基于Kubernetes的模板化应用编排

王昕@轻元科技

CNUTCon2016 全球容器技术大会

主办方 Geekbang. 极客邦科技 InfoQ

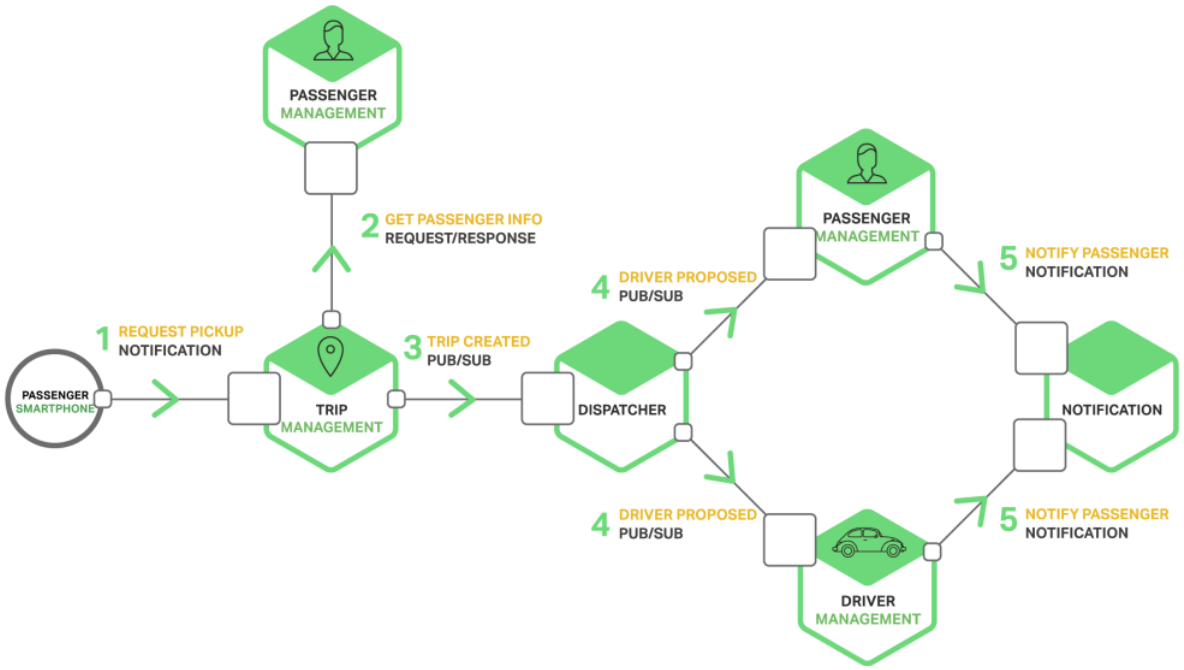
基于Kubernetes的模板部署

目录

为什么要有应用模板

云原生应用和微服务架构

- 微服务架构使得单体应用分解为多个微服务的组合



为什么要有应用模板

微服务是相当于Unix中的进程

- 操作系统用什么管理多种进程和配置的应用？

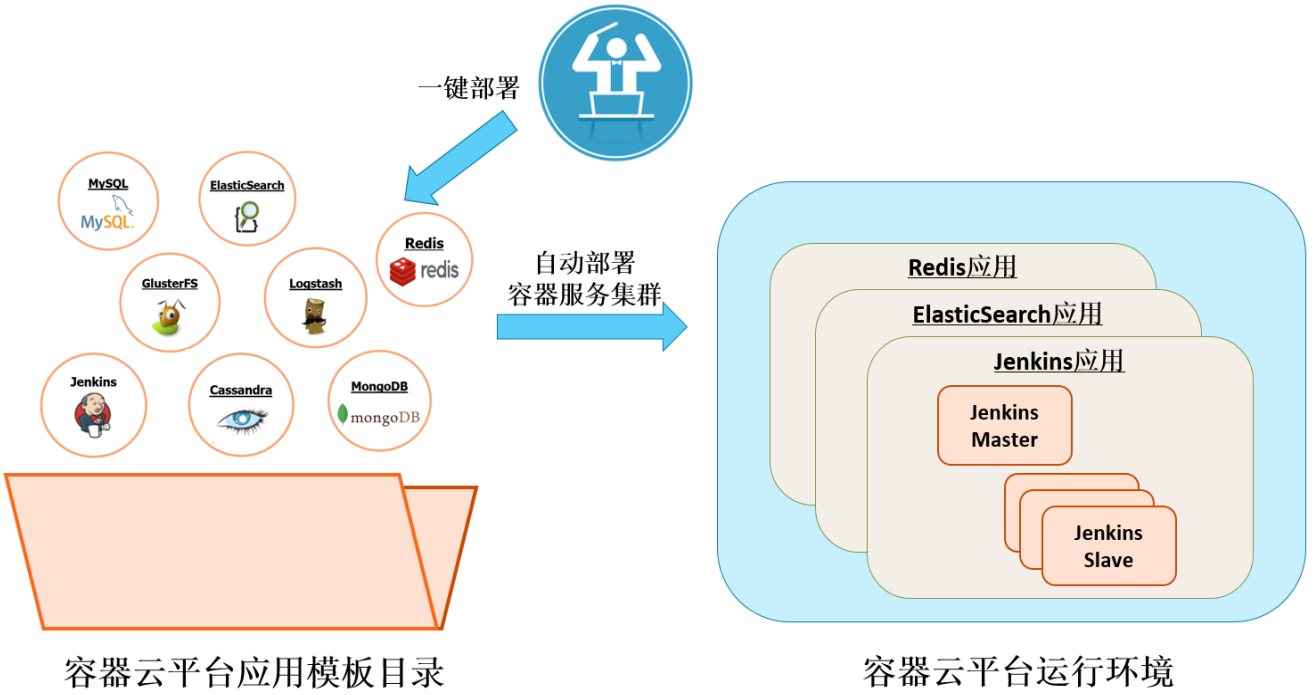
HomeBrew

Apt

Yum

为什么要有应用模板

模板部署相当于Unix上程序包管理器



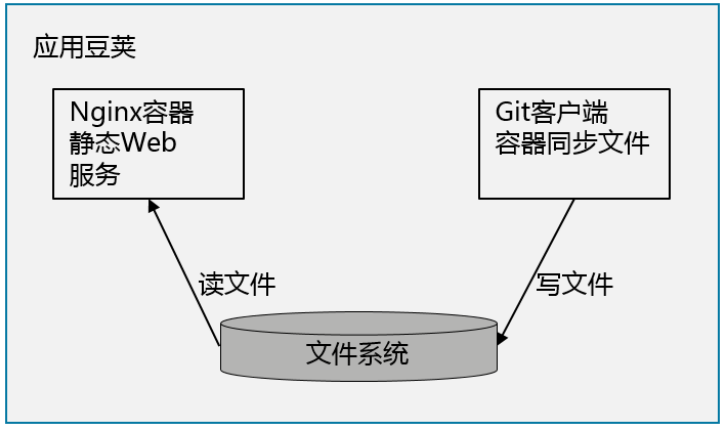
基于Kubernetes的模板部署

目录

Kubernetes的应用模型

微服务实例（微服务豆荚）Pod

- 包括一个或多个容器
- 多个容器共享一个文件系统
