

# 基于Helm和Operator的 K8S应用管理

李平辉

RancherLabs

K8S集群部署 ... done

应用容器化 ... done

deployment

RoleBinding

PersistentVolumeClaim

Role

ServiceAccount

Ingress

Secrets

Service

Configmap

Statefulset

# 如何做

- 应用资源的共享、存档、不同环境的配置管理
- 应用资源的生命周期管理



# Kubernetes的包管理工具

应用包 == Charts

# Charts

- 应用的定义描述
- 包括
  - Metadata
  - K8S 资源定义
  - 文档说明
- 存在chart仓库中

# 让我们用起来

```
$ helm init
```



# Tiller

- 部署于K8S中的Helm服务端
- 帮助管理应用包的release
  - Release == Chart 的安装实例



Kubernetes

```
$ helm install <chart>
```

```
# (stable/mariadb, ./nginx-1.2.3.tgz, ./nginx,  
https://example.com/charts/nginx-1.2.3.tgz)
```

```
$ helm upgrade <release>
```

```
$ helm delete <release>
```

# Chart的结构

```
$ helm create demoapp
```

```
demoapp/  
├── Chart.yaml  
├── charts  
├── templates  
└── values.yaml
```

# 模板-templates

```
demoapp/  
├── Chart.yaml  
├── charts  
├── templates  
│   ├── NOTES.txt  
│   ├── _helpers.tpl  
│   ├── deployment.yaml  
│   ├── ingress.yaml  
│   └── service.yaml  
└── values.yaml
```



+

Go template

# 配置选项

```
demoapp/
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── ingress.yaml
│   └── service.yaml
└── values.yaml
```

```
##### values.yaml
replicaCount: 1
image:
  repository: nginx
  tag: stable
  pullPolicy: IfNotPresent
...
```

```
##### deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  #...
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    #...
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
...
```

# 配置选项

```
demoapp/  
├── Chart.yaml  
├── charts  
├── templates  
│   ├── NOTES.txt  
│   ├── _helpers.tpl  
│   ├── deployment.yaml  
│   ├── ingress.yaml  
│   └── service.yaml  
└── values.yaml
```

```
##### values.yaml  
replicaCount: 1  
image:  
  repository: nginx  
  tag: stable  
  pullPolicy: IfNotPresent  
...
```

```
$ helm install --set image.tag=latest ./demoapp
```

```
$ helm install -f stagingvalues.yaml ./demoapp
```

# Metadata

```
demoapp/  
├── Chart.yaml  
├── charts  
├── templates  
└── values.yaml
```

```
##### Chart.yaml  
apiVersion: v1  
description: A Helm chart for Kubernetes  
name: demoapp  
version: 0.1.0
```

# 文档说明

```
demoapp/  
├── Chart.yaml  
├── README.md  
├── charts  
├── templates  
│   ├── NOTES.txt  
│   ├── _helpers.tpl  
│   ├── deployment.yaml  
│   ├── ingress.yaml  
│   └── service.yaml  
└── values.yaml
```



# 应用依赖

```
demoapp/  
├── Chart.yaml  
├── README.md  
├── charts  
│   └── mysql-0.1.0.tgz  
├── requirements.yaml  
├── templates  
└── values.yaml
```

```
##### requirements.yaml  
dependencies:  
- name: mysql  
  version: 0.1.0  
  repository: https://kubernetes-charts-incubator.storage.googleapis.com/
```

