GIAC

全球互联网架构大会

GLOBAL INTERNET ARCHITECTURE CONFERENCE

面向Kubernetes的DevSecOps
——JFrog的Kubernetes之旅

高欣 JFrog 架构师

演讲大纲

JFrog内部的Kubernetes实践

- 我们正在做的：
  - JFrog的应用和服务全面Kubernetes化
  - 内部的研发和测试环境全面Kubernetes化
- 我们要分享的：
  - 实践中积累的教训和经验

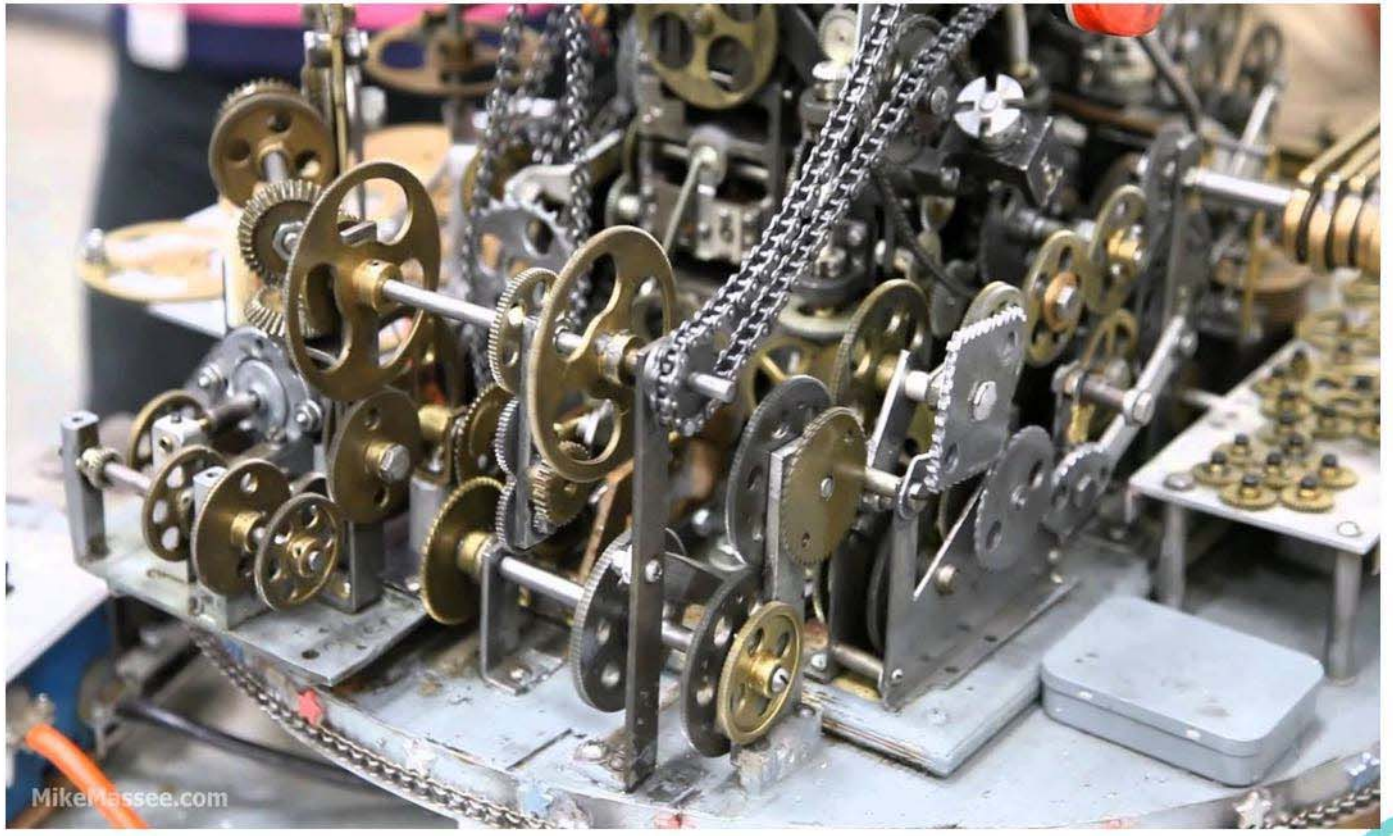
背景介绍

要解决的问题

- 用户安装部署JFrog产品复杂
- 无法快速搭建JFrog产品的全功能测试环境
  - 无法实现按需使用：开发、测试、技术支持、产品、解决方案。。。任意团队
- 无法为每个分支提供独立的CI/CD流水线支撑
  - 无法让研发有独立的沙箱环境进行自测
  - CI/CD流程混乱

背景介绍

Kubernetes is hard!



案例成果

实践的成果

- 为客户提供全产品线的Helm Charts交付方式
  - Helm install stable/Artifactory-ha
- 云端服务Kubernetes化
  - GoCenter: http://gocenter.io
- 产品的CI/CD直接对接到Kubernetes环境
  - 每周部署100+不同产品线、任意版本组合的测试环境，每次部署超过50种微服务
  - 为每个研发、每个分支，按需提供完全独立的测试环境

成功要点

- 起步：熟悉Kubernetes
- 规划：面向Kubernetes的改造
- 编排：Helm Charts
- 部署：自动、监测
- 安全：DevSecOps

起步：熟悉Kubernetes —— 从小处入手

- 第一个Kubernetes环境
  - 自己探索：Kubernetes The Hard Way，Kelsey Hightower (https://github.com/kelseyhightower/kubernetes-the-hard-way)
  - 公有服务：AKS、EKS、GKE、阿里、腾讯、。。。。
  - 私有部署：miniKube、Rancher、。。。。
- 第一个Kubernetes应用
  - 从小的示例应用开始，如Nginx
  - 每次只设定一个小的、具体的目标
  - 充分利用现有的教程和演示

规划：面向Kubernetes的改造

THE TWELVE-FACTOR APP

The Twelve-Factor App （https://12factor.net/zh_cn/）

I. **基准代码**：一份基准代码，多份部署

II. **依赖**：显式声明依赖关系

III. **配置**：在环境中存储配置

IV. **后端服务**：把后端服务当作附加资源

V. **构建，发布，运行**：严格分离构建和运行

VI. **进程**：以一个或多个无状态进程运行应用

VII. **端口绑定**：通过端口绑定提供服务

VIII. **并发**：通过进程模型进行扩展

IX. **易处理**：快速启动和优雅终止可最大化健壮性

X. **开发环境与线上环境等价**：尽可能的保持开发，预发布，线上环境相同

XI. **日志**：把日志当作事件流

XII. **管理进程**：后台管理任务当作一次性进程运行

规划：应用的Kubernetes改造

仅仅把应用装进Docker是远远不够的

- 日志
  - STDOUT/STDERR
  - 处理足够多的日志文件
- 持久化数据
  - 哪些数据需要持久化存储？
- 合理地处理SIGTERM信号
  - Shutdown必须是受控的
  - Recovery必须是容易的
- 重启
  - 如何处理上一次运行的遗留数据？

规划：应用的Kubernetes改造

高可用将是新的标准配置

- 保证良好的持久性和可用性
- 支持同时运行多个应用实例
  - 支持负载均衡
  - Scale-up、Scale-down必须是顺畅的
- 不停机的滚动升级
  - 保证向后兼容
- K8s调整中的0宕机
  - Cluster Scale-up、Scale-down
  - 计划中的Node维护
  - 非计划的Node宕机

规划：配置的Kubernetes改造

运行环境的资源使用必须是受限的

- Pod的资源限制！
- 应用本身也需要资源限制
- 改造
  - requests/limits
  - 跨Node的HA
  - Out Of Resource Handling (https://kubernetes.io/docs/tasks/administer-cluster/out-of-resource/)
  - Pod Priority and Preemption (https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/)

```
...
resources:
  requests:
    memory: "1Gi"
    cpu: "100m"
  limits:
    memory: "2Gi"
    cpu: "250m"
...
```

