Helm教程

由 温兆钦创建, 最后修改于五月 27, 2019

1.概述

Helm是k8s的包管理工具，类似Linux系统常用的 apt、yum等包管理工具。

使用helm可以简化k8s应用部署

2.基本概念

- **Chart**：一个 Helm 包，其中包含了运行一个应用所需要的镜像、依赖和资源定义等，还可能包含 Kubernetes 集群中的服务定义，类似 Homebrew 中的 formula、APT 的 dpkg 或者 Yum 的 rpm 文件。
- **Release**：在 Kubernetes 集群上运行的 Chart 的一个实例。在同一个集群上，一个 Chart 可以安装很多次。每次安装都会创建一个新的 release。例如一个 MySQL Chart，如果想在服务器上运行两个数据库，就可以把这个 Chart 安装两次。每次安装都会生成自己的 Release，会有自己的 Release 名称。
- **Repository**：用于发布和存储 Chart 的存储库。

3.架构

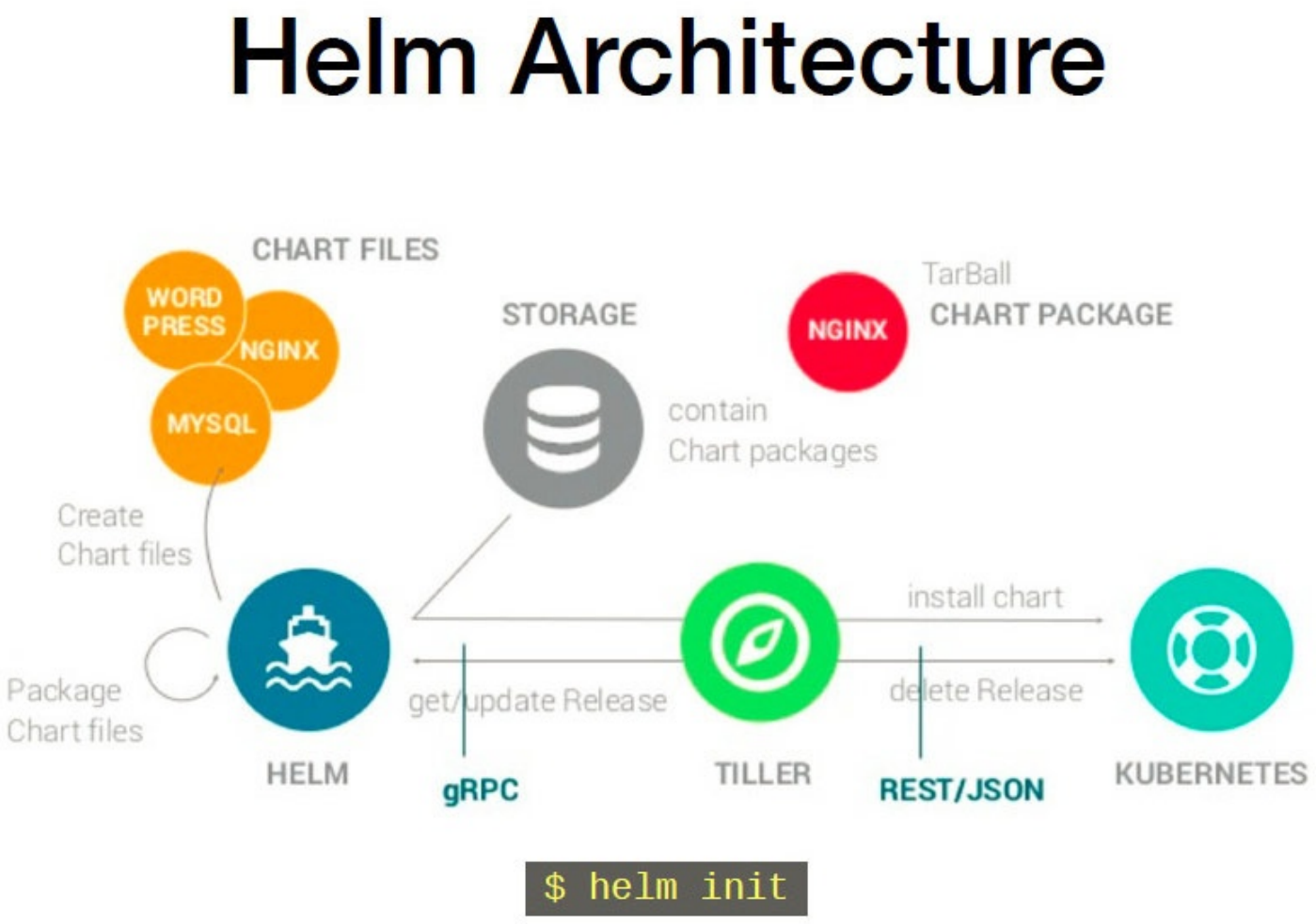

Chart Install 过程：

1. Helm从指定的目录或者tgz文件中解析出Chart结构信息
2. Helm将指定的Chart结构和Values信息通过gRPC传递给Tiller
3. Tiller根据Chart和Values生成一个Release
4. Tiller将Release发送给Kubernetes运行。

Chart Update过程：

1. Helm从指定的目录或者tgz文件中解析出Chart结构信息
2. Helm将要更新的Release的名称和Chart结构，Values信息传递给Tiller
3. Tiller生成Release并更新指定名称的Release的History
4. Tiller将Release发送给Kubernetes运行

4.安装helm

helm主要包括helm客户端和Tiller服务端两部分，Tiller部署在k8s集群中。

ps： 如果使用阿里云容器服务kubernetes版,默认已经安装了helm的服务端（Tiller）,只要安装helm客户端即可。

可以根据自己的环境从github地址下载对应的安装包:

下载地址：https://github.com/helm/helm/releases