# Chart 仓库

- HTTP 服务器
- 提供 index.yaml索引文件和打包的charts

```
$ helm serve
```

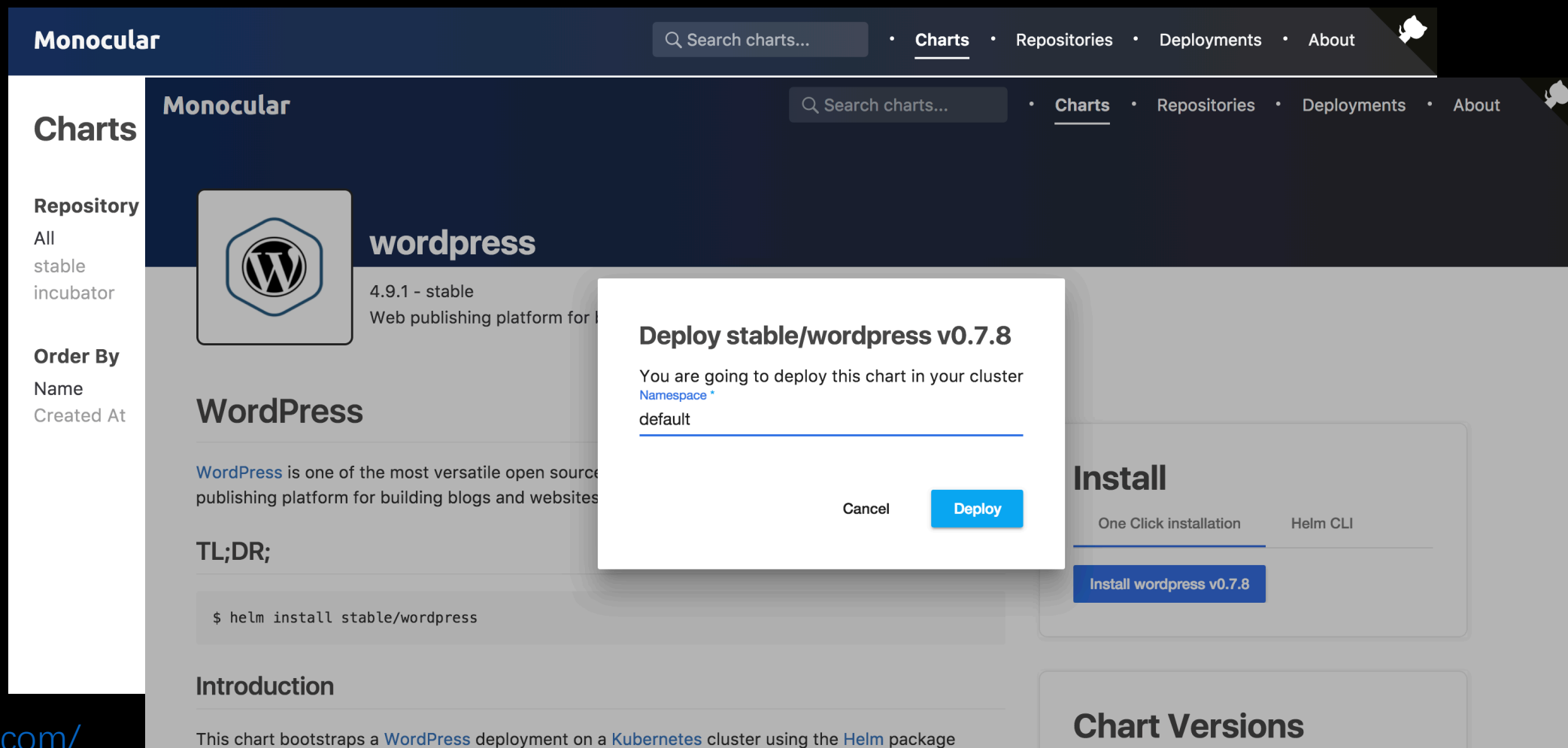
例如 <https://example.com/charts> 的仓库目录结构

```
charts/  
├── index.yaml  
├── demoapp-0.1.0.tgz.prov  
└── demoapp-0.1.0.tgz  
...
```

# 相关项目

- Stable和incubator chart 仓库:
  - <https://github.com/kubernetes/charts>
- Monocular: chart的UI管理项目
  - <https://github.com/kubernetes-helm/monocular>
- Helm Plugins
  - Helm-templates, helm-github, etc.

# Monocular



The screenshot displays the Monocular web interface. The top navigation bar includes the Monocular logo, a search bar, and links for Charts, Repositories, Deployments, and About. The main content area shows the details for the 'wordpress' chart, including its version (4.9.1 - stable) and description. A modal dialog is open in the center, titled 'Deploy stable/wordpress v0.7.8', asking for confirmation to deploy the chart in the 'default' namespace. The dialog has 'Cancel' and 'Deploy' buttons. The background content is dimmed, showing sections for 'WordPress', 'TL;DR;', 'Introduction', and 'Chart Versions'.

# Helm的作用

- 利用已有的Chart快速部署进行实验
- 创建自定义Chart，方便地在团队间共享
- 便于管理应用的生命周期
- 便于应用的依赖管理和重用
- 将K8S集群作为应用发布协作中心



Operator

# 理念

注入领域知识，用软件管理复杂应用

# 场景一

Scale up 一个无状态应用



```
$ kubectl scale replicas=3
```



scale up

ReplicaSet  
app=web, env=dev

desired=3



Pod  
app=web  
env=dev

count=1

\$ kubectl scale replicas=3



# 场景二

## 有状态应用的管理

- scale调整数量
- 升级
- 配置更新
- 备份
- 灾难恢复

# Scale up etcd

- 在集群中添加新节点的url记录，获取集群信息
- 使用获取的集群信息启动etcd节点



kubernetes

⚡ OPERATOR

 etcd

The etcd logo icon, which is a white gear with a human-like face inside, set against a dark blue background.

# 部署etcd

```
$ kubectl create -f etcd-cluster.yaml
```

```
apiVersion: "etcd.database.coreos.com/v1beta2"  
kind: "EtcdCluster"  
metadata:  
  name: "example-etcd-cluster"  
spec:  
  size: 3  
  version: "3.2.11"
```

# Scale up etcd

```
$ kubectl apply -f upgrade-example.yaml
```

```
apiVersion: "etcd.database.coreos.com/v1beta2"  
kind: "EtcdCluster"  
metadata:  
  name: "example-etcd-cluster"  
spec:  
  size: 5  
  version: "3.2.11"
```



# 原理

- 创建类型为EtcdCluster的自定义资源定义（CRD）
- 通过Kubernetes API管理维护目标应用状态
- Kubernetes Controller 模式：

```
for {  
    desired := getDesiredState()  
    current := getCurrentState()  
    makeChange(desired, current)  
}
```



VS



Operator