

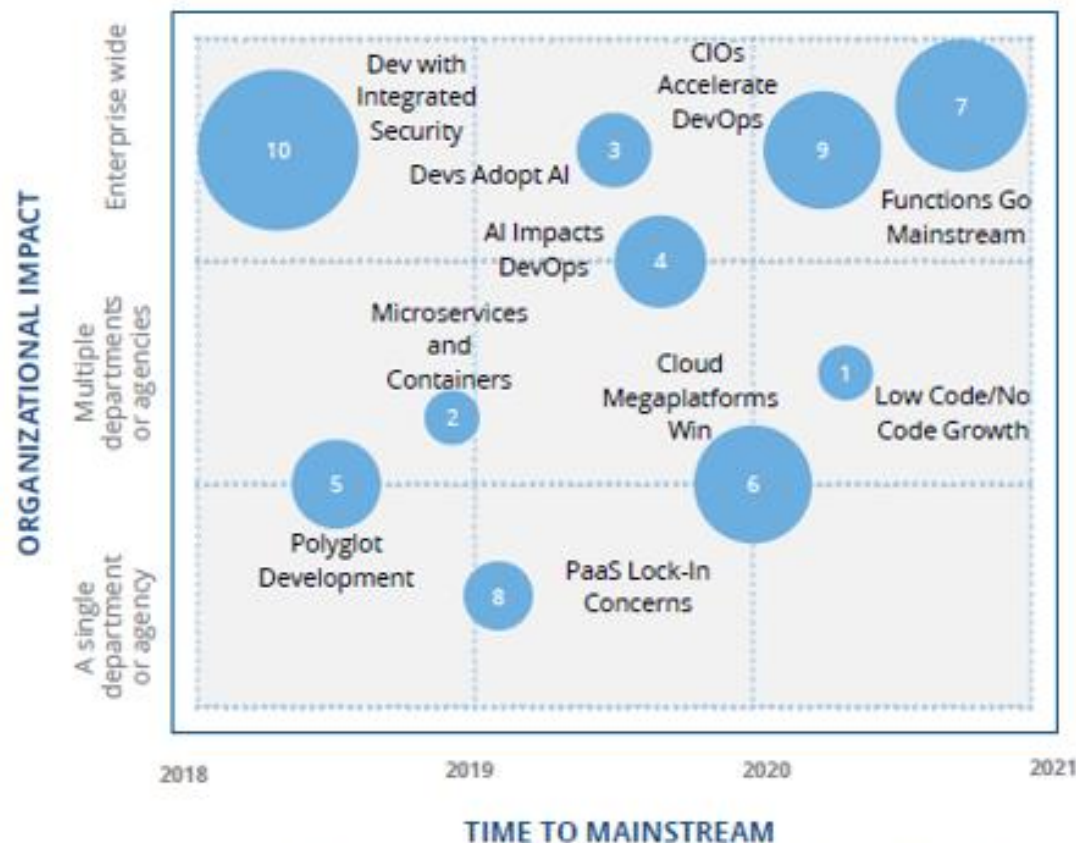


基于Red Hat OpenShift 4

构建PaaS、DevOps平台

魏新宇 红帽资深解决方案架构师

IDC: DevOps、微服务、PaaS成为IT技术发展的新趋势



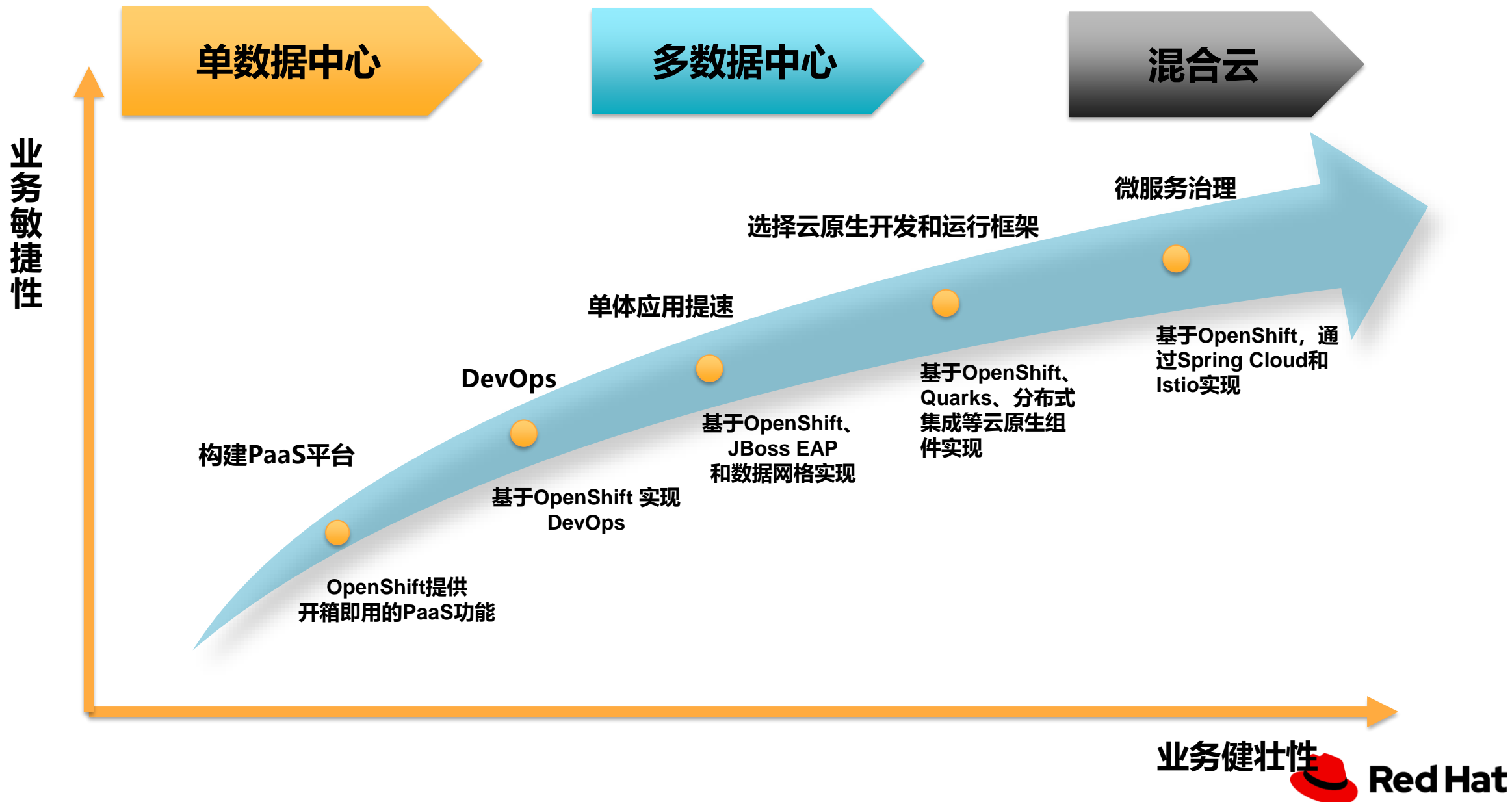
2019年，超过**70%**的日常开发和生命周期任务将实现自动化，并通过现有数据流提供AI支持，通过灵活的

DevOps管道驱动和孵化 生命周期和应用开发的智能化

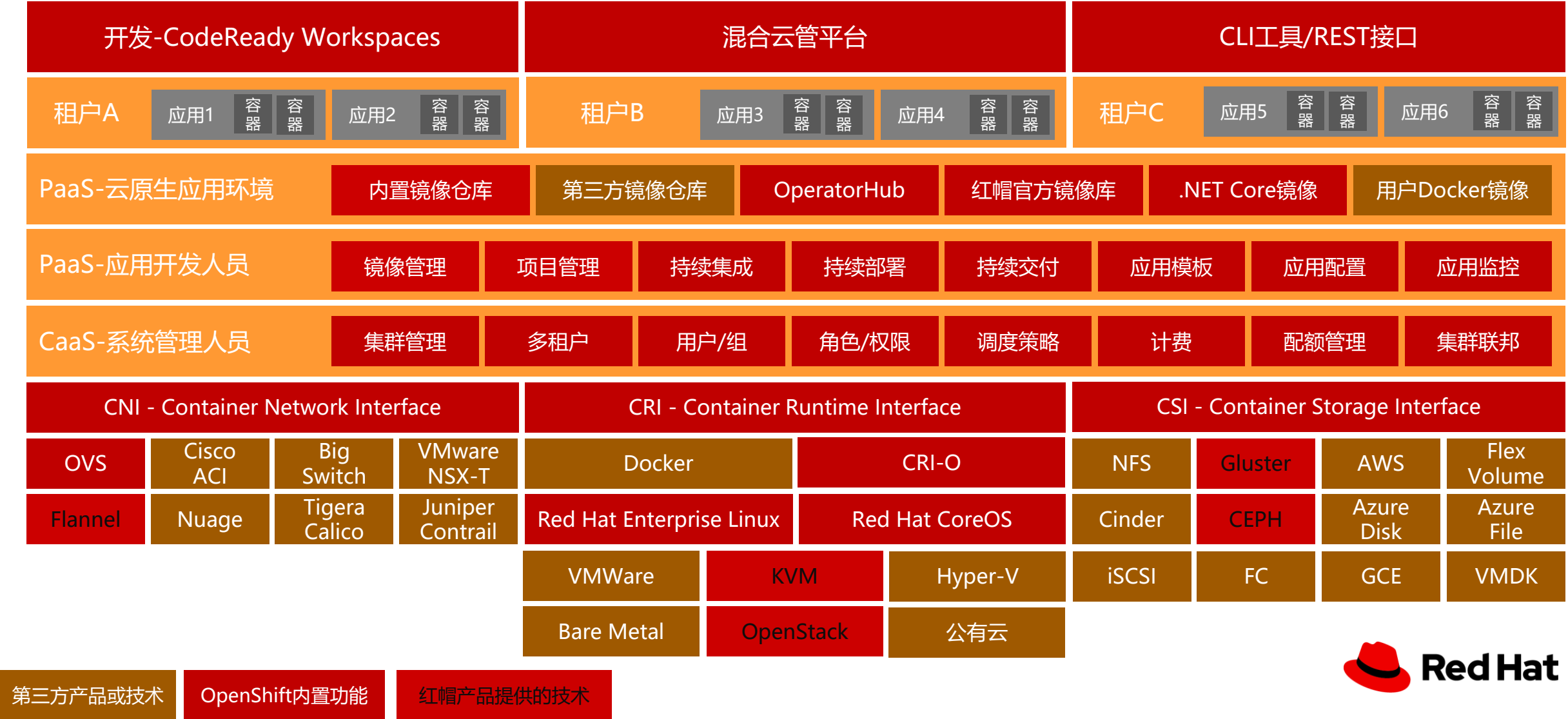
2021年，**80%**的企业应用将部署在**PaaS**平台上，并使用**微服务架构**；其中，超过**95%**的新的微服务将部署在**容器**中