- windows 64位版: https://storage.googleapis.com/kubernetes-helm/helm-v2.13.1-windows-amd64.zip
- linux 64位版：https://storage.googleapis.com/kubernetes-helm/helm-v2.13.1-linux-arm64.tar.gz

下载后解压到自己喜欢的目录，然后配置下对应的PATH环境变量。

默认情况helm操作k8s集群，需要借助**kubectl命令的集群配置**，可以参考这里配置kubectl命令-（k8s应用配置详解），当然也可以直接给helm命令指定**--kubeconfig** 参数指定k8s集群证书路径。

```
#这是通过--kubeconfig参数指定k8s证书的方式操作k8s集群
#下面命令是部署一个名字叫app-demo的应用，helm包在./chart目录中
/alidata/server/helm-v2.13.1/helm --kubeconfig ./config/test-k8s.conf install app-demo ./ch
```

安装服务端：

使用helm init 命令，可以一键安装。

ps: 关于chart仓库（Repository），通过helm命令：helm serve 就可以启动仓库服务，但是通常很多时候我们每个项目自己的chart包都跟着源码一起提交到git仓库，所以这里的chart仓库不是必须的。

5.基本用法

这里以制作一个简单的网站应用chart包为例子介绍helm的基本用法。

ps: 这里跳过docker镜像制作过程，镜像制作可以参考：Docker基础教程

5.1.创建chart包

通过helm create命令创建一个新的chart包

例子:

#在当前目录创建一个myapp chart包

$ helm create myapp

创建完成后，得到的目录结构如下:

myapp **- chart 包目录名**
├── charts - 依赖的子包目录，里面可以包含多个依赖的chart包
├── Chart.yaml - chart定义，可以定义chart的名字，版本号信息。
├── templates - k8s配置模版目录， 我们编写的k8s配置都在这个目录， 除了**NOTES.txt**和**下划线**开头命

