简介

Welcome

DO280：红帽 OpenShift 是实验为基础，动手实践课程
指导系统管理员如何安装、配置和管理红帽 OpenShift 容器平台集群
OpenShift 是一个容器化的应用平台，供企业用来管理容器器部署以及缩放使用 kubernetes 的应用
OpenShift 提供了预定义应用环境并在 kubernetes 基础上构建，帮助满足 DevOps 原则
如缩短面市时间、基础架构即代码、持续集成（CI）和持续交付（CD）等

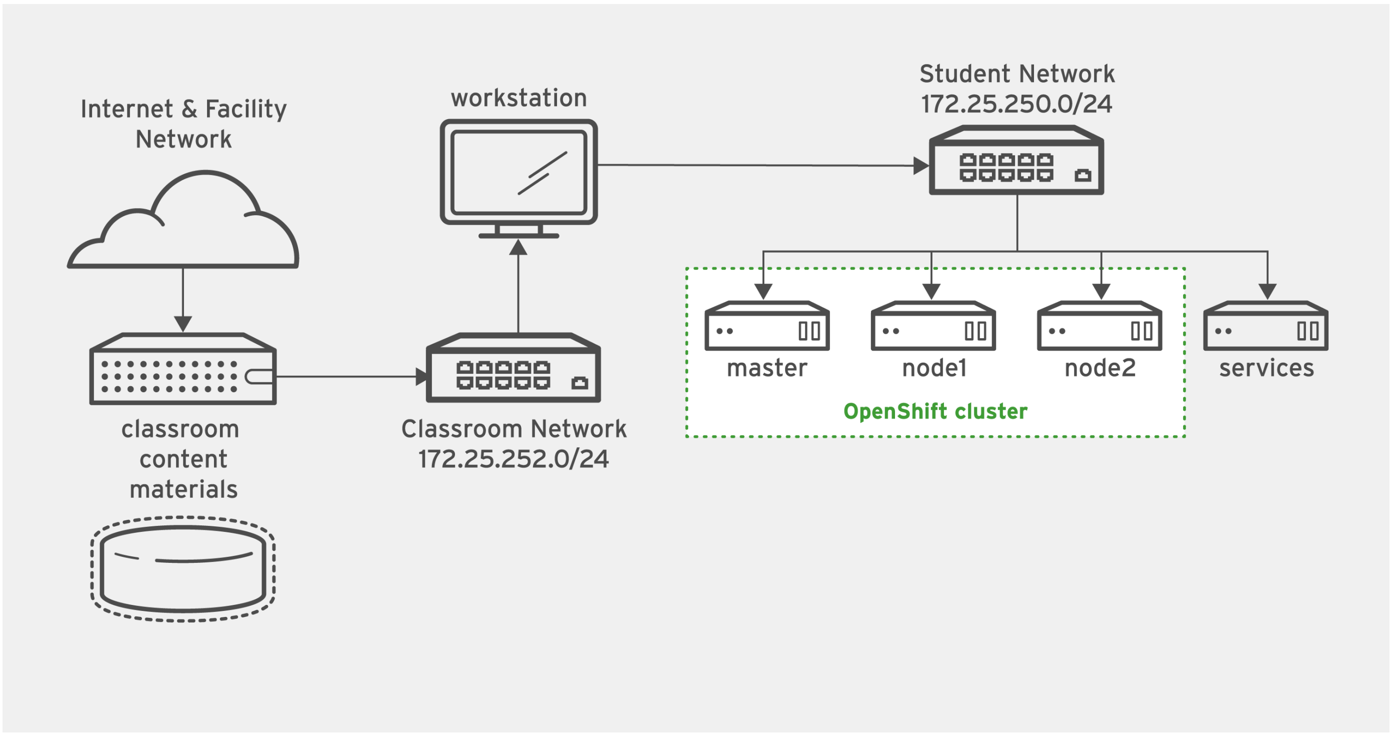
Course Objectives and Structure

安装、配置、监控和管理 OpenShift 集群
安装、配置和管理 OpenShift 集群的持久存储
利用 Source-to-Image(S2I) 构建，在 OpenShift 集群上部署应用

Schedule

第一天	第二天	第三天
红帽 OpenShift 容器平台简介	执行命令	管理应用部署
安装 OpenShift 容器平台	控制 OpenShift 资源访问	安装和配置指标子系统
描述和探索 OpenShift 网络概念	分配持久存储	管理和监控 OpenShift 容器平台
执行命令	管理应用应用布署	总复习

Orientation to the Classroom Lab Environment


Internet & Facility Network
workstation
Student Network 172.25.250.0/24
classroom content materials
Classroom Network 172.25.252.0/24
master
node1
node2
services
OpenShift cluster

- 课堂计算机

SOFT	计算机名称	IP 地址	角色
VMware	foundation	172.25.254.250	平台
KVM	classroom (materials, content)	172.25.254.254	实用工具服务器
KVM	workstation	172.25.250.254	图形工作站
KVM	master	172.25.250.10	OpenShift 容器平台 cluster 服务器
KVM	node1 node2	172.25.250.11 172.25.250.12	OpenShift 容器平台 cluster 节点
KVM	services registry	172.25.250.13	Classroom private registry

- 系统和应用凭据

计算机名称	特权用户	普通用户
foundation	root%Asimov	kiosk%redhat
classroom	root%Asimov	instructor%redhat
workstation, master, node1, node2, services	root%redhat	student%student
OpenShift web console	admin%redhat	developer%redhat

- 实验练习配置和判分

[workstation]

```
$ lab SCRIPT setup
$ lab SCRIPT grade
$ lab SCRIPT cleanup
```

- rht-vmctl 命令

命令	操作
$ rht-vmctl start classroom $ rht-vmctl start all	启动 虚拟机
$ rht-vmctl status classroom $ rht-vmctl status all	确认 虚拟机 状态
F8$ rht-vmview view workstation F7$ rht-vmctl view workstation	查看 虚拟机 物理控制台
$ rht-vmctl reset master	重置 虚拟机

Internationalization

建议：默认英文

```
$ localectl status
$ echo $LANG

$ localectl list-locales | grep CN
$ LANG=zh_CN.utf8 date
```

1. 红帽 OpenShift 容器平台简介

说明 OpenShift 容器平台功能

红帽 OpenShift 容器平台是一种容器应用平台，它为(**dev**eloper)开发人员和(**op**erater) IT组织提供云应用平台，以最少的配置和管理开销在安全的可扩展资源上部署新应用。

OpenShift 构建于红帽企业 Linux、Docker和Kubernetes 基础上，为当今的企业级应用提供安全的可扩展多租房操作系统，同时提供集成的应用运行时和库。OpenShift 为客户数据中心带来稳健、灵活且可扩展的容器平台，让企业能够部署满足安全性、隐私性、合规性和监管要求的平台。

客户如果不希望自己管理 OpenShift 集群，可以使用红帽提供的公共云平台，即红帽 OpenShift Online。OpenShift 容器平台和 OpenShift Online 都基于 OpenShift Origin 开源软件项目，后者则构建于 Docker 和 kubernetes 等许多其他开源项目基础之上。

应用作为容器运行，后者是单一操作系统内相互隔离的分区。容器提供许多与虚拟机相同的益处，如安全、存储和网络隔离等，但要求的资源要少得多，而且启动和终止的速度也更快。利用 OpenShift 提供的容器有助于提升平台本身以及其托管的应用的效率、弹性和可移植性。

下方列出 OpenShift 的主要功能：

- 自助服务平台：
  - OpenShift 允许开发人员利用 Source-to-Image(S2I)，从模板或自己的源代码管理存储库创建应用。
  - 系统管理员可以为用户和项目定义资源配额和限值来控制对系统资源的使用。
- 多语言支持：
  - OpenShift 支持 Java、Node.js、PHP、Perl 和直接用红帽的 Ruby，以及来自合作伙伴和广大 Docker 社区的许多其他语言。
  - 支持 MySQL、PostgreSQL和 MongoDB 数据库，包括直接来自红帽公司，以及来自合作伙伴和 Docker 社区的数据库。
  - 红帽还支持在 OpenShift 上原生运行 Apache httpd、Apache Tomcat、JBoss EAP、ActiveMQ 和 Fuse 等中间件产品。
- 自动化：
  - OpenShift 提供应用生命周期管理功能，以便在上游源或容器镜像更改时自动重新构建和重新部署容器。
  - 基于调试和策略扩展和故障切换应用。
  - 组合从独立组件或服务构建的复合应用。
- 用户界面：
  - OpenShift 提供 **Web UI** 来部署和监控应用，还提供 **CLI** 来远程管理应用和资源。
  - 它支持 Eclipse IDE 和 JBoss Developer Studio 插件，让开发人员能够继续使用熟悉的工具，同时也支持通过 REST API 与第三方或企业内部工具集成。
- 协作：
  - OpenShift 允许您在组织内部或与广大社区共享项目和自定义运行时。
- 可缩放性和高可用性：
  - OpenShift 提供容器多租户，以及能够按需弹性处理流量增长的分布式应用平台。
  - 它提供了高可用性，让应用能够在物理机丢失等事件中在存活。
  - OpenShift 提供自动发现状态不良的容器和自动重新部署的功能。
- 容器的可移植性：
  - 在 OpenShift 中，利用标准的容器镜像打包应用和服务，并通过 Kubernetes 管理复合应用。
  - 这些镜像可以部署到在这些基础技术上构建的其他平台。
- 开源：
  - 无供应商锁定。
- 安全性：
  - OpenShift 提供利用 SELinux 的多层安全性、基于角色的访问控制 rbac，以及与 LDAP 和 OAuth 等外部身份验证系统集成的功能。

- 动态存储管理：
  - OpenShift 利用 Kubernetes 的持久卷 pv 和持久卷声明 pvc 概念为容器数据提供静态和动态存储管理。
- 选择云（或非云）：
  - 将 OpenShift 容器平台部署到裸机服务器、来自不同供应商的虚拟机监控程序，以及大多数 IaaS 云提供商。
- 企业级：
  - 红帽提供对 OpenShift 、精选容器镜像和应用运行时的支持。
  - 红帽为可信的第三方容器镜像、运行时和应用提供认证。
  - 你可以利用 OpenShift 提供的高可用性，在强化而安全的环境中运行企业内部或第三方应用。
- 日志聚合和指标：
  - 可以在一个中央位置收集、聚合和分析来自 OpenShift 中部署的应用的日志信息。
  - OpenShift 让你能够实时收集与应用相关的指标和运行时信息，帮助你不断优化性能。

OpenShift 是微服务架构的驱动者，同时也支持更为传统的工作负载。许多组织还会发现，OpenShift 原生功能足以实现 Devops 流程，而且它能够与标准和自定义持续集成/持续部署工具轻松集成。

测验: OpenShift 容器平台功能

选择以下问题的正确答案：

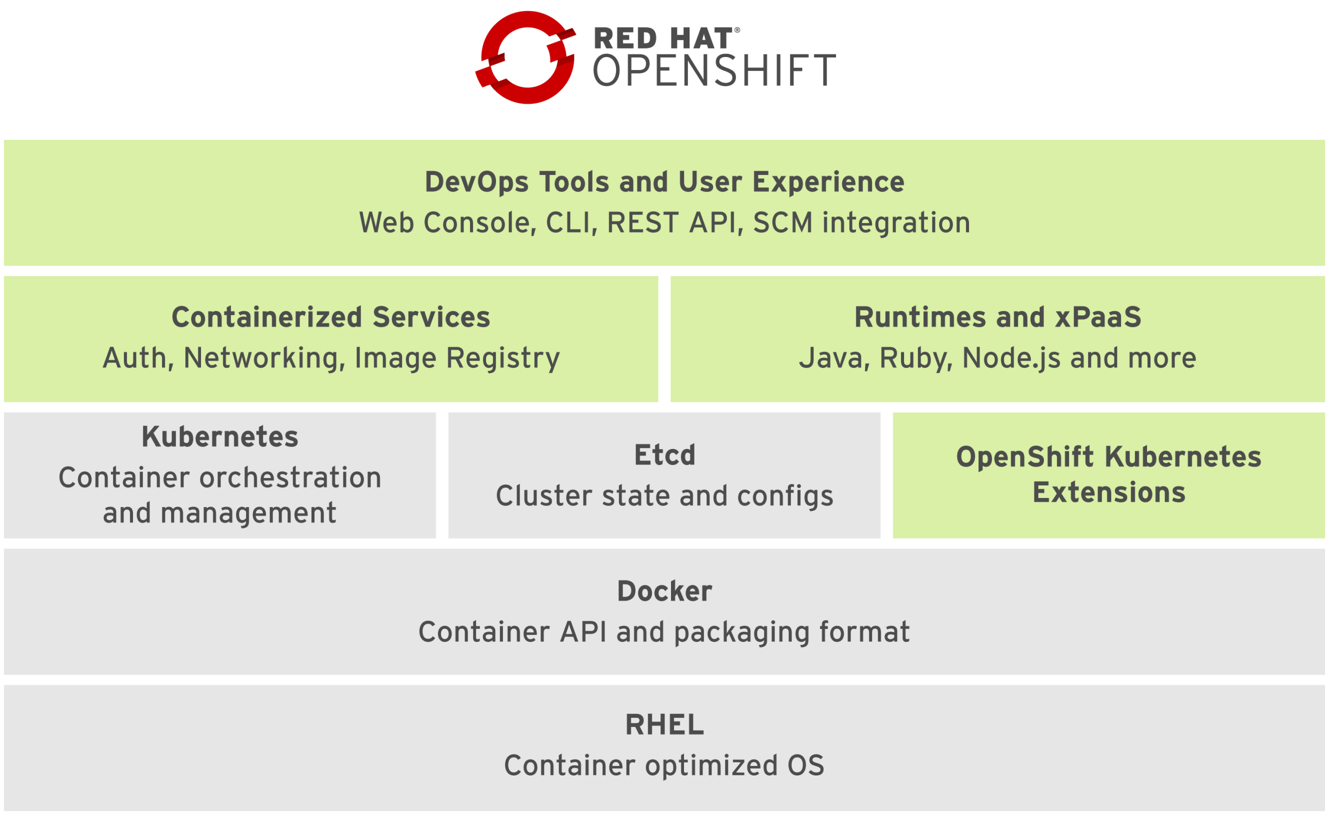
1. 以下关于 OpenShift 的陈述中哪两项正确？（请选择两项）
   a. 应用在 OpenShift 中作为虚拟机运行。虚拟机为应用提供安全性、存储和网络隔离
   b. 应用在 OpenShift 中作为容器运行。容器为应用提供安全性、存储和网络隔离
   c. OpenShift 采用专有的应用打包和部署格式，该格式无法移动且只能在 OpenShift 中使用
   d. 应用和服务使用标准的容器镜像打包，这些容器镜像可以部署到其他平台
2. 以下关于 OpenShift 的陈述中哪三项正确？（请选择三项）
   a. 它只能在裸机物理服务器上运行
   b. 它为许多常见的应用运行时提供经认证的容器镜像
   c. 开发人员可以直接从源代码存储库创建和启动云应用
   d. 它允许通过 REST API 与第三方工具轻松集成
   e. 只有基于 RHEL的容器才能在 OpenShift 中运行
   f. 它基于仅面向红帽订阅者提供的专有代码
3. 以下哪四种环境支持 OpenShift 部署？（请选择四项）
   a. 运行 RHEL 7 的裸机服务器
   b. 运行 Windows Server 的裸机服务器
   c. 常见的公共 IaaS 云提供商
   d. 常见的私有 IaaS 云环境
   e. 常见的公共 PaaS 云提供商
   f. 由常见虚拟机监控程序托管的虚拟服务器

4. 以下关于 OpenShift 的陈述中哪两项正确？（请选择两项）
   a. OpenShift 中仅支持基于 Java 的应用
   b. 您可以在 OpenShift 中部署 Wordpress 博客软件（Wordpress 构建于Apache、MySQL和PHP基础之上）
   c. 不支持 NoSQL 数据库
   d. 支持 MongoDB 等 NoSQL 数据库
5. 以下关于 OpenShift 高可用性和缩放能力的陈述中哪两项正确（请选择两项）
   a. 默认情况下不提供高可用性。您需要使用第三方高可用性产品
   b. 默认情况下提供高可用性
   c. 高可用性和缩放能力仅限于基于Java 的应用
   d. OpenShift 可以按需向上和向下扩展
   e. OpenShift 无法自动向上或向下扩展。管理员必须停止集群，再手动缩放应用

说明 OpenShift 容器平台架构

- Overview of OpenShift Container Platform Architecture

OpenShift 容器平台是构建于红帽企业 Linux、Docker和 Kubernetes 基础上的一组模块化组件和服务。OpenShift 为开发人员添加的功能包括远程管理、多租户、安全性增强、应用生命周期管理和自助服务接口。下图演示了 Openshift 软件堆栈：


RED HAT OPENSHIFT
DevOps Tools and User Experience
Web Console, CLI, REST API, SCM integration
Containerized Services
Auth, Networking, Image Registry
Runtimes and xPaaS
Java, Ruby, Node.js and more
Kubernetes
Container orchestration and management
Etcd
Cluster state and configs
OpenShift Kubernetes Extensions
Docker
Container API and packaging format
RHEL
Container optimized OS

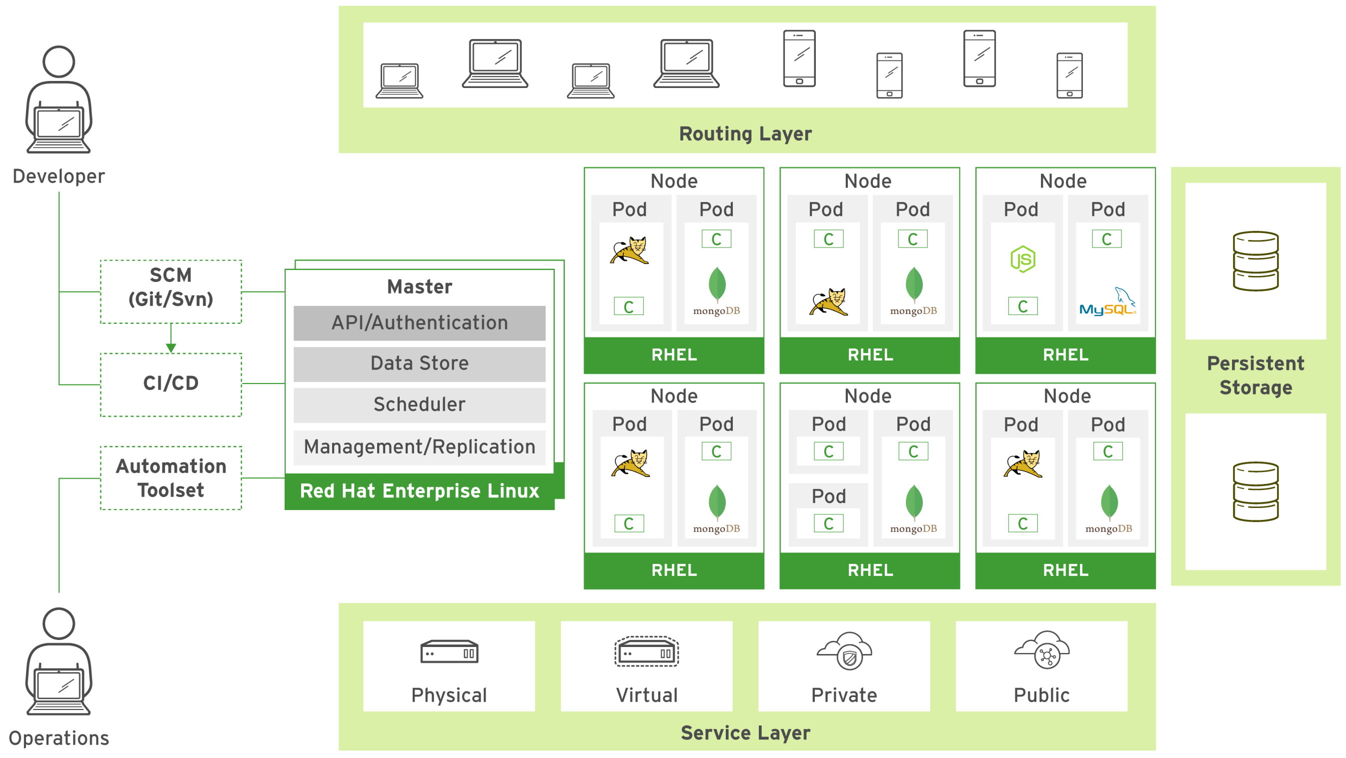
注意

直到最近，Docker 社区不具有支持将复合应用作为多个互联容器运行的功能，而这是传统分层企业应用和新型微服务基础所需要的。该社区启动了 Docker Swarm 项目来填补这一空缺，但 Kubernetes 已经是满足此需求的常见选择。 Kubernetes 已被部署到现实生产环境中，每天管理着超过20亿个 Docker 容器。

- Master and Nodes

OpenShift 集群是一组节点服务器，它们运行容器并由一组主控机服务器管理。服务器可以同时充当主控机和节点，但这两种角色通常会分隔以增强稳定性。

OpenShift 软件堆栈展现了组成 OpenShift 的软件包的一个静态透视图，下图展示了 OpenShift 工作方式的动态视图：



- OpenShift Projects and Applications

除了 Pod 和服务等 Kubernetes 资源外， OpenShift 还管理项目和用户。项目对 Kubernets 资源进行分组，以便将访问权限分配给用户。也可以为项目分配配额，限制其定义的 Pod、卷、服务和其他资源的数量。

OpenShift 中没有应用的概念。OpenShift 客户端提供 **new-app** 命令。此命令在项目内创建资源，但它们都不是应用资源。此命令是一种快捷方式，用于利用常见资源配置项目以形成标准开发工作流。OpenShift 使用标签来分类集群中的资源。默认情况下，OpenShift 使用 app 标签将相关的资源组合成一个应用。

- Building Images with Source-to-Image

开发人员和系统管理员可以将普通的 Docker 和 Kubernetes 工作流用于 OpenShift，但这要求他们了解如何构建容器镜像文件，操作注册表，以及使用其他低级别功能。OpenShift 允许开发人员利用标准的源代码控制台管理（SCM）存储库和集成的开发环境（IDE）。

OpenShift 中的 Source-to-Image(S2I) 流程从 SCM 存储库提取代码，自动检测源代码需要的运行时种类，并从专用于该运行时种类的基础镜像启动 Pod。在这个 Pod 内，OpenShift 像开发人员一样构建应用（例如，运行 maven 来构建 Java 应用）。如果构建成功，则创建另一个镜像，在应用的运行时上对应用二进制文件进行分层；此镜像推送到 OpenShift 内部的镜像注册表。然后，可以从镜像创建新的 Pod 来运行应用。S2I 可以视为 OpenShift 中已内建的完事 CI/CD 管道。

- Managing OpenShift Resources

`image` 镜像、 `docker` 容器、 `Pod` 、 `service` 服务、 `build` 构建器和 `template` 模板等 OpenShift 资源存储在 Etcd 中，可以通过 OpenShift CLI、Web 控制台或 REST API 进行管理。这些资源可以作为 JSON 或 YAML 文本文件查看，并在 `Git` 或 Subversion 等 SCM 检索这些资源定义。

大部分 OpenShift 操作都不是强制性的。OpenShift 命令和 API 调用不要求立即执行某一项操作。OpenShift 命令和 API 通常创建或修改存储在 Etcd 中的资源描述。 Etcd 随后通知 OpenShift 控制器，提醒这些资源的变化。这些控制器采取操作，使得云状态最终反映出变化。

警告

虽然 Docker 和 Kubernetes 是由 OpenShift 公开的，但开发人员和管理员应当主要使用 OpenShift CLI 和 OpenShift API 来管理应用和基础架构。OpenShift 添加了额外的安全和自动化功能，它们必须要手动配置，否则在直接使用 Docker 或 Kubernetes 命令和 API 时无法使用。对系统管理员而言，访问这些核心组件在故障排除期间具有重要价值。

- OpenShift Networking

Docker 联网非常简单。Docker 创建一个虚拟内核网桥，并将各个容器网络接口连接到其上。Docker 本身不提供将一个主机上的 Pod 和另一个主机上的 Pod 连接的方式。Docker也不提供向应用分配公共固定 IP 地址以便外部用户可以访问的途径。

Kubernetes 提供服务和路由资源，以管理 Pod 之间的网络可见性并且路由从外部世界到 Pod 的流量。服务在 Pod 之间平衡收到的网络请求负载，同时为该服务的所有客户端（通常是其它Pod）提供一个内部 IP 地址。容器和 Pod 不需要知道其他 Pod 的位置，它们只需要与服务连接。route 路由为服务提供固定的唯一 DNS 名称，使它对于 OpenShift 集群外部的客户端可见。

Kubernetes 服务和路由资源需要外部帮助来履行其职责。服务需要由软件定义型网络（SDN）提供不同主机上 Pod 之间的可见性，而路由需要通过某种方式将来自外部客户端的数据包转发或重定向到服务内部 IP。OpenShift 基于 **Open vSwitch** 提供 SDN，而路由则由一个分布式 **HAProxy** 提供。

- Persistent Storage

可能会随时出现 Pod 在一个节点上停止并在另一个节点上重启的情况。因此，普通的 Docker 存储由于默认具有临时性而不适合。如果数据库 Pod 被停止并在另一节点上重启，存储的任何数据将会丢失。

Kubernetes 提供用于为容器管理外部永久存储的框架。Kubenetes 识别 PersistentVolume 资源，该资源定义本地或网络存储。Pod 资源可以引用 `PersistentVolumeClaim` 资源，从而访问 `PersistentVolume` 中特定大小的存储。

Kubernetes 也指定 PersistentVolume 资源是否能在 Pod 之间共享，或者是否各个 Pod 需要独占访问的专用 PersistentVolume。当 Pod 移动到其他节点时，它会保持与相同 PersistentVolumeClaim 和 PersistentVolume 实例的连接。这意味着 Pod 的持久存储数据会跟随它，无论它被调度到哪一节点上运行。

OpenShift 为 Kubernetes 添加了大量 VolumeProvider，提供企业级存储的访问。如 **NFS**、**iSCSI**、**光纤通道**、**Gluster** 或 **OpenStack Cinder** 等云块存储卷服务。

OpenShift 还通过 `StorageClass` 资源为应用存储提供动态调配。使用动态存储时，您可选择不同类型的后端存储。后端存储划分到不同的“层”中，具体取决于您应用的需求。例如集群管理员可以使用名称“fast”定义一个 StorageClass 来利用较高质量的后端存储，同时定义另一个名为“slow”的 StorageClass 来提供商用级存储。在请求存储时，最终用户可以通过标注来指定 PersistentVolumeClaim，该标注将指定他们首选的 StorageClass 的值。

- OpenShift High Availability

OpenShift 容器平台集群的高可用性（HA）具有两个不同的方面：OpenShift 基础架构本身（即主控机）的 HA，以及 OpenShift 集群中运行的应用的 HA。

默认情况下，OpenShift 为主控机提供全面支持的原生 HA 机制。

对于应用或“Pod”，Kubernetes 默认负责对此进行处理。如果某一 Pod 因任何原因而丢失，Kubernetes 将调试另一个副本，并将它连接到服务层和持久存储。如果整个节点都丢失，Kubernetes 将为他的所有 Pod 调试替代项，最终所有应用都能重新可用。Pod 内的应用对自己的状态负责：因此，它们需要自行维护应用状态（例如，通过运用 HTTP 会话复制或数据库复制等可靠技术）。

- Image Streams

若要在 OpenShift 中创建新应用，除了应用源代码外，还需要基础镜像（S2I构建镜像）。这两个组件中有任何一个更新时，会创建新的容器镜像。使用旧容器镜像创建的 Pod 将被使用新镜像创建的 Pod 取代。

应用代码更改时，很明显需要更新容器镜像，但构建器镜像更改时，可能不容易看出也需要更新部署的 Pod。

镜像流由任意数量的容器镜像组成，它们通过标签来标识。它提供相关镜像的单一虚拟视图。应用参照镜像流进行构建。镜像流可用于在新镜像创建时自动执行操作。构建和部署可以监控镜像流，在添加新镜像时获得通过并分别通过执行构建或部署来响应。OpenShift 默认提供了几个镜像流，其中包含了许多常用语言运行时和框架。

镜像流标签是一种指向镜像流内某一镜像的别名。它通常简写为 `istag` 。它包含一个镜像历史记录表示为该标签曾经指向的所有镜像的堆栈。每当使用特定 istag 标记某一新的或现有的镜像时，它会被放在历史记录堆栈的第一位（标为 `latest` ）。之前 标为 latest 的镜像将放在第二位。这可方便回滚，使标签重新指向较旧的镜像。

练习: OpenShift 容器架构

描述	名称
存储 OpenShift 集群资源定义	Etcd
定义容器镜像格式	Docker
管理和调试 OpenShift 集群中的应用 Pod	Kubernetes
提供 JBoss 中间件认证容器镜像	xPaaS
在封闭的 Pod 内共享网络和存储配置	容器
运行 OpenShift REST API 、身份认证、调试程序和配置数据存储	Master 主控机
从源代码构建和部署应用	S2I
运行 Pod、kubelet 和代理	Node 节点
用于描述 OpenShift 集群资源的文件格式	JSON
平衡同一应用对复制的 Pod 的请求负载	Service 服务
为关系数据库等有状态应用提供持久存储	PersistentVolume
基于存储层为应用动态调配存储	StorageClass
相关容器镜像的集合的别名	Image Stream 镜像流
允许从外部网络访问应用	Route 路由
必须在同一节点上运行的容器集合	Pod
允许不同节点的 Pod 组成同一服务的软件定义型网络	Open vSwitch
可以分配有资源配额	Project 项目

总结

- 红帽 OpenShift 容器平台是一种基于红帽企业 Linux (RHEL)、容器和 Kubernetes 的容器应用平台
- OpenShift 容器平台使开发人员能够将精力放在源代码上，并依赖容器平台基础架构来构建和部署运行应用所需的容器
- OpenShift 架构利用主机服务器管理节点服务器，节点服务器将应用作为容器运行
- OpenShift 在默认的 Kubernetes 功能基础上，提供额外的身份验证、安全、调试、网络、存储、日志、指标和应用生命周期管理
- OpenShift 为主控机和 pods 提供内置的高可用性 (HA)

2. 安装 OpenShift 容器平台

准备服务器以进行安装

- 一般安装概述

红帽 OpenShift 容器平台由红帽公司以 RPM 软件包和容器镜像的组合形式交付。RPM 软件包可通过红由订阅下载，容器镜像则来自红帽私有容器注册表

OpenShift 容器平台安装需要多台服务器，它们可以是物理机和虚拟机的任意组合。其中一些称为主控机，另一些则为节点，分别需要不同的软件包和配置。为了使 OpenShift 集群引导更为方便，红帽提供了基于 Ansible 的安装程序，可以通过回答一系列的问题进行交互式运行，或者利用包含有环境配置详情的应答文件以自动化的非交互方式运行

在运行安装程序之前，系统管理员需要执行安装前任务：安装之后还需要执行安装后任务，以便获得功能完整的 OpenShift 容器平台集群

红帽为安装 OpenShift 容器平台提供了两种不同的方法。
  第一种方法，使用快速安装程序，它可用于简单的集群设置。
  第二种方法，设计用于更为复杂的安装，利用 Ansible Playbook 来自动化相关的流程

- 什么是 Ansible?