2021年，超过**50%**的首席信息官将任命交付负责人；整合他们的开发、项目管理和运营团队；减少应用竖井；扩展他们的**DevOps**实践；并实施了迁移测试以加速创新

敏态IT构建之路



红帽基于 OpenShift 的容器云生态



OpenShift - 企业级PaaS技术栈架构

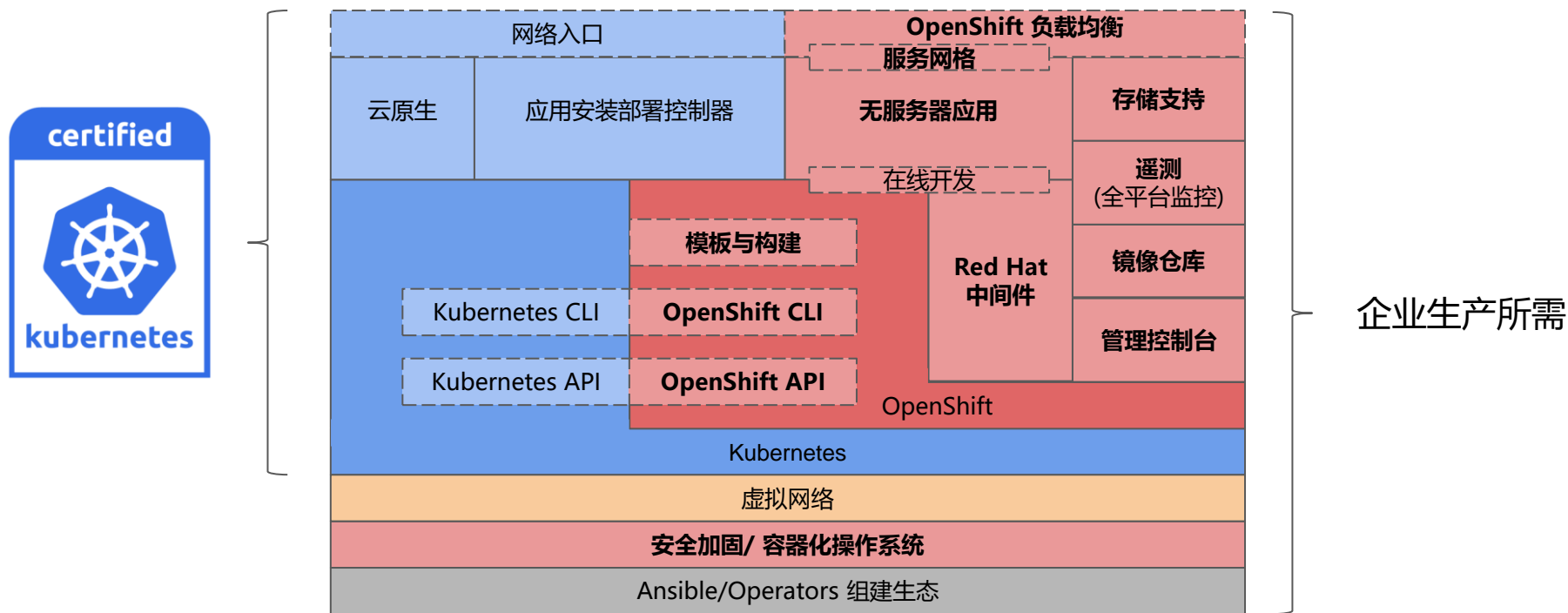


红帽容器平台对于K8S之上增强了哪些？

- 提供**多集群能力**，通过Web控制台动态管理多集群及节点，提供节点自动扩展能力
- **全栈自动安装**，简化安装过程，实现自动更新和升级
- 通过**Operator框架**，完善容器化应用的生命周期管理
- 提供了**统一的产品界面展现**，界面更漂亮
- 通过MULTUS提供支持**多网络平面**
- 通过容器原生虚拟化(CNV) 提供**虚拟机管理**
- 通过(Service Mesh: Istio)提供了**微服务框架**
- 通过Serverless (Knative) 提供最新的**Serverless能力**
- 通过CodeReady Workspaces (Che) 提供**云端开发能力**
- 全新的应用拓扑控制台，提供**以业务应用为中心的统一视图**

红帽容器平台对于K8S之上扩展

OpenShift是全球第一的商业开源PaaS平台



OpenShift让容器安全一目了然

jboss-webserver-3/webserver31-tomcat8-openshift

JBoss Web Server 3.1 - Tomcat 8 ›

Architecture

AMD64

Tag

1.4-6

以前镜像版本

⚠ Updated image available

by Red Hat, Inc. | in Product Red Hat JBoss Web Server

⚠ Updated image available. Red Hat strongly recommends updating to the newest image version 1.4-7 unless otherwise defined by the support policy of Red Hat JBoss Web Server.

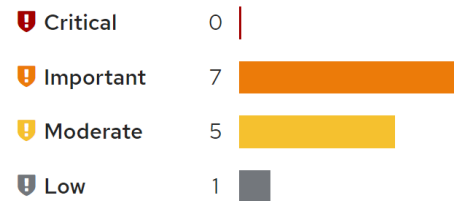
安全级别

Health Index ⓘ



This image is affected by Critical (no older than 90 days) or Important (no older than 12 months) security updates. The Container Health Index analysis is based on RPM packages signed and created by Red Hat, and does not grade other software that may be included in a container image.

13 security vulnerabilities affecting 10 packages



安全风险项目

Impact	Affected Package
Important CVE-2019-9636	python-libs-2.7.5-76.el7.x86_64 python-2.7.5-76.el7.x86_64
Important CVE-2019-6454	systemd-219-62.el7_6.2.x86_64 systemd-libs-219-62.el7_6.2.x86_64
Important CVE-2019-3863	libssh2-1.4.3-12.el7.x86_64
Important CVE-2019-3857	libssh2-1.4.3-12.el7.x86_64
Important CVE-2019-3856	libssh2-1.4.3-12.el7.x86_64
Important CVE-2019-3855	libssh2-1.4.3-12.el7.x86_64

风险

jboss-webserver-3/webserver31-tomcat8-openshift

JBoss Web Server 3.1 - Tomcat 8 ›

Architecture

AMD64

Tag

1.4-7

最新镜像版本

by Red Hat, Inc. | in Product Red Hat JBoss Web Server

Latest This is the newest image version

安全级别

Health Index ⓘ



This image does not have any unapplied Critical or Important security updates.

The Container Health Index analysis is based on RPM packages signed and created by Red Hat, and does not grade other software that may be included in a container image.