名的文件，其他文件可以随意命名。

```
│   ├── deployment.yaml
│   ├── _helpers.tpl            - 下划线开头的文件，helm视为公共库定义文件，主要用于定义通用的子模版、函数等，helm不会将这些公共库文件的渲染结果提交给k8s处理。
│   ├── ingress.yaml
│   ├── NOTES.txt               - chart包的帮助信息文件，执行helm install命令安装成功后会输出这个文件的内容。
│   └── service.yaml
└── values.yaml                 - chart包的参数配置文件，模版可以引用这里参数。
```

我们要在k8s中部署一个网站应用，需要编写**deployment、service、ingress**三个配置文件，刚才通过helm create命令已经创建好了。

5.2.编写k8s应用部署配置文件

为了演示chart包模版的用法，我们先把**deployment、service、ingress**三个配置文件的内容清空，重新编写k8s部署文件。

deployment.yaml 配置文件定义如下：

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: myapp           #deployment应用名
  labels:
    app: myapp          #deployment应用标签定义
spec:
  replicas: 1           #pod副本数
  selector:
    matchLabels:
      app: myapp          #pod选择器标签
  template:
    metadata:
      labels:
        app: myapp          #pod标签定义
    spec:
      containers:
        - name: myapp           #容器名
          image: registry-vpc.cn-hangzhou.aliyuncs.com/zhipuzi/cysystem:1.7.9     #镜像地址
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
```

service.yaml定义如下：

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-svc #服务名
spec:
  selector: #pod选择器定义
    app: myapp
```

```
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

ingress.yaml定义如下：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: myapp-ingress #ingress应用名
spec:
  rules:
    - host: www.zhipuzi.com  #域名
      http:
        paths:
          - path: /
            backend:
              serviceName: myapp-svc #服务名
              servicePort: 80
```

5.3.提取k8s应用部署配置文件中的参数，作为chart包参数。

上面已经完成k8s应用部署配置文件的编写。

为什么要提取上面配置文件中的参数，作为chart包的参数？

思考下面的问题：

我们制作好一个chart包之后，如实现chart包更具有通用性，我们如何换域名？换镜像地址？改一下应用部署的名字？ 部署多套环境（例如：dev环境、test环境分别以不同的应用名字部署一套）

5.2定义的k8s配置文件还不能称之为模版，都是固定的配置。**（这里所说的模版就类似大家平时做前端开发的时候用的模版技术是一个概念）**

我们通过提**取配置中的参数**，**注入模版变量，模版表达式**将配置文件转化为**模版文件**，helm在运行的时候**根据参数动态的将模版文件**渲染成最终的配置文件。

下面将**deployment、service、ingress**三个配置文件转换成模版文件。

ps: {{ }} 两个花括号包裹的内容为模版表达式，具体含义，后面会说明，这里不用理会。

deployment.yaml 配置模版如下：

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: {{ .Release.Name }}  #deployment应用名
  labels:
    app: {{ .Release.Name }}          #deployment应用标签定义
spec:
  replicas: {{ .Values.replicas}}           #pod副本数
```

```
  selector:
    matchLabels:
      app: {{ .Release.Name }}          #pod选择器标签
  template:
    metadata:
      labels:
        app: {{ .Release.Name }}          #pod标签定义
    spec:
      containers:
        - name: {{ .Release.Name }}           #容器名
          image: {{ .Values.image }}:{{ .Values.imageTag }}    #镜像地址
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
```

service.yaml定义如下：

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}-svc #服务名
spec:
  selector: #pod选择器定义
    app: {{ .Release.Name }}
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

ingress.yaml定义如下：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: {{ .Release.Name }}-ingress #ingress应用名
spec:
  rules:
    - host: {{ .Values.host }}  #域名
      http:
        paths:
          - path: /
            backend:
              serviceName: {{ .Release.Name }}-svc #服务名
              servicePort: 80
```

values.yaml chart包参数定义：

```
#域名
host: www.zhipuzi.com

#镜像参数
image: registry-vpc.cn-hangzhou.aliyuncs.com/zhipuzi/cysystem
imageTag: 1.7.9

#pod 副本数
replicas:1
```