Ansible 是一种开源自动化平台，用于以一致的方式自定义和配置多台服务器。

- 安装 Ansible

  [kiosk@foundation]

```
ssh root@workstation yum install -y ansible
```

- Ansible Playbook 概述

```
---
- name: Install a File
  hosts: workstations
  vars:
    sample_content: "Hello World!"
  tasks:
  - name: "Copy a sample file to each workstation."
    copy:
      content: "{{ sample_content }}"
      dest: /tmp/sample.txt
- name: Hello OpenShift Enterprise v3.x
  hosts: OSEv3
  roles:
  - hello
```

- Ansible 主机清单文件

```
$ vim ./inventory
```

```
[workstations]
workstation.lab.example.com

[nfs]
services.lab.example.com

[masters]
master.lab.example.com

[etcd]
master.lab.example.com

[nodes]
master.lab.example.com hello_message="I am an OSEv3 master."
node1.lab.example.com
node2.lab.example.com

[OSEv3:children]
masters
etcd
nodes
nfs

[OSEv3:vars]
hello_message="I am an OSEv3 machine."

[workstations:vars]
sample_content="This is a workstation machine."
```

- 运行 Ansible Playbook

```
$ vim ansible.cfg
```

```
[defaults]
remote_user = student
inventory = ./inventory
roles_path = /home/student/do280-ansible/roles
log_path = ./ansible.log

[privilege_escalation]
become = yes
```

```
$ ansible-playbook <playbook-filename>
$ ansible-playbook -i <inventory-file> <playbook-filename>
```

- 准备环境

[foundation]

```
$ ssh student@master 'sudo whoami'
$ ssh student@node1 'sudo whoami'
$ ssh student@node2 'sudo whoami'
```

[master|node1|node2]

```
$ ping -c 4 master.lab.example.com
$ ping -c 4 node1.lab.example.com
$ ping -c 4 node2.lab.example.com
```

[master|node1|node2]

```
$ dig test.apps.lab.example.com
$ dig tes.apps.lab.example.com
$ dig te.apps.lab.example.com
```

OpenShift 高级安装还有一些附加需求。在当前培训环境中已经准备好了这些需求。需求列表如下：

- 每个 OpenShift 容器平台集群机器需要 RHEL 7.3， 7.4 或 7.5
- 每个 OpenShift 集群主机（包括 masters 和 nodes）使用红帽订阅管理（RHSM）注册，而不是 RHN。注册主机使用命令 subscription-manager register
- 每个主机附加可用的 OpenShift 容器平台订阅。附加主机订阅使用命令 **subscription-manager attach**
- 只有需要的仓库被启用。仓库（**rhel-7-server-rpms**，**rhel-7-server-extras-rpms**，**rhel-7-fast-datapath-rpms**，**rhel-7-server-ansible-2.4-rpms**）被启用。**rhel-7-server-ose-3.9-rpms** 仓库提供必要的 OpenShift 容器平台包。启用需要的仓库，使用命令 **subscription-manager repos --enable**。启用这些仓库在所有 OpenShift 集群中的主控和节点主机
- 在所有的 OpenShift 主机需要安装最基本的包： **wget**，**git**，**net-tools**，**bind-utils**，**yum-utils**，**iptables-services**，**bridge-utils**，**bash-completion**，**kexec-tools**，**sos**，**psacct**，**atomic-openshift-utils**。高级安装方式使用 playbooks，其他安装工具在包 **atomic-openshift-utils**
- docker 被安装和配置在每一个 OpenShift 主机。默认 Docker 在回环设备上使用瘦装配池存储容器镜像。红帽 OpenShift 集群产品， Docker 必须在逻辑卷上使用瘦装配池。使用命令 **docker-storage-setup** 给 Docker 配置默认的存储。红帽 OpenShift 文档，涵盖了在 OpenShift 主机上设置 Docker 存储的许多注意事项。

- 运行主机准备任务

一个 Ansible 剧本 **preprare_install.yml** 在教室环境中自动运行准备任务已经被提供。执行这个 playbook 以准备主机安装红帽 OpenShift 容器平台。

注意

prepare_install.yml 文件是专门为教室环境编写的自定义剧本。此剧本不包含在任何官方存储库或软件包中

引导式练习：准备安装

[student@workstation]

```
$ lab install-prepare setup

Setting up workstation for lab exercise work:
Downloading files for Workshop: Preparing for installation
 · Creating DO280 directory...................................  SUCCESS
 · Setting up labs folder.....................................  SUCCESS
 · Setting up solutions folder................................  SUCCESS
 · Downloading starter project................................  SUCCESS
 · Downloading solution project...............................  SUCCESS
Download successful.
 · Setting up lab files:......................................  SUCCESS

$ cd ~student/DO280/labs/install-prepare
```

```
$ sudo yum install -y ansible
$ ansible --version
ansible 2.4.3.0
  config file = /home/student/DO280/labs/install-prepare/ansible.cfg
  ...
$ cat /home/student/DO280/labs/install-prepare/ansible.cfg -n
```

```
[defaults]
remote_user = student
inventory = ./inventory
log_path = ./ansible.log

[privilege_escalation]
become = yes
become_user = root
become_method = sudo
```

```
$ cat ./inventory
```

```
[workstations]
workstation.lab.example.com

[nfs]
services.lab.example.com

[masters]
master.lab.example.com

[etcd]
master.lab.example.com

[nodes]
master.lab.example.com
```

```
node1.lab.example.com
node2.lab.example.com

[OSEv3:children]
masters
etcd
nodes
nfs

#Variables needed by the prepare_install.yml playbook.
[nodes:vars]
registry_local=registry.lab.example.com
use_overlay2_driver=true
insecure_registry=false
run_docker_offline=true
docker_storage_device=/dev/vdb
```

```
$ ansible-inventory --graph
@all:
  |--@OSEv3:
  |  |--@etcd:
  |  |  |--master.lab.example.com
  |  |--@masters:
  |  |  |--master.lab.example.com
  |  |--@nfs:
  |  |  |--services.lab.example.com
  |  |--@nodes:
  |  |  |--master.lab.example.com
  |  |  |--node1.lab.example.com
  |  |  |--node2.lab.example.com
  |--@ungrouped:
  |--@workstations:
  |  |--workstation.lab.example.com
$ cat ping.yml
```

```
---
- name: Verify Connectivity
  hosts: all
  gather_facts: no
  tasks:
    - name: "Test connectivity to machines."
      shell: "whoami"
      changed_when: false
```

```
$ ansible-playbook -v ping.yml
Using /home/student/DO280/labs/install-prepare/ansible.cfg as config file

PLAY [Verify Connectivity]
***************************************************************************

```

```
TASK [Test connectivity to machines.]
*****************************************************************************
ok: [services.lab.example.com] => {"changed": false, "cmd": "whoami", "delta": "0:00:00.032100", "end": "2020-02-17 06:47:20.500002", "rc": 0, "start": "2020-02-17 06:47:20.467902", "stderr": "", "stderr_lines": [], "stdout": "root", "stdout_lines": ["root"]}
ok: [workstation.lab.example.com] => {"changed": false, "cmd": "whoami", "delta": "0:00:00.054198", "end": "2020-02-17 06:47:20.594852", "rc": 0, "start": "2020-02-17 06:47:20.540654", "stderr": "", "stderr_lines": [], "stdout": "root", "stdout_lines": ["root"]}
ok: [node1.lab.example.com] => {"changed": false, "cmd": "whoami", "delta": "0:00:00.042519", "end": "2020-02-17 06:47:20.674222", "rc": 0, "start": "2020-02-17 06:47:20.631703", "stderr": "", "stderr_lines": [], "stdout": "root", "stdout_lines": ["root"]}
ok: [master.lab.example.com] => {"changed": false, "cmd": "whoami", "delta": "0:00:00.036040", "end": "2020-02-17 06:47:20.759704", "rc": 0, "start": "2020-02-17 06:47:20.723664", "stderr": "", "stderr_lines": [], "stdout": "root", "stdout_lines": ["root"]}
ok: [node2.lab.example.com] => {"changed": false, "cmd": "whoami", "delta": "0:00:00.017570", "end": "2020-02-17 06:47:20.760721", "rc": 0, "start": "2020-02-17 06:47:20.743151", "stderr": "", "stderr_lines": [], "stdout": "root", "stdout_lines": ["root"]}

PLAY RECAP *********************************************************************
master.lab.example.com     : ok=1    changed=0    unreachable=0    failed=0
node1.lab.example.com      : ok=1    changed=0    unreachable=0    failed=0
node2.lab.example.com      : ok=1    changed=0    unreachable=0    failed=0
services.lab.example.com   : ok=1    changed=0    unreachable=0    failed=0
workstation.lab.example.com : ok=1    changed=0    unreachable=0    failed=0
```

```
$ cat prepare_install.yml
```

```
---
- name: "Host Preparation: Docker tasks"
  hosts: nodes
  roles:
    - docker-storage
    - docker-registry-cert
    - openshift-node

  # 上面的角色未处理下面的任务
  tasks:
    - name: Student Account - Docker Access
      user:
        name: student
        groups: docker
        append: yes
...
```

```
$ cat roles/docker-storage/tasks/main.yml
$ cat roles/docker-registry-cert/tasks/main.yml
$ cat roles/docker-registry-cert/vars/main.yml
$ cat roles/openshift-node/tasks/main.yml
```

```
$ ansible-playbook prepare_install.yml
...
PLAY RECAP ***************************************************************************
master.lab.example.com     : ok=28   changed=24   unreachable=0    failed=0
node1.lab.example.com      : ok=28   changed=24   unreachable=0    failed=0
node2.lab.example.com      : ok=28   changed=24   unreachable=0    failed=0
```

```
# 验证 docker
for vm in master node{1,2}; do
  echo -e "\n$vm:"
  ssh -o LogLevel=QUIET $vm sudo systemctl is-active docker
  ssh -o LogLevel=QUIET $vm sudo systemctl is-enabled docker
done
```

```
# 验证存储
for vm in master node{1,2}; do
  echo -e "\n$vm : df -h /var/lib/docker"
  ssh -o LogLevel=QUIET $vm sudo df -h | grep vg-docker
done
```

```
# 验证私有镜像仓库可用
for vm in master node{1,2}; do
  echo -e "\n$vm: "
  ssh -o LogLevel=QUIET $vm docker pull rhel7:latest
done
```

```
# 验证依赖包已安装
for vm in master node{1,2}; do
  echo -e "\n$vm"
  ssh -o LogLevel=QUIET $vm \
    rpm -q wget git net-tools bind-utils yum-utils iptables-services \
    bridge-utils bash-completion kexec-tools sos psacct \
    atomic-openshift-utils
done
```

安装红帽 OpenShift 容器平台

- 高级安装简介

准备好主机后，高级安装方法包含四步：

- 编写一个主机清单文件，来描述所需的集群特性和架构
- 执行 OpenShift **prerequisites.yml** 剧本

- 执行 OpenShift **deploy_cluster.yml** 剧本
- 确认安装

- 编写高级安装主机清单文件

```
$ vim ansible.cfg
```

```
[defaults]
remote_user = student
inventory = ./inventory
log_path = ./ansible.log

[privilege_escalation]
become = yes
become_user = root
become_method = sudo
```

```
$ vim ./inventory
```

```
[workstations]
workstation.lab.example.com

[nfs]
services.lab.example.com

[masters]
master.lab.example.com

[etcd]
master.lab.example.com

[nodes]
master.lab.example.com
node1.lab.example.com
node2.lab.example.com

[OSEv3:children]
masters
etcd
nodes
nfs
```

以上主机清单作为 OpenShift 高级安装的清单文件的起点。添加组和主机变量，以定义已安装群集的特性。在教室环境中，清单文件必须添加下列要求：

- 安装所需版本的 OpenShift 容器平台
- 用户使用 htpasswd 身份验证，对集群进行身份验证
- 通配符 DNS 条目 apps.lab.example.com，用作托管 OpenShift 应用程序的子域

    - nfs 存储用于 OpenShift etcd 服务和 OpenShift 内部注册表
    - 教室容器注册表用作外部注册表，因为没有连接到 **docer.io** 或 **registry.access.redhat.com**

  - 安装变量

    OpenShift 安装变量记录在清单的 **[OSEv3:vars]** 部分。安装变量用于配置许多 OpenShift 组件，例如：

      - 私有容器注册表
      - 使用 Gluster、Ceph或其他第三方云提供商的持久存储
      - 群集度量
      - 群集日志
      - 自定义群集证书

    本节仅介绍教室安装所需的变量。

    注意

    如果要在该类之外安装群集，请花时间研究并了解可用的选项和变量。有关详细信息，请参阅“参考”部分中列出的 Installation and Configuration Guide 的“高级安装”部分。

- 配置 OpenShift 安装版本

红帽建议系统管理员决定 OpenShift 的目标为主要版本，并允许安装行动手册采用该主要版本的最新次要版本。要指定要安装的 OpenShift 容器平台部署类型和版本，在 `[OSEv3:vars]` 部分中分别使用 `openshift_deployment_type` 和 `openshift_release` 变量。

```
openshift_deployment_type=openshift-enterprise
openshift_release=v3.9
```

教室 OpenShift 集群使用另外两个变量：

- 容器化的 OpenShift 服务使用标记为 `v3.9.14` 的镜像，这会阻止集群自动升级到更高版本的容器镜像
- 教室虚拟机不符合生产使用的推荐系统要求。OpenShift 剧本被设计为，在安装过程的早期一个节点不满足最低要求时失败。对于非生产群集，可以禁用对系统要求的检查。

```
openshift_image_tag=v3.9.14
openshift_disable_check=disk_availability,docker_storage,memory_availability
```

- **配置身份验证**

  OpenShift 容器平台身份验证基于 OAuth，它提供基于 HTTP 的 API， 用于交互式和非交互式客户端的身份验证。OpenShift 主控机在 OAuth 服务器上运行，而且 OpenShift 可以配置多个身份提供程序，它们可以和特定于组织的身份管理产品集成。支持的 OpenShift 身份提供程序有：

  - **HTTP Basic，委派至外部的单点登录（SSO）系统**
  - GitHub 和 GitLab，使用 GitHub 和 GitLab 帐户
  - OpenID Connect，使用兼容 OpenID 的 SSO 以及 Google 帐户
  - OpenStack Keystone v3 服务器
  - LDAP v3 服务器

  OpenShift 安装程序采用默认安全的方法，其中 DenyAllPasswordIdentityProvider 是默认的提供程序。使用此提供程序时，仅 master 主机上的本地 root 用户可以使用 OpenShift 客户端命令和 API。

  您必须配置另一个身份提供程序，以便外部用户可以访问 OpenShift 集群。

  - htpasswd身份验证

    OpenShift HTPasswdPasswordIdentityProvider 对照由 Apache HTTPD htpasswd 实用程序生成的平面文件验证用户和密码。这不是企业级身份管理，但对概念验证（POC）OpenShift 部署而言已经足够。

    在 Ansible 的主机清单中添加 `openshift_master_identity_providers` 变量：

    ```
    openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/htpasswd'}]
    ```

    要指定用户和密码的初始列表，请将 `openshift_master_htpasswd_users` 变量添加到主机清单文件中。请参阅以下示例：

    ```
    openshift_master_htpasswd_users="{'admin':'$apr1$.NHMsZYc$MdmfWN5DM3q280/w7c51c/','devops':'$apr1$.NHMsZYc$MdmfWN5DM3q280/w7c51c/'}"
    ```

    ```
    $ htpasswd -nb admin redhat
    or
    $ openssl passwd -apr1 redhat
    ```

- 配置网络要求

  - Wildcard DNS

基础结构节点的通配符 dns 条目允许自动将任何新创建的路由路由到子域下的群集。通配符 DNS 条目必须存在于唯一的子域中，如 `apps.lab.example.com` 。并解析为基础结构节点的主机名或IP地址。通配符 dns 项的主机清单文件中变量是 `openshift_master_default_subdomain`

```
openshift_master_default_subdomain=apps.lab.example.com
```

- Master Service Ports

  `openshift_master_api_port` 变量定义主 API 的侦听端口。尽管默认值是 8443，当使用专用主机作为主控时，你可以使用端口 443 并从连接URL中省略端口号。主控控制端口设置为 `openshift_master_console_port` 变量的值；默认端口为 8443。主控控制台也可以设置为使用端口 443，端口号可以从连接 URL 中省略。

- Firewalld

  OpenShift 节点上的默认防火墙服务是 iptables。要将 firewalld 用作所有节点上的防火墙服务，请将 `os_firewall_use_firewalld` 变量设置为 true

  ```
  os_firewall_use_firewalld=true
  ```

- 配置持久存储

容器用于提供一些 OpenShift 服务，例如 OpenShift 容器注册表。默认情况下，容器数据是短暂的，在容器被销毁时丢失。Kubernetes 持久卷框架为容器请求和使用持久存储提供了一种机制。为了避免数据丢失，这些服务被配置为使用持久卷。

在这个教室，OpenShift 容器注册表和 OpenShift Anible 代理服务被配置为使用 NFS 持久存储。

注意

生产 OpenShift 群集不支持 NFS 永久存储。要允许非生产群集上的 NFS 持久存储，请添加 **openshift_enable_unsupported_configurations=true** 到主机清单文件中。

- OpenShift Container Registry

  ```
  openshift_hosted_registry_storage_kind=nfs
  openshift_hosted_registry_storage_nfs_directory=/exports
  openshift_hosted_registry_storage_volume_name=registry
  openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
  openshift_hosted_registry_storage_volume_size=40Gi
  openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
  ```

- OpenShift Ansible Broker

  > OpenShift Ansible 代理（OAB）是一个容器化的 OpenShift 服务，它部署自己的 **`etcd`** 服务。持久化 Etcd 存储所需的变量与注册表所需的变量类似：

  ```
  openshift_hosted_etcd_storage_kind=nfs
  openshift_hosted_etcd_storage_nfs_directory=/exports
  openshift_hosted_etcd_storage_volume_name=etcd-vol2
  openshift_hosted_etcd_storage_nfs_options="*(rw,root_squash,sync,no_delay)"
  openshift_hosted_etcd_storage_volume_size=1G
  openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
  openshift_hosted_etcd_storage_labels={'storage': 'etcd'}
  ```

- Configuring a Disconnected OpenShift Cluster

> 默认情况下，OpenShift 安装行动手册假定来自集群的 internet 连接。当需要 RPM 或容器映像时，可以从外部源（如access.redhat.com）下载该映像。没有连接到这些外部资源的群集称为断开连接的群集或断开连接的安装。教室OpenShift 集群是一个断开连接的安装，因为没有互联网连接。
>
> 在教室环境，RPM 软件包在主机 http://content.example.com 。合适的仓库存在于所有 OpenShift 节点的 **`/etc/yum.repos.d/training.repo`**

- Configuring a Different Registry

  ```
  #Modifications Needed for a Disconnected Install
  oreg_url=registry.lab.example.com/openshift3/ose-${component}:${version}
  openshift_examples_modify_imagestreams=true
  openshift_docker_additional_registries=registry.lab.example.com
  openshift_docker_blocked_registries=registry.access.redhat.com,docker.io

  #Image Prefixes Modifications
  openshift_web_console_prefix=registry.lab.example.com/openshift3/ose-
  openshift_cockpit_deployer_prefix='registry.lab.example.com/openshift3/'
  openshift_service_catalog_image_prefix=registry.lab.example.com/openshift3/ose-
  template_service_broker_prefix=registry.lab.example.com/openshift3/ose-
  ansible_service_broker_image_prefix=registry.lab.example.com/openshift3/ose-
  ansible_service_broker_etcd_image_prefix=registry.lab.example.com/rhel7/
  ```

- Configuring Node Labels

> 节点标签是分配给每个节点的任意键/值元数据对。节点标签通常用于区分地理数据中心或标识节点上的可用资源。应用程序可以在其部署配置中以节点标签的形式声明节点选择器。如果存在，应用程序的 pods 必须部署在具有匹配节点标签的节点上。节点标签是在清单文件中使用主机变量 **`openshift_node_labels`** 设置的。
>
> OpenShift 集群的一个常见架构模式是区分 **`master`** 主节点、 **`infra`** 基础结构节点和 **`compute`** 计算节点。在该模式中，基础设施节点托管 OpenShift 托管的注册表和路由器的 pod，而计算节点托管来自用户项目的应用程序 pod。主节点不承载应用程序或基础结构 pod。使用节点标签标识特定节点的角色。

OpenShift基础设施服务的默认节点选择器是 `region=infra` 。承载基础设施 pod 的任何节点都必须具有 `region=infra` 的节点标签。
应用程序 pods 的默认节点选择器是 `node-role.kubernetes.io/compute=true` 。承载应用程序 pod 的任何节点都必须具有此节点标签。任何不是主节点或基础结构节点的节点都会在安装期间接收此节点标签。

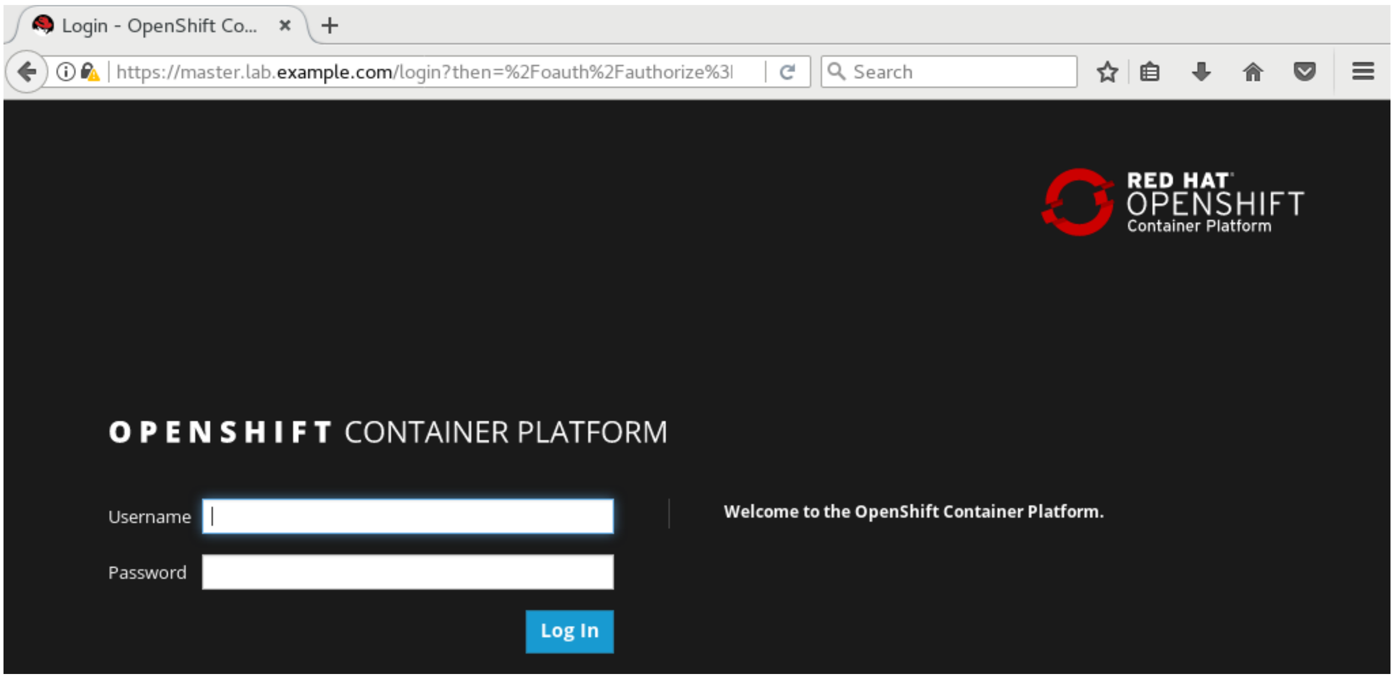
- Executing the OpenShift Playbooks

执行两个剧本来安装 OpenShift: `prerequisites.yml` 和 `deploy_cluster.yml` 。 `atomic-openshift-utils` 包提供了这些剧本和其他可移植的工件。在执行剧本的机器上安装这个包。

首先执行此行动手册，以确保满足所有 OpenShift 集群计算机的所有系统要求和先决条件。这个剧本试图修改和修复不满足 OpenShift 部署的必要先决条件的节点。

```
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- Verifying the Installation


Login - OpenShift Co...
https://master.lab.example.com/login?then=%2Foauth%2Fauthorize%3I
Search
RED HAT OPENSHIFT Container Platform
OPENSHIFT CONTAINER PLATFORM
Username
Password
Log In
Welcome to the OpenShift Container Platform.

引导式练习： 安装红帽 OpenShift 容器平台

[student@workstation]

```
$ cd
$ lab install-run setup
Setting up workstation for lab work:

Downloading files for GE: Running the Installer

```

```
  · Downloading starter project.................................  SUCCESS
  · Downloading solution project................................  SUCCESS

Download successful.

Downloading additional artifacts for the lab:

  · Downloading Ansible artifacts...............................  SUCCESS
  · Install 'crudini' if necessary..............................  SUCCESS

Setup successful.

$ cd DO280/labs/install-run
```

```
$ sudo yum install -y atomic-openshift-utils
$ cp inventory.initial inventory

$ vim general_vars.txt
```

```
...
[OSEv3:vars]
#General Variables
openshift_deployment_type=openshift-enterprise
openshift_release=v3.9
openshift_image_tag=v3.9.14
openshift_disable_check=disk_availability,docker_storage,memory_availability
```

```
$ openssl passwd -apr1 redhat
$ openssl passwd -apr1 redhat
$ vim authentication_vars.txt
```

```
#Cluster Authentication Variables
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename':
'/etc/origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin':'$apr1$elGra5BM$KKqzyJrsSOAfPMlWH2d9a.',
'developer':'$apr1$dtyByzYg$gi5sqkeCLgaECwKPyUtTD0'}

```

```
$ vim networking_vars.txt
```

```
#OpenShift Networking Variables
os_firewall_use_firewalld=true
openshift_master_api_port=443
openshift_master_console_port=443
openshift_master_default_subdomain=apps.lab.example.com

```

```
$ vim persistence_vars.txt
```

```
 #NFS is an unsupported configuration
```

```
openshift_enable_unsupported_configurations=true

#OCR configuration variables
openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=40Gi

#OAB's etcd configuration variables
openshift_hosted_etcd_storage_kind=nfs
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_etcd_storage_nfs_directory=/exports
openshift_hosted_etcd_storage_nfs_options="*(rw,root_squash,sync,no_wdelay)"
openshift_hosted_etcd_storage_volume_name=etcd-vol2
openshift_hosted_etcd_storage_volume_size=1G
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}

```

```
$ vim disconnected_vars.txt
```

```
#Modifications Needed for a Disconnected Install
oreg_url=registry.lab.example.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true
openshift_docker_additional_registries=registry.lab.example.com
openshift_docker_blocked_registries=registry.access.redhat.com,docker.io

#Image Prefixes
openshift_web_console_prefix=registry.lab.example.com/openshift3/ose-
openshift_cockpit_deployer_prefix='registry.lab.example.com/openshift3/'
openshift_service_catalog_image_prefix=registry.lab.example.com/openshift3/ose-
template_service_broker_prefix=registry.lab.example.com/openshift3/ose-
ansible_service_broker_image_prefix=registry.lab.example.com/openshift3/ose-
ansible_service_broker_etcd_image_prefix=registry.lab.example.com/rhel7/

```

```
$ vim inventory
```

```
...
[nodes]
master.lab.example.com
node1.lab.example.com openshift_node_labels="{'region':'infra', 'node-
role.kubernetes.io/compute':'true'}"
node2.lab.example.com openshift_node_labels="{'region':'infra', 'node-
role.kubernetes.io/compute':'true'}"
...
```

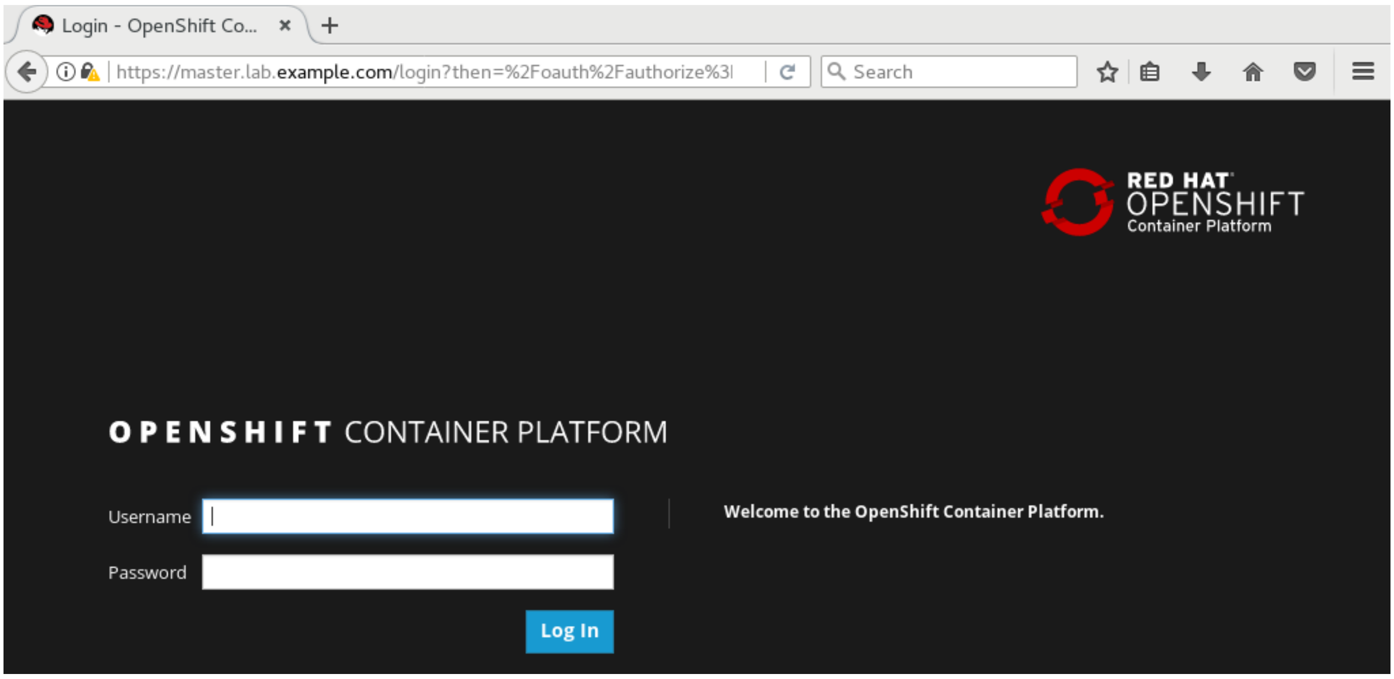
```
$ cat general_vars.txt networking_vars.txt authentication_vars.txt persistence_vars.txt
disconnected_vars.txt >> inventory
$ lab install-run grade
```

```

Checking the OpenShift Advanced Installation method inventory file

 · Detecting solution inventory...............................  PASS
 · Detecting student inventory................................  PASS

Comparing Entries in [OSEv3:children]

 · Checking masters...........................................  PASS
 · Checking etcd..............................................  PASS
 · Checking nodes.............................................  PASS
 · Checking nfs...............................................  PASS

Comparing Entries in [OSEv3:vars]

 · Checking openshift_disable_check...........................  PASS
 · Checking openshift_deployment_type.........................  PASS
 · Checking openshift_release.................................  PASS
 · Checking openshift_image_tag...............................  PASS
 · Checking os_firewall_use_firewalld.........................  PASS
 · Checking openshift_master_api_port.........................  PASS
 · Checking openshift_master_console_port.....................  PASS
 · Checking openshift_master_default_subdomain................  PASS
 · Checking openshift_master_identity_providers...............  PASS
 · Skipping openshift_master_htpasswd_users...................  PASS
 · Checking openshift_enable_unsupported_configurations.......  PASS
 · Checking openshift_hosted_registry_storage_kind............  PASS
 · Checking openshift_hosted_registry_storage_access_mode.....  PASS
 · Checking openshift_hosted_registry_storage_nfs_directo.....  PASS
 · Checking openshift_hosted_registry_storage_nfs_options.....  PASS
 · Checking openshift_hosted_registry_storage_volume_name.....  PASS
 · Checking openshift_hosted_registry_storage_volume_size.....  PASS
 · Checking openshift_hosted_etcd_storage_kind................  PASS
 · Checking openshift_hosted_etcd_storage_nfs_options.........  PASS
 · Checking openshift_hosted_etcd_storage_nfs_directory.......  PASS
 · Checking openshift_hosted_etcd_storage_volume_name.........  PASS
 · Checking openshift_hosted_etcd_storage_access_modes........  PASS
 · Checking openshift_hosted_etcd_storage_volume_size.........  PASS
 · Checking openshift_hosted_etcd_storage_labels..............  PASS
 · Checking oreg_url..........................................  PASS
 · Checking openshift_examples_modify_imagestreams............  PASS
 · Checking openshift_docker_additional_registries............  PASS
 · Checking openshift_docker_blocked_registries...............  PASS
 · Checking openshift_web_console_prefix......................  PASS
 · Checking openshift_cockpit_deployer_prefix.................  PASS
 · Checking openshift_service_catalog_image_prefix............  PASS
 · Checking template_service_broker_prefix....................  PASS
 · Checking ansible_service_broker_image_prefix...............  PASS
 · Checking ansible_service_broker_etcd_image_prefix..........  PASS

Comparing Entries in [etcd]

```

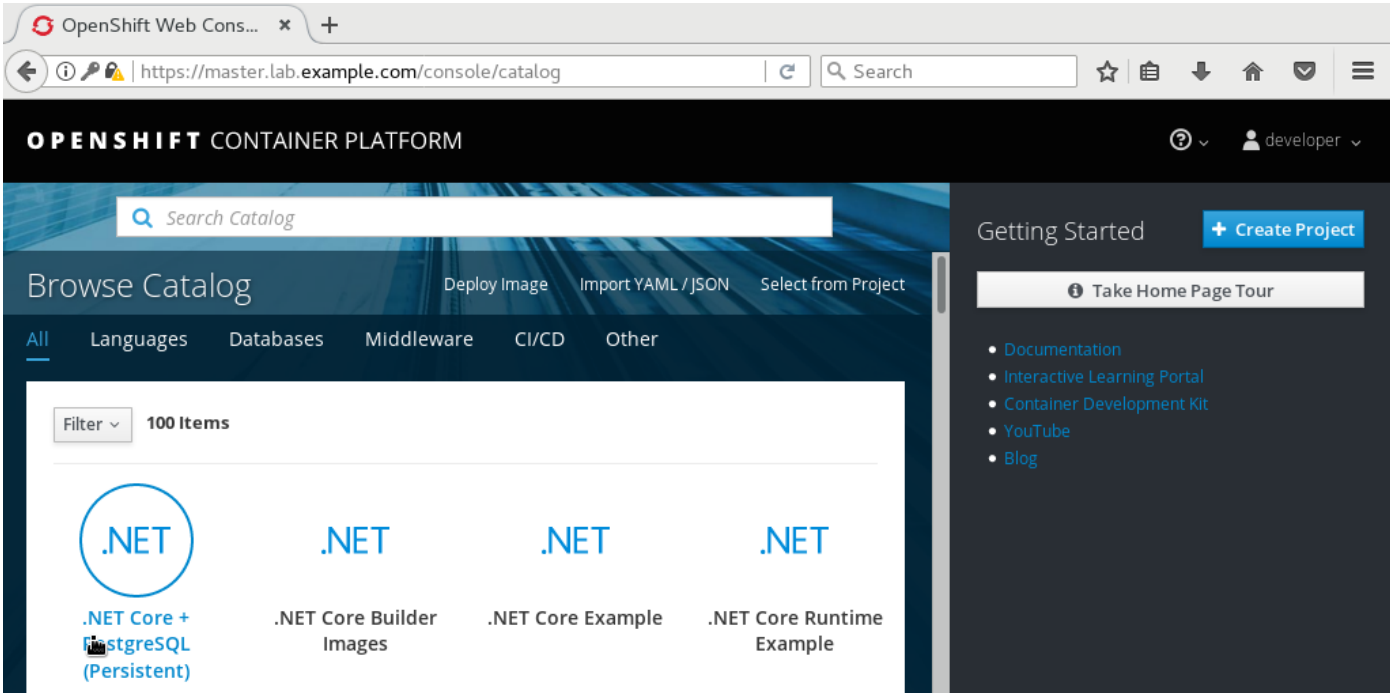
```
  · Checking master.lab.example.com.............................  PASS

Comparing Entries in [masters]

  · Checking master.lab.example.com.............................  PASS

Comparing Entries in [nfs]

  · Checking services.lab.example.com...........................  PASS

Comparing Entries in [nodes]

  · Checking master.lab.example.com.............................  PASS
  · Checking node1.lab.example.com openshift_node_labels........  PASS
  · Checking node2.lab.example.com openshift_node_labels........  PASS

Overall inventory file check: ................................  PASS

$ sudo yum install -y openshift-ansible-playbooks
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
...
PLAY RECAP ***********************************************
localhost                  : ok=12   changed=0    unreachable=0    failed=0
master.lab.example.com     : ok=67   changed=12   unreachable=0    failed=0
node1.lab.example.com      : ok=60   changed=12   unreachable=0    failed=0
node2.lab.example.com      : ok=60   changed=12   unreachable=0    failed=0
services.lab.example.com   : ok=36   changed=4    unreachable=0    failed=0
workstation.lab.example.com : ok=2    changed=0    unreachable=0    failed=0

INSTALLER STATUS ***********************************************
Initialization             : Complete (0:00:47)

$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
...
PLAY RECAP ***********************************************
localhost                  : ok=13   changed=0    unreachable=0    failed=0
master.lab.example.com     : ok=600  changed=250  unreachable=0    failed=0
node1.lab.example.com      : ok=133  changed=52   unreachable=0    failed=0
node2.lab.example.com      : ok=133  changed=51   unreachable=0    failed=0
services.lab.example.com   : ok=31   changed=8    unreachable=0    failed=0
workstation.lab.example.com : ok=21   changed=0    unreachable=0    failed=0

INSTALLER STATUS ***********************************************
Initialization             : Complete (0:00:40)
Health Check               : Complete (0:00:47)
etcd Install               : Complete (0:01:17)
NFS Install                : Complete (0:00:19)
Master Install             : Complete (0:03:12)
Master Additional Install  : Complete (0:01:39)
Node Install               : Complete (0:07:08)
Hosted Install             : Complete (0:04:39)
Web Console Install        : Complete (0:01:44)
```

```
107    Service Catalog Install    : Complete (0:04:33)
```

Alt + F2 / **firefox https://master.lab.example.com**


Login - OpenShift Co...
https://master.lab.example.com/login?then=%2Foauth%2Fauthorize%3
Search
RED HAT OPENSHIFT Container Platform
OPENSHIFT CONTAINER PLATFORM
Username
Password
Log In
Welcome to the OpenShift Container Platform.

Username: **developer**
Password: **redhat**



执行安装后任务

- Overview

安装完红帽 OpenShift 容器平台后，需要测试和验证所有 OpenShift 组件。仅仅从示例容器映像启动 pod 是不够的，因为这不使用 OpenShift 构建器、部署器、路由器或内部注册表。要验证 OpenShift 安装，请执行以下操作：

1. 检索所有 OpenShift 节点的状态。所有节点都应处于 **Ready** 就绪状态。
2. 检索 OpenShift 注册表和路由器 pods 的状态。所有的 pods 都应该处于 **Running** 运行状态。
3. 使用 OpenShift 集群从源代码构建应用程序。
   OpenShift 从构建结果生成一个容器映像，并从该映像启动一个 pod。
   这将测试集群是否可以从内部注册表拉入和推送到内部注册表。
   它还测试应用程序是否正确调度并部署到 OpenShift 节点。
4. 创建一个路由，以便可以从 OpenShift 集群内部网络之外的计算机访问应用程序。
   这将测试 OpenShift 路由器是否工作，并将外部请求路由到应用程序 pods。

- Configuring a Cluster Administrator

  [foundation]

  ```
  $ ssh student@master
  ```

  [student@master]

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

- lab2.1 Verifying the Installation

  [student@workstation]

  ```
  $ oc login
  Server [https://localhost:8443]: `https://master.lab.example.com`
  The server uses a certificate signed by an unknown authority.
  You can bypass the certificate check, but any data you send to the server could be intercepted by others.
  Use insecure connections? (y/n): `y`

  Authentication required for https://master.lab.example.com:443 (openshift)
  Username: `admin`
  Password: `redhat`
  Login successful.
  ...
  $ rm -rf ~/.kube
  $ oc login https://master.lab.example.com -u admin -p redhat --insecure-skip-tls-verify=true
  ```

- lab2.2 Verifying Node Status

  [student@workstation]

  ```
  $ oc get nodes
  ```

- lab2.3 Verifying Router and Registry Status

  [student@workstation]

  ```
  $ oc get pods
  ```

- lab2.4 Building an Application

[student@workstation]

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

-image2container
$ oc get image | grep php
-git/source
firefox http://services/php-helloworld
$ oc new-app php:5.6~http://services/php-helloworld \
  --name hello
$ oc get pods
NAME            READY     STATUS      RESTARTS   AGE
hello-1-719tm   1/1      `Running`    0          1h
hello-1-build   0/1       Completed   0          1h

$ oc get svc
NAME      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
`hello`   ClusterIP   172.30.153.125   <none>        8080/TCP,8443/TCP   1h

$ oc get route
No resources found.

$ oc expose svc hello
route "hello" exposed

$ oc get route
NAME      HOST/PORT                          PATH      SERVICES   PORT
TERMINATION   WILDCARD
hello     `hello-test.apps.lab.example.com`            hello      8080-tcp
    None

$ curl hello-test.apps.lab.example.com
Hello, World! php version is 5.6.25
```