- 多个容器共享一个网络地址和端口空间
- 微服务的原子实例
- 服务伸缩的原子粒度

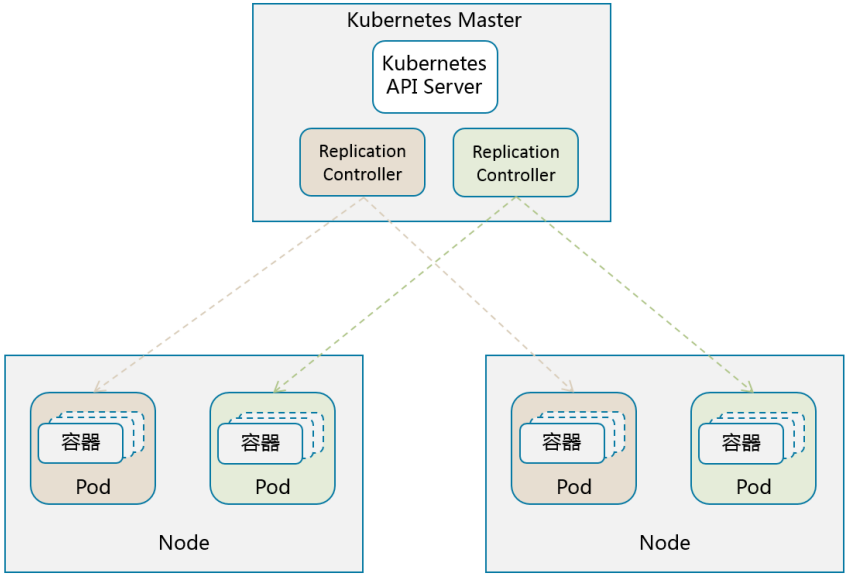


Kubernetes的应用模型

微服务复制控制器RC——Replication Controller

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

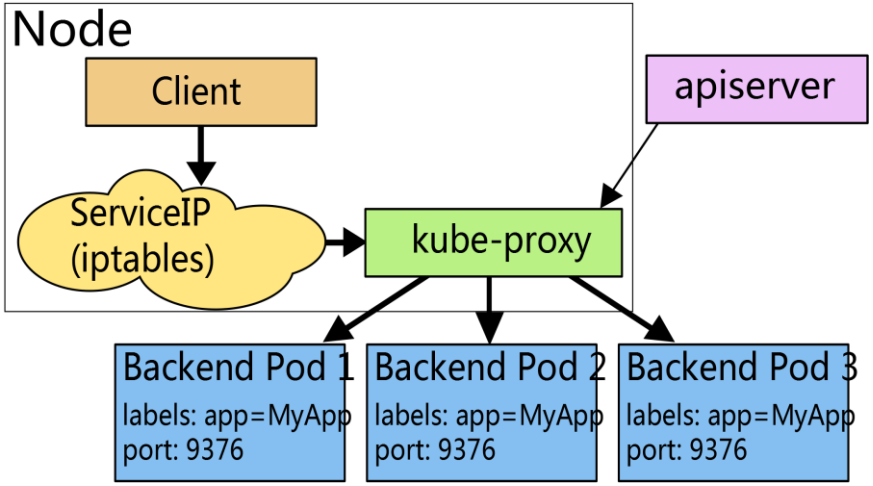
- 控制运行同样的Pod的数量
- 提供高可用能力
- 保证运行的数量：多退少补



Kubernetes的应用模型

微服务的逻辑资源——Service

- 作为访问微服务的统一入口
- 对应多个后端Pod
- 有一个虚拟IP: ServiceIP or ClusterIP
- CusterIP只在集群内可见
- Service到Pod的负载均衡转发由Kube-proxy完成
- Kube-proxy运行在每个Node上

