

DevOps

遗留系统的持续集成流水线改造

刘斌 2017.6.22

关于DevOps

- 传统运维

- 传统运维方式

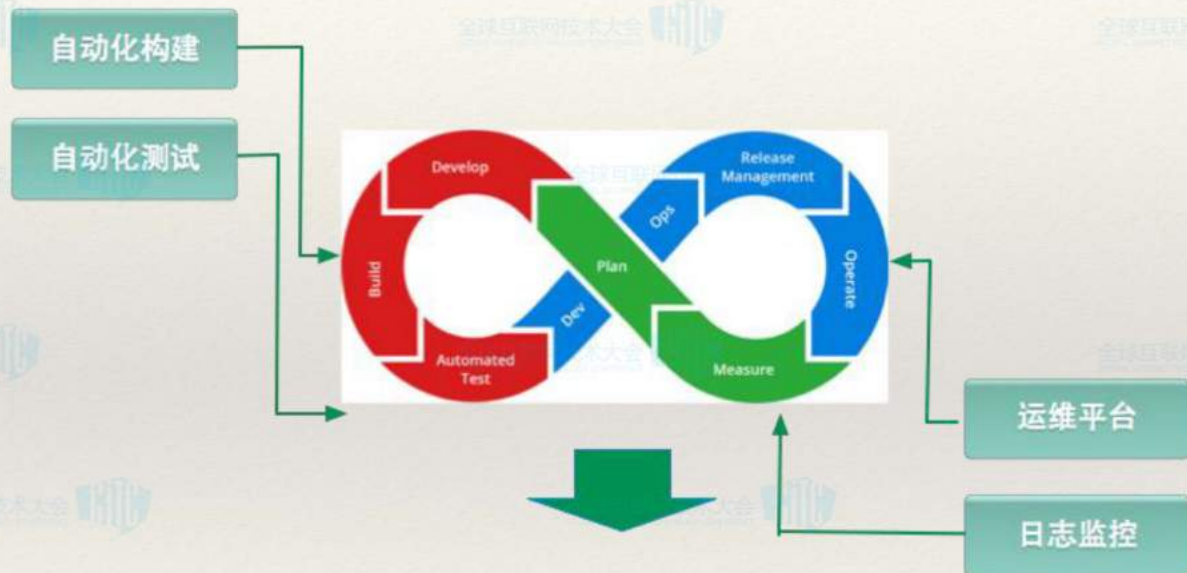
- 人肉运维、面向运维操作的工单
 - 开发和运维之间的沟壑
 - 互相不懂对方的苦
 - 一个优秀的团队，不应该有苦逼的角色
 - 认命吧？