容器化与容器平台的应用准入条件

1. 已建立了清晰的可自动化的编译及构建流程

应用使用了如Maven、Gradle、Make

2. 已实现应用配置参数外部化

应用已将配置参数外部化与配置文件或环境变量中，

3. 已提供合理可靠的健康检查接口

容器平台将通过健康检查接口判断容器状态，对应用服务

4. 已实现状态外部化，实现应用实例无状态化

应用状态信息存储于数据库或缓存等外部系统，应用实例本身实现

5. 不涉及底层的操作系统依赖及复杂的网络通信机制

应用以处理业务为主，不强依赖于底层操作系统及组播等网络通信

6. 部署交付件及运行平台的大小在2GB以内

轻量级的应用便于在大规模集群中快速传输分发，更符合容器敏捷的

7. 启动时间在5分钟以内

过长的启动时间将不能发挥容器敏捷的特性。



应用容器化的方法

名称	举例	类别	适用场景	特点
本地构建	工程师编写 Dockerfile ，并手工在一台或多台主机上手工执行 Docker build 命令构建应用镜像	人工	开发测试	<ul style="list-style-type: none">● 简单● 每个开发团队均需要编写Dockerfile● 重复多次执行时间和人力成本陡增
CI构建	Jenkins 集群在 CI 流程中调用 Maven 执行构建， Maven 通过插件按指定的 Dockerfile 生成应用的容器镜像。	自动化	传统持续集成	<ul style="list-style-type: none">● 常规CI方案● 每个开发团队均需要编写Dockerfile● 微服务多语言多版本混合构建时无隔离● 构建资源池不可动态扩展● 资源利用率较低
OpenShift Source 2 Image (S2I)	OpenShift 在隔离的容器环境中进行应用的构建编译并生成应用的容器镜像。	自动化	容器及微服务场景下的持续集成	<ul style="list-style-type: none">● 基于容器集群的CI方案● 开发团队通过容器镜像精确定义构建环境● 基于容器的构建环境，提供更好的隔离性● 满足多语言多版本微服务混合构建的场景● 构建资源池可动态扩展，更灵活● 构建与应用运行共享资源池，介绍运维工作量● 资源按需投入及回收，利用率较高

灵活
高效

Dockerfile方式

```
FROM fabric8/java-jboss-openjdk8-jdk:1.5.2

ENV JAVA_APP_DIR=/deployments

ENV JAEGER_SERVICE_NAME=customer

    JAEGER_ENDPOINT=http://jaeger-collector.istio-
system.svc:14268/api/traces

    JAEGER_PROPAGATION=b3

    JAEGER_SAMPLER_TYPE=const

    JAEGER_SAMPLER_PARAM=1

EXPOSE 8080 8778 9779

COPY target/customer.jar /deployments/
```

Import from Dockerfile

Git

Git Repo URL *

[Show Advanced Git Options](#)

Dockerfile

Dockerfile Path

Dockerfile

Allows the builds to use a different path to locate your Dockerfile, relative to the Context Dir field.

Container Port

8080

Port number the container exposes.

Resources

Select the resource type to generate

☒ Deployment

apps/Deployment

A Deployment enables declarative updates for Pods and ReplicaSets.

☐ Deployment Config

apps.openshift.io/DeploymentConfig

A Deployment Config defines the template for a pod and manages deploying new images or configuration changes

☐ Knative Service

Tech Preview

serving.knative.dev/Service

A Knative Service enables scaling to zero when idle

Advanced Options

☒ Create a route to the application

Exposes your application at a public URL

Source to Image 架构

Overview

▼ Source-to-Image (S2I)

Overview

Java

.NET Core

Node.js

Perl

PHP

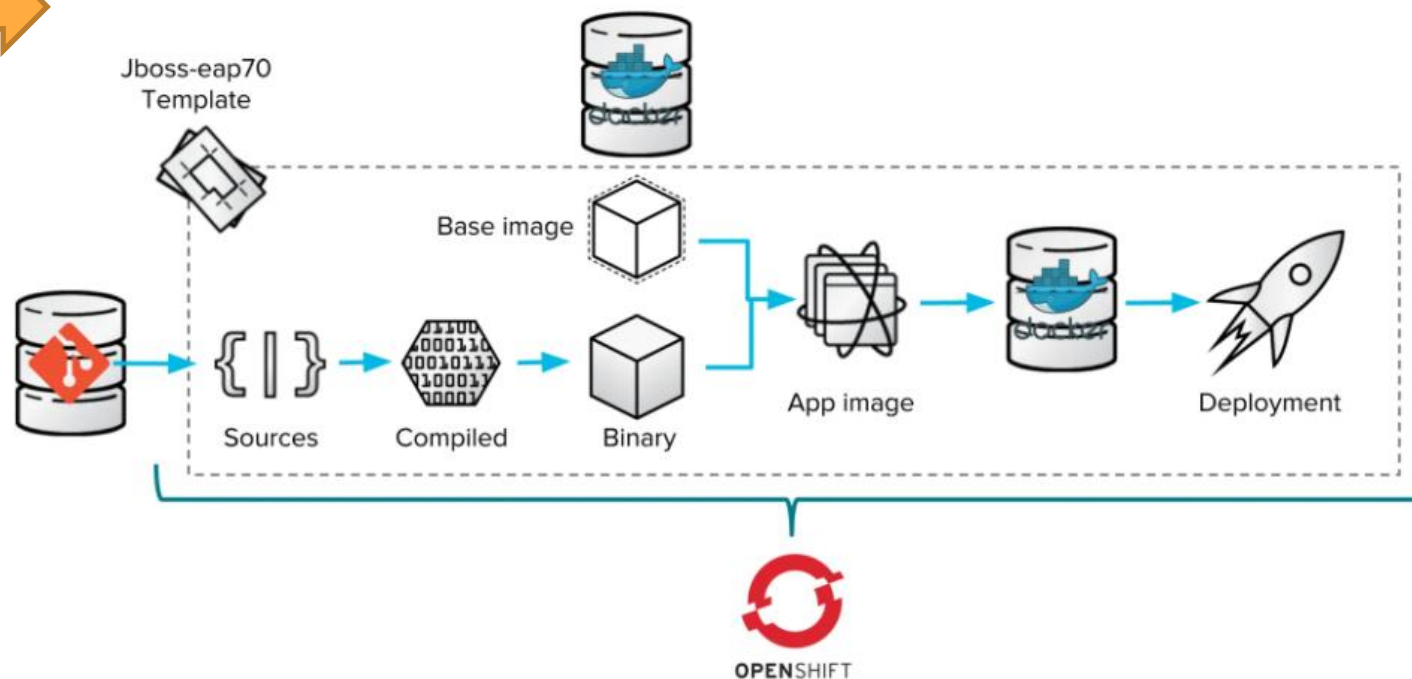
Python

Ruby

Customizing S2I Images

► Database Images

► Other Images



S2I流程与脚本

S2I的流程

- 启动构建之后，也就是docker-builder容器启动，首先实例化基于S2I Builder的容器。从Git上获取应用源代码，创建一个包含S2I scripts（如果Git上没有，则使用Builder Image中的脚本）和应用源码的tar文件，将tar文件传入到S2I Builder实例化的容器中。
- tar文件被解压缩到S2I builder容器中io.openshift.s2i.destination标签指定的目录位置，默认是/tmp。
- 如果是增量构建，assemble脚本会先恢复被save-artifacts脚本保存build artifacts。
- assemble脚本从源代码构建应用程序，并将生成的二进制文件放入应用运行的目录。
- 如果是增量构建，执行save-artifacts脚本并保存所有构建以来的artifacts到tar文件。
- assemble脚本执行完成以后，生成最终应用镜像。
- docker-builder容器调用docker命令，将构建好的应用镜像推送到内部镜像仓库。
- 应用镜像推送到内部镜像仓库之后，触发DeploymentConfig中的Image Stream触发器自动部署应用镜像。
- 应用镜像运行，并执行RUN脚本配置应用参数以及启动应用。

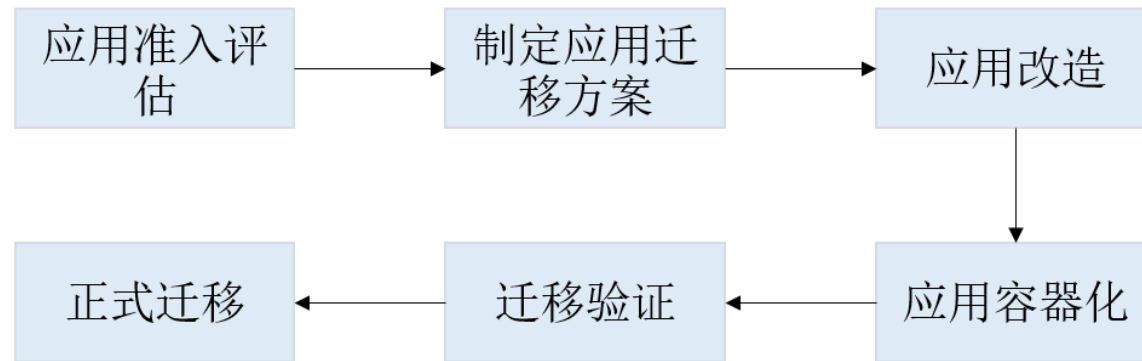
S2I的几个脚本

- assemble：负责将已经下载到本地的外部代码进行编译打包，然后将打包好的应用包拷贝到镜像对应的运行目录中。
- run：负责启动运行assemble编译、拷贝好的应用。
- usage：告诉使用者如果使用该镜像。
- save-artifacts：脚本负责将构建所需要的所有依赖包收集到一个tar文件中。save-artifacts的好处是可以加速构建的过程。
- test/run：通过 test/run 脚本，您可以创建简单的流程来验证镜像是否正常工作。

文件↵	构建语言↵	编程语言↵
Dockerfile↵	无↵	Dockerfile 构建（非 S2I）↵
pom.xml↵	jee↵	Java（使用 JBoss EAP）↵
app.json、package.json↵	nodejs↵	Node.js (JavaScript)↵
composer.json、index.php↵	php↵	PHP↵