- Failed Verification

[kiosk@foundation]

```
$ rht-vmctl reset -y master
$ rht-vmctl reset -y node1
$ rht-vmctl reset -y node2
```

[student@workstation]

```
$ lab install-prepare setup
$ cd ~/do280-ansible
$ ./install.sh
```

引导式练习：完成安装后任务

[student@workstation]

```
$ cd
$ lab install-post setup
Setting up workstation for lab exercise work:

 · Checking master VM connectivity............................  SUCCESS
 · Checking node1 VM connectivity.............................  SUCCESS
 · Checking node2 VM connectivity.............................  SUCCESS
 · Downloading classroom ansible artifacts....................  SUCCESS
 · Restarting docker..........................................  SUCCESS

$ oc help
$ oc login --help

登陆A：交互式
$ oc login
Server [https://localhost:8443]: `https://master.lab.example.com`
The server uses a certificate signed by an unknown authority.
You can bypass the certificate check, but any data you send to the server could be intercepted by others.
Use insecure connections? (y/n): `y`

Authentication required for https://master.lab.example.com:443 (openshift)
Username: `admin`
Password: `redhat`
Login successful.
...

登陆B：回显式
$ oc login -u admin -p redhat \
  https://master.lab.example.com \
  --insecure-skip-tls-verify=true

$ oc whoami
admin

$ oc get nodes
Error from server (Forbidden): nodes is forbidden: User "admin" cannot list nodes at the cluster scope: User "admin" cannot list all nodes in the cluster

$ ssh master
```

[student@master]

```
$ oc whoami
`system:admin`

$ cat ~/.kube/config

$ oc get clusterrole | grep admin
admin
`cluster-admin`
...

$ oc adm policy add-role-to-user cluster-admin admin
role "cluster-admin" added: "admin"

$ oc get rolebinding
cluster-admin  /cluster-admin  `admin`
...

$ exit
```

[student@workstation]

```
$ oc get nodes
NAME                     STATUS    ROLES     AGE       VERSION
master.lab.example.com  `Ready`    master    10h       v1.9.1+a0ce1bc657
node1.lab.example.com   `Ready`    compute   10h       v1.9.1+a0ce1bc657
node2.lab.example.com   `Ready`    compute   10h       v1.9.1+a0ce1bc657

$ oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-1-kpclw    1/1      `Running`   1          7h
docker-registry-1-qx4bg    1/1      `Running`   2          7h
registry-console-1-657ff   1/1      `Running`   1          7h
router-1-gq2c8             1/1      `Running`   1          7h
router-1-k2579             1/1      `Running`   1          7h

所有名字空间,-n指定的名字空间
$ oc get pods --all-namespaces
$ oc get pods -n default

$ oc login -u developer -p redhat
*$ oc new-project test
Now using project "test" on server "https://master.lab.example.com:443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git

to build a new example application in Ruby.

*$ oc project

$ oc new-app \
  php:5.6~http://services.lab.example.com/php-helloworld \
  --name hello
--> Found image 520f0e9 (22 months old) in image stream "openshift/php" under tag "5.6"
for "php:5.6"

```

```
    Apache 2.4 with PHP 5.6
    -----------------------
    PHP 5.6 available as container is a base platform for building and running various PHP 5.6 applications and frameworks. PHP is an HTML-embedded scripting language. PHP attempts to make it easy for developers to write dynamically generated web pages. PHP also offers built-in database integration for several commercial and non-commercial database management systems, so writing a database-enabled webpage with PHP is fairly simple. The most common use of PHP coding is probably as a replacement for CGI scripts.

    Tags: builder, php, php56, rh-php56

    * A source build using source code from http://services.lab.example.com/php-helloworld will be created
      * The resulting image will be pushed to image stream "hello:latest"
      * Use 'start-build' to trigger a new build
    * This image will be deployed in deployment config "hello"
    * Ports 8080/tcp, 8443/tcp will be load balanced by service "hello"
      * Other containers can access this service through the hostname "hello"

--> Creating resources ...
    imagestream "hello" created
    buildconfig "hello" created
    deploymentconfig "hello" created
    service "hello" created
--> Success
    Build scheduled, use 'oc logs -f bc/hello' to track its progress.
    Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
     'oc expose svc/hello'
    Run 'oc status' to view your app.

$ oc logs -f bc/hello
Cloning "http://services.lab.example.com/php-helloworld" ...
  Commit: 6d61e75647124d02aa761f994532ef29eae46f8e (Establish remote repository)
  Author: root <root@services.lab.example.com>
  Date: Thu Aug 9 11:33:29 2018 -0700
---> Installing application source...
=> sourcing 20-copy-config.sh ...
---> 10:38:09     Processing additional arbitrary httpd configuration provided by s2i ...
=> sourcing 00-documentroot.conf ...
=> sourcing 50-mpm-tuning.conf ...
=> sourcing 40-ssl-certs.sh ...

Pushing image docker-registry.default.svc:5000/test/hello:latest ...
Pushed 0/6 layers, 6% complete
Pushed 1/6 layers, 20% complete
Pushed 2/6 layers, 36% complete
Pushed 3/6 layers, 58% complete
Pushed 4/6 layers, 83% complete
Pushed 5/6 layers, 100% complete
Pushed 6/6 layers, 100% complete
Push successful
```

```

A. 看输出提示
$ oc expose svc/hell
B. 命令<Tab>
$ oc expose service hello
route "hello" exposed

$ oc get route
NAME      HOST/PORT                          PATH      SERVICES   PORT       TERMINATION
WILDCARD
hello     `hello-test.apps.lab.example.com`            hello      8080-tcp
  None

$ curl hello-test.apps.lab.example.com
Hello, World! php version is 5.6.25

$ oc logout
Logged "admin" out on "https://master.lab.example.com:443"

$ oc delete project test
```

总结

- 准备环境，使用 Ansible Playbook 安装 OpenShift 容器平台（OCP）
- 配置 OpenShift 高级安装主机 **清单文件**，使用适当的主机组，**组变量** 和 **主机变量**
- 使用 OpenShift 高级安装 Ansible Playbooks 来配置 **master** 和 **node** 服务器
- 通过从源代码 **创建应用程序**，并将其部署到 OpenShift，来验证正在运行的 OpenShift 集群

3. 描述和探索 OpenShift 网络概念

说明 OpenShift 的软件定义网络实施

- Software-Defined Networking (SDN)

  默认 Docker 网络使用仅限主机的虚拟网桥，主机内所有容器将附加到该网桥

  SDN实现控制平面与数据平面通信

  管理员可以为 Pod 配置三个 SDN 插件：

  - **ovs-subnet**

    默认插件，提供 flat Pod 网络

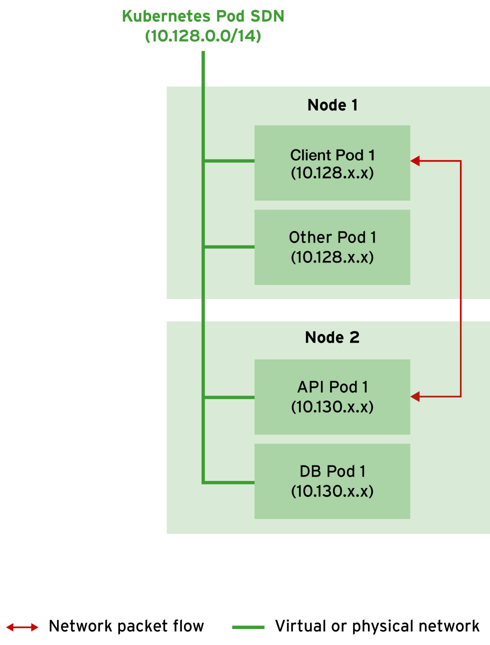
  - **ovs-multitenant**

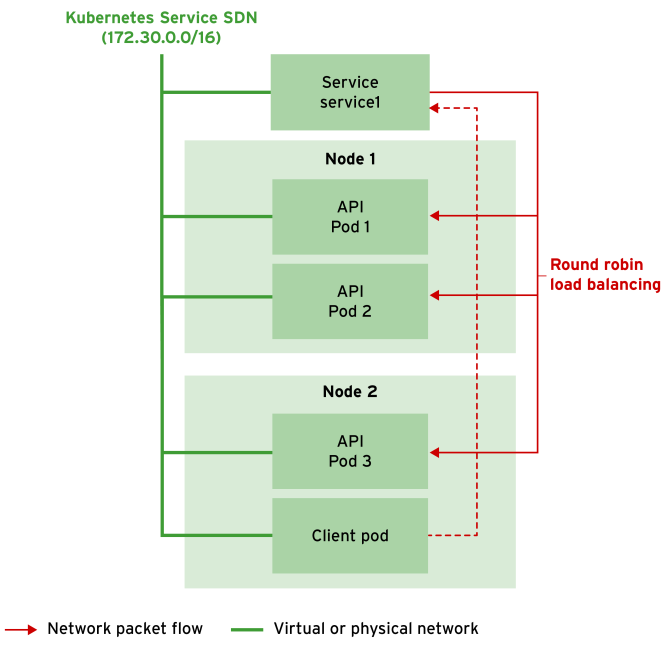
插件为 Pod 和服务提供额外的隔离层。每一个项目唯一的虚拟网络 ID （VNID）

- **ovs-networkpolicy**

  技术预览插件

master 主控节点不能通过集群网络访问容器





- OpenShift Network Topology

svc (service) 服务背后运行的 Pod 集合由 OpenShift 自动管理

与 **Selector** 匹配的各个 Pod 作为端点添加到服务资源中

- Getting Traffic into and out of the Cluster

如果应用需要从 OpehShift 集群外部访问服务，可以使用三种方法：

- **OpenShift routes**

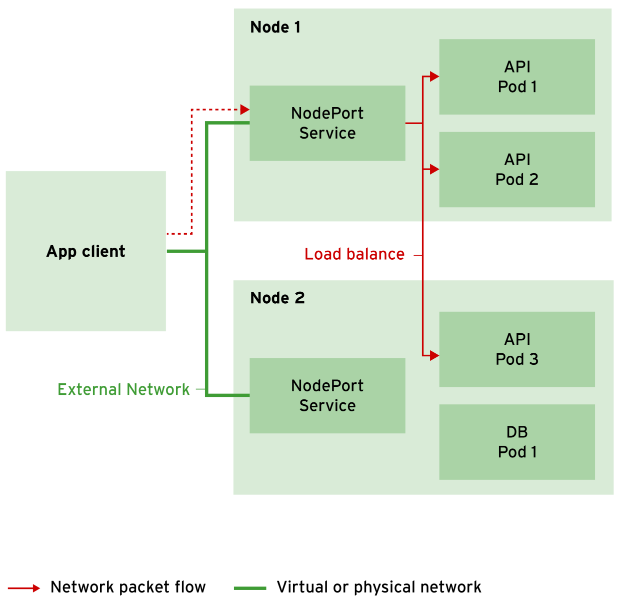
  首选方法，它利用唯一 URL 来公开服务

- **NodePode**

  Kubernetes 旧方法，服务将公开给外部客户端

- **NodePort/HostNetwork**

  这种方法需要升级特权才能运行



- Accessing External Networks

Pod 可以通过它所驻留的主机地址与外部网络通信

Pod 使用网络地址转换（**NAT**）与目标服务器通信

引导式练习: 探索软件定义型网络

[student@workstation]

```
$ lab openshift-network setup

Checking prerequisites for GE: Exploring Software-Defined Networking

 Checking all VMs are running:
 · master VM is up.............................................  SUCCESS
 · node1 VM is up..............................................  SUCCESS
 · node2 VM is up..............................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
 · Check router................................................  SUCCESS
 · Check registry..............................................  SUCCESS

Overall setup status...........................................  SUCCESS
```

```
$ oc login -u developer -p redhat

$ oc new-project test-network
$ oc project

$ oc new-app \
  --name nt \
  -i php:7.0 \
  http://registry.lab.example.com/scaling
--> Found image c101534 (2 years old) in image stream "openshift/php" under tag "7.0" for "php:7.0"

    Apache 2.4 with PHP 7.0
    -----------------------
    PHP 7.0 available as docker container is a base platform for building and running various PHP 7.0 applications and frameworks. PHP is an HTML-embedded scripting language. PHP attempts to make it easy for developers to write dynamically generated web pages. PHP also offers built-in database integration for several commercial and non-commercial database management systems, so writing a database-enabled webpage with PHP is fairly simple. The most common use of PHP coding is probably as a replacement for CGI scripts.

    Tags: builder, php, php70, rh-php70

    * The source repository appears to match: php
    * A source build using source code from http://registry.lab.example.com/scaling will be created
      * The resulting image will be pushed to image stream "nt:latest"
      * Use 'start-build' to trigger a new build
    * This image will be deployed in deployment config "nt"
    * Port 8080/tcp will be load balanced by service "nt"
      * Other containers can access this service through the hostname "nt"

--> Creating resources ...
    imagestream "nt" created
    buildconfig "nt" created
    deploymentconfig "nt" created
    service "nt" created
```

```
--> Success
    Build scheduled, use 'oc logs -f bc/nt' to track its progress.
    Application is not exposed. You can expose services to the outside world by executing
one or more of the commands below:
     'oc expose svc/nt'
    Run 'oc status' to view your app.

$ oc get pods
NAME          READY     STATUS      RESTARTS   AGE
nt-1-build    0/1       Completed   0          34m
nt-1-w7x5p    1/1       Running     0          22m

$ oc scale --replicas=2 dc nt
eploymentconfig "nt" scaled

$ oc get pods -o wide
NAME          READY     STATUS      RESTARTS   AGE       IP              NODE
nt-1-build    0/1       Completed   0          37m       10.129.0.12    node1...
nt-1-kfj5l    1/1       Running     0          1m       `10.128.0.17`   node2...
nt-1-w7x5p    1/1       Running     0          25m      `10.129.0.14`   node1...

$ curl http://10.128.0.17:8080
curl: (7) Failed connect to 10.128.0.17:8080; Network is unreachable
$ curl http://10.129.0.14:8080
curl: (7) Failed connect to 10.129.0.14:8080; Network is unreachable

$ ssh root@node1 \
    curl -s http://10.128.0.17:8080
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <br/> Server IP: 10.128.0.17
 </body>
</html>
$ ssh root@node2 \
    curl -s http://10.129.0.14:8080
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <br/> Server IP: 10.129.0.14
 </body>
</html>

$ oc get svc nt
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
nt        ClusterIP  `172.30.135.23`  <none>        8080/TCP   44m

$ curl http://172.30.135.23:8080
```

```
curl: (7) Failed connect to 172.30.135.23:8080; Network is unreachable

$ ssh node1 \
    curl -s http://172.30.135.23:8080
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <br/> Server IP: `10.129.0.14`
 </body>
</html>
$ ssh node1 \
    curl -s http://172.30.135.23:8080
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <br/> Server IP: `10.128.0.17`
 </body>
</html>

$ oc describe svc nt
Name:                   nt
Namespace:              test-network
Labels:                 app=nt
Annotations:            openshift.io/generated-by=OpenShiftNewApp
Selector:               app=nt,deploymentconfig=nt
Type:                   ClusterIP
IP:                     172.30.135.23
Port:                   8080-tcp  8080/TCP
TargetPort:             8080/TCP
Endpoints:              10.128.0.17:8080,10.129.0.14:8080
Session Affinity:       None
Events:                 <none>

$ oc describe pod nt-1-kfj5l
Labels:         app=nt
                deployment=nt-1
                deploymentconfig=nt
...输出被忽略...

$ oc edit svc nt
...
spec:
...
    targetPort: 8080
    nodePort: 30800
  selector:
    app: nt
    deploymentconfig: nt
    sessionAffinity: None
    type: NodePort
  status:
    loadBalancer: {}
service "nt" edited

$ oc describe svc nt | egrep 'Type|NodePort'
Type:                     NodePort
NodePort:                 8080-tcp  30800/TCP

$ curl http://node1.lab.example.com:30800
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <br/> Server IP: 10.129.0.14
 </body>
</html>
$ curl http://node2.lab.example.com:30800
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <br/> Server IP: 10.128.0.17
 </body>
</html>

$ oc rsh nt-1-kfj5l
```

[nt-1-kfj51]

```
sh-4.2$ curl http://services.lab.example.com
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-US" lang="en-US">
<!-- git web interface version 1.8.3.1, (C) 2005-2006, Kay Sievers <kay.sievers@vrfy.org>,
Christian Gierke -->
<!-- git core binaries version 1.8.3.1 -->
...输出被忽略...

sh-4.2$ exit
exit
```

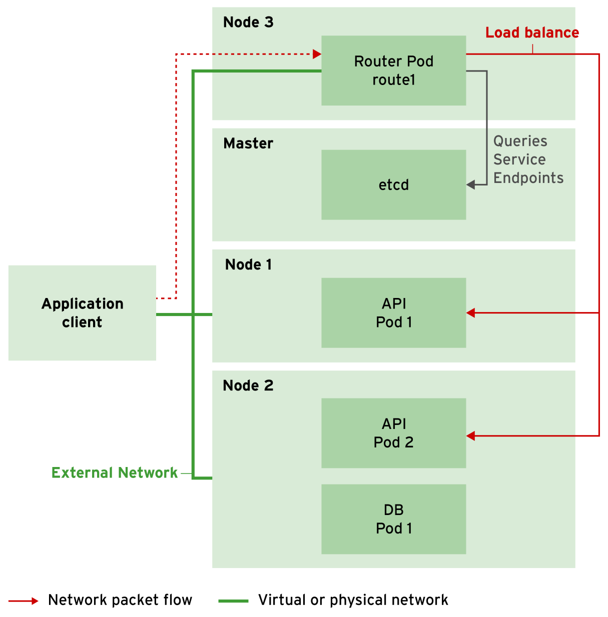
[student@workstation]

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

创建路由

- Describing the OpenShift Router

实现从外部 OpenShift 实例到 Pod 的网络访问



- Creating Routes

```
$ oc expose svc ...
```

- Finding the Default Routing Subdomain

[root@master]

```
# grep subdomain /etc/origin/master/master-config.yaml
```

- Routing Options and Types

受保护的路由指定路由的 TLS 终止。下方列出了可用的终止类型：

- **Edge Termination** 边缘终止：

  TLS 终止在流量路由到 pods 之前发生在路由器上。TLS 证书由路由器提供，因此它们必须配置到路由内。

- **Pass-through Termination** 传递终止：

  加密的流量直接发送到目的地 pod，无需路由器提供 TLS 终止。不需要密钥或证书。目的地 Pod 负责在端点为流量提供证书。

- **Re-encryption Termination** 再加密终止：

  再加密终止是边缘终止的一种变体，即路由器通过证书终止 TLS，然后再加密它与端点的连接，这可能有不同的证书。

  - Creating Secure Routes

```
private key: 新生儿
$ openssl genrsa \
  -out hello.apps.lab.example.com.key \
  2048

request: 准生证
$ openssl req \
  -new-key hello.apps.lab.example.com.key \
  -out hello.apps.lab.example.com.csr \
  -subj "/C=US/ST=NC/L=Raleigh/O=RedHat/OU=RHT/CN=hello.apps.lab.example.com"

public key: 证书
$ openssl x509 \
  -req \
  -days 366 \
  -in hello.apps.lab.example.com.csr \
  -signkey hello.apps.lab.example.com.key \
  -out hello.apps.lab.example.com.crt
```

  - Wildcard Routes for Subdomains

```
$ oc login -u admin

$ oc scale dc/router --replicas=0

$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc scale dc/router --replicas=1

$ oc expose svc test --wildcard-policy=Subdomain --hostname='www.lab.example.com'
```

- Monitoring Routes

  [**root@master**]

```
$ oc project default

$ oc get pods
```

引导式练习: 创建路由

[**student@workstation**]

```
$ lab secure-route setup

Checking prerequisites for GE: Create a Route

 Checking all VMs are running:
 · master VM is up.............................................  SUCCESS
 · node1 VM is up..............................................  SUCCESS
 · node2 VM is up..............................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
  · Check router........................................  SUCCESS
  · Check registry......................................  SUCCESS

Downloading files for GE: Create a Route

  · Downloading starter project.........................  SUCCESS
  · Downloading solution project........................  SUCCESS

Download successful.

Overall setup status....................................  SUCCESS
```

```
$ oc login -u developer -p redhat
Login successful.

You don\'t have any projects. You can try to create a new project, by running

    oc new-project <projectname>

$ oc new-project secure-route
Now using project "secure-route" on server "https://master.lab.example.com:443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git

to build a new example application in Ruby.

$ docker-registry-cli \
    registry.lab.example.com \
    search hello \
    ssl
available options:-

-----------
1) Name: openshift/hello-openshift
Tags: latest

1 images found !

$ oc new-app \
  --docker-image=registry.lab.example.com/openshift/hello-openshift \
  --name hello
--> Found Docker image 7af3297 (22 months old) from registry.lab.example.com for
"registry.lab.example.com/openshift/hello-openshift"

    * An image stream will be created as "hello:latest" that will track this image
    * This image will be deployed in deployment config "hello"
    * Ports 8080/tcp, 8888/tcp will be load balanced by service "hello"
      * Other containers can access this service through the hostname "hello"

```

```
--> Creating resources ...
    imagestream "hello" created
    deploymentconfig "hello" created
    service "hello" created
--> Success
    Application is not exposed. You can expose services to the outside world by executing
one or more of the commands below:
     'oc expose svc/hello'
    Run 'oc status' to view your app.

$ oc get pods -o wide
NAME            READY     STATUS    RESTARTS   AGE       IP            NODE
hello-1-xckfp   1/1       Running   0          54s       10.129.0.16   node1...

$ cat ~student/DO280/labs/secure-route/create-cert.sh
...
echo "Generating a private key..."
openssl genrsa -out hello.apps.lab.example.com.key 2048
...输出被忽略...
echo "Generating a CSR..."
openssl req -new -key hello.apps.lab.example.com.key -out hello.apps.lab.example.com.csr
-subj "/C=US/ST=NC/L=Raleigh/O=RedHat/OU=RHT/CN=hello.apps.lab.example.com"
...输出被忽略...
echo "Generating a certificate..."
openssl x509 -req -days 366 -in hello.apps.lab.example.com.csr -signkey
hello.apps.lab.example.com.key -out hello.apps.lab.example.com.crt
...输出被忽略...

$ cd ~student/DO280/labs/secure-route
$ ./create-cert.sh
Generating a private key...
Generating RSA private key, 2048 bit long modulus
........................................................+++
.........+++
e is 65537 (0x10001)

Generating a CSR...

Generating a certificate...
Signature ok
subject=/C=US/ST=NC/L=Raleigh/O=RedHat/OU=RHT/CN=hello.apps.lab.example.com
Getting Private key

DONE.

$ ls
commands.txt    hello.apps.lab.example.com.crt  hello.apps.lab.example.com.key
create-cert.sh  hello.apps.lab.example.com.csr

$ cat commands.txt
# Login as developer
oc login -u developer -p redhat https://master.lab.example.com
```

```

# Create new application
oc new-app
  --docker-image=registry.lab.example.com/openshift/hello-openshift
  --name=hello

# Create a secure edge route
oc create route edge
  --service=hello
  --hostname=hello.apps.lab.example.com
  --key=hello.apps.lab.example.com.key
  --cert=hello.apps.lab.example.com.crt

# plain http
curl http://hello.apps.lab.example.com

# secure https
curl -k -vvv https://hello.apps.lab.example.com

# Pod IP
curl -vvv http://<pod ip>:8080

$ oc create route edge --service=hello \
  --hostname=hello.apps.lab.example.com \
  --key=hello.apps.lab.example.com.key \
  --cert=hello.apps.lab.example.com.crt
route "hello" created

$ oc get routes
NAME     HOST/PORT                    PATH     SERVICES   PORT       TERMINATION
WILDCARD
hello    hello.apps.lab.example.com            hello      8080-tcp   edge          None

$ oc get route hello -o yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  creationTimestamp: 2020-02-17T19:52:46Z
  labels:
    app: hello
  name: hello
  namespace: secure-route
  resourceVersion: "128445"
  selfLink: /apis/route.openshift.io/v1/namespaces/secure-route/routes/hello
  uid: 11ee8b89-51bf-11ea-809e-52540000fa0a
spec:
  host: hello.apps.lab.example.com
  port:
    targetPort: 8080-tcp
  tls:
    certificate: |
      -----BEGIN CERTIFICATE-----
      MIIDXDCCAkQCCQCnTeIQvS+75TANBgkqhkiG9w0BAQsFADBwMQswCQYDVQQGEwJV
      UzELMAkGA1UECAwCTkMxEDAOBgNVBAcMB1JhbGVpZ2gxDzANBgNVBAoMBlJlZEhh
      dDEMMAoGA1UECwwDUkhUMSMwIQYDVQQDDBpoZWxsby5hcHBzLmxhYi5leGFtcGxl
      LmNvbTAeFw0yMDAyMTcxOTQ4NTlaFw0yMTAyMTcxOTQ4NTlaMHAxCzAJBgNVBAYT
      AlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEPMA0GA1UECgwGUmVk
      SGF0MQwwCgYDVQQLDANSSFQxIzAhBgNVBAMMGmhlbGxvLmFwcHMubGFiLmV4YW1w
      bGUuY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqv9b36mgWni6
      TWZARVXWNZHWGRERt7wAu/BWmfeKJ3ZBbY6ahGrARVPxmvZd7VA5s8z0LxaNxg41
      YzAr0T5pbT0HbCMnK1JsNVhcRI1KRCnznuuo/FZosAoczS6sVoHgGx3BzKheyY+q
      TEfh3HECmw8f1jbNQiQkL83eOui1o57uQ/dHkYJfPVX8wI6u4SiV981LhljEO12/
      cCUhE+V+NcDR/PR5yEaJy1KVXcFTgA/AGVZbL5ok5m8kNNKshGoF8Wu6j/e0qovg
      lxzFV9x+VPcWQ2iSFurYS49WJ8S3FydTM5OcavpZZj6QHfPjeOcauFCwExk2UHLP
      1p89eUe9UwIDAQABMA0GCSqGSIb3DQEBCwUAA4IBAQCkd9hqZxK02RxMQPos0Dbx
      Yw3Fv4ukPYR1oDxxXObFatzaZHukYbLUifcJYZ8qm9ht9OLvoTnZ5QZnC/ntibv6
      kLzAE3hUvWbnKgCCx/R8nDfhW6WzXxrIffTgVkE9Zr+VUmTUzqCy4BXeoA5A5eOh
      PGfzdNDd+2JjEY5+gFTJkqcyvrhWj3uXFQR3YGTV6/d/5Svt9amk/vEXd72iYApI
      sHzcKDMsS5z9MTft/J8oGE3IwdWQqMECcDHvST0XAG2eMnohJFFzfSFTBbhDiHW6
      GN2x+QVcQGnwIHa7xj9IgVu85v/7THbo2wP4lHIUQ91ZOn0nlP+NbzEwSNF/qvrf
      -----END CERTIFICATE-----
    key: |
      -----BEGIN RSA PRIVATE KEY-----
      MIIEogIBAAKCAQEAqv9b36mgWni6TWZARVXWNZHWGRERt7wAu/BWmfeKJ3ZBbY6a
      hGrARVPxmvZd7VA5s8z0LxaNxg41YzAr0T5pbT0HbCMnK1JsNVhcRI1KRCnznuuo
      /FZosAoczS6sVoHgGx3BzKheyY+qTEfh3HECmw8f1jbNQiQkL83eOui1o57uQ/dH
      kYJfPVX8wI6u4SiV981LhljEO12/cCUhE+V+NcDR/PR5yEaJy1KVXcFTgA/AGVZb
      L5ok5m8kNNKshGoF8Wu6j/e0qovglxzFV9x+VPcWQ2iSFurYS49WJ8S3FydTM5Oc
      avpZZj6QHfPjeOcauFCwExk2UHLP1p89eUe9UwIDAQABAoIBAG/bocb62Hm2VfDB
      vbNdhkX+w3YcU2HEqxpGCvCnHInZ8szvJycOCf6P/hFnrmPKQiTbIrUW5OE1dDkR
      TuiPEjoyXQOhL0NIpJ500c7KOlXCt6oy8JU5FTxrMRILwRLJ3McAPUFatr7VqwpB
      T397sb+rMiFYMgddSwq2efRBPGjuQSKpgzucT6zjl96u392yB1AQDztvTymDkDHe
      fAcow5NL8LPPu6TjiJtKJhK90lt+NMf9ucKZbgBwoEOAkMrtjUDVJC52RwOkLoNN
      pOAvrAqurjyk1wmmN7Saw8bhxI37dp4GQTcTvKpGxLSTAfv59Hq8v/t+2uwOYVlS
      daFq3SECgYEA3mFD0qS4EZljfxTlYrDMk90xt41hWdbxaXIfSDN8h2GTXWQt7JjT
      Fh2ODSDqqNStFDzfHF+bhXMYnlcDH153v6Asp26HS8IcCJMJjC6NPg4nJIdoGXbv
      cVLedskOKxfy0BjAZz/j7DSQsiDrTX2AV/Doa3dSdyDEWD9C9ll/5fECgYEAxNly
      itC9RBjP27NZGTCRrFSFqsxZ06Opd4ZsbK7RmkcWkSvibFzhXth6BNNdxOHOMNZR
      dbYsUwqPXElsIz+Y9A8a0n+fkCiRjrO9qflftPe3fGNu/GACRfBPar6fW3gDMNPw
      dEWpHx4uEToa0z4k5kff6y3nTOf1Cm4IlTsXQ4MCgYAc6PgwQknLv+03cDgCBIoU
      DwWPn0mwrEjmNHfsowTldMH7ujJeN9/5WA5HlqfrGvsFToSS47sMNlJVA2rcgSOA
      PgqQGcZtCucqFjN/je2+y4g7L39REC1Axk01lB3LbGmctBsPUTcIVi0Zez4b7Nzq
      kd8lWXXXFuNvtYm3DRubgQKBgAYPtAk2OD26jdvz/9BYwIOP7rW9qR5tMbCugPQv
      xeB8Q+OgeE5h5can38n6QC7pzRGP5946B89eyd9Lm3rSYIFTXb4Rk/Y6aZD9U9/C
      AAJwhkPcQ/SdeDRzG97rk7ibT23XeNX7tyNwKHb7VQwgI767g9eYCEFD+zWhAb6m
      nSbFAoGARktNht1ZyALw0DCKJtl7xi0969Fq34//GXduznFhumhNBY4K6ddAKAeH
      zmbLDB4aMKfxpiUlHdDRr9+FVvCiOcIxVWZUYV2EndURAf3PdKgt7NPMrfhgVEMA
      FOJ+Qgn8S9zexe+AX4P7Uu9gEMq1CLL5XunuAnP8FvOzCHU568Y=
      -----END RSA PRIVATE KEY-----
    termination: edge
  to:
    kind: Service
    name: hello
    weight: 100
  wildcardPolicy: None
status:
  ingress:
  - conditions:
    - lastTransitionTime: 2020-02-17T19:52:47Z
      status: "True"
      type: Admitted
    host: hello.apps.lab.example.com
    routerName: router
    wildcardPolicy: None

$ curl http://hello.apps.lab.example.com
...输出被忽略...
      <h1>Application is not available</h1>
      <p>The application is currently not serving requests at this endpoint. It may not have been started or is still starting.</p>
...输出被忽略...

$ curl -k -vvv https://hello.apps.lab.example.com
* About to connect() to hello.apps.lab.example.com port 443 (#0)
*   Trying 172.25.250.11...
* Connected to hello.apps.lab.example.com (172.25.250.11) port 443 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
* skipping SSL peer certificate verification
* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*   subject: CN=hello.apps.lab.example.com,OU=RHT,O=RedHat,L=Raleigh,ST=NC,C=US
*   start date: Feb 17 19:48:59 2020 GMT
*   expire date: Feb 17 19:48:59 2021 GMT
*   common name: hello.apps.lab.example.com
*   issuer: CN=hello.apps.lab.example.com,OU=RHT,O=RedHat,L=Raleigh,ST=NC,C=US
> GET / HTTP/1.1
> User-Agent: curl/7.29.0
> Host: hello.apps.lab.example.com
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 17 Feb 2020 19:57:39 GMT
< Content-Length: 17
< Content-Type: text/plain; charset=utf-8
< Set-Cookie: 0dca6369ebce37a9206a19316b32350e=586d2fc9e3f702fd01e8b07dd7f8607a; path=/; HttpOnly; Secure
< Cache-control: private
<
Hello OpenShift!
* Connection #0 to host hello.apps.lab.example.com left intact

$ ssh node1 curl -vvv http://10.129.0.16:8080
* About to connect() to 10.129.0.16 port 8080 (#0)
*   Trying 10.129.0.16...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected
to 10.129.0.16 (10.129.0.16) port 8080 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.29.0
> Host: 10.129.0.16:8080
> Accept: */*
>
Hello OpenShift!
...输出被忽略...
```

```
$ oc delete project secure-route
project "secure-route" deleted
```

实验: 探索 OpenShift 网络概念

[student@workstation]

```
$ lab network-review setup
Checking prerequisites for Lab: Exploring OpenShift Networking

 Checking all VMs are running:
 · master VM is up..............................................  SUCCESS
 · node1 VM is up...............................................  SUCCESS
 · node2 VM is up...............................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
 · Check router.................................................  SUCCESS
 · Check registry...............................................  SUCCESS
 Setting up for the lab:
 . Logging in as the developer user.............................  SUCCESS
 . Creating the network-review project..........................  SUCCESS
 . Creating resources for the network-review project...........  SUCCESS

 Back to OpenShift as system:admin..............................  SUCCESS

Overall setup status............................................  SUCCESS
```

```
$ oc login -u developer -p redhat
Login successful.

You have one project on this server: "network-review"

Using project "network-review".

$ oc get pods -o wide
NAME                       READY     STATUS    RESTARTS   AGE       IP              NODE
hello-openshift-1-m4c47    1/1       Running   0          2m        10.129.0.17
node1.lab.example.com

$ oc get svc
NAME              TYPE         CLUSTER-IP    EXTERNAL-IP   PORT(S)             AGE
```

```
hello-openshift   ClusterIP   172.30.13.4   <none>        8080/TCP,8888/TCP   2m

$ oc get routes
NAME              HOST/PORT                   PATH      SERVICES          PORT
TERMINATION   WILDCARD
hello-openshift   hello.apps.lab.example.com            hello-opensift   8080-tcp
        None

$ curl http://hello.apps.lab.example.com
...输出被忽略...
<h1>Application is not available</h1>
      <p>The application is currently not serving requests at this endpoint. It may not
have been started or is still starting.</p>
...输出被忽略...

$ ssh master curl -s http://10.129.0.17:8080
Hello OpenShift!
$ ssh master curl http://172.30.13.4:8080
...输出被忽略...
curl: (7) Failed connect to 172.30.13.4:8080; Connection refused

$ oc describe svc hello-openshift
Name:              hello-openshift
Namespace:         network-review
Labels:            app=hello-openshift
Annotations:       openshift.io/generated-by=OpenShiftNewApp
Selector:          app=hello_openshift,deploymentconfig=hello-openshift
Type:              ClusterIP
IP:                172.30.13.4
Port:              8080-tcp  8080/TCP
TargetPort:        8080/TCP
Endpoints:         <none>
Port:              8888-tcp  8888/TCP
TargetPort:        8888/TCP
Endpoints:         <none>
Session Affinity:  None
Events:            <none>

$ oc describe pod hello-openshift-1-m4c47
Name:         hello-openshift-1-m4c47
Namespace:    network-review
Node:         node1.lab.example.com/172.25.250.11
Start Time:   Tue, 18 Feb 2020 04:09:12 +0800
Labels:       app=hello-openshift
...输出被忽略...

$ oc edit svc hello-openshift
...输出被忽略...
  selector:
    app: hello-openshift
...输出被忽略...
service "hello-openshift" edited
```

```

$ ssh master curl -s http://172.30.13.4:8080
Hello OpenShift!

$ curl http://hello.apps.lab.example.com
...输出被忽略...
<h1>Application is not available</h1>
      <p>The application is currently not serving requests at this endpoint. It may not have been started or is still starting.</p>
...输出被忽略...

$ oc describe route hello-openshift
Name:     hello-openshift
Namespace:    network-review
Created:    19 minutes ago
Labels:     app=hello-openshift
Annotations:    <none>
Requested Host:   hello.apps.lab.example.com
        exposed on router router 19 minutes ago
Path:     <none>
TLS Termination:  <none>
Insecure Policy:  <none>
Endpoint Port:    8080-tcp

Service:  hello-opensift
Weight:   100 (100%)
Endpoints:  <error: endpoints "hello-opensift" not found>

$ oc edit route hello-openshift
...输出被忽略...
    kind: Service
    name: hello-openshift
...输出被忽略...
route "hello-openshift" edited

$ curl http://hello.apps.lab.example.com
Hello OpenShift!
```

```
$ lab network-review grade
Grading the student's work for Lab: Exploring OpenShift Networking

 · Check if the hello-openshift pod is in Running state........  PASS
 . Checking if service configuration was fixed correctly.......  PASS
 . Checking if route configuration was fixed correctly.........  PASS
 . Checking if route can be invoked successfully...............  PASS

Overall exercise grade.........................................  PASS

$ oc delete project network-review
project "network-review" deleted
```