- 期望达到

- 开发自运维

- 自己把握节奏，自己挖的坑，自己含着泪给填上
 - 快速的产品迭代和交付过程
 - 围绕应用生命周期的活动

关于DevOps

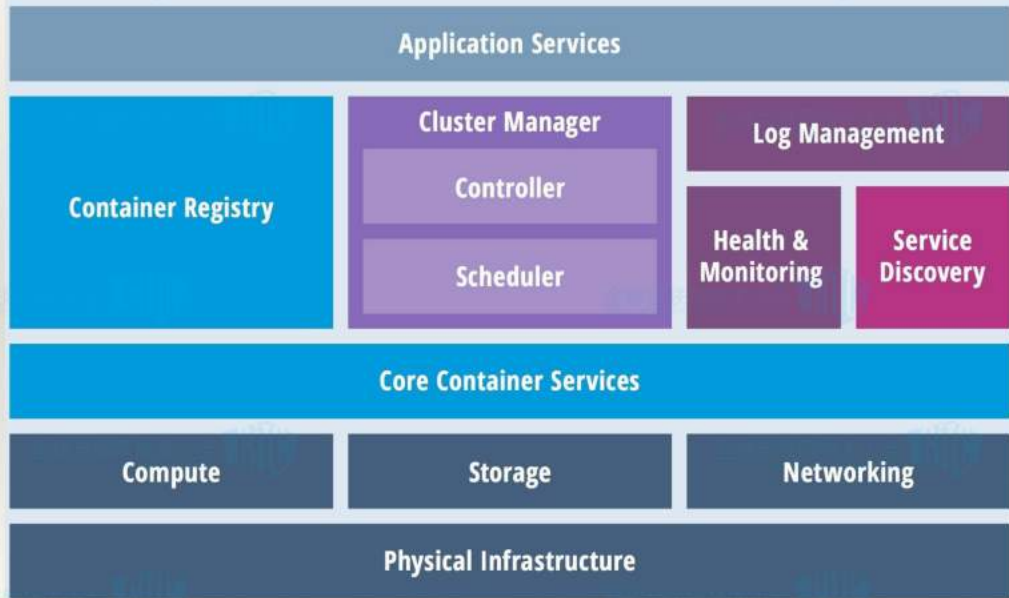


CI Continuous Integration

CD Continuous Delivery

DevOps工具集

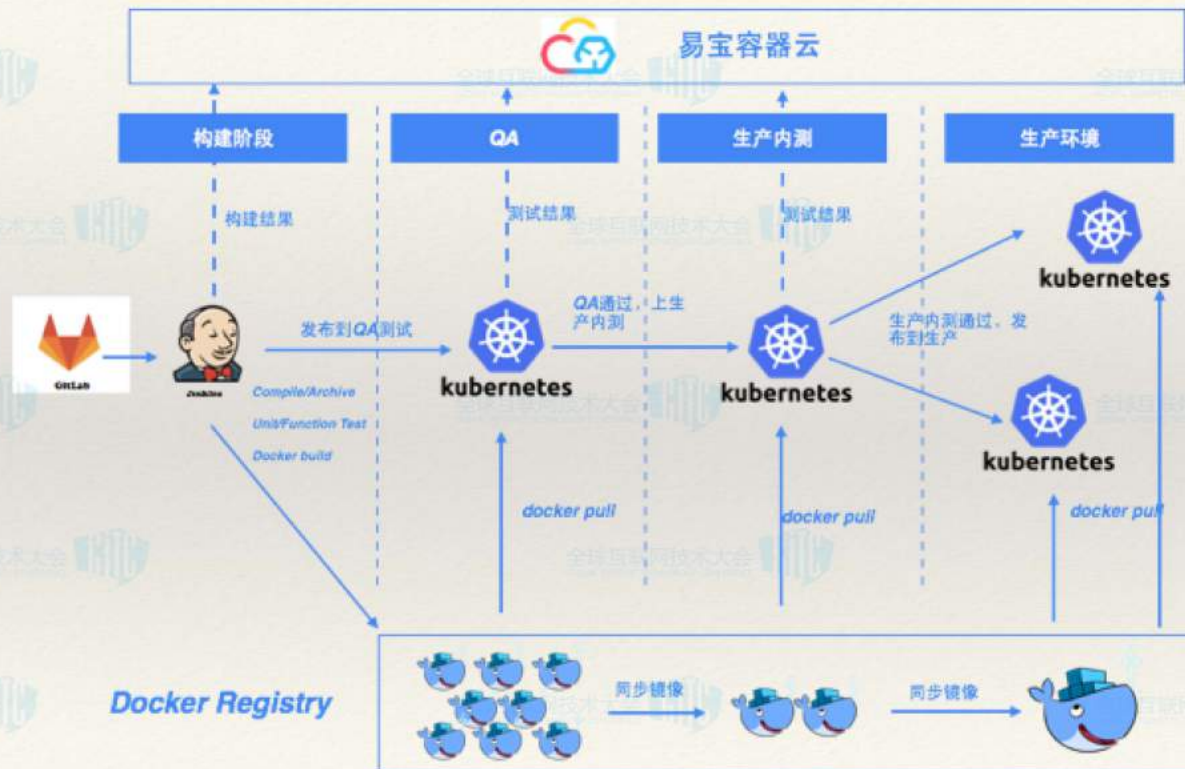
Containers as a Service Architecture



容器云架构



部署流水线



构建方案要解决的问题

- 一次构建所产生的二进制包可在：测试、生产环境运行，不需要多次构建；
- 可以回滚到之前任意版本；
- 整个构建过程的可追溯性：从二进制包追溯到源代码版本、构建任务号、数据库版本（甚至自动化测试脚本版本）；
- 项目依赖的基础组件（如日志组件）、web容器（比如tomcat），能够控制版本，并可控的分批升级；

