红帽 OpenShift 4 基础（12小时）

应用部署

DevOps 和 CI/CD 简介

那么，DevOps 是什么?

- 通过推动企业文化、业务自动化和平台设计等诸多转变，帮助企业提升业务价值和响应能力的一种方式
- 目标：提升交付新功能和服务的速度及灵活性

红帽 DevOps 愿景

- 红帽文化以开放性和透明性为基础
  - 逾 25 年以来一直秉承这一理念
- 红帽具有指引客户逐步实现 DevOps 的传统
- 红帽不断构建各种工具、平台、基础架构，为客户的整个 DevOps 流程提供强大支持

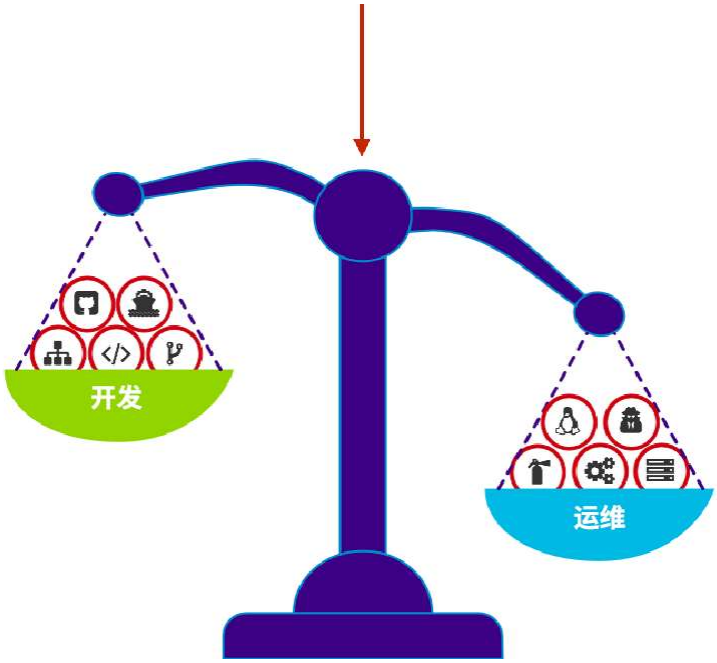
实现 DevOps 的技术不断演进

当前的技术	演进中的技术
平台即服务 (PaaS)	中立的容器平台
复杂的集群部署	通过集群管理的自助部署
平台更新中断业务价值	无线更新
自定义配置自动化	不可变基础架构
基础架构即服务 (IaaS)	集群驱动的基础架构
简单应用部署	OpenShift 服务网格驱动的复杂拓扑
微服务	红帽 OpenShift 无服务器计算
容器来自互联网	已验证、受支持的运行时和中间件容器
定制式基础架构部署咨询	DevOps 咨询和红帽开放创新实验室

信息孤岛效率低下

- **开发人员关注的是**:
  - 构建应用
  - 自动化测试
  - 持续集成
  - 性能调优
  - 调试
- **运维人员关注的是**:
  - 部署应用

- 管理应用、基础架构
- 可靠性
- 安全性
- 合规性



无开放容器平台的应用开发

- 为每个应用项目重复以下流程

流程图	
物理 1. 产生想法 2. 获得预算 3. 提交硬件采购请求 4. 等待 5. 获得硬件 6. 上架硬件 7. 安装操作系统 8. 安装操作系统补丁/修复包 9. 创建用户帐户 10. 部署框架/应用服务器 11. 部署测试工具 12. 试用测试工具 13. 编写代码 14. 配置生产服务器（必要时购买） 15. 推送到生产 16. 启动 17. 订购更多服务器以满足需求 18. 等待 19. 部署新服务器 20. 等等	**虚拟** 1. 产生想法 2. 获得预算 3. 提交虚拟机请求 4. 等待 5. 部署框架/应用服务器 6. 部署测试工具 7. 试用测试工具 8. 编写代码 9. 配置生产虚拟机 10. 推送到生产 11. 启动 12. 订购更多生产虚拟机以满足需求 13. 等待 14. 向新虚拟机部署应用 15. 等等

“通过运用以开发人员为中心的容器平台技术，*IT* 组织能够变得更加敏捷，对企业务需求的响应能力也更强。”