总结

- OpenShift 软件定义的网络（SDN）实施基于 **Open vSwitch**（OVS），以及它如何提供统一集群网络来实现 OpenShift 集群内不同 pod 之间的通信。
  - OpenShift 服务：
    - 具有唯一的 IP 地址，代客户端连接以访问集群中的 pods。
    - 也来自 OpenShift SDN 的 IP 地址，它有别于 pod 的内部网络，但仅在集群内部可见。
    - 确保与 **选择器** 匹配的各个 pod 作为端点添加到服务资源中。随着 pod 的创建和终止，服务背后的端点会自动更新。
  - 如果应用需要从 OpenShift 集群外部访问服务，可以通过两种方式来实现这个目标：
    - **NodePort**：服务将公开给外部客户端，方法是先绑定至节点主机上的可用端口，再将连接代理到服务 IP 地址。用于节点端口的端口号限制为 **30000-32767** 范围。
    - **OpenShift 路由**：此方法使用唯一的 URL 公开服务。使用 **oc expose** 命令公开服务的外部访问，或者从 OpenShift Web 控制台公开服务。
  - 借助网络地址转换（NAT），Pods 可以使用主机地址与 OpenShift 集群外的服务器通信。NAT 通过主机 IP 地址传输网络流量。
  - OpenShift 路由由一个共享路由器服务来实施，该服务作为 OpenShift 实例内的 pod 运行，可以像任何其他常规的 Pod 一样进行缩放和复制。此路由器服务基于开源软件 **HAProxy**。
  - 可以像创建任何其他 OpenShift 资源一样创建路由资源，即为 **oc create** 提供 JSON 或 YAML 资源定义文件，或者使用 **oc expose** 命令。
  - 如果从模板或通过 **oc expose** 命令创建的路由，但不使用显式 **--hostname** 选项，则会生成格式以下形式的 DNS 名称： **<route name>-<project name>.<default domain>**。
  - 路由支持下列协议：
    - HTTP 超文本传输协议
    - HTTPS（使用 SNI）
    - WebSocket
    - TLS（使用 SNI）
  - 你可以创建不同类型的路由：
    - **Edge Termination** 边缘终止：TLS 终止在流量路由到 pods 之前发生在路由器上。TLS 证书由路由提供，因此它们必须配置到路由内。
    - **Pass-through Termination** 传递终止：加密的流量直接发送到目的地 pod，无需路由器提供 TLS 终止。不需要密钥或证书。目的地 Pod 负责在端点为流量提供证书。
    - **Re-encryption Termination** 再加密终止：再加密终止是边缘终止的一种变体，即路由器通过证书终止 TLS，然后再加密它与端点的连接，这可能有不同的证书。

- 利用通配符策略，用户可以定义覆盖一个域内所有主机的路由。通过 `wildcardPolicy` 字段，路由可以指定通配符策略作为其配置的一部分。OpenShift 路由器支持通配符路由，通过将 `ROUTER_ALLOW_WILDCARD_ROUTES` 环境变量设置为 `true` 来实现。

4. 执行命令

使用 CLI 配置资源

- Accessing Resources from the Managed OpenShift Instance
- Installing the oc Command-line Tool

  ```
  $ yum provides oc

  $ sudo yum install -y atomic-openshift-clients
  ```

  Useful Commands to Manage OpenShift Resources

  - `oc get all`
  - `oc describe RESOURCE RESOURCE_NAME`
  - `oc export`
  - `oc create`
  - `oc delete RESOURCE_TYPE name`
  - `oc exec`
  - `oc rsh POD`

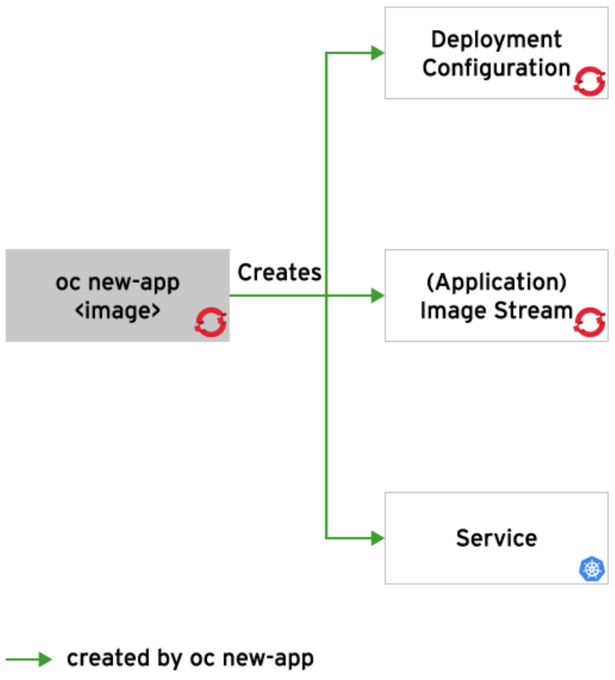
    ```
    $ oc new-project test

    $ oc new-app php:7.0~http://registry/php-helloworld

    $ oc get all

    $ oc export svc/php-helloworld > file.yaml

    $ vim file.yaml

    $ oc create -f file.yaml

    $ oc get svc hello

    $ oc delete svc hello

    $ oc exec php-helloworld-1-d44sj -- date
    $ oc exec -i -t php-helloworld-1-d44sj -- bash
    ```

```

$ oc rsh php-helloworld-1-d44sj
```

- OpenShift Resource Types

```
$ oc types
```

- Creating Applications Using **oc new-app**



引导式练习: 使用 oc 管理 OpenShift 实例

[student@workstation]

```
$ lab manage-oc setup

Checking prerequisites for GE: Managing an OpenShift Instance Using oc

 Checking all VMs are running:
 · master VM is up.............................................  SUCCESS
 · node1 VM is up..............................................  SUCCESS
 · node2 VM is up..............................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
 · Check router................................................  SUCCESS
 · Check registry..............................................  SUCCESS

Overall setup status...........................................  SUCCESS
```

```
$ oc login -u admin -p redhat

$ oc project default
Already on project "default" on server "https://master.lab.example.com:443".
```

```

$ oc get nodes
NAME                    STATUS    ROLES     AGE       VERSION
master.lab.example.com  Ready     master    3d        v1.9.1+a0ce1bc657
node1.lab.example.com   Ready     compute   3d        v1.9.1+a0ce1bc657
node2.lab.example.com   Ready     compute   3d        v1.9.1+a0ce1bc657

$ oc describe node master.lab.example.com
Name:               master.lab.example.com
Roles:              master
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=master.lab.example.com
                    node-role.kubernetes.io/master=true
                    openshift-infra=apiserver
Annotations:        volumes.kubernetes.io/controller-managed-attach-detach=true
Taints:             <none>
...输出被忽略...
System Info:
...输出被忽略...
 Kernel Version:             3.10.0-862.el7.x86_64
 OS Image:                   Red Hat Enterprise Linux Server 7.5 (Maipo)
 Operating System:           linux
 Architecture:               amd64
 Container Runtime Version:  docker://1.13.1
 Kubelet Version:            v1.9.1+a0ce1bc657
 Kube-Proxy Version:         v1.9.1+a0ce1bc657
ExternalID:                  master.lab.example.com
...输出被忽略...
Events:
...输出被忽略...
  Normal   Starting  28m   kubelet, master.lab.example.com  Starting kubelet.
  ...输出被忽略...
  Normal   NodeReady  28m   kubelet, master.lab.example.com  Node master.lab.example.com status is now: NodeReady

$ oc describe node node1.lab.example.com
Name:               node1.lab.example.com
Roles:              compute
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=node1.lab.example.com
                    node-role.kubernetes.io/compute=true
                    region=infra
Annotations:        volumes.kubernetes.io/controller-managed-attach-detach=true
Taints:             <none>
CreationTimestamp:  Mon, 17 Feb 2020 10:41:37 +0800
...输出被忽略...
  Normal   NodeReady                33m   kubelet, node1.lab.example.com  Node node1.lab.example.com status is now: NodeReady

$ oc exec docker-registry-1-qx4bg -- hostname
```

```
docker-registry-1-qx4bg

$ oc exec docker-registry-1-qx4bg -- cat /etc/hostname
docker-registry-1-qx4bg

$ oc exec -it docker-registry-1-qx4bg -- bash
bash-4.2$ hostname
bash-4.2$ exit

$ oc rsh docker-registry-1-qx4bg
sh-4.2$ hostname
docker-registry-1-qx4bg
sh-4.2$ exit

$ oc status -v
In project default on server https://master.lab.example.com:443

https://docker-registry-default.apps.lab.example.com (passthrough) (svc/docker-registry)
  dc/docker-registry deploys registry.lab.example.com/openshift3/ose-docker-registry:v3.9.14
    deployment #1 deployed 3 days ago - 2 pods

svc/kubernetes - 172.30.0.1 ports 443, 53->8053, 53->8053

https://registry-console-default.apps.lab.example.com (passthrough) (svc/registry-console)
  dc/registry-console deploys registry.lab.example.com/openshift3/registry-console:v3.9
    deployment #1 deployed 3 days ago - 1 pod

svc/router - 172.30.197.168 ports 80, 443, 1936
  dc/router deploys registry.lab.example.com/openshift3/ose-haproxy-router:v3.9.14
    deployment #1 deployed 3 days ago - 2 pods

View details with 'oc describe <resource>/<name>' or list everything with 'oc get all'.

$ oc get events

$ oc get all
NAME                                 REVISION   DESIRED   CURRENT   TRIGGERED BY
deploymentconfigs/docker-registry    1          2         2         config
deploymentconfigs/registry-console   1          1         1         config
deploymentconfigs/router             1          2         2         config

NAME                             DOCKER REPO                                          TAGS      UPDATED
imagestreams/registry-console   docker-registry.default.svc:5000/default/registry-console   v3.9      3 days ago
...输出被忽略...
NAME                          READY     STATUS    RESTARTS   AGE
po/docker-registry-1-kpclw    1/1       Running   2          3d
po/docker-registry-1-qx4bg    1/1       Running   3          3d
po/registry-console-1-657ff   1/1       Running   2          3d
```

```
po/router-1-gq2c8             1/1       Running   2          3d
po/router-1-k2579             1/1       Running   2          3d

NAME                    DESIRED   CURRENT   READY     AGE
rc/docker-registry-1    2         2         2         3d
rc/registry-console-1   1         1         1         3d
rc/router-1             2         2         2         3d
...输出被忽略...

$ oc export pod docker-registry-1-qx4bg
apiVersion: v1
kind: Pod
metadata:
  annotations:
    openshift.io/deployment-config.latest-version: "1"
    openshift.io/deployment-config.name: docker-registry
    openshift.io/deployment.name: docker-registry-1
    openshift.io/scc: restricted
  creationTimestamp: null
  generateName: docker-registry-1-
  labels:
    deployment: docker-registry-1
    deploymentconfig: docker-registry
    docker-registry: default
  ownerReferences:
  - apiVersion: v1
    blockOwnerDeletion: true
    controller: true
    kind: ReplicationController
    name: docker-registry-1
...输出被忽略...

$ oc export svc,dc docker-registry --as-template=docker-registry
apiVersion: v1
kind: Template
metadata:
  creationTimestamp: null
  name: docker-registry
objects:
- apiVersion: v1
  kind: Service
  metadata:
    creationTimestamp: null
    labels:
      docker-registry: default
    name: docker-registry
  spec:
    ports:
    - name: 5000-tcp
      port: 5000
      protocol: TCP
      targetPort: 5000
    selector:
      docker-registry: default
    sessionAffinity: ClientIP
    sessionAffinityConfig:
      clientIP:
        timeoutSeconds: 10800
    type: ClusterIP
  status:
    loadBalancer: {}
...输出被忽略...

$ oc export svc,dc docker-registry > docker-registry.yml
```

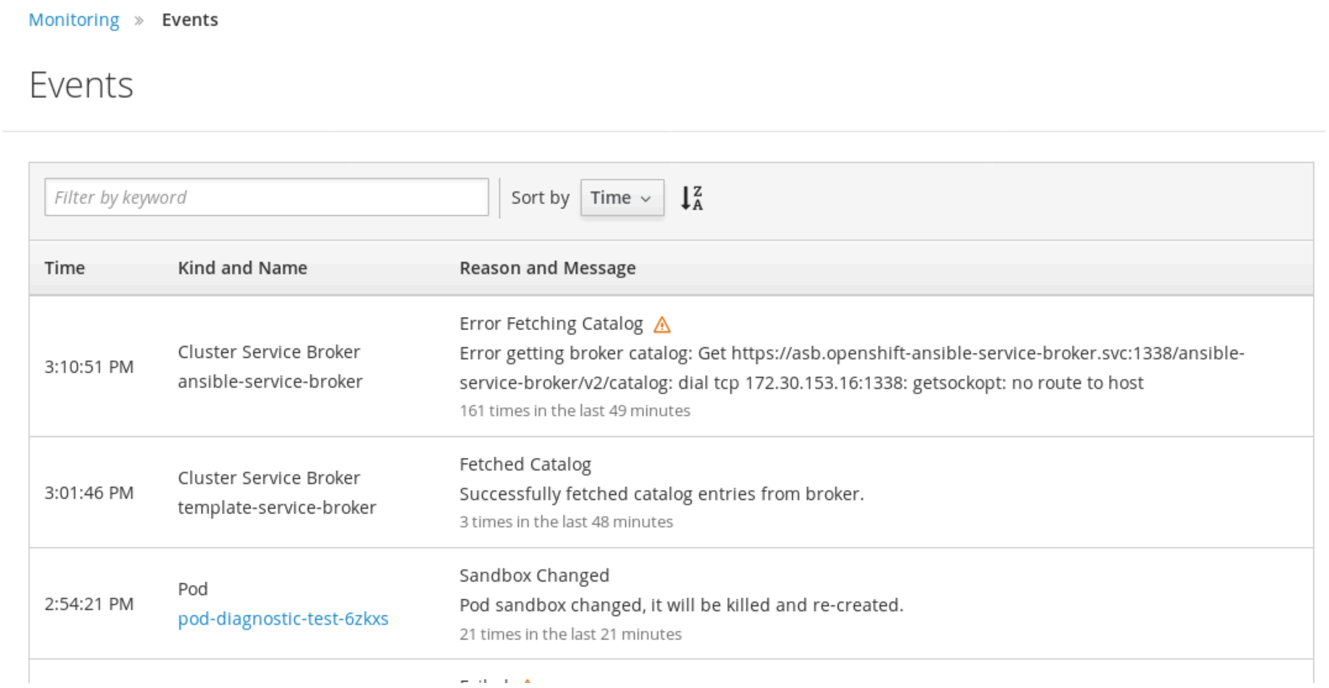
执行故障排除命令

- General Environment Information

  [root@master]

```
# sosreport -h
# sosreport -l | grep docker
# sosreport -k docker.all=on -k docker.logs=on
Press ENTER to continue, or CTRL-C to quit.
`<Enter>`
Please enter your first initial and last name [master.lab.example.com]: `<Enter>`
Please enter the case id that you are generating this report for []: `<Enter>`
...
```

- OpenShift Troubleshooting Commands
  - **`oc get events`**



Time | Kind and Name | Reason and Message
3:10:51 PM | Cluster Service Broker ansible-service-broker | Error Fetching Catalog
Error getting broker catalog: Get https://asb.openshift-ansible-service-broker.svc:1338/ansible-service-broker/v2/catalog: dial tcp 172.30.153.16:1338: getsockopt: no route to host
161 times in the last 49 minutes
3:01:46 PM | Cluster Service Broker template-service-broker | Fetched Catalog
Successfully fetched catalog entries from broker.
3 times in the last 48 minutes
2:54:21 PM | Pod pod-diagnostic-test-6zkxs | Sandbox Changed
Pod sandbox changed, it will be killed and re-created.
21 times in the last 21 minutes

  - **`oc logs`**

    - **`oc rsync`**
    - **`oc port-forward`**
- Troubleshooting Common Issues
    - Resource Limits and Quota Issues
    - Source-to-Image (S2I) Build Failures
    - ErrImagePull and ImgPullBackOff Errors
    - Incorrect Docker Configuration
    - Master and Node Service Failures
    - Failures in Scheduling Pods

引导式练习: 常见问题故障排除

[student@workstation]

```
$ lab common-troubleshoot setup

Checking prerequisites for GE: Troubleshooting Common Problems

 Checking all VMs are running:
 · master VM is up.............................................  SUCCESS
 · node1 VM is up..............................................  SUCCESS
 · node2 VM is up..............................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
 · Check router................................................  SUCCESS
 · Check registry..............................................  SUCCESS

Please wait for setup script to complete...

Overall setup status...........................................  SUCCESS
```

```
$ oc login -u developer -p redhat
Login successful.
...输出被忽略...

$ oc new-project ct
Now using project "ct" on server "https://master.lab.example.com:443".
...输出被忽略...

$ oc new-app --name=hello -i php:5.4 http://services.lab.example.com/php-helloworld
error: multiple images or templates matched "php:5.4": 2

The argument "php:5.4" could apply to the following Docker images, OpenShift image
streams, or templates:

* Image stream "php" (tag "5.6") in project "openshift"
  Use --image-stream="openshift/php:5.6" to specify this image or template
```

```

* Image stream "php" (tag "7.0") in project "openshift"
  Use --image-stream="openshift/php:7.0" to specify this image or template

$ oc describe is php -n openshift
Name:     php
Namespace:    openshift
Created:    3 days ago
Labels:     <none>
Annotations:    openshift.io/display-name=PHP
      openshift.io/image.dockerRepositoryCheck=2020-02-17T02:36:17Z
Docker Pull Spec: docker-registry.default.svc:5000/openshift/php
Image Lookup:   local=false
Unique Images:    2
Tags:     5

7.1 (latest)
  tagged from registry.lab.example.com/rhscl/php-71-rhel7:latest

  Build and run PHP 7.1 applications on RHEL 7. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-php-container/blob/master/7.1/README.md.
  Tags: builder, php
  Supports: php:7.1, php
  Example Repo: https://github.com/openshift/cakephp-ex.git

  ! error: Import failed (NotFound): dockerimage.image.openshift.io "registry.lab.example.com/rhscl/php-71-rhel7:latest" not found
      3 days ago

7.0
  tagged from registry.lab.example.com/rhscl/php-70-rhel7:latest

  Build and run PHP 7.0 applications on RHEL 7. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-php-container/blob/master/7.0/README.md.
  Tags: builder, php
  Supports: php:7.0, php
  Example Repo: https://github.com/openshift/cakephp-ex.git

  * registry.lab.example.com/rhscl/php-70-rhel7@sha256:23765e00df8d0a934ce4f2e22802bc0211a6d450bfbb69144b18cb0b51008cdd
      3 days ago

5.6
  tagged from registry.lab.example.com/rhscl/php-56-rhel7:latest

  Build and run PHP 5.6 applications on RHEL 7. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-php-container/blob/master/5.6/README.md.
  Tags: builder, php
  Supports: php:5.6, php
  Example Repo: https://github.com/openshift/cakephp-ex.git

  * registry.lab.example.com/rhscl/php-56-rhel7@sha256:920c2cf85b5da5d0701898f0ec9ee567473fa4b9af6f3ac5b2b3f863796bbd68
      3 days ago

5.5
  tagged from registry.lab.example.com/openshift3/php-55-rhel7:latest

  Build and run PHP 5.5 applications on RHEL 7. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-php-container/blob/master/5.5/README.md.
  Tags: hidden, builder, php
  Supports: php:5.5, php
  Example Repo: https://github.com/openshift/cakephp-ex.git

  ! error: Import failed (NotFound): dockerimage.image.openshift.io "registry.lab.example.com/openshift3/php-55-rhel7:latest" not found
      3 days ago

$ oc new-app --name=hello -i php:7.0 http://services.lab.example.com/php-helloworld
--> Found image c101534 (2 years old) in image stream "openshift/php" under tag "7.0" for "php:7.0"
...输出被忽略...
--> Success
    Build scheduled, use 'oc logs -f bc/hello' to track its progress.
    Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
     'oc expose svc/hello'
    Run 'oc status' to view your app.

$ oc get pods -o wide
NAME            READY     STATUS    RESTARTS   AGE       IP        NODE
hello-1-build   0/1       Pending   0          3m        <none>    <none>

$ oc logs hello-1-build

$ oc get events
LAST SEEN   FIRST SEEN   COUNT     NAME                             KIND          SUBOBJECT   TYPE      REASON                        SOURCE                   MESSAGE
15s         4m           21        hello-1-build.15f5048f138ba9d7   Pod                       Warning   FailedScheduling              default-scheduler        0/3 nodes are available: 1 MatchNodeSelector, 2 NodeNotReady.
4m          4m           1         hello.15f5048f05b63a78           BuildConfig               Warning   BuildConfigTriggerFailed   buildconfig-controller   error triggering Build for BuildConfig ct/hello: Internal error occurred: build config ct/hello has already instantiated a build for imageid registry.lab.example.com/rhscl/php-70-rhel7@sha256:23765e00df8d0a934ce4f2e22802bc0211a6d450bfbb69144b18cb0b51008cdd

$ oc describe pod hello-1-build
Name:           hello-1-build
Namespace:      ct
```

```
Node:           <none>
Labels:         openshift.io/build.name=hello-1
Annotations:    openshift.io/build.name=hello-1
                openshift.io/scc=privileged
Status:         Pending
...输出被忽略...
Events:
  Type     Reason            Age                From               Message
  ----     ------            ----               ----               -------
  Warning  FailedScheduling  12s (x26 over 6m)  default-scheduler  0/3 nodes are
available: 1 MatchNodeSelector, 2 NodeNotReady.

$ ssh master oc get nodes
NAME                      STATUS     ROLES     AGE       VERSION
master.lab.example.com    Ready      master    3d        v1.9.1+a0ce1bc657
node1.lab.example.com    `NotReady`   compute   3d        v1.9.1+a0ce1bc657
node2.lab.example.com    `NotReady`   compute   3d        v1.9.1+a0ce1bc657

$ ssh node1 systemctl status atomic-openshift-node
...输出被忽略...
Feb 20 13:27:43 node1.lab.example.com atomic-openshift-node[1987]: E0220 13:27:43.066534
   1987 generic.go:197] GenericPLEG: Unable to retrieve pods: rpc error: code = Unknown
desc = Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker
daemon `running`?
...输出被忽略...

$ ssh node1 systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; `disabled`; vendor preset:
disabled)
   Active: `inactive` (dead) since Thu 2020-02-20 13:09:35 CST; 20min ago
   ...输出被忽略...

$ ssh root@node1 systemctl start docker
$ ssh root@node2 systemctl start docker

$ oc get nodes
NAME                      STATUS     ROLES     AGE       VERSION
master.lab.example.com    Ready      master    3d        v1.9.1+a0ce1bc657
node1.lab.example.com    `Ready`     compute   3d        v1.9.1+a0ce1bc657
node2.lab.example.com    `Ready`     compute   3d        v1.9.1+a0ce1bc657
$ oc get pods
NAME            READY     STATUS     RESTARTS   AGE
hello-1-build   1/1      `Running`   0          16m

$ oc describe is
Name:     hello
Namespace:    ct
Created:    17 minutes ago
Labels:     app=hello
Annotations:    openshift.io/generated-by=OpenShiftNewApp
Docker Pull Spec: docker-registry.default.svc:5000/ct/hello
```

```
Image Lookup:   local=false
Tags:     <none>

$ oc get pods
NAME            READY     STATUS      RESTARTS   AGE
hello-1-build   0/1       Completed   0          18m
hello-1-rpdsq   1/1       Running     0          1m
```

```
$ oc delete project ct
project "ct" deleted
```

实验: 执行命令

[student@workstation]

步骤0. 准备工作

```
$ lab execute-review setup

Checking prerequisites for Lab: Executing Commands

 Checking all VMs are running:
  · master VM is up..............................................  SUCCESS
  · node1 VM is up...............................................  SUCCESS
  · node2 VM is up...............................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
  · Check router.................................................  SUCCESS
  · Check registry...............................................  SUCCESS
 Setting up for the lab:
  . Logging in as the developer user.............................  SUCCESS

Downloading files for Lab: Executing Commands

  · Downloading starter project..................................  SUCCESS
  · Downloading solution project.................................  SUCCESS

Download successful.

Please wait. Do not press any keys or interrupt the script...

  . Creating the execute-review project..........................  SUCCESS

Overall setup status.............................................  SUCCESS
```

步骤1. 下载源码，并创建新的容器

```
$ cd ~student/DO280/labs/execute-review/
$ git clone http://services/node-hello
Cloning into 'node-hello'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (5/5), done.
```

```
remote: Total 5 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.

$ cd node-hello/
$ docker build -t node-hello:latest .
Sending build context to Docker daemon 54.27 kB
Step 1/6 : FROM registry.lab.example.com/rhscl/nodejs-6-rhel7
...输出被忽略...
 ---> fba56b5381b7
Step 2/6 : MAINTAINER username "username@example.com"
 ---> Running in 5aaf97ff5aa2
 ---> 949a985ed033
Removing intermediate container 5aaf97ff5aa2
Step 3/6 : EXPOSE 3000
 ---> Running in ebcd6460831b
 ---> 74026107ed62
Removing intermediate container ebcd6460831b
Step 4/6 : COPY . /opt/app-root/src
 ---> fd4305160490
Removing intermediate container a4ff44c9afae
Step 5/6 : RUN source scl_source enable rh-nodejs6 &&     npm install --registry=http://services.lab.example.com:8081/nexus/content/groups/nodejs/
 ---> Running in abe388900635
...输出被忽略...
Removing intermediate container c6e9c47f4271
Successfully built 1510f143594f

$ docker images
REPOSITORY                                      TAG                 IMAGE ID            CREATED             SIZE
node-hello                                      latest              1510f143594f        2 minutes ago       495 MB
registry.lab.example.com/rhscl/nodejs-6-rhel7   latest              fba56b5381b7        2 years ago         489 MB

$ docker tag 1510f143594f registry.lab.example.com/node-hello:latest
$ docker images
REPOSITORY                                      TAG                 IMAGE ID            CREATED             SIZE
registry.lab.example.com/node-hello             latest              1510f143594f        7 minutes ago       495 MB
node-hello                                      latest              1510f143594f        7 minutes ago       495 MB
registry.lab.example.com/rhscl/nodejs-6-rhel7   latest              fba56b5381b7        2 years ago         489 MB

$ docker push registry.lab.example.com/node-hello
The push refers to a repository [registry.lab.example.com/node-hello]
f69f4e98b676: Pushed
d63c2be05424: Pushed
82dfac496b77: Mounted from rhscl/nodejs-6-rhel7
aa29c7023a3c: Mounted from rhscl/nodejs-6-rhel7
```

```
45f0d85c3257: Mounted from rhscl/nodejs-6-rhel7
5444fe2e6b50: Mounted from rhscl/nodejs-6-rhel7
d4d408077555: Mounted from rhscl/nodejs-6-rhel7
latest: digest: sha256:4db31968b9d1e6f362691ac118bfd021da9864b6b7671b02dd953e9510eb6672
size: 1790
$ docker-registry-cli registry.lab.example.com search hello ssl
available options:-

-----------
1) Name: node-hello
Tags: latest
-----------
2) Name: openshift/hello-openshift
Tags: latest

2 images found !
$ cd
```

步骤2. 新建应用

```
$ oc login -u developer -p redhat
Login successful.

You have one project on this server: "execute-review"

Using project "execute-review".

$ oc new-app registry.lab.example.com/node-hello --name hello
--> Found Docker image 1510f14 (11 minutes old) from registry.lab.example.com for
"registry.lab.example.com/node-hello"
...输出被忽略...
--> Creating resources ...
    imagestream "hello" created
    deploymentconfig "hello" created
    service "hello" created
--> Success
    Application is not exposed. You can expose services to the outside world by executing
one or more of the commands below:
     'oc expose svc/hello'
    Run 'oc status' to view your app.

$ oc get all
NAME                       REVISION   DESIRED   CURRENT   TRIGGERED BY
deploymentconfigs/hello    1          1         1         config,image(hello:latest)

NAME                 DOCKER REPO                                          TAGS
UPDATED
imagestreams/hello   docker-registry.default.svc:5000/execute-review/hello   latest    2
minutes ago

NAME                READY     STATUS              RESTARTS   AGE
po/hello-1-deploy   1/1       Running             0          2m
```

```
po/hello-1-zjnpz    0/1       ImagePullBackOff   0          2m

NAME         DESIRED   CURRENT   READY     AGE
rc/hello-1   1         1         0         2m

NAME        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)             AGE
svc/hello   ClusterIP   172.30.31.210   <none>        3000/TCP,8080/TCP   2m
```

步骤3. 检查日志

```
$ oc logs hello-1-zjnpz
Error from server (BadRequest): container "hello" in pod "hello-1-zjnpz" is waiting to start: trying and failing to pull image

$ oc describe pod hello-1-deploy
Name:          hello-1-deploy
Namespace:     execute-review
Node:          node1.lab.example.com/172.25.250.11
Start Time:    Thu, 20 Feb 2020 14:11:12 +0800
Labels:        openshift.io/deployer-pod-for.name=hello-1
Annotations:   openshift.io/deployment-config.name=hello
               openshift.io/deployment.name=hello-1
               openshift.io/scc=restricted
Status:        Running
IP:            10.129.0.36
...输出被忽略...

$ oc get events --sort-by='.metadata.creationTimestamp'
...输出被忽略...
11m         11m          2         hello-1-zjnpz.15f5077d690a1553    Pod
  spec.containers{hello}        Warning   Failed                        kubelet, node2.lab.example.com   Failed to pull image "registry.lab.example.com/node-hello@sha256:4db31968b9d1e6f362691ac118bfd021da9864b6b7671b02dd953e9510eb6672": rpc error: code = Unknown desc = All endpoints blocked.
10m         11m          7         hello-1-zjnpz.15f5077d8b201680    Pod
                                Normal    SandboxChanged                kubelet, node2.lab.example.com   Pod sandbox changed, it will be killed and re-created.
11m         11m          2         hello-1-zjnpz.15f5077d690a70b2    Pod
  spec.containers{hello}        Warning   Failed                        kubelet, node2.lab.example.com   Error: ErrImagePull
11m         11m          2         hello-1-zjnpz.15f5077d688d3bbd    Pod
  spec.containers{hello}        Normal    Pulling                       kubelet, node2.lab.example.com   pulling image "registry.lab.example.com/node-hello@sha256:4db31968b9d1e6f362691ac118bfd021da9864b6b7671b02dd953e9510eb6672"
6m          11m          32        hello-1-zjnpz.15f5077e2faaa807    Pod
  spec.containers{hello}        Warning   Failed                        kubelet, node2.lab.example.com   Error: ImagePullBackOff
10m         11m          5         hello-1-zjnpz.15f5077e2faa50f5    Pod
  spec.containers{hello}        Normal    BackOff                       kubelet, node2.lab.example.com   Back-off pulling image "registry.lab.example.com/node-hello@sha256:4db31968b9d1e6f362691ac118bfd021da9864b6b7671b02dd953e9510eb6672"
```

```
1m          1m          1         hello-1.15f50808c0e12081          ReplicationController
                                Normal     SuccessfulDelete              replication-
controller          Deleted pod: hello-1-zjnpz
1m          1m          1         hello.15f50808bde47f2c            DeploymentConfig
                                 Normal     ReplicationControllerScaled   deploymentconfig-
controller      Scaled replication controller "hello-1" from 1 to 0
```

步骤4. 排错

```
$ oc get dc hello -o yaml
...输出被忽略...
    spec:
      containers:
      - image: registry.lab.example.com/node-
hello@sha256:4db31968b9d1e6f362691ac118bfd021da9864b6b7671b02dd953e9510eb6672
        imagePullPolicy: Always
        name: hello
        ports:
        - containerPort: 3000
          protocol: TCP
        - containerPort: 8080
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
...输出被忽略...

$ oc get pods -o wide
NAME              READY     STATUS             RESTARTS   AGE       IP              NODE
hello-1-deploy    1/1       Running            0          9m        10.129.0.123
node1.lab.example.com
hello-1-nzdjs     0/1       ImagePullBackOff   0          9m        10.128.0.108
node2.lab.example.com

$ ssh root@node1
# vim /etc/sysconfig/docker
...
BLOCK_REGISTRY='--block-registry registry.access.redhat.com --block-registry docker.io'
# systemctl restart docker
# exit

$ ssh root@node2
# vim /etc/sysconfig/docker
...
BLOCK_REGISTRY='--block-registry registry.access.redhat.com --block-registry docker.io'
# systemctl restart docker
# exit
```

步骤5. 排错后回滚

```
$ oc rollout latest hello
deploymentconfig "hello" rolled out

$ oc get pods -o wide
NAME              READY     STATUS     RESTARTS    AGE       IP            NODE
hello-1-deploy    0/1       Error      0           31m       10.129.0.36
node1.lab.example.com
hello-2-w7n8x     1/1       Running    0           2m        10.128.0.60
node2.lab.example.com

$ oc logs hello-2-w7n8x
nodejs server running on http://0.0.0.0:3000
```

步骤6. 测试

```
$ oc expose svc hello --hostname=hello.apps.lab.example.com
route "hello" exposed

$ curl http://hello.apps.lab.example.com
Hi! I am running on host -> hello-2-bcf2q
```

步骤7. 评估

```
 $ lab execute-review grade

Grading the student's work for Lab: Executing Commands

  · Check if the hello pod is in Running state..................  PASS
  · Check if Docker image is present............................  PASS
  · Check if Docker image is pushed.............................  PASS
  . Checking if docker configuration on node1 is fixed..........  PASS
  . Checking if docker configuration on node2 is fixed..........  PASS
  . Checking if route can be invoked successfully...............  PASS

Overall exercise grade..........................................  PASS
```

步骤8. 清除

```
$ oc delete project execute-review
project "execute-review" deleted
```

总结

- 红帽 OpenShift 容器平台提供 **oc** 命令行客户端，可以查看、编辑和管理 OpenShift 集群中的资源。
- 在具有有效订阅的红帽企业 Linux（RHEL）系统上，此工具作为 RPM 文件提供，可通过 yum install 命令安装。
- 对于其它的 Linux 分发和其他操作系统，如 Windows 和macOS，可以从红帽客户门户下载原生客户端。
- 有几个基本命令可用于管理 OpenShift 资源，例如：

    - **`oc get resourceType resourceName`**：输出包含 resourceName 的重要信息的摘要。
    - **`oc describe resourceType resourceName`**：输出 resourceName详细信息。
    - **`oc create`**：从某一输入创建资源，如文件或输入流。
    - **`oc delete resourceType resourceName`**：从 OpenShift 删除资源。
- **`oc new-app`** 命令可以许多不同的方式创建在 OpenShift 中运行的应用 Pod。它可以从现有的 Docker 镜像或 Dockerfile 创建 Pod，或通过 Source-to-Image（S2I）流程从原始的源代码创建。
- **`oc get events`** 命令提供 OpenShift 命名空间内事件的相关信息。事件在故障排除期间很有用处。管理员可以获取关于集群中故障和问题的高级信息。
- **`oc logs`** 命令检索特定构建、部署和 Pod 的日志输出。此命令适用于构建、构建配置、部署配置和 Pod。
- **`oc rsh`** 命令开启与容器连接的远程 shell 会话。这可用于登录正在运行的容器并调查其中的问题。
- **`oc rsync`** 命令将内容复制到正在运行的 Pod 内的某一目录，或从中复制内容。如果 Pod 具有多个容器，你可以使用 -c 选项指定容器 ID。否则，默认为 Pod 中的第一个容器。这可用于从容器传输日志文件和配置文件非常有用。
- 你可以使用 **`oc port-forward`** 命令将一个或多个本地端口转发到 Pod。这样，你可以在本地监听一个指定或随机端口，并且与 Pod 中的给定端口来回转发数据。