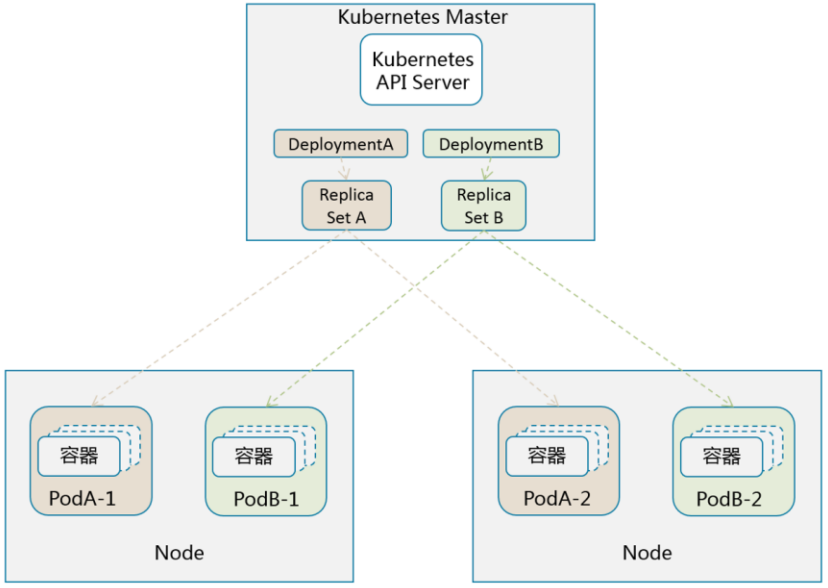


Kubernetes的应用模型

微服务部署和副本集——Deployment & ReplicaSet (RS)

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

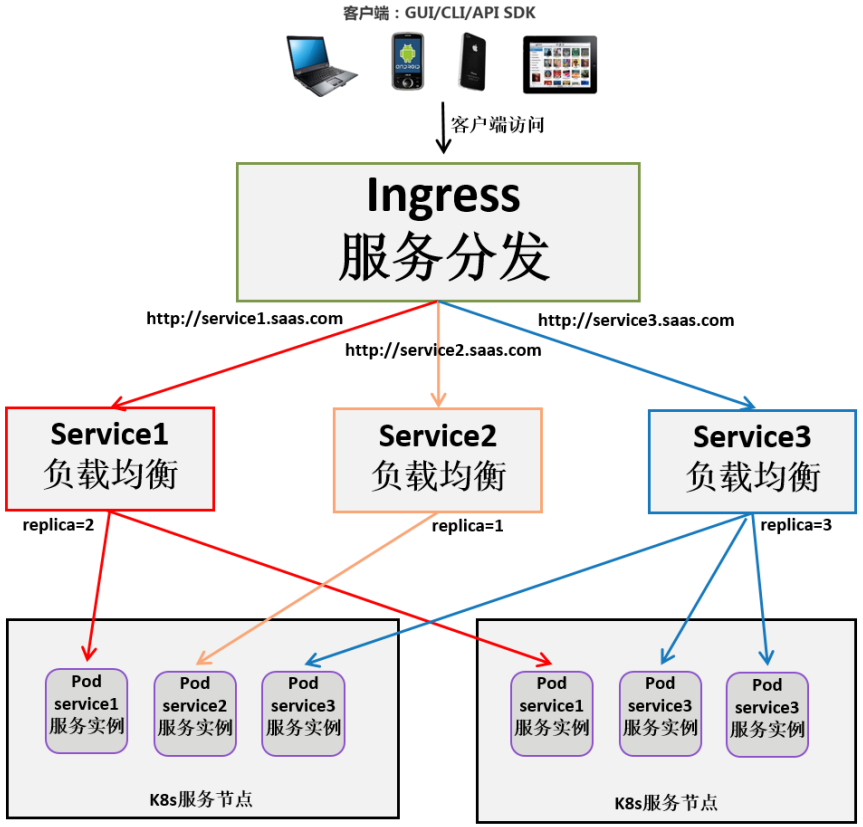
- RS是下一代RC，推荐的，支持更多Selector
- Deployment是Pod和RS的更新Transaction，类似于Transaction之于Finance Account
- 滚动升级，灰度发布，都会用到Deployment
- 对应12 factors的Deployment



Kubernetes的应用模型

微服务的对外发布——服务入口Ingress

- 名称
  - 有些PaaS平台称为负载均衡
  - 有些PaaS平台称为应用路由
- 集群外数据流→集群内数据流的映射
- 协议层
  - 可以是7层（默认HTTP）或4层负载均衡
- 对接方式
  - 集群外主机有个集群内IP: 例如LoadBalance+nodePort
  - 集群内主机有个集群外IP: 例如给集群内节点分配Floating IP



基于Kubernetes的模板部署

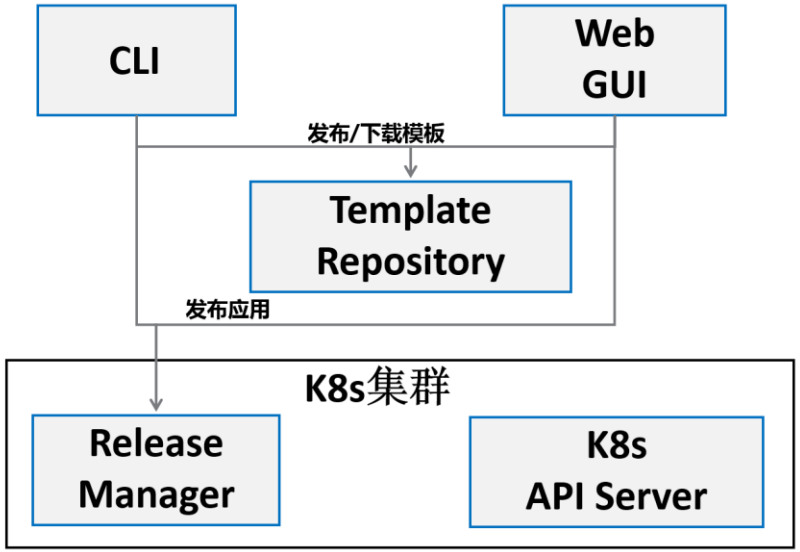
目录

- 为什么要有应用模版
- Kubernetes应用模型
- 系统架构
- 模板的编辑和管理
- 模板应用的发布和管理

应用模版系统架构

系统模块架构

- Template Repository
  - 发布存储模板的HTTP服务
- CLI
  - 管理模板和部署应用的命令行
- Web GUI
  - 管理模板和部署应用的Web界面
- Release Manager
  - 工作在K8s集群中
  - 将环境配置与模板集成
  - 部署和管理应用



