虑它。您可以检查相关的服务是否具有任何端点：

```
$ oc describe service logging-kibana
Name:                   logging-kibana
[...]
Endpoints:              <none>
```

  如果有任何 Kibana Pod 处于活动状态，则应当列出端点。如果没有，请检查 Kibana Pod 和部署的状态。

- 用于访问 Kibana 服务的指定路由可能会被屏蔽。
  如果您在一个项目中执行测试部署，然后在另一项目中进行部署，但没有彻底删除第一个部署，可能会发生这种情况。当多个路由发送到同一目的地时，默认路由器仅路由到创建的第一个目的地。检查有问题的路由，以查看是否在多个位置定义了该路由：

```
$ oc get route  --all-namespaces --selector logging-infra=support
NAMESPACE   NAME        HOST/PORT                 PATH      SERVICE
logging     kibana      kibana.example.com                  logging-kibana
logging     kibana-ops  kibana-ops.example.com              logging-kibana-ops
```

  在本例中，没有重叠的路由。

第 37 章 聚合日志记录分类指南

37.1. 概述

Elasticsearch、Fluentd 和 Kibana (EFK)堆栈聚合了 OpenShift Container Platform 安装内部运行的节点和应用程序的日志。部署后，它使用 Fluentd 将所有节点和 Pod 的日志聚合到 Elasticsearch(ES) 中。它还提供了一个中央化的 Kibana Web UI，用户和管理员可以在其中使用汇总的数据创建丰富的视觉化和仪表板。

37.2. 安装

在 OpenShift Container Platform 中安装聚合日志堆栈的一般步骤请参考聚合容器日志。在完成安装指南时需要考虑一些重要事项：

要让日志记录 pod 在您的集群间平均分配，应在创建项目时使用空白 节点选择器。

```
$ oc adm new-project logging --node-selector=""
```

与稍后执行的节点标签结合使用时，这会控制日志记录项目中的 pod 放置。

Elasticsearch(ES)应该部署集群大小至少为 3 个，以应对节点故障。这可以通过在清单主机文件中设置 **openshift_logging_es_cluster_size** 参数指定。

如需完整的参数列表，请参阅 Ansible 变量。

Kibana 需要主机名，该主机名可以从 中解析，无论浏览器将用于访问它。例如，您可能需要将 Kibana 的 DNS 别名添加到公司名称服务，以便从笔记本电脑上运行的 Web 浏览器访问 Kibana。日志记录部署会在其中一个"infra"节点或 OpenShift 路由器运行的位置创建到 Kibana 的 Route。Kibana 主机名别名应指向此机器。此主机名被指定为 Ansible **openshift_logging_kibana_hostname** 变量。

安装可能需要一些时间，具体取决于镜像是否已从 registry 中检索，以及集群大小。

在 **openshift-logging** 项目中，您可以使用 **oc get all** 检查您的部署。

```
$ oc get all

NAME                 REVISION            REPLICAS     TRIGGERED BY
logging-curator            1                   1
logging-es-6cvk237t        1                   1
logging-es-e5x4t4ai        1                   1
logging-es-xmwvnorv        1                   1
logging-kibana             1                   1
NAME                 DESIRED             CURRENT      AGE
logging-curator-1          1                   1            3d
logging-es-6cvk237t-1      1                   1            3d
logging-es-e5x4t4ai-1      1                   1            3d
logging-es-xmwvnorv-1      1                   1            3d
logging-kibana-1           1                   1            3d
NAME                 HOST/PORT           PATH         SERVICE           TERMINATION
LABELS
logging-kibana             kibana.example.com                 logging-kibana       reencrypt
component=support,logging-infra=support,provider=openshift
logging-kibana-ops         kibana-ops.example.com             logging-kibana-ops   reencrypt
component=support,logging-infra=support,provider=openshift
```

```
NAME                    CLUSTER-IP          EXTERNAL-IP   PORT(S)          AGE
logging-es              172.24.155.177      <none>        9200/TCP         3d
logging-es-cluster      None                <none>        9300/TCP         3d
logging-es-ops          172.27.197.57       <none>        9200/TCP         3d
logging-es-ops-cluster  None                <none>        9300/TCP         3d
logging-kibana          172.27.224.55       <none>        443/TCP          3d
logging-kibana-ops      172.25.117.77       <none>        443/TCP          3d
NAME                    READY               STATUS        RESTARTS         AGE
logging-curator-1-6s7wy 1/1                 Running       0                3d
logging-deployer-un6ut  0/1                 Completed     0                3d
logging-es-6cvk237t-1-cnpw3 1/1             Running       0                3d
logging-es-e5x4t4ai-1-v933h 1/1             Running       0                3d
logging-es-xmwvnorv-1-adr5x 1/1             Running       0                3d
logging-fluentd-156xn   1/1                 Running       0                3d
logging-fluentd-40biz   1/1                 Running       0                3d
logging-fluentd-8k847   1/1                 Running       0                3d
```

您应该最后有一个类似于下文的设置：

```
$ oc get pods -o wide

NAME                    READY     STATUS      RESTARTS   AGE       NODE
logging-curator-1-6s7wy 1/1       Running     0          3d        ip-172-31-24-239.us-west-
2.compute.internal
logging-deployer-un6ut  0/1       Completed   0          3d        ip-172-31-6-152.us-west-
2.compute.internal
logging-es-6cvk237t-1-cnpw3 1/1   Running     0          3d        ip-172-31-24-238.us-west-
2.compute.internal
logging-es-e5x4t4ai-1-v933h 1/1   Running     0          3d        ip-172-31-24-235.us-west-
2.compute.internal
logging-es-xmwvnorv-1-adr5x 1/1   Running     0          3d        ip-172-31-24-233.us-west-
2.compute.internal
logging-fluentd-156xn   1/1       Running     0          3d        ip-172-31-24-241.us-west-
2.compute.internal
logging-fluentd-40biz   1/1       Running     0          3d        ip-172-31-24-236.us-west-
2.compute.internal
logging-fluentd-8k847   1/1       Running     0          3d        ip-172-31-24-237.us-west-
2.compute.internal
logging-fluentd-9a3qx   1/1       Running     0          3d        ip-172-31-24-231.us-west-
2.compute.internal
logging-fluentd-abvgj   1/1       Running     0          3d        ip-172-31-24-228.us-west-
2.compute.internal
logging-fluentd-bh74n   1/1       Running     0          3d        ip-172-31-24-238.us-west-
2.compute.internal
...
...
```

默认情况下，分配给每个 ES 实例的 RAM 量为 16GB。**openshift_logging_es_memory_limit** 是 **openshift-ansible** 主机清单文件中使用的参数。请记住，这个值 **的一半** 将传递给单个 elasticsearch pod java 进程 堆大小。

了解有关安装 EFK 的更多信息。

37.2.1. 大型集群

在 100 个节点或更多节点上，建议首先从 **docker pull registry.redhat.io/openshift3/logging-fluentd:v3.11** 中拉取日志记录镜像。部署日志记录基础架构 Pod（Elasticsearch、Kibana 和 Curator）后，应一次在 20 个节点中执行节点标签。例如：

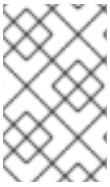
使用一个简单的循环：

```
$ while read node; do oc label nodes $node logging-infra-fluentd=true; done < 20_fluentd.lst
```

以下也可以工作：

```
$ oc label nodes 10.10.0.{100..119} logging-infra-fluentd=true
```

为节点分组 OpenShift 日志记录使用的速度，帮助避免在镜像 registry 等共享资源上发生冲突。



注意

检查任何 "CrashLoopBackOff | ImagePullFailed | Error" 问题的发生情况。**oc logs <pod>**、**oc describe pod <pod>** 和 **oc get event** are helpful diagnostic commands are helpful diagnostic 命令。

37.3. SYSTEMD-JOURNALD 和 RSYSLOG

在 Red Hat Enterprise Linux(RHEL)7 中，**systemd-journald.socket** 单元会在引导过程中创建 */dev/log*，然后将输入传递给 **systemd-journald.service**。每个 **syslog（）** 调用都会进入日志。

systemd-journald 的默认速率限制会导致在 Fluentd 读取 Fluentd 前丢弃一些系统日志。要防止将以下添加到 */etc/systemd/journald.conf* 文件：

```
# Disable rate limiting
RateLimitInterval=1s
RateLimitBurst=10000
Storage=volatile
Compress=no
MaxRetentionSec=30s
```

然后重启该服务。

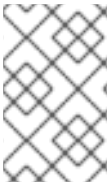
```
$ systemctl restart systemd-journald.service
$ systemctl restart rsyslog.service
```

这些设置负责大量上传的突发性。

删除速率限制后，您可能会看到系统日志记录守护进程的 CPU 使用率增加，因为它会处理之前可以被限制的消息。

37.4. 扩展 EFK 日志

如果您没有在首次部署中指明所需的规模，则在使用更新的 **openshift_logging_es_cluster_size** value. 参数更新清单文件后，调整集群要运行 Ansible 日志记录 playbook 的最小破坏性方式。如需更多信息，请参阅执行 Elasticsearch 操作 部分。



注意

高可用性 Elasticsearch 环境 需要至少三个 Elasticsearch 节点，每个都在不同的主机上，并将 **openshift_logging_es_number_of_replicas** Ansible 变量设置为 **1** 或更高版本来创建副本。

37.4.1. Master 事件被聚合到 EFK 作为日志

eventrouter pod 从 kubernetes API 中提取事件，并输出到 **STDOUT**。**fluentd** 插件转换日志消息并将其发送到 Elasticsearch(ES)。

通过将 **openshift_logging_install_eventrouter** 设置为 **true** 来启用它。默认是 off。**Listener** 被部署到 default 命名空间。收集的信息位于 ES 的操作索引中，只有集群管理员具有可视化访问权限。

37.5. 存储注意事项

Elasticsearch 索引是分片及其对应副本的集合。这是 ES 如何在内部实现高可用性，因此不需要使用基于硬件的镜像 RAID 变体。RAID 0 仍可用于提高整体磁盘性能。

每个 Elasticsearch 部署配置中都添加了一个持久性卷。在 OpenShift Container Platform 中，这通常通过 持久性卷声明 来实现。

PVC 基于 **openshift_logging_es_pvc_prefix** 设置来命名。如需更多详细信息，请参阅持久性 Elasticsearch 存储。

Fluentd 将 **systemd journal** 和 **/var/lib/docker/containers/*.log** 的所有日志发送到 Elasticsearch。 了解更多。

建议本地 SSD 驱动器以获得最佳性能。在 Red Hat Enterprise Linux(RHEL)7 中，截止时间 IO 调度程序是除 SATA 磁盘外的所有块设备的默认设置。对于 SATA 磁盘，默认的 IO 调度程序是 **cfq**。

为 ES 调整存储大小取决于如何优化索引。因此，请提前考虑需要的数据量，并且注意要聚合应用程序的日志数据。一些 Elasticsearch 用户发现，有必要使绝对存储消耗始终保持在 50% 左右并处于 70% 以下。这有助于避免 Elasticsearch 在进行大型数据合并操作期间变得无响应。