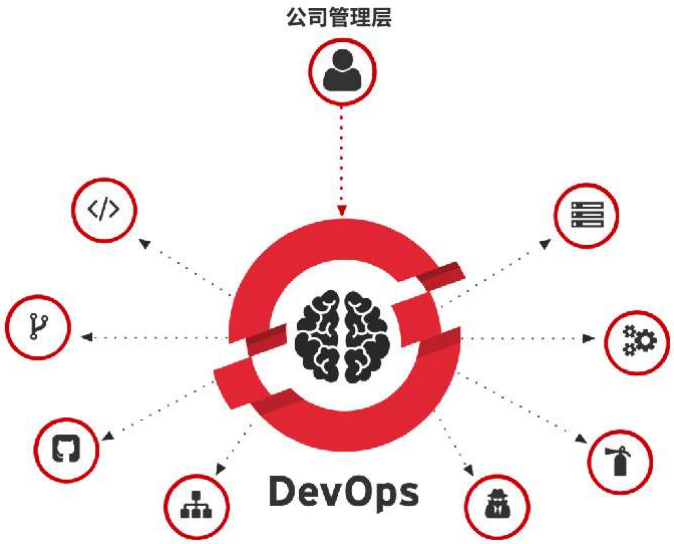
— Gartner

利用开发容器平台的应用开发

流程图	流水线	
物理 1. 产生想法 2. 获得预算 3. 提交硬件采购请求 4. 等待 5. 获得硬件 6. 上架硬件 7. 安装操作系统 8. 安装操作系统补丁/修复包 9. 创建用户帐户 10. 部署框架/应用服务器 11. 部署测试工具 12. 试用测试工具 13. 编写代码 14. 配置生产服务器（必要时购买） 15. 推送到生产 16. 启动 17. 订购更多服务器以满足需求 18. 等待 19. 部署新服务器 20. 等等	**虚拟** 1. 产生想法 2. 获得预算 3. 提交虚拟机请求 4. 等待 5. 部署框架/应用服务器 6. 部署测试工具 7. 试用测试工具 8. 编写代码 9. 配置生产虚拟机 10. 推送到生产 11. 启动 12. 订购更多生产虚拟机以满足需求 13. 等待 14. 向新虚拟机部署应用 15. 等等	**利用容器平台** 1. 产生想法 2. 获得预算 3. 选择最佳工具 4. 编写代码 5. 测试 6. 启动 7. 自动扩展

DevOps 的重点 - 协作与配合

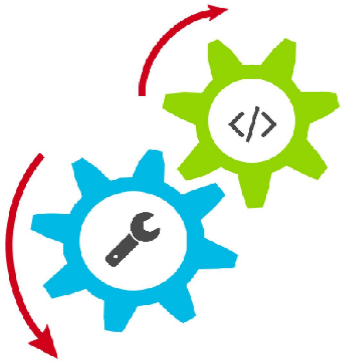
- 无信息孤岛：
  - 改进沟通和提高共识
  - 提升敏捷性和效率
  - 加快上市时间



涉及的人员?

- 最终的产品是团队协作成果
- 不再有“我们和他们”或“这个部门和那个部门”之分

- 共同利益转变为每个人的精益、敏捷理念



- 开发人员
  - 更多参与产品/代码
  - 不需要在要求阶段花费数月，再转交到运维

- 运维
  - 利用持续部署等方法，大大增强绩效
  - 随着绩效提升，盈利能力、市场份额和生产效率也会提高

部署

- OpenShift 部署提供精细的应用管理
  - 基于用户定义的模板（称为“部署配置”）
- 有许多不同的模板表单适合团队的风格
  - 示例：模板、Helm、Kustomize
- 在响应部署配置时，部署系统会创建复制控制器来运行应用
- OpenShift 就是 Kubernetes
  - Kubernetes“部署”⟺ OpenShift“部署配置”
  - Kubernetes“ReplicaSet”⟺ OpenShift“复制控制器”

部署系统提供的功能

- 部署配置：用于运行应用的模板
  - 包含版本号
    - 每次从配置创建新的复制控制器 (Kubernetes ReplicaSet) 时递增
  - 包含最后部署的副本控制器的原因
- 为响应事件而触发自动化部署的触发器
- 从前一版本过渡到新版本的策略
- 部署失败时回滚到上一版本
  - 不论是手动还是自动

- 手动复制扩展和自动扩展

回滚

- 部署允许回滚
  - 回滚将应用恢复到上一修订
  - 通过 REST API、CLI 或 Web 控制台操作
- 部署配置支持自动回滚到配置的最近一次成功修订
  - 在最新模板部署失败时触发

部署策略

- 由部署配置定义
- 决定部署流程
  - 部署过程中，每一应用具有不同的要求
    - 可用性
    - 其他考虑
- OpenShift 提供支持各种不同部署场景的策略
- 就绪度检查决定新 pod 是否就绪可用
  - 如果就绪度检查失败，部署配置将重试至超时

滚动部署策略

- 执行滚动更新
- 支持用于注入代码到部署过程中的生命周期 Hook
- 等待 pod 通过就绪度检查，然后再下线旧组件
  - 在超时期限内，不允许未通过就绪度检查的 pod
- 部署配置中未指定策略时默认采用

滚动部署策略流程

- 滚动策略流程中的步骤：
  1. 执行 `前期` 生命周期 Hook
  2. 通过一个或多个 pod 向上扩展新部署（基于 `maxSurge` 值）
     1. 等待就绪度检查完成
  3. 通过一个或多个 pod 向下扩展旧部署（基于 `maxUnavailable` 值）
  4. 重复执行扩展，直到新部署达到所需的副本数，并且旧部署扩展到零
  5. 执行任何 `后期` 生命周期 Hook
- 在向下扩展时，策略会等待 pod 变为就绪
  - 决定进一步扩展是否影响可用性
- 如果向上扩展的 pod 永不就绪，部署会超时
  - 导致部署失败

重建策略

- 具有基本的规模部署行为
- 支持用于注入代码到部署过程中的生命周期 Hook

- 重建策略部署中的步骤：
  - 执行 `前期` 生命周期 Hook
  - 将上一部署向下扩展到零
  - 向上扩展新的部署
  - 执行 `后期` 生命周期 Hook

自定义部署策略

- 示例：

```
"strategy": {
  "type": "Custom",
  "customParams": {
    "image": "organization/strategy",
    "command": ["command", "arg1"],
    "environment": [
      {
        "name": "ENV_1",
        "value": "VALUE_1"
      }
    ]
  }
}
```

  - `organization/strategy` 容器镜像提供部署行为
  - 可选的 `command` 数组覆盖镜像 Dockerfile 中指定的 CMD 指令
  - 可选的 `environment` 变量添加到策略流程的执行环境中

CI/CD 工作流

CI/CD 作为 DevOps 实践

- DevOps = 持续集成 (CI) + 持续部署 (CD)

CI 是什么?