基于Kubernetes的模板部署

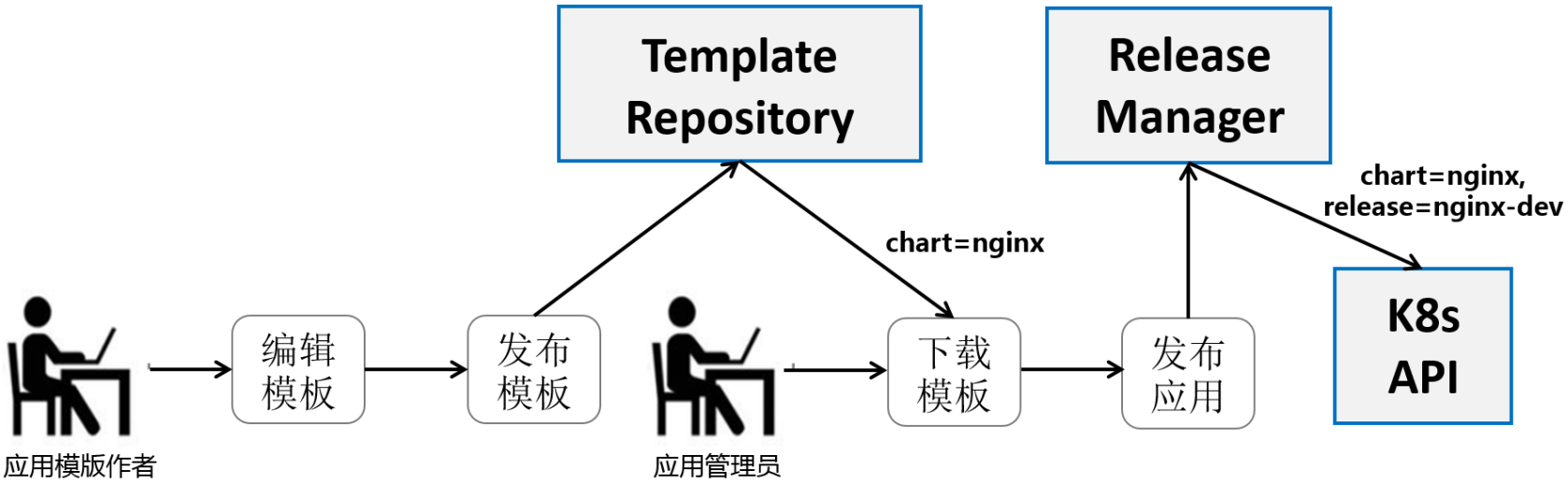
目录

- 为什么要有应用模版
- Kubernetes应用模型
- 系统架构
- 模板的编辑和管理
- 模板应用的发布和管理

模板的编辑和管理

应用模版编辑、发布和使用的流程

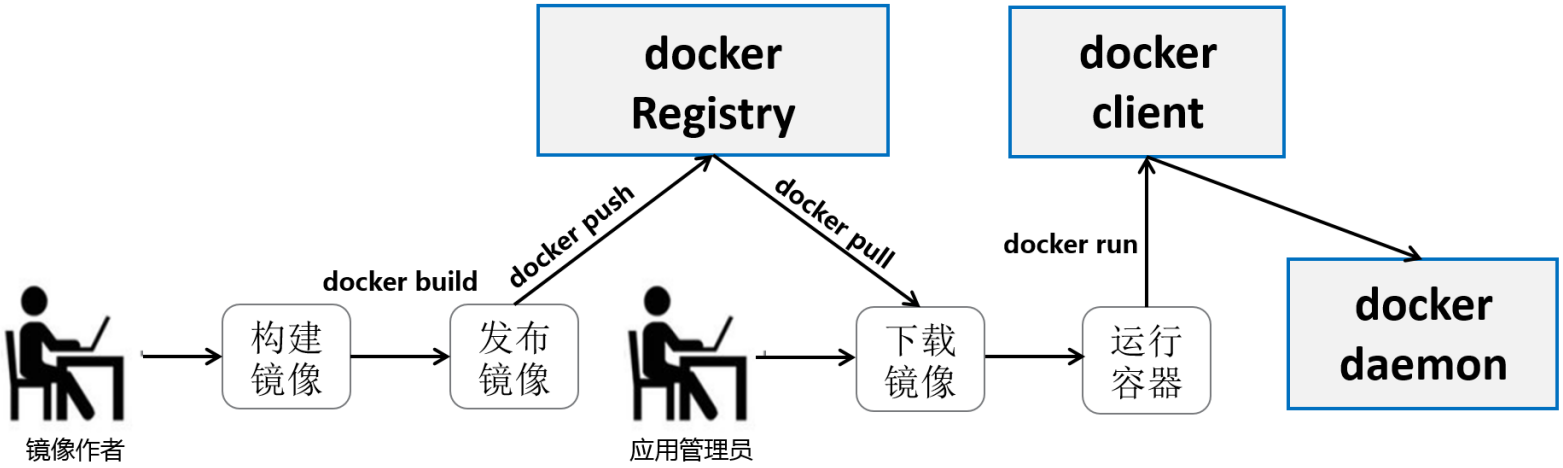
- 编辑模板
  - 模版作者编辑模板文件和模版清单文件
- 发布模版
  - 模版作者将模板上载到公共的模板仓库中去
- 下载模板
  - 应用管理员/模版使用者将模板下载到客户端本地
- 发布应用
  - 将模板和环境配置结合，发布应用到K8s集群中