第 38 章 启用集群指标

38.1. 概述

kubelet 会公开可由 Heapster 收集并存储在后端的指标。

作为 OpenShift Container Platform 管理员，您可以在一个用户界面中查看所有容器和组件的集群指标。

注意

早期版本的 OpenShift Container Platform 使用 Heapster 中的指标来配置 Pod 横向自动扩展。现在，pod 横向自动扩展使用 OpenShift Container Platform 指标服务器的指标。如需更多信息，请参阅使用 Horizontal Pod Autoscaler 的要求。

本主题描述使用 Hawkular Metrics 作为指标引擎，将其数据存储到 Cassandra 数据库中。当进行了配置时，可以从 OpenShift Container Platform Web 控制台查看此 CPU、内存和基于网络的指标。

Heapster 从主服务器检索所有节点列表，然后通过 **/stats** 端点单独联系每个节点。从那里，Heapster 会提取 CPU、内存和网络使用量的指标，然后将它们导出到 Hawkular Metrics。

kubelet 上可用的存储卷指标无法通过 **/stats** 端点提供，但可通过 **/metrics** 端点提供。如需更多信息，请参阅 Prometheus Monitoring。

在 web 控制台中浏览各个 pod 会显示单独的 sparkline chart 用于内存和 CPU。显示的时间范围是可选择的，这些 chart 每 30 秒自动更新。如果 pod 上有多个容器，则可以选择一个特定的容器来显示其指标。

如果为项目定义了 资源限值，您还可看到每个 Pod 的圆环图。donut chart 显示有关资源限值的使用量。例如：**145 个大小为 200 MiB**，其圆环图显示 **55 MiB Used**。

38.2. 开始前

Ansible playbook 可用于部署和升级集群指标。您应该熟悉 安装集群 指南。这提供了准备使用 Ansible 的信息，并包含有关配置的信息。参数添加到 Ansible 清单文件中，以配置集群指标的不同区域。

下面描述了各种区域，以及可以添加到 Ansible 清单文件中的参数，以修改默认值。

38.3. 指标数据存储

您可以将指标数据存储至 持久存储 或临时 pod 卷 中。

38.3.1. 持久性存储

使用持久性存储运行 OpenShift Container Platform 集群指标意味着您的指标存储在持久性卷中，并可在 Pod 重新启动或重新创建后保留。如果您需要指标数据防御数据丢失，这是理想选择。对于生产环境，强烈建议为您的指标 pod 配置持久性存储。

Cassandra 存储的大小要求取决于 pod 的数量。管理员负责确保大小要求足够了设置，并监控使用量以确保磁盘不会满。持久性卷声明的大小指定了 **openshift_metrics_cassandra_pvc_size** ansible 变量，默认设为 10 GB。

如果要使用动态置备的持久性卷，将清单文件中的 **openshift_metrics_cassandra_storage_type** 变量设置为 **dynamic**。

38.3.2. 集群指标的容量规划

运行 **openshift_metrics** Ansible 角色后， **oc get pod** 的输出应类似：

```
# oc get pods -n openshift-infra
NAME                         READY     STATUS    RESTARTS   AGE
hawkular-cassandra-1-l5y4g   1/1       Running   0          17h
hawkular-metrics-1t9so       1/1       Running   0          17h
heapster-febru               1/1       Running   0          17h
```

OpenShift Container Platform 指标使用 Cassandra 数据库存储，该数据库使用 **openshift_metrics_cassandra_limits_memory 设置进行部署：2g** ；此值可以根据 Cassandra 起始脚本确定的可用内存进一步调整。这个值应该涵盖大多数 OpenShift Container Platform 指标安装，但在使用环境变量，您可以在部署集群指标前修改 **MAX_HEAP_SIZE** 和堆新生成大小 **HEAP_NEWSIZE**。

默认情况下，指标数据存储 7 天。在 7 天后，Cassandra 开始清除最旧的指标数据。已删除 pod 和项目的指标数据不会自动清除，它只在数据超过七天后删除。

例 38.1. 由 10 个节点和 1000 个 Pod 累计的数据

在包括 10 个节点和 1000 个 pod 的测试场景中，持续 24 小时内指标数据的 2.5 GB 周期。因此，在这种情况下，指标数据的容量规划公式为：

$(((2.5 \times 10^9) \div 1000) \div 24) \div 10^6$ = ~0.125 MB/hour per pod.

例 38.2. 120 节点和 10000 个 Pod 收集的数据

在包括 120 个节点和 10000 个 pod 的测试场景中，持续 24 小时内的数据累积了 25 GB 指标数据。因此，在这种情况下，指标数据的容量规划公式为：

（（11.410 的值 10^9）IFL 1000 IFL 24）leveloffset 10^6 = 0.475 MB/hour

	1000 个 pod	10000 个 pod
Cassandra 存储数据在 24 小时内累计超过 24 小时（默认指标参数）	2.5 GB	11.4 GB

如果 **openshift_metrics_duration** 的默认值（7 天）， **openshift_metrics_resolution** 的默认值（30 秒）被保留，对于 Cassandra pod 的每周的存储要求为：

	1000 个 pod	10000 个 pod
Cassandra 存储数据持续 7 天（默认指标参数）	20 GB	90 GB

在上表中，在预期的存储空间中添加了一个额外的 10%，作为意外监控 pod 的使用情况。

警告

如果 Cassandra 持久的卷超出足够空间，则会发生数据丢失。

要使集群指标使用持久性存储，请确保持久性卷有 **ReadWriteOnce** 访问模式。如果此模式不活动，则持久卷声明无法找到持久卷，并且 Cassandra 无法启动。

要将持久性存储与指标组件搭配使用，请确保持久性卷有足够大小的持久性卷。OpenShift Ansible **openshift_metrics** 角色处理 持久性卷声明 的创建。

OpenShift Container Platform 指标还支持动态置备的持久性卷。要将此功能用于 OpenShift Container Platform 指标，需要将 **openshift_metrics_cassandra_storage_type** 的值设置为 动态。您可以使用 EBS、GCE 和 Cinder 存储后端 来动态置备持久性卷。

有关配置性能和扩展集群指标 pod 的详情，请参考扩展集群指标主题。

表 38.1. Cassandra 基于集群中的节点/pod 数量的数据库存储要求

节点数量	Pod 数	Cassandra 存储增长速度	Cassandra 每天增长存储	每周增长的 Cassandra 存储
210	10500	每小时 500 MB	15 GB	75 GB
990	11000	每小时 1 GB	30 GB	210 GB

在上个计算中，预计大小大约有 20% 增加了开销，以确保存储要求不会超过计算的值。

如果 **METRICS_DURATION** 和 **METRICS_RESOLUTION** 值保持在默认值（**7** 天和 **15** 秒），则安全地规划星期的 Cassandra 存储要求。

警告

由于 OpenShift Container Platform 指标使用 Cassandra 数据库作为指标数据的数据存储，如果在指标 设置过程中设置了 **USE_THREEISTANT_STORAGE=true**，**PV** 将位于网络存储中，而 NFS 作为默认数据。不过，不建议将网络存储与 Cassandra 搭配使用，如 Cassandra 文档中所述。

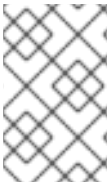
已知问题和限制

测试发现，**堆** ster指标组件能够处理最多 25,000 个 Pod。如果 pod 数量超过该数量，Heapster 开始在指标处理后落入，从而不再出现指标图形的可能性。工作正在持续增加 Heapster 可以收集指标的 pod 数量，以及另一 metrics-gathering 解决方案的上游开发。

38.3.3. 非持久性存储

使用非持久性存储运行 OpenShift Container Platform 集群指标意味着在 pod 被删除时删除所有存储的指标。虽然运行带有非持久性数据的集群指标会更容易运行，但使用非持久性数据运行会存在持久性数据丢失的风险。但是，指标仍可在容器被重启后保留。

要使用非持久性存储，必须将 **openshift_metrics_cassandra_storage_type** 变量设置为 inventory 文件中的 **emptydir**。



注意

使用非持久性存储时，指标数据会被写入节点上的 ***/var/lib/origin/openshift.local.volumes/pods***，其中的 Cassandra pod 运行 Ensure ***/var*** 有足够可用空间来容纳指标存储。

38.4. 指标 ANSIBLE 角色

OpenShift Container Platform Ansible **openshift_metrics** 角色使用 配置 Ansible 清单文件中的变量来配置和部署所有指标组件。

38.4.1. 指定 Metrics Ansible 变量

OpenShift Ansible 中包含的 **openshift_metrics** 角色定义了用于部署集群指标的任务。以下是需要覆盖它们时可以添加到清单文件中的角色变量列表。

表 38.2. Ansible 变量

变量	描述
openshift_metrics_install_metrics	如果为 **true**，则部署指标。否则，取消部署。
openshift_metrics_start_cluster	在部署组件后启动 metrics 集群。
openshift_metrics_startup_timeout	在尝试重启前，等待 Hawkular Metrics 和 Heapster 的时间（以秒为单位）。
openshift_metrics_duration	在清除指标前存储指标的天数。
openshift_metrics_resolution	收集指标的频率。定义为编号和时间标识符：秒(s)、分钟(m)、小时(h)。
openshift_metrics_cassandra_pvc_name	使用此变量指定要使用的 Cassandra 卷的确切名称。如果具有指定名称的卷不存在，则创建它。此变量只能与单个 Cassandra 副本一起使用。对于多个 Cassandra 副本，改为使用变量 **openshift_metrics_cassandra_pvc_prefix**。
openshift_metrics_cassandra_pvc_prefix	为 Cassandra 创建的持久卷声明前缀。从 1 开始，会将序列号附加到前缀中。
openshift_metrics_cassandra_pvc_size	各个 Cassandra 节点的持久卷声明大小。

变量	描述
openshift_metrics_cassandra_pvc_storage_class_name	指定要使用的存储类。如果要显式设置存储类，还要设置 **openshift_metrics_cassandra_storage_type=pv**。
openshift_metrics_cassandra_storage_type	对于临时存储，使用 **emptydir**（用于测试）；对于持久性卷（需要在安装前创建它），使用 **pv**；对于动态持久性卷，使用 **dynamic**。如果要显式设置存储类，请指定 **pv** 并设置 **openshift_metrics_cassandra_pvc_storage_class_name**。
openshift_metrics_cassandra_replicas	指标堆栈的 Cassandra 节点数量。此值指定 Cassandra 复制控制器的数量。
openshift_metrics_cassandra_limits_memory	Cassandra容器集的内存限值。例如，值 **2Gi** 会将 Cassandra 限制为 2 GB 内存。根据要在其上调度节点的可用内存，启动脚本可以进一步调整这个值。
openshift_metrics_cassandra_limits_cpu	Cassandra 容器集的 CPU 限值。例如，值 **4000m** (4000 millicores)会将 Cassandra 限制为 4 个 CPU。
openshift_metrics_cassandra_requests_memory	为 Cassandra 容器集请求的内存量。例如，**2Gi** 值将请求 2 GB 内存。
openshift_metrics_cassandra_requests_cpu	Cassandra 容器集的 CPU 请求。例如，值 **4000m**（4000 毫秒）会请求 4 个 CPU。
openshift_metrics_cassandra_storage_group	用于 Cassandra 的补充组。
openshift_metrics_cassandra_nodeselector	设置为所需的现有 节点选择器，以确保 pod 放置到具有特定标签的节点上。例如，**{"node-role.kubernetes.io/infra":"true"}**。如果未指定，则 Cassandra Pod 将部署到任何可调度节点上。
openshift_metrics_hawkular_ca	用于为 Hawkular 证书签名的可选证书颁发机构 (CA) 文件。
openshift_metrics_hawkular_cert	用于重新加密到 Hawkular 指标的证书文件。证书必须包含路由使用的主机名。如果未指定，则使用默认路由器证书。
openshift_metrics_hawkular_key	与 Hawkular 证书一起使用的密钥文件。