5.4.通过helm命令安装/更新应用

安装应用:

#命令格式: helm install chart包目录

$ helm install ./myapp

通过命令注入参数

#命令格式: helm install --set key=value chart包目录

#–set 参数可以指定多个参数，他的值会覆盖values.yaml定义的值，对象类型数据可以用 . (点)分割属性名,例子: --set apiAppResources.requests.cpu=1

$ helm install \

--set replicas=2 \

--set host=www.lewaimai.com \

./myapp

更新应用：

#命令格式: helm upgrade release名字 chart包目录

$ helm upgrade myapp ./myapp

#也可以指定–set参数

$ helm upgrade \

--set replicas=2 \

--set host=www.lewaimai.com \

myapp ./myapp

#默认情况下，如果release名字不存在，upgrade会失败，可以加上-i 参数当release不存在的时候则安装，存在则更新，将install和uprade命令合并。

$ helm upgrade **-i** \

--set replicas=2 \

--set host=www.lewaimai.com \

myapp ./myapp

6.模版语法

6.1.表达式

模版表达式： {{ 模版表达式 }}

模版表达式： {{- 模版表达式 -}} ， 表示去掉表达式输出结果前面和后面的空格，去掉前面空格可以这么写{{- 模版表达式 }}, 去掉后面空格 {{ 模版表达式 -}}

6.2.变量

默认情况点(.), 代表全局作用域，用于引用全局对象。

例子：

#这里引用了全局作用域下的Values对象中的key属性。

{{ .Values.key }}

helm全局作用域中有两个重要的全局对象：**Values和Release**

Values代表的就是values.yaml定义的参数，通过**.Values**可以引用任意参数。

例子：

{{ .Values.replicaCount }}

#引用嵌套对象例子，跟引用json嵌套对象类似

{{ .Values.image.repository }}

Release代表一次应用发布，下面是Release对象包含的属性字段：

- Release.Name - release的名字，一般通过Chart.yaml定义，或者通过helm命令在安装应用的时候指定。
- Release.Time - release安装时间
- Release.Namespace - k8s名字空间
- Release.Revision - release版本号，是一个递增值，每次更新都会加一
- Release.IsUpgrade - true代表，当前release是一次更新.
- Release.IsInstall - true代表，当前release是一次安装

例子:

{{ .Release.Name }}

除了系统自带的变量，我们自己也可以自定义模版变量。

#变量名以$开始命名， 赋值运算符是 **:= (冒号+等号)**

{{- $relname := .Release.Name -}}

引用自定义变量:

#不需要 . 引用

{{ $relname }}

6.3.函数&管道运算符

调用函数的语法：{{ functionName arg1 arg2... }}

例子:

#调用quote函数，将结果用“”引号包括起来。

{{ quote .Values.favorite.food }}

管道（ pipelines ）运算符 |

类似linux shell命令，通过管道 | 将多个命令串起来，处理模版输出的内容。

例子：

#将.Values.favorite.food传递给quote函数处理，然后在输出结果。

{{ .Values.favorite.food | quote }}

#先将.Values.favorite.food的值传递给upper函数将字符转换成大写，然后专递给quote加上引号包括起来。

{{ .Values.favorite.food | upper | quote }}

#如果.Values.favorite.food为空，则使用default定义的默认值

{{ .Values.favorite.food | default "默认值" }}

#将.Values.favorite.food输出5次

{{ .Values.favorite.food | repeat 5 }}

#对输出结果缩进2个空格

{{ .Values.favorite.food | nindent 2 }}

常用的关系运算符>、 >=、 <、!=、与或非在helm模版中都以函数的形式实现。

关系运算函数定义：

```
eq  相当于 =
ne  相当于 !=
lt  相当于 <=
gt  相当于 >=
and  相当于 &&
or   相当于 ||
not  相当于 !
```