规划：配置的Kubernetes改造

应用的运行状态必须有可信的健康数据

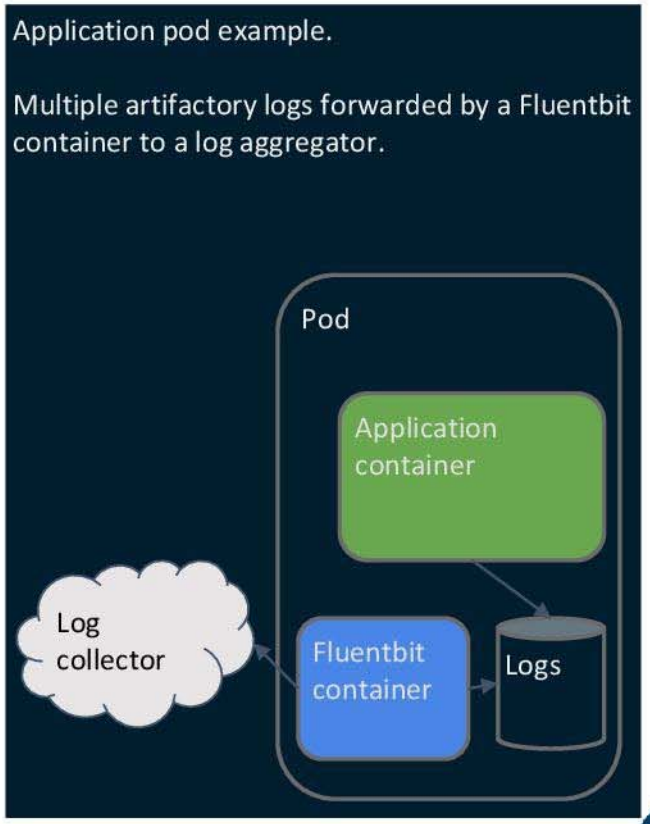
- readinessProbe
- livenessProbe
- 探针类型
  - Http - return < 400 on success
  - Tcp - succeed to open a socket on a given port
  - Exec - return 0 on success

```
...
readinessProbe:
  httpGet:
    path: /api/system/health
    port: 8080
...
livenessProbe:
  exec:
    command:
    - mongo
    - --eval
    - "db.adminCommand('ping')"
...
livenessProbe:
  tcpSocket:
    port: 5672
...
```

规划：配置的Kubernetes改造

Pod里只有应用的容器是不够的

- init容器 —— 在应用容器启动前运行
  - 准备存储
  - 初始化设置
- sidecar容器 —— 和应用容器同时运行
  - 维护
  - 日志收集
  - 监测
  - 代理



编排：Kubernetes原生 — kubectl + yaml

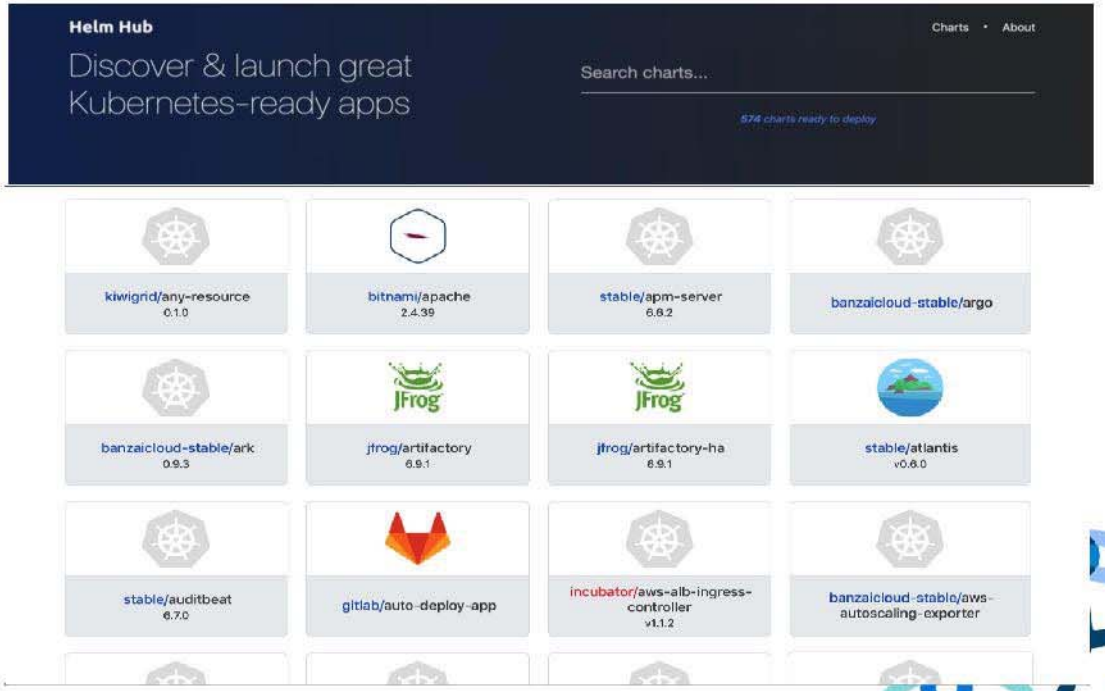
利用yaml文件编排已经足够好了吗？

- 多个组件、模块，对应多个yaml文件
- 应用的版本化怎么管理？
  - 怎么管理多个yaml文件的版本组合？
- 配置数据怎么管理？
  - 怎么部署到多个目标环境？ 自测、开发、测试、产品。。。
- 怎么回滚到之前的特定版本？

编排： Helm Charts

Helm —— 容器云应用的安装部署工具

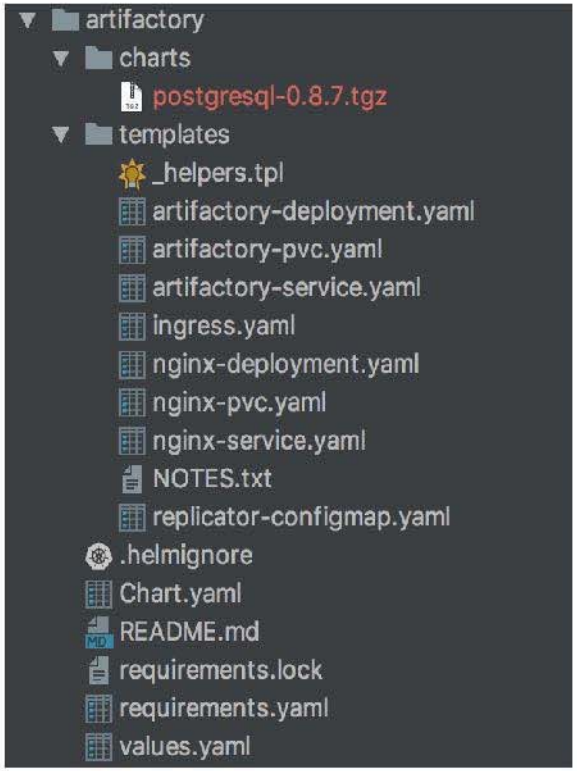
- https://helm.sh
- Helm Hub
  - https://hub.helm.sh



编排： Helm Charts

版本化管理

- Helm Chart
  - 一个包包含了所有模块——templates/
- Chart的版本化管理
  - Chart.yaml
- 运行态的版本化管理
  - Release