变量	描述
openshift_metrics_hawkular_limits_memory	限制 Hawkular 容器集的内存量。例如，**2Gi** 值会将 Hawkular pod 限制为 2 GB 内存。根据要在其上调度节点的可用内存，启动脚本可以进一步调整这个值。
openshift_metrics_hawkular_limits_cpu	Hawkular pod 的 CPU 限制。例如，值 **4000m** (4000 millicores)会将 Hawkular pod 限制为 4 个 CPU。
openshift_metrics_hawkular_replicas	Hawkular 指标的副本数量。
openshift_metrics_hawkular_requests_memory	对 Hawkular 容器集请求的内存量。例如，**2Gi** 值将请求 2 GB 内存。
openshift_metrics_hawkular_requests_cpu	Hawkular Pod 的 CPU 请求。例如，值 **4000m**（4000 毫秒）会请求 4 个 CPU。
openshift_metrics_hawkular_nodeselector	设置为所需的现有 节点选择器，以确保 pod 放置到具有特定标签的节点上。例如，**{"node-role.kubernetes.io/infra":"true"}**。如果未指定，Hawkular Pod 将部署到任何可调度节点上。
openshift_metrics_heapster_allowed_users	要接受的、以逗号分隔的 CN 列表。默认情况下，这设置为允许 OpenShift 服务代理进行连接。在覆盖时，将 **system:master-proxy** 添加到列表中，以允许 pod 横向自动扩展 正常工作。
openshift_metrics_heapster_limits_memory	限制 Heapster Pod 的内存量。例如，**2Gi** 值会将 Heapster pod 限制为 2 GB 内存。
openshift_metrics_heapster_limits_cpu	Heapster pod 的 CPU 限值。例如，值 **4000m** (4000 millicores)会将 Heapster pod 限制为 4 个 CPU。
openshift_metrics_heapster_requests_memory	对 Heapster Pod 请求的内存量。例如，**2Gi** 值将请求 2 GB 内存。
openshift_metrics_heapster_requests_cpu	Heapster Pod 的 CPU 请求。例如，值 **4000m**（4000 毫秒）会请求 4 个 CPU。
openshift_metrics_heapster_standalone	仅部署 Heapster，但不部署 Hawkular Metrics 和 Cassandra 组件。
openshift_metrics_heapster_nodeselector	设置为所需的现有 节点选择器，以确保 pod 放置到具有特定标签的节点上。例如，**{"node-role.kubernetes.io/infra":"true"}**。如果未指定，则 Heapster Pod 将部署到任何可调度的节点上。
openshift_metrics_hawkular_hostname	在执行 **openshift_metrics** Ansible 角色时设置，因为它将主机名用于 Hawkular Metrics 路由。这个值应该与完全限定域名对应。

如需有关如何 指定请求和限值的进一步讨论，请参阅计算资源。

如果您通过 Cassandra 使用 持久性存储，管理员负责使用 **openshift_metrics_cassandra_pvc_size** 变量为集群设置足够的磁盘大小。管理员还负责监控磁盘使用情况，以确保其不会被完全生效。

警告

如果 Cassandra 持久的卷超出足够空间，则数据丢失结果。

所有其他变量都是可选的，允许进行更大的自定义。例如，如果您有一个自定义安装，其中 Kubernetes master 在 **https://kubernetes.default.svc:443** 下不可用，则可以指定要与 **openshift_metrics_master_url** 参数搭配使用的值。

38.4.2. 使用 secret

OpenShift Container Platform Ansible **openshift_metrics** 角色会自动生成自签名证书，以便在其组件之间使用，并生成一个重新加密路由 来公开 Hawkular 指标服务。此路由允许 Web 控制台访问 Hawkular Metrics 服务。

要让运行 Web 控制台的浏览器信任通过此路由的连接，它必须信任路由的证书。这可以通过提供由可信证书颁发机构签名的您自己的证书来实现。**openshift_metrics** 角色允许您指定自己的证书，然后在创建路由时使用该证书。

如果没有提供您自己的证书，则使用路由器的默认证书。

38.4.2.1. 提供您拥有的证书

要提供自己的证书（由 re-encrypting 路由使用）,您可以在清单文件中设置 **openshift_metrics_hawkular_cert**、**openshift_metrics_hawkular_key** 和 **openshift_metrics_hawkular_ca** 变量。

hawkular-metrics.pem 值需要以 *.pem* 格式包含证书。您可能还需要提供通过 **hawkular-metrics-ca.cert** secret 签名此 *pem* 文件的证书颁发机构的证书。

如需更多信息，请参阅 重新加密路由文档。

38.5. 部署指标数据组件

由于部署和配置所有指标组件都是使用 OpenShift Container Platform Ansible 处理的，所以您可以在一个步骤中部署所有内容。

以下示例演示了如何使用默认参数部署带有和不使用持久性存储的指标。

重要

运行 Ansible playbook 的主机，对于清单（inventory）中的每个主机都至少需要有 75MiB 可用内存。

重要

根据上游 Kubernetes 规则，指标只能从 **eth0** 的默认接口收集。

例 38.3. 使用持久性存储部署

以下命令将 Hawkular Metrics 路由设置为使用 **hawkular-metrics.example.com**，并使用持久存储进行部署。

您必须具有足够大小的持久性卷。

```
$ ansible-playbook [-i </path/to/inventory>] <OPENSHIFT_ANSIBLE_DIR>/playbooks/openshift-metrics/config.yml \
  -e openshift_metrics_install_metrics=True \
  -e openshift_metrics_hawkular_hostname=hawkular-metrics.example.com \
  -e openshift_metrics_cassandra_storage_type=pv
```

例 38.4. 在没有持久性存储的情况下部署

以下命令将 Hawkular Metrics 路由设置为使用 **hawkular-metrics.example.com** 并部署没有持久性存储。

```
$ ansible-playbook [-i </path/to/inventory>] <OPENSHIFT_ANSIBLE_DIR>/playbooks/openshift-metrics/config.yml \
  -e openshift_metrics_install_metrics=True \
  -e openshift_metrics_hawkular_hostname=hawkular-metrics.example.com
```

警告

由于这是在没有持久性存储的情况下被部署，因此可能会发生指标数据丢失。

38.5.1. Metrics Diagnostics

是指标的一些诊断，可帮助评估指标堆栈的状态。为指标执行诊断：

```
$ oc adm diagnostics MetricsApiProxy
```

38.6. 设置 METRICS PUBLIC URL

OpenShift Container Platform Web 控制台使用来自 Hawkular Metrics 服务的数据来显示其图形。访问 Hawkular Metrics 服务的 URL 必须使用 master webconsole-config configmap 文件中的 **metricsPublicURL** 选项进行配置。此 URL 对应于在部署指标组件期间使用 **openshift_metrics_hawkular_hostname** 清单变量创建的路由。

注意

您必须能够从访问控制台的浏览器中解析 **openshift_metrics_hawkular_hostname**。

例如，如果您的 **openshift_metrics_hawkular_hostname** 对应于 **hawkular-metrics.example.com**，您必须在 webconsole-config configmap 文件中进行以下更改：

```
clusterInfo:
  ...
  metricsPublicURL: "https://hawkular-metrics.example.com/hawkular/metrics"
```

在更新并保存 **webconsole-config** configmap 文件后，Web 控制台 Pod 会在存活度探测配置设定的延迟后重启，默认为 30 秒。

当 OpenShift Container Platform 服务器备份并运行时，指标会显示在 pod 概述页中。

小心

如果您使用自签名证书，请记住 Hawkular Metrics 服务以不同的主机名托管，并使用与控制台不同的证书。您可能需要显式打开 metricsPublicURL 中 **指定的值** 并接受该证书。

要避免这个问题，请使用您的浏览器可接受的证书。

38.7. 直接访问 HAWKULAR METRICS

要更直接访问和管理指标，请使用 Hawkular Metrics API。

注意

从 API 访问 Hawkular Metrics 时，您只能执行读取。默认情况下禁用写入指标。如果您希望单个用户也可以编写指标，您必须将 **openshift_metrics_hawkular_user_write_access** 变量设置为 **true**。

不过，建议您使用默认配置，并且只有指标通过 Heapster 输入系统。如果启用了写入访问权限，任何用户都可以向系统写入指标，这可能会影响性能，并导致 Cassandra 磁盘用量不可预测。

Hawkular Metrics 文档 介绍了如何使用 API，但处理 OpenShift Container Platform 上配置的 Hawkular Metrics 版本时有几个区别：

38.7.1. OpenShift Container Platform 项目和 Hawkular Tenants

Hawkular Metrics 是多租户应用。它被配置为 OpenShift Container Platform 中的项目对应于 Hawkular Metrics 中的租户。

因此，在访问名为 **MyProject** 的项目的指标时，您必须将 **Hawkular-Tenant** 标头设置为 **MyProject**。

还有一个特殊的租户，名为 **_system**，其中包含系统级别指标。这需要一个 **cluster-reader** 或 **cluster-admin** 级别才能访问。

38.7.2. 授权

Hawkular Metrics 服务针对 OpenShift Container Platform 验证用户身份，以确定该用户是否能够访问该项目。

Hawkular Metrics 从客户端接受 bearer 令牌，并使用 **SubjectAccessReview** 验证与 OpenShift Container Platform 服务器的令牌。如果用户对项目有正确的读取权限，可以读取该项目的指标。对于 **_system** 租户，请求从此租户读取的用户必须具有 **cluster-reader** 权限。

在访问 Hawkular Metrics API 时，您必须在 **Authorization** 标头中传递 bearer 令牌。

38.8. 扩展 OPENSHIFT CONTAINER PLATFORM 集群指标 POD

有关扩展集群指标功能的信息，请参阅 扩展和性能指南。

38.9. CLEANUP

您可以通过执行以下步骤来删除 OpenShift Container Platform Ansible **openshift_metrics** 角色部署的所有内容：

```
$ ansible-playbook [-i </path/to/inventory>] <OPENSHIFT_ANSIBLE_DIR>/playbooks/openshift-metrics/config.yml \
  -e openshift_metrics_install_metrics=False
```

第 39 章 自定义 WEB 控制台

39.1. 概述

管理员可以使用扩展来自定义 Web 控制台，允许您在 web 控制台 加载时运行脚本和载入自定义风格表。扩展脚本允许您覆盖 Web 控制台的默认行为并根据您的需要进行自定义。

例如，扩展脚本可用于添加您自己的公司的品牌或添加特定于公司的功能。这种情况的常见用例是为不同的环境重新分配或标记空格。您可以使用同一扩展代码，但提供更改 Web 控制台的设置。

小心

请谨慎更改 Web 控制台风格或行为，但不记录在下面。添加任何脚本或风格表，但在升级时可能需要对重要的自定义操作，因为 web 控制台标记和行为在将来的版本中有所变化。

