日常开发的痛点

本地能跑，服务器上有问题？

这个问题的答案很简单，那就是环境配置不统一。

- 系统的环境变量问题

```
root@poc:~# export
declare -x DBUS_SESSION_BUS_ADDRESS="unix:path=/run/user/0/bus"
declare -x HOME="/root"
declare -x LANG="en_US.UTF-8"
declare -x LESSCLOSE="/usr/bin/lesspipe %s %s"
declare -x LESSOPEN="| /usr/bin/lesspipe %s"
declare -x LOGNAME="root"
declare -x LS_COLORS="rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;3
declare -x MOTD_SHOWN="pam"
declare -x OLDPWD
declare -x PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bi
declare -x PWD="/root"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x SSH_TTY="/dev/pts/0"
declare -x TERM="xterm"
declare -x USER="root"
declare -x XDG_RUNTIME_DIR="/run/user/0"
declare -x XDG_SESSION_CLASS="user"
declare -x XDG_SESSION_ID="14825"
declare -x XDG_SESSION_TYPE="tty
```

- 不同机器上，资源文件不一样

1. .env 场景：一些本地开发使用的变量文件，通常不会上传到git仓库
2. package-lock.json 场景：前端开发过程中，本地锁定了版本
3. 字体文件 场景：合同需要宋体
4. node、java、dotnet.....版本不一致 场景：自己在电脑上升级过版本，但是服务器没升级呀
5. .....

- 网络配置，例如hosts文件

```
root@poc:~# cat /etc/hosts
127.0.0.1    localhost

10.0.1.152    poc    poc
```

测试环境好好的，生产环境就不对？

这里拿java举例，问题分析：

```
# 生产环境的jar包
md5sum msb-edu-exam-prod.jar
b04e8fb3afe6fde9079b6d271821a880  msb-edu-exam-prod.jar

# 预发环境的jar包
md5sum msb-edu-exam-beta.jar
608a878657f0d551f457443c2de3a4a5  msb-edu-exam-beta.jar
```

从上面可以看到jar包不一致。为什么呢？

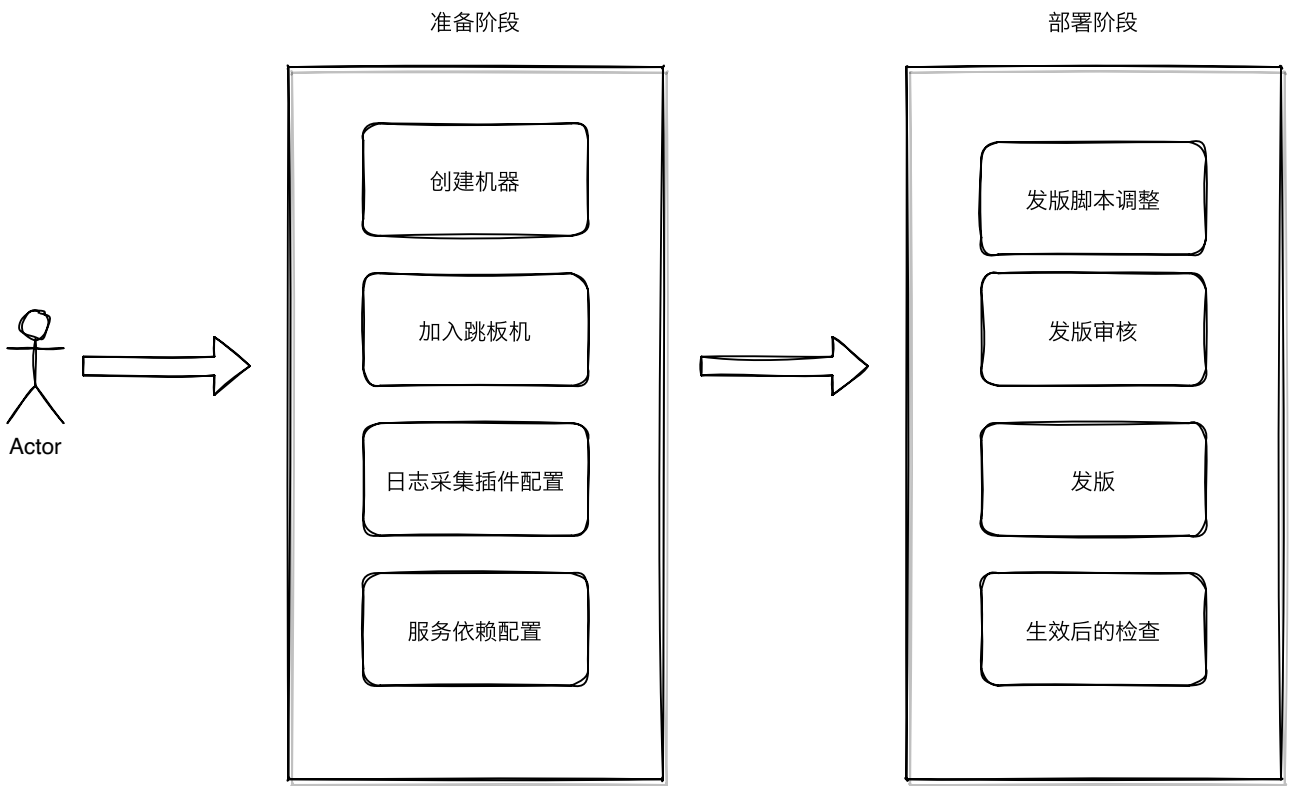
1. 明明用的是同样的分支
2. jdk已经精确到了小版本
3. maven也精确到了小版本
4. maven源也一致
5. .....