编排： Helm Charts

配置数据分离

- 缺省配置
  - values.yaml
- 目标环境配置
  - values-test.yaml
  - values-prod.yaml

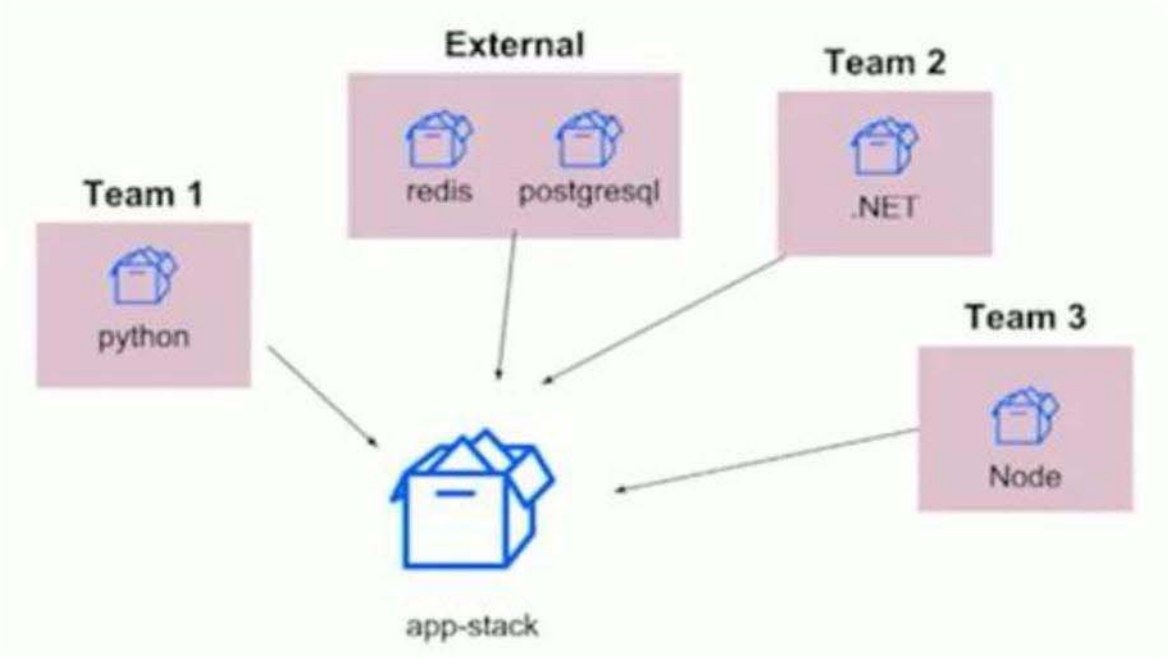
```yaml
apiVersion: v1
kind: Service
metadata:
  name: {{ template "artifactory.fullname" . }}
  labels:
    app: {{ template "artifactory.name" . }}
    chart: {{ template "artifactory.chart" . }}
    component: {{ .Values.artifactory.name }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
{{- if .Values.artifactory.service.annotations }}
  annotations:
{{ toYaml .Values.artifactory.service.annotations | indent 4 }}
{{- end }}
spec:
  type: {{ .Values.artifactory.service.type }}
  ports:
  - port: {{ .Values.artifactory.externalPort }}
    targetPort: {{ .Values.artifactory.internalPort }}
    protocol: TCP
    name: {{ .Release.Name }}
```

```yaml
service:
  ## For minikube, set this to NodePort, elsewhere use LoadBalancer
  type: LoadBalancer
  ## For supporting whitelist on the Nginx LoadBalancer service
  ## Set this to a list of IP CIDR ranges
  ## Example: loadBalancerSourceRanges: ['10.10.10.5/32', '10.11.10.5/32']
  ## or pass from helm command line
  ## Example: helm install ... --set nginx.service.loadBalancerSourceRanges
  loadBalancerSourceRanges: []
  annotations: {}
  ## Provide static ip address
  loadBalancerIP:
  ## There are two available options: "Cluster" (default) and "Local".
  externalTrafficPolicy: Cluster
externalPortHttp: 80
internalPortHttp: 80
externalPortHttps: 443
internalPortHttps: 443
```

