云原生

云原生的特点

- 微服务

  非单体应用，拆分为微服务

- 健康报告

  如borg/kubectl下运行的任务都内置了http服务器，用于检测应用现在是否健康——即是否准备好去使用

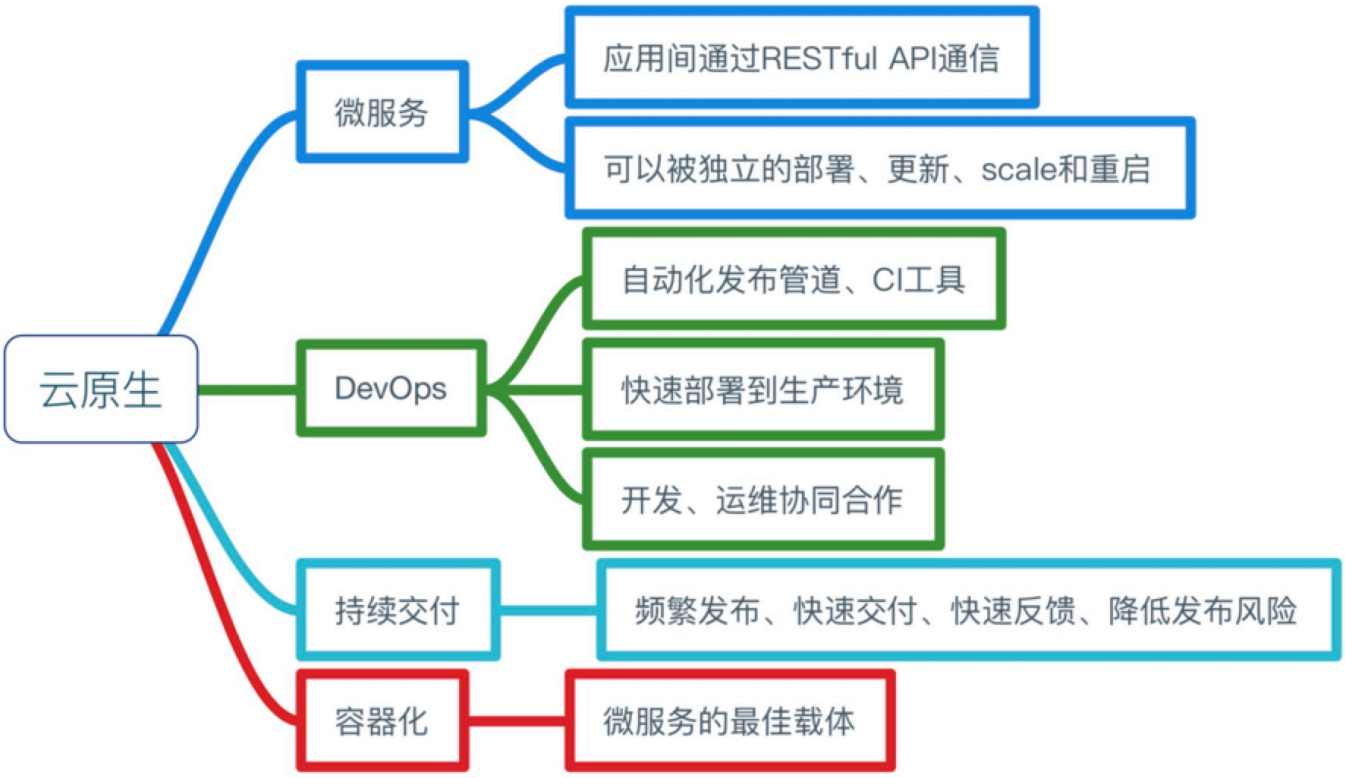
- 遥测数据

  收集应用的量化指标，如

  - 请求率，收到了多少个请求
  - 错误，应用有多少个错误
  - 时间，应用多久回复

- 弹性
  - 为失败而设计，分布式系统往往不可靠
  - 优雅降级，当高负载情况下，要保证应用总能回复response，即使应用挂掉也要有保底的缓存或降级服务
- 声明式，非交互式
  - 交互式，输入一行shell则输出该条指令的结果，即使应用死亡也有可能输出正确结果
  - 声明式，区别于shell的脚本，声明式信任这样一个方式：给期望结果一个声明，应用就一定能完成

云原生的概念



云原生的12 factors+3

1. 基准代码：代码用git管理，打包为docker image放到repository统一管理
2. 依赖：使用包管理工具，如java的gradle maven，go的glide
3. 配置：配置与应用分离
4. 后端服务：计算存储分离降低耦合，如mysql
5. 构建/发布/运行：docker
6. 进程：应用作为无状态的进程，重启后还能恢复到原状态，如不存储session
7. 端口绑定：pod有独立ip，端口不会冲突
8. 并发：容器都是一个进程，可以增加容器的replica实现并发
9. 易处理：快速启动和优雅终止
10. 开发环境和线上环境等价：namespace的资源隔离保证了镜像的使用下， 环境可以轻易复制一套
11. 日志：日志作为事件流使用stout输出，如输出到es保存
12. 管理进程：如kubectl进入到容器内部操作
13. api优先：团队协作，服务间的使用要声明api的规约
14. 监控与告警：应用的性能/健康/日志的监控
15. 认证授权：在应用的上层就做，与服务解耦