应用容器化迁移流程

- **应用准入评估**：根据制定的应用准入评估准则对要迁移的应用或系统进行评估，如果满足运行在容器上的准入要求，则进行应用迁移方案的制定。
- **制定应用迁移方案**：在应用迁移方案制定中，需要综合考虑应用使用技术语言、通信协议、中间件版本、配置传入的方式、日志如何输出、应用灰度发布等等应用或系统的技术实现细节，并结合OpenShift的特性以及约束制定迁移方案，期间可能需要进行必要的技术验证。
- **应用改造**：待应用迁移方案确定并得到认可之后，可能需要对应用进行必要的改造，以最佳的形式在OpenShift上运行，如日志的输出的形式，配置外部化等。
- **应用容器化**：应用容器化指将应用改造或打包为可以容器形式运行的过程。应用容器化通常包括如下几个方面：基础镜像制作、应用容器化构建、其他技术组件容器化。
- **迁移验证和正式迁移**：在应用容器化完成之后，就可以进行迁移验证，如果过程中，出现问题可能需要随时调整，最终达到符合预期的效果就可以正式迁移了。

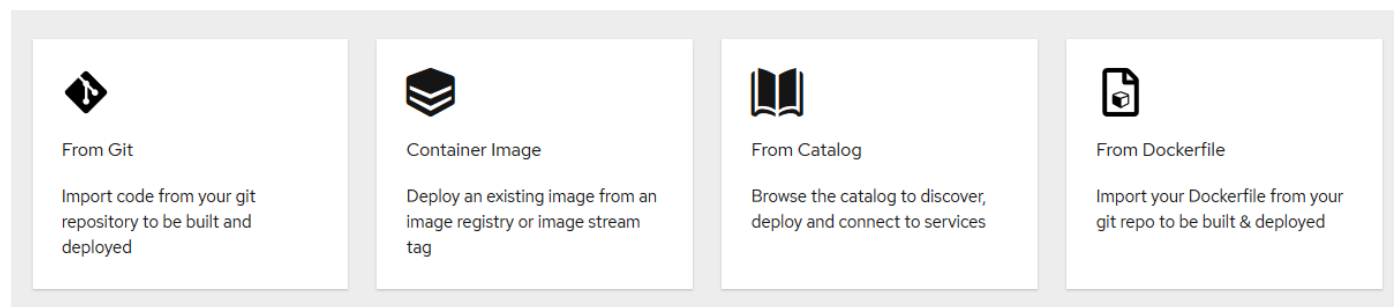


OpenShift应用主要部署方法

- 基于dockerfile方式：可以实现应用的容器化和部署。
- 基于ContainerImage方式：可以实现对容器化完后的应用进行部署。
- 基于S2I：可以实现应用的容器化和部署。
- 基于模板部署：模板部署方式很灵活，既可以部署现有的应用容器镜像，也可以调用S2I将应用容器化和部署一起完成。
- 基于Operator部署：实现对容器化应用的全生命周期管理。

Add

Select a way to create an application, component or service from one of the options.


















模板方式

Select from Project

Selection Information Configuration Results

1 2 3 4

 JBoss Web Server 3.1 Apache Tomcat 8 + PostgreSQL (with...	 Jenkins	 Jenkins (Ephemeral)	 MariaDB	 MariaDB (Ephemeral)
 MongoDB	 MongoDB (Ephemeral)	 MySQL	 MySQL (Ephemeral)	 Nginx HTTP server and a reverse proxy
 Node.js + MongoDB	 Node.js + MongoDB (Ephemeral)	 OpenJDK 8	 PostgreSQL	 PostgreSQL (Ephemeral)

<application-name>-<project>.<default-domain-suffix>

Git Repository URL *

https://github.com/jboss-openshift/openshift-quickstarts.git

Git source URI for application

Git Reference

1.2

Git branch/tag reference

Context Directory

tomcat-websocket-chat

Path within Git project to build; empty for root project directory.

JWS Admin Username

(generated if empty)

JWS Admin User

JWS Admin Password

(generated if empty)

JWS Admin Password

Github Webhook Secret

(generated if empty)

GitHub trigger secret

OperatorHub:容器应用商店



Red Hat

OperatorHub.io



Google Cloud

Welcome to OperatorHub.io

OperatorHub.io is a new home for the Kubernetes community to share Operators. Find an existing Operator or list your own today.

CATEGORIES 33 ITEMS VIEW SORT A-Z

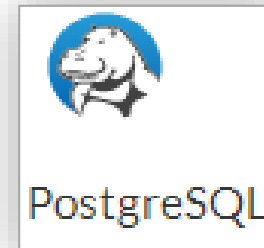
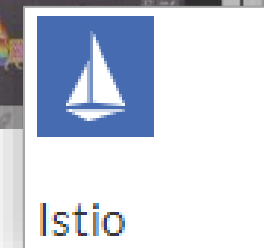
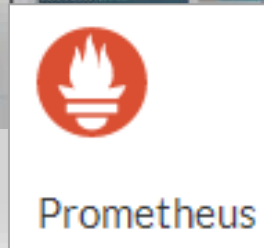
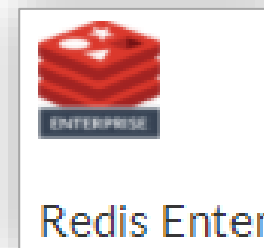
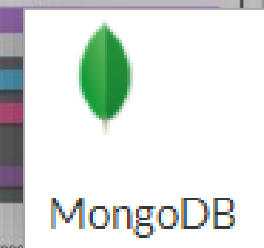
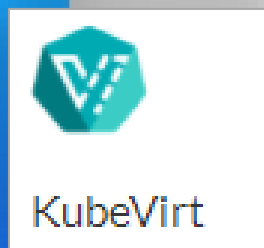
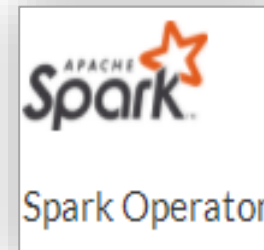
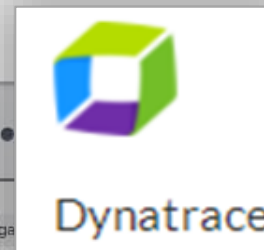
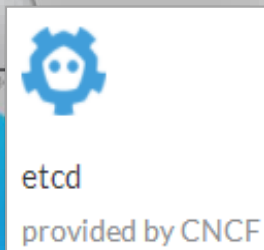
AI/Machine Learning	Aqua Security Operator provided by Aqua Security, Inc.	AWS Service Operator provided by Amazon Web Services, Inc.	CockroachDB provided by Helm Community	Community Jaeger Operator provided by CNCF	Couchbase Operator provided by Couchbase	Crunchy PostgreSQL Enterprise provided by Crunchy Data
Big Data	Dynatrace OneAgent provided by Dynatrace LLC	etcd provided by CNCF	Falco Operator provided by Sysdig	Federation provided by Red Hat	FederatorAI provided by ProphetStor Data Services, Inc.	Hazelcast Operator provided by Hazelcast, Inc.
Cloud Provider	infinispan provided by Infinispan	Instana Agent Operator provided by Instana	Istio provided by Banzai Cloud	Kiali Operator provided by Kiali	KubeVirt provided by KubeVirt project	Microcks Operator provided by Microcks.io
Database	MongoDB provided by MongoDB, Inc.	MyVirtualDirectory provided by Tremolo Security, Inc.	Opsmx Spinnaker Operator provided by OpsMx	Percona Xtradb Cluster Operator provided by Percona	PlanetScale Operator for Vitess provided by PlanetScale	Prometheus Operator provided by Red Hat
Integration & Delivery	Redis Enterprise provided by Redis Labs, Inc.	Robin Storage provided by Robin.io	Spark Operator provided by GoogleCloudPlatform	Spinnaker Operator provided by OpsMx	StorageOS provided by StorageOS, Inc.	Strimzi Kafka provided by Red Hat
Logging & Tracing						
Monitoring						
OpenShift Optional						
Security						
Storage						
Streaming & Messaging						

PROVIDER

- Amazon Web Services(1)
- Aqua Security(1)
- Banzai Cloud(2)
- CNCF(2)
- Couchbase(1)
- Show 22 more

CAPABILITY LEVEL

- Basic Install(15)
- Seamless Upgrades(4)
- Full Lifecycle(13)
- Deep Insights(1)



Red Hat

Operator方式

[Installed Operators](#) > [Operator Details](#)



ActiveMQ Artemis
0.9.2 provided by Red Hat, Inc.