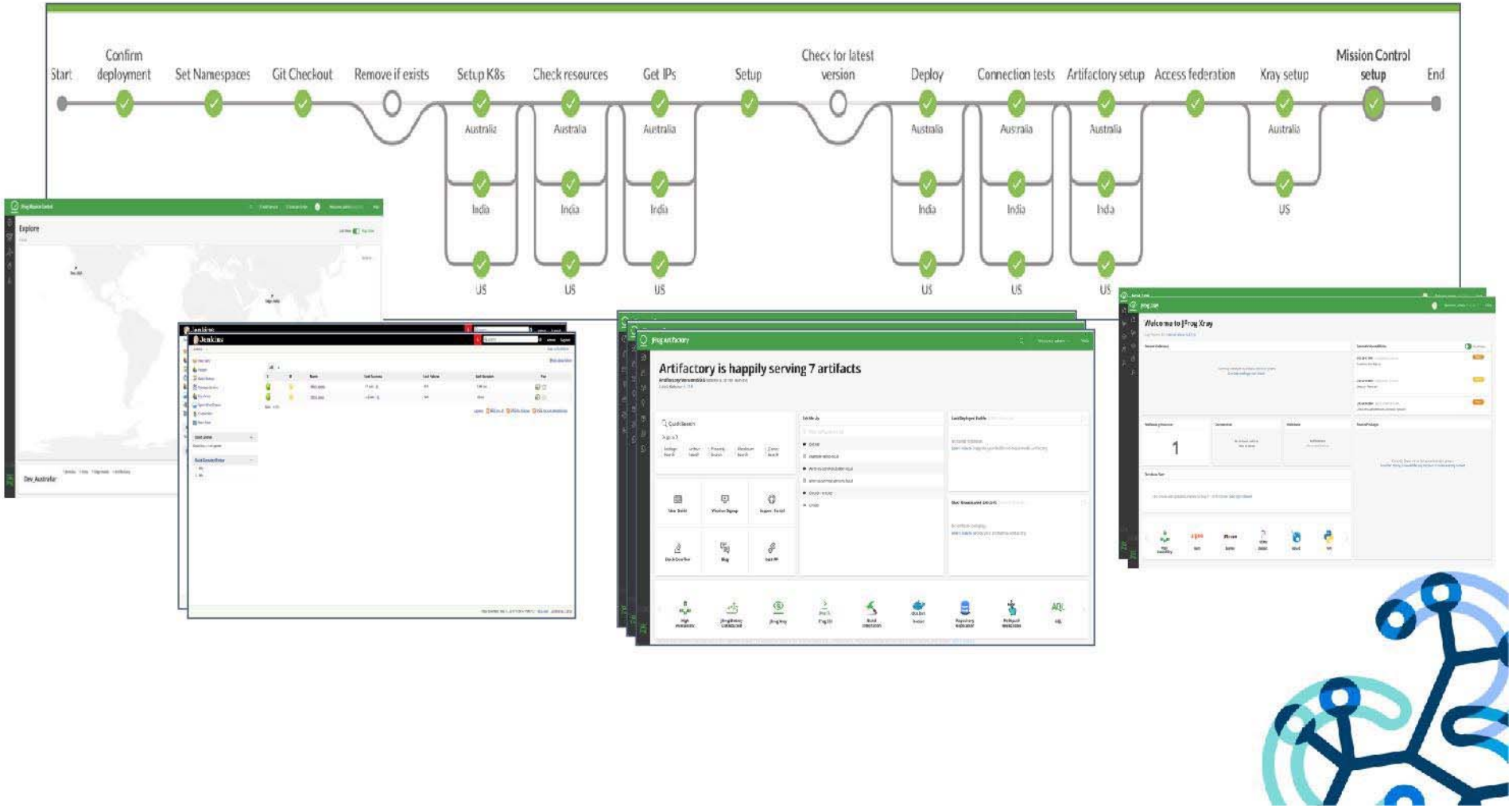
编排： Helm Charts

共享、依赖

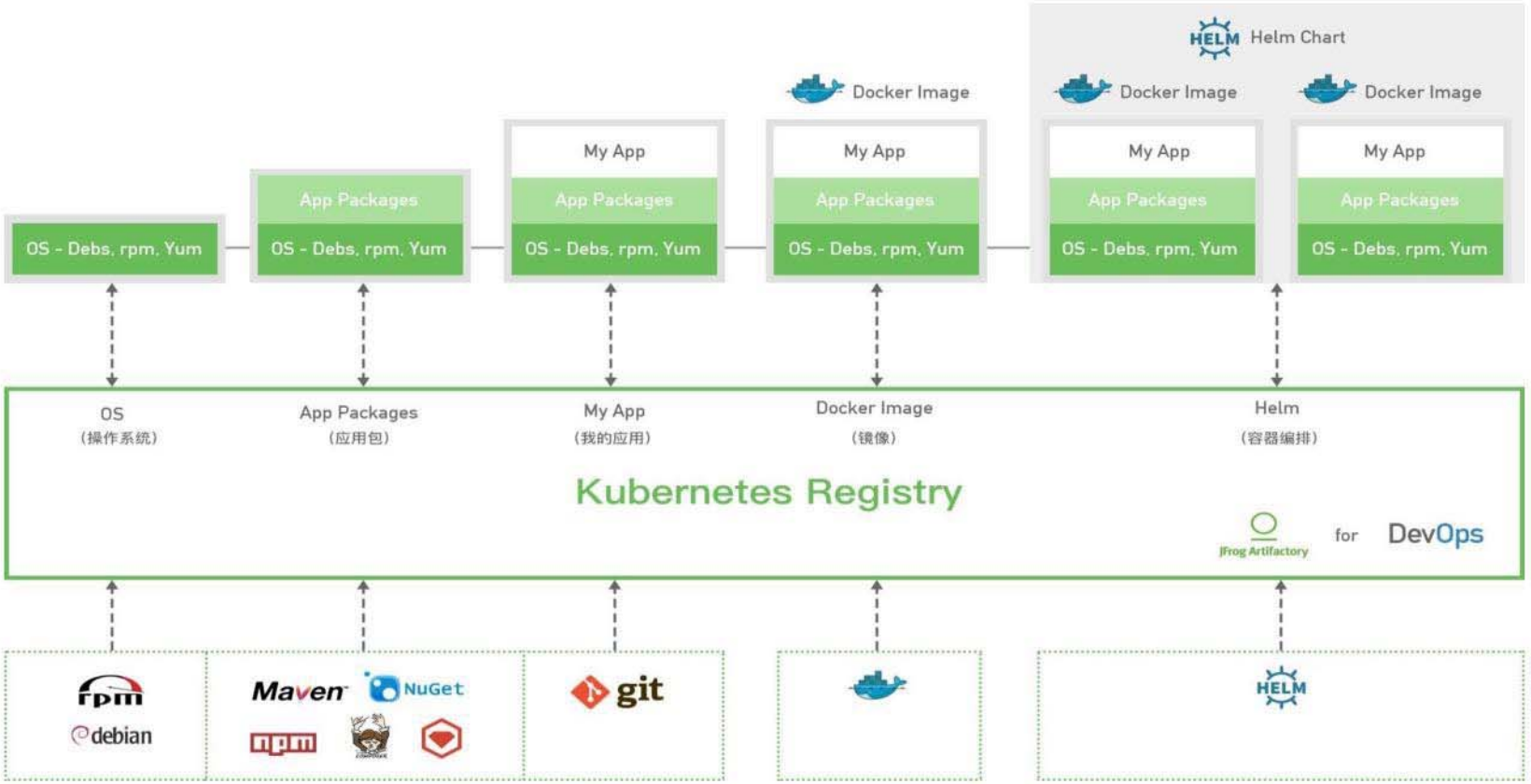
- requirements.yaml
- charts/



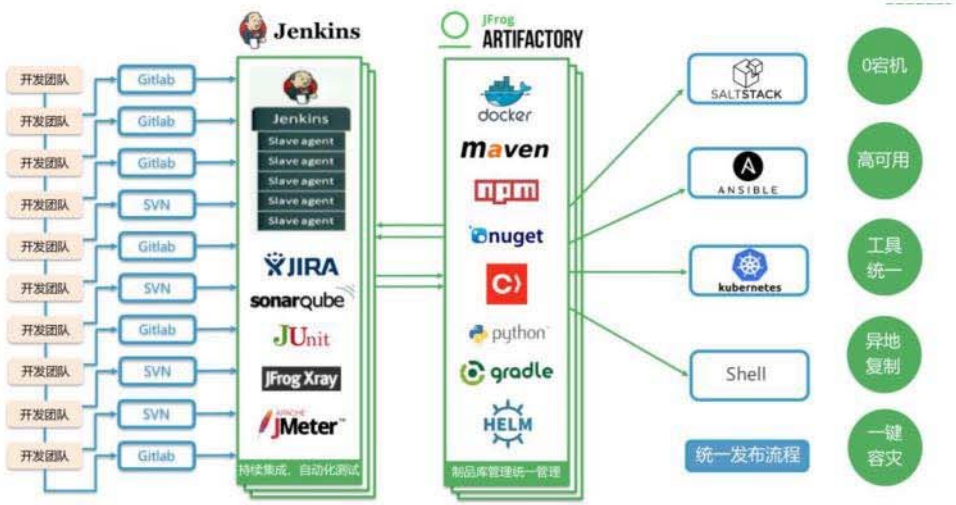
流程：自动、复用


Start
Confirm deployment
Set Namespaces
Git Checkout
Remove if exists
Setup K8s
Check resources
Get IPs
Setup
Check for latest version
Deploy
Connection tests
Artifactory setup
Access federation
Xray setup
Mission Control setup
End
Australia
India
US
Artifactory is happily serving 7 artifacts
Welcome to JFrog Xray

流程：制品的统一管理

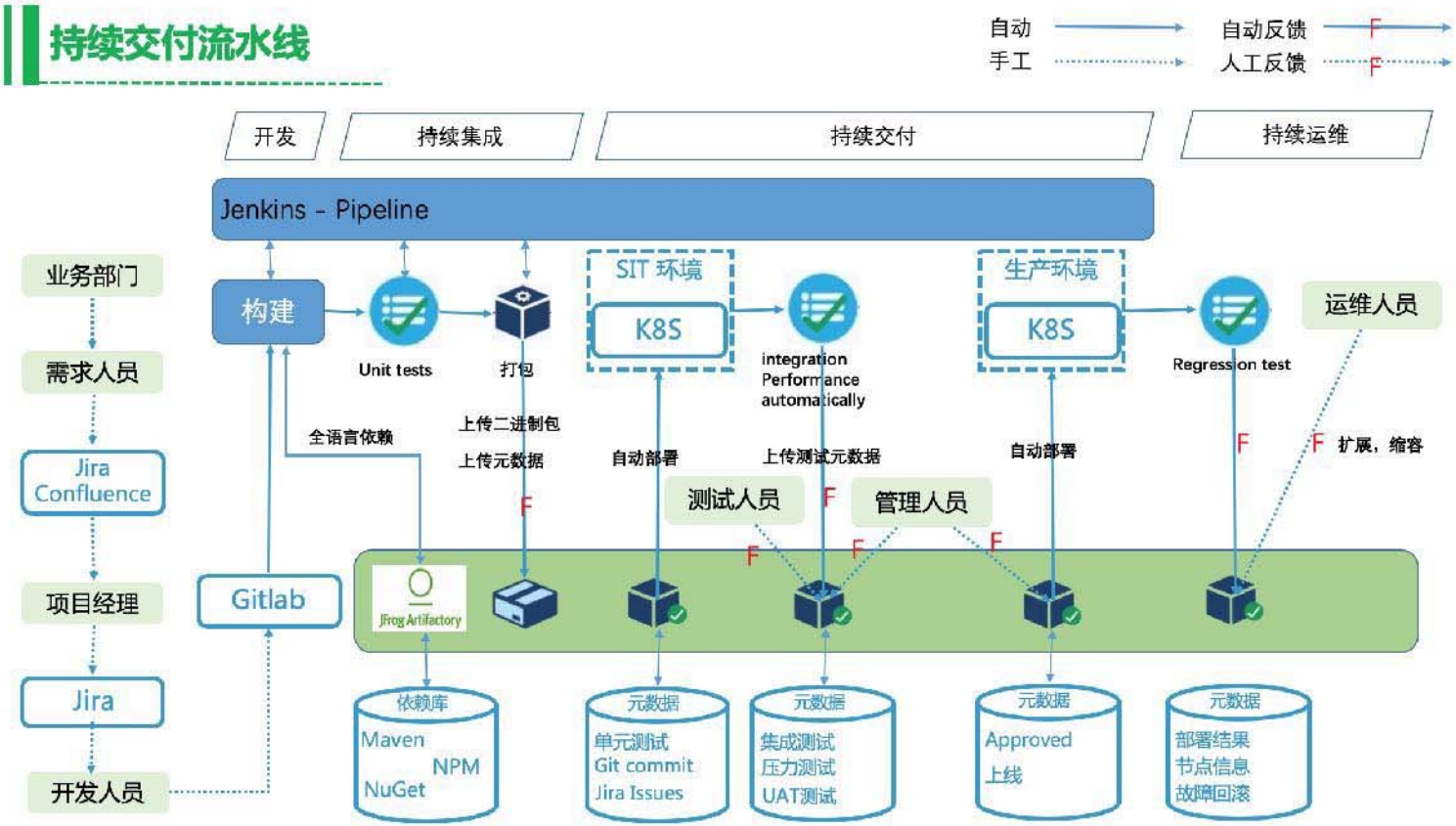

Helm Chart
Docker Image
Docker Image
Docker Image
My App
App Packages
OS - Debs, rpm, Yum
OS
（操作系统）
App Packages
（应用包）
My App
（我的应用）
Docker Image
（镜像）
Helm
（容器编排）
Kubernetes Registry
JFrog Artifactory
for
DevOps
fpm
debian
Maven
NuGet
npm
git
HELM