1.Codebase	2.Dependency	3.Config	4.Backing services	5.Build,release,run	6.Processes	7.Port binding	8.Concurrency	9.Disposability	10.Dev/Prod parity	11.Logs	12. Admin process
GitHub GitLab TortoiseSVN	maven Gradle	Env ConfigMap	Attached resources	docker	Stateless Share-nothing	Self-contained Port-bound Web process	Scale HPA	Graceful termination	Dev Ops	Event stream Stdout	One-off process kubectl exec

Source http://jimmysong.io

云原生与kubernetes

k8s的资源对象

在k8s中，kubernetes对象是一个重要概念。一个对象描述了几件事：

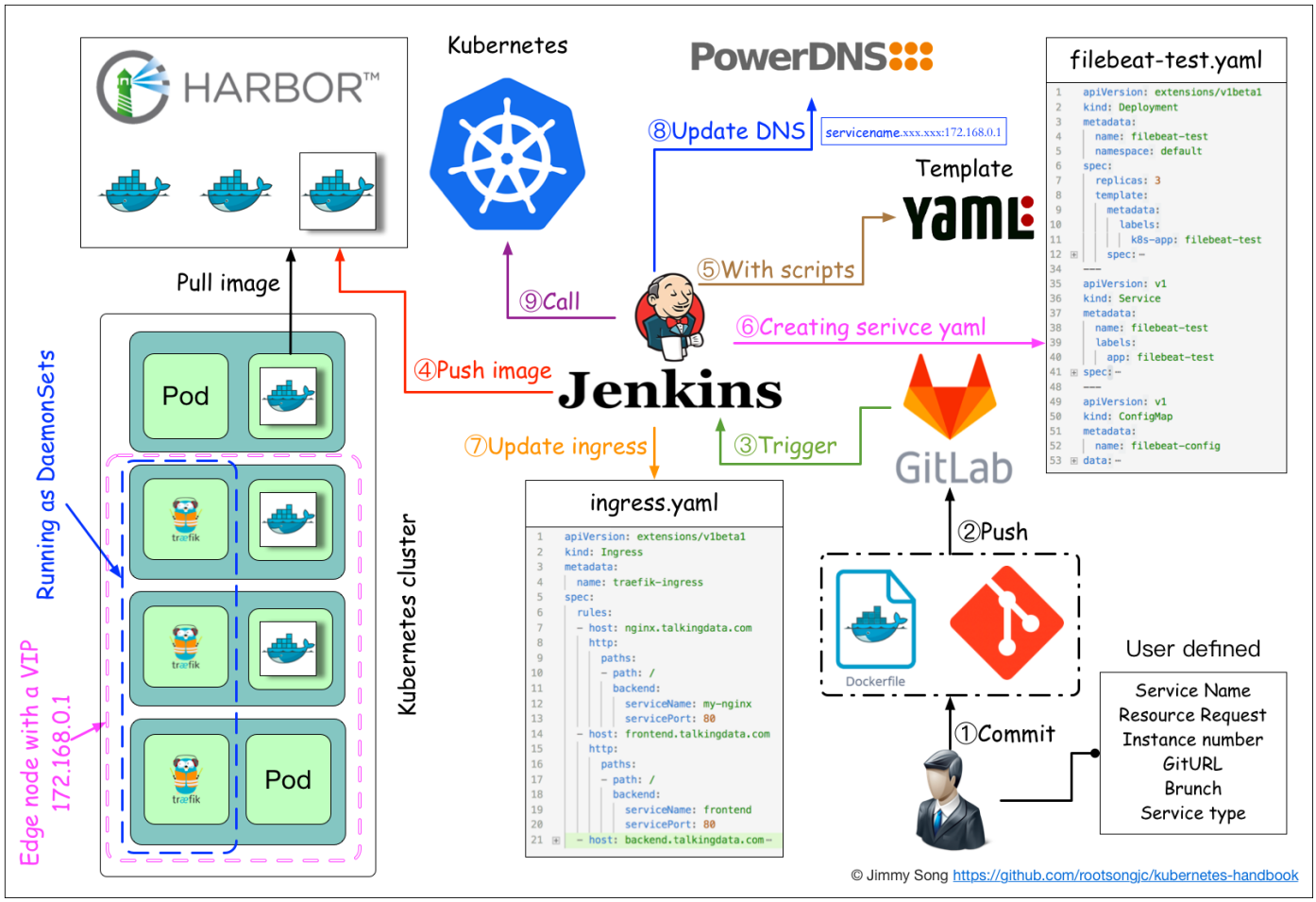
- 什么应用在容器化部署运行，运行在哪个node上
- 资源的描述
- 应用的重启策略/升级策略/容错策略