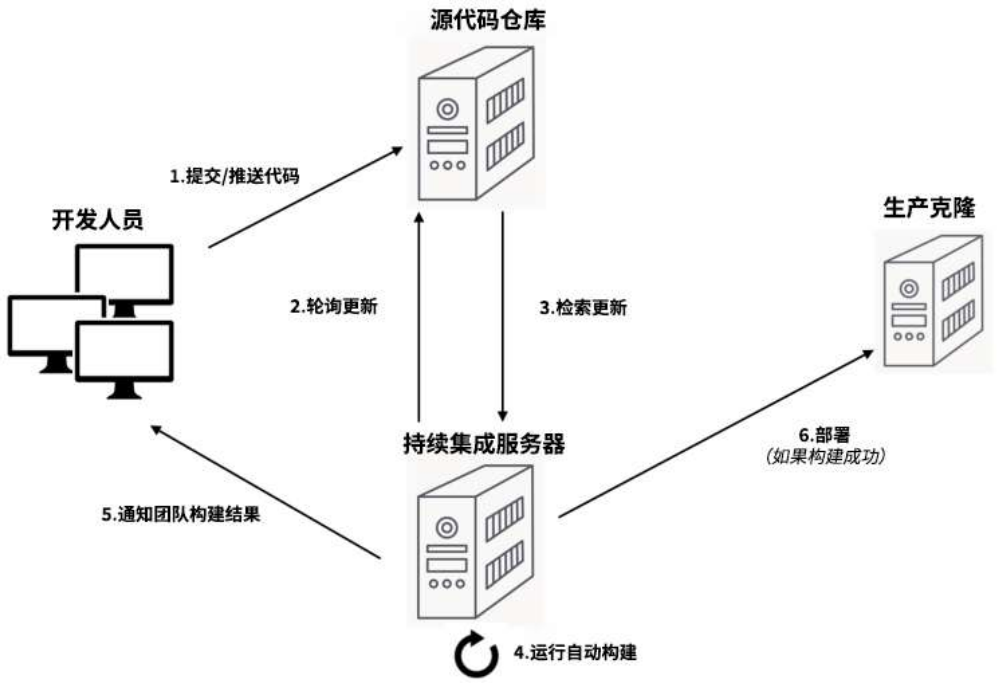
功能	描述
验证构建完整性	检查能否从仓库拉取源代码，能否进行部署所需的构建构建流程可以包含对软件进行编译、打包和配置
验证测试结果	在类生产环境中运行开发人员创建的测试验证源代码没有因为提交所带来的副作用而被破坏
检查多个系统之间的集成	利用集成测试验证软件使用的系统集成
确定问题	提醒受影响的团队修复问题

如何执行 CI?

- CI 可以手动执行，但通常通过工具管理

- 在 DevOps 中，CI 必不可少，并由工具执行
  - 工具运行由开发人员创建的自动化脚本
  - 避免 CI 流程中的人为干预

CI 工作流



CI 的优点

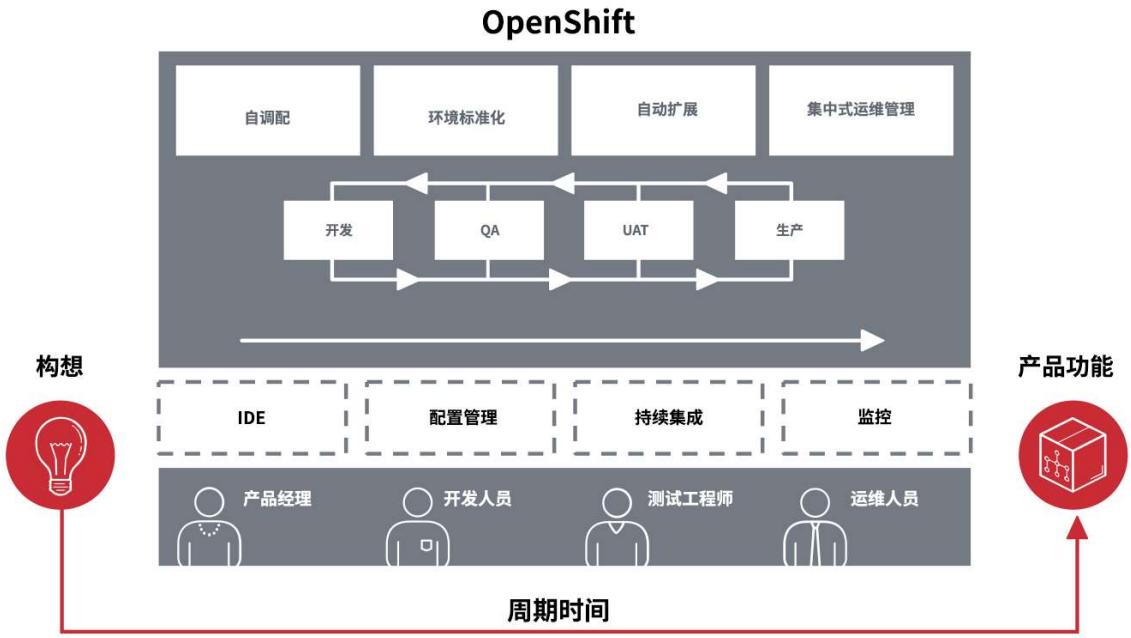
优点	说明
快速反馈	构建执行后立即发送状态通知 缩短发现和修复新缺陷的时间
降低风险	一天集成多次，降低项目中的风险 软件健康度可通过单元测试和代码检查报告衡量
团队负责	不再有“我们和他们”之分 每个人定期收到构建状态报告 项目可见性更强，人人都能察觉趋势并做出有效的决策 增强为项目添加功能的信心 参与人员全部知晓最新的项目状况
构建可部署的软件	构建流程生成可部署的软件 目标是创建可以随时部署的软件 许多开发团队对此难以应付 不意味着您必须要部署，只表示这是良好的候选版本
自动化流程	自动构建节省时间、金钱和精力 流程运作始终一致 让开发人员有时间去完成更多高价值工作