39.2. 加载扩展脚本和 STYLESHEETS

只要可从浏览器中访问 URL，就可以在任何 **https://** URL 中托管扩展脚本和样式表。文件可以使用公开访问的路由，或 OpenShift Container Platform 之外的其他服务器上，从平台上的 pod 托管。

要添加脚本和风格表，请编辑 **openshift-web-console** 命名空间中的 **webconsole-config** ConfigMap。Web 控制台配置包括在 ConfigMap 的 **webconsole-config.yaml** 键中。

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

要添加脚本，请更新 **extensions.scriptURLs** 属性。值是 URL 的数组。

要添加风格表，更新 **extensions.stylesheetURLs** 属性。值是 URL 的数组。

extensions.stylesheetURLs 设置示例

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    extensions:
      scriptURLs:
        - https://example.com/scripts/menu-customization.js
        - https://example.com/scripts/nav-customization.js
      stylesheetURLs:
        - https://example.com/styles/logo.css
        - https://example.com/styles/custom-styles.css
    [...]
```

保存 ConfigMap 后，Web 控制台容器将在几分钟后自动更新新扩展文件。

注意

脚本和样式表必须以正确的内容类型提供，否则不会由浏览器运行。脚本必须使用 **Content-Type: application/javascript**，样式表为 **Content-Type: text/css**。

最佳实践是将扩展脚本嵌套在 Immediately Invoked function Expression(IIFE)中。这样可确保您不会创建与 web 控制台或其他扩展所使用的名称冲突的全局变量。例如：

```
(function() {
  // Put your extension code here...
}());
```

以下部分中的示例显示了您可以自定义 Web 控制台的常见方法。

注意

GitHub 上的 OpenShift Origin 存储库中提供了额外的扩展示例。

39.2.1. 设置扩展属性

如果您有一个特定的扩展，但要为每个环境使用不同的文本，您可以在 web 控制台配置中定义环境，并跨环境使用相同的扩展脚本。

要添加扩展属性，请编辑 **openshift-web-console** 命名空间中的 **webconsole-config** ConfigMap。Web 控制台配置包括在 ConfigMap 的 **webconsole-config.yaml** 键中。

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

更新 **extensions.properties** 值，它是键值对映射。

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    extensions:
      [...]
      properties:
        doc_url: https://docs.openshift.com
        key1: value1
        key2: value2
    [...]
```

这会生成一个全局变量，它由扩展访问，就像执行以下代码一样：

```
window.OPENSHIFT_EXTENSION_PROPERTIES = {
  doc_url: "https://docs.openshift.com",
  key1: "value1",
  key2: "value2"
}
```

39.3. 外部日志记录解决方案的扩展选项

您可以使用扩展选项来链接外部日志记录解决方案，而不使用 OpenShift Container Platform 的 EFK 日志记录堆栈：

```
'use strict';
angular.module("mylinkextensions", ['openshiftConsole'])
```

```
      .run(function(extensionRegistry) {
        extensionRegistry.add('log-links', _.spread(function(resource, options) {
          return {
            type: 'dom',
            node: '<span><a href="https://extension-point.example.com">' + resource.metadata.name +
'</a><span class="action-divider">|</span></span>'
          };
        }));
      });
hawtioPluginLoader.addModule("mylinkextensions");
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.4. 自定义和禁用指南

当用户在特定浏览器中首次登录时，会弹出一个指导式导览。您可以为新用户启用 **auto_launch** ：

```
window.OPENSHIFT_CONSTANTS.GUIDED_TOURS.landing_page_tour.auto_launch = true;
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.5. 自定义文档链接

登录页面上的文档链接可自定义。**window.OPENSHIFT_CONSTANTS.CATALOG_HELP_RESOURCES** 是包含标题和 **href** 的对象数组。这些将转换为链接。您可以完全覆盖阵列、推送或弹出附加链接，或者修改现有链接的属性。例如：

```
window.OPENSHIFT_CONSTANTS.CATALOG_HELP_RESOURCES.links.push({
  title: 'Blog',
  href: 'https://blog.openshift.com'
});
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.6. 自定义徽标

在 web 控制台标头中更改徽标的以下样式：

```
#header-logo {
  background-image: url("https://www.example.com/images/logo.png");
  width: 190px;
  height: 20px;
}
```

将 **example.com** URL 替换为实际镜像的 URL，并调整宽度和高度。理想的高度是 20px。

如 Loading Extension Scripts 和 Stylesheets 所述添加风格表。

39.7. 自定义成员资格列表

membership 页面中的默认白名单显示集群角色的子集，如 **admin**、**basic-user**、**edit** 等等。它还显示项目中定义的自定义角色。

例如，将您自己的自定义集群角色集合添加到白名单中：

```
window.OPENSHIFT_CONSTANTS.MEMBERSHIP_WHITELIST = [
  "admin",
  "basic-user",
  "edit",
  "system:deployer",
  "system:image-builder",
  "system:image-puller",
  "system:image-pusher",
  "view",
  "custom-role-1",
  "custom-role-2"
];
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.8. 更改到文档的链接

Web 控制台的不同部分中会显示到外部文档的链接。以下示例更改了两个给定到文档的链接的 URL：

```
window.OPENSHIFT_CONSTANTS.HELP['get_started_cli']     = "https://example.com/doc1.html";
window.OPENSHIFT_CONSTANTS.HELP['basic_cli_operations'] = "https://example.com/doc2.html";
```

另外，您可以更改所有文档链接的基本 URL。

这个示例会导致默认帮助 URL **https://example.com/docs/welcome/index.html** ：

```
window.OPENSHIFT_CONSTANTS.HELP_BASE_URL = "https://example.com/docs/"; 1
```

1 路径必须以 / 结尾。

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.9. 添加或更改链接以下载 CLI

Web 控制台中的 **About** 页面为 命令行界面(CLI) 工具提供下载链接。这些链接可通过提供链接文本和 URL 来配置，以便您可以选择将其直接指向文件软件包，或指向实际软件包的外部页面。

例如，要直接指向可下载的软件包，链接文本是软件包平台：

```
window.OPENSHIFT_CONSTANTS.CLI = {
  "Linux (32 bits)": "https://<cdn>/openshift-client-tools-linux-32bit.tar.gz",
  "Linux (64 bits)": "https://<cdn>/openshift-client-tools-linux-64bit.tar.gz",
  "Windows":         "https://<cdn>/openshift-client-tools-windows.zip",
  "Mac OS X":        "https://<cdn>/openshift-client-tools-mac.zip"
};
```

或者，使用 **Latest Release** 链接文本指向链接实际下载软件包的页面：

```
window.OPENSHIFT_CONSTANTS.CLI = {
  "Latest Release": "https://<cdn>/openshift-client-tools/latest.html"
};
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.9.1. 自定义关于页面

为 web 控制台提供自定义 **About** 页面：

1. 编写类似如下的扩展：

```
angular
  .module('aboutPageExtension', ['openshiftConsole'])
  .config(function($routeProvider) {
    $routeProvider
      .when('/about', {
        templateUrl: 'https://example.com/extensions/about/about.html',
        controller: 'AboutController'
      });
    }
  );

hawtioPluginLoader.addModule('aboutPageExtension');
```

2. 编写自定义模板。
   从您使用的 OpenShift Container Platform 发行版本的 ***about.html*** 版本开始。在模板中，可以使用两个单范围变量： **version.master.openshift** 和 **version.master.kubernetes**。

3. 通过 web 控制台正确的 Cross-Origin Resource Sharing(CORS)响应标头，在 URL 中托管模板。

   a. 设置 **Access-Control-Allow-Origin** 响应，以允许来自 web 控制台域的请求。

   b. 将 **Access-Control-Allow-Methods** 设置为包含 **GET**。

   c. 将 **Access-Control-Allow-Headers** 设置为包含 **Content-Type**。

或者，您可以使用 AngularJS ***$templateCache*** 直接将模板包含在 JavaScript 中。

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.10. 配置导航菜单

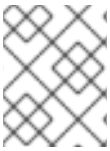
39.10.1. 顶部导航菜单

Web 控制台的顶部导航栏包含帮助图标和用户下拉菜单。您可以使用 angular-extension-registry 添加额外的菜单项。

可用的扩展点包括：

- **nav-help** -Dropdown - 帮助图标下拉菜单，在桌面屏幕宽度中可见
- **nav-user-dropdown** - 用户下拉菜单，在桌面屏幕宽度中可见
- **nav-dropdown-mobile** - 顶部导航项目的单一菜单，位于移动屏幕宽度上

以下示例扩展了 **nav-help-drop 下拉菜单**，名称为 < **myExtensionModule** > ：



注意

<myExtensionModule& gt; 是一个占位符名称。每个下拉菜单扩展必须足够唯一，这样它不会给以后任何模块冲突。

```
angular
  .module('<myExtensionModule>', ['openshiftConsole'])
  .run([
    'extensionRegistry',
    function(extensionRegistry) {
      extensionRegistry
        .add('nav-help-dropdown', function() {
          return [
            {
              type: 'dom',
              node: '<li><a href="http://www.example.com/report" target="_blank">Report a Bug</a></li>'
            }, {
              type: 'dom',
              node: '<li class="divider"></li>'  // If you want a horizontal divider to appear in the menu
            }, {
              type: 'dom',
              node: '<li><a href="http://www.example.com/status" target="_blank">System Status</a>
</li>'
            }
          ];
        });
    }
  ]);

hawtioPluginLoader.addModule('<myExtensionModule>');
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.10.2. application Launcher

顶部导航栏还包含链接到其他 Web 应用程序的可选应用程序启动程序。默认情况下，此下拉菜单为空，但是当添加链接时，masthead 中的帮助菜单左侧会出现。

```
// Add items to the application launcher dropdown menu.
window.OPENSHIFT_CONSTANTS.APP_LAUNCHER_NAVIGATION = [{
  title: "Dashboard",                 // The text label
  iconClass: "fa fa-dashboard",       // The icon you want to appear
  href: "http://example.com/dashboard",  // Where to go when this item is clicked
  tooltip: 'View dashboard'           // Optional tooltip to display on hover
}, {
  title: "Manage Account",
  iconClass: "pficon pficon-user",
  href: "http://example.com/account",
  tooltip: "Update email address or password."
}];
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.10.3. 系统状态 Badge

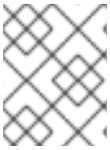
顶部导航栏也可以包括一个可选的系统状态徽标，以告知用户系统范围的事件，如维护窗口。要使用一个黄色警告图标使用 existing 风格，请按照以下示例操作。

```
'use strict';

angular
  .module('mysystemstatusbadgeextension', ['openshiftConsole'])
  .run([
    'extensionRegistry',
    function(extensionRegistry) {
      // Replace http://status.example.com/ with your domain
      var system_status_elem = $('<a href="http://status.example.com/"' +
      'target="_blank" class="nav-item-iconic system-status"><span title="' +
      'System Status" class="fa status-icon pficon-warning-triangle-o">' +
      '</span></a>');

      // Add the extension point to the registry so the badge appears
      // To disable the badge, comment this block out
      extensionRegistry
        .add('nav-system-status', function() {
          return [{
            type: 'dom',
            node: system_status_elem
          }];
        });
    }
  ]);

hawtioPluginLoader.addModule('mysystemstatusbadgeextension');
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.10.4. 项目左导航