配置和制品分离

- 一次构建所产生的二进制包可在：测试、生产环境运行，不需要多次构建；

```
<execution>
  <id>copy-resources</id>
  <phase>compile</phase>
  <goals>
    <goal>copy-resources</goal>
  </goals>
  <configuration>
    <encoding>UTF-8</encoding>
    <outputDirectory>${project.build.directory}</outputDirectory>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
        <includes>
          <include>*.conf</include>
          <include>log4j.xml</include>
          <include>*.properties</include>
        </includes>
        <filtering>true</filtering>
      </resource>
```

```
<profiles>
  <profile>
    <id>develop</id>
    <properties>
      <dbuser>xxxxx</dbuser>
      <dbpwd>xxxxxx</dbpwd>
    </properties>
  </profile>
</profiles>
```

传统做法(但没解决问题)

配置和制品分离

- ❖ 从开发阶段就将配置分离出来
- ❖ 配置信息集中管理，或许能带来额外的好处~

单一版本的问题

❖ 1.0-SNAPSHOT（开发、测试）—> 1.0（生产）

❖ 可以回滚到之前任意版本？

- 把二进制包保存下来，可以保证回滚到任意版本，但更希望是能从源代码构建出想要的版本。

❖ 多版本——各自指定自己依赖的版本？

- 版本战线太长，技术负债

依赖管理

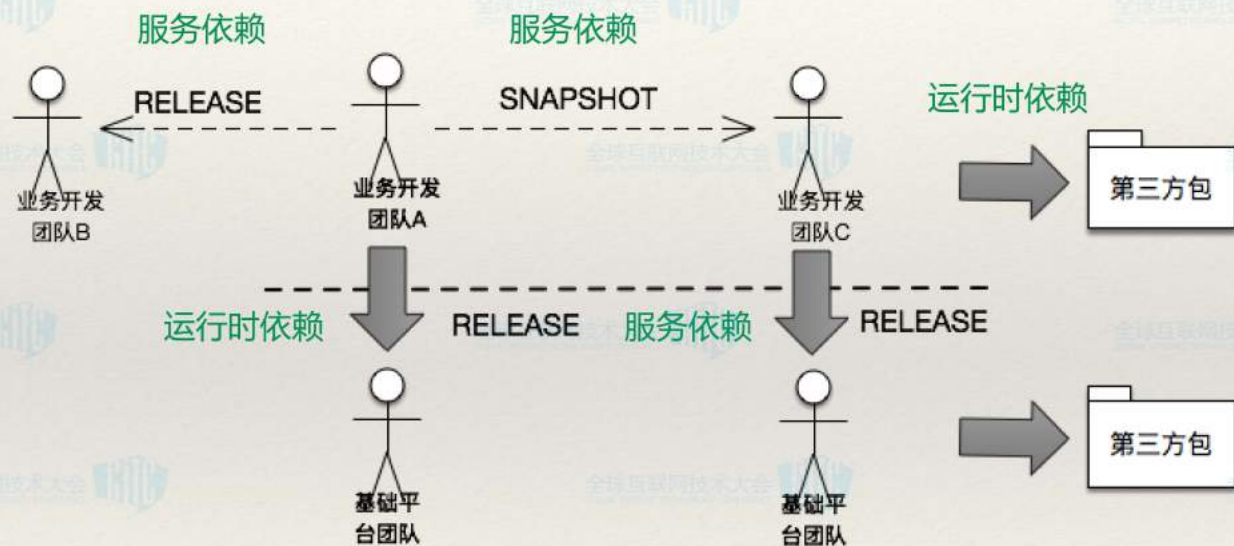
A \longrightarrow **B**

- A的最新版依赖B的最新版（粗糙的管理）
- A只允许依赖B的通过测试的版本（精细的管理）
- 依赖传递、拉构建、推构建.....可能想多了

依赖的管理

- **理想的解决办法**：代码提交触发构建，之后通过推或拉的方式，触发其它工程构建及测试
 - 需要把构建的验证、自动化测试，覆盖的比较完整和合理，否则就像过于敏感的报警信息一样，总提示集成失败，也没人搭理了
 - 虚拟机部署的情况下，需要大量资源（容器下，则不是问题）
 - 解决依赖传递的问题（都需要解决）
- **折中的解决办法**：手动触发构建，最新版就依赖最新版——但还得区分版本号，为了回滚

