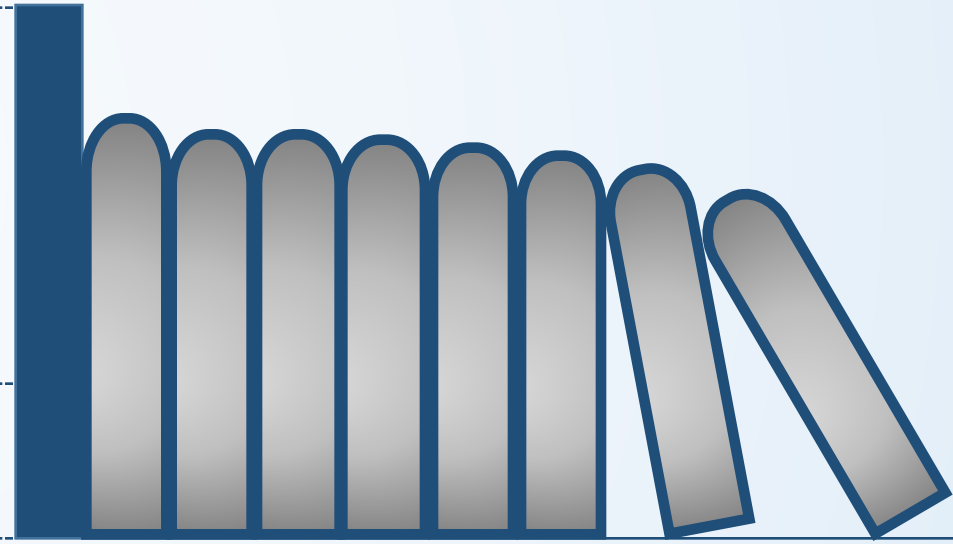
Kubernetes包管理工具Helm

主讲人：宋小金



目录

预期收获

- *学习Helm的使用*
- **生产中的实践和注意事项**

背景

随着业务容器化与向微服务架构转变，通过分解巨大的单体应用为多个服务的方式，分解了单体应用的复杂性，使每个微服务都可以独立部署和扩展，实现了敏捷开发和快速迭代和部署。

但任何事情都有两面性，虽然微服务给我们带来了很多便利，但由于应用被拆分成多个组件，导致服务数量大幅增加，对于Kubernetest编排来说，每个组件有自己的资源文件，并且可以独立的部署与伸缩，这给采用Kubernetes做应用编排带来了诸多挑战，比如管理、编辑与更新大量的K8s配置文件，控制一个部署周期中某一些环节等。

Helm介绍

微服务和容器化给复杂应用部署与管理带来了极大的挑战。Helm是目前Kubernetes服务编排领域的唯一开源子项目，做为Kubernetes应用的一个包管理工具，可理解为Kubernetes的apt-get / yum，由Deis 公司发起，该公司已经被微软收购。