原因就是：每一次打包，生成出来的代码都有可能不一样。因为在一些文件中会记录编译时的timestamp，导致难以确定哪些文件一致。

那么应该怎么做呢？一次打包，处处运行！

```
# 通过命令行参数，来覆盖jar包内的默认配置
java -jar app.jar --spring.profiles.active=prod
```

业务高峰，扩容速度慢？



其中很多内容可以自动化，但是自动化依赖一个功能全面的自研系统。
例如：

1. 我们可以通过ansible批量操作，可是每个服务的依赖项呢？需要一个地方存储
2. 发版流程是通过固定分支构建的，可这时候分支代码有变化怎么办？需要一个灵活的发版流程
3. 需要维护ansible脚本，那么就需要较多人力成本。包括了开发、测试、验证、信息维护过程中的沟通成本
4. 一些网络因素、api限速、也会有影响

从这里可以看出：我们需要以应用为维度来管理，而不是具体的物理资源。

团队规范与k8s

统一概念与规范

– 操作流程规范
上线？下线？扩容？回滚？

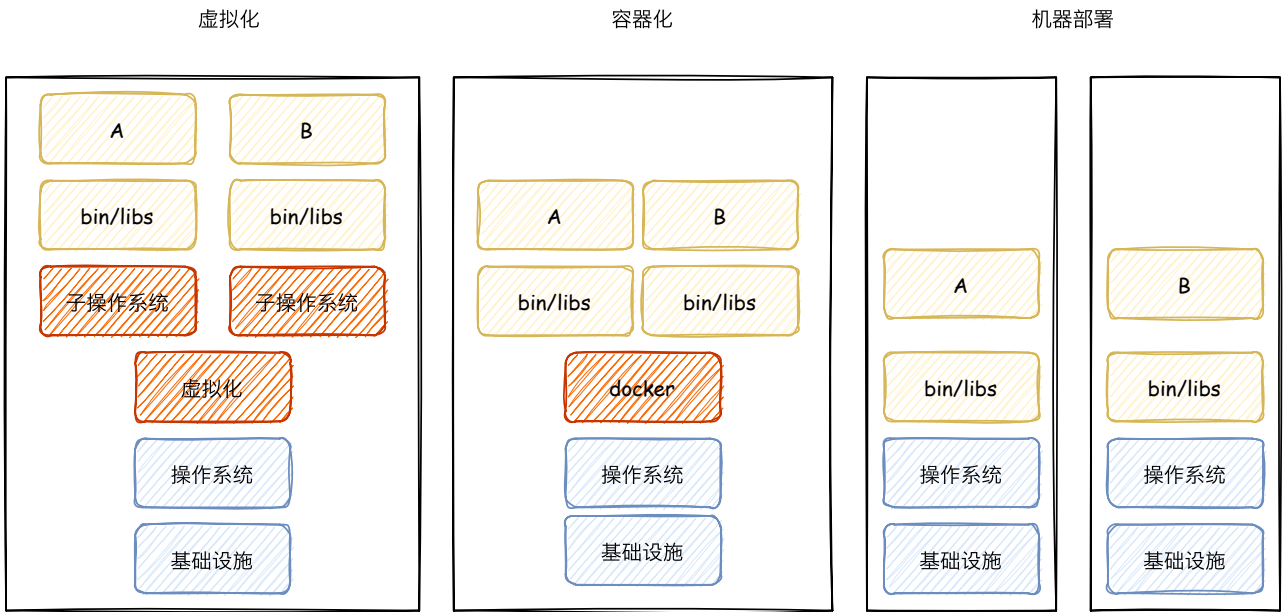
– 日志规范
日志格式的统一、日志内容的统一接入（traceID）

– 统一健康检查方式
所有应用都可以遵循的健康检查规范、程序自定义实现

– 明确的弹性伸缩规则
定时扩缩容？cpu/内存扩缩容？沉默时间？

– git工作流、代码覆盖率等等.....