依赖管理



依赖包版本

xxxx-hessian-1.0-SNAPSHOT
xxxx-facade-1.0-SNAPSHOT
xxxx-core-1.0-SNAPSHOT

— dependency —

xxxx-1.1-SNAPSHOT
xxxx-2.1.3
xxxx-2.0-SNAPSHOT

可统一指定版本
`mvn versions:set`

指定使用发布版

`mvn versions:use-releases`

如无发布版，对于facade包，可以发布一个

`mvn versions:set`
`mvn deploy`

对所有使用snapshot版本的尝试更新为release版
`mvn versions:use-release`

检查依赖冲突
`mvn`
`com.ning.maven.plugins:maven-dependency-versions-check-plugin:check`

检查依赖包版本号是否符合要求
`mvn`
`com.ning.maven.plugins:maven-dependency-versions-check-plugin:list`

部署测试

发布
`mvn deploy`

配置与制品包分离

src/main/resources

/dev
conf.properties
/test
conf.properties
/templ
context.conf

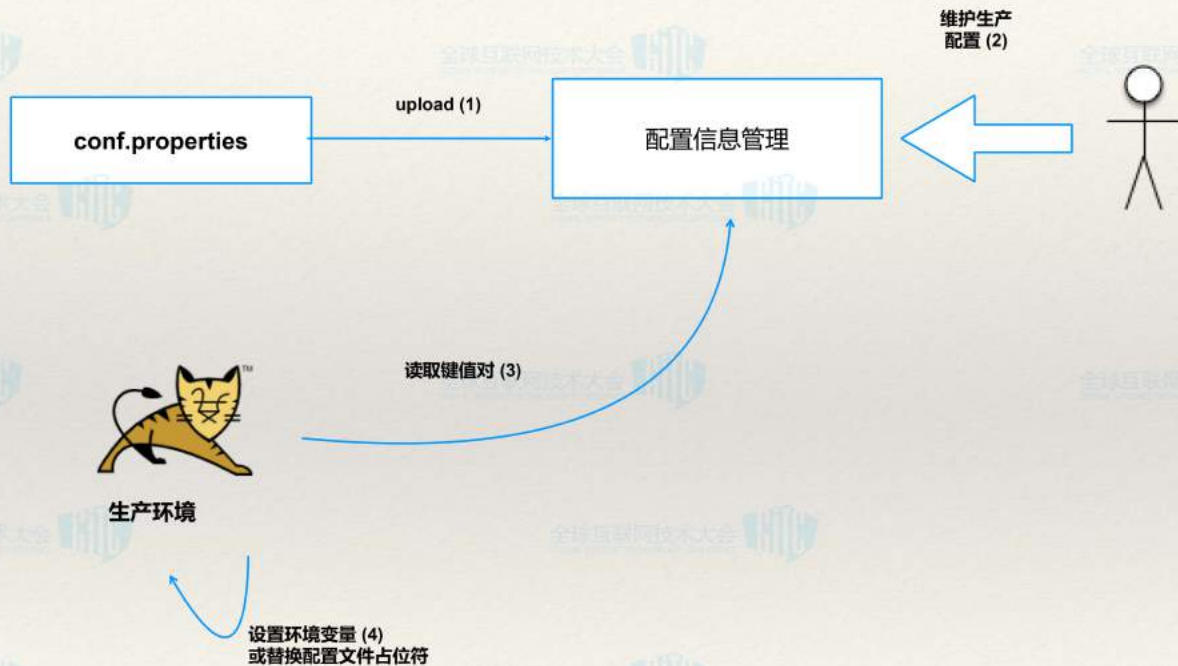
db.schema=open
db.user=share

\${db.schema}

```
<plugin>
  <groupId>it.ozimov</groupId>
  <artifactId>yaml-properties-maven-plugin</artifactId>
  <version>1.1.1</version>
  <executions>
    <execution>
      <phase>initialize</phase>
      <goals>
        <goal>read-project-properties</goal>
      </goals>
      <configuration>
        <files>
          <file>src/main/resources/${env}/conf.properties</file>
        </files>
      </configuration>
    </execution>
  </executions>
</plugin>
```

```
<profiles>
  <profile>
    <id>develop</id>
    <properties>
      <env>dev</env>
    </properties>
  </profile>
</profiles>
```