CI 最佳做法

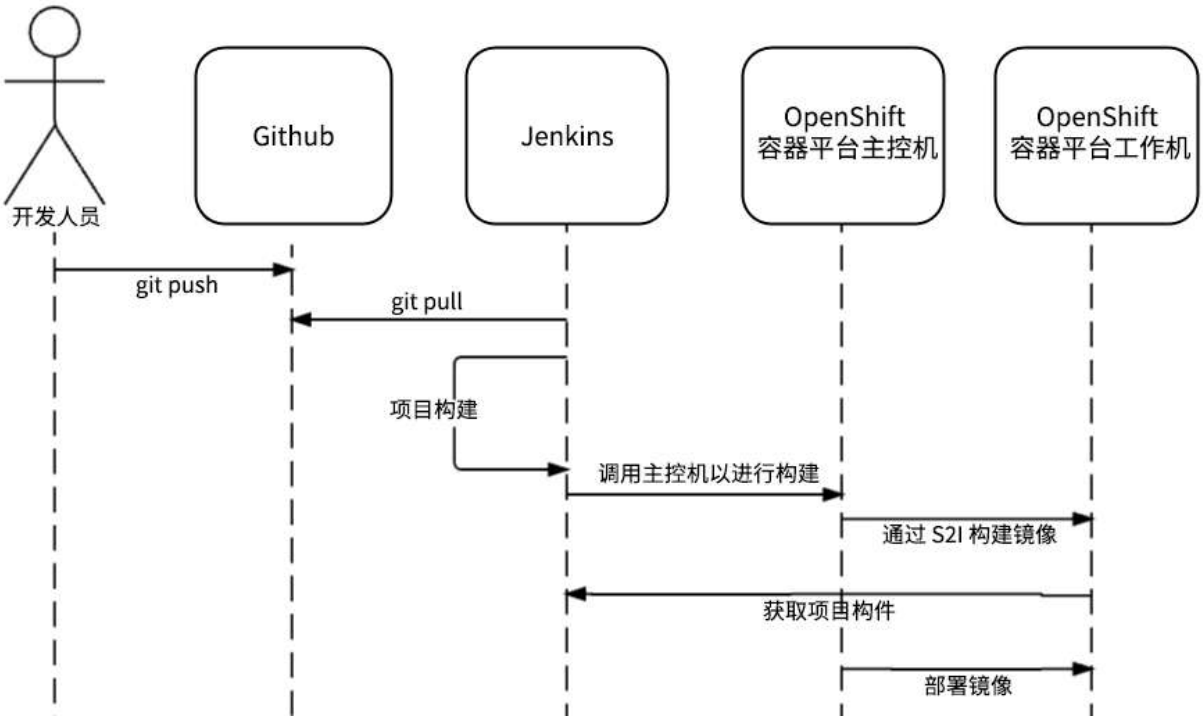
- 维护代码仓库
- 自动化构建
- 使构建具备自我测试能力
- 确保人人每天提交
- 保持快速构建
- 在生产克隆中测试
- 轻松获得可交付品
- 确保每个人都能查看构建结果

OpenShift 推动 DevOps

- OpenShift 提供“代码内配置”
- 通过标准化环境、Linux® 容器、自动化调配来实现
- 内置对 Jenkins CI 服务器的支持
- Jenkins 为 OpenShift 环境提供自动化应用测试、部署
- 功能可以缩短从概念到完成产品部署所需的周期时间



采用 OpenShift 的 CI



构建和 S2I

构建

- 将输入参数转换为结果对象的过程
  - 最常用于将输入参数或源代码转换为可运行的镜像
- `BuildConfig` 对象：定义整个构建过程
- OpenShift 利用 Kubernetes
  - 从构建镜像创建容器
  - 将容器推送到集成式注册表

构建策略

- OpenShift 中的构建系统提供"构建策略"的可扩展支持
  - 构建 API 中指定的可选择策略
- 构建策略主要有四种：
  - 容器构建
  - Source-to-Image (S2I) 构建
  - 自定义构建
  - 流水线构建（已弃用）
- 从构建生成的对象取决于创建它时所用的构建器
  - 容器和 S2I：可运行的镜像
  - 自定义：构建器镜像作者指定的任何镜像
  - 流水线：可由 Jenkins 流水线插件（已弃用）执行的 Jenkins 流水线

容器构建

- 容器构建策略单纯调用 `podman build` 命令
- 需要含有 Dockerfile 的仓库以及的必要构件，从而生成可运行的镜像

参考

- `podman build` 命令：podman-build docs
- `podman` 在功能上等同于 `docker`：Intro to Podman

Source-to-Image (S2I) 构建

- S2I：用于构建可再生容器镜像的工具
  - 生成可随时运行的镜像
    - 注入应用源代码到容器镜像，汇编新的容器镜像
  - 新镜像融合了基础镜像（构建器）和构建的源
  - 可以随时通过 `podman run` 命令来使用
- S2I 支持递增构建
  - 重复利用以前下载的依赖项，以及以前构建的构件等。