流程：信息的统一管理

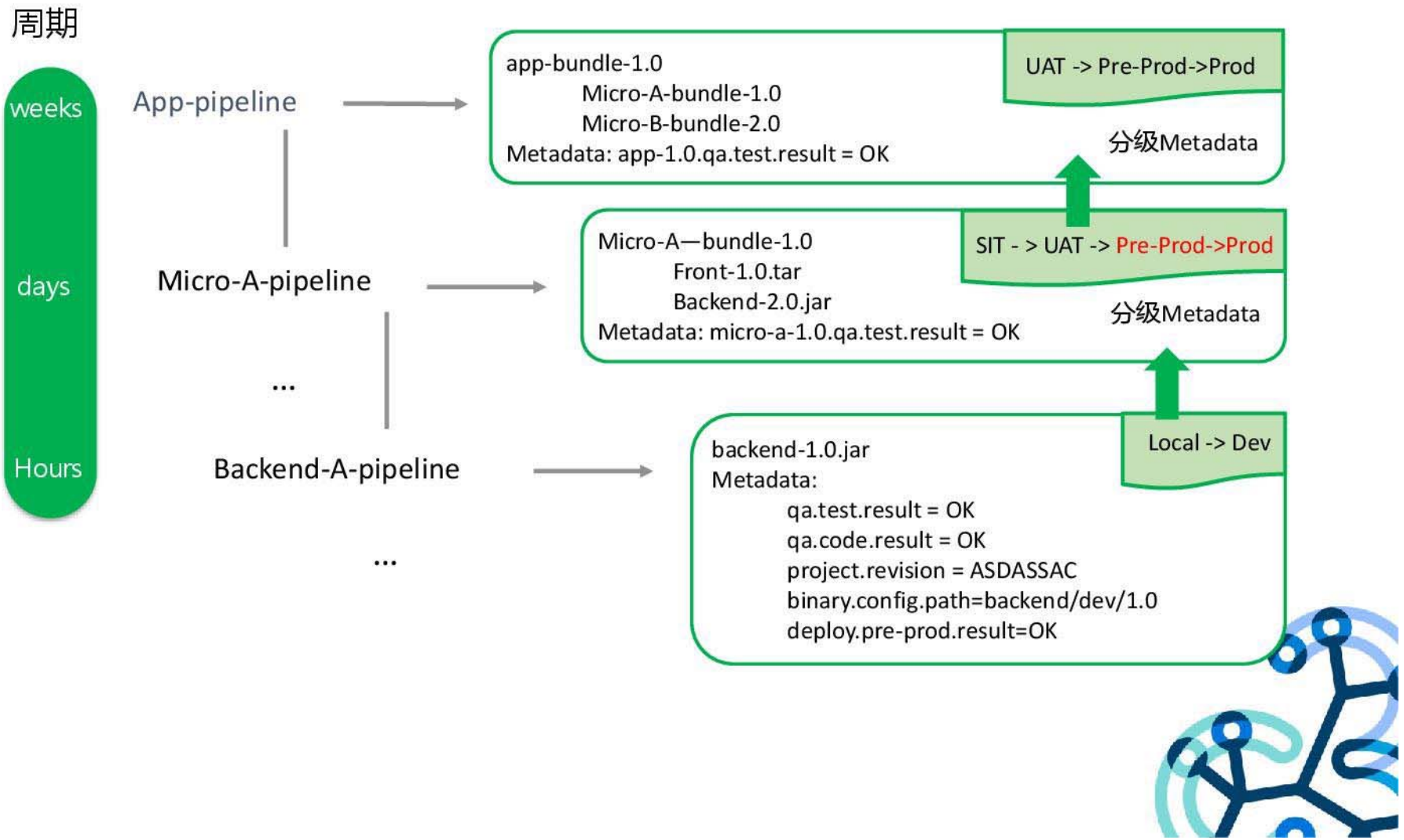

Jenkins
JFrog ARTIFACTORY
开发团队
Gitlab
SVN
Jenkins
Slave agent
JIRA
sonarqube
JUnit
JFrog Xray
JMeter
持续集成，自动化测试
docker
maven
npm
nuget
python
gradle
HELM
制品库管理统一管理
SALTSTACK
ANSIBLE
kubernetes
Shell
统一发布流程
0宕机
高可用
工具统一
异地复制
一键容灾


JFrog Artifactory
JUnit
sonarqube
JIRA
kubernetes
ANSIBLE
需求
提交者、分支
测试关键信息
部署关键信息
1
2
3
4
5

流程：持续交付


持续交付流水线
自动
手工
自动反馈
人工反馈
F
开发
持续集成
持续交付
持续运维
Jenkins - Pipeline
业务部门
需求人员
Jira Confluence
项目经理
Jira
开发人员
构建
Unit tests
打包
全语言依赖
上传二进制包
上传元数据
Gitlab
JFrog Artifactory
SIT 环境
K8S
自动部署
integration
Performance
automatically
上传测试元数据
测试人员
管理人员
生产环境
K8S
自动部署
Regression test
运维人员
扩展，缩容
依赖库
Maven
NPM
NuGet
元数据
单元测试
Git commit
Jira Issues
元数据
集成测试
压力测试
UAT测试
元数据
Approved
上线
元数据
部署结果
节点信息
故障回滚

流程：分级持续交付


周期
weeks
days
Hours
App-pipeline
Micro-A-pipeline
...
Backend-A-pipeline
...
app-bundle-1.0
Micro-A-bundle-1.0
Micro-B-bundle-2.0
Metadata: app-1.0.qa.test.result = OK
UAT -> Pre-Prod->Prod
分级Metadata
Micro-A—bundle-1.0
Front-1.0.tar
Backend-2.0.jar
Metadata: micro-a-1.0.qa.test.result = OK
SIT - > UAT -> Pre-Prod->Prod
分级Metadata
backend-1.0.jar
Metadata:
qa.test.result = OK
qa.code.result = OK
project.revision = ASDASSAC
binary.config.path=backend/dev/1.0
deploy.pre-prod.result=OK
Local -> Dev

流程：应用监测

需要新的监测手段

- ssh不管用了
- 直接用kubectl调试也是不合适的
- Dev/Ops应该能方便地访问可信的数据
  - 监视
  - 日志
- 基于可控的OOB（Out-of-Band）工具
- 微服务监测的基本原则