配置与制品包分离

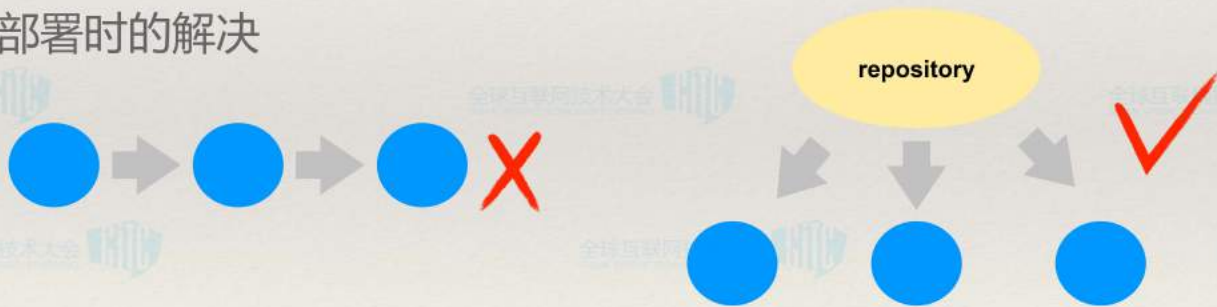


包版本的可追溯性

- 在部署的二进制包中打入信息，可以关联到源代码版本号、构建任务的版本号、二进制自身的版本

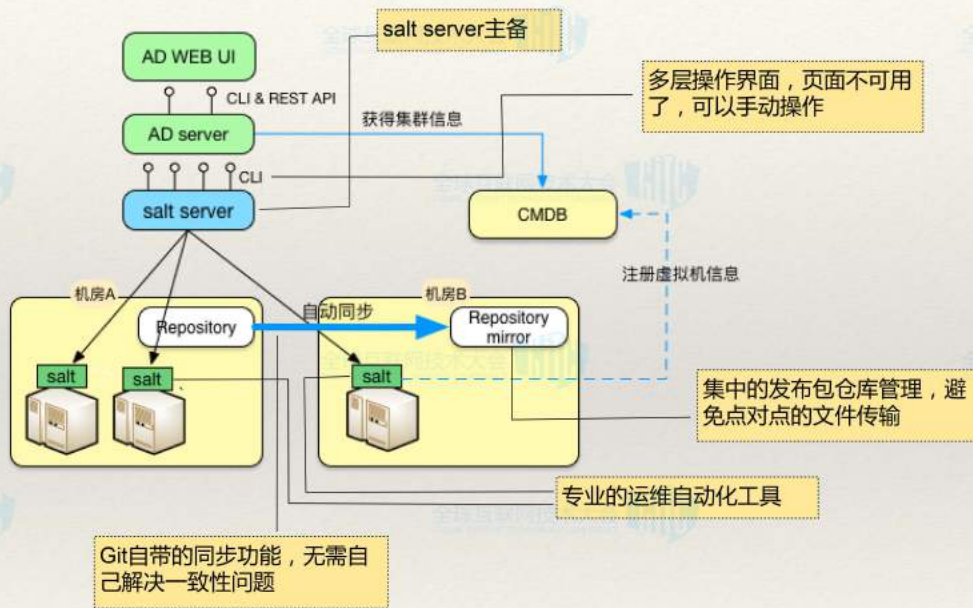
包的一致性问题

- 为什么会有一致性问题？