总体来说，k8s对象是一个声明式的描述，描述了目的——对象被创建后，就会按照声明的spec所运行。这就是k8s的**期望状态**，也即声明式描述

资源配额与限制

- pod级别，最小的资源调度单位
- namespace级别，限制资源隔离的配额

CI/CD



1. 用户提交的代码内含dockerfile
2. push到gitlab
3. 发布应用的时候，填写git repo/branch/service name/实例个数/资源数量等信息，触发jenkins脚本
4. jenkins ci流水线编译打包docker image，push到repository
5. jenkins ci流水线应用脚本，替换里面的kubernetes的yaml模板的入参
6. 生成应用专属的yaml配置
7. jenkins更新ingress的配置，在ingress代表的lb中加入一条路由信息
8. Jenkins更新dns记录，插入一条边缘节点的ip记录
9. jenkins调用kubernetes的api，部署一个应用

总结就是用户提交到gitlab，gitlab触发jenkins去将应用编译打包（ci），之后jenkins更新lb与dns记录，并触发k8s的api完成应用的部署（cd）。

devops

如何践行一个devops？笔者归纳：

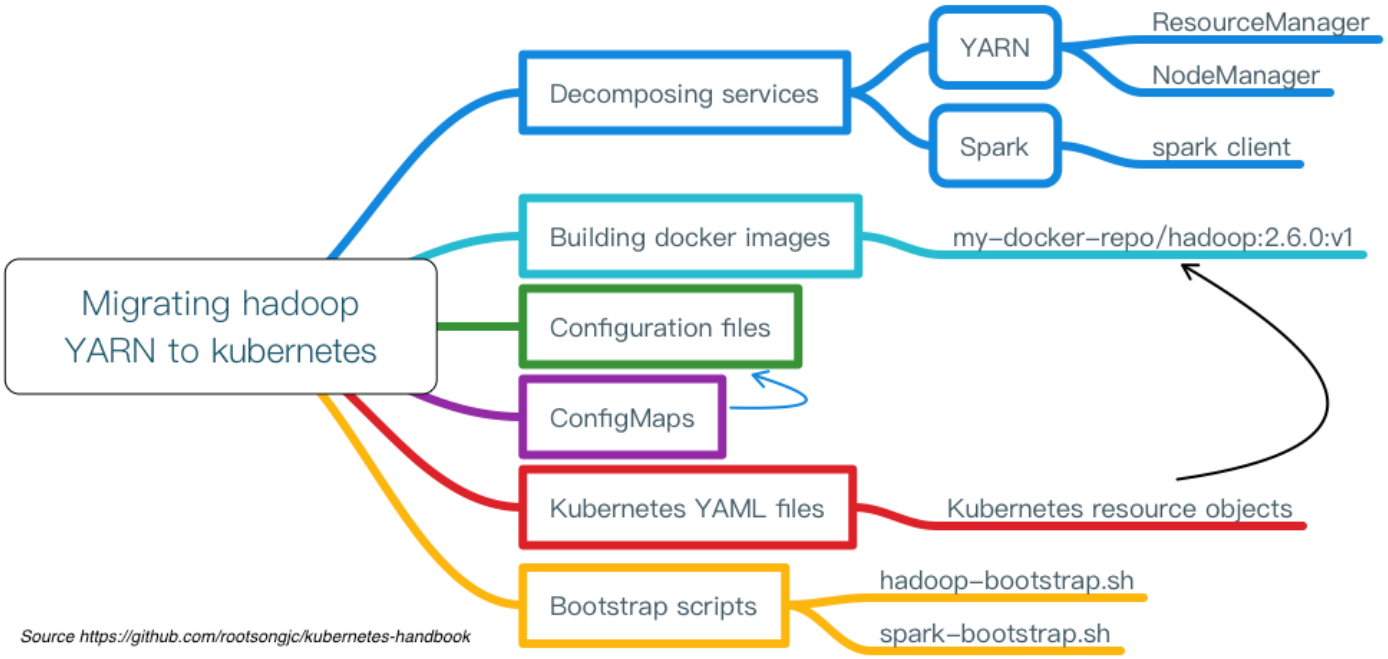
1. 根据环境（开发/测试/生产）划分namespace，也可以根据项目划分
2. 根据用户划分namespace
3. 封装kubectl，比如加上一个用户身份校验
4. 管理员可以通过dashboard查看不同namespace的状态
5. 所有应用的日志推送到es
6. 通过grafana可以查看所有namespace中应用的状态,即集群监控