[Details](#) [YAML](#) [Subscription](#) [Events](#) [All Instances](#) [ActiveMQ Artemis](#) [ActiveMQ Artemis Address](#) [ActiveMQ Artemis Scaledown](#)

Provided APIs

ActiveMQ Artemis

An instance of Active MQ Artemis

[+ Create Instance](#)

ActiveMQ Artemis Address

Adding and removing addresses via custom resource definitions

[+ Create Instance](#)

ActiveMQ Artemis Scaledown

Provides message migration on clustered broker scaledown

[+ Create Instance](#)

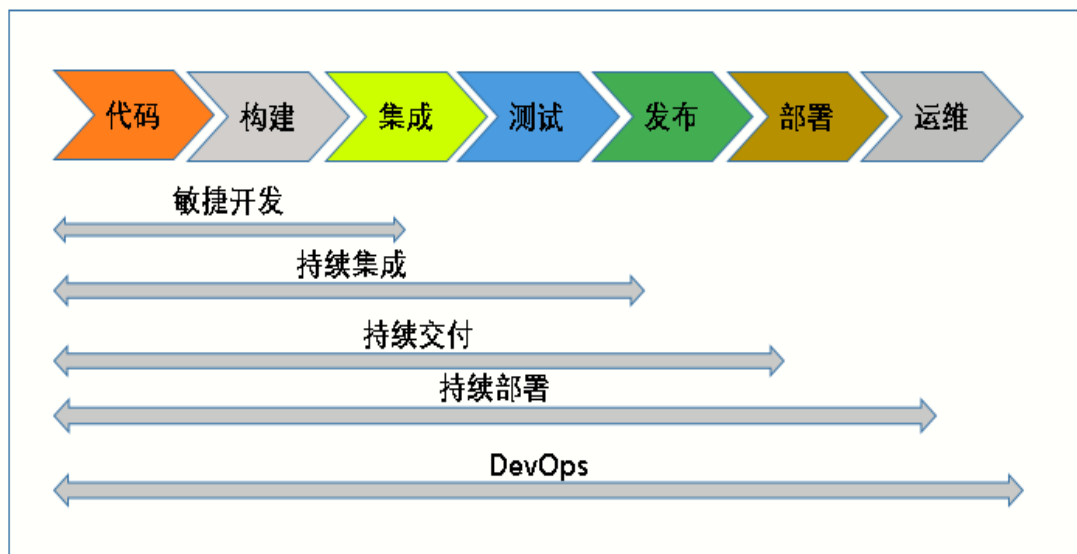
Description

ActiveMQ Artemis Operator provides the ability to deploy and manage stateful ActiveMQ Artemis broker clusters

应用部署方式对比

部署方法	优点	缺点	适用场景
Dockerfile	可实现应用容器化，部署简单	对于定制化的容器镜像，我们可能需要手工调整OpenShift自动生成的Deployment/DC。当应用的源码经常发生变化、部署方式较为复杂时，这种方法的工作量和难度较大	小规模开发测试环境
容器镜像	部署方法简单	只能部署已经容器化完毕的应用	小规模运维环境
S2I	可以实现从源码到最终容器化应用的部署。这种方法了解决了dockerfile效率低、复杂度高的问题。红帽官网提供大量基于OpenShift的builder image，红帽订阅客户可以直接使用，也可以在现有builder image上构建自己的builder image。如果应用源码经常发生变化，使用S2I的方式最为适合。	技术人员需要掌握S2I的原理	适合开发、测试环境。
模板部署	模板既可以调用S2I实现用源码构建，也可以只调用deployments部署已有的容器化应用。模板的优势是通过UI做参数传递。	定制模板需要一定工作量。此外应用的升级、扩容的操作通过模板无法完成。	适合构建容器应用发布平台。
Operator的部署	红帽OperatorHub上有上百种应用，Operator也是目前K8S社区主流的容器化应用开发（Operator支持go、Ansible、Helm的开发方式）、部署模式。红帽OpenShift自身的能力也通过Operator实现。目前，通过Operator部署的应用，版本升级和扩容非常简便、做到了可视化。Operator部署方式适合企业大规模容器化生产环境，便于运维部门对容器化应用的运维。	Oprator主要是实现有状态应用全生命周期管理，主要通过CR进行配置。无法对客户现有业务实现容器化。	适合企业大规模容器化生产环境，便于运维部门对容器化应用的运维

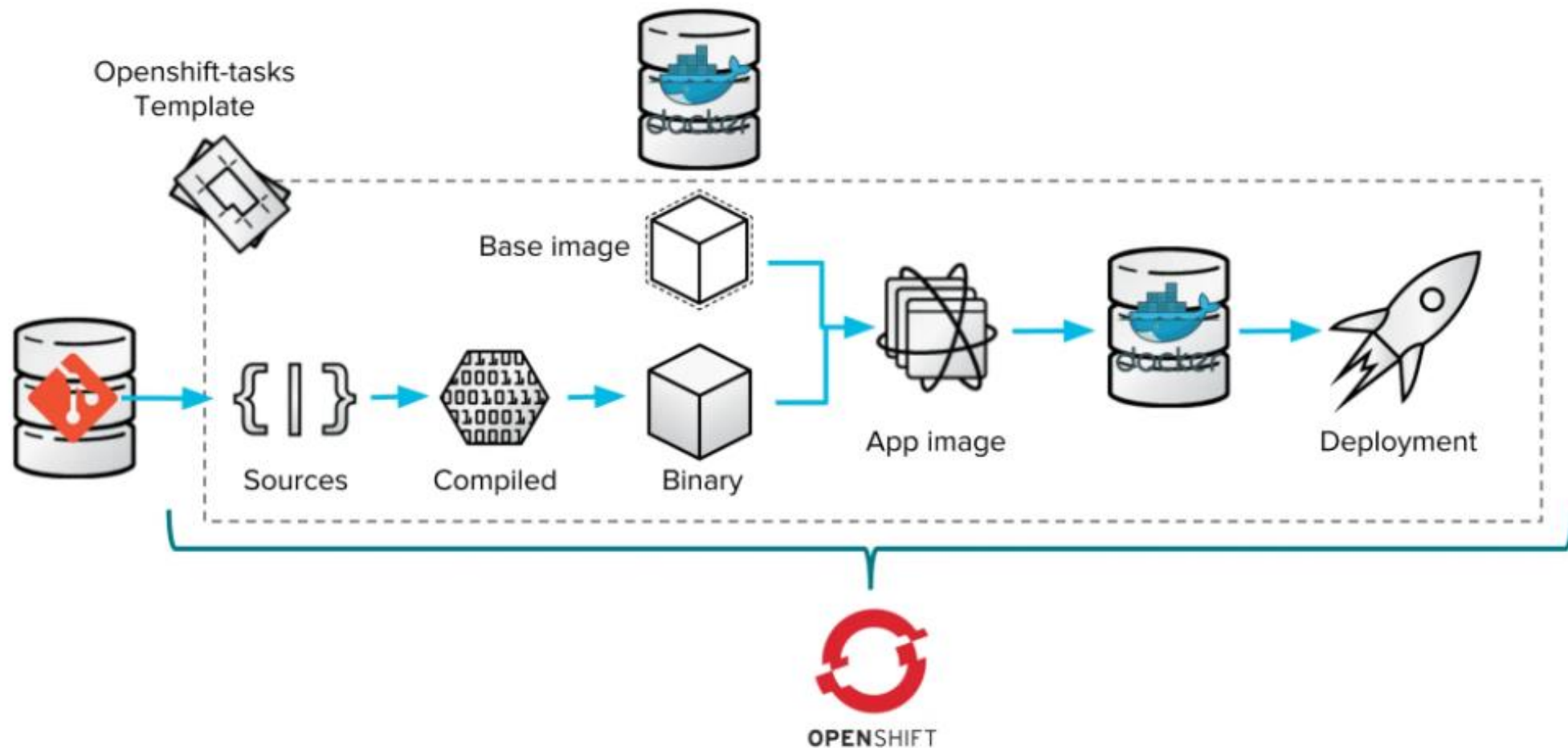
OpenShift平台CI/CD实现方式



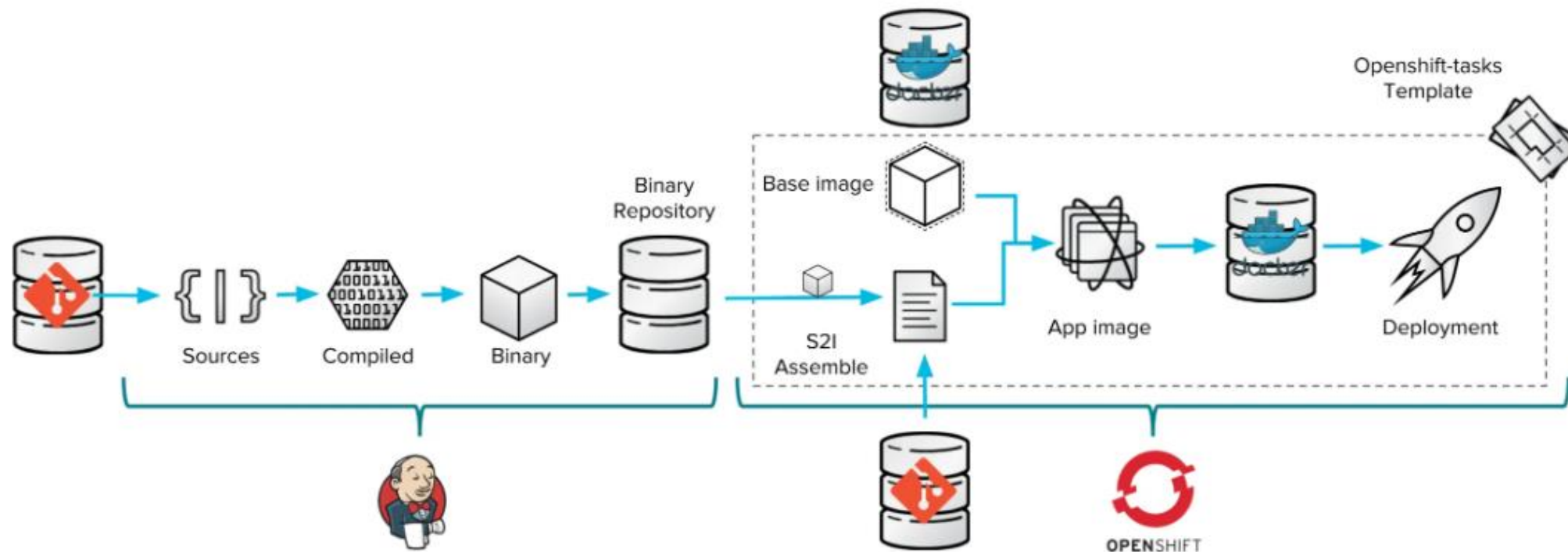
- Jenkins 负责 CI、S2I 负责 CD
- Jenkins 管理 Pipeline, 调度 OpenShift S2I 完成 CI/CD
- 在 OpenShift 上使用 Jenkins file, 调用 Jenkins 实现 CI、OpenShift 负责 CD。我们称这种方式为 OpenShift Pipeline
- 通过Teckton实现