 - 从构建环境向生产环境同步；pool下多个节点的同步；跨机房的同步
 - 为了自证清白
- 部署时的解决

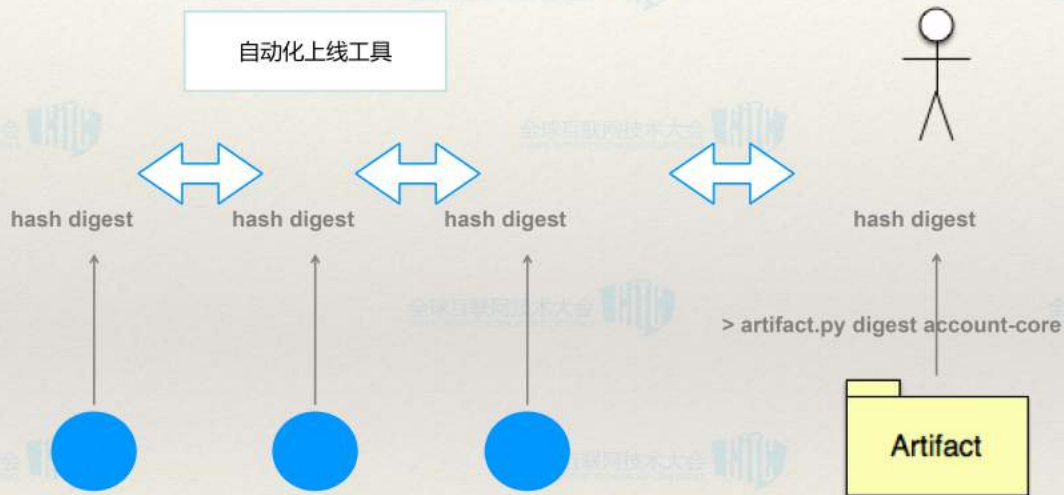


- 校验哈希值 (Merkle tree)

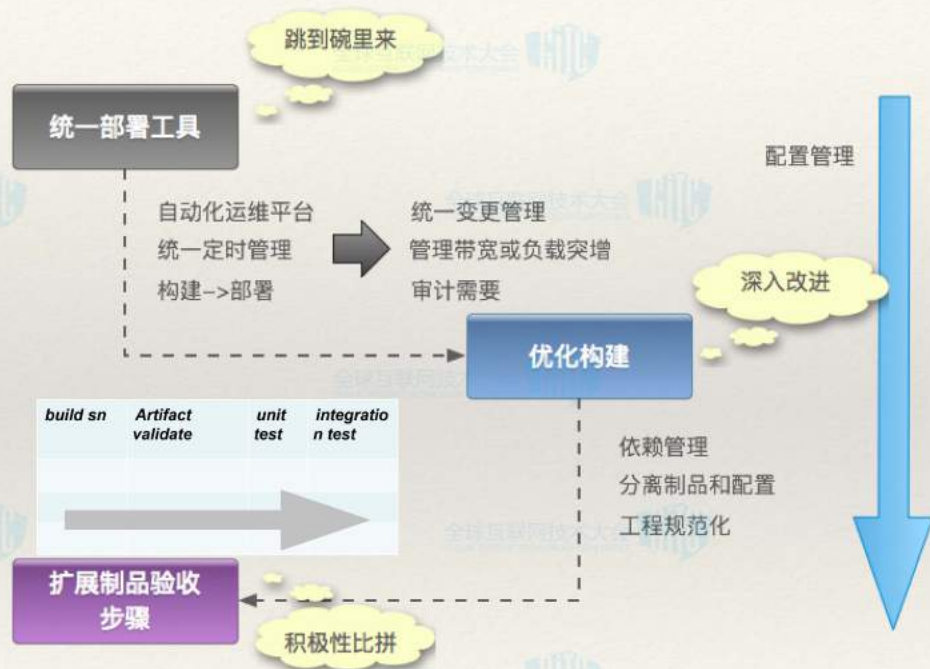
部署



一致性校验



推动迁移的策略





谢谢各位！