原文为[k8s handbook:devops行动指南]

cicd和devops是指什么

- cicd
  - continuous integration：持续集成，指coding完项目，pipeline编译打包的过程
  - continuous development：持续部署，指pipeline发布部署的过程
- devops：指开发与运维紧密贴合的部分

如何迁移到云原生架构

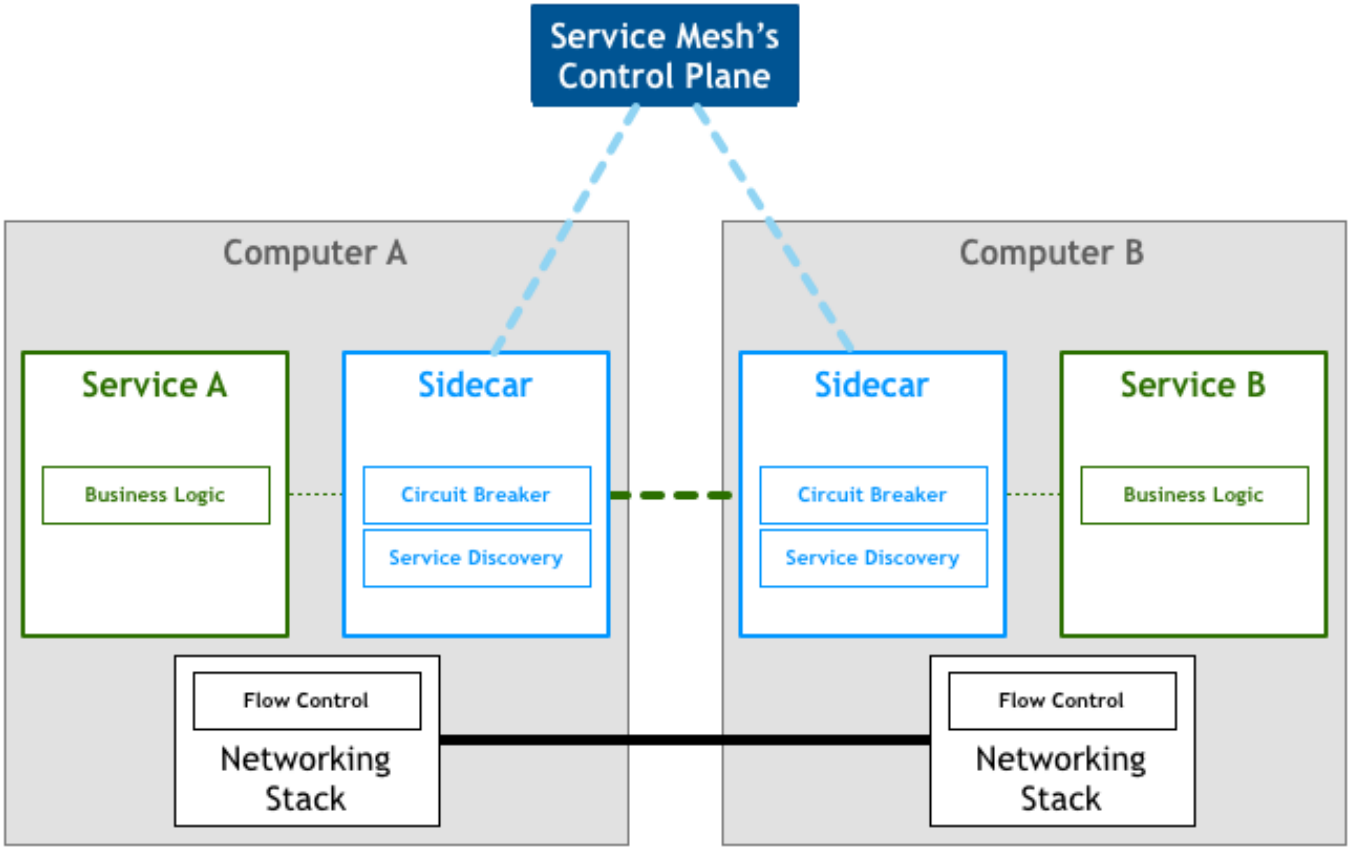


1. 将原有应用拆解为服务
2. 定义服务的api通信文档
3. 编写启动脚本作为容器的入口
4. 拆解出配置文件
5. 制作容器镜像

详情可见迁移到云原生应用架构

service mesh

个人理解是解耦与原应用，将日志/服务发现等服务解耦其实就是service mesh，常见的sidecar部署就属于service mesh。


Service Mesh's Control Plane
Computer A
Service A
Business Logic
Sidecar
Circuit Breaker
Service Discovery
Flow Control
Networking Stack
Computer B
Sidecar
Circuit Breaker
Service Discovery
Service B
Business Logic
Flow Control
Networking Stack