例子：

```
#相当于 if (.Values.fooString && (.Values.fooString == "foo"))
{{ if and .Values.fooString (eq .Values.fooString "foo") }}
    {{ ... }}
{{ end }}
```

6.4.流程控制语句

6.4.1. IF/ELSE

语法:

{{ if 条件表达式 }}
Do something
{{ else if 条件表达式 }}
Do something else
{{ else }}
Default case
{{ end }}

例子:

```
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | default "tea" | quote }}
  food: {{ .Values.favorite.food | upper | quote }}
  {{if eq .Values.favorite.drink "coffee"}}
    mug: true
  {{end}}
```

6.4.2. with

with主要就是用来修改 . 作用域的，默认 . 代表全局作用域，with语句可以修改.的含义.

语法：

{{ with 引用的对象 }}
这里可以使用 . (点)， 直接引用with指定的对象
{{ end }}

例子:

#.Values.favorite是一个object类型

{{- with .Values.favorite }}
drink: {{ .drink | default "tea" | quote }} #相当于.Values.favorite.drink
food: {{ .food | upper | quote }}
{{- end }}

ps: 不能在with作用域内使用 . 引用全局对象, 如果非要在with里面引用全局对象，可以先在with外面将全局对象复制给一个变量，然后在with内部使用这个变量引用全局对象。

例子:

{{- $release:= .Release.Name -}} #先将值保存起来

{{- with .Values.favorite }}
drink: {{ .drink | default "tea" | quote }} #相当于.Values.favorite.drink
food: {{ .food | upper | quote }}

release: {{ $release }} #间接引用全局对象的值
{{- end }}

6.4.3. range

range主要用于循环遍历数组类型。

语法1:

#遍历map类型，用于遍历键值对象

#变量$key代表对象的属性名，$val代表属性值

{{- range $key, $val := 键值对象 }}
{{ $key }}: {{ $val | quote }}
{{- end}}

语法2：

{{- range 数组 }}

{{ . | title | quote }} # . (点)，引用数组元素值。

```
{{- end }}
```

例子：

```
#values.yaml定义

#map类型
favorite:
  drink: coffee
  food: pizza

#数组类型
pizzaToppings:
  - mushrooms
  - cheese
  - peppers
  - onions

map类型遍历例子:
{{- range $key, $val := .Values.favorite }}
{{ $key }}: {{ $val | quote }}
{{- end}}

数组类型遍历例子:
{{- range .Values.pizzaToppings}}
{{ . | quote }}
{{- end}}
```

6.5.子模版定义

我们可以在_(下划线)开头的文件中定义子模版，方便后续复用。

helm create默认为我们创建了_helpers.tpl 公共库定义文件，可以直接在里面定义子模版，也可以新建一个，只要以下划线开头命名即可。

子模版语法:

定义模版

```
{{ define "模版名字" }} 模版内容 {{ end }}
```

引用模版：

```
{{ include "模版名字" }}
```

例子：

```
#模版定义
{{- define "mychart.app" -}}
app_name: {{ .Chart.Name }}
app_version: "{{ .Chart.Version }}+{{ .Release.Time.Seconds }}"
{{- end -}}

apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: {{ .Release.Name }}-configmap
  labels:
    {{ include "mychart.app" . | nindent 4 }} #引用mychart.app模版内容，并对输出结果缩进4个空格
data:
  myvalue: "Hello World"
```

6.6.调试

编写好chart包的模版之后，我们可以给helm命令加上--debug --dry-run 两个参数，**让helm输出模版结果，但是不把模版输出结果交给k8s处理。**

例子：

#helm install命令类似，加上--debug --dry-run两个参数即可

$ helm upgrade **--debug --dry-run** -i \

--set replicas=2 \

--set host=www.lewaimai.com \

myapp ./myapp

无标签