模板的编辑和管理

似曾相识的流程？

- 构建镜像
  - 镜像作者用docker build构建镜像
- 发布镜像
  - 镜像作者将镜像push到共享的镜像仓库中
- 下载镜像
  - 应用管理员/镜像使用者将镜像下载到客户端本地
- 运行容器
  - 将镜像与环境变量相结合，运行容器

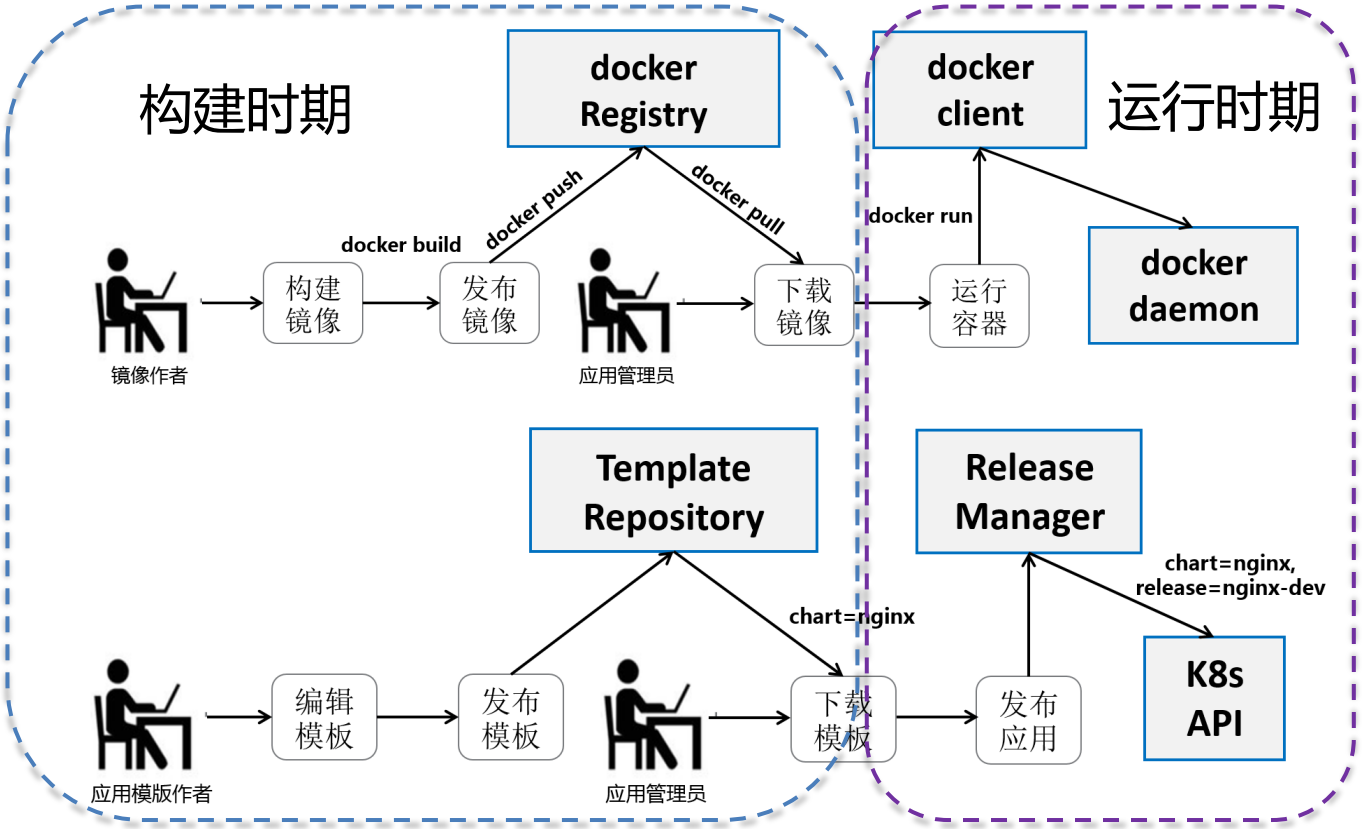


模板的编辑和管理

为何流程如此相似？

云原生应用12 factors

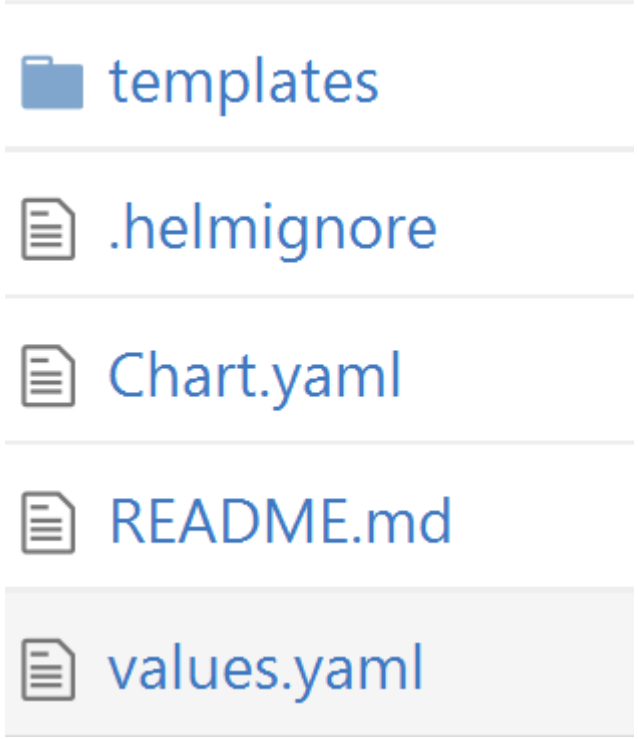
- 一组代码，多次部署
- 严格区分构建和运行



模板的编辑和管理

模板格式-目录结构

- templates/
  - 存储所有模版文件
- vhart.yaml
  - 模板元数据
- values.yaml
  - 模板中变量的默认值



模板的编辑和管理

模板格式——模版文件列表: templates/

- deployment.yaml
  - 部署文件，声明部署多少实例
- svc.yaml
  - 服务描述文件，声明如何访问服务
- secrets.yaml
  - 存储密钥