CI/CD方式1:

通过二进制文件基于S2I构建应用镜像

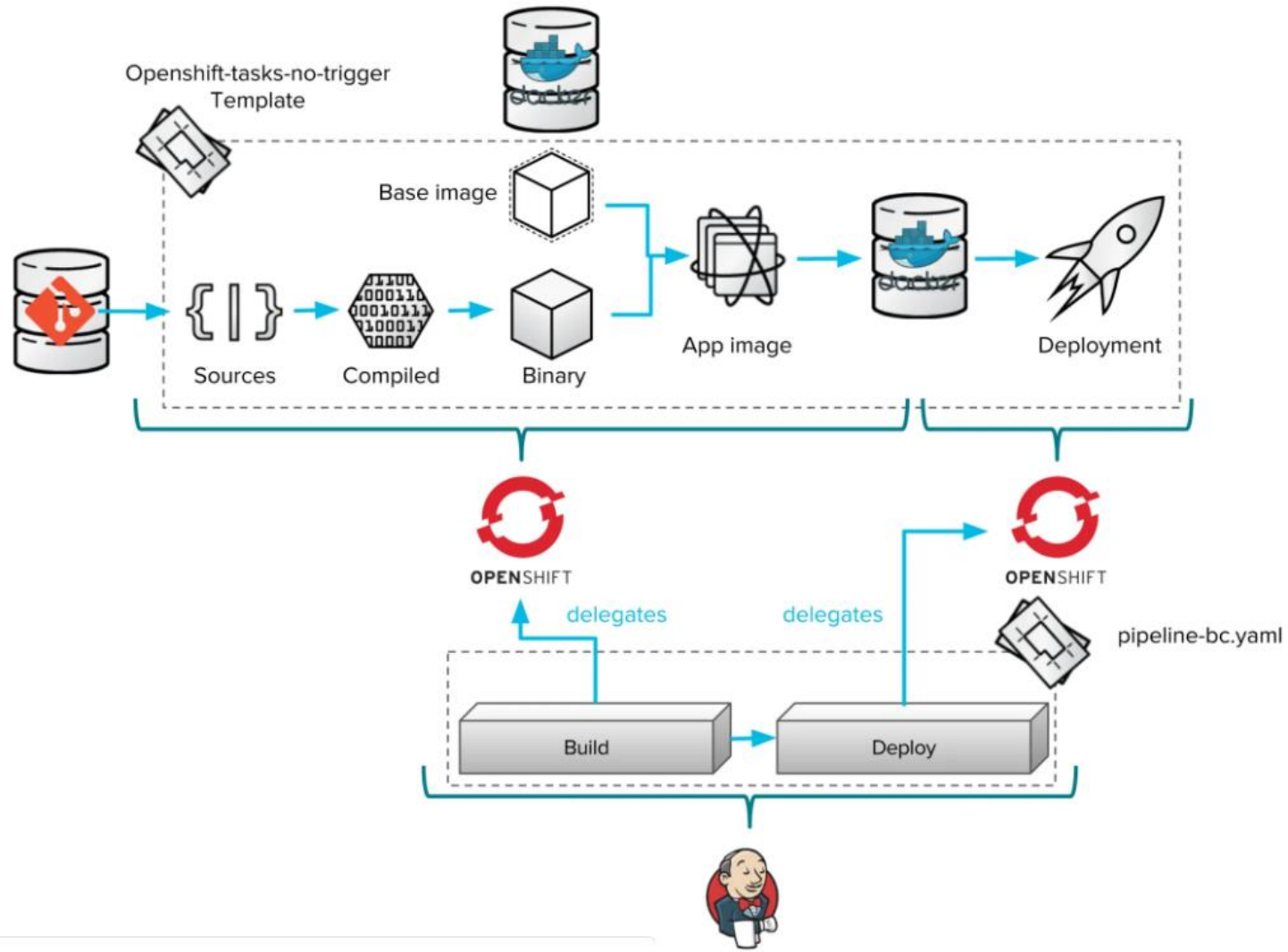


CI/CD方式2: 在源码外构建pipeline部署应用



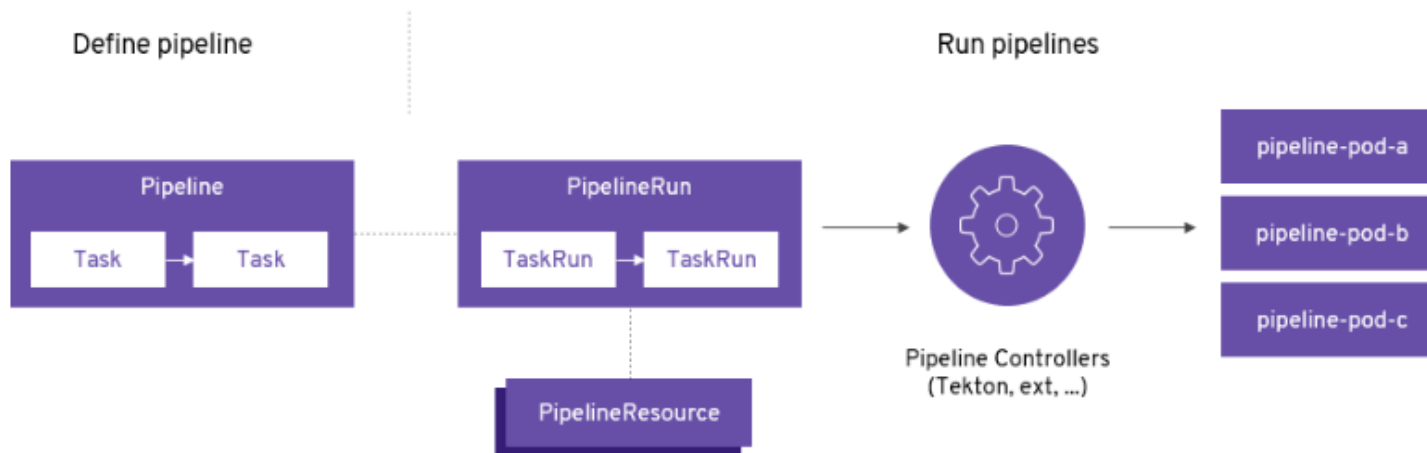
CI/CD方式3:

在源码内构建pipeline部署应用



CI/CD方式4: 通过Tekton

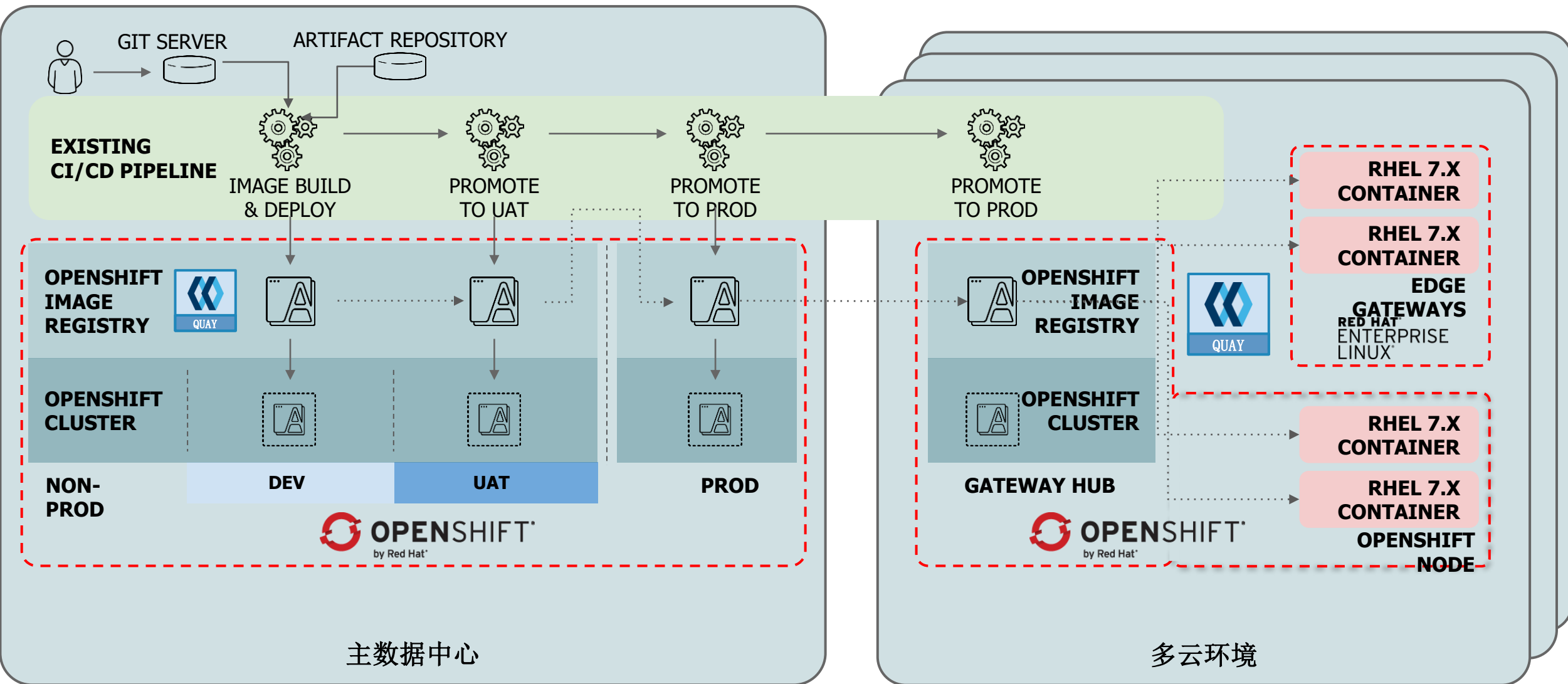
Tekton 是由谷歌主导的开源项目，它是一个功能强大且灵活的 Kubernetes 原生开源框架，用于创建持续集成和交付（CI/CD）。通过抽象底层实现细节，用户可以跨多云平台和本地系统进行构建、测试和部署。Tekton 将许多 Kubernetes 自定义资源（CR）定义为构建块，这些自定义资源是 Kubernetes 的扩展，允许用户使用 kubectl 和其他 Kubernetes 工具创建这些对象并与之交互。



DevOps架构示例



构建跨多云的CI/CD 应用发布



OpenShift对项目的改进与收益

使用前	使用后
中间件环境单点物理机部署	基于基础镜像和模板统一、快速部署容器化中间件集群
JDK、应用服务器版本不统一，各自为阵	基于标准镜像统一版本，有效控制运行环境的合规性
人工物理环境发布，十多步打包发布流程/命令	一键点击，部署流程自动化、可视化，关键节点人工确认，极大提高了开发、运维人员的生产效率
应用部署时间：30分钟+	应用部署时间：5分钟-
人工维护历史war/jar包，人工回退	版本可追溯（回退），减少人为操作可能带来的失误
人工维护不同环境的配置文件，发布不同环境需要重新编译	git 统一管理，启动时替换配置文件以适配不同环境，确保基准镜像一致性

企业对微服务治理的需求

功能列表	描述
服务注册与发现	在平台软件部署服务时，能自动进行服务的注册，其他调用方可以即时获取新服务的注册信息。
配置中心	可以管理微服务的配置
支持命名空间、项目名称配置	基于NameSpace隔离微服务
微服务间路由管理	实现微服务之间相互访问
支持负载均衡	客户端发起请求在微服务端的负载均衡
日志收集	收集微服务的日志
内部API网关	为所有客户端请求的入口
微服务链路可视化	可以生成微服务之间调度的拓扑关系图
无源码修改方式的应用迁移	将应用迁移到微服务架构时不修改应用源代码
灰度/蓝绿发布	实现应用版本的动态切换
灰度上线	允许将实时流量进行副本，客户无感知。

安全策略	实现微服务访问控制的RBAC，对于微服务入口流量可设置加密访问。
性能监控	监控微服务的实施性能
支持混沌测试	模拟各种微服务的故障
支持服务间调用限流、熔断	避免微服务出现雪崩效应
实现微服务见的访问控制黑白名单	灵活设置微服务之间的相互访问策略
支持服务间路由控制	灵活设置微服务之间的相互访问策略
支持对外部应用的访问	微服务内的应用可以访问微服务体系外的应用
支持链路追踪	实时追踪微服务之间访问的链路，包括流量、成功率等。
支持应用自拓扑	实时展示微服务之间的调用关系
服务追踪	展示微服之间调用，已经调用的层级关系

Spring Cloud优缺点

优点

- 可定制强，可以通过精致满足各种业务场景
- 基于springboot的统一编程模型，有快速创建应用的能力
- 丰富的SDK和类库支持，覆盖大多数运行时需求
- 配置灵活，控制能力强
- 注解驱动，spring cloud应用兼容性好

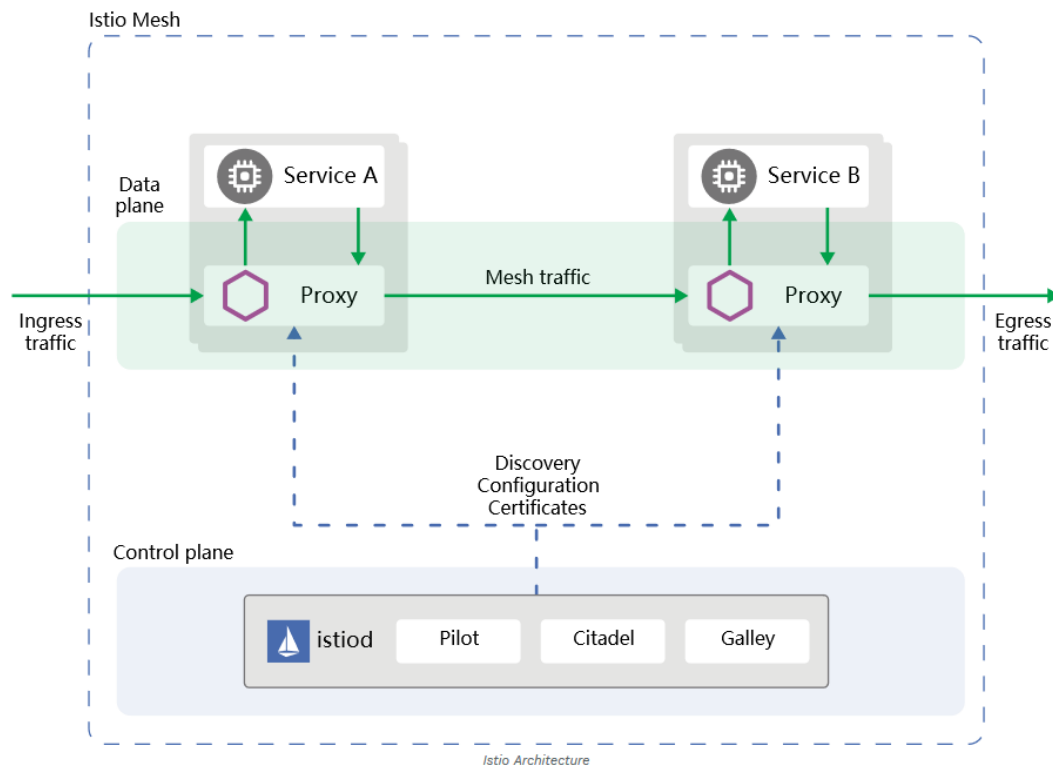
缺点

- 只适用于Java技术栈，不支持跨语言的能力
- 使用门槛较高，开发人员需处理大量与业务无关的逻辑
- 覆盖面有限，缺少完整的微服务生命周期支持，比如自动部署、调度、资源管理、进程隔离、故障恢复等。

Spring Cloud在OpenShift上的实现



Istio架构



- Pilot是流量管理的核心组件，在Istio中承担的主要职责是向Envoy proxy提供服务发现，以及为高级路由（如：A / B测试、金丝雀部署等）提供流量管理功能和异常控制（如：超时、重试、断路器等）。
- Citadel：用于密钥管理和证书管理，下发到Envoy等负责通信转发的组件。
- Galley在Istio中，承担配置的导入、处理和分发任务，为Istio提供了配置管理服务，提供在Kubernetes服务端验证Istio的CRD 资源的合法性。

Istio的运维建议

Istio运维方面建议包括：版本选择、备用环境准备、评估范围、配置生效、功能健壮性参考、入口流量选择。当然，这些建议只是基于目前我们在测试过程中得到的数据。后续随着Istio使用越来越广泛，相信最佳实践将会越来越丰富。

版本选择

Istio是一个迭代很快的开源项目。截止到2019年6月，社区最新的Istio版本为1.2。频繁的版本迭代，会给企业带来一些困扰：是坚持适应目前已经测试过的版本，还是使用社区的最新版本？在前文中我们已经提到，红帽针对Istio有自己的企业版，通过Operator进行部署和管理。出于安全性和稳定性的考虑，红帽Istio往往比社区要晚两个小版本左右。因此建议使用红帽Istio的最新版本。目前看，社区的最新版本的Istio稳定性往往不尽如人意。