（https://thenewstack.io/five-principles-monitoring-microservices）

流程：应用监测

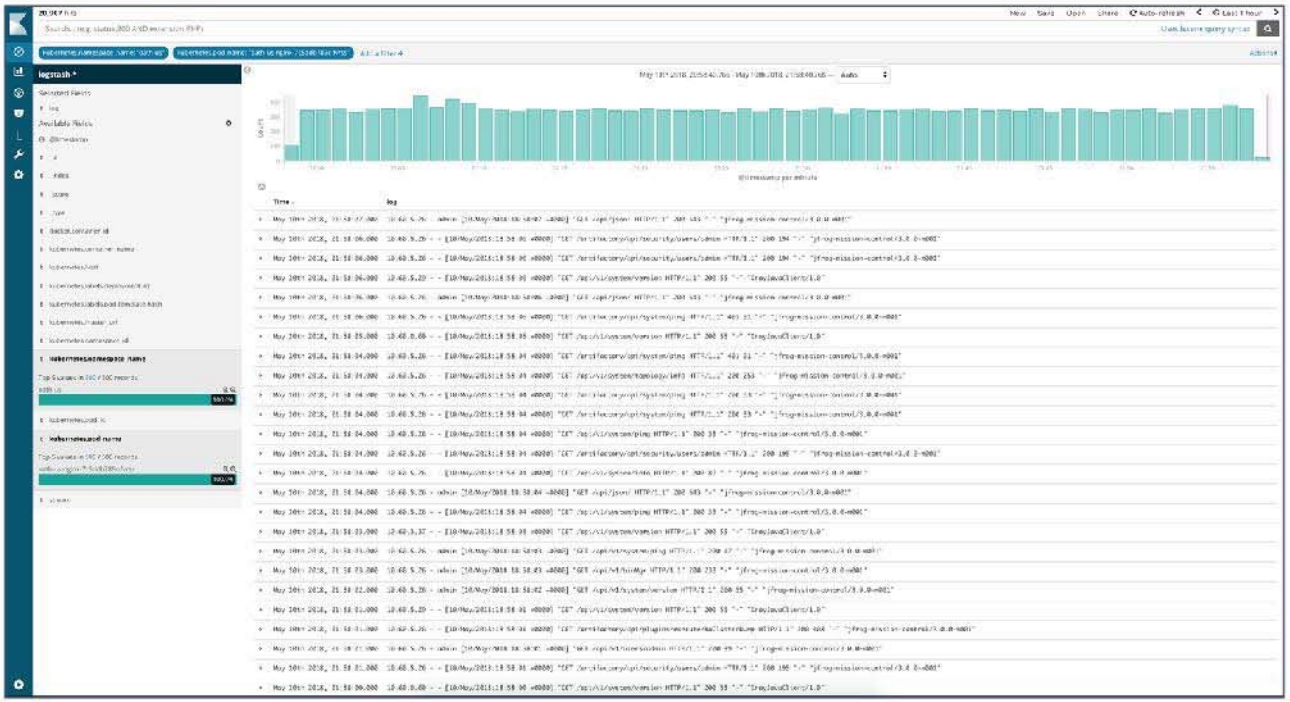
监视

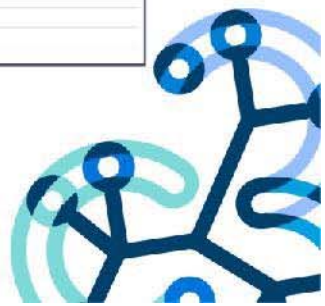
- Prometheus
- Graphana

流程：应用监测

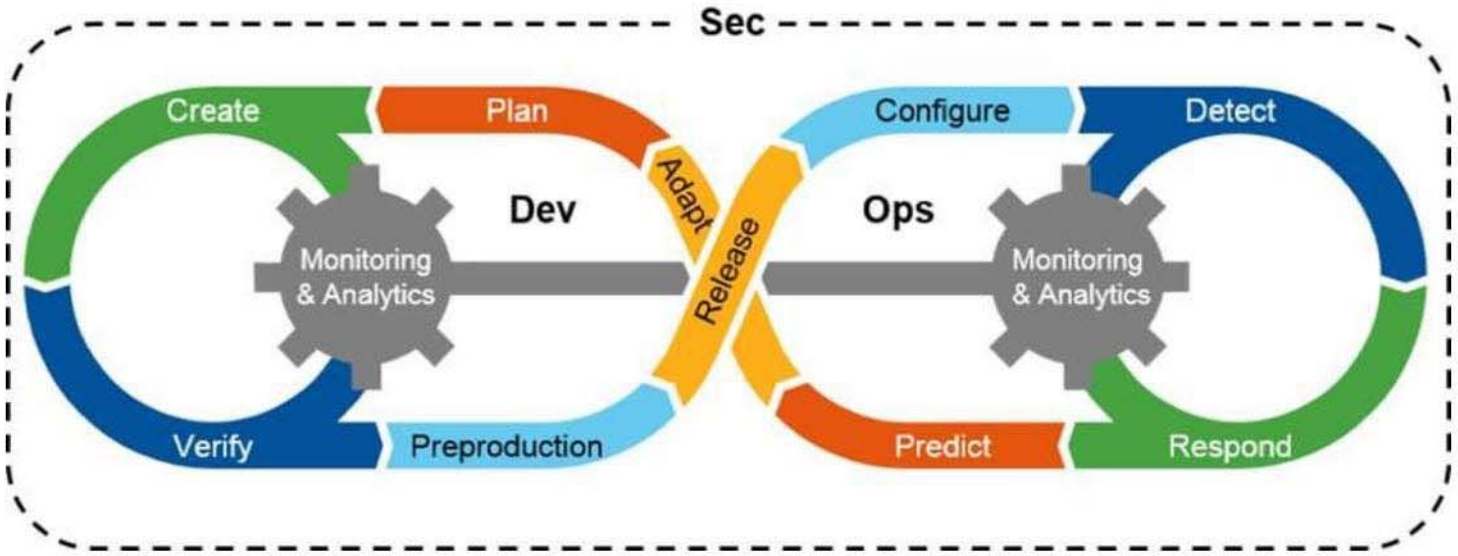
日志

oEFK

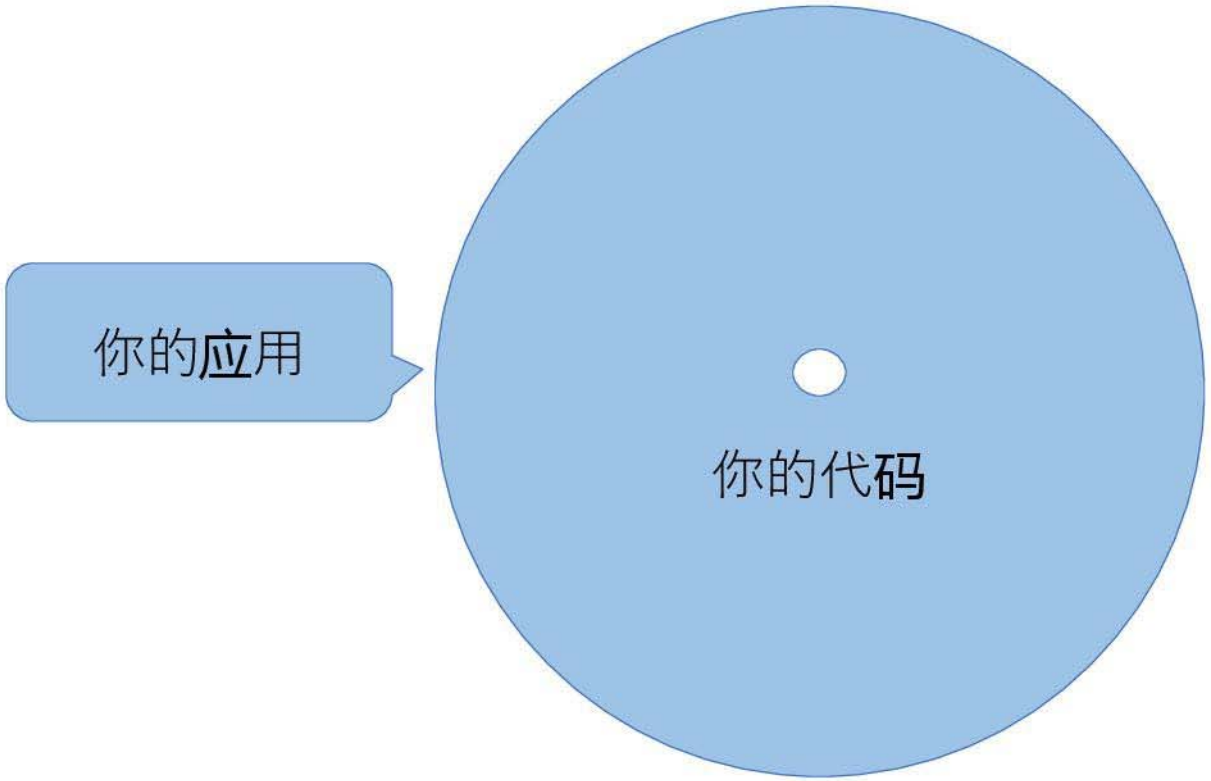