模板的编辑和管理

模板格式——模版文件deployment.yaml

- Release.Name
  - 发行版名称
- “fullname”
  - Release.Name+Chart.Name
  - 避免名字冲突
- svc.yaml
  - 服务描述文件，声明如何访问服务
- secrets.yaml
  - 存储密钥

```
kind: Deployment
metadata:
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "fullname" . }}
    chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
    release: "{{ .Release.Name }}"
    heritage: "{{ .Release.Service }}"
spec:
  template:
    labels:
      app: {{ template "fullname" . }}
      chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
      release: "{{ .Release.Name }}"
      heritage: "{{ .Release.Service }}"
    spec:
      containers:
      - name: {{ template "fullname" . }}
        image: "bitnami/mariadb:{{ .Values.imageTag }}"
        imagePullPolicy: {{ template "imagePullPolicy" . }}
        env:
```

模板的编辑和管理

模板格式——模版文件Service和Secret

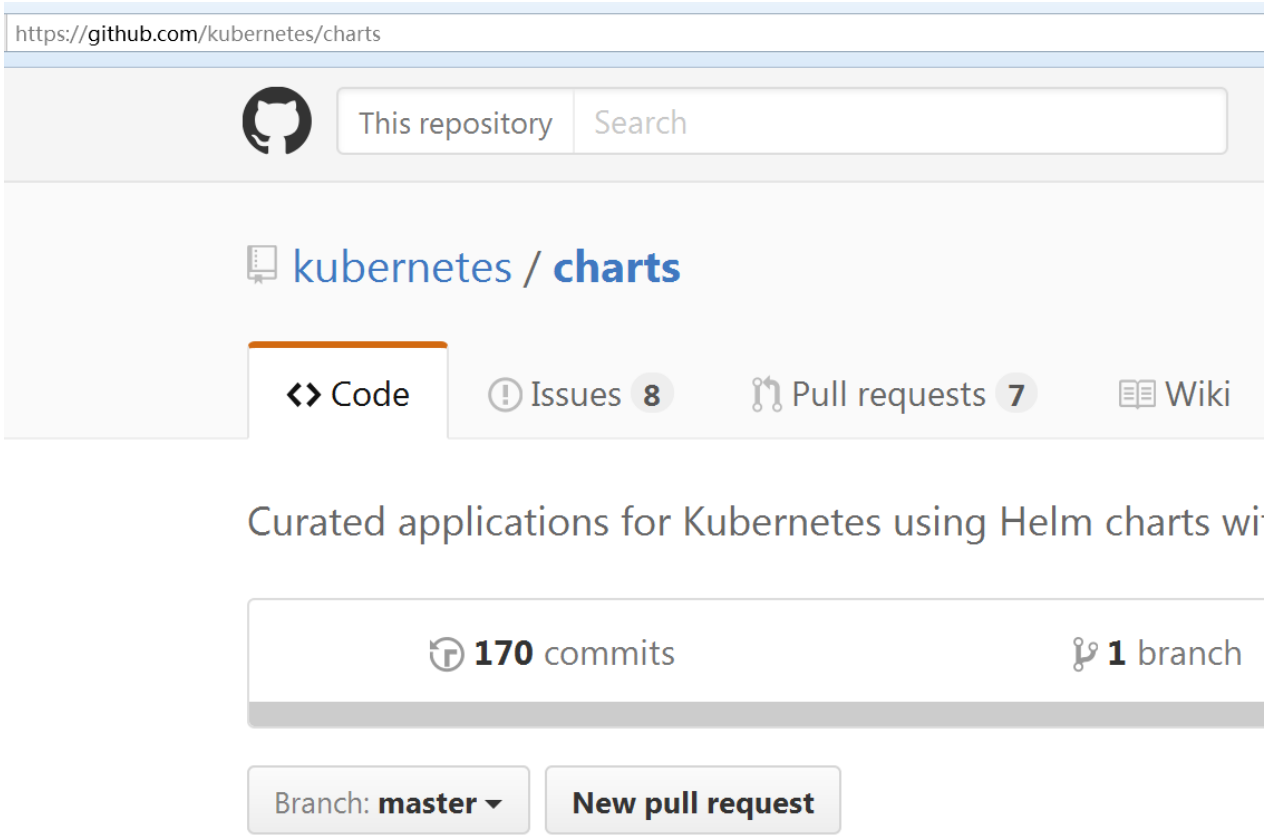
- Label
  - 通过release和app标签可以查找到所有相关的资源

```
kind: Secret
metadata:
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "fullname" . }}
    chart: "{{ .Chart.Name }}-{{ .Chart.Vers
    release: "{{ .Release.Name }}"
    heritage: "{{ .Release.Service }}"
type: Opaque
data:
  mariadb-root-password: {{ default "" .Valu
  mariadb-password: {{ default "" .Values.ma
```

```
kind: Service
metadata:
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "fullname" . }}
    chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
    release: "{{ .Release.Name }}"
    heritage: "{{ .Release.Service }}"
spec:
  ports:
  - name: mysql
    port: 3306
    targetPort: mysql
  selector:
    app: {{ template "fullname" . }}
```