5. 控制 OpenShift 资源的访问

保护 OpenShift 资源的访问

- Kubernetes Namespaces
    - Projects
- Cluster Administration

```
$ oc adm policy \
  remove-cluster-role-from-group self-provisioner \
  system:authenticated system:authenticated:oauth

$ oc adm policy \
  add-cluster-role-to-group self-provisioner \
  system:authenticated system:authenticated:oauth
```

    - Creating a Project

```
$ oc new-prject demoproject --description="Demo"
```

- Introducing Roles in Red Hat OpenShift Container Platform

    - 常规用户
    - 系统用户
    - 服务帐户

- Reading Local Policies
- Managing Role Bindings

```
$ oc adm policy who-can VERB RESOURCE

$ oc adm policy add-role-to-user ROLE USERNAME
$ oc adm policy remove-role-from-user ROLE USERNAME
```

```
$ oc adm policy add-cluster-role-to-user ROLE USERNAME
$ oc adm policy remove-cluster-role-from-user ROLE USERNAME
```

- Security Context Constraints (SCCs)

```
$ oc get scc
$ oc describe scc scc_name

$ oc adm policy add-scc-to-user scc_name user_name
$ oc adm policy remove-scc-from-user scc_name user_name
```

- Use Case for a Service Account

```
$ oc create sa useroot

$ oc patch dc/demo-app \
  --patch \
  '{"spec":{"templdate":{"spec":{"serviceAccountName": "useroot"}}}}'

*$ oc adm policy add-scc-to-user anyuid -z useroot
```

- Managing User Membership
    - Membership Management Using the Web Console

    - Membership Management Using the CLI

    ```
    $ oc create user demo-user
    # htpasswd /etc/origin/openshift-passwd demo-user

    $ oc project test
    $ oc policy add-role-to-user edit demo-user
    $ oc policy remove-role-from user edit demo-user

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

- Authentication and Authorization Layers
  - Users and Groups
  - Authentication Tokens

    ```
    $ oc whoami
    ```

- Authentication Types

  - 基础身份验证
  - 请求标头身份验证
  - Keystone 身份验证
  - LDAP 身份验证
  - GitHub 身份验证

引导式练习: 管理项目和帐户

[student@workstation]
步骤0. 准备

```
$ lab secure-resources setup

Checking prerequisites for GE: Managing projects and accounts

 Checking all VMs are running:
 · master VM is up.............................................  SUCCESS
 · node1 VM is up..............................................  SUCCESS
 · node2 VM is up..............................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
 · Check router................................................  SUCCESS
 · Check registry..............................................  SUCCESS

Downloading files for GE: Managing projects and accounts

 · Download exercise files.....................................  SUCCESS

Overall setup status...........................................  SUCCESS
```

步骤1. 创建用户

```
$ ssh root@master
# htpasswd -b /etc/origin/master/htpasswd user1 redhat
Adding password for user user1
# htpasswd -b /etc/origin/master/htpasswd user2 redhat
Adding password for user user2
# logout
Connection to master closed.
```

步骤2. 配置用户

```
$ oc login -u admin -p redhat
Login successful.
...输出被忽略...
Using project "default".

$ oc adm policy remove-cluster-role-from-group \
self-provisioner system:authenticated:oauth
cluster role "self-provisioner" removed: "system:authenticated:oauth"
```

步骤3. 验证

```
$ oc login -u user1 -p redhat
Login successful.

You don't have any projects. Contact your system administrator to request a project.

$ oc new-project test
Error from server (Forbidden): You may not request a new project via this API.
```

步骤4. 管理员创建项目

```
$ oc login -u admin -p redhat
Login successful.
...输出被忽略...
Using project "default".

$ oc new-project project-user1
Now using project "project-user1" on server "https://master.lab.example.com:443".
...输出被忽略...

$ oc new-project project-user2
Now using project "project-user2" on server "https://master.lab.example.com:443".
...输出被忽略...
```

步骤5. 在项目中分配用户

```
$ oc project project-user1
Now using project "project-user1" on server "https://master.lab.example.com:443".
$ oc policy add-role-to-user admin user1
role "admin" added: "user1"
$ oc policy add-role-to-user edit user2
role "edit" added: "user2"

$ oc project project-user2
Now using project "project-user2" on server "https://master.lab.example.com:443".
$ oc policy add-role-to-user edit user2
role "edit" added: "user2"
```

步骤6. 测试

```
$ oc login -u user1 -p redhat
Login successful.
```

```

You have one project on this server: "project-user1"

Using project "project-user1".

$ oc project project-user2
error: You are not a member of project "project-user2".
You have one project on this server: project-user1

$ oc login -u user2 -p redhat
Login successful.

You have access to the following projects and can switch between them with 'oc project <projectname>':

  * project-user1
    project-user2

Using project "project-user1".

$ oc project project-user2
Now using project "project-user2" on server "https://master.lab.example.com:443".
```

步骤7. 确认布署

```
$ oc project project-user1
Now using project "project-user1" on server "https://master.lab.example.com:443"

$ oc new-app --name=nginx --docker-image=registry.lab.example.com/nginx:latest
--> Found Docker image c825216 (19 months old) from registry.lab.example.com for "registry.lab.example.com/nginx:latest"
...输出被忽略...
    * WARNING: Image "registry.lab.example.com/nginx:latest" runs as the 'root' user which may not be permitted by your cluster administrator
...输出被忽略...

$ oc get pods
NAME           READY     STATUS             RESTARTS   AGE
nginx-1-6rd7w   0/1       CrashLoopBackOff   3          1m
```

步骤8. 减少特定项目的安全限制

```
$ oc login -u user1 -p redhat
Login successful.
...输出被忽略...
Using project "project-user1".
$ oc create serviceaccount useroot
serviceaccount "useroot" created

$ oc login -u admin -p redhat
Login successful.
...输出被忽略...
```

```
$ oc project project-user1
Already on project "project-user1" on server "https://master.lab.example.com:443".
$ oc adm policy add-scc-to-user anyuid -z useroot
scc "anyuid" added to: ["system:serviceaccount:project-user1:useroot"]

$ oc login -u user2 -p redhat
Login successful.
...输出被忽略...
Using project "project-user1".
$ oc patch dc/nginx --patch '{"spec":{"template":{"spec":{"serviceAccountName":
"useroot"}}}}'
deploymentconfig "nginx" patched
$ oc get pods
NAME            READY     STATUS        RESTARTS   AGE
nginx-1-6rd7w    0/1       Terminating   6          10m
nginx-2-tr29b    1/1       Running       0          50s
```

步骤9. 测试容器

```
$ oc expose svc nginx
route "nginx" exposed

$ curl -s http://nginx-project-user1.apps.lab.example.com
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

步骤10. 清理

```
$ oc login -u admin -p redhat
```

```
Login successful.
...输出被忽略...
Using project "project-user1".

$ oc adm policy add-cluster-role-to-group self-provisioner system:authenticated
system:authenticated:oauth
cluster role "self-provisioner" added: ["system:authenticated"
"system:authenticated:oauth"]

$ oc delete project project-user1
project "project-user1" deleted
$ oc delete project project-user2
project "project-user2" deleted

$ ssh root@master htpasswd -D /etc/origin/master/htpasswd user1
Deleting password for user user1
$ ssh root@master htpasswd -D /etc/origin/master/htpasswd user2
Deleting password for user user2
```

利用机密管理敏感信息

- Secrets

  > 提供用于存放敏感信息的机制
  >
  > 利用卷插件 将机密挂载到容器上，或者系统可以使用机密代表 Pod 执行操作

  - Features of Secrets

    > - 可以独立其定义被引用
    > - 由临时文件存储提供支持
    > - 可以在全名空间内共享

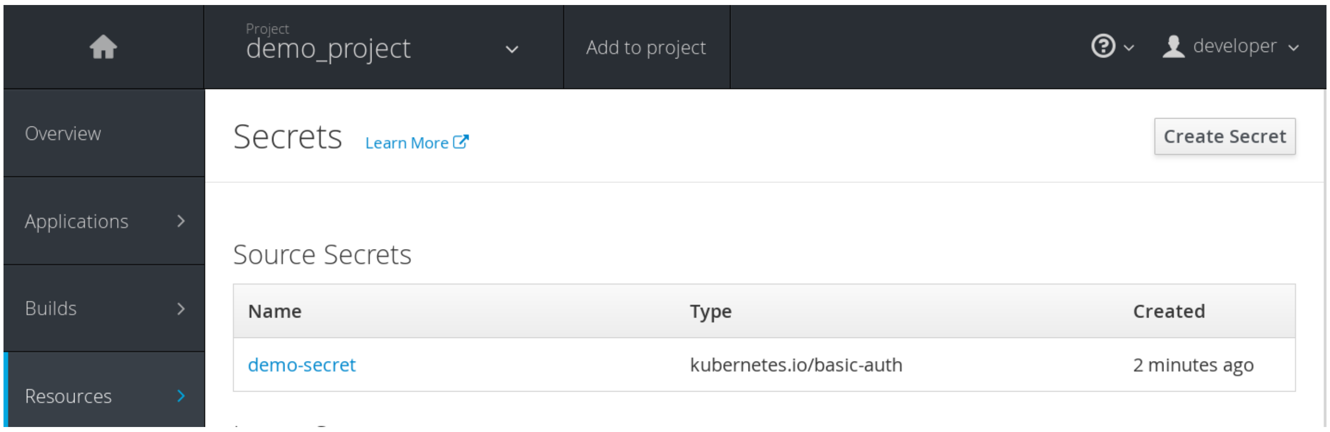
  - Creating a Secret

    > 先创建机密，后创建 Pod

  - How Secrets are Exposed to Pods

    > 先创建机密，环境变量引用

  - Managing Secrets from the Web Console



Secrets

Source Secrets

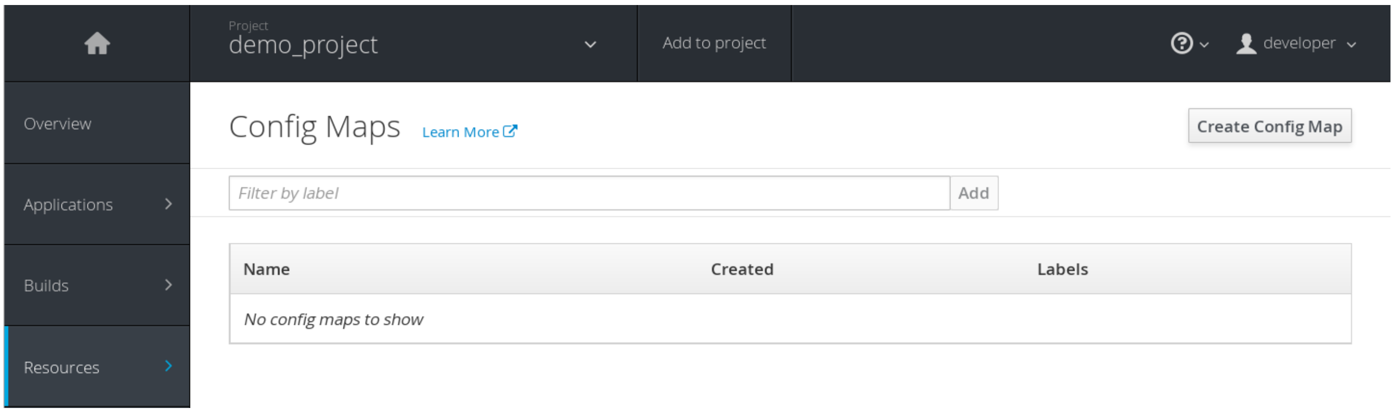
Name	Type	Created
demo-secret	kubernetes.io/basic-auth	2 minutes ago

- Use Cases for Secrets
    - Passwords and User Names
    - Transport Layer Security (TLS) and Key Pairs
- ConfigMap Objects
    - Creating a ConfigMap from the CLI

```
$ oc create configmap special-config \
  --from-literal=serverAddress=172.20.30.40
```

```
env:
  - name: APISERVER
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: serverAddress
```

- Managing ConfigMaps from the Web Console



Config Maps

Name	Created	Labels
No config maps to show		

引导式练习: 保护数据库密码

[student@workstation]
步骤0. 准备

```
$ lab secure-secrets setup

Checking prerequisites for GE: Protecting a Database Password

 Checking all VMs are running:
 · master VM is up.........................................  SUCCESS
 · node1 VM is up..........................................  SUCCESS
 · node2 VM is up..........................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
 · Check router............................................  SUCCESS
 · Check registry..........................................  SUCCESS

Downloading files for GE: Protecting a Database Password

 · Download exercise files.................................  SUCCESS

Overall setup status.......................................  SUCCESS
```

步骤1. 创建新项目

```
$ oc login -u developer -p redhat
Login successful.

You don't have any projects. You can try to create a new project, by running

    oc new-project <projectname>

$ oc new-project secure-secrets
Now using project "secure-secrets" on server "https://master.lab.example.com:443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git

to build a new example application in Ruby.

$ cd DO280/labs/secure-secrets
$ less mysql-ephemeral.yml
...输出被忽略...
      spec:
        containers:
        - capabilities: {}
          env:
          - name: MYSQL_USER
            valueFrom:
              secretKeyRef:
                key: database-user
```

```
                  name: ${DATABASE_SERVICE_NAME}
            - name: MYSQL_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: database-password
                  name: ${DATABASE_SERVICE_NAME}
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: database-root-password
                  name: ${DATABASE_SERVICE_NAME}
            - name: MYSQL_DATABASE
              value: ${MYSQL_DATABASE}
              ...输出被忽略...
parameters:
...输出被忽略...
- description: The name of the OpenShift Service exposed for the database.
  displayName: Database Service Name
  name: DATABASE_SERVICE_NAME
  required: true
  value: mysql
...输出被忽略...
```

步骤2. 根据模板的请求，创建包含MySQL容器映像使用的凭据的机密

```
$ oc create secret generic mysql \
  --from-literal='database-user'='mysql' \
  --from-literal='database-password'='redhat' \
  --from-literal='database-root-password'='do280-admin'
secret "mysql" created

$ oc get secret mysql -o yaml
apiVersion: v1
data:
  database-password: cmVkaGF0
  database-root-password: ZG8yODAtYWRtaW4=
  database-user: bXlzcWw=
kind: Secret
...输出被忽略...
```

步骤3. 创建数据库 MySQL 容器

```
$ oc new-app --file=mysql-ephemeral.yml
--> Deploying template "secure-secrets/mysql-ephemeral" for "mysql-ephemeral.yml" to project secure-secrets

     MySQL (Ephemeral)
     ---------
     MySQL database service, without persistent storage. For more information about using this template, including OpenShift considerations, see https://github.com/sclorg/mysql-container/blob/master/5.7/README.md.

     WARNING: Any data stored will be lost upon pod destruction. Only use this template for testing

     The following service(s) have been created in your project: mysql.
      Connection URL: mysql://mysql:3306/

     For more information about using this template, including OpenShift considerations, see https://github.com/sclorg/mysql-container/blob/master/5.7/README.md.

     * With parameters:
        * Memory Limit=512Mi
        * Namespace=openshift
        * Database Service Name=mysql
        * MySQL Database Name=sampledb
        * Version of MySQL Image=5.7

--> Creating resources ...
    service "mysql" created
    deploymentconfig "mysql" created
--> Success
    Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
     'oc expose svc/mysql'
    Run 'oc status' to view your app.
```

步骤4. 等待 Pod 运行

```
$ oc get pods -w
NAME            READY     STATUS    RESTARTS   AGE
mysql-1-wg5jj   1/1       Running   0          1m
```

步骤5. 创建到 MySQL pod 的端口转发隧道

```
$ oc port-forward mysql-1-wg5jj 3306:3306 &
Forwarding from 127.0.0.1:3306 -> 3306
```

步骤6. 连接数据库，确认访问

```
$ echo show databases | mysql -u root -pdo280-admin -h 127.0.0.1
Database
information_schema
mysql
performance_schema
sampledb
sys
```

步骤7. 清理

```
$ kill %1
$ oc delete project secure-secrets
project "secure-secrets" deleted
```

管理安全策略

- Red Hat OpenShift Container Platform Authorization

Default Roles	Description
cluster-admin	集群中的用户可以管理集群
cluster-status	集群中的用户可以查看集群的信息

Default Roles	Description
edit	创建、更改、删除应用资源
basic-user	访问项目
self-provisioner	创建项目
admin	管理所有资源，包括授权

- User Types

  - Regular users `devops`
  - System users `system:admin`
  - Service accounts `system:serviceaccount:default:deployer`

- Security Context Constraints (SCCs)

  SCC 限制 OpenShift 中正在运行的 pod 对主机环境的访问。SCC 控制：

  - 运行特权容器
  - 使用主机目录作为卷向容器请求额外功能
  - 更改容器的 SELinux 上下文
  - 更改用户 ID

  OpenShift 有七种 SCCs

  - anyuid
  - hostaccess
  - hostmount-anyuid
  - nonroot
  - privileged restricted

- OpenShift and SELinux

```
$ oc export scc restricted
```

- Privileged Containers

  某些容器可能需要访问主机的运行时环境

测验: 管理安全策略

选择以下问题的正确答案:

1. 哪一命令可以从 `student` 用户移除 `cluster-admin` 角色?

   a. oc adm policy delete-cluster-role-from-user cluster-admin student

   b. oc adm policy rm-cluster-role-from-user cluster-admin student

   c. oc adm policy remove-cluster-role-from-user cluster-admin student

   d. oc adm policy del-cluster-role-from-user cluster-admin student

2. 哪一命令可以向 `example` 项目中的 `student` 用户添加 `admin` 角色?

   a. oc adm policy add-role-to-user owner student -p example

   b. oc adm policy add-role-to-user cluster-admin student -n example

   c. oc adm policy add-role-to-user admin student -p example

   d. oc adm policy add-role-to-user admin student -n example

3. 哪一命令为 `developers` 组中的用户提供 `example` 项目的只读访问权限?

   a. oc adm policy add-role-to-group view developers -n example

   b. oc adm policy add-role-to-group view developers -p example

   c. oc adm policy add-role-to-group display developers -p example

   d. oc adm policy add-role-to-user display developers -n example

4. 哪一命令可以获取能够对节点资源执行 `get` 操作的所有用户的列表?

   a. oc adm policy who-can get

   b. oc adm policy roles all

   c. oc adm policy who-can get nodes

   d. oc adm policy get nodes users

实验: 控制 OpenShift 资源的访问

[student@workstation]
步骤0. 准备

```
$ cd
$ lab secure-review setup

```

```
Checking prerequisites for Controlling Access to OpenShift Resources

 Checking all VMs are running:
  · master VM is up.............................................  SUCCESS
  · node1 VM is up..............................................  SUCCESS
  · node2 VM is up..............................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
  · Check router................................................  SUCCESS
  · Check registry..............................................  SUCCESS

Downloading files for Controlling Access to OpenShift Resources

  · Download exercise files.....................................  SUCCESS

Overall setup status............................................  SUCCESS
```

步骤1. 创建用户

```
$ ssh root@master htpasswd -b /etc/origin/master/htpasswd user-review redhat
Adding password for user user-review
```

步骤2. 禁用所有常规用户的项目创建功能

```
$ oc login -u admin -p redhat
Login successful.
...输出被忽略...

$ oc adm policy remove-cluster-role-from-group self-provisioner system:authenticated system:authenticated:oauth
cluster role "self-provisioner" removed: ["system:authenticated" "system:authenticated:oauth"]
```

步骤3. 验证常规用户无法在 OpenShift 中创建项目

```
$ oc login -u user-review -p redhat
Login successful.
...输出被忽略...

$ oc new-project test
Error from server (Forbidden): You may not request a new project via this API.
```

步骤4. 创建项目

```
$ oc login -u admin -p redhat
Login successful.
...输出被忽略...

$ oc new-project secure-review
Now using project "secure-review" on server "https://master.lab.example.com:443".
...输出被忽略...
```

步骤5. 将用户与项目关联

```
$ oc policy add-role-to-user edit user-review
role "edit" added: "user-review"
```

步骤6. 用提供的模板布署数据库

```
$ cd ~/DO280/labs/secure-review/
$ less mysql-ephemeral.yml
...输出被忽略...
      spec:
        containers:
        - capabilities: {}
          env:
          - name: MYSQL_USER
            valueFrom:
              secretKeyRef:
                key: database-user
                name: ${DATABASE_SERVICE_NAME}
          - name: MYSQL_PASSWORD
            valueFrom:
              secretKeyRef:
                key: database-password
                name: ${DATABASE_SERVICE_NAME}
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                key: database-root-password
                name: ${DATABASE_SERVICE_NAME}
...输出被忽略...
parameters:
...输出被忽略..
- description: The name of the OpenShift Service exposed for the database.
  displayName: Database Service Name
  name: DATABASE_SERVICE_NAME
  required: true
  value: mysql
  ...输出被忽略..
```

步骤7. 使用开发人员身份，创建机密

```
$ oc create secret generic mysql \
  --from-literal=database-user=mysql \
  --from-literal=database-password=redhat \
  --from-literal=database-root-password=do280-admin
secret "mysql" created

$ oc get secret mysql -o yaml
apiVersion: v1
data:
  database-password: cmVkaGF0
  database-root-password: ZG8yODAtYWRtaW4=
  database-user: bXlzcWw=
kind: Secret
```

```
...输出被忽略..
```

步骤8. 使用模板创建数据库容器

```
$ oc new-app --file=mysql-ephemeral.yml
--> Deploying template "secure-review/mysql-ephemeral" for "mysql-ephemeral.yml" to project
secure-review
...输出被忽略..
$ oc get pods
NAME            READY     STATUS    RESTARTS   AGE
mysql-1-cfm5l   1/1       Running   0          23s
```

步骤9. 测试数据库服务器

```
$ oc port-forward mysql-1-cfm5l 3306:3306 &
Forwarding from 127.0.0.1:3306 -> 3306

$ echo show databases | mysql -u mysql -predhat -h 127.0.0.1
Database
information_schema
sampledb
```

步骤10. 布署容器

```
$ oc new-app --name=phpmyadmin \
  --docker-image=registry.lab.example.com/phpmyadmin/phpmyadmin:4.7 \
  -e PMA_HOST=mysql.secure-review.svc.cluster.local
--> Found Docker image f51fd61 (23 months old) from registry.lab.example.com for
"registry.lab.example.com/phpmyadmin/phpmyadmin:4.7"
...输出被忽略..

$ oc get pods
NAME                 READY     STATUS             RESTARTS   AGE
mysql-1-cfm5l        1/1       Running            0          8m
phpmyadmin-1-79xdq   0/1       CrashLoopBackOff   5          4m
```

步骤11. 降低项目的安全性限制

```
$ oc login -u admin -p redhat
Login successful.
...输出被忽略..

$ oc create serviceaccount phpmyadmin-account
serviceaccount "phpmyadmin-account" created

$ oc get scc
*$ oc adm policy add-scc-to-user \
   anyuid \
   -z phpmyadmin-account
scc "anyuid" added to: ["system:serviceaccount:secure-review:phpmyadmin-account"]

```

```
$ oc patch dc/phpmyadmin --patch '{"spec":{"template":{"spec":{"serviceAccountName":
"phpmyadmin-account"}}}}'
deploymentconfig "phpmyadmin" patched

$ oc login -u user-review -p redhat
Login successful.
...输出被忽略..

$ oc get pods
NAME                 READY     STATUS        RESTARTS   AGE
mysql-1-cfm5l        1/1       Running       0          14m
phpmyadmin-1-79xdq   0/1       Terminating   6          11m
phpmyadmin-2-bl2t8   1/1       Running       0          29s
```

步骤12. 通过 Web 浏览器测试应用

```
$ oc expose svc/phpmyadmin --hostname=phpmyadmin.apps.lab.example.com
route "phpmyadmin" exposed

$ sudo yum install -y elinks
...输出被忽略..
Complete!
$ elinks -dump http://phpmyadmin.apps.lab.example.com
   [1]phpMyAdmin

                             Welcome to phpMyAdmin

   Javascript must be enabled past this point!
   Language [[2]________________________________]
   Log in[3]Documentation
   Username: [4]_________________________
   Password: [5]_________________________
   [6][ Go ]
...输出被忽略..
```

步骤13. 运行评分脚本，验证

```
$ lab secure-review grade

Grading the student's work for Controlling Access to OpenShift Resources

· Check whether file /etc/origin/master/htpasswd exists........  PASS
· Check whether the username user-review exists................  PASS
· Check whether the password for the user-review...............  PASS
· Check whether the project autocreation was removed for users authenticated  PASS
· Check whether the project autocreation was removed...........  PASS
· Check whether the project secure-review was created..........  PASS
· Check whether the user-review can create apps in secure-review  PASS
· Check secret was created.....................................  PASS
· Check mysql pod was created..................................  PASS
· Check whether the service account phpmyadmin-account was created  PASS
· Check whether the SCC for the serviceaccount was bound to anyuid  PASS
· Check whether the DeploymentConfig was changed...............  PASS
· Check phpmyadmin was redeployed............................  PASS
· Check phpmyadmin route was created..........................  PASS
```

步骤14. 清理

```
$ oc login -u admin -p redhat
Login successful.
...输出被忽略..

$ oc adm policy add-cluster-role-to-group self-provisioner system:authenticated
system:authenticated:oauth
cluster role "self-provisioner" added: ["system:authenticated"
"system:authenticated:oauth"]

$ oc delete project secure-review
project "secure-review" deleted

$ ssh root@master htpasswd -D /etc/origin/master/htpasswd user-review
Deleting password for user user-review

$ oc delete user user-review
user "user-review" deleted

$ kill %1
```

总结

- Kubernetes 命名空间提供将集群中的一组相关的资源分组在一起的方式。项目是一种 Kubernetes 命名空间；通过项目，一组授权用户可以组织和管理项目资源，并与其他群组区隔开来。 **`-n`**
- 集群管理员可以创建项目，并将项目的管理权限委托给任何用户。管理员可以授予用户特定项目的访问权限，让他们能够创建自己的项目，还可以授予他们个别项目中的管理权限。
- 身份验证层确定与对 OpenShift 容器平台 API 的请求关联的用户。然后，授权层使用与发出请求的用户相关的信息确定是否应当允许其请求。
- OpenShift 提供安全性上下文约束（SCC），它可以控制 pod 能够执行的操作以及有权访问的资源。默认情况下，在创建容器器后，它仅具有受限制的 SCC 所定义的功能。
  `oc get scc` 命令列出可用的 SCC。
  `oc describe scc` 命令显示安全性上下文约束的详细描述。
- Secret 对象类型提供用于存放敏感信息的机制，如密码、OpenShift 容器平台客户端配置文件、dockercfg 文件，以及私有源存储库凭据。机密将敏感内容与 Pod 分隔开。你可以利用卷插件将机密装载到容器上，或者系统可以使用机密代表Pod 执行操作。
- ConfigMaps 类似于 secrets，但设计为支持与不包含敏感信息的字符串搭配使用。
- OpenShift 定义用户可以执行的两大类操作：项目相关（也称为本地策略）操作和与管理相关（也称为集群策略）操作。

- **OpenShift 要求在各个主机上启用 SELinux，从而使用强制访问控制来提供资源的安全访问。类似地，由 OpenShift 管理的 Docker 容器需要管理 SELinux 上下文来避免兼容性问题。**

6. 分配持久存储

调配持久存储

- Persistent Storage

  默认情况下，运行容器使用容器内的临时存储。

  使用临时存储意味着当容器停止时，写入容器内文件系统的数据将丢失。

  - Use Case for Persistent Storage

    如果使用持久存储，则数据库将数据存储到 pod 外部的持久卷。如果 pod 被销毁并重新创建，数据库应用程序将继续访问存储数据的同一外部存储器。

- Providing Persistent Storage for an Application

  pv 持久卷是 OpenShift 资源，只有 OpenShift 管理员才能创建和销毁这些资源。

  - Persistent Storage Components

    OpenShift 容器平台使用 Kubernetes 持久卷 **PV** 框架来允许管理员为集群提供持久存储。

    开发人员使用持久卷声明 **PVC** 来请求 PV资源

  - OpenShift-supported Plug-ins for Persistent Storage

    OpenShift 使用插件为持久性存储支持以下不同的后端：

    - **NFS** 『RH358』
    - iSCSI 『RH358』
    - GlusterFS 『RH236』
    - OpenStack Cinder 『CL210』
    - Ceph RBD 『CEPH125-=>CL260』
    - AWS 弹性块存储（EBS）
    - Azure 磁盘和 Azure 文件
    - VMWare vSphere

    - GCE 持久磁盘
    - 光纤通道
    - FlexVolume（允许扩展没有内置插件的存储后端
    - 动态资源调配和正在使用的存储类
    - 卷安全
    - 选择器标签卷绑定

- Persistent Volume Access Modes

Access Mode	CLI 缩写	Description
ReadWriteOnce	RWO	卷可以由 **单个节点** 以 **读/写** 方式装入
ReadWriteMany	RWX	卷可以由许 **多节点** 以 **读/写** 方式装入
ReadOnlyMany	ROX	卷可以由许 **多节点** 以 **只读** 方式装入

> 具有相同模式的所有卷都被分组，然后按从最小到最大排序。

- Persistent Volume Storage Classes

> 只有与 pvc 具有相同存储类名的请求类的 pv 才能绑定到 pvc

- Creating PVs and PVC Resources

> pv 和 pvc 之间的交互具有以下生命周期：
>
> - 创建持久卷
> - 定义持久卷声明
> - 使用持久存储

- Using NFS for Persistent Volumes

```
# chown nfsnobody:nfsnobody /exports/folder
# chomd 0700 /exports/folder
# vim /etc/exports
/exports/folder *(rw,all_squash)
```

```
# setsebool -P virt_use_nfs=true
# setsebool -P virt_sandbox_use_nfs=true
```

- Reclamation Policies: Recycling

默认情况下，持久卷设置为保留 `Retain` 。保留回收策略允许手动回收资源。删除持久卷声明后，持久卷仍然存在，并且该卷被视为已释放。管理员可以手动回收卷。

- Using Supplemental Groups for File-Based Volumes

  补充组是常规的 Linux 组。当进程在 Linux 中运行时，它有一个UID、一个 GID 和一个或多个补充组。可以为容器的主进程设置这些属性。补充组标识通常用于控制对共享存储（如 NFS 和GlusterFS）的访问，而 fsGroup 用于控制对块存储（如 Ceph RBD 和iSCSI）的访问。

- Using FS Groups for Block Storage-Based Volumes

  对于文件系统组，fsGroup 定义 pod 的“文件系统组” ID，该 ID 被添加到容器的补充组中。补充组 ID 适用于共享存储，而 fsGroup ID 用于块存储。
  块存储，如 Ceph RBD、iSCSI 和各种类型的云存储，通常专用于单个 pod。块存储通常不共享。

- SELinux and Volume Security

  SELinux标签可以在pod的securityContext中定义。seLinuxOptions部分，并支持 user、role、type 和 level 标签。

  SELinuxContext Options:

  - **MustRunAs**
    如果不使用 peallocated 值，则需要配置 selinuxOptions。使用seLinuxOptions 作为默认值，根据 seLinuxOptions 进行验证。
  - **RunAsAny**
    未提供默认值。允许指定任何seLinuxOptions。

引导式练习: 实施持久数据库存储

[student@workstation]
步骤0. 准备

```
$ lab deploy-volume setup

Setting up master for lab exercise work:

 · Check that master host is reachable.........................  SUCCESS
 · OpenShift master is running.................................  SUCCESS
 · Check that node1 is reachable...............................  SUCCESS
 · Check that node2 is reachable...............................  SUCCESS
 · Check that OpenShift node service is running on node1.......  SUCCESS
 · Check that OpenShift node service is running on node2.......  SUCCESS
 · OpenShift runtime is clean..................................  SUCCESS

```

```
Downloading files for Guided Exercise: Implementing Persistent Database Storage

 · Downloading starter project................................  SUCCESS
 · Downloading solution project...............................  SUCCESS

Download successful.
 · Copying support files to the master VM.....................  SUCCESS
```

步骤1. services 虚拟机上配置 NFS 共享

```
$ ssh root@services
# less DO280/labs/deploy-volume/config-nfs.sh
# /root/DO280/labs/deploy-volume/config-nfs.sh
Export directory /var/export/dbvol created.
# showmount -e
Export list for services.lab.example.com:
/exports/prometheus-alertbuffer  *
/exports/prometheus-alertmanager *
/exports/prometheus              *
/exports/etcd-vol2               *
/exports/logging-es-ops          *
/exports/logging-es              *
/exports/metrics                 *
/exports/registry                *
/var/export/dbvol                *
# exit
logout
Connection to services closed.
```

步骤2. 验证 node1, node2 可访问 services 虚拟机上导出的 NFS

```
$ ssh root@node1
# mount services:/var/export/dbvol /mnt
# mount | grep mnt
services:/var/export/dbvol on /mnt type nfs4
(rw,relatime,vers=4.1,rsize=262144,wsize=262144,namlen=255,hard,proto=tcp,port=0,timeo=600
,retrans=2,sec=sys,clientaddr=172.25.250.11,local_lock=none,addr=172.25.250.13)
# umount /mnt
# exit
logout
Connection to node1 closed.

$ ssh root@node2
# mount services:/var/export/dbvol /mnt
# mount | grep mnt
services:/var/export/dbvol on /mnt type nfs4
(rw,relatime,vers=4.1,rsize=262144,wsize=262144,namlen=255,hard,proto=tcp,port=0,timeo=600
,retrans=2,sec=sys,clientaddr=172.25.250.12,local_lock=none,addr=172.25.250.13)
# umount /mnt
# exit
logout
Connection to node2 closed.
```

步骤3. admin 创建一个持久卷供 MySQL 数据库 Pod 使用

```
$ less -F ~/DO280/labs/deploy-volume/mysqldb-volume.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysqldb-volume
spec:
  capacity:
    storage: 3Gi
  accessModes:
  - ReadWriteMany
  nfs:
    path: /var/export/dbvol
    server: services.lab.example.com
  persistentVolumeReclaimPolicy: Recycle

$ oc create -f ~/DO280/labs/deploy-volume/mysqldb-volume.yml
persistentvolume "mysqldb-volume" created

$ oc get pv
NAME               CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
                        STORAGECLASS   REASON    AGE
etcd-vol2-volume   1G         RWO            Retain           Bound       openshift-
ansible-service-broker/etcd                                  3d
mysqldb-volume     3Gi        RWX            Recycle          Available
                                                        6s
registry-volume    40Gi       RWX            Retain           Bound
default/registry-claim
```

步骤4. developer 创建新项目

```
$ oc login -u developer -p redhat
Login successful.
...输出被忽略...

$ oc new-project persistent-storage
Now using project "persistent-storage" on server "https://master.lab.example.com:443".
...输出被忽略...
```

步骤5. 创建新应用

```
$ oc new-app --name=mysqldb \
  --docker-image=registry.lab.example.com/rhscl/mysql-57-rhel7 \
  -e MYSQL_USER=ose \
  -e MYSQL_PASSWORD=openshift \
  -e MYSQL_DATABASE=quotes
--> Found Docker image 4ae3a3f (2 years old) from registry.lab.example.com for
"registry.lab.example.com/rhscl/mysql-57-rhel7"
...输出被忽略...
```

步骤6. 创建持久卷声明来修改部署配置以使用持久卷

```
$ oc status
In project persistent-storage on server https://master.lab.example.com:443

svc/mysqldb - 172.30.246.204:3306
  dc/mysqldb deploys istag/mysqldb:latest
    deployment #1 deployed 44 seconds ago - 1 pod

2 infos identified, use 'oc status -v' to see details.