在项目中导航时，左侧会显示一个菜单，其中包含主要和次要导航。此菜单结构定义为一个常态，可以被覆盖或修改。



注意

对项目导航的显著自定义可能会影响用户体验，并应谨慎考虑。如果您修改现有的导航项目，您可能需要在将来的升级中更新此自定义。

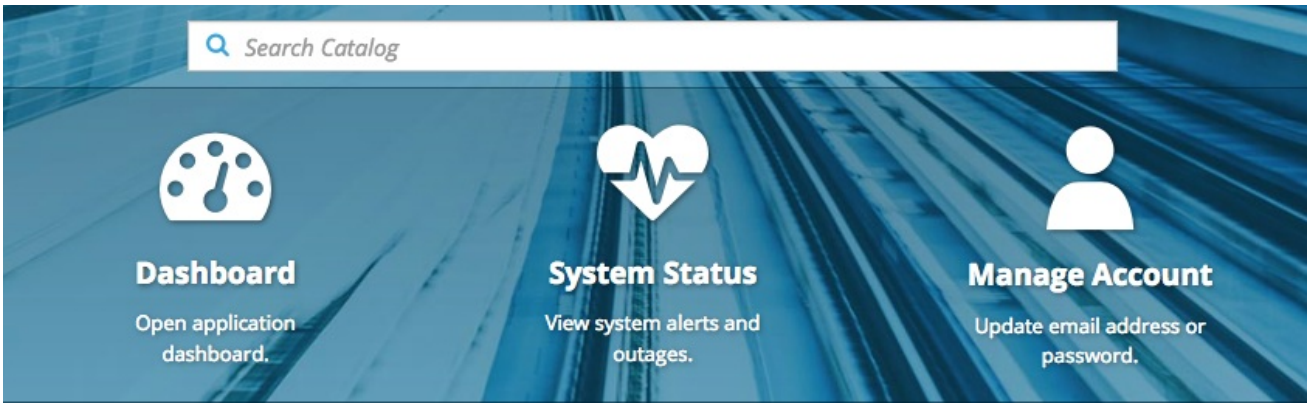
```
// Append a new primary nav item.  This is a simple direct navigation item
// with no secondary menu.
window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION.push({
  label: "Dashboard",          // The text label
  iconClass: "fa fa-dashboard", // The icon you want to appear
  href: "/dashboard"           // Where to go when this nav item is clicked.
                               // Relative URLs are pre-pended with the path
                               // '/project/<project-name>'
});
```

```
// Splice a primary nav item to a specific spot in the list.  This primary item has
// a secondary menu.
window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION.splice(2, 0, { // Insert at the third spot
  label: "Git",
  iconClass: "fa fa-code",
  secondaryNavSections: [       // Instead of an href, a sub-menu can be defined
    {
      items: [
        {
          label: "Branches",
          href: "/git/branches",
          prefixes: [
            "/git/branches/"     // Defines prefix URL patterns that will cause
                                 // this nav item to show the active state, so
                                 // tertiary or lower pages show the right context
          ]
        }
      ]
    },
    {
      header: "Collaboration",   // Sections within a sub-menu can have an optional header
      items: [
        {
          label: "Pull Requests",
          href: "/git/pull-requests",
          prefixes: [
            "/git/pull-requests/"
          ]
        }
      ]
    }
  ]
});

// Add a primary item to the top of the list.  This primary item is shown conditionally.
window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION.unshift({
  label: "Getting Started",
  iconClass: "pficon pficon-screen",
  href: "/getting-started",
  prefixes: [                  // Primary nav items can also specify prefixes to trigger
    "/getting-started/"        // active state
  ],
  isValid: function() {        // Primary or secondary items can define an isValid
    return isNewUser;          // function. If present it will be called to test whether
                               // the item should be shown, it should return a boolean
  }
});

// Modify an existing menu item
var applicationsMenu = _.find(window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION, { label:
'Applications' });
applicationsMenu.secondaryNavSections.push({ // Add a new secondary nav section to the
Applications menu
  // my secondary nav section
});
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.11. 配置功能的应用程序

Web 控制台在其登录页面目录中有一个可选的应用程序链接列表。这些显示在页面顶部，并且可以具有一个图标、标题、简短描述和链接。

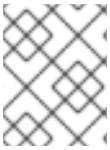



```
// Add featured applications to the top of the catalog.
window.OPENSHIFT_CONSTANTS.SAAS_OFFERINGS = [{
  title: "Dashboard",                    // The text label
  icon: "fa fa-dashboard",                  // The icon you want to appear
  url: "http://example.com/dashboard",       // Where to go when this item is clicked
  description: "Open application dashboard."  // Short description
}, {
  title: "System Status",
  icon: "fa fa-heartbeat",
  url: "http://example.com/status",
  description: "View system alerts and outages."
}, {
  title: "Manage Account",
  icon: "pficon pficon-user",
  url: "http://example.com/account",
  description: "Update email address or password."
}];
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.12. 配置目录类别

目录类别组织 web 控制台目录登录页面中的项目显示。每个类别都有一个或多个子类别。如果构建器镜像、模板或服务在子类别中分组，如果它包含匹配子类别标签中列出的标签，则项目可以出现在多个子类别中。类别和子类别仅在其中至少一个项目时才会显示。



注意

对目录类别进行大量自定义可能会影响用户体验，并应谨慎考虑。如果您修改现有类别项目，则可能需要在将来的升级中更新此自定义。

```
// Find the Languages category.
var category = _.find(window.OPENSHIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES,
                      { id: 'languages' });
```

```
// Add Go as a new subcategory under Languages.
category.subCategories.splice(2,0,{ // Insert at the third spot.
  // Required. Must be unique.
  id: "go",
  // Required.
  label: "Go",
  // Optional. If specified, defines a unique icon for this item.
  icon: "icon-go-gopher",
  // Required. Items matching any tag will appear in this subcategory.
  tags: [
    "go",
    "golang"
  ]
});

// Add a Featured category as the first category tab.
window.OPENSHIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES.unshift({
  // Required. Must be unique.
  id: "featured",
  // Required
  label: "Featured",
  subCategories: [
    {
      // Required. Must be unique.
      id: "go",
      // Required.
      label: "Go",
      // Optional. If specified, defines a unique icon for this item.
      icon: "icon-go-gopher",
      // Required. Items matching any tag will appear in this subcategory.
      tags: [
        "go",
        "golang"
      ]
    },
    {
      // Required. Must be unique.
      id: "jenkins",
      // Required.
      label: "Jenkins",
      // Optional. If specified, defines a unique icon for this item.
      icon: "icon-jenkins",
      // Required. Items matching any tag will appear in this subcategory.
      tags: [
        "jenkins"
      ]
    }
  ]
});
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.13. 配置配额通知消息

当用户达到配额时，配额通知会被放入 notification drawer 中。可以向通知添加自定义配额通知消息，针对配额资源类型。例如：

```
Your project is over quota. It is using 200% of 2 cores CPU (Limit). Upgrade
to <a href='https://www.openshift.com'>OpenShift Online Pro</a> if you need
additional resources.
```

"Upgrade to..."通知的一部分是自定义消息，可以包含 HTML，如链接到其他资源。

注意

由于配额消息是 HTML 标记，所以需要针对 HTML 正确转义任何特殊字符。

在扩展脚本中设置 **window.OPENSHIFT_CONSTANTS.QUOTA_MESSAGE** 属性来自定义每个资源的消息。

```
// Set custom notification messages per quota type/key
window.OPENSHIFT_CONSTANTS.QUOTA_NOTIFICATION_MESSAGE = {
  'pods': 'Upgrade to <a href="https://www.openshift.com">OpenShift Online Pro</a> if you need
additional resources.',
  'limits.memory': 'Upgrade to <a href="https://www.openshift.com">OpenShift Online Pro</a> if you
need additional resources.'
};
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.14. 配置 CREATE FROM URL 命名空间 WHITELIST

从 URL 创建 仅可用于来自 **OPENSHIFT_CONSTANTS.CREATE_FROM_URL_WHITELIST** 中明确指定的命名空间的镜像流或模板。要在白名单中添加命名空间，请按照以下步骤执行：

注意

openshift 默认包含在白名单中。不要删除它。

```
// Add a namespace containing the image streams and/or templates
window.OPENSHIFT_CONSTANTS.CREATE_FROM_URL_WHITELIST.push(
  'shared-stuff'
);
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.15. 禁用复制登录命令

Web 控制台允许用户将登录命令（包括当前访问令牌）复制到用户菜单和 Command Line Tools 页面中的剪贴板。可以更改这个功能，以便复制的命令中不包含用户的访问令牌。

```
// Do not copy the user's access token in the copy login command.
window.OPENSHIFT_CONSTANTS.DISABLE_COPY_LOGIN_COMMAND = true;
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

39.15.1. 启用通配符路由

如果为路由器启用了通配符路由，也可以在 web 控制台中启用通配符路由。这可让用户在创建路由时使用星号（如 ***.example.com** ）输入主机名。启用通配符路由：

```
window.OPENSHIFT_CONSTANTS.DISABLE_WILDCARD_ROUTES = false;
```

添加脚本，如 Loading Extension Scripts 和 Stylesheets 所述。

了解如何配置 HAProxy 路由器以允许通配符路由。

39.16. 自定义登录页面

您还可以更改登录页面，以及 web 控制台的登录供应商选择页面。运行以下命令来创建您可以修改的模板：

```
$ oc adm create-login-template > login-template.html
$ oc adm create-provider-selection-template > provider-selection-template.html
```

编辑该文件以更改样式或添加内容，但要小心不要删除大括号内任何需要的参数。

要使用您的自定义登录页面或供应商选择页面，请在 master 配置文件中设置以下选项：

```
oauthConfig:
  ...
  templates:
    login: /path/to/login-template.html
    providerSelection: /path/to/provider-selection-template.html
```

相对路径相对于主配置文件解析。更改此配置后，您必须重新启动服务器。

当配置了多个登录供应商时，或在 *master-config.yaml* 文件中的 **alwaysShowProviderSelection** 选项被设置为 **true** 时，每次用户的令牌过期时，用户都会显示这个自定义页面，然后才能继续其他任务。

39.16.1. 用法示例

自定义登录页面可用于创建服务信息条款。如果您使用第三方登录提供程序（如 GitHub 或 Google），它们也会非常有用，以便在重定向到身份验证提供程序前向用户显示他们信任的品牌页面。

39.17. 自定义 OAUTH 错误页面

身份验证过程中发生错误时，您可以更改显示的页面。

1. 运行以下命令来创建您可以修改的模板：

   ```
   $ oc adm create-error-template > error-template.html
   ```

2. 编辑该文件以更改风格或添加内容。
   您可以使用模板中的 **Error** 和 **ErrorCode** 变量。要使用您的自定义错误页面，请在 master 配置文件中设置以下选项：

   ```
   oauthConfig:
     ...
   ```

```
templates:
  error: /path/to/error-template.html
```

相对路径相对于主配置文件解析。

3. 更改此配置后，您必须重新启动服务器。

39.18. 更改 LOGOUT URL

您可以通过修改 **webconsole-config** ConfigMap 中的 **clusterInfo.logoutPublicURL** 参数，在注销控制台时更改控制台用户的位置。