模板的编辑和管理

模板发布



基于Kubernetes的模板部署

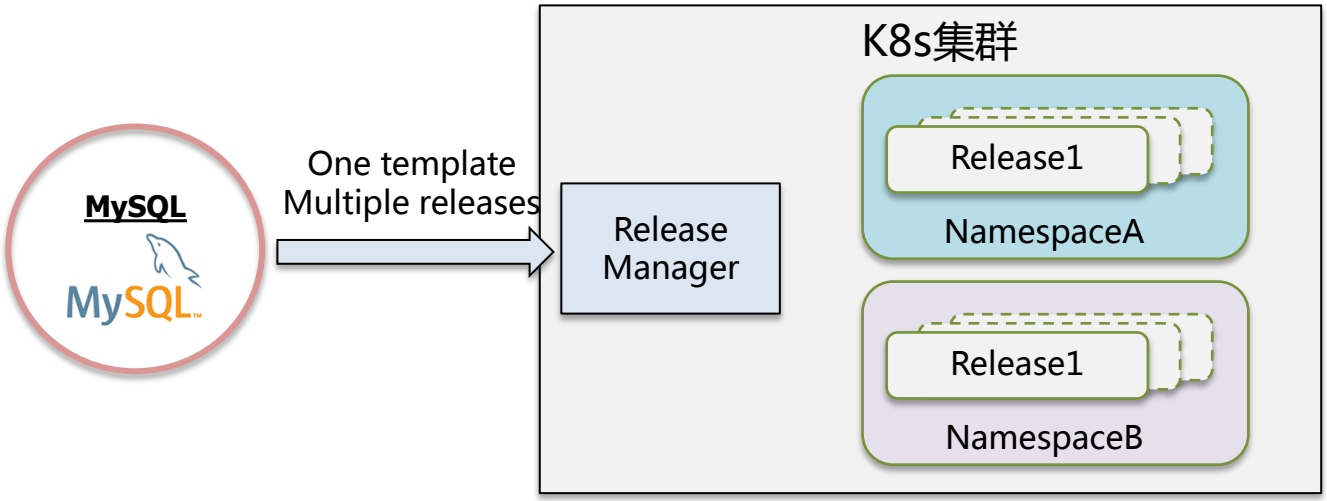
目录

- 为什么要有应用模版
- Kubernetes应用模型
- 系统架构
- 模板的编辑和管理
- 模板应用的发布和管理

模板应用的部署和管理

同一模板，多次应用部署

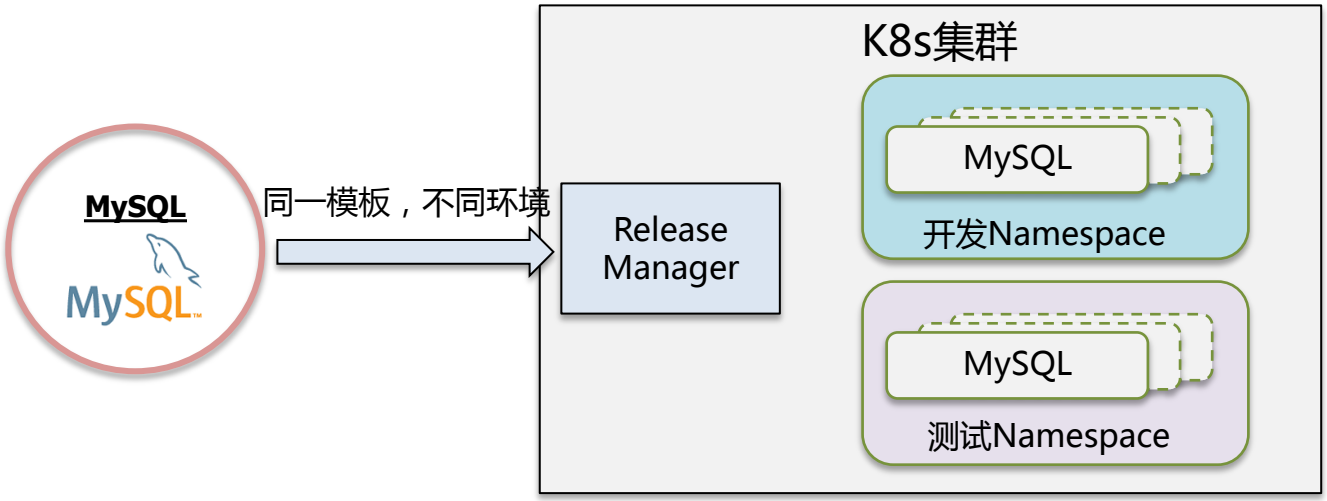
- 同一模板可以在多个隔离的Namespace部署同名应用
- 同一模板可以在同一Namespace部署不同名的Release
- 模板更新可以触发同一Release的更新