$ oc describe pod mysqldb | grep -A 2 Volumes
Volumes:
  mysqldb-volume-1:
    Type:    `EmptyDir` (a temporary directory that shares a pod\'s lifetime)

$ oc set volume dc/mysqldb \
  --add --overwrite --name=mysqldb-volume-1 \
  -t pvc \
  --claim-name=mysqldb-pvclaim \
  --claim-size=3Gi \
  --claim-mode='ReadWriteMany'
persistentvolumeclaims/mysqldb-pvclaim
deploymentconfig "mysqldb" updated

$ oc describe pod mysqldb | grep -E -A 2 'Volumes|ClaimName'
Volumes:
  mysqldb-volume-1:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same
namespace)
    ClaimName:  mysqldb-pvclaim
    ReadOnly:   false
  default-token-rrmtv:
```

步骤7. 验证持久卷声明已绑定持久卷

```
$ oc get pvc
NAME              STATUS    VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
mysqldb-pvclaim   Bound     mysqldb-volume   3Gi        RWX                           1m
```

步骤8. quote.sql 填充数据库，并做端口转发

```
$ oc get pods
NAME              READY     STATUS    RESTARTS   AGE
mysqldb-2-2jbbj   1/1       Running   0          2m

$ oc port-forward mysqldb-2-2jbbj 3306:3306 &
Forwarding from 127.0.0.1:3306 -> 3306

$ mysql -h 127.0.0.1 -u ose -popenshift quotes < ~student/DO280/labs/deploy-
volume/quote.sql

$ mysql -h 127.0.0.1 -u ose -popenshift quotes -e "select count(*) from quote;"
Handling connection for 3306
```

```
+----------+
| count(*) |
+----------+
|        3 |
+----------+

$ ssh root@services ls -lh /var/export/dbvol
...输出被忽略...
drwxr-x---. 2 nfsnobody nfsnobody   54 Feb 21 08:54 quotes
-rw-r--r--. 1 nfsnobody nfsnobody 1.1K Feb 21 08:48 server-cert.pem
-rw-------. 1 nfsnobody nfsnobody 1.7K Feb 21 08:48 server-key.pem
drwxr-x---. 2 nfsnobody nfsnobody 8.0K Feb 21 08:48 sys

$ ssh root@services ls -lh /var/export/dbvol/quotes
total 208K
-rw-r-----. 1 nfsnobody nfsnobody   65 Feb 21 08:48 db.opt
-rw-r-----. 1 nfsnobody nfsnobody 8.4K Feb 21 08:54 quote.frm
-rw-r-----. 1 nfsnobody nfsnobody  96K Feb 21 08:54 quote.ibd
```

步骤9. 清理

```
$ oc delete project persistent-storage
project "persistent-storage" deleted

$ oc login -u admin -p redhat
Login successful.
...输出被忽略...
$ oc delete pv mysqldb-volume
persistentvolume "mysqldb-volume" deleted

$ ssh root@services ls -lh /var/export/dbvol | grep quotes
drwxr-x---. 2 nfsnobody nfsnobody   54 Feb 21 08:54 quotes
$ ssh root@services rm -rf /var/export/dbvol/*

$ lab deploy-volume cleanup

Cleaning up the lab on workstation:

 · Removing lab files from workstation.........................  SUCCESS
 · Removed persistent-storage project..........................  SUCCESS
 · Removing database files.....................................  SUCCESS

$ kill %1
```

配置 OpenShift 内部注册表以实现持久性

- Making the OpenShift Internal Image Registry Persistent

> OpenShift 容器平台内部镜像注册表，是源到映像（S2I）过程的重要组成部分，用于从应用程序源代码创建 pod。S2I 进程的最终输出是一个容器映像，该映像被推送到 OpenShift 内部注册表，然后可用于部署。对于生产设置来说，注册表配置持久存储，是一个更好的建议。

测验: 创建持久注册表

选择以下问题的正确答案：

1. 以下哪个 Ansible 变量，定义了要用持久注册表的存储后端?

   a. openshift_hosted_registry_nfs_backend

   b. openshift_hosted_registry_storage_kind

   c. openshift_integrated_registry_storage_type

2. 以下哪两个对象，是由高级安装程序为持久注册表存储创建的? （选择两个）

   a. An image stream

   b. A persistent volume claim

   c. A storage class

   d. A persistent volume

   e. A deployment configuration

3. 以下哪个 ansible 变量，创建 **访问模式** 为 **RWX** 的持久卷?

   a. openshift_set_hosted_rwx

   b. openshift_integrated_registry_nfs_option

   c. openshift_hosted_registry_storage_access_modes

   d. openshift_hosted_registry_storage_nfs_options

4. 以下哪个命令允许你验证持久注册表的存储后端的正确使用?

   a. oc describe dc/docker-registry | grep -A4 Volumes

   b. oc describe pvc storage-registry | grep nfs

   c. oc describe sc/docker-registry

   d. oc describe pv docker-persistent

实验: 分配持久存储

[student@workstation]
步骤0. 准备

```
$ lab storage-review setup

Checking prerequisites for Lab: Allocating Persistent Storage

 Checking all VMs are running:
  · master VM is up.........................................  SUCCESS
  · node1 VM is up..........................................  SUCCESS
  · node2 VM is up..........................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
  · Check router............................................  SUCCESS
  · Check registry..........................................  SUCCESS

Downloading files for Lab: Allocating Persistent Storage

  · Downloading starter project.............................  SUCCESS
  · Downloading solution project............................  SUCCESS

Download successful.
  · Copy lab files to the services VM.......................  SUCCESS
  · Copy solution files to the services VM..................  SUCCESS

Overall setup status........................................  SUCCESS
```

步骤1. services 虚拟机上使用 config-review-nfs.sh 创建持久卷 NFS 共享

```
$ ssh root@services
Last login: Fri Feb 21 08:59:57 2020 from workstation.lab.example.com
# less -F ~/DO280/labs/storage-review/config-review-nfs.sh
# ~/DO280/labs/storage-review/config-review-nfs.sh
Export directory /var/export/review-dbvol created.
# showmount -e | grep review
/var/export/review-dbvol         *
# exit
logout
Connection to services closed.
```

步骤2. 使用 review-volume-pv.yaml 创建持久存储

```
$ oc login -u admin -p redhat
Login successful.
...输出被忽略...

$ less -F ~/DO280/labs/storage-review/review-volume-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: review-pv
```

```
spec:
  capacity:
    storage: 3Gi
  accessModes:
  - ReadWriteMany
  nfs:
    path: /var/export/review-dbvol
    server: services.lab.example.com
  persistentVolumeReclaimPolicy: Recycle

$ oc create -f ~/DO280/labs/storage-review/review-volume-pv.yaml
persistentvolume "review-pv" created
```

步骤3. 导入 instructor-template.yaml 模板

```
$ less -F ~/DO280/labs/storage-review/instructor-template.yaml
apiVersion: v1
kind: Template
labels:
  template: instructor
...输出被忽略...
        from:
          kind: ImageStreamTag
          name: php:7.0
...输出被忽略...
        from:
          kind: ImageStreamTag
          name: mysql:5.7
...输出被忽略...

$ oc create -f ~/DO280/labs/storage-review/instructor-template.yaml -n openshift
template "instructor" created
```

步骤4. 创建新项目 instructor

```
$ oc login -u developer -p redhat
Login successful.
...输出被忽略...

$ oc new-project instructor
Now using project "instructor" on server "https://master.lab.example.com:443".
...输出被忽略...
```

步骤5. https://master.lab.example.com 选择模板，添加字段。创建应用

Username: **developer**
Password: **redhat**

单击项目 **instructor**，单击 **Browse Catalog**

单击 **Languages**，选择 **PHP**，选择 **The Instructor Application Template**

1 information，单击 **Next>** 命令按钮

2 Configuration，

  * ...

  Application Hostname **instructor.apps.lab.example.com**

  单击 **Next>** 命令按钮

3 Binding，

  ◉ **Create a secret in instructor to be used later**

  单击 **create** 命令按钮

4 Results，

  The Instructor Application Template has been added to instructor successfully.
  The binding instructor-nn4nw-gbvsm has been created successfully.

  单击 **Continue to the project overview.** 链接

步骤6. 端口转发，添加数据库

```
$ oc get pods
NAME                   READY     STATUS      RESTARTS   AGE
instructor-1-7pcmq     1/1       Running     0          40m
instructor-1-build     0/1       Completed   0          40m
mysql-1-98rfp          1/1       Running     0          40m

$ oc port-forward mysql-1-98rfp 3306:3306 &
Forwarding from 127.0.0.1:3306 -> 3306

$ mysql -h 127.0.0.1 -u instructor -ppassword instructor < ~student/DO280/labs/storage-
review/instructor.sql

$ mysql -h 127.0.0.1 -u instructor -ppassword instructor -e "select * from instructors;"
Handling connection for 3306
    +-----------------+--------------------------------+----------------+
... | instructorName  | email                          | city           |...
    +-----------------+--------------------------------+----------------+
... | DemoUser1       | duser1@workstation.example.com | Raleigh        |...
    | InstructorUser1 | iuser1@workstation.example.com | Rio de Janeiro |
    | InstructorUser2 | iuser2@workstation.example.com | Raleigh        |
    | InstructorUser3 | iuser3@workstation.example.com | Sao Paulo      |
    +-----------------+--------------------------------+----------------+

$  kill %1
```

步骤7. [workstation] **firefox** http://instructor.apps.lab.example.com ， 填加新记录

单击 **Add new instructor** 按钮

Name **InstructorUser4**

Email address **iuser4@workstation.example.com**

City **Raleigh**

Country **United States**

单击 **Add new Instructor**

步骤8. 评估

```
$ lab storage-review grade

Grading the student's work for Lab: Allocating Persistent Storage

 · Check if the mysql pod is in Running state..................  PASS
 · Check if the instructor pod is in Running state.............  PASS
 . Checking if REST interface can be invoked successfully......  PASS
 . Checking if the instructor template was imported correctly..
PASS
 . Checking if the instructor route can be invoked successfully  PASS

Overall exercise grade.........................................  PASS
```

步骤9. 清理

```
$ oc login -u admin -p redhat
Login successful.
...输出被忽略...

$ oc delete project instructor
project "instructor" deleted

$ oc delete pv review-pv
persistentvolume "review-pv" deleted

$ oc delete template instructor -n openshift
template "instructor" deleted

$ ssh root@services rm -rf /var/export/review-dbvol /etc/exports.d/review-dbvol.exports
```

总结

- 红帽 OpenShift 容器平台使用 PersistentVolumes（PV）为 Pod 提供持久存储。
- OpenShift 项目使用 PersistentVolumeClaim（PVC）资源来请求分配至项目的 PV。
- OpenShift 安装程序配置并启动默认注册表，它使用从 OpenShift 主控机导出的 NFS 共享。

- 一组 Ansible 变量允许为 OpenShift 默认注册表配置外部 NFS 存储。这将创建一个持久卷和一个持久卷声明。

7. 管理应用部署

应用缩放

- Replication Controllers

  > 确保时刻运行指定数量的 Pod 副本
  >
  > 复制控制器的定义包括：
  >
  > - 需要的副本数
  > - 用于创建复制的 Pod 的 定义
  > - 用于识别受管 Pod 的选择器

- Creating Replication Controllers from a Deployment Configuration
- Changing the Number of Replicas for an Application

```
$ oc get dc
$ oc scale --replicas=5 dc myapp
```

- Autoscaling Pods

  > 存在 **指标子系统**，主要指 Heapster 组件

```
$ oc autoscale dc/myapp --min 1 --max 10 --cpu-percent=80

$ oc get hpa/frontend
$ oc describe hpa/frontend
```

引导式练习: 缩放应用

[student@workstation]
步骤1. 创建一个项目

```
$ oc login -u developer -p redhat
Login successful.
...输出被忽略...

$ oc new-project scaling
Now using project "scaling" on server "https://master.lab.example.com:443".
...输出被忽略...
```

步骤2. 创建应用来测试缩放

```
$ oc new-app -i php:7.0 http://registry.lab.example.com/scaling -o yaml > ~/scaling.yml
$ vim ~/scaling.yml
...
- apiVersion: v1
  kind: DeploymentConfig
  metadata:
    annotations:
      openshift.io/generated-by: OpenShiftNewApp
    creationTimestamp: null
    labels:
      app: scaling
    name: scaling
  spec:
    replicas: 3
    selector:
...

$ oc create -f ~/scaling.yml
imagestream "scaling" created
buildconfig "scaling" created
deploymentconfig "scaling" created
service "scaling" created

$ watch -n 3 oc get builds
Every 3.0s: oc get builds                          Fri Feb 21 12:37:34 2020

NAME        TYPE      FROM          STATUS     STARTED              DURATION
scaling-1   Source    Git@0bdae71   "Complete"   About a minute ago   1m38s
<Ctrl-C>

$ oc get pods
NAME                 READY     STATUS      RESTARTS   AGE
"scaling-1-5t89c"    1/1       Running     0          1m
scaling-1-build      0/1       Completed   0          3m
"scaling-1-vdsrk"    1/1       Running     0          1m
"scaling-1-whwwb"    1/1       Running     0          1m
```

步骤3. 为应用创建路由，以均衡各个 Pod 的请求

```
$ oc expose service scaling --hostname=scaling.apps.lab.example.com
route "scaling" exposed
```

步骤4. Web 控制台检索 Pod 的 IP 地址。与 scaling 应用报告的 IP 地址比较

firefox https://master.lab.example.com
Username **developer**
Password **redhat**

"My Projects" / **scaling**

"OverView" / 单击 **>** 按钮

"Application" / "Pods"

```
$ oc get pod -o wide
NAME              READY     STATUS      RESTARTS   AGE       IP             NODE
scaling-1-5t89c   1/1       Running     0          4m        10.128.0.131   
node2.lab.example.com
scaling-1-build   0/1       Completed   0          6m        10.129.0.222   
node1.lab.example.com
scaling-1-vdsrk   1/1       Running     0          4m        10.129.0.223   
node1.lab.example.com
scaling-1-whwwb   1/1       Running     0          4m        10.128.0.130   
node2.lab.example.com
```

步骤5. 确保路由器正在平衡对该应用的请求

```
$ for i in {1..5}; do
  curl -s http://scaling.apps.lab.example.com | grep IP
  done
 <br/> Server IP: 10.128.0.130
 <br/> Server IP: 10.128.0.131
 <br/> Server IP: 10.129.0.223
 <br/> Server IP: 10.128.0.130
 <br/> Server IP: 10.128.0.131
```

步骤6. 缩放应用来运行更多 Pod

```
$ oc describe dc scaling | grep Replicas
Replicas: 3
  Replicas: 3 current / 3 desired

$ oc scale --replicas=5 dc scaling
deploymentconfig "scaling" scaled

$ oc get pods -o wide
NAME              READY     STATUS      RESTARTS   AGE       IP             NODE
scaling-1-5t89c   1/1       Running     0          22m       10.128.0.131   
node2.lab.example.com
scaling-1-build   0/1       Completed   0          24m       10.129.0.222   
node1.lab.example.com
scaling-1-f5zdz   1/1       Running     0          54s       10.129.0.224   
node1.lab.example.com
```

```
scaling-1-sln9k   1/1       Running     0          54s       10.128.0.132
node2.lab.example.com
scaling-1-vdsrk   1/1       Running     0          22m       10.129.0.223
node1.lab.example.com
scaling-1-whwwb   1/1       Running     0          22m       10.128.0.130
node2.lab.example.com

$ for i in {1..5}; do
  curl -s http://scaling.apps.lab.example.com | grep IP
  done
 <br/> Server IP: 10.128.0.130
 <br/> Server IP: 10.128.0.131
 <br/> Server IP: 10.128.0.132
 <br/> Server IP: 10.129.0.223
 <br/> Server IP: 10.129.0.224
```

步骤7. 清理

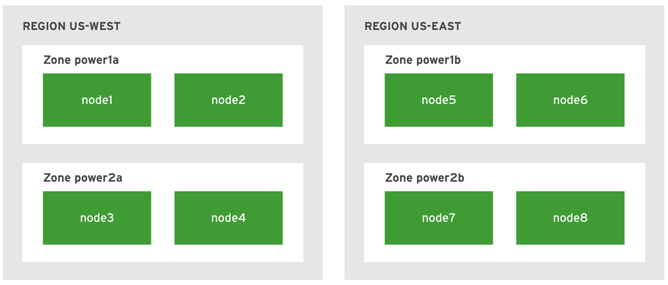
```
$ oc delete project scaling
project "scaling" deleted
```

控制 Pod 调度

- Introduction to the OpenShift Scheduler Algorithm

> 遵循一个包含三个步骤的流程：
>
> 1. 过滤节点
> 2. 排列过滤后节点列表的优先顺序
> 3. 选择最合适的节点

- Scheduling and Topology



```
$ oc label node1.lab.example.com region=us-west zone=power1a --overwrite
$ oc label node2.lab.example.com region=us-west zone=power1a --overwrite
$ oc label node3.lab.example.com region=us-west zone=power2a --overwrite
$ oc label node4.lab.example.com region=us-west zone=power2a --overwrite
$ oc label node5.lab.example.com region=us-east zone=power1b --overwrite
$ oc label node6.lab.example.com region=us-east zone=power1b --overwrite
$ oc label node7.lab.example.com region=us-east zone=power2b --overwrite
$ oc label node8.lab.example.com region=us-east zone=power2b --overwrite

$ oc get node node1.lab.example.com --show-labels
$ oc get node node1.lab.example.com -L region
$ oc get node nod1.lab.example.com -L region -L zone
```

- Unschedulable Nodes

```
新 pod 不用
$ oc adm manage-node --schedulable=false node2.lab.example.com

已存在 pod ， 排干
$ oc adm drain node2.lab.example.com
```

- Controlling Pod Placement

```
亲和性
$ oc patch dc myapp --patch '{"spec":{"template":{"nodeSelector":{"env": "qa"}}}}'
```

- Managing the default Project

```
$ oc annotate --overwrite namespace default openshift.io/node-selector='region=infra'
```

引导式练习: 控制 Pod 调度

[student@workstation]
步骤0. 准备

```
$ lab schedule-control setup

Checking prerequisites for GE: Controlling Pod Scheduling

 Checking all VMs are running:
 · master VM is up.............................................  SUCCESS
 · node1 VM is up..............................................  SUCCESS
 · node2 VM is up..............................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
 · Check router................................................  SUCCESS
 · Check registry..............................................  SUCCESS

Overall setup status...........................................  SUCCESS
```

步骤1. 检查 node1 和 node2 主机的标签。同一地区，同一应用的 Pod 被调试到这些节点上

```
$ oc login -u admin -p redhat
Login successful.
...输出被忽略...

$ oc get nodes -L region
NAME                     STATUS    ROLES     AGE       VERSION             REGION
master.lab.example.com   Ready     master    4d        v1.9.1+a0ce1bc657
node1.lab.example.com    Ready     compute   4d        v1.9.1+a0ce1bc657   infra
node2.lab.example.com    Ready     compute   4d        v1.9.1+a0ce1bc657   infra

$ oc new-project schedule-control
Now using project "schedule-control" on server "https://master.lab.example.com:443".
...输出被忽略...

$ oc new-app --name=hello --docker-image=registry.lab.example.com/openshift/hello-
openshift
--> Found Docker image 7af3297 (22 months old) from registry.lab.example.com for
"registry.lab.example.com/openshift/hello-openshift"
...输出被忽略

$ oc scale dc/hello --replicas=5
deploymentconfig "hello" scaled

$ oc get pod -o wide
NAME            READY     STATUS    RESTARTS   AGE       IP             NODE
hello-1-2zz95   1/1       Running   0          9s        10.128.0.135
node2.lab.example.com
hello-1-479vt   1/1       Running   0          1m        10.129.0.225
node1.lab.example.com
hello-1-hn8kd   1/1       Running   0          9s        10.129.0.226
node1.lab.example.com
hello-1-rrp44   1/1       Running   0          9s        10.128.0.136
node2.lab.example.com
hello-1-shl4g   1/1       Running   0          9s        10.128.0.134
node2.lab.example.com
```

步骤2. 将 node2 上的 region 标签更改为 apps

```
$ oc label node node2.lab.example.com region=apps --overwrite=true
node "node2.lab.example.com" labeled

$ oc get nodes -L region
NAME                     STATUS    ROLES     AGE       VERSION             REGION
master.lab.example.com   Ready     master    4d        v1.9.1+a0ce1bc657
node1.lab.example.com    Ready     compute   4d        v1.9.1+a0ce1bc657   infra
node2.lab.example.com    Ready     compute   4d        v1.9.1+a0ce1bc657   apps
```

步骤3. 配置部署配置，以请求 Pod 仅调度到 apps 地区中的节点上运行

```
$ oc get dc/hello -o yaml > dc.yml

$ vim dc.yml
```

```
...
    spec:
      nodeSelector:
        region: apps
      containers:
...

$ oc apply -f dc.yml
Warning: oc apply should be used on resource created by either oc create --save-config or
oc apply
deploymentconfig "hello" configured

$ oc get pod -o wide
NAME            READY     STATUS    RESTARTS   AGE       IP             NODE
hello-2-2vz5n   1/1       Running   0          2m        10.128.0.139
"node2.lab.example.com"
hello-2-5kcrh   1/1       Running   0          1m        10.128.0.142
"node2.lab.example.com"
hello-2-lc4dt   1/1       Running   0          2m        10.128.0.138
"node2.lab.example.com"
hello-2-n5r5j   1/1       Running   0          2m        10.128.0.140
"node2.lab.example.com"
hello-2-x9bfq   1/1       Running   0          2m        10.128.0.141
"node2.lab.example.com"
```

步骤4. 添加 node1 到 apps 地区

```
$ oc label node node1.lab.example.com region=apps --overwrite=true
node "node1.lab.example.com" labeled

$ oc get nodes -L region
NAME                     STATUS    ROLES     AGE       VERSION             REGION
master.lab.example.com   Ready     master    4d        v1.9.1+a0ce1bc657
node1.lab.example.com    Ready     compute   4d        v1.9.1+a0ce1bc657   apps
node2.lab.example.com    Ready     compute   4d        v1.9.1+a0ce1bc657   apps
```

步骤5. 静止 node2 主机

```
$ oc adm manage-node --schedulable=false node2.lab.example.com
NAME                     STATUS                        ROLES     AGE       VERSION
node2.lab.example.com    Ready,SchedulingDisabled   compute   4d        v1.9.1+a0ce1bc657

$ oc adm drain node2.lab.example.com --delete-local-data
node "node2.lab.example.com" already cordoned
pod "hello-2-n5r5j" evicted
pod "router-1-bqpkc" evicted
pod "hello-2-x9bfq" evicted
pod "hello-2-lc4dt" evicted
pod "docker-registry-1-hg6ck" evicted
pod "hello-2-2vz5n" evicted
pod "hello-1-2btg6" evicted
pod "hello-2-5kcrh" evicted
```

```
node "node2.lab.example.com" drained

$ oc get pods -o wide
NAME            READY     STATUS    RESTARTS   AGE       IP             NODE
hello-2-8f5gf   1/1       Running   0          48s       10.129.0.231
"node1.lab.example.com"
hello-2-bfkwc   1/1       Running   0          49s       10.129.0.230
"node1.lab.example.com"
hello-2-ggwv7   1/1       Running   0          49s       10.129.0.229
"node1.lab.example.com"
hello-2-r92td   1/1       Running   0          48s       10.129.0.228
"node1.lab.example.com"
hello-2-zqmq2   1/1       Running   0          48s       10.129.0.232
"node1.lab.example.com"
```

步骤6. 清理

```
$ oc adm manage-node --schedulable=true node2.lab.example.com
NAME                    STATUS    ROLES     AGE       VERSION
node2.lab.example.com   Ready     compute   4d        v1.9.1+a0ce1bc657

$ oc label node node1.lab.example.com region=infra --overwrite=true
node "node1.lab.example.com" labeled
$ oc label node node2.lab.example.com region=infra --overwrite=true
node "node2.lab.example.com" labeled

$ oc get nodes -L region
NAME                     STATUS    ROLES     AGE       VERSION             REGION
master.lab.example.com   Ready     master    4d        v1.9.1+a0ce1bc657
node1.lab.example.com    Ready     compute   4d        v1.9.1+a0ce1bc657   "infra"
node2.lab.example.com    Ready     compute   4d        v1.9.1+a0ce1bc657   "infra"

$ oc delete project schedule-control
project "schedule-control" deleted
```

管理镜像、镜像流和模板

- Introduction to Images

  可部署的运行时模板，其中包含运行单一容器的所有要求，还包含描述镜像需求和功用的元数据。

  Docker 不使用版本号，使用 **标签** 来管理镜像

  ```
  # docker images |  grep gog
  "registry.lab.example.com/openshiftdemos/gogs   0.9.97"   d44302ef5b2f   2 years ago
  449 MB
  "openshiftdemos/gogs                            latest"   d44302ef5b2f   2 years ago
  449 MB
  ```

- Image Streams

由任意数量的容器镜像组成，它们通过 **标签** 来标识

```
# oc get image | grep php
"sha256:23765e00df8d0a934ce4f2e22802bc0211a6d450bfbb69144b18cb0b51008cdd"
registry.lab.example.com/rhscl/php-70-
rhel7@sha256:23765e00df8d0a934ce4f2e22802bc0211a6d450bfbb69144b18cb0b51008cdd
"sha256:920c2cf85b5da5d0701898f0ec9ee567473fa4b9af6f3ac5b2b3f863796bbd68"
registry.lab.example.com/rhscl/php-56-
rhel7@sha256:920c2cf85b5da5d0701898f0ec9ee567473fa4b9af6f3ac5b2b3f863796bbd68
```

- Tagging Images

```
$ oc tag SOURCE DESTINATION
```

- Recommended Tagging Conventions

描述	示例
修订	myimage:v2.0.1
架构	myimage:v2.0-x86_64
基础镜像	myimage:v1.2-rhel7
最新镜像	myimage:latest
最新的稳定镜像	myimage:stable

- Introduction to Templates

描述带有参数的对象集合，经处理后生成一系列的对象

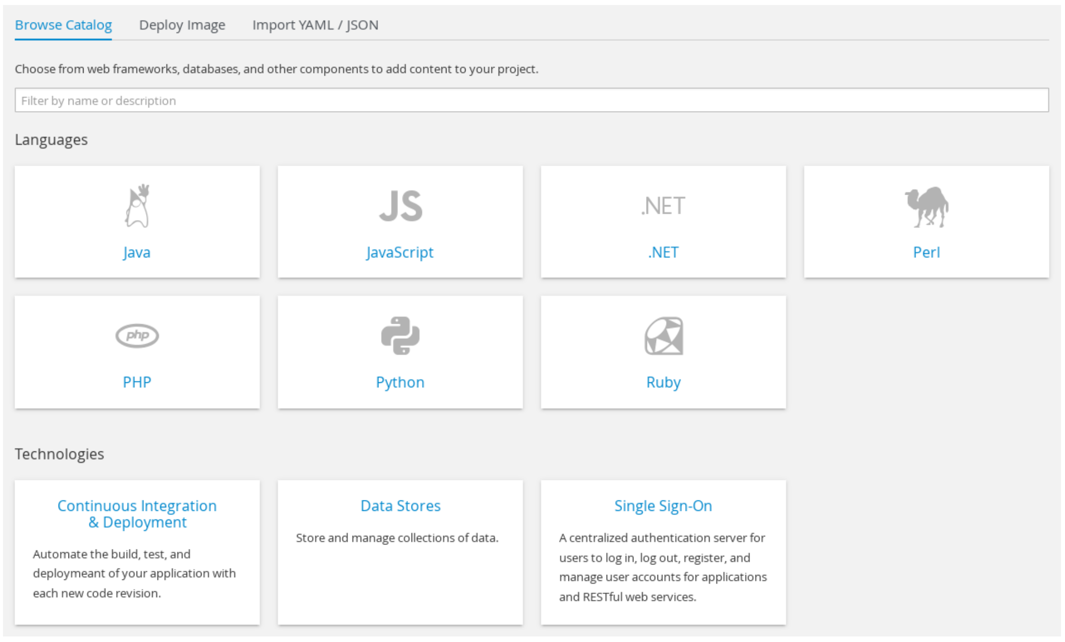
  - Managing Templates

    ```
    $ oc create -f FILENAME
    ```

- Instant App and QuickStart Templates

OpenShift 容器平台提供了多个默认的即时应用程序和 QickStart 模板，让开发人员能够 **快速** 创建不同语言的新应用

```
$ oc get templates -n openshift
```


Browse Catalog Deploy Image Import YAML / JSON

Choose from web frameworks, databases, and other components to add content to your project.

Filter by name or description

Languages

Java JavaScript .NET Perl PHP Python Ruby

Technologies

Continuous Integration & Deployment
Automate the build, test, and deploymeant of your application with each new code revision.

Data Stores
Store and manage collections of data.

Single Sign-On
A centralized authentication server for users to log in, log out, register, and manage user accounts for applications and RESTful web services.

引导式练习: 管理镜像流

[student@workstation]

步骤0. 准备

```
$ lab schedule-is setup

Checking prerequisites for GE: Managing Image Streams

 Checking all VMs are running:
 ? master VM is up.....................................  SUCCESS
 ? node1 VM is up......................................  SUCCESS
 ? node2 VM is up......................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
 ? Check router........................................  SUCCESS
 ? Check registry......................................  SUCCESS

Downloading files for GE: Managing Image Streams

 ? Download exercise files.............................  SUCCESS

Overall setup status...................................  SUCCESS
```

步骤1. 新项目中布署应用

```
$ oc login -u developer -p redhat
Login successful.
...输出被忽略...

$ oc new-project schedule-is
Now using project "schedule-is" on server "https://master.lab.example.com:443".
...输出被忽略...

$ oc new-app --name=phpmyadmin --docker-image=registry.lab.example.com/phpmyadmin/phpmyadmin:4.7
--> Found Docker image f51fd61 (23 months old) from registry.lab.example.com for "registry.lab.example.com/phpmyadmin/phpmyadmin:4.7"
...输出被忽略...
```

步骤2. 创建服务帐号

```
$ oc login -u admin
Logged into "https://master.lab.example.com:443" as "admin" using existing credentials.
...输出被忽略...
Using project "schedule-is".

$ oc create serviceaccount phpmyadmin-account
serviceaccount "phpmyadmin-account" created

$ oc adm policy add-scc-to-user anyuid -z phpmyadmin-account
scc "anyuid" added to: ["system:serviceaccount:schedule-is:phpmyadmin-account"]
```

步骤3. 使用新创建的服务帐号更新 dc

```
$ oc login -u developer
Logged into "https://master.lab.example.com:443" as "developer" using existing credentials.
...输出被忽略...

$ cat ~/DO280/labs/secure-review/patch-dc.sh
$ oc patch dc/phpmyadmin --patch '{"spec":{"template":{"spec":{"serviceAccountName": "phpmyadmin-account"}}}}'
deploymentconfig "phpmyadmin" patched

$ oc get pods
NAME                 READY     STATUS    RESTARTS   AGE
phpmyadmin-2-52b5p   1/1       Running   0          1m
```

步骤4. 在内部镜像注册表更新镜像

```
$ cd ~/DO280/labs/schedule-is/
[student@workstation schedule-is]$ docker load -i phpmyadmin-latest.tar
cd7100a72410: Loading layer 4.403 MB/4.403 MB
f06b58790eeb: Loading layer 2.873 MB/2.873 MB
730b09e0430c: Loading layer 11.78 kB/11.78 kB
931398d7728c: Loading layer 3.584 kB/3.584 kB
...输出被忽略...
```

```
Loaded image ID: sha256:93d0d7db5ce2...输出被忽略...

$ docker images
...输出被忽略...
<none>                                          <none>            93d0d7db5ce2        20 months ago       166 MB

$ docker tag 93d0d7db5ce2 docker-registry-default.apps.lab.example.com/schedule-is/phpmyadmin:4.7

$ docker images
...输出被忽略...
docker-registry-default.apps.lab.example.com/schedule-is/phpmyadmin   4.7     93d0d7db5ce2        20 months ago       166 MB
registry.lab.example.com/rhscl/nodejs-6-rhel7                         latest  fba56b5381b7        2 years ago         489 MB

$ TOKEN=$(oc whoami -t)
$ echo $TOKEN
$ docker login -u developer -p ${TOKEN} docker-registry-default.apps.lab.example.com
Error response from daemon: Get https://docker-registry-default.apps.lab.example.com/v1/users/: x509: certificate signed by unknown authority

$ cat ~/DO280/labs/schedule-is/trust_internal_registry.sh
$ ~/DO280/labs/schedule-is/trust_internal_registry.sh
Fetching the OpenShift internal registry certificate.
done.

Copying certificate to the correct directory.
done.

System trust updated.

Restarting docker.
done.

$ docker login -u developer -p ${TOKEN} docker-registry-default.apps.lab.example.com
Login Succeeded

$ docker push docker-registry-default.apps.lab.example.com/schedule-is/phpmyadmin:4.7
The push refers to a repository [docker-registry-default.apps.lab.example.com/schedule-is/phpmyadmin]
...输出被忽略...
4.7: digest: sha256:b003fa5555dcb0a305d26aec3935b3a1127179ea8ad9d57685df4e4eab912ca8 size: 3874
```

步骤5. 验证新镜像触发了新的布署进程

```
$ oc get pods
NAME                    READY     STATUS        RESTARTS    AGE
phpmyadmin-2-52b5p      0/1       Terminating   1           1h
phpmyadmin-3-zlmrd      1/1       Running       0           31s
```

步骤6. 清理

```
$ oc delete project schedule-is
project "schedule-is" deleted
```

实验: 管理应用部署

[student@workstation]
步骤0. 准备

```
$ lab manage-review setup

Checking prerequisites for Lab: Managing Application Deployments

 Checking all VMs are running:
 · master VM is up..............................................  SUCCESS
 · node1 VM is up...............................................  SUCCESS
 · node2 VM is up...............................................  SUCCESS
 Checking all OpenShift default pods are ready and running:
 · Check router.................................................  SUCCESS
 · Check registry...............................................  SUCCESS

Overall setup status............................................  SUCCESS
```

步骤1. 更新节点上的标签

```
$ oc login -u admin -p redhat
Login successful.
...输出被忽略...

$ oc get nodes -L region
NAME                    STATUS   ROLES     ...  REGION
master.lab.example.com  Ready    master    ...
node1.lab.example.com   Ready    compute   ...  infra
node2.lab.example.com   Ready    compute   ...  infra

$ oc label node node1.lab.example.com region=services --overwrite
node "node1.lab.example.com" labeled

$ oc label node node2.lab.example.com region=applications --overwrite
node "node2.lab.example.com" labeled

$ oc get nodes -L region
NAME                    STATUS   ROLES     ...  REGION
master.lab.example.com  Ready    master    ...
node1.lab.example.com   Ready    compute   ...  services
node2.lab.example.com   Ready    compute   ...  applications
```

步骤2. 创建新项目

```
$ oc new-project manage-review
Now using project "manage-review" on server "https://master.lab.example.com:443".
...输出被忽略...
```

步骤3. 部署三个新应用

```
$ oc new-app -i php:7.0 http://registry.lab.example.com/version
--> Found image c101534 (2 years old) in image stream "openshift/php" under tag "7.0" for
"php:7.0"
...输出被忽略...

$ oc scale dc/version --replicas=3
deploymentconfig "version" scaled

$ oc get pod -o wide
NAME                READY     STATUS       ...   NODE
version-1-9j6kf     1/1       Running      ...   node2.lab.example.com
version-1-build     0/1       Completed    ...   node2.lab.example.com
version-1-kptz6     1/1       Running      ...   node2.lab.example.com
version-1-lz78q     1/1       Running      ...   node2.lab.example.com
```

步骤4. 更改部署配置

```
$ oc export dc/version -o yaml > version-dc.yml

$ vim version-dc.yml
...输出被忽略...
  template:
...输出被忽略...
    spec:
      nodeSelector:
        region: applications
      containers:
...输出被忽略...

$ oc replace -f version-dc.yml
deploymentconfig "version" replaced
```

步骤5. 验证一个新的部署已经开始

```
$ oc get pod -o wide
NAME                READY     STATUS       ...   NODE
version-1-build     0/1       Completed    ...   node2.lab.example.com
version-2-2vcxl     1/1       Running      ...   node2.lab.example.com
version-2-drhdh     1/1       Running      ...   node2.lab.example.com
version-2-z2jnn     1/1       Running      ...   node2.lab.example.com
```

步骤6. 更改节点上的标签

```
$ oc label node node1.lab.example.com region=applications --overwrite node
"node1.lab.example.com" labeled

$ oc get nodes -L region
NAME                     STATUS    ROLES     ...   REGION
master.lab.example.com   Ready     master    ...
node1.lab.example.com    Ready     compute   ...   applications
node2.lab.example.com    Ready     compute   ...   applications
```

步骤7. node2 节点设为不可调试并排空该节点

```
$ oc adm manage-node node2.lab.example.com --schedulable=false
NAME                     STATUS                     ROLES     AGE    ...
node2.lab.example.com    Ready,SchedulingDisabled   compute   5d     ...