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

以下是将注销 URL 更改为 **https://www.example.com/logout** 的示例：

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    clusterInfo:
      [...]
      logoutPublicURL: "https://www.example.com/logout"
    [...]
```

这在通过 Request Header 和 OAuth 或 OpenID 身份提供程序进行身份验证时，这将需要访问外部 URL 来销毁单点登录会话。

39.19. 使用 ANSIBLE 配置 WEB 控制台自定义

在集群安装过程中，可以使用以下参数来配置对 Web 控制台的很多修改，这些参数可在清单文件中进行配置：

- **openshift_master_logout_url**
- **openshift_web_console_extension_script_urls**
- **openshift_web_console_extension_stylesheet_urls**
- **openshift_master_oauth_templates**
- **openshift_master_metrics_public_url**
- **openshift_master_logging_public_url**

使用 Ansible 自定义 Web 控制台示例

```
# Configure `clusterInfo.logoutPublicURL` in the web console configuration
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#changing-the-logout-url
#openshift_master_logout_url=https://example.com/logout
```

```
# Configure extension scripts for web console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#loading-
custom-scripts-and-stylesheets
#openshift_web_console_extension_script_urls=['https://example.com/scripts/menu-
customization.js','https://example.com/scripts/nav-customization.js']

# Configure extension stylesheets for web console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#loading-
custom-scripts-and-stylesheets
#openshift_web_console_extension_stylesheet_urls=
['https://example.com/styles/logo.css','https://example.com/styles/custom-styles.css']

# Configure a custom login template in the master config
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#customizing
the-login-page
#openshift_master_oauth_templates={'login': '/path/to/login-template.html'}

# Configure `clusterInfo.metricsPublicURL` in the web console configuration for cluster metrics.
Ansible is also able to configure metrics for you.
# See: https://docs.openshift.com/enterprise/latest/install_config/cluster_metrics.html
#openshift_master_metrics_public_url=https://hawkular-metrics.example.com/hawkular/metrics

# Configure `clusterInfo.loggingPublicURL` in the web console configuration for aggregate logging.
Ansible is also able to install logging for you.
# See: https://docs.openshift.com/enterprise/latest/install_config/aggregate_logging.html
#openshift_master_logging_public_url=https://kibana.example.com
```

39.20. 更改 WEB 控制台 URL 端口和证书

要在用户访问 web 控制台 URL 时提供自定义证书，请将证书和密钥 URL 添加到 ***master-config.yaml*** 文件的 **namedCertificates** 部分。如需更多信息，请参阅为 Web 控制台或 CLI 配置自定义证书。

要设置或修改 web 控制台的重定向 URL，修改 **openshift-web-console oauthclient** ：

```
$ oc edit oauthclient openshift-web-console
```

要确保用户已被正确重定向，请更新 **openshift-web-console configmap** 的 **PublicUrls** ：

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

然后，更新 **consolePublicURL** 的值。

第 40 章 部署外部持久性卷置备器

40.1. 概述

重要

OpenShift Container Platform 上 AWS EFS 的外部置备程序是一个技术预览功能。技术预览功能不被红帽产品服务级别协议(SLA)支持，且可能无法完成。红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。如需更多信息，请参阅红帽技术预览功能支持范围。

外部置备程序是一个应用程序，为特定存储供应商启用动态置备。外部置备程序可以和 OpenShift Container Platform 提供的置备程序插件一同运行，并以类似的方式配置 ***StorageClass*** 对象，如 Dynamic Provisioning 和 Creating Storage Classes 部分所述。由于这些置备程序是外部的，您可以独立于 OpenShift Container Platform 部署和更新它们。

40.2. 开始前

Ansible Playbook 也可用于部署和升级外部置备程序。

注意

在继续之前，熟悉 配置集群指标 和配置 集群日志记录 部分。

40.2.1. 外部 Provisioners Ansible 角色

OpenShift Ansible **openshift_provisioners** 角色使用 Ansible 清单文件中的变量配置和部署外部置备程序。您需要指定要安装的置备程序，方法是将它们的 **install** 变了设置为 **true**。

40.2.2. 外部置备器 Ansible 变量

以下是应用于 **安装** 变量为 **true** 的所有置备程序的角色变量列表。

表 40.1. Ansible 变量

变量	描述
openshift_provisioners_install_provisioners	如果为 **true**，则部署所有其各自 **安装** 变量的置备程序，否则将变量设置为 **true**，否则将它们删除。
openshift_provisioners_image_prefix	组件镜像的前缀。例如，使用 Defaults to **registry.redhat.io/openshift3/**，如果您使用的是替代 registry，则将其设置为不同的值。
openshift_provisioners_image_version	组件镜像的版本。例如，对于 **openshift3/ose-efs-provisioner:v3.11**，将 version 设为 **v3.11**。
openshift_provisioners_project	在其中部署置备程序的项目。默认为 **openshift-infra**。

40.2.3. AWS EFS Provisioner Ansible 变量

AWS EFS 置备程序动态置备由给定 EFS 文件系统目录中创建的目录支持的 NFS PV。在配置 AWS EFS Provisioner Ansible 变量前，您必须满足以下要求：

- 由 *AmazonElasticFileSystemReadOnlyAccess* 策略分配的 IAM 用户（或更好）。
- 集群区域中的 EFS 文件系统。
- 挂载目标和安全组，以便任何节点（在集群区域的任意区域中）都可以通过其文件系统 DNS 名称挂载 EFS 文件系统。

表 40.2. 所需的 EFS Ansible 变量

变量	描述
openshift_provisioners_efs_fsid	EFS 文件系统的文件系统 ID，例如：**fs-47a2c22e**
openshift_provisioners_efs_region	EFS 文件系统的 Amazon EC2 区域。
openshift_provisioners_efs_aws_access_key_id	IAM 用户的 AWS 访问密钥（检查是否存在指定的 EFS 文件系统）。
openshift_provisioners_efs_aws_secret_access_key	IAM 用户的 AWS secret 访问密钥（检查指定的 EFS 文件系统是否存在）。

表 40.3. 可选的 EFS Ansible 变量

变量	描述
openshift_provisioners_efs	如果为 **true**，则 AWS EFS 置备程序会根据 **openshift_provisioners_install_provisioners** 为 **true** 或 **false** 来进行安装或卸载。默认值为 **false**。
openshift_provisioners_efs_path	EFS 文件系统中的目录的路径，EFS 置备程序会创建一个目录来备份它所创建的每个 PV。它必须存在，且可以被 EFS 置备程序挂载。默认为 **/persistentvolumes**。
openshift_provisioners_efs_name	*StorageClasses* 指定的 **置备程序** 名称。默认为 **openshift.org/aws-efs**。
openshift_provisioners_efs_nodeselector	标签映射，以选择 pod 将登录的节点。例如：**{"node":"infra","region":"west"}**。
openshift_provisioners_efs_supplementalgroup	要为 pod 提供 pod 的补充组，如果需要它才能写入 EFS 文件系统。默认为 **65534**。

40.3. 部署 PROVISIONERS

您可以根据 OpenShift Ansible 变量中指定的配置，一次或一次部署所有置备程序。以下示例演示了如何部署给定置备程序，然后创建并配置对应的 *StorageClass*。

40.3.1. 部署 AWS EFS Provisioner

以下命令将 EFS 卷中的目录设置为 **/data/persistentvolumes**。这个目录必须存在于文件系统中，且必须可以被 provisioner pod 挂载并可写入。进入 playbook 目录并运行以下 playbook：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v -i <inventory_file> \
    playbooks/openshift-provisioners/config.yml \
   -e openshift_provisioners_install_provisioners=True \
   -e openshift_provisioners_efs=True \
   -e openshift_provisioners_efs_fsid=fs-47a2c22e \
   -e openshift_provisioners_efs_region=us-west-2 \
   -e openshift_provisioners_efs_aws_access_key_id=AKIAIOSFODNN7EXAMPLE \
   -e
openshift_provisioners_efs_aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEK
EY \
   -e openshift_provisioners_efs_path=/data/persistentvolumes
```

40.3.1.1. AWS EFS 对象定义

aws-efs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: slow
provisioner: openshift.org/aws-efs (1)
parameters:
  gidMin: "40000" (2)
  gidMax: "50000" (3)
```

1. 设置此值与 **openshift_provisioners_efs_name** 变量的值相同，它默认为 **openshift.org/aws-efs**。
2. *StorageClass* 的 GID 范围最小值。（可选）
3. *StorageClass* 的 GID 范围的最大值。（可选）

每个动态置备的卷对应的 NFS 目录都会从范围 **gidMin-gidMax** 中分配一个唯一的 GID 拥有者。如果没有指定，**gidMin** 默认为 **2000**，**gidMax** 默认为 **2147483647**。任何通过声明消耗置备的卷的 pod 会自动使用所需的 GID 作为补充组运行，并可读取和写入卷。其他没有补充组（且不是以 root 身份运行）的挂载者将无法对该卷进行读取或写入。有关使用补充组管理 NFS 访问的详情，请查看 NFS 卷安全主题的 组 ID 部分。

40.4. CLEANUP

您可以运行以下命令来删除 OpenShift Ansible **openshift_provisioners** 角色部署的所有内容：

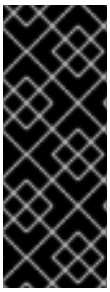
```
$ cd /usr/share/ansible/openshift-ansible
```