Helm通过软件打包的形式，支持发布的版本管理和控制，很大程度上简化了Kubernetes应用部署和管理的复杂性。

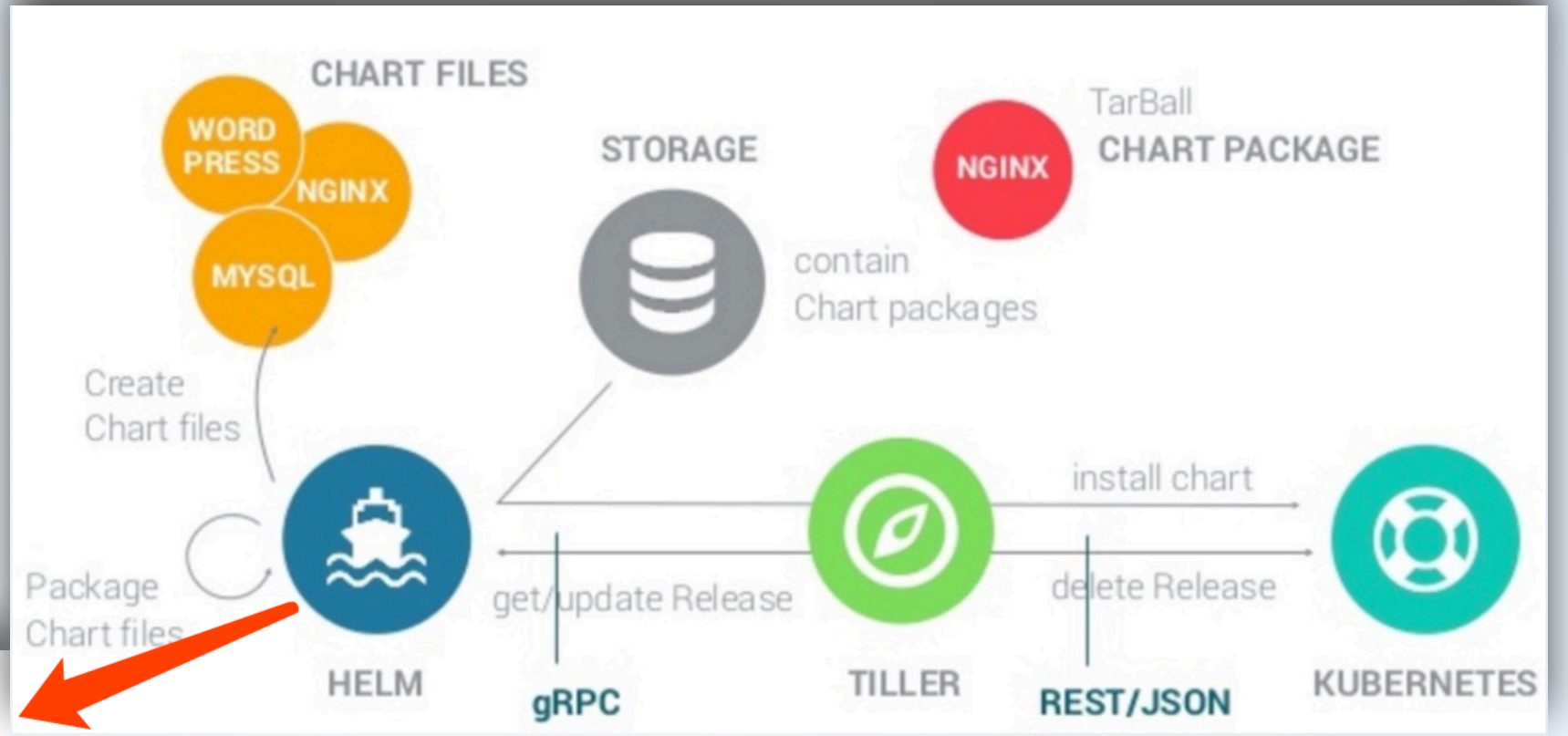
Helm介绍

HELM是一个kubernetes的包管理工具，用来管理charts

- Chart：应用描述，定义了应用中所需要的deployment、 configmap、 ingress、 service、 secret、 job等对象所组成的模板包
  - 支持定义dependency的包依赖
  - 支持go-template语法定义模板和变量参数

- **Release**：基于Chart的部署实体，将在k8s中创建出真实运行的资源对象

Helm架构



输入chart模板

输入values变量

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

一个应用/服务对应一个chart包

Helm操做

利用helm create mychart命令创建一个mychart目录：

```
$ helm create mychart
Creating mychart
```

生成的mychart的文件说明：

- Charts.yaml，描述了Chart名称、描述信息与版本。
- values.yaml：存储了模板文件变量。
- templates/：记录了全部模板文件。
- charts/：依赖chart存储路径。
- NOTES.txt：给出了部署后的信息，例如如何使用chart、列出默认的设置等等。

生成的mychart的文件结构如下：

```
mychart/
|-- charts
|-- Chart.yaml
|-- templates
|   |-- deployment.yaml
|   |-- _helpers.tpl
|   |-- ingress.yaml
|   |-- NOTES.txt
|   `-- service.yaml
`-- values.yaml

2 directories, 7 files
```

模板

配置

服务依赖处理

Kubernetes:

- InitContainer

- postStart
- preStop

Helm Hook:

- pre-install
- post-install

- pre-delete
- post-delete

- pre-upgrade
- post-upgrade

- pre-rollback
- post-rollback

写一个Hook

例如右面的模板中，定义了一个Job类型的Kubernetes resource，并定义了一个post-install hook，该hook的用途是当Kubernetes加载全部resource完毕后，按照Values对象设置的值sleepyTime执行sleep操作（如sleepyTime未设置，那么sleep 10秒）。

```
name: "{{.Release.Name}}"
labels:
heritage: {{.Release.Service | quote }}
release: {{.Release.Name | quote }}
chart: "{{.Chart.Name}}-{{.Chart.Version}}"
annotations:
# This is what defines this resource as a hook. Without this line,
 the
# job is considered part of the release.
"helm.sh/hook": post-install
"helm.sh/hook-weight": "-5"
"helm.sh/hook-delete-policy": hook-succeeded
spec:
template:
metadata:
  name: "{{.Release.Name}}"
  labels:
    heritage: {{.Release.Service | quote }}
    release: {{.Release.Name | quote }}
    chart: "{{.Chart.Name}}-{{.Chart.Version}}"
spec:
  restartPolicy: Never
  containers:
  - name: post-install-job
    image: "alpine:3.3"
    command: ["/bin/sleep","{{default "10" .Values.sleepyTime}}"]
```

利用Hook处理服务启动顺序依赖

可以通过Helm hook来实现服务启动依赖的处理，举例：serviceA服务依赖serviceB， serviceA的pre-install hook中实现:

```yaml
apiVersion: batch/v1
kind: Job
metadata:
name: "{{.Release.Name}}"
labels:
chart: "{{.Chart.Name}}-{{.Chart.Version}}"
annotations:
"helm.sh/hook": pre-install
"helm.sh/hook-weight": "-5"
spec:
template:
metadata:
  name: "{{.Release.Name}}"
  labels:
    chart: "{{.Chart.Name}}-{{.Chart.Version}}"
spec:
  restartPolicy: Never
  containers:
  - name: pre-install-job
    image: "registry.docker.dev.fwmrm.net/busybox:latest"
    command: ['sh', '-c', "curl --connect-timeout 3 --max-time 5 -
-retry 10 check_endpoint"]
```

Helm生产中实践

实际使用过程中：

- 一类应用准备一个模板，不要每个应用都写一个模板，方便模板维护
- Helm 模板 docker image tag，使用—set-string代替—set
- 需Helm支持多机房升级/回滚，为版本的一致性，可全部采用helm upgrade实现
- 可采用开源工具chartmuseum做为Chart Repo

课程回顾

已学知识要点

学习Helm的常见使用命令、生产中最佳实践