模板应用的部署和管理

同一模板，部署在多个Namespace

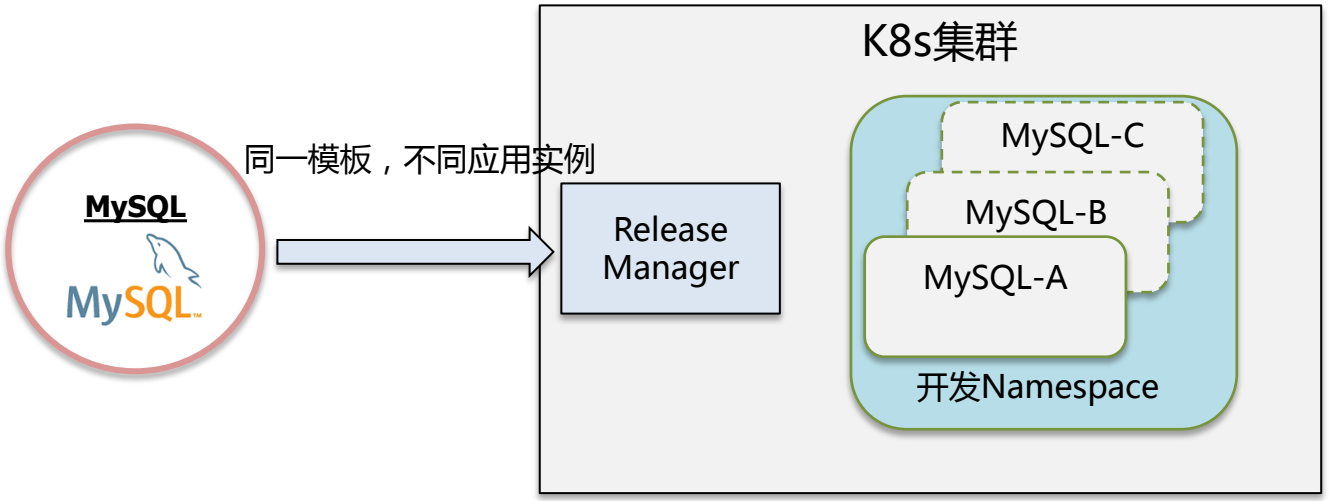
- 不同Namespace，Release名相同
- 不同的应用实例，配置一致，彼此隔离
- 可以分别用于开发、测试、模拟和生产环境



模板应用的部署和管理

同一模板，部署在同一Namespace，不同Release

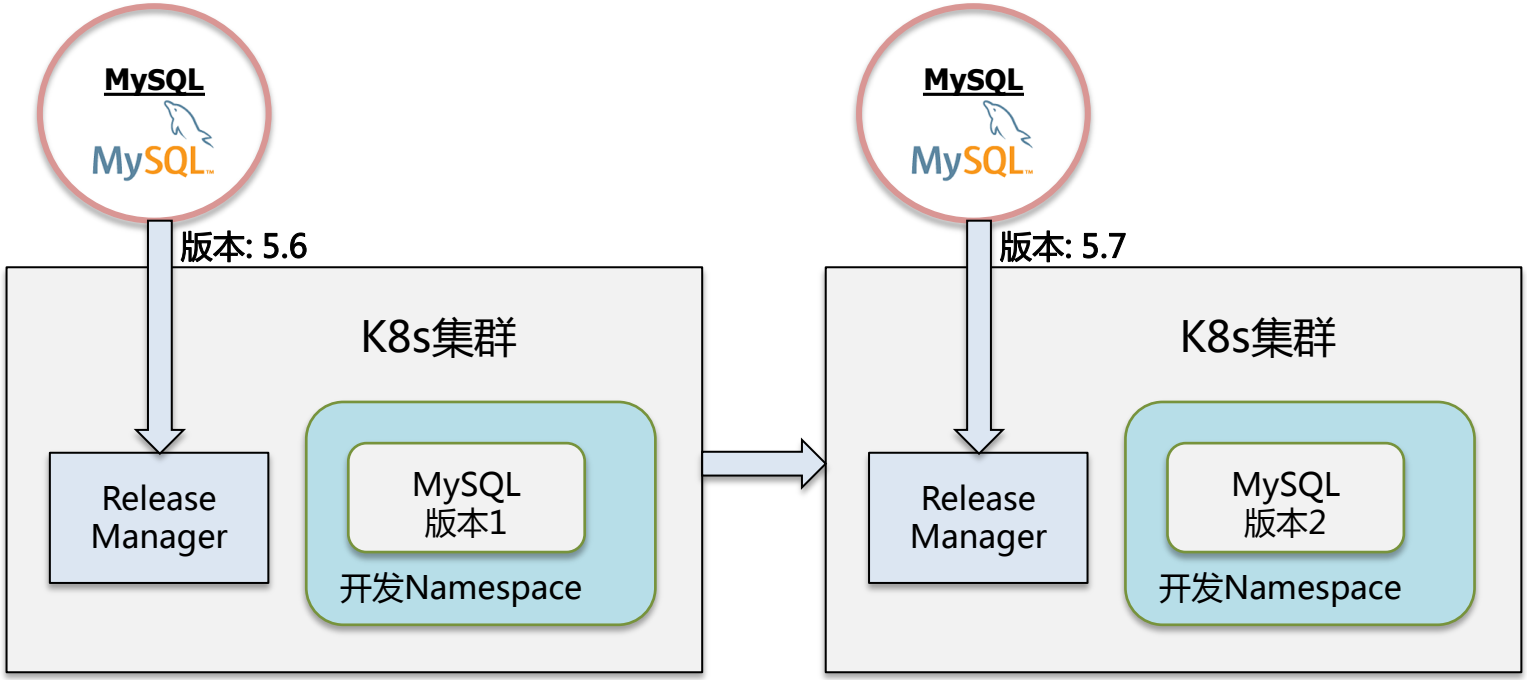
- 同一Namespace，Release名不同
- 配置不同，用于不同应用服务



模板应用的部署和管理

同一模板，同一Release，模板更新触发应用更新

- 同一Namespace，Release名相同: release=MySQL
- 模版更新触发服务MySQL的滚动升级



基于Kubernetes的模板部署

总结

- 为什么要有应用模版
  - 分布式系统的包管理器
- Kubernetes应用模型
  - Pod, Replica Set, Deployment, Service, Ingress
- 系统架构
  - Template Repository, Release Manager, CLI, Web GUI
- 模板的编辑和管理
  - 编辑、发布、下载、部署
- 模板应用的发布和管理
  - 一个模板，对应多个应用实例

THANKS!