Source-to-Image (S2I) 构建优势

优势	说明
镜像灵活性	S2I 脚本可以将应用代码注入到大部分容器镜像，以此利用现有的生态系统使用 `tar` 来注入应用源代码
速度	汇编流程将执行大量复杂操作，无需在每一步创建新层S2I 脚本可重复利用应用镜像旧版本中存储的构件
可修补性	如果基础镜像因为安全问题而需要补丁，以一致的方式重新构建应用
操作效率	限制构建操作并且防止 Dockerfile 允许的任意操作避免意外或故意滥用构建系统
操作安全性	S2I 限制以 `root` 用户身份执行操作，并且能够以非 root 用户身份运行脚本； 避免主机系统面临通过任意 Dockerfile 构建进行 root 特权升级的风险； 容器构建流程不再由具有容器特权的用户来运行
生态系统	S2I 倡导共享镜像生态系统； 允许您将最佳实践运用于自己的应用
可再生性	生成的镜像可以包含所有输入； 构建工具和依赖项的特定版本； 确保镜像可以精确再生

自定义构建

- 自定义构建策略：您可以定义负责整个构建流程的特定构建器镜像
  - 通过利用自己的构建器镜像，您可以自定义构建流程
- 自定义构建器镜像是嵌入了构建流程逻辑的干净 OCI 合规容器镜像
  - 示例：构建 RPM 或基础容器镜像
  - 自定义构建器镜像的示例实施：
    - `openshift/origin-custom-docker-builder` 镜像，位于 Docker Hub