备用环境

针对于相同的应用，在OpenShift环境中部署一套不被Istio管理的环境。比如我们文中的三层微服务，独立启动一套不被Istio管理的应用，使用OpenShift原本的访问方式即可。这样做的好处是，每当进行Istio升级或者是部分参数调整时都可以提前进行主备切换，让流量切换到没有被Istio管理的环境中。Istio升级调整验证完毕后再将流量切换回来。

评估范围

由于Istio对微服务的管理是非代码侵入式的。因此通常情况下，业务服务需要进行微服务治理，需要被Istio纳管。而对于没有微服务治理要求的非业务容器，不必强行纳管在Istio中。当非业务容器需要承载业务时，被Istio纳管也不需要修改源代码，重新在OpenShift上注入Sidecar部署即可。

配置生效

如果系统中已经有相关对象的配置，我们需要使用`oc replace -f`指定配置文件来替换此前配置的对象。Istio有的配置策略能否较快生效，有的配置需要一段时间才能生效，如限流、熔断等。新创建策略生效（`oc create -f`）的速度要高于替换性策略（`oc replace -f`）。因此在不影响业务的前提下，可以在应用新策略之前，先将旧策略删除。

此外，Istio的配置生效，大多数都是针对微服务所在的项目的，有的配置则是针对Istio系统的。因此，在应用配置的时候，要注意指定对应的项目。在OpenShift中，Virtual Service和Destination Rules都是针对项目生效。因此应用配置的时候需要指定项目。

功能健壮性参考

从笔者大量的测试效果看，健壮性较强的功能有：

基于目标端蓝绿/灰度发布。

基于源端的蓝绿/灰度发布。

灰度上线。

服务推广。

延迟和重试。

错误注入。

mTLS

黑白名单

健壮性有待提升的功能有：

限流

熔断

所以，整体上看，Istio的功能日趋完善，但仍有待提升。

入口流量方式选择

在文中笔者已经提到，在创建ingressgateway的时候，会自动在OpenShift的Router上创建响应的路由。Ingressgateway能够暴露的端口要多于Router。所以，我们可以根据需求选择通过哪条路径来访问应用。在Istio体系里不使用Router，我们一样可以正常访问微服务。但是PaaS上运行的应用未必都是Istio体系下的，其他非微服务或者非Istio体系下的服务还是要经过Router访问。此外，包括Istio本身的监控系统和Kiali的界面都是通过Router访问的。

相比于Spring Cloud，Istio较好地实现了微服务的路由管理。但在实际上生产中，仅有微服务的管理是不够的。例如不同微服务之间的业务系统集成、微服务的API管理、微服务中的规则流程管理等。

基于OpenShift实现的企业微服务治理需求1

功能列表	描述	Spring Cloud	Spring Cloud on OpenShift	Istio	Istio on OpenShift
服务注册与发现	在部署应用时，会自动进行服务的注册，其他调用方可以即时获取新服务的注册信息。	支持，基于Eureka或Consul等组件实现，提供Server和Client管理。	ETCD+OpenShift Service+内置DNS。	必须依赖PaaS实现。	ETCD+OpenShift Service+内置DNS。
配置中心	可以管理微服务的配置	支持，Spring Cloud Config组件实现。	OpenShift ConfigMap。	必须依赖PaaS实现。	OpenShift ConfigMap。
支持Namespace隔离	基于Namespace隔离微服务	必须依赖PaaS实现。	OpenShift的Project实现。	必须依赖PaaS实现。	目前版本暂不支持（目前版本Istio只支持OVS-Subnet）
微服务间路由管理	实现微服务之间相互访问的管理	基于网关Zuul实现，需要代码级别配置。	基于Camel实现。	基于声明配置文件，最终转化成路由规则实现，Istio Virtual Service和Destination Rule	基于声明配置文件，最终转化成路由规则实现，Istio VirtualService和Destination Rule
支持负载均衡	客户端发起请求在微服务端的负载均衡	Ribbon或Feigin。	Service的负载均衡，通常是Kube-proxy。	Envoy，基于声明配置文件，最终转化成路由规则实现。	Service的负载均衡和Envoy实现。
应用日志收集	收集微服务的日志	支持，提供Client对接第三方日志系统，例如ELK	OpenShift集成的EFK。	Istio提供Mixer适配器Fluentd，并提供Elasticsearch。	Istio提供Mixer适配器Fluentd，并提供Elasticsearch。OpenShift默认也提供集成的EFK。

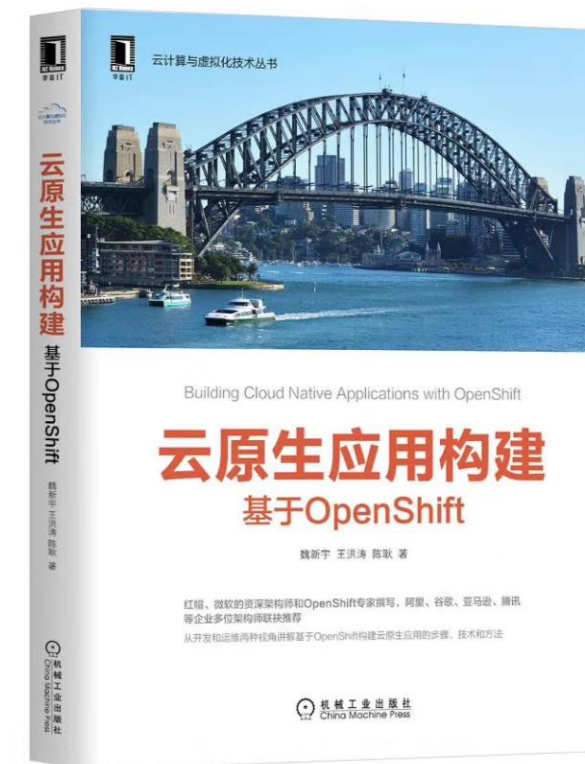
基于OpenShift实现的企业微服务治理需求2

对外访问API网关	为所有客户端请求的入口	基于Zuul或者spring-cloud-gateway实现。	基于Camel实现。	基于Ingress gateway以及egressgateway实现入口和出口的管理。	基于Ingress gateway、Router以及egressgateway实现入口和出口的管理。
微服务调用链路追踪	可以生成微服务之间调用的拓扑关系图	Zipkin实现。	Zipkin或JAEGER实现。	Istio自带的JAEGER，并通过Kiali展示。	Istio自带的JAEGER，并通过Kiali展示。
无源码修改方式的应用迁移	将应用迁移到微服务架构时不修改应用源代码	不支持。	不支持。	必须依赖PaaS实现，在部署的容器化应用的时候进行Sidecar注入。	在部署的时候进行Sidecar注入。
灰度/蓝绿发布	实现应用版本的动态切换	需要修改代码实现。	OpenShift Router。	Envoy实现，基于声明配置文件，最终转化成路由规则实现。	Envoy实现，基于声明配置文件，最终转化成路由规则实现。
灰度上线	允许将实时流量的副本，客户无感知。	不支持。	不支持。	Envoy，基于声明配置文件，最终转化成路由规则实现。	Envoy，基于声明配置文件，最终转化成路由规则实现。
安全策略	实现微服务访问控制的RBAC，对于微服务入口流量可设置加密访问。	支持，基于Spring Security组件实现，包括认证，鉴权等，支持通信加密。	OpenShift RBAC和加密Route实现。	Istio的认证和授权。	除了Istio本身的认证和授权之外，还包括OpenShift RBAC和加密Router。
性能监控	监控微服务的实施性能	支持，基于Spring Cloud提供的监控组件收集数据，对接第三方的监控数据存储。	通过OpenShift集成Prometheus和Grafana实现。	Istio自带的Prometheus和Grafana实现。	Istio自带的Prometheus和Grafana实现。
支持故障注入	模拟微服务的故障，增加可用性	不支持。	不支持。	支持退出和延迟两类故障注入。	支持退出和延迟两类故障注入。

基于OpenShift实现的企业微服务治理需求3

支持服务间调用限流、熔断	避免微服务出现雪崩效应	基于Hystrix实现，需要代码注释。	基于Hystrix实现，需要代码注释。	Envoy，基于声明配置文件，最终转化成路由规则实现。	Envoy，基于声明配置文件，最终转化成路由规则实现。
实现微服务见的访问控制黑白名单	灵活设置微服务之间的相互访问策略	需要代码注释。	通过OpenShift OVS中的networkpolicy实现。	基于声明配置文件，最终转化成路由规则实现。	基于声明配置文件，最终转化成路由规则实现。
支持服务间路由控制	灵活设置微服务之间的相互访问策略	需要代码注释。	需要代码注释。	Envoy，基于声明配置文件，最终转化成路由规则实现。	Envoy，基于声明配置文件，最终转化成路由规则实现。
支持对外部应用的访问	微服务内的应用可以访问微服务体系外的应用	需要代码注释。	OpenShift Service Endpoint	ServiceEntry。	ServiceEntry。
支持链路访问数据可视化	实时追踪微服务之间访问的链路，包括流量、成功率等。	不支持	不支持	Istio自带的Kiali。	Istio自带的Kiali。

图书推荐



Thank you



linkedin.com/company/red-hat



youtube.com/user/RedHatVideos



facebook.com/redhatinc



twitter.com/RedHat