```
$ ansible-playbook -v -i <inventory_file> \
    playbooks/openshift-provisioners/config.yml \
   -e openshift_provisioners_install_provisioners=False
```

第 41 章 安装 OPERATOR FRAMEWORK（技术预览）

红帽已宣布了 Operator Framework，它是一个开源工具包，用来以更有效、自动化且可扩展的方式管理 Kubernetes 原生应用程序（称为 *Operators*）。

以下小节提供了作为集群管理员在 OpenShift Container Platform 3.11 中尝试技术预览 Operator Framework 的说明。



重要

Operator Framework 是一个技术预览功能。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

如需红帽技术预览功能支持范围的更多信息，请参阅 https://access.redhat.com/support/offerings/techpreview/。

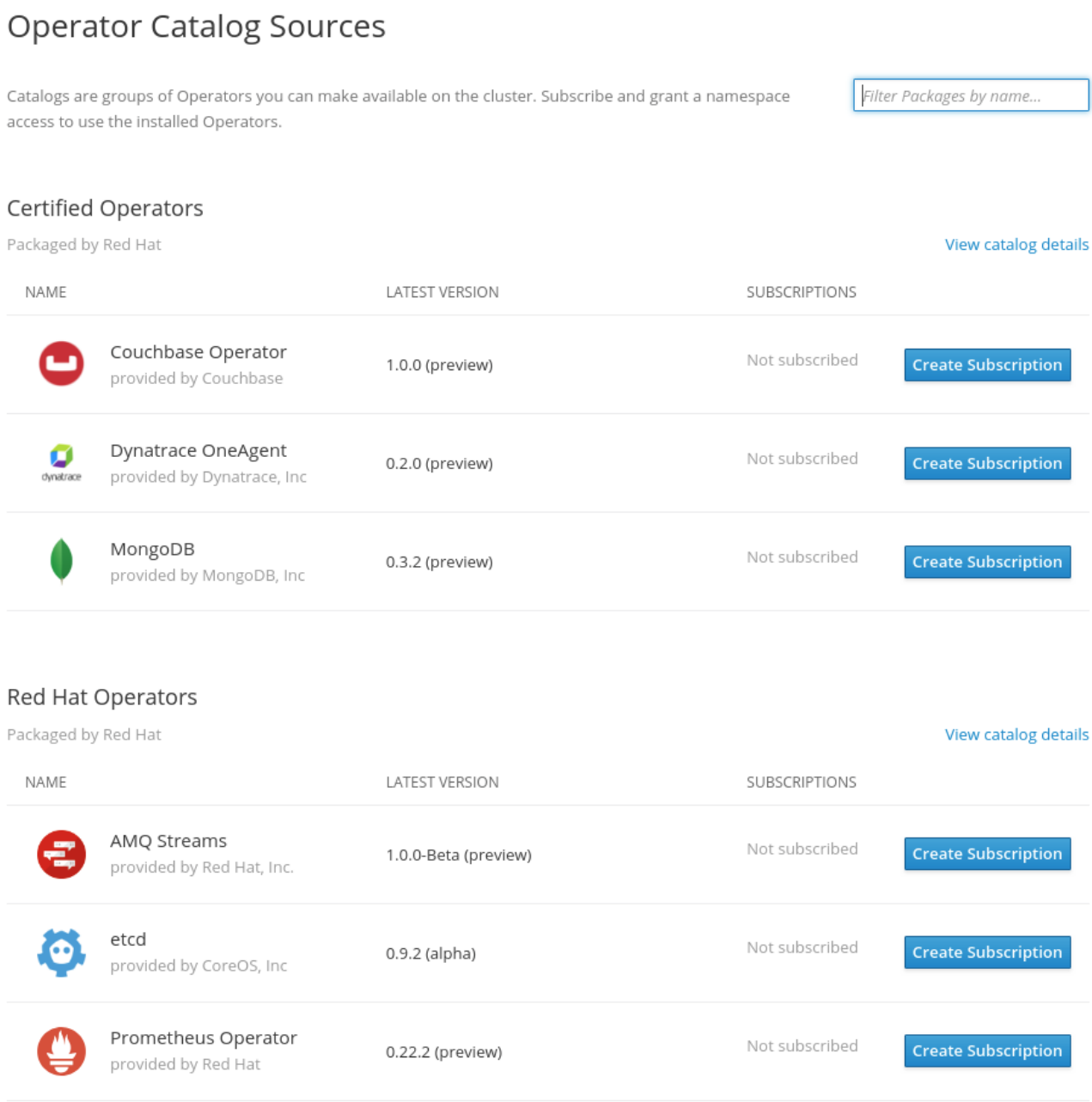
41.1. 什么是技术预览？

技术预览 Operator Framework 安装 Operator Lifecycle Manager(OLM)，辅助集群管理员安装、升级和授予其 OpenShift Container Platform 集群上运行的 Operator 访问权限。

OpenShift Container Platform Web 控制台也会使用新的管理界面更新为安装 Operator，并授予特定的项目访问权限以使用集群中可用的 Operator 目录。

开发人员通过自助服务体验，无需成为相关问题的专家也可自由置备和配置数据库、监控和大数据服务的实例，因为 Operator 已将相关知识融入其中。

图 41.1. Operator 目录源



在截屏中，您可以看到来自主要软件供应商的合作伙伴 Operator 的预加载目录源：

Couchbase Operator

Couchbase 提供了一个 NoSQL 数据库，它为存储和检索数据提供了机制，这比相关数据库中使用的表格关系以外的方式进行建模。这个 Operator 作为开发者预览在 OpenShift Container Platform 3.11 中提供，由 Couchbase 支持。您可在 OpenShift Container Platform 上原生运行 Couchbase 部署。它安装并可以更有效地故障转移 NoSQL 集群。

Dynatrace Operator

Dynatrace 应用程序监控实时提供性能指标，并帮助自动检测和诊断问题。Operator 将更轻松地安装以容器为中心的监控堆栈，并将其重新连接到 Dynatrace 监控云，监视自定义资源和监控所需状态。

MongoDB Operator

MongoDB 是一个分布式、事务型数据库，可在灵活、类似于 JSON 的文档中存储数据。Operator 支持部署 production-ready 副本集和分片集群，以及独立 dev/test 实例。它与 MongoDB Ops Manager 协同工作，确保所有集群都根据操作的最佳实践部署。

另外还包括以下红帽提供的 Operator：

Red Hat AMQ Streams Operator

Red Hat AMQ Streams 是一个基于 Apache Kafka 项目的可大规模扩展、分布式和高性能数据流平台。它提供分布式主干，使微服务和其他应用能够共享高吞吐量和低延迟的数据。

etcd Operator

etcd 是一种分布式键值存储，提供可靠的在机器集群中存储数据的方法。这个 Operator 允许用户使用一个简单的声明性配置来配置和管理 etcd 的复杂性，该配置是创建、配置和管理 etcd 集群。

Prometheus Operator

Prometheus 是 CNCF 内与 Kubernetes 托管的云端原生监控系统。该 Operator 包括应用程序域知识，用于处理常见任务，如创建/销毁、简单配置、通过标签自动生成监控目标配置等。

41.2. 使用 ANSIBLE 安装 OPERATOR LIFECYCLE MANAGER

要安装技术预览 Operator Framework，您可以在安装集群后在 OpenShift Container Platform **openshift-ansible** 安装程序中使用附带的 playbook。

注意

另外，技术预览 Operator Framework 可在初始集群安装过程中安装。如需了解单独的说明，请参阅配置 您的清单文件。

先决条件

- 现有 OpenShift Container Platform 3.11 集群
- 使用具有 **cluster-admin** 权限的账户访问该集群
- 最新 **openshift-ansible** 安装程序提供的 Ansible playbook

流程

1. 在用来安装和管理 OpenShift Container Platform 集群的清单文件中，在 **[OSEv3:vars]** 部分添加 **openshift_additional_registry_credentials** 变量，设置拉取 Operator 容器所需的凭证：

```
openshift_additional_registry_credentials=
[{'host':'registry.connect.redhat.com','user':'<your_user_name>','password':'<your_password>','test_image':'mongodb/enterprise-operator:0.3.2'}]
```

将 **user** 和 **password** 设置为您用来登录到红帽客户门户网站的凭证，地址为 https://access.redhat.com。

test_image 代表将用来测试您提供的凭证的镜像。

2. 进入 playbook 目录，并使用您的清单文件运行 registry 授权 playbook，使用上一步中的凭证授权您的节点：

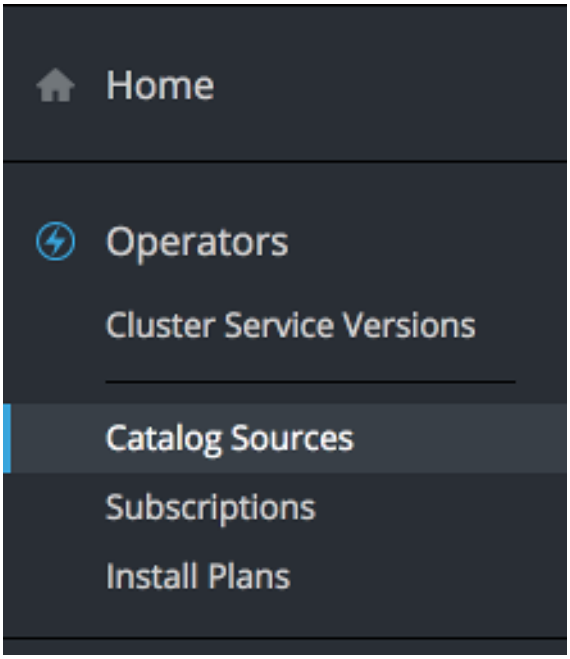
```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/updates/registry_auth.yml
```

3. 进入 playbook 目录并使用您的清单文件运行 OLM 安装 playbook：

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/olm/config.yml
```

4. 使用浏览器导航到集群的 Web 控制台。现在，在页面左侧的导航中应该有一个新部分：

图 41.2. 新的 Operator 导航部分



在这里，您可以安装 Operator，授予项目对它们的访问权限，然后为所有环境启动实例。

41.3. 启动第一个 OPERATOR

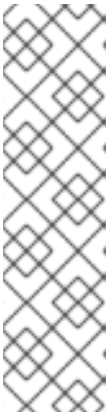
本节介绍了如何使用 Couchbase Operator 来创建新的 Couchbase 集群。

先决条件

- 启用了技术预览 OLM 的 OpenShift Container Platform 3.11
- 使用具有 **cluster-admin** 权限的账户访问该集群
- Couchbase Operator 加载到 Operator 目录（默认为以技术预览 OLM 加载）

流程

1. 作为集群管理员（具有 **cluster-admin** 角色的用户），在 OpenShift Container Platform Web 控制台中为此流程创建一个新项目。本例使用名为 **couchbase-test** 的项目。
2. 在项目内安装 Operator 通过 Subscription 对象来完成，集群管理员可在整个集群中创建并管理。要查看可用的 Subscriptions，请从下拉菜单中选择 **Cluster Console**，然后进入左侧导航中的 **Operators → Catalog Sources** 屏幕。



注意

如果要启用其他用户查看、创建和管理项目中的订阅，则必须具有该项目的 **admin** 和 **view** 角色，以及 **operator-lifecycle-manager** 项目的 **view** 角色。集群管理员可以使用以下命令添加这些角色：

```
$ oc policy add-role-to-user admin <user> -n <target_project>
$ oc policy add-role-to-user view <user> -n <target_project>
$ oc policy add-role-to-user view <user> -n operator-lifecycle-manager
```

以后的 OLM 版本中会简化这一体验。

3. 从 Web 控制台或 CLI 将所需的项目订阅到 Couchbase 目录源。
选择以下任一方法：

   - 对于 Web 控制台方法，请确保您正在查看所需项目，然后从此屏幕中，对一个 Operator 点 **Create Subscription**，将它安装到项目中。
   - 对于 CLI 方法，使用以下定义创建 YAML 文件：

     ***Couchbase-subscription.yaml* 文件**

     ```
     apiVersion: operators.coreos.com/v1alpha1
     kind: Subscription
     metadata:
       generateName: couchbase-enterprise-
       namespace: couchbase-test 1
     spec:
       source: certified-operators
       name: couchbase-enterprise
       startingCSV: couchbase-operator.v1.0.0
       channel: preview
     ```