镜像流

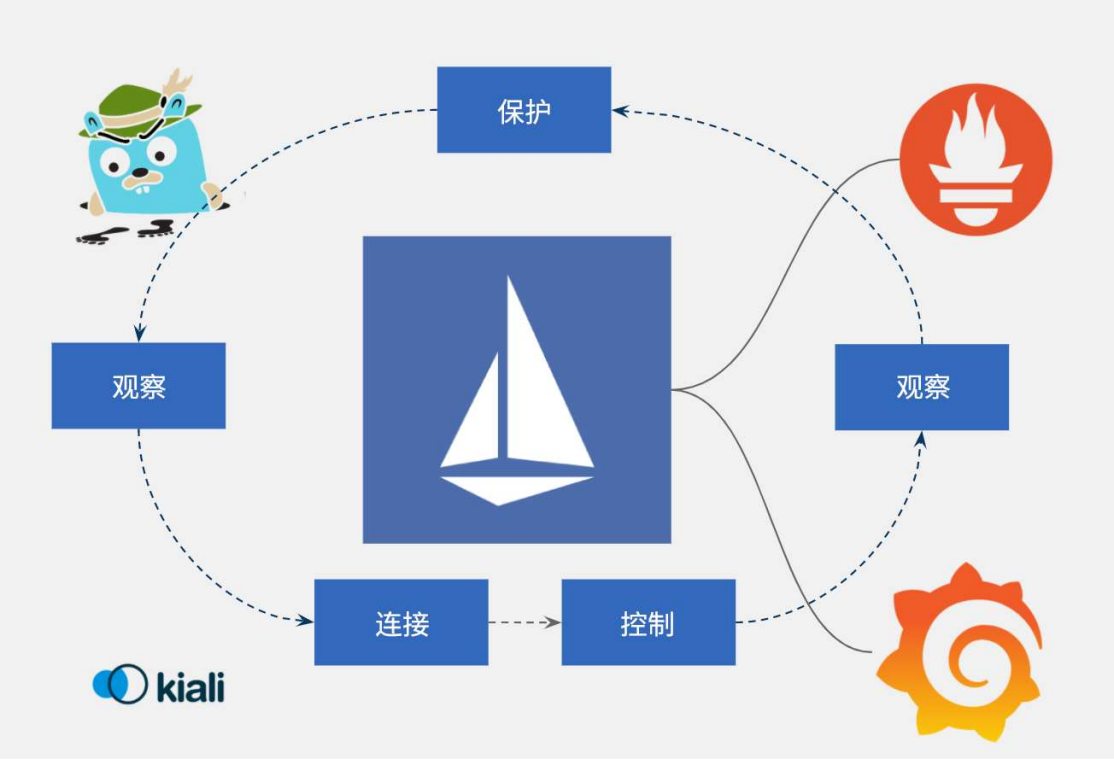
- 由通过标签标识的 OCI 合规容器镜像组成

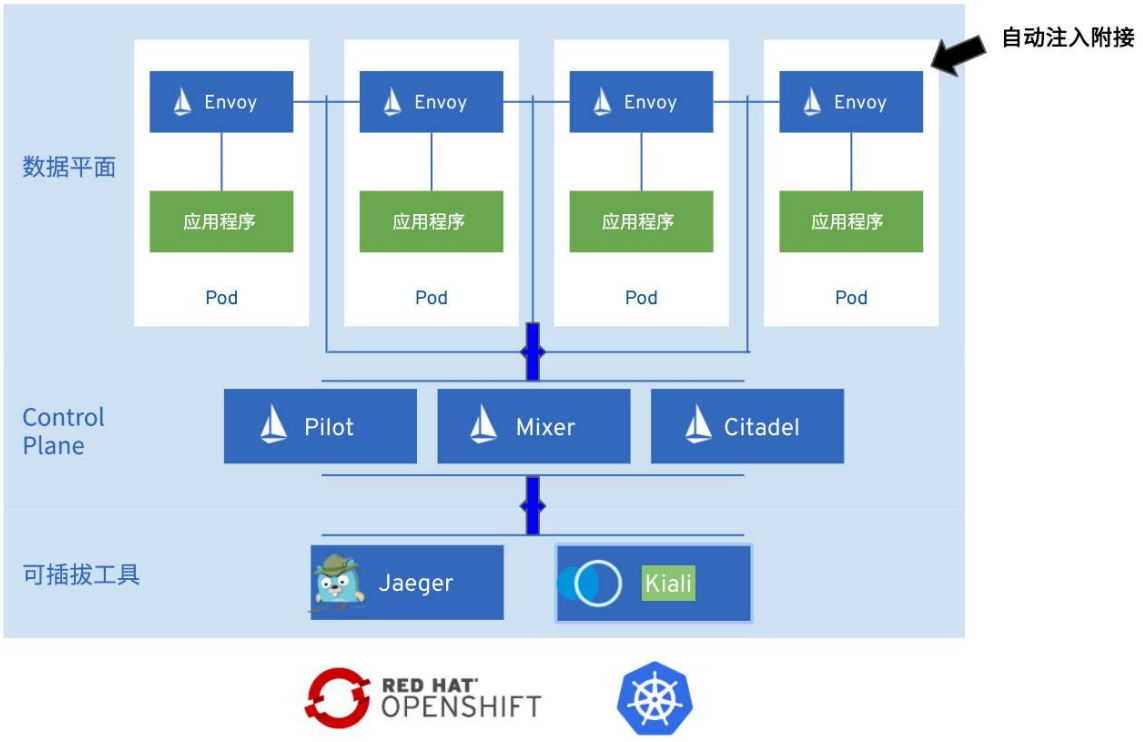
- 提供相关镜像的单一虚拟视图
    - 类似于容器镜像仓库
- 可以包含以下任何来源的镜像：
    - 集成式 OpenShift 容器平台注册表中的镜像仓库
    - 其他镜像流
    - 来自外部注册表的容器镜像仓库
- 新镜像添加至相关的镜像流
- 构建和部署观察镜像流
    - 在新增镜像时接收通知
    - 通过执行构建或部署来做出反应
- 示例：如果部署使用特定镜像并且已创建了镜像的新版本，部署会自动执行

红帽® OpenShift® 服务网格

概述

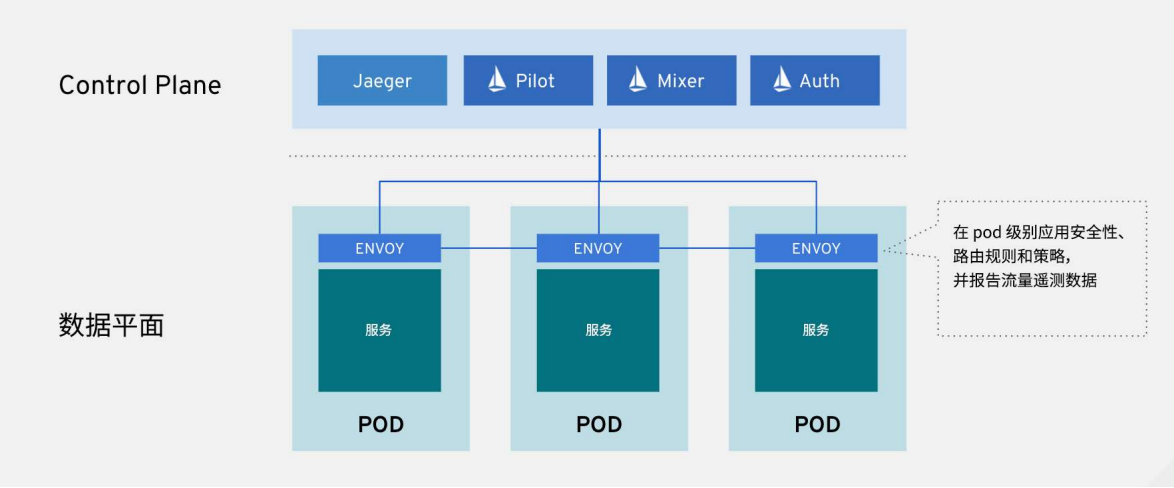
- 增强微服务的部署选项
- CI/CD 系统提供最小配置信息
- 产品
    - Istio：面向服务对服务的附接网络代理
    - Jaeger：追踪
    - Prometheus 和 Grafana：遥测
    - Kiali：可观察性





Istio

- 服务对服务通信层
- 管理发现、负载平衡、故障恢复、指标和监控
- 微服务操作支持 A/B 测试、金丝雀推出部署、速率限制、访问控制和端到端身份验证
- 以作为附接容器注入的一组轻量型代理来实施