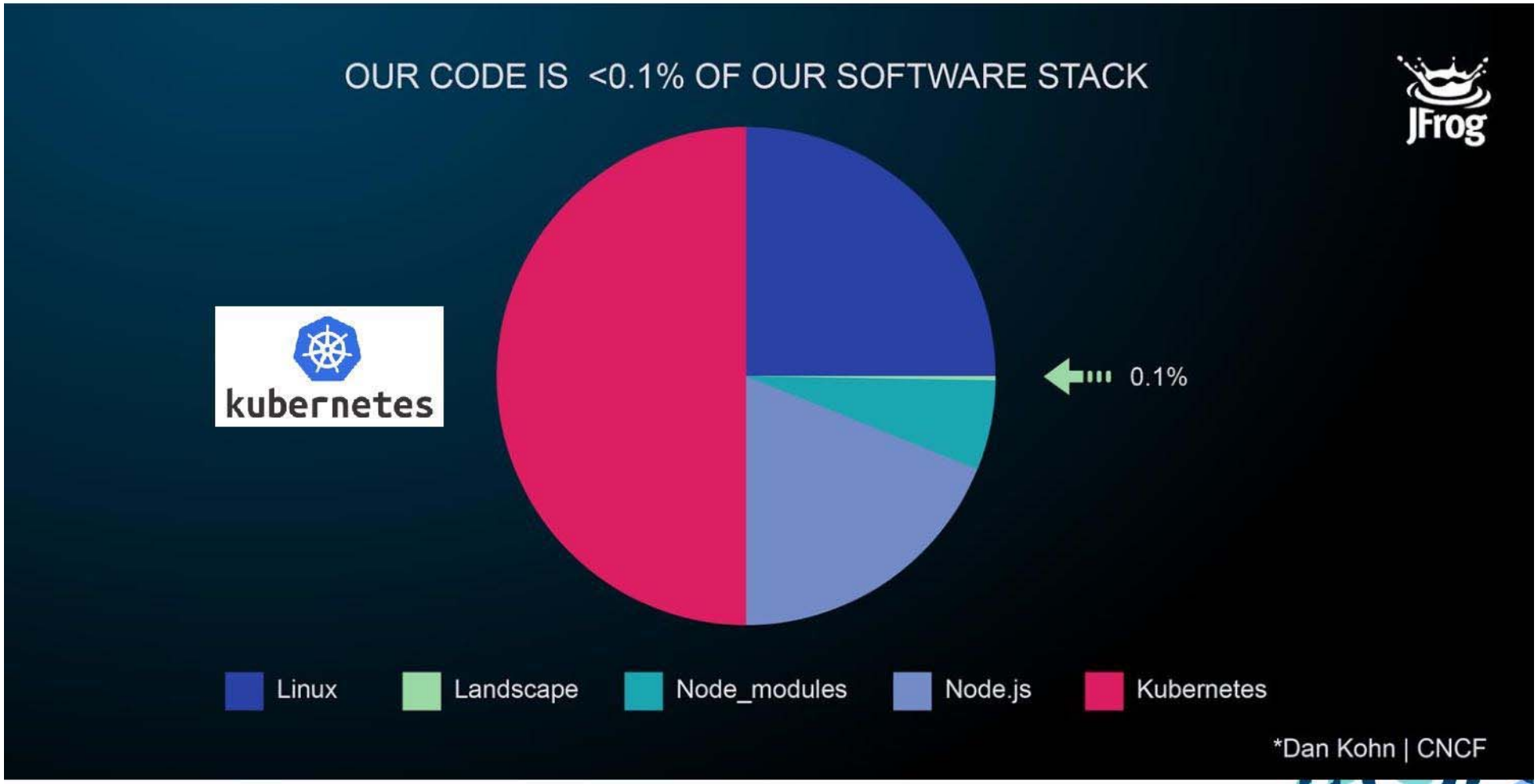
安全：DevSecOps



安全：DevSecOps



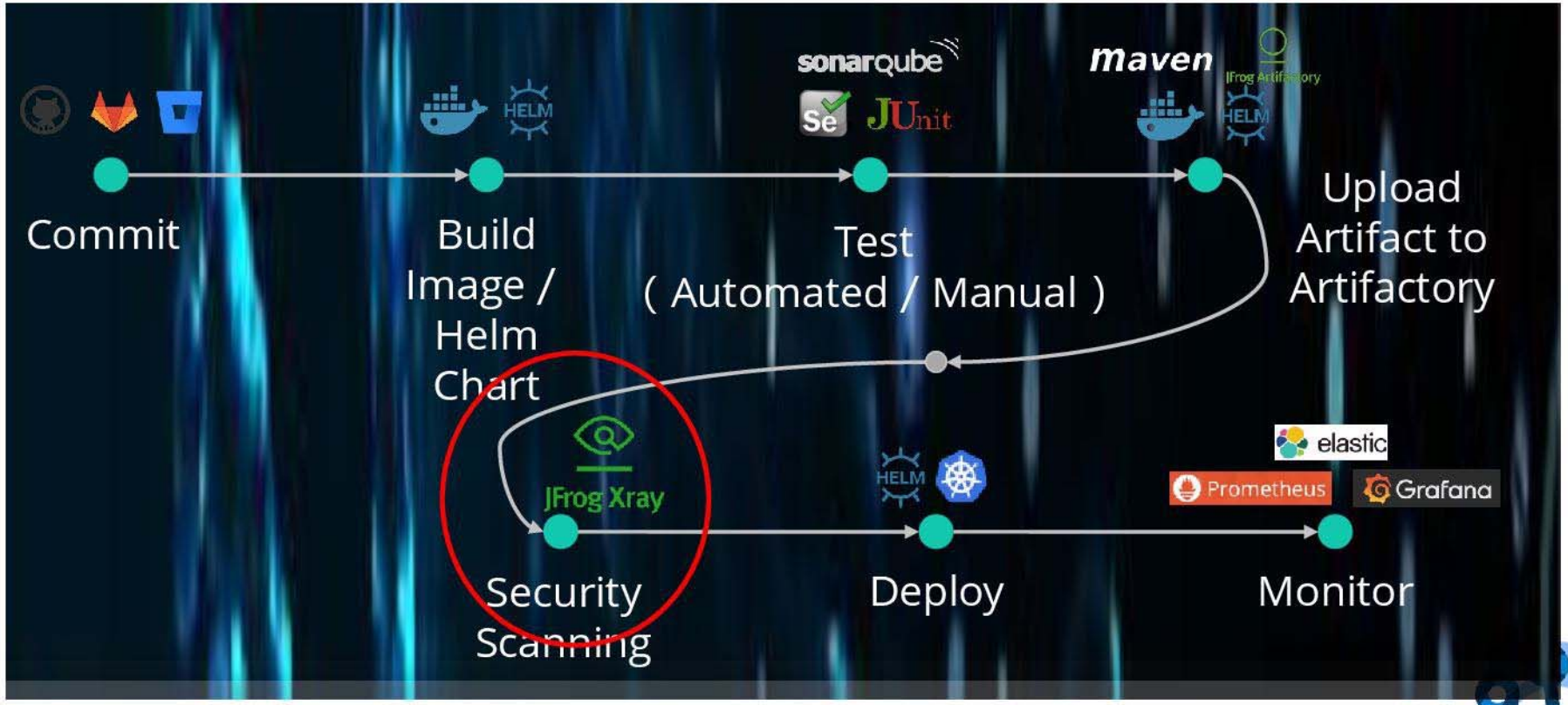
安全：DevSecOps



安全：外部依赖的安全漏洞

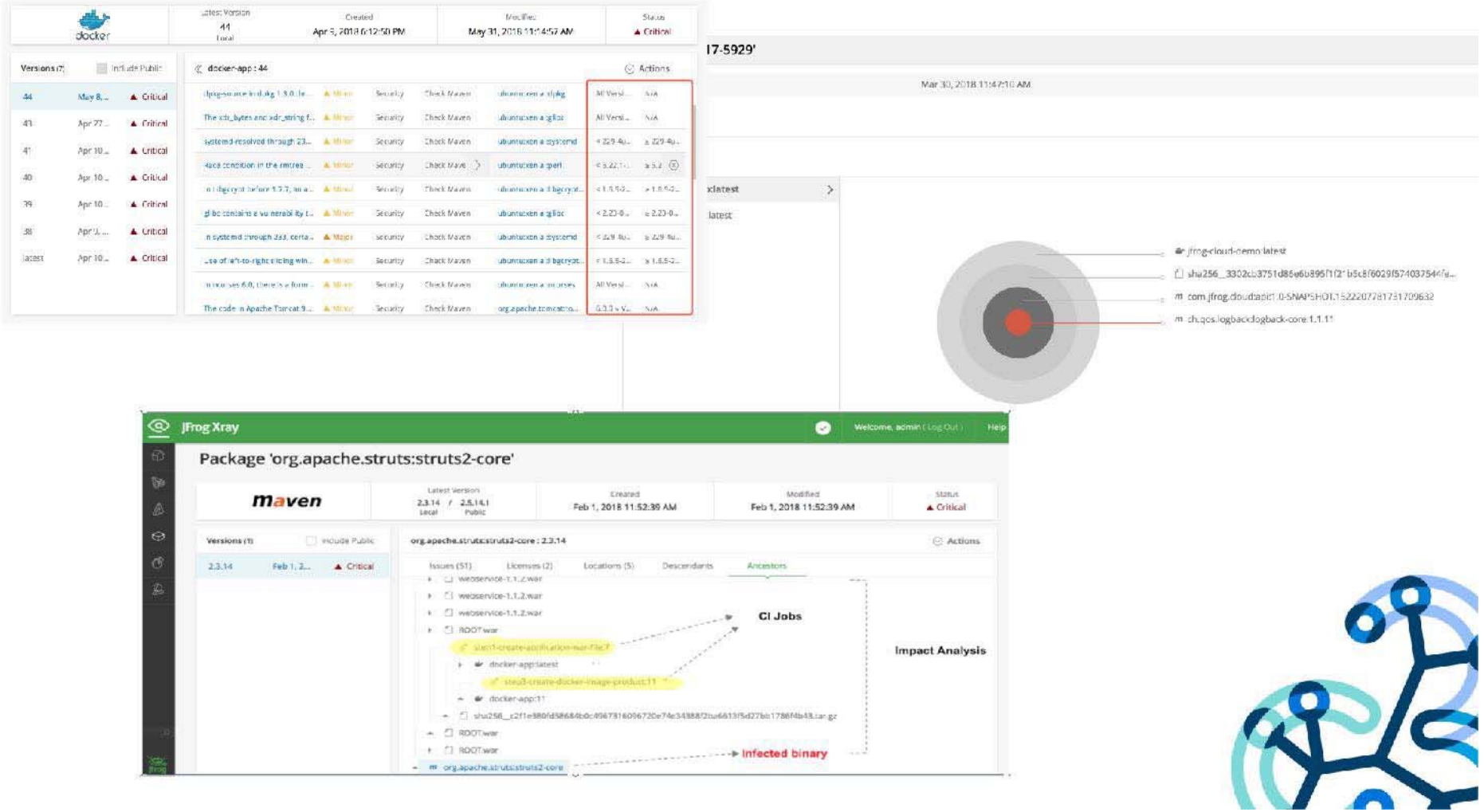
- Docker
  - 30%的DockerHub上的镜像有安全漏洞
- NPM
  - 14%的NPM包有安全漏洞
- Maven
  - 59%的已知安全漏洞还没有解决
  - MTTR(平均修复时间)：290天
  - CVSS10: 265天