docker容器隔离



声明式api

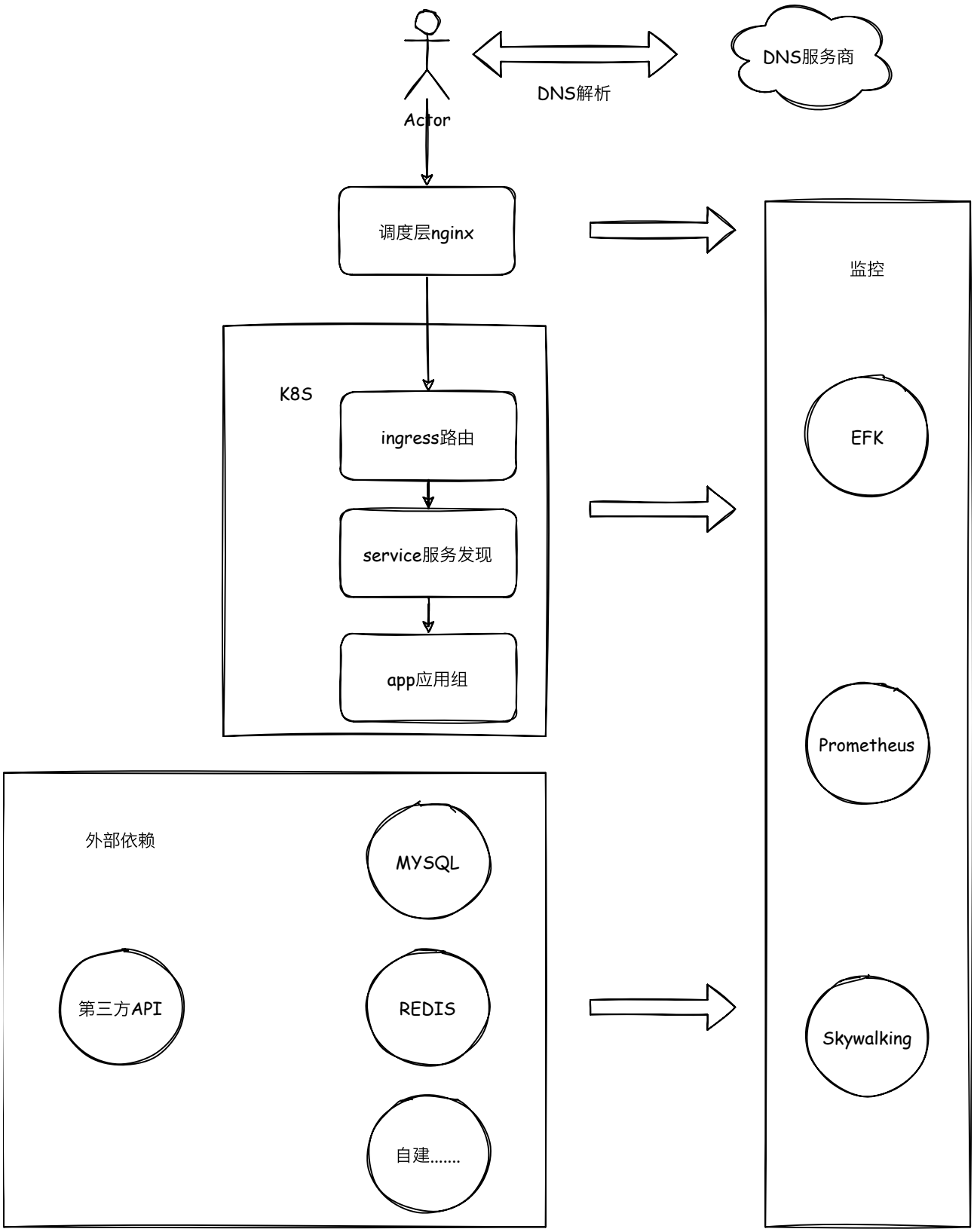
版本1	版本2

```
kind: Pod
appName: user
metadata:
  labels:
    name: msb-om-scheduler-deployme
    team: develop
spec:
  replicas: 1
```

```
kind: Pod
appName: user
metadata:
  labels:
    name: msb-om-scheduler-deployme
    team: om
spec:
  replicas: 2
```

1. 程序会对所有的修改进行分析
2. 此案例中检测到了2个变化。team和replicas
3. 我们直接通过修改应用的标签，把user应用迁移到了om团队
4. replicas从1变为2，以为这我们这个应用需要2个容器。于是k8s通过计算，直接新增一个容器

严选网络架构讲解


Actor
DNS解析
DNS服务商
调度层nginx
监控
K8S
ingress路由
EFK
service服务发现
app应用组
Prometheus
外部依赖
MYSQL
第三方API
REDIS
Skywalking
自建.....

什么是一个健康的应用?

通过前面的网络规划路线，我们可以知道开发其实只需要聚焦到应用。

而一个健康的应用，应该要做到如下几点：

1. 支持健康检查（deployment的健康巡检）

```
  ...................
          livenessProbe:
            failureThreshold: 2
            httpGet:
              path: /xx-service/health
              port: port
              scheme: HTTP
            initialDelaySeconds: 60
            periodSeconds: 30
            successThreshold: 1
            timeoutSeconds: 1
  ...................
```

2. 内存使用的稳定（prometheus监控）
3. 响应速度（skywalking的P99）
4. 规范的日志记录（efk的日志查询）
5. 尽量做到无状态化（docker），任意迁移

结束