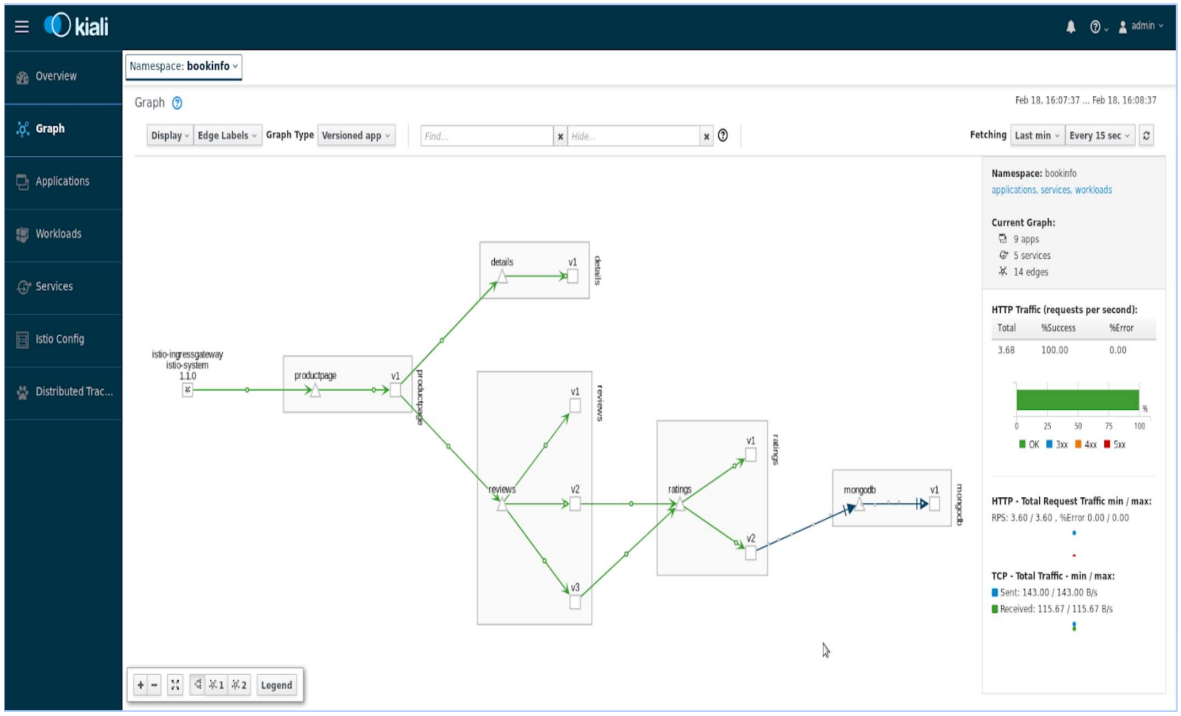
Jaeger

- 符合 OpenTracing 标准
- 高扩展性
- 多个存储后端
- 现代 Web UI
- 云原生

    - 可由 Operator 部署
    - Prometheus 指标
- 实现微服务可观察性
    - 分布式上下文传播
    - 分布式事务监控
    - 根本原因分析
    - 服务依赖关系分析
    - 性能和延迟优化

Kiali

- 可视化呈现 OpenShift 服务网格拓扑
- 图形：工作负载、应用、服务、版本
- 详细视图：网格配置、指标、服务、工作负载、运行时



红帽 OpenShift 无服务器技术

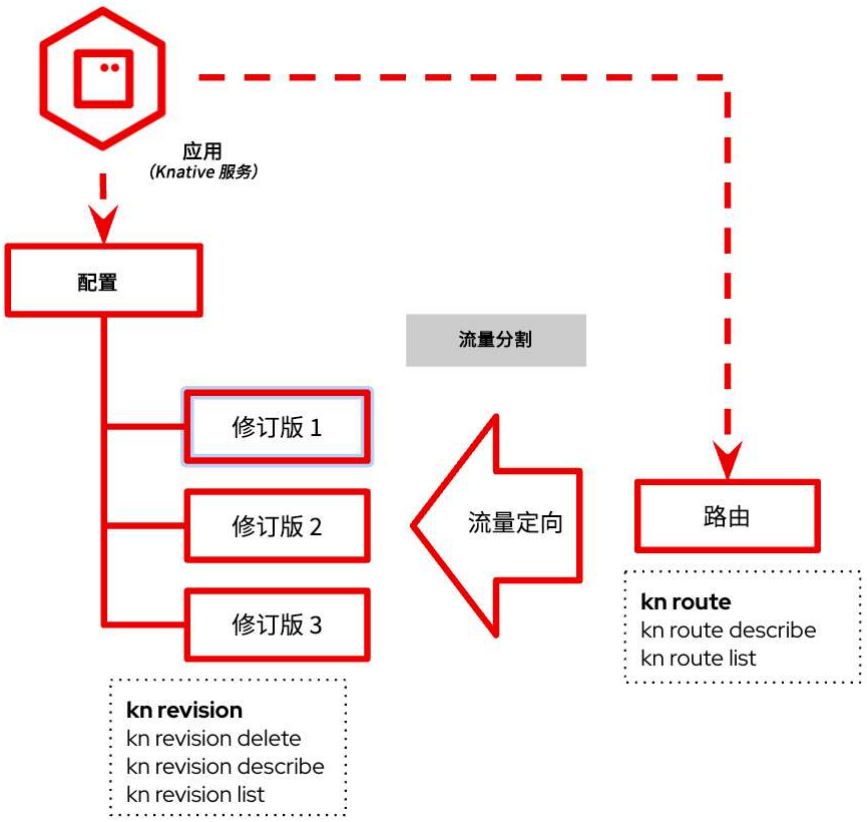
概述

- 根据应用事件触发部署
- Knative 服务
- Knative 事件
- Camel K + 接口
- Kafka + Strimzi
- 几乎任何容器化应用都可实现无服务器运行
- 自动扩展
- 事件源
- 接收 Kafka 消息