安全：DevOpsSec



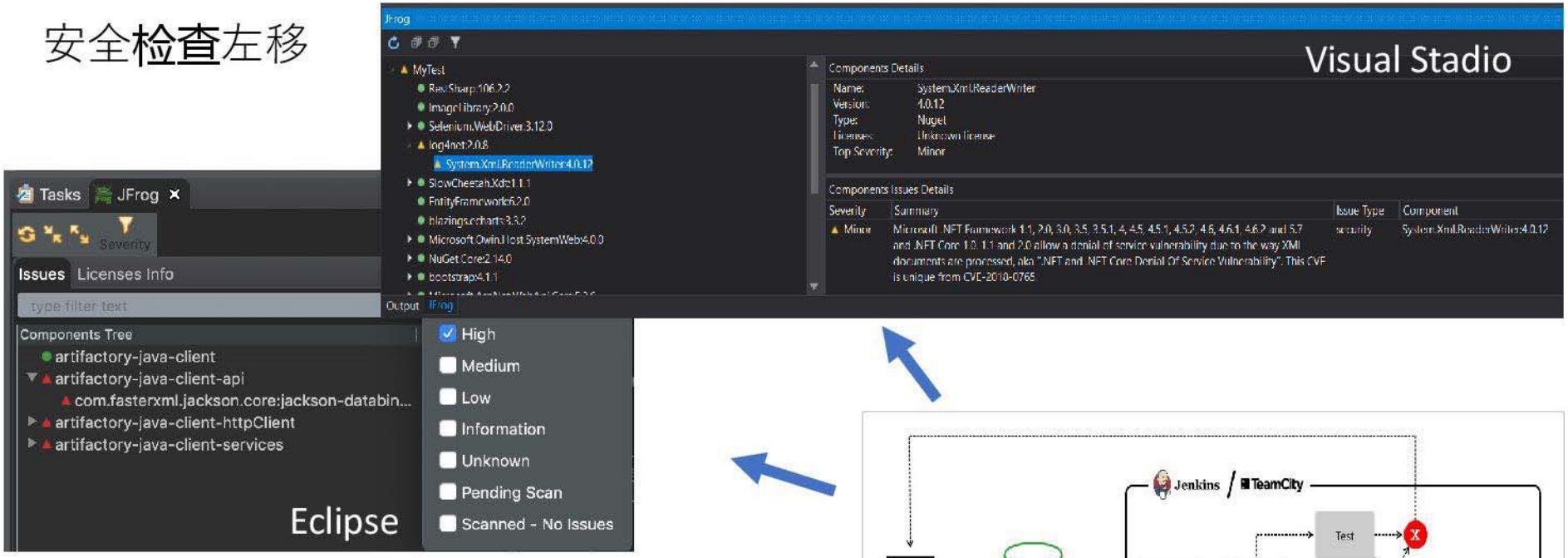
安全：DevOpsSec

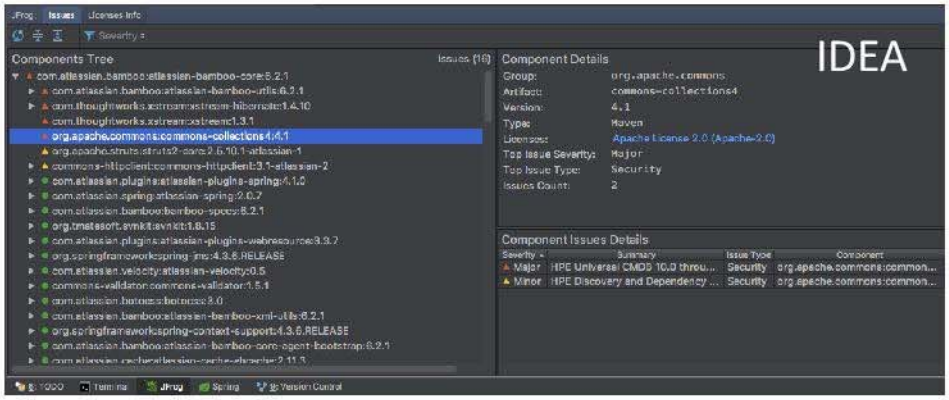
扫描、定位、分析

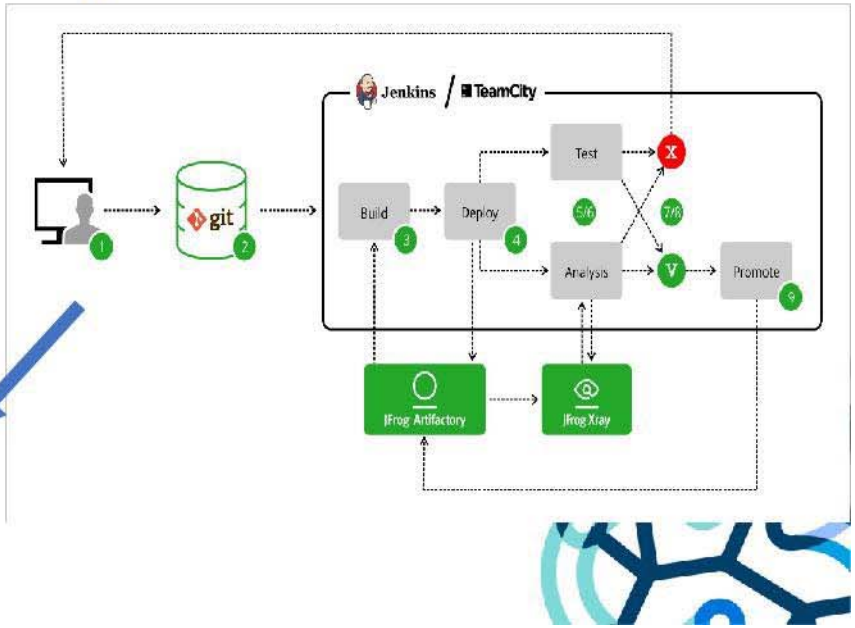


安全：DevOpsSec

安全检查左移







案例启示

Kubernetes尚未成功。。。

- 起步：熟悉Kubernetes
- **规**划：面向Kubernetes的改造
- **编**排：Helm Charts
- 部署：自**动**、**监测**
- 安全：DevSecOps

GIAC

主办方：msup | 架構 ARCHNOTES

关注msup微信公众账号

获取更多技术实践干货

关注高可用架构公众账号

改变互联网的构建方式