$ oc adm drain node2.lab.example.com --delete-local-data
node "node2.lab.example.com" already cordoned
...输出被忽略...
pod "version-2-drhdh" evicted
...输出被忽略...
node "node2.lab.example.com" drained

$ oc get pods -o wide
NAME                READY     STATUS    ...   NODE
version-2-6pczp     1/1       Running   ...   node1.lab.example.com
version-2-ccjxz     1/1       Running   ...   node1.lab.example.com
version-2-xskfl     1/1       Running   ...   node1.lab.example.com
```

步骤8. 创建一个路由

```
$ oc expose service version --hostname=version.apps.lab.example.com
route "version" exposed
```

步骤9. curl 测试应用

```
$ curl http://version.apps.lab.example.com
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <p>Version v1</p>
 </body>
</html>
```

步骤10. 评分

```
$ lab manage-review grade

Grading the student's work for Lab: Managing Application Deployments

Grading the lab.

Checking the manage-review project..............................  PASS
Check the labels from the node 1................................  PASS
Check the labels from the node 2................................  PASS
Checking the pod scale..........................................  PASS

Overall exercise grade..........................................  PASS
```

步骤11. 清理

```
$ oc adm manage-node node2.lab.example.com --schedulable
NAME                    STATUS    ROLES     AGE       ...
node2.lab.example.com   Ready     compute   5d        ...

$ oc label --overwrite node node1.lab.example.com region=infra
node "node1.lab.example.com" labeled
$ oc label --overwrite node node2.lab.example.com region=infra
node "node2.lab.example.com" labeled

$ oc get nodes -L region
NAME                     STATUS    ROLES     ...   REGION
master.lab.example.com   Ready     master    ...
node1.lab.example.com    Ready     compute   ...   infra
node2.lab.example.com    Ready     compute   ...   infra

$ oc delete project manage-review
project "manage-review" deleted
```

总结

- 复本控制器确保时刻运行指定数量的 Pod 副本。
- OpenShift HorizontalPodAutoscaler 根据当前的负载执行自动缩放。
- 调度程序决定新 Pod 在 OpenShift 集群中节点上的位置。若要限制可运行 Pod 的节点集，集群管理员可以标记节点，开发人员则可以定义节点选择器。
- 触发器根据 OpenShift 内部和外部事件来触发新部署创建。镜像流提供相关镜像的单一虚拟视图，类似于 Docker 镜像存储库。
- 镜像流由任意数量的容器镜像组成，它们通过标签来标识。镜像流提供相关镜像的单一虚拟视图，类似于 Docker 镜像存储库。

8. 安装和配置指标子系统

说明指标子系统的架构

- Metrics Subsystem Components

实现 OpenShift 集群性能指标的采集和长期存储。可以为节点以及各个节点上运行的所有容器收集指标

基于下列开源项目部署为一组容器：

- **Heapster**

  从集群内所有节点收集指标，并将它们转发到存储引擎

- **Hawkular Metrics**

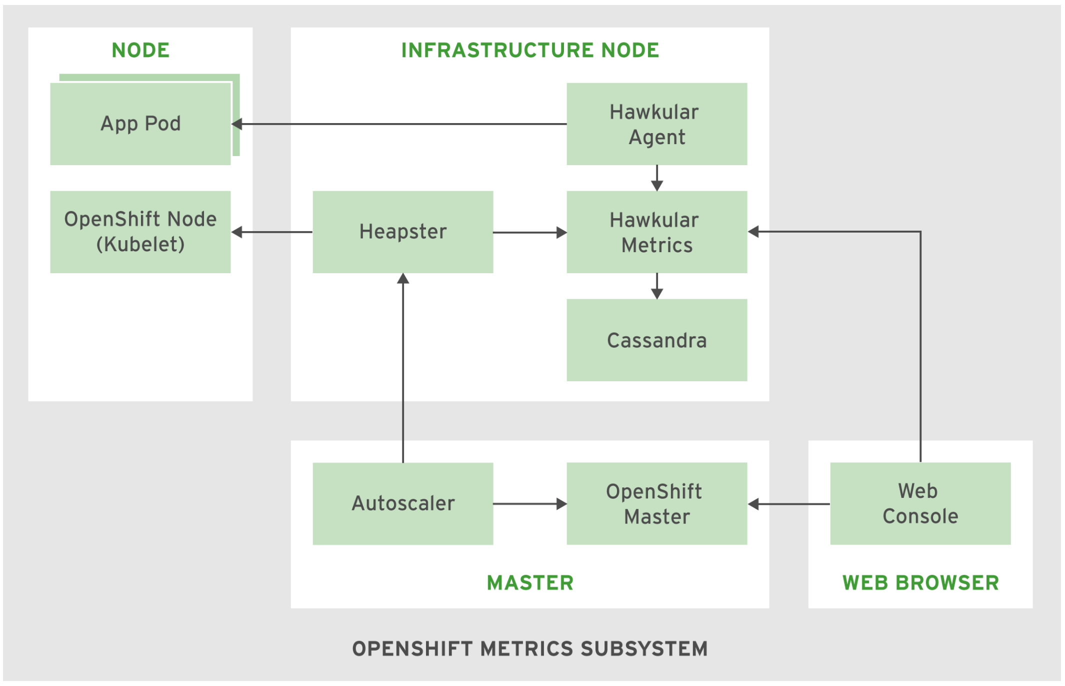
  提供存储和查询时间序列数据的 REST API

- **Hawkular Agent**

  从各个应用收集自定性能指标，并将它们转发到 Hawkular Metrics 进行存储

- **Cassandra**

  将时间序列数据存储在非关系分布式数据库中



- Accessing Heapster and Hawkular
- Sizing the Metrics Subsystem

系统管理员可以使用 `oc` 命令配置 Heapster 和 Hawkular 部署

必须使用 Metrics 安装 playbook 来缩放和配置 Cassandra 部署

- Providing Persistent Storage for Cassandra

Cassandra 可以利用单一持久卷部署为单一 Pod。

至少需要三个 Cassandra Pod，才能实现指标子系统的高可用性（HA）

每一 Pod 需要一个独占的卷

测验: 指标子系统的架构

选择以下问题的正确答案：

1. OpenShift 指标子系统的哪一组件从集群节点及其运行的容器收集性能指标?

   a. Heapster

   b. Hawkular Agent

   c. Hawkular Metrics

   d. cassandra

2. OpenShift 指标子系统的哪一组件使用持久卷来长期存储指标?

   a. Heapster

   b. Hawkular Agent

   c. Hawkular Metrics

   d. cassandra

3. OpenShift 指标子系统的哪一组件提供 REST API，供 Web 控制台用于显示项目内 Pod 的性能图形?

   a. Heapster

   b. Hawkular Agent

   c. Hawkular Metrics

   d. cassandra

4. 以下哪两项 OpenShift 功能可用于获取节点的当前 CPU 使用量信息? （请选择两项）

   a. 通过 -o 选项向 oc get node 命令输出添加额外的列

   b. 使用 Master API 代理调用 Heapster API

   c. 过滤 oc describe node 输出以获取 Allocated resources: 表

   d. 打开 Web 控制台的 Cluster Admin 菜单子系统ter Admin 菜单

   e. 使用 oc adm top 命令调用 Heapster API

5. 在调整 OpenShift 指标子系统所用持久卷的大小时，需要考虑以下哪四个因素? （请选择四项）

   a. 指标的保留时间（持续时间）

   b. 指标收集的频率（解析）

   c. 集群中的节点数

   d. 集群中 Pod 预期总数量

   e. Hawkular Pod 副本的数量

f. 集群中的主控机节点的数量

6. 在更改 OpenShift 指标子系统配置时，如各个 Pod 的副本数或指标的存储时长，推荐的做法是哪一项?

   a. 更改各个指标子系统部署配置中的环境变量

   b. 为指标子系统组件创建自定义容器镜像

   c. 使用 Ansible 变量的新值运行 Metrics 安装 playbook

   d. 在部署配置中覆盖各个指标子系统 Pod 的配置卷

安装指标子系统

- Deploying the Metrics Subsystem

```
$ ansible-playbook \
  -i OPENSHIFT_ANSIBLE_INVENTORY \
  OPENSHIFT_ANSIBLE_DIR/openshift-metrics.yml \
  -e openshift_metrics_install_metrics=True
```

- Uninstalling the Metrics Subsystem

```
$ ansible-playbook \
  -i OPENSHIFT_ANSIBLE_INVENTORY \
  OPENSHIFT_ANSIBLE_DIR/openshift-metrics.yml \
  -e openshift_metrics_install_metrics=False
```

- Verifying the Deployment of the Metrics Subsystem

```
$ oc get pod -n openshift-infra
```

- Post-Installation Steps

`firefox` https://hawkular-metrics.apps.lab.example.com

- Ansible Variables for the metrics Subsystem

安装：

openshift_metrics_install_metrics=True

用于提取指标子系统容器镜像的注册表：

`openshift_metrics_image_prefix=registry.lab.example.com/openshift3/ose-`

openshift_metrics_image_version= **`v`** 3.9

各个组件的 Pod 的资源请求和限值：

openshift_metrics_heapster_requests_memory=300M
openshift_metrics_hawkular_requests_memory=750M
openshift_metrics_cassandra_requests_memory=750M

Cassandra Pod 的持久卷声明属性：

openshift_metrics_cassandra_storage_type=pv
openshift_metrics_cassandra_pvc_size=5Gi
openshift_metrics_cassandra_pvc_prefix=metrics

引导式练习: 安装指标子系统

[student@workstation]
步骤0. 准备

```
$ lab install-metrics setup

Checking prerequisites for GE: Installing the Metrics Subsystem

 Checking all VMs are running:
 · master VM is up..........................................  SUCCESS
 · node1 VM is up...........................................  SUCCESS
 · node2 VM is up...........................................  SUCCESS

Downloading files for GE: Installing the Metrics Subsystem

 · Download exercise files..................................  SUCCESS
 · Download solution files..................................  SUCCESS

Overall setup status........................................  SUCCESS
```

步骤1. 验证私有注册表中已包含指标子系统需要的容器镜像

```
$ docker-registry-cli \
  registry.lab.example.com \
  search metrics-cassandra \
  ssl
available options:-

-----------
1) Name: openshift3/ose-metrics-cassandra
Tags: latest  v3.9

1 images found !

$ docker-registry-cli \
  registry.lab.example.com \
  search metrics-hawkular-metrics \
  ssl
available options:-
```

```

-----------
1) Name: openshift3/ose-metrics-hawkular-metrics
Tags: latest  v3.9

1 images found !

$ docker-registry-cli \
  registry.lab.example.com \
  search metrics-heapster \
  ssl
available options:-

-----------
1) Name: openshift3/ose-metrics-heapster
Tags: latest  v3.9

1 images found !
```

步骤2. 查询

```
$ docker-registry-cli registry.lab.example.com search ose-recycler ssl
available options:-

-----------
1) Name: openshift3/ose-recycler
Tags: latest  v3.9

1 images found !
```

步骤3.

```
$ ssh root@services
Last login: Fri Feb 21 10:38:07 2020 from workstation.lab.example.com

# ls -alZ /exports/metrics/
drwxrwxrwx. nfsnobody nfsnobody unconfined_u:object_r:default_t:s0 .
drwxr-xr-x. root      root      unconfined_u:object_r:default_t:s0 ..

# grep metric /etc/exports.d/openshift-ansible.exports
"/exports/metrics" *(rw,root_squash)

# exit
logout
Connection to services closed.
```

步骤4. 创建持久卷

```
$ cat ~/DO280/labs/install-metrics/metrics-pv.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: metrics
spec:
  capacity:
    storage: 5Gi
  accessModes:
  - ReadWriteOnce
  nfs:
    path: /exports/metrics
    server: services.lab.example.com
  persistentVolumeReclaimPolicy: Recycle

$ oc login -u admin
Logged into "https://master.lab.example.com:443" as "admin" using existing credentials.
...输出被忽略...

$ oc create -f ~/DO280/labs/install-metrics/metrics-pv.yml
persistentvolume "metrics" created

$ oc get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      ...
...输出被忽略...
metrics   5Gi        RWO            Recycle          Available   ...
```

步骤5.

```
$ cd ~/DO280/labs/install-metrics
$ cat metrics-vars.txt
$ vim inventory
...
# Metrics Variables
# Append the variables to the [OSEv3:vars] group
openshift_metrics_install_metrics=True
openshift_metrics_image_prefix=registry.lab.example.com/openshift3/ose-
openshift_metrics_image_version=v3.9
openshift_metrics_heapster_requests_memory=300M
openshift_metrics_hawkular_requests_memory=750M
openshift_metrics_cassandra_requests_memory=750M
openshift_metrics_cassandra_storage_type=pv
openshift_metrics_cassandra_pvc_size=5Gi
openshift_metrics_cassandra_pvc_prefix=metrics

$ lab install-metrics grade
...输出被忽略...
Overall inventory file check: .................................  PASS

$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openshift-
metrics/config.yml
...输出被忽略...
PLAY RECAP **************************************************************************
localhost                  : ok=12   changed=0    unreachable=0    failed=0
master.lab.example.com     : ok=212  changed=47   unreachable=0    failed=0
node1.lab.example.com      : ok=0    changed=0    unreachable=0    failed=0
```

```
node2.lab.example.com      : ok=0    changed=0    unreachable=0    failed=0
services.lab.example.com   : ok=1    changed=0    unreachable=0    failed=0
workstation.lab.example.com : ok=4    changed=0    unreachable=0    failed=0

INSTALLER STATUS ***************************************************************************
Initialization             : Complete (0:00:30)
Metrics Install            : Complete (0:04:30)
```

步骤6. 验证

```
$ oc get pvc -n openshift-infra
NAME        STATUS    VOLUME    CAPACITY    ACCESS MODES    STORAGECLASS    AGE
metrics-1   Bound     metrics   5Gi         RWO                             4m

$ oc get pod -n openshift-infra
NAME                         READY     STATUS    RESTARTS    AGE
hawkular-cassandra-1-85j5w   1/1       Running   0           3m
hawkular-metrics-6szpv       0/1       Running   0           3m
heapster-79v98               0/1       Running   0           3m
```

步骤7. 访问 Hawkular 主网

```
$ oc get route -n openshift-infra
NAME               HOST/PORT                               ...
hawkular-metrics   hawkular-metrics.apps.lab.example.com   ...
```

firefox https://hawkular-metrics.apps.lab.example.com
"Metrics Service : STARTED "

步骤8.

```
$ oc login -u developer
Logged into "https://master.lab.example.com:443" as "developer" using existing credentials.
...输出被忽略...

$ oc new-project load
Now using project "load" on server "https://master.lab.example.com:443".
...输出被忽略...

$ oc new-app --name=hello --docker-image=registry.lab.example.com/openshift/hello-openshift
--> Found Docker image 7af3297 (22 months old) from registry.lab.example.com for "registry.lab.example.com/openshift/hello-openshift"
...输出被忽略...

$ oc scale --replicas=9 dc/hello
deploymentconfig "hello" scaled

$ oc get pod -o wide
NAME             READY    STATUS    ...   IP             NODE
```

```
hello-1-4cq66   1/1      Running   ...   10.129.1.17    node1.lab.example.com
hello-1-4kl9h   1/1      Running   ...   10.129.1.16    node1.lab.example.com
hello-1-8dnv4   1/1      Running   ...   10.129.1.15    node1.lab.example.com
hello-1-g8bpj   1/1      Running   ...   10.128.0.173   node2.lab.example.com
hello-1-gbgjc   1/1      Running   ...   10.128.0.174   node2.lab.example.com
hello-1-gfd7f   1/1      Running   ...   10.129.1.18    node1.lab.example.com
hello-1-gspbs   1/1      Running   ...   10.128.0.175   node2.lab.example.com
hello-1-mcvfc   1/1      Running   ...   10.128.0.176   node2.lab.example.com
hello-1-php6w   1/1      Running   ...   10.128.0.177   node2.lab.example.com

$ oc expose svc/hello
route "hello" exposed

$ ab -n 300000 -c 20 http://hello-load.apps.lab.example.com/ &
...输出被忽略...
Benchmarking hello-load.apps.lab.example.com (be patient)
Completed 30000 requests
...输出被忽略...
```

步骤9.

```
$ oc login -u admin
...输出被忽略...

$ oc adm top node --heapster-namespace=openshift-infra --heapster-scheme=https
NAME                     CPU(cores)   CPU%     MEMORY(bytes)   MEMORY%
master.lab.example.com   186m         9%       1312Mi          75%
node1.lab.example.com    512m         25%      617Mi           7%
node2.lab.example.com    492m         24%      3146Mi          40%
```

步骤10.

```
$ cat ~/DO280/labs/install-metrics/node-metrics.sh
#!/bin/bash

oc login -u admin -p redhat >/dev/null

TOKEN=$(oc whoami -t)
APIPROXY=https://master.lab.example.com:/api/v1/proxy/namespaces/openshift-infra/services
HEAPSTER=https:heapster:/api/v1/model
NODE=nodes/node1.lab.example.com
START=$(date -d '1 minute ago' -u '+%FT%TZ')

curl -kH "Authorization: Bearer $TOKEN" \
 -X GET $APIPROXY/$HEAPSTER/$NODE/metrics/memory/working_set?start=$START

curl -kH "Authorization: Bearer $TOKEN" \
 -X GET $APIPROXY/$HEAPSTER/$NODE/metrics/cpu/usage_rate?start=$START

$ ~/DO280/labs/install-metrics/node-metrics.sh
{
  "metrics": [
      {
        "timestamp": "2020-02-22T09:19:00Z",
        "value": 649232384
      },
      {
        "timestamp": "2020-02-22T09:19:30Z",
        "value": 650362880
      }
    ],
    "latestTimestamp": "2020-02-22T09:19:30Z"
  }{
    "metrics": [
      {
        "timestamp": "2020-02-22T09:19:00Z",
        "value": 534
      },
      {
        "timestamp": "2020-02-22T09:19:30Z",
        "value": 482
      }
    ],
    "latestTimestamp": "2020-02-22T09:19:30Z"
```

步骤11. Web 控制台

```
`firefox` https://master.lab.example.com
  developer%redhat
```

步骤12. 清理

```
$ oc delete project load
project "load" deleted
```

总结

- 红帽 OpenShift 容器平台提供了 **可选** 的指标子系统，它能够收集和长期存储集群节点和容器相关的性能指标。
- 指标子系统由三大组件组成，它们作为 OpenShift 集群中的容器运行：
  - **Heapster** 从 OpenShift 节点和各个节点上运行的容器收集指标。Kubernetes自动缩放器需要 Heapster 才能工作。
  - **Hawkular Metrics** 存储指标并提供查询功能。OpenShift Web 控制台需要 Hawkular 来显示项目的性能图形。
  - **Cassandra** 是 Hawkular 用来存储指标的数据库。
- Heapster 和 Hawkular Metrics 提供与外部监控系统集成 REST API。
- 必须使用 OpenShift Master API 代理，才能访问 Heapster API 并检索关于节点当前内存使用量、CPU 使用量和其他指标的信息。

- 配置指标子系统的建议方式是使用更改的 Ansible 变量运行安装程序 playbook。
- 高速指标子系统大小涉及多个参数：各个 Pod 的 CPU 和内存请求、各个持久卷的容量、以及各个 Pod 的副本数等。它们取决于OpenShift 集群中的节点数、预期的 Pod 数、指标存储的时长，以及收集指标的解析。
- 指标子系统安装 playbook 要求通过快速或高级 OpenShift 安装途径使用 Ansible 清单文件。同一 playbook 也用于卸载和重新配置指标子系统。
- 在运行安装 playbook 并验证所有指标子系统 Pod 已就绪并在运行后，所有 OpenShift 用户需要访问 Hawkular 欢迎页面来信任其 TLS 证书。若不执行此操作，Web 控制台将无法显示性能图形。

9. 管理和监控 OpenShift 容器平台

限制资源使用量

- Resource Requests and Limits for `Pods`
  - **资源请求**

    用于调度，并且指明 Pod 无法在计算资源少于指定数量下运行
  - **资源限值**

    用于防止 Pod 用尽节点上的所在计算资源。cgroup
- Applying Quotas: `project`

  跟踪和限制两种资源的使用量：
  - **对象数**

    Pod、服务和路由等 k8s 资源的数量
  - **计算资源**

    CPU、内存和存储的数量
- Applying Limit Ranges

  limit 为某一 Pod 或项目中定义的某一容器定义计算资源请求和限值的默认值、最小值和最大值
- Applying Quotas to Multiple Projects
  - 利用 openshift.io/requester 标来指定项目所有者
  - 使用选择器

引导式练习: 限制资源使用量

[student@workstation]
步骤0. 准备

```
$ lab monitor-limit setup

Checking prerequisites for GE: Limiting Resource Usage

 Checking all VMs are running:
  · master VM is up.....................................  SUCCESS
  · node1 VM is up......................................  SUCCESS
  · node2 VM is up......................................  SUCCESS

Downloading files for GE: Limiting Resource Usage

  · Download exercise files.............................  SUCCESS

Overall setup status....................................  SUCCESS
```

步骤1. 创建一个项目来验证创建新 Pod 时没有默认的资源请求

```
$ oc login -u admin
Logged into "https://master.lab.example.com:443" as "admin" using existing credentials.
...输出被忽略...

$ oc describe node node1.lab.example.com | grep -A 4 Allocated
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests  CPU Limits  Memory Requests  Memory Limits
  ------------  ----------  ---------------  -------------
  300m (15%)    0 (0%)      768Mi (9%)       0 (0%)

$ oc describe node node2.lab.example.com | grep -A 4 Allocated
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests  CPU Limits  Memory Requests   Memory Limits
  ------------  ----------  ---------------   -------------
  100m (5%)     0 (0%)      2068435456 (25%)  8250M (101%)

$ oc new-project resources
Now using project "resources" on server "https://master.lab.example.com:443".
...输出被忽略...

$ oc new-app --name=hello --docker-image=registry.lab.example.com/openshift/hello-openshift
--> Found Docker image 7af3297 (22 months old) from registry.lab.example.com for "registry.lab.example.com/openshift/hello-openshift"
...输出被忽略...
```

```
$ oc get pod \
  -o wide
NAME            READY     STATUS    ...  IP              NODE
hello-1-b4jzr   1/1       Running   ...  10.128.0.179   `node2.lab.example.com`

$ oc describe node node2.lab.example.com | grep -A 4 Allocated
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests  CPU Limits  Memory Requests   Memory Limits
  ------------  ----------  ---------------   -------------
  100m (5%)     0 (0%)      2068435456 (25%)  8250M (101%)
```

步骤2. 为项目添加 配额和限值范围

```
$ cat ~/DO280/labs/monitor-limit/limits.yml
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "project-limits"
spec:
  limits:
    - type: "Container"
      default:
        cpu: "250m"

$ oc create -f ~/DO280/labs/monitor-limit/limits.yml
limitrange "project-limits" created

$ oc describe limits
Name:       project-limits
Namespace:  resources
Type        Resource  Min  Max  Default Request  Default Limit  ...
----        --------  ---  ---  ---------------  -------------  ...
Container   cpu       -    -    250m             250m           ...

$ cat ~/DO280/labs/monitor-limit/quota.yml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: project-quota
spec:
  hard:
    cpu: "900m"

$ oc create -f ~/DO280/labs/monitor-limit/quota.yml
resourcequota "project-quota" created

$ oc describe quota
Name:       project-quota
Namespace:  resources
Resource    Used  Hard
--------    ----  ----
```

```
cpu         0     900m

$ oc adm policy add-role-to-user edit developer
role "edit" added: "developer"
```

步骤3. 在项目中创建 Pod，验证 Pod 会消耗项目配额中的资源

```
$ oc login -u developer
...输出被忽略...
Using project "resources".

$ oc get limits
NAME             AGE
project-limits   5m

$ oc delete limits project-limits
Error from server (Forbidden): limitranges "project-limits" is forbidden: User "developer" cannot delete limitranges in the namespace "resources": User "developer" cannot delete limitranges in project "resources"

$ oc get quota
NAME            AGE
project-quota   3m

$ oc new-app --name haha --docker-image=registry.lab.example.com/openshift/hello-openshift
--> Found Docker image 7af3297 (22 months old) from registry.lab.example.com for "registry.lab.example.com/openshift/hello-openshift"
...输出被忽略...

$ oc get pod
NAME            READY     STATUS    RESTARTS   AGE
haha-1-c5ms2    1/1       Running   0          30s

$ oc describe quota
Name:       project-quota
Namespace:  resources
Resource    Used  Hard
--------    ----  ----
cpu         250m  900m
```

步骤4. 可选：检查节点的可用资源是否变少

```
$ oc login -u admin
...输出被忽略...
Using project "resources".

$ oc get pod -o wide
NAME            READY     STATUS    ...  IP             NODE
haha-1-c5ms2    1/1       Running   ...  10.128.0.181   node2.lab.example.com
hello-1-b4jzr   1/1       Running   ...  10.128.0.179   node2.lab.example.com

$ oc describe node node2.lab.example.com | grep -A 4 Allocate
```

```
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests  CPU Limits  Memory Requests   Memory Limits
  ------------  ----------  ---------------   -------------
  350m (17%)    250m (12%)  2068435456 (25%)  8250M (101%)

$ oc describe pod haha-1-c5ms2 | grep -A 2 Requests
    Requests:
      cpu:        250m
    Environment:  <none>

$ oc login -u developer
...输出被忽略...
Using project "resources".
```

步骤5. 扩展部署配置

```
$ oc scale dc haha --replicas=2
deploymentconfig "haha" scaled

$ oc get pod
NAME             READY     STATUS    RESTARTS   AGE
haha-1-c5ms2     1/1       Running   0          7m
haha-1-rxgng     1/1       Running   0          6s
hello-1-b4jzr    1/1       Running   0          20m

$ oc describe quota
Name:       project-quota
Namespace:  resources
Resource    Used  Hard
--------    ----  ----
cpu         500m  900m

$ oc scale dc haha --replicas=4
deploymentconfig "haha" scaled

$ oc get pod
NAME             READY     STATUS    RESTARTS   AGE
haha-1-c5ms2     1/1       Running   0          8m
haha-1-lhdt8     1/1       Running   0          7s
haha-1-rxgng     1/1       Running   0          1m
hello-1-b4jzr    1/1       Running   0          21m

$ oc describe dc haha | grep Replicas
Replicas: 4
  Replicas: 3 current / 4 desired

$ oc get events | grep -i error
...输出被忽略...
          Error creating: pods "haha-1-b9kvp" is forbidden: exceeded quota: project-quota, requested: cpu=250m, used: cpu=750m, limited: cpu=900m

```

```
$ oc scale dc haha --replicas=1
deploymentconfig "haha" scaled

$ oc get pod
NAME            READY     STATUS    RESTARTS   AGE
haha-1-c5ms2    1/1       Running   0          11m
hello-1-b4jzr   1/1       Running   0          24m
```

步骤6. 添加不受项目配额限制的资源请求

```
$ oc set resources dc haha --requests=memory=256Mi
deploymentconfig "haha" resource requirements updated

$ oc get pod
NAME            READY     STATUS        RESTARTS   AGE
haha-1-c5ms2    0/1       Terminating   0          13m
haha-3-m2xd9    1/1       Running       0          13s

$ oc describe pod haha-3-m2xd9 | grep -A 3 Requests
    Requests:
      cpu:        250m
      memory:     256Mi
    Environment:  <none>

$ oc describe quota
Name:       project-quota
Namespace:  resources
Resource    Used  Hard
--------    ----  ----
cpu         250m  900m
```

步骤7. 将内存资源请求增加到超过集群中任何节点的容量值

```
$ cat ~/DO280/labs/monitor-limit/increase-toomuch.sh
#!/bin/bash -x

oc set resources dc hello --requests=memory=8Gi

ok="no"
while [ "$ok" != "yes" ]
do
  sleep 3
  oc get pod

  echo -n "Type 'yes' to proceed."
  read ok
done

oc get events | grep hello-3.*Failed

$ oc set resources dc haha --requests=memory=8Gi
deploymentconfig "haha" resource requirements updated
```

```
$ oc get pod
NAME             READY     STATUS    RESTARTS   AGE
haha-3-m2xd9     1/1       Running   0          20m
haha-4-deploy    0/1       Error     0          16m
hello-1-b4jzr    1/1       Running   0          46m
$ oc logs haha-4-deploy
--> Scaling up haha-4 from 0 to 1, scaling down haha-3 from 1 to 0 (keep 1 pods available,
don't exceed 2 pods)
    Scaling haha-4 up to 1
error: timed out waiting for any update progress to be made

$ oc status
...输出被忽略...
svc/haha - 172.30.245.206 ports 8080, 8888
  dc/haha deploys istag/haha:latest
    deployment #4 failed 21 minutes ago: config change
    deployment #3 deployed 25 minutes ago - 1 pod
    deployment #2 failed 26 minutes ago: newer deployment was found running
...输出被忽略...

$ oc get events | grep haha-4.*Failed
17m   23m   26   haha-4-hjlck.15f5b199c23cb0bd   Pod   Warning   FailedScheduling
default-scheduler                                                0/3 nodes are available:
1 MatchNodeSelector, 3 Insufficient memory.
13m   13m   1   haha-4-hjlck.15f5b2261a56621f   Pod   Warning   FailedScheduling
default-scheduler                                                skip schedule deleting
pod: resources/haha-4-hjlck
```

步骤8. 清理

```
$ oc login -u admin
Logged into "https://master.lab.example.com:443" as "admin" using existing credentials.
...输出被忽略...

$ oc delete project resources
project "resources" deleted
```

升级 OpenShift 容器平台

- Upgrading OpenShift

  应用 **最新功能** 和 **漏洞修补**

- Upgrade Methods
  - In-place Upgrades 就地升级
  - Blue-green Deployments 蓝绿布署
- Performing an Automated Cluster Upgrade

- Preparing for an Automated Upgrade

```
# subscription-manager repos \
  --disable="rhel-7-server-ose-3.7-rpms" \
  --enable="rhel-7-server-ose-3.9-rpms" \
  --enable="rhel-7-server-ose-3.8-rpms" \
  --enable="rhel-7-server-rpms" \
  --enable="rhel-7-server-extras-rpms" \
  --enable="rhel-7-server-ansible-2.4-rpms" \
  --enable="rhel-7-fast-datapath-rpms"
# yum clean all

# yum update atomic-openshift-utils

# oc label node1.lab.example.com region=infra --overwrite

# vim inventory
...
openshift_disable_swap=false
```

- Upgrading Master and Application Nodes

```
# vim inventory
...
openshift_deployment_type=openshift-enterprise
openshift_web_console_prefix=registry.lab.example.com/openshift3/ose-
template_service_brokder_prefix=registry.lab.example.com/openshift3/ose-
# ansible-playbook upgrade.yml
# for i in master node1 node2; do
  ssh root@$i reboot
  done
```

- Upgrading the Cluster in Multiple Phases

```
# ansible-playbook \
  /usr/share/ansible/openshift-ansible/playbooks/common/openshift-
cluster/upgrades/v3_9/upgrade_nodes.yml \
  -e openshift_upgrade_nodes_serial="50%"
# ansible-playbook \
  /usr/share/ansible/openshift-ansible/playbooks/common/openshift-
cluster/upgrades/v3_9/upgrade_nodes.yml \
  -e openshift_upgrade_nodes_serial="2"
  -e openshift_upgrade_nodes_label="region=HA"
# ansible-playbook \
  /usr/share/ansible/openshift-ansible/playbooks/common/openshift-
cluster/upgrades/v3_9/upgrade_nodes.yml \
  -e openshift_upgrade_nodes_serial=10 \
  -e openshift_upgrade_nodes_max_fail_percentage=20 \
  -e openshift_upgrade_nodes_drain_timeout=600
```

- Using Ansible Hooks

```
$ vim inventory
```

```
...
[OSEv3:vars]
openshift_master_upgrade_pre_hook=/usr/share/custom/pre_master.yml
openshift_master_upgrade_hook=/usr/share/custom/master.yml
openshift_master_upgrade_post_hook=/usr/share/custom/post_master.yml
```

- Verifying the Upgrade