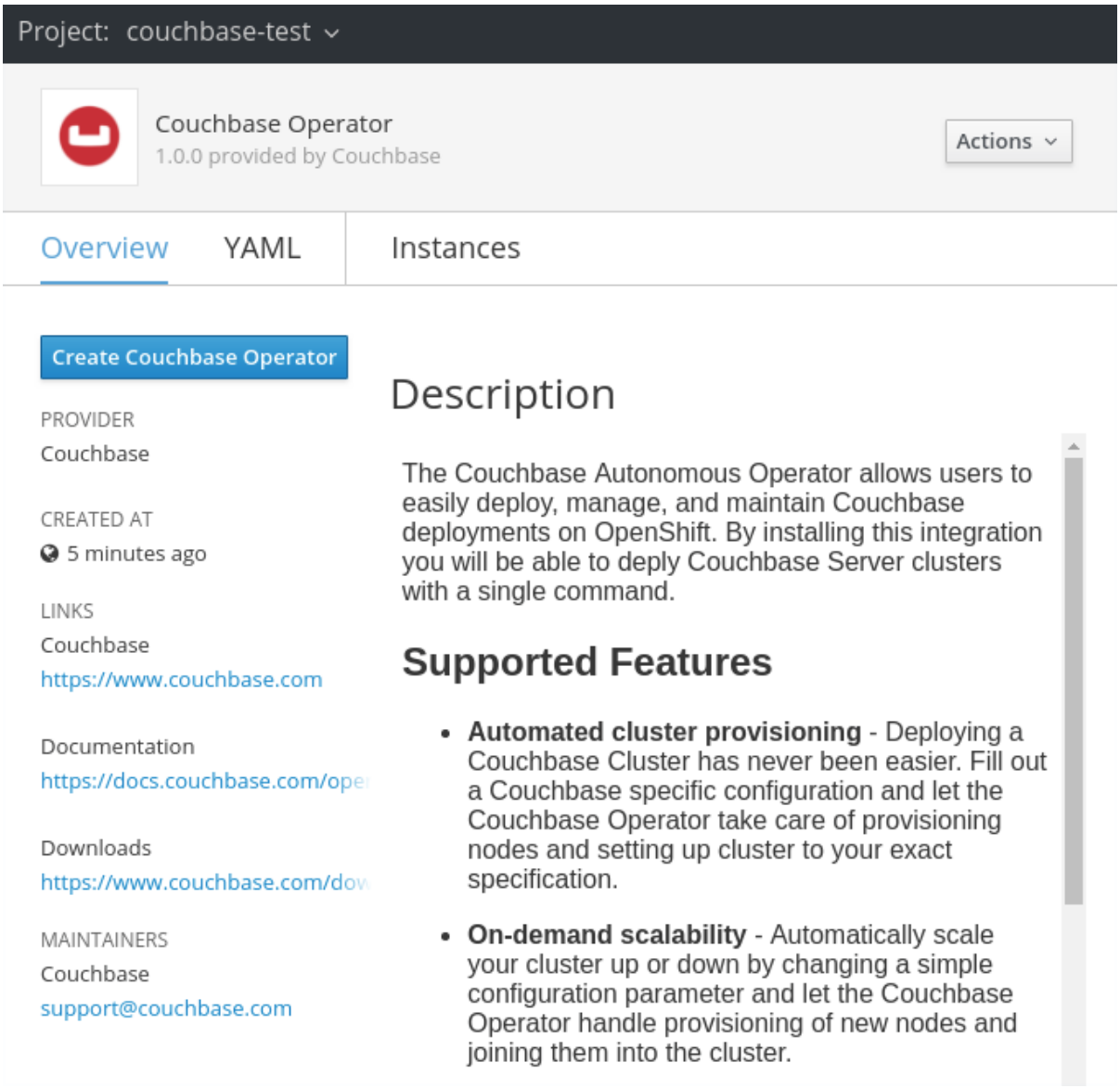
     1 确保 **metadata** 部分中的 **namespace** 字段设置为所需的项目。

     然后，使用 CLI 创建订阅：

     ```
     $ oc create -f couchbase-subscription.yaml
     ```

4. 创建订阅后，Operator 会出现在 **Cluster Service Versions** 屏幕中，目录用户可以使用它来启动 Operator 提供的软件。点击 Couchbase Operator 查看此 Operator 功能的更多详情：

图 41.3. Couchbase Operator 概述



5. 在创建 Couchbase 集群之前，请使用 Web 控制台或 CLI 创建含有超级用户帐户凭证的 Web 控制台或 CLI 的 secret。Operator 会在启动时读取它，并使用以下详情配置数据库：

Couchbase secret

```
apiVersion: v1
kind: Secret
metadata:
  name: couchbase-admin-creds
  namespace: couchbase-test (1)
type: Opaque
stringData:
  username: admin
  password: password
```

(1) 确保 **metadata** 部分中的 **namespace** 字段设置为所需的项目。

选择以下任一方法：

- 对于 Web 控制台方法，从左侧导航中点 **Workloads → Secrets**，然后点 **Create**，然后从 **YAML** 选择 **Secret** 以进入 secret 定义。
- 对于 CLI 方法，将 secret 定义保存到 YAML 文件（例如： ***couchbase-secret.yaml***）并使用 CLI 在所需项目中创建：

```
$ oc create -f couchbase-secret.yaml
```

6. 创建新的 Couchbase 集群。

注意

给定项目中具有 **edit** 角色的所有用户均可创建、管理和删除应用程序实例（本例中为 Couchbase 集群），由项目中已安装的 Operator 以自助服务方式管理，就像云服务一样。如果要启用具有此功能的其他用户，集群管理员可以使用以下命令添加角色：

```
$ oc policy add-role-to-user edit <user> -n <target_project>
```

a. 在 Web 控制台的 **Cluster Service Versions** 部分中，从 Operator 的 **Overview** 屏幕中单击 **Create Couchbase Operator**，以开始创建新的 **CouchbaseCluster** 对象。此对象是集群中 Operator 提供的新类型。对象的工作方式类似于内置的 **Deployment** 或 **ReplicaSet** 对象，但包含特定于管理 Couchbase 的逻辑。

提示

单击 **Create Couchbase Operator** 按钮时，您第一次可能会收到 404 错误。这是一个已知问题；作为临时解决方案，请刷新此页面以继续。(BZ#1609731)

Web 控制台包含一个最小的起始模板，但您可以参阅 Operator 支持的所有功能的 Couchbase 文档。

图 41.4. 创建 Couchbase 集群

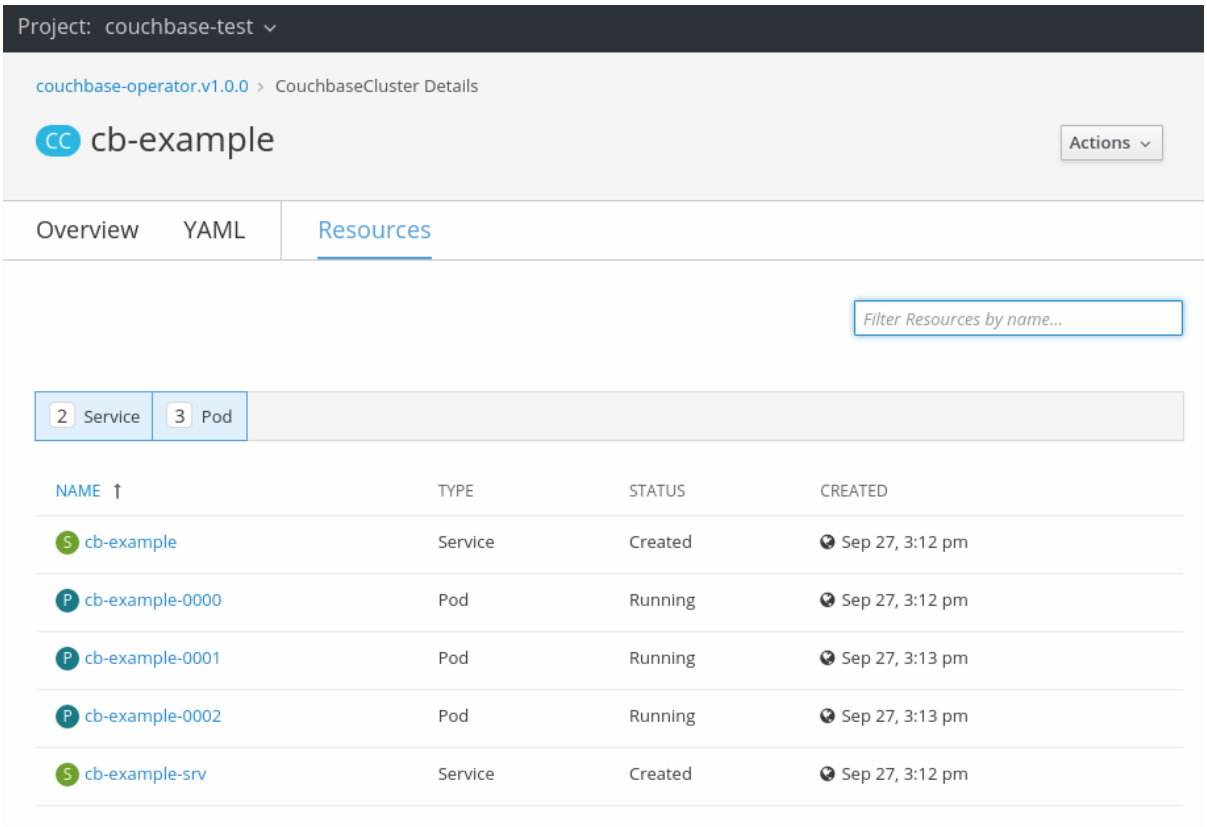
b. 确保配置包含 **admin** 凭证的 secret 名称：

```
apiVersion: couchbase.com/v1
kind: CouchbaseCluster
metadata:
  name: cb-example
  namespace: couchbase-test
spec:
  authSecret: couchbase-admin-creds
  baseImage: registry.connect.redhat.com/couchbase/server
  [...]
```

c. 完成对象定义后，点 web 控制台中的 **Create** （或使用 CLI）创建对象。这会触发 Operator 启动 pod、服务和 Couchbase 集群的其他组件。

7. 您的项目现在包含很多由 Operator 自动创建和配置的资源：

图 41.5. Couchbase 集群详情



点 **Resources** 选项卡，以验证已创建了支持您从项目中的其他 pod 访问数据库的 Kubernetes 服务。

使用 **cb-example** 服务，您可以使用 secret 中保存的凭据来连接数据库。其他应用容器集可以挂载并使用此 secret 并与服务进行通信。

现在，您有容错安装 Couchbase，它会对故障做出反应并重新平衡数据，因为 pod 变得不健康，或在集群中的节点之间迁移。最重要的是，集群管理员或开发人员可以通过提供高级别配置来轻松获取此数据库集群；不需要深入了解 Couchbase 集群或故障转移的细微知识。

参阅 官方 Couchbase 文档，了解更多有关 Couchbase Autonomous Operator 的功能。

41.4. 参与

OpenShift 团队希望了解您使用 Operator Framework 和您想要了解作为 Operator 提供的服务的建议。

通过电子邮件联系 openshift-operators@redhat.com 与团队联系。

第 42 章 卸载 OPERATOR LIFECYCLE MANAGER

安装集群后，您可以使用 OpenShift Container Platform **openshift-ansible** 安装程序卸载 Operator Lifecycle Manager。

42.1. 使用 ANSIBLE 卸载 OPERATOR LIFECYCLE MANAGER

安装集群后，您可以在 OpenShift Container Platform **openshift-ansible** 安装程序中使用此流程卸载技术预览 Operator Framework。

在卸载技术预览 Operator Framework 前，您必须检查以下先决条件：

- 现有 OpenShift Container Platform 3.11 集群
- 使用具有 **cluster-admin** 权限的账户访问该集群
- 最新 **openshift-ansible** 安装程序提供的 Ansible playbook

1. 在 **config.yml** playbook 中添加以下变量：

```
operator_lifecycle_manager_install=false
operator_lifecycle_manager_remove=true
```

2. 进入 playbook 目录：

```
$ cd /usr/share/ansible/openshift-ansible
```

3. 运行 OLM 安装 playbook，使用清单文件卸载 OLM：

```
$ ansible-playbook -i <inventory_file> playbooks/olm/config.yml
```