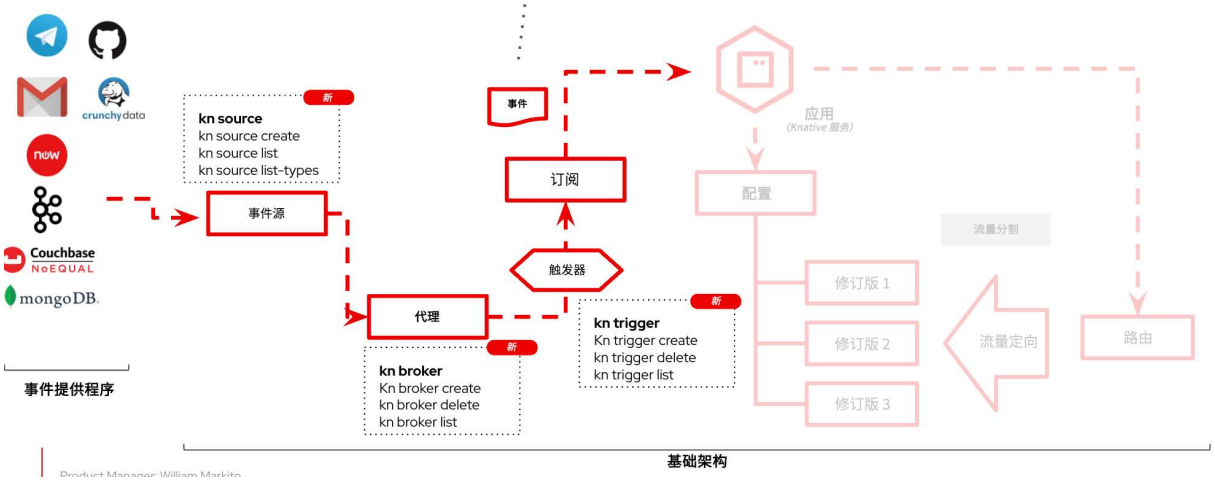
- 定时器等等

服务

- 在几秒内从容器服务到 URL
- 简化开发人员的 Kubernetes 体验
- 内置版本控制和流量分割等功能
- 简化 Kourier 安装体验
- 自动为应用实施 TLS/SSL



事件



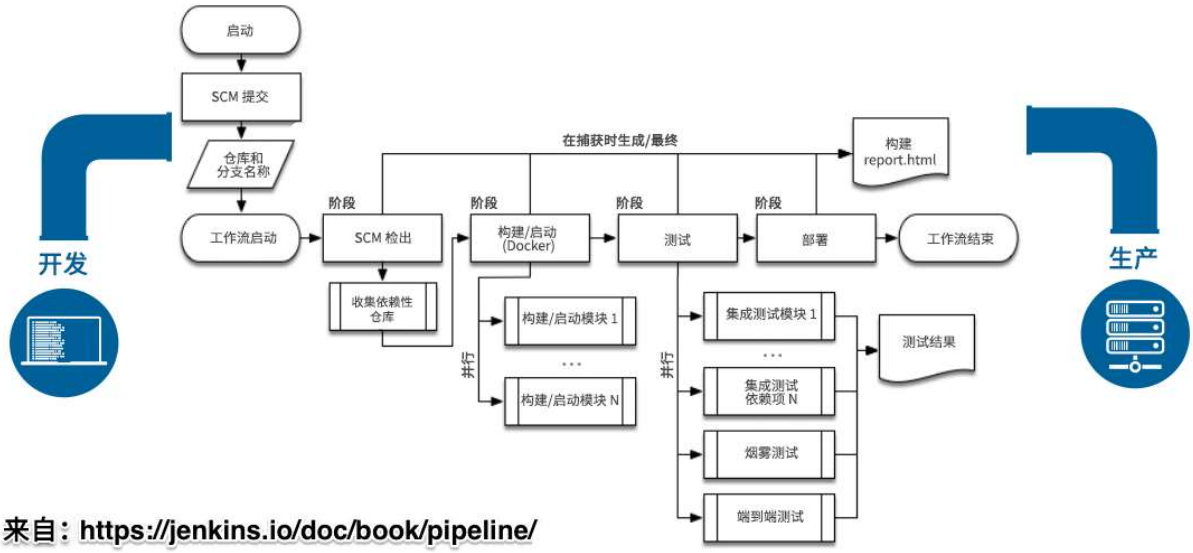
流水线

- 基于 Tekton 的流水线处于技术预览阶段
  - 无服务器 CI/CD
- 由流水线插件实现的 Jenkins 作业
  - 通过简单文本脚本构建，利用基于 Groovy 的流水线 DSL
- 使一个脚本能够应付复杂工作流中的所有步骤
- 利用多重步骤的功能，根据用户确立的参数执行简单和复杂的任务
- 构建代码并编排所需的工作，以推动应用从提交走向交付

参考

- https://www.openshift.com/learn/topics/pipelines
- https://jenkins.io/doc/book/pipeline/

Jenkins 流水线



来自：https://jenkins.io/doc/book/pipeline/

- 让您能够定义整个应用生命周期
- 支持持续集成/持续部署 (CI/CD)
- 流水线插件的构建满足对灵活、可扩展的基于脚本型 CI/CD 工作流的要求

流水线特征

- 可持久
  - 在计划和非计划 Jenkins 主控机重启后继续有效
- 可暂停
  - 可以选择停止并等待人工输入或批准，然后完成作业
- 多功能
  - 支持复杂的真实世界 CD 要求
    - 能够分叉或结合
    - 能够循环
    - 能够与其他流水线并行工作
- 可扩展
  - 插件支持利用其 DSL 自定义扩展，也提供与其他插件集成的多个选项

注意：流水线构建策略已在 OpenShift 容器平台 4 中弃用。

流水线构建策略

- 允许开发人员定义 Jenkins 流水线并由 Jenkins 流水线插件执行
- 构建由 OpenShift 启动、监控和管理，与其他构建类型相同
- Pipeline 工作流在 `Jenkinsfile` 中定义
  - 直接嵌入在构建配置中
  - 或者在 Git 仓库中提供，并由构建配置引用

总结

- DevOps 和 CI/CD 简介
- 部署
- CI/CD 工作流
- 构建和 S2I
- 红帽 OpenShift 服务网格
- 红帽 OpenShift 无服务器技术
- 流水线