```
$ oc get nodes
$ oc get -n default dc/docker-registry -o json | grep \"image\"
$ oc get -n default dc/router -o json | grep \"image\"
$ oc adm diagnostics
```

测验: 升级 OpenShift

下方显示了自动升级 OpenShift 集群的步骤。请标明运行这些步骤的正确顺序

_2_a. 确保每个 RHEL7 上都有最新版本的 **`atomic-openshift-utils`** 包

_6_b. 可选，如果你使用自定义 Docker 注册表，请将注册表的地址指定给变量 **`openshift_web_console_prefix`** 和 **`template_service_broker_prefix`**

_4_c. 禁用所有节点上的交换内存

_8_d. 重启所有主机。重新启动后，请检查升级

_3_e. 可选，查看清单文件中的节点选择器

_1_f. 禁用 3.7 存储库，并在每个 maste 和 node 上启用 3.8 和 3.9 存储库

_7_g. 通过使用适当的 ansible 剧本，使用单阶段或多阶段策略进行更新

_5_h. 在主机清单中设置变量 **`openshift_deployment_type=openshift-enterprise`**

使用探测监控应用

- Introduction to OpenShift Probes

探测监控应用。

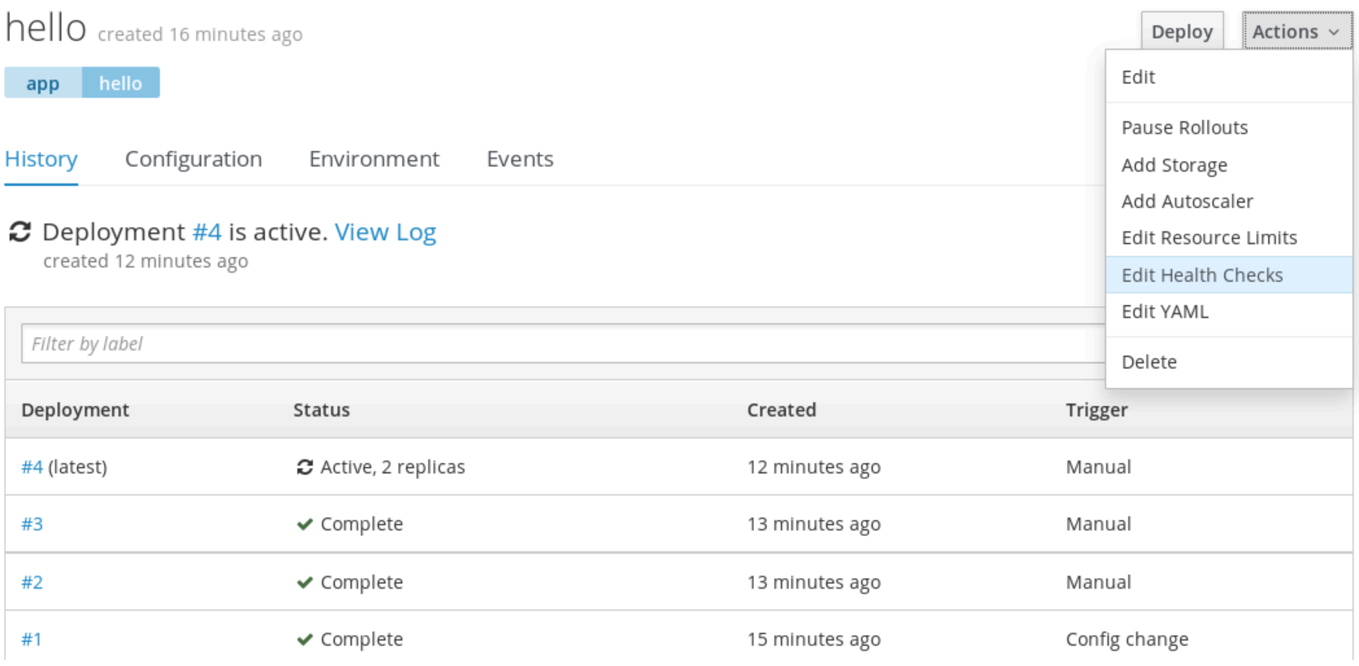
两种探测类型：

- 存活度探测
- 就绪度探测

- Methods of Checking Application Health

> 三种方式：
>
> - HTTP Checks（HTTP 检查）
> - Container Execution Checks（容器执行检查）
> - TCP Socket Checks（TCP 套接字检查）

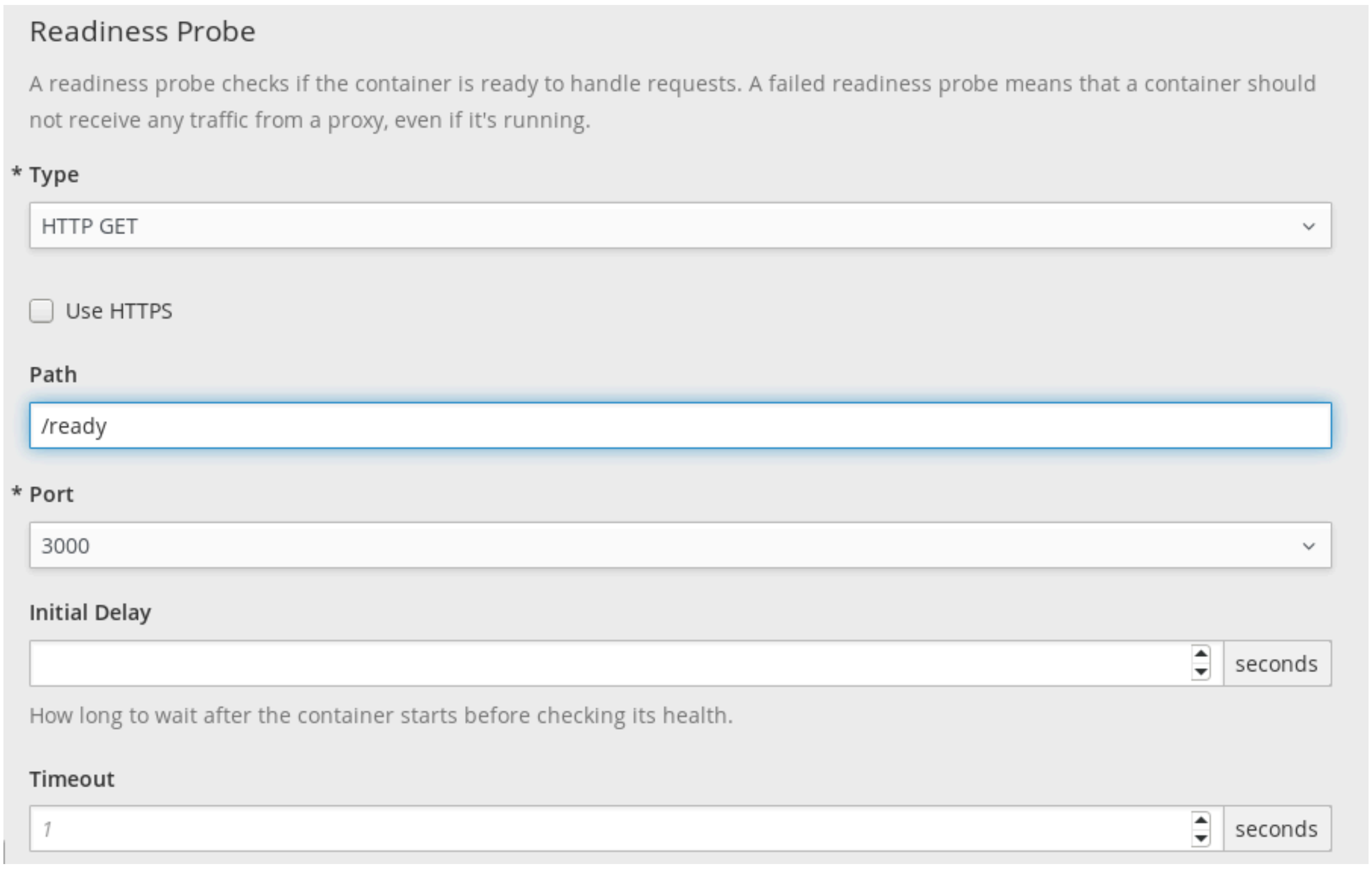
- Using the Web Console to Manage Probes


hello created 16 minutes ago

app hello

Deployment #4 is active. View Log
created 12 minutes ago

Deployment	Status	Created	Trigger
#4 (latest)	Active, 2 replicas	12 minutes ago	Manual
#3	✓ Complete	13 minutes ago	Manual
#2	✓ Complete	13 minutes ago	Manual
#1	✓ Complete	15 minutes ago	Config change



Readiness Probe

A readiness probe checks if the container is ready to handle requests. A failed readiness probe means that a container should not receive any traffic from a proxy, even if it's running.

* **Type**

HTTP GET

☐ Use HTTPS

Path

/ready

* **Port**

3000

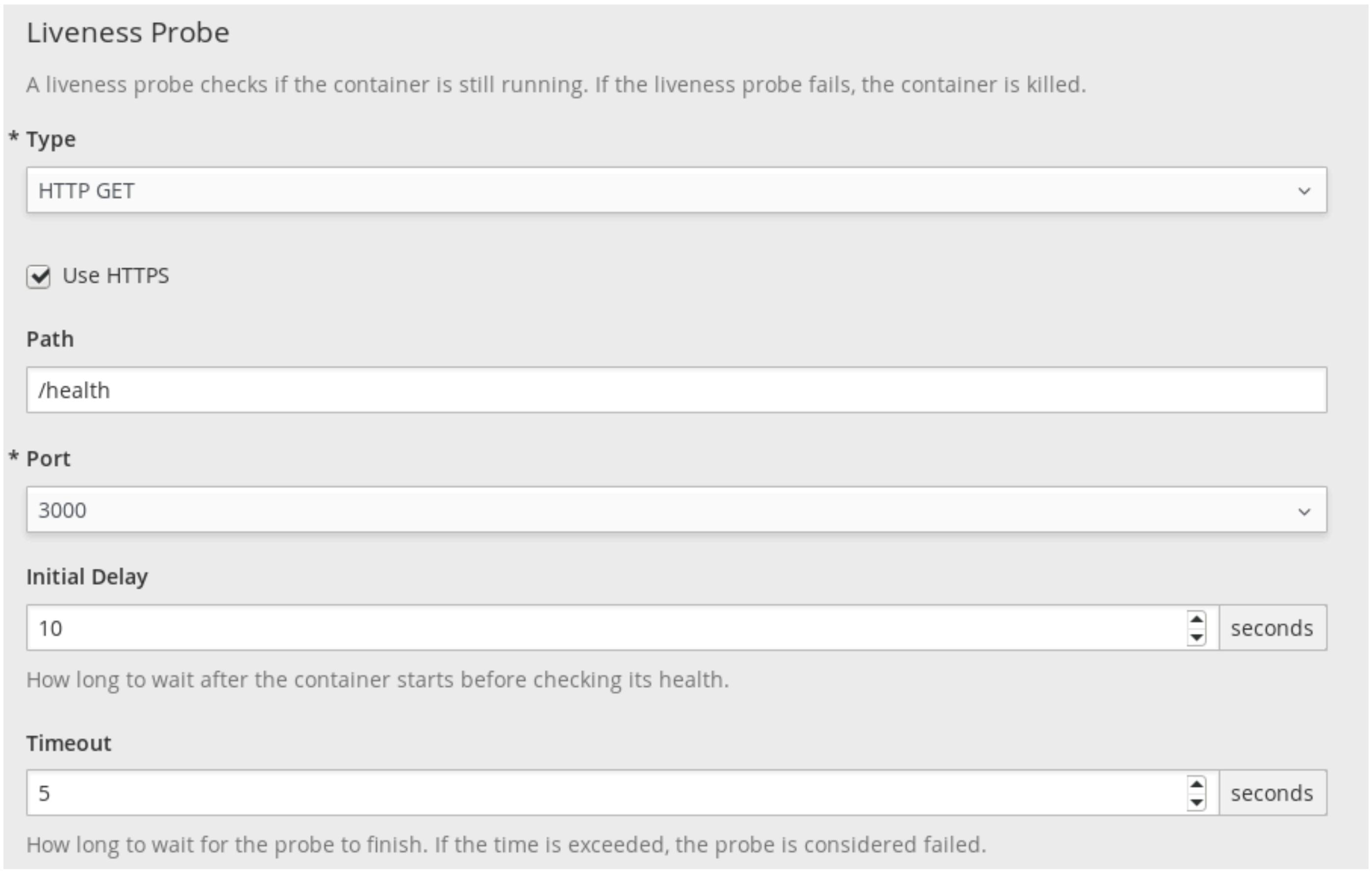
Initial Delay

seconds

How long to wait after the container starts before checking its health.

Timeout

1 seconds



Liveness Probe

A liveness probe checks if the container is still running. If the liveness probe fails, the container is killed.

* **Type**

HTTP GET

[✓ Use HTTPS]

Path

/health

* **Port**

3000

Initial Delay

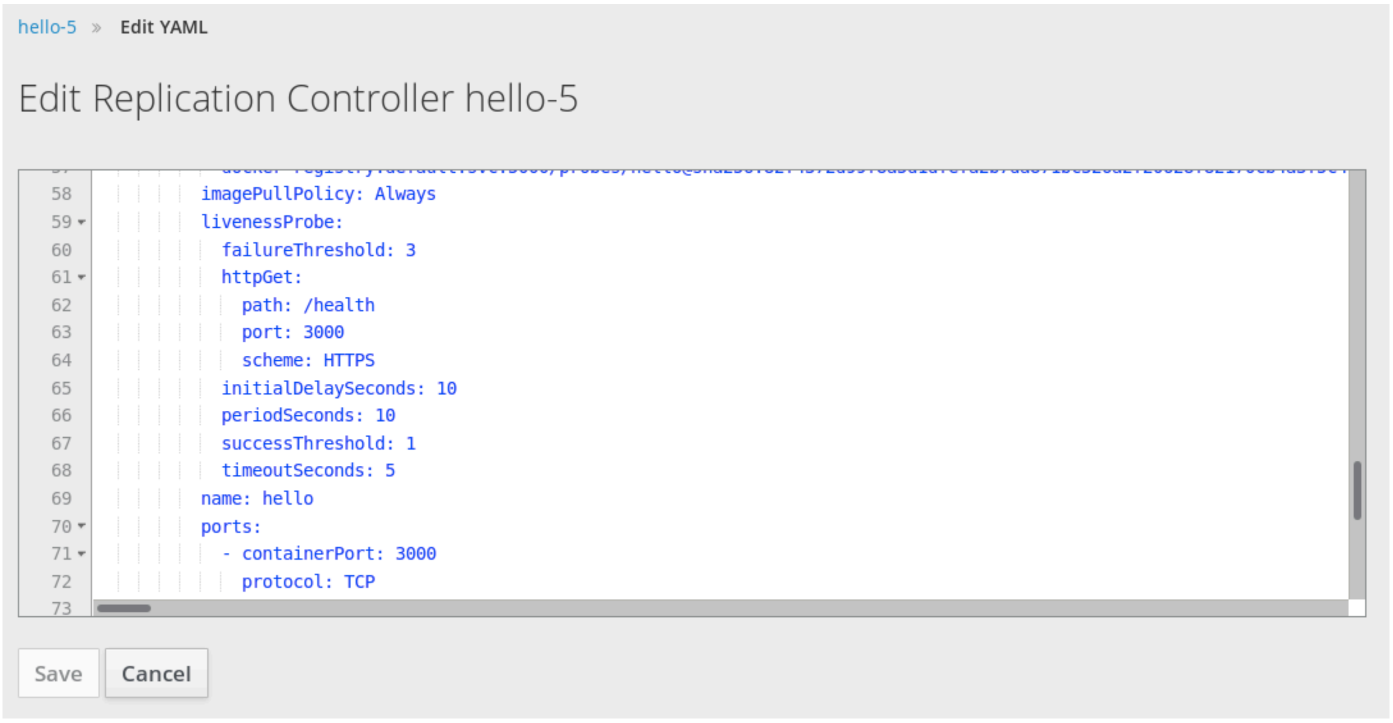
10 seconds

How long to wait after the container starts before checking its health.

Timeout

5 seconds

How long to wait for the probe to finish. If the time is exceeded, the probe is considered failed.


hello-5 » Edit YAML

Edit Replication Controller hello-5

```
        imagePullPolicy: Always
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /health
            port: 3000
            scheme: HTTPS
          initialDelaySeconds: 10
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        name: hello
        ports:
          - containerPort: 3000
            protocol: TCP
```

Save Cancel

引导式练习: 使用探测监控应用

[student@workstation]
步骤0. 准备

```
$ lab probes setup

Checking prerequisites for GE: Monitoring Applications with Probes

 Checking all VMs are running:
  · master VM is up.............................................  SUCCESS
  · node1 VM is up..............................................  SUCCESS
  · node2 VM is up..............................................  SUCCESS

 Checking all OpenShift default pods are ready and running:

  · Check router................................................  SUCCESS
  · Check registry..............................................  SUCCESS

Overall setup status............................................  SUCCESS
```

步骤1. 创建项目

```
$ oc login -u developer
Logged into "https://master.lab.example.com:443" as "developer" using existing credentials.
...输出被忽略...

$ oc new-project probes
Now using project "probes" on server "https://master.lab.example.com:443".
...输出被忽略...
```

步骤2. 创建应用

```
$ oc new-app --name=probes http://services.lab.example.com/node-hello
--> Found Docker image fba56b5 (2 years old) from registry.lab.example.com for
"registry.lab.example.com/rhscl/nodejs-6-rhel7"
...输出被忽略...

$ oc status
In project probes on server https://master.lab.example.com:443

svc/probes - 172.30.44.9:3000
  dc/probes deploys istag/probes:latest <-
    bc/probes docker builds http://services.lab.example.com/node-hello on istag/nodejs-6-
rhel7:latest
      build #1 running for 38 seconds - aaf02db: Establish remote repository (root
<root@services.lab.example.com>)
    deployment #1 waiting on image or update

2 infos identified, use 'oc status -v' to see details.

$ oc get pods
NAME             READY     STATUS     RESTARTS    AGE
probes-1-build   1/1       Running    0           1m
```

步骤3. 公开服务路由

```
$ oc expose svc probes --hostname=probe.apps.lab.example.com
route "probes" exposed
```

步骤4. curl 命令测试

```
$ curl http://probe.apps.lab.example.com
Hi! I am running on host -> probes-1-xgcmt
```

步骤5. curl 命令 GET /health , GET /ready

```
$ curl http://probe.apps.lab.example.com/health
OK
$ curl http://probe.apps.lab.example.com/ready
READY
```

步骤6. Web 控制台再创建就绪度探测，创建存活探测

firefox https://master.lab.example.com
developer%redhat

"My Projectc" **probes**
Applications > **Deployments**
probes **#1**
Action > **Edit Health Checks**
Add Readiness Probe
* Type **HTTP GET**
Path **/ready**
* Port 3000
Initial Delay **3**
Timout **2**
Add Liveness Probe
* Type **HTTP GET**
Path **/healtz**
* Port 3000
Initial Delay **3**
Timout **3**
单击 **Save** 命令按钮

步骤7.

Monitoring > **Events**
Notice "Unhealthy"

View Details /

```
$ oc get events --sort-by='.metadata.creationTimestamp' | grep 'probe fail'
7m          9m           7         probes-2-twf2w.15f5b59ba2e30f92    Pod
  spec.containers{probes}                    Warning   Unhealthy              kubelet,
node1.lab.example.com   Liveness probe failed: HTTP probe failed with statuscode: 404
```

步骤8. 编辑 liveness probe

```
  `Applications` > `Deployments`
      probes `#2`
        `Add Liveness Probe`
            Path `/health`
          单击`Save`命令按钮
```

步骤9.

```
$ oc get events --sort-by='.metadata.creationTimestamp'
...输出被忽略...
1m          1m           1         probes-3-xdmwl.15f5b638b43231ff    Pod                  spec.containers{probes}                 Normal    Pulling                  kubelet, node2.lab.example.com   pulling image "docker-registry.default.svc:5000/probes/probes@sha256:8a57a71937e6499a2045e52a0ecc5adff324c5f72e9ad198f7c2871ad6a1bbd3"
1m          1m           1         probes-3-xdmwl.15f5b6392f73a629    Pod                  spec.containers{probes}                 Normal    Created                  kubelet, node2.lab.example.com   Created container
1m          1m           1         probes-3-xdmwl.15f5b6391f1bc96d    Pod                  spec.containers{probes}                 Normal    Pulled                   kubelet, node2.lab.example.com   Successfully pulled image "docker-registry.default.svc:5000/probes/probes@sha256:8a57a71937e6499a2045e52a0ecc5adff324c5f72e9ad198f7c2871ad6a1bbd3"
1m          1m           1         probes-3-xdmwl.15f5b639662fd11b    Pod                  spec.containers{probes}                 Normal    Started                  kubelet, node2.lab.example.com   Started container
...输出被忽略...
```

步骤10. 清理

```
$ oc delete project probes
project "probes" deleted
```

使用 Web 控制台监控资源

- Introduction to the Web Console

- Managing Metrics with Hawkular

- Managing Deployments and Pods

- Managing Storage

引导式练习: 使用 Web 控制台探索指标

[student@workstation]
步骤0. 准备

```
$ lab web-console setup

Checking prerequisites for GE: Monitoring Resources with the Web Console

 Checking all VMs are running:
 · master VM is up.........................................  SUCCESS
 · node1 VM is up..........................................  SUCCESS
 · node2 VM is up..........................................  SUCCESS

 Checking all OpenShift default pods are ready and running:

 · Checking pod router.....................................  SUCCESS
 · Checking pod registry...................................  SUCCESS
 · Checking pod hawkular-cassandra.........................  SUCCESS
 · Checking pod hawkular-metrics...........................  SUCCESS
 · Checking pod heapster...................................  SUCCESS

Setting storage for the exercise

 · Creating NFS directory .................................  SUCCESS
 · Setting NFS configuration...............................  SUCCESS
 . Creating Persistent Volume..............................  SUCCESS

Overall setup status.......................................  SUCCESS
```

步骤1. 创建项目

```
$ oc login -u developer
Logged into "https://master.lab.example.com:443" as "developer" using existing credentials.
...输出被忽略...

$ oc new-project load
Now using project "load" on server "https://master.lab.example.com:443".
...输出被忽略...
```

步骤2. 创建应用

```
$ oc new-app --name=load http://services.lab.example.com/node-hello
--> Found Docker image fba56b5 (2 years old) from registry.lab.example.com for
"registry.lab.example.com/rhscl/nodejs-6-rhel7"
...输出被忽略...

$ oc expose svc/load
route "load" exposed

$ oc get pods
NAME            READY     STATUS    RESTARTS   AGE
load-1-build    1/1       Running   0          39s
```

步骤3. 生成负载

```
$ ab -n 3000000 -c 20 http://load-load.apps.lab.example.com/ &
```

步骤4. Web 控制台，向上扩展 Pod

firefox https://master.lab.example.com

developer%redhat

load

Overview

load / **>** / ⬆️

步骤5. 检查指标

Application > **Deployment**

load-2 / **Metrics**

步骤6. Monitoring

Monitoring

>

步骤7. 创建卷声明

Storage
  Create Storage
    * Name **web-storage**
    * Access Mode **shared Access (RWX)**
    * Size **1** GiB
    Create

步骤8. 添加存储到你的应用

Applications > **Deployments**
  load / **Actions** / **Add Storage**
    Mount Path **/web-storage**
    Volume Name **web-storage**
    Add

步骤9. 检查存储

Applications > **Pods**
  load-3-x7jtk / **Terminal**

```
sh-4.2$ mount | grep web-storage
master.lab.example.com:/var/export/web-storage-ge on /web-storage type nfs4 ...
```

步骤10. 清理

```
$ kill %1

$ oc delete project load
project "load" deleted
```

实验: 管理和监控 OpenShift

[student@workstation]
步骤0. 准备

```
$ lab review-monitor setup

Checking prerequisites for Lab: Managing and Monitoring OpenShift Container Platform

 Checking all VMs are running:
 · master VM is up........................................  SUCCESS
 · node1 VM is up.........................................  SUCCESS
 · node2 VM is up.........................................  SUCCESS

 Checking all OpenShift default pods are ready and running:

 · Checking pod hawkular-cassandra............................  SUCCESS
 · Checking pod hawkular-metrics..............................  SUCCESS
 · Checking pod heapster......................................  SUCCESS

Downloading files for Lab: Managing and Monitoring OpenShift Container Platform

 · Download exercise files....................................  SUCCESS

Overall setup status...........................................  SUCCESS
```

步骤1. 创建项目

```
$ oc login -u developer
Logged into "https://master.lab.example.com:443" as "developer" using existing credentials.
...输出被忽略...

$ oc new-project load-review
Now using project "load-review" on server "https://master.lab.example.com:443".
...输出被忽略...
```

步骤2. limits.yml

```
$ cat ~/DO280/labs/monitor-review/limits.yml
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "review-limits"
spec:
  limits:
    - type: "Container"
      max:
        memory: "300Mi"
      default:
        memory: "200Mi"

$ oc login -u admin
...输出被忽略...
Using project "load-review".

$ oc create -f ~/DO280/labs/monitor-review/limits.yml
limitrange "review-limits" created

$ oc describe limits
Name:       review-limits
Namespace:  load-review
Type        Resource  Min  Max    Default Request  Default Limit  ...
----        --------  ---  ---    ---------------  -------------  ...
Container   memory    -    300Mi  200Mi            200Mi
```

步骤3. 创建应用

```
$ oc login -u developer
...输出被忽略...
Using project "load-review".

$ oc new-app --name load http://services.lab.example.com/node-hello--> Found Docker image fba56b5 (2 years old) from registry.lab.example.com for "registry.lab.example.com/rhscl/nodejs-6-rhel7"
...输出被忽略...
```

步骤4. 确认限制值匹配项目设置

```
$ oc get pod
NAME           READY     STATUS      RESTARTS   AGE
load-1-build   0/1       Completed   0          2m
load-1-j7hlk   1/1       Running     0          18s

$ oc describe pod load-1-j7hlk | grep -A 3 Limits
    Limits:
      memory:  200Mi
    Requests:
      memory:     200Mi
```

步骤5. 请求 350M 内存被拒绝，恢复为 200M

```
$ oc set resources dc load --requests=memory=350Mi
deploymentconfig "load" resource requirements updated

$ oc get events | grep Warning.*350
19s         37s          4         load-2.15f5bb8da16dd6e5          ReplicationController               Warning   FailedCreate   replication-controller           (combined from similar events): Error creating: Pod "load-2-pmwpl" is invalid: spec.containers[0].resources.requests: Invalid value: "350Mi": must be less than or equal to memory limit

$ oc set resources dc load --requests=memory=200Mi
deploymentconfig "load" resource requirements updated

$ oc status; oc get pod
In project load-review on server https://master.lab.example.com:443

svc/load - 172.30.218.185:3000
  dc/load deploys istag/load:latest <-
    bc/load docker builds http://services.lab.example.com/node-hello on istag/nodejs-6-rhel7:latest
    deployment #3 deployed about a minute ago - 1 pod
    deployment #2 failed 4 minutes ago: newer deployment was found running
    deployment #1 deployed 7 minutes ago

2 infos identified, use 'oc status -v' to see details.
NAME           READY     STATUS      RESTARTS   AGE
load-1-build   0/1       Completed   0          9m
load-3-d2brs   1/1       Running     0          1m
```

步骤6. quotas.yml

```
$ cat ~/DO280/labs/monitor-review/quotas.yml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: review-quotas
spec:
  hard:
    requests.memory: "600Mi"

$ oc login -u admin
Logged into "https://master.lab.example.com:443" as "admin" using existing credentials.
...输出被忽略...

$ oc create -f ~/DO280/labs/monitor-review/quotas.yml
resourcequota "review-quotas" created

$ oc describe quota
Name:            review-quotas
Namespace:       load-review
Resource         Used   Hard
---------        ----   ----
requests.memory  200Mi  600Mi
```

步骤7. 向上扩容四个副本，无法创建第四个

```
$ oc login -u developer
...输出被忽略...
Using project "load-review".

$ oc scale --replicas=4 dc load
deploymentconfig "load" scaled

$ oc get pods
NAME           READY     STATUS      RESTARTS   AGE
load-1-build   0/1       Completed   0          15m
load-3-5qtck   1/1       Running     0          10s
load-3-d2brs   1/1       Running     0          8m
load-3-s48jj   1/1       Running     0          10s

$ oc get events | grep Warning.*quota
...输出被忽略...
39s         47s          7         load-3.15f5bc230dae371d          ReplicationController
                                                    Warning    FailedCreate
replication-controller           (combined from similar events): Error creating: pods
"load-3-wv76h" is forbidden: exceeded quota: review-quotas, requested:
requests.memory=200Mi, used: requests.memory=600Mi, limited: requests.memory=600Mi

$ oc scale --replicas=1 dc load
deploymentconfig "load" scaled
```

步骤8. 公开路由

```
$ oc expose svc load --hostname=load-review.apps.lab.example.com
route "load" exposed
```

步骤9. Web 控制台创建存活度探测

firefox https://master.lab.example.com
  developer%redhat

"My Projectc" **load-review**
  Applications > **Deployments**
    load
      Action > **Edit Health Checks**
        Add Liveness Probe
          * Type **HTTP GET**
          Path **/health**
          * Port 3000
          Initial Delay **10**
          Timout **3**
      单击 **Save** 命令按钮

步骤10. 验证

"My Projectc" **load-review**
  Applications > **Deployments**
    load / **History** / **#4 (latest)**

步骤11. 评分

```
$ lab review-monitor grade

Grading the student's work for Lab: Managing and Monitoring OpenShift Container Platform

 · Ensuring load-review is created............................  PASS
 · Ensuring limits for load-review is created..................  PASS
 · Reviewing limits for load-review............................  PASS
 · Ensuring application load is created........................  PASS
 · Checking events for limits violation........................  PASS
 · Checking the DC to make sure limit is set to 200 Mi.........  PASS
 · Ensuring quota for load-review is existing..................  PASS
 · Reviewing quotas for load-review............................  PASS
 · Checking events for quota violation.........................  PASS
 · Ensuring route is exposed...................................  PASS

Reviewing Liveness Probe

 · Ensuring Liveness probe is created..........................  PASS
```

```
      · Checking failureThreshold..................................  PASS
      · Checking Type..............................................  PASS
      · Checking Path..............................................  PASS
      · Checking Port..............................................  PASS
      · Checking Initial Delay.....................................  PASS
      · Checking Timeout...........................................  PASS

   Overall exercise grade..........................................  PASS
```

步骤12. 清理

```
$ oc delete project load-review
project "load-review" deleted
```

总结

- OpenShift 容器平台可以实施配额来跟踪和限制以下两种资源的使用量：对象数和计算资源。
- 可以通过两种方法执行 OpenShift 容器平台集群升级：通过 Ansible Playbooks 就地升级，或使用蓝绿部署方法升级。
- 群集升级一次不能跨越多个次要版本，因此，如果群集的版本早于 3.6，则必须首先增量升级。例如，3.5 到 3.6，然后3.6 到 3.7。否则可能导致升级失败。
- OpenShift 应用可能会因为临时连接丢失、配置错误、应用错误和类似问题而变得不健康。开发人员可以使用探测来监视其应用，从而帮助管理这些问题。
- Web 控制台集成了提供实时反馈的一组功能，如显示部署、Pod、服务和其他资源的状态，以及提供关于系统范围事件的信息。

A. 参考和附录

A1. 相关文件

1	**`DO280-OCP3.9.md`**	课堂笔记
2	do280 文件夹	培训环境
3	**`VMware Workstation`** == Linux or Windows **`VMware Fusion`** == MacOS	软件
4	Applications / Education / **`Slides`** foundation:/content/slides/*	幻灯
5	**`ex280.iso`**	模拟考试环境
6	**`EX280-OCP3.9-QA.html`**	考试类型题+QA

A2. 必须掌握 Linux 学习思路

ID	STEP	COMMENT
1	word	背单词
2	TAB	一下补全，两个列出
3	$ **`man`** COMMND $ COMMAND **`-h`**	看帮助
4	$ **`echo $?`**	看回显 == 0 == true

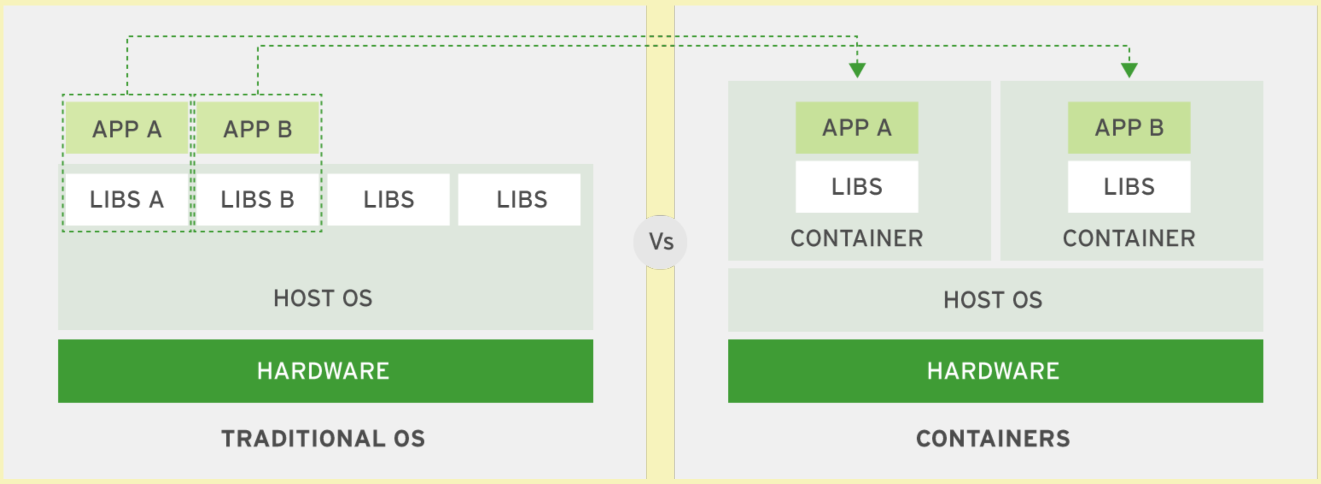
A3. 常用网址

ID	名称	网址
1	Product Documentation for OpenShift Container Platform 3.9	https://access.redhat.com/documentation/zh-cn/openshift_container_platform/3.9/
2	Product Documentation for OpenShift Container Platform 4.3	https://access.redhat.com/documentation/zh-cn/openshift_container_platform/4.3/?extIdCarryOver=true&sc_cid=701f2000001OH74AAG
3	okd	https://www.okd.io
4	kubernetes	https://kubernetes.io
5	docker	https://www.docker.com

A4. 什么是 PaaS

缩写	IaaS	PaaS	SaaS
全拼	Infrastructure-as-a-Service	Platform-as-a-Service	Software-as-a-Service
中文	基础设施即服务	平台即服务	软件即服务
示例	亚马逊、IBM等	Google、Microsoft Azure 等	阿里的钉钉、苹果的 iCloud 等
	CL210 OpenStack	DO280 OpenShift	

A5. 容器和操作系统对比



A6. RHCA

ID	COURSE	CONTENT	COMMENT
1	DO407	Ansible	自动化工具
2	CL210	OpenStack	IaaS
3	DO280	OpenShift	PaaS
4	Ceph125	Ceph	存储
5	RH236	Glusterfs	存储

A7. vim

`i` command mode

`Esc` exit edit mode

`:x` lastline mode **`x`** = **`write`** + **`quit`**

`/targetport` , **`o`** , `Space` *4, ...

`:%s/hello/hellos/g`

`d` `t` `'`

A8. yaml

1. **`---`** firstline
2. **`-`** line head , every play
3. **`key:`** next level
4. **`key: content`** `Space`
5. `TAB` no, `Space` *2

A9. ssh no-pass

```
$ ssh-keygen -N "" -f ~/.ssh/id_new
$ ssh-copy-id -i ~/.ssh/id_new.pub instructor@materials
$ ssh -i ~/.ssh/id_new instructor@materials

$ ssh-keygen -N "" -f ~/.ssh/id_rsa
$ ssh-copy-id instructor@materials
$ ssh instructor@materials
```

A10. sudo no-pass

```
[materials]# visudo
...
instructor      ALL=(ALL)       NOPASSWD: ALL
[workstation]$ ssh -i ~/.ssh/id_new instructor@materials sudo whoami
```

A11. 模拟考试环境

STEP	
1	虚拟机， **恢复到快照**
2	开机
3	插入光盘镜像 **exam280.iso**
4	[kiosk@foundation0 ~]$ **bash /run/media/kiosk/do280/exam280/exam-setup.sh**
5	[kiosk@foundation0 ~]# **shutdown -h 0**
6	**虚拟机** / **快照** / **拍摄此虚拟机的快照**
7	开机/[kiosk@foundation0 ~]# **rht-vmctl start all**
Q	~kiosk/Desktop/ **EX280-OCP2.9-Q.html**
T	[root@master]#

A12. EX280-Q10

prepare

[kiosk@foundation]

```
$ ssh student@workstation lab install-metrics setup
$ scp -r student@workstation:~student/DO280/labs/install-metrics/ root@master:~
$ ssh root@master sed -i '/default/ahost_key_checking = False' ~student/DO280/labs/install-metrics/ansible.cfg
$ scp ~/.ssh/id_rsa root@master ~/.ssh
```

exam

[root@master]

```
# vim pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: metric
spec:
  capacity:
    storage: 5Gi
  accessModes:
  - ReadWriteOnce
  nfs:
    path: /exports/metrics
    server: services.lab.example.com
  persistentVolumeReclaimPolicy: Recycle
# oc create -f pv.yaml
# oc get pv

# cd install-metrics/
# vim inventory
...
openshift_metrics_image_prefix=registry.lab.example.com/openshift3/ose-
openshift_metrics_image_version=v3.9
openshift_metrics_heapster_requests_memory=300M
openshift_metrics_hawkular_requests_memory=750M
openshift_metrics_cassandra_requests_memory=750M
openshift_metrics_cassandra_storage_type=pv
openshift_metrics_cassandra_pvc_size=5Gi
openshift_metrics_cassandra_pvc_prefix=metrics
openshift_metrics_install_metrics=True
# ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openshift-metrics/config.yml
```

A13. DDNS

- dhcp+dns
- 花生壳
- 一个静态DNS == 动态IP

A14. SD?

SDN	软件定义型网络	Network	
SDS	软件定义型存储	Storage	Glusters, Cephfs

A15. Kvm

- mac intel

  EFI

```
# 0. lvextend

COURSES=$(ssh root@localhost rht-usb f0list 2>/dev/null | awk -F: '/icmf/ {print $2}' | cut -f1 -d- | grep -v RHCI | tr A-Z a-z)

select CS in $COURSES; do
    echo $CS
    break
done

# 恢复
rht-clearcourse 0
```

```

rht-setcourse $CS

source /etc/rht

# Main Area
for i in $RHT_VM0 $RHT_VMS; do
    ## kvm xml
    case $i in
        classroom)
            XML_FILE=/var/lib/libvirt/images/$RHT_COURSE-$i.xml
            ;;
        *)
            XML_FILE=/content/$RHT_VMTREE/vms/$RHT_COURSE-$i.xml
            ;;
    esac

    ## xml_modify
    ## cpu, secboot, features
    cat > /tmp/kvm_all.xml <<EOF
/cpu.*mode/s+host-model+custom+
/cpu.*mode/s+check+match='exact' check+
/cpu.*mode/s+/++
/cpu.*mode/a\                <model fallback='allow'>Westmere</model>\n\
        </cpu>
/<\/os/i\                <loader readonly='yes' secure='yes'
type='pflash'>/usr/share/OVMF/OVMF_CODE.secboot.fd</loader>
/<\/features/i\                <smm state='on'/>"
EOF

    ## apply
    if grep -wq host-model $XML_FILE; then
        ssh root@localhost \
            "sed -i.bk -f /tmp/kvm_all.xml $XML_FILE"
    fi
done
```

- amd cpu

```
# foundation
cat >> /etc/modprobe.d/kvm.conf <<EOF
options kvm_amd nested=1
options kvm ignore_msrs=1
EOF

# 立即生效
echo 1 > /sys/module/kvm/parameters/ignore_msrs
```

A16. docker

```
# docker pull wordpress

# docker save -o wordpress.tar wordpress

# file wordpress.tar

# tar -tf wordpress.tar
```

A17. registry

```
[root@master ~]# oc get route
NAME                HOST/PORT                                        PATH      SERVICES
    PORT      TERMINATION   WILDCARD
docker-registry    `docker-registry-default.apps.lab.example.com`             docker-
registry    <all>      passthrough   None
registry-console   `registry-console-default.apps.lab.example.com`            registry-
console    <all>      passthrough   None
```

https://registry-console-default.apps.lab.example.com

Username: admin
Password: redhat

A18. dns

[kiosk@foudantion]

```
nmcli con mod "Bridge br0" ipv4.dns 172.25.250.254

nmcli con mod "Bridge br0" +ipv4.dns 172.25.254.250

nmcli con up "Bridge br0"

```