Helm简介

Helm类似与yum,apt等软件打包工具，从而实现Kubernetes资源的动态配置和依赖管理。为大规模部署提供便利。

先决条件

想成功使用Helm需要一下前置条件

1. 一个 Kubernetes 集群
2. 确定你安装版本的安全配置
3. 安装和配置Helm

Helm服用kubectl的配置文件以实现与Kubernetes集群的通信，获取二进制可执行文件

```
wget https://get.helm.sh/helm-v3.6.3-linux-amd64.tar.gz
```

基本命令

1. 添加chart repo

```
helm repo add <REPO NAME> <REPO URI>
```

2. 更新仓库缓存

```
helm repo update
```

3. 获取chart

```
helm fetch <CHAER URI>
```

4. 查看已经部署的CHART

```
helm list <RELEASE-NAME>
```

5. 删除某个RELEASE

```
helm uninstall <RELEASE-NAME>
```

6. 部署某个RELEASE

```
helm install <RELEASE-NAME> <CHART-NAME>
```

使用Helm

Helm用来管理 Kubernetes 集群上的软件包

核心概念

- `Chart` 代表着 Helm 包。它包含在 Kubernetes 集群内部运行应用程序，工具或服务所需的所有资源定义。你可以把它看作是 Homebrew formula，Apt dpkg，或 Yum RPM 在 Kubernetes 中的等价物。
- `Repository` 是用来存放和共享 charts 的地方。
- `Release` 是运行在 Kubernetes 集群中的 chart 的实例。一个 chart 通常可以在同一个集群中安装多次。每一次安装都会创建一个新的 release。以 MySQL chart为例，如果你想在你的集群中运行两个数据库，你可以安装该chart两次。每一个数据库都会拥有它自己的 release 和 release name。

Helm 自带一个强大的搜索命令，可以用来从两种来源中进行搜索：

- `helm search hub` 从 Artifact Hub 中查找并列出 helm charts。 Artifact Hub中存放了大量不同的仓库。
- `helm search repo` 从你添加（使用 helm repo add）到本地 helm 客户端中的仓库中进行查找。该命令基于本地数据进行搜索，无需连接互联网。

Helm按如下顺序创建响应的资源：

- Namespace
- NetworkPolicy
- ResourceQuota
- LimitRange
- PodSecurityPolicy
- PodDisruptionBudget
- ServiceAccount
- Secret
- SecretList
- ConfigMap
- StorageClass

- PersistentVolume
- PersistentVolumeClaim
- CustomResourceDefinition
- ClusterRole
- ClusterRoleList
- ClusterRoleBinding
- ClusterRoleBindingList
- Role
- RoleList
- RoleBinding
- RoleBindingList
- Service
- DaemonSet
- Pod
- ReplicationController
- ReplicaSet
- Deployment
- HorizontalPodAutoscaler
- StatefulSet
- Job
- CronJob
- Ingress
- APIService

创建chart模板

```
helm create deis-workflow
```

详解HELM

CHART

Helm使用的包格式称为 chart。 chart就是一个描述Kubernetes相关资源的文件集合。单个chart可以用来部署一些简单的， 类似于memcache pod，或者某些复杂的HTTP服务器以及web全栈应用、数据库、缓存等等。

Chart是作为特定目录布局的文件被创建的。它们可以打包到要部署的版本存档中。

如果你想下载和查看一个发布的chart，但不安装它，你可以用这个命令： `helm pull chartrepo/chartname`。

Chart文件结构

```
[root@controller01 ~]# helm create test
Creating test

[root@controller01 ~]# tree test
test
├── charts                          # 包含chart依赖的其他chart
├── Chart.yaml                      # 包含了chart信息的YAML文件
├── templates                       # 模板目录， 当和values 结合时，可生成有效的Kubernetes
manifest文件
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt                  # 可选：包含简要使用说明的纯文本文件
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml                     # chart 默认的配置值


# 可添加crds用以添加自定义资源
```

Chart.yaml 文件

Chart.yaml文件是chart必需的。包含了以下字段：

```
apiVersion: v2
name: chart名称 （必需）
version: Chart版本
kubeVersion: 兼容Kubernetes版本的语义化版本（可选）
description: 一句话对这个项目的描述（可选）
type: chart类型 （可选）application or library
keywords:
  - 关于项目的一组关键字（可选）
home: 项目home页面的URL （可选）
sources:
  - 项目源码的URL列表（可选）
dependencies: # chart 必要条件列表 （可选）helm dependency update可下载以来chart
  - name: chart名称（nginx）
    version: chart版本 ("1.2.3")
    repository: （可选）仓库URL ("https://example.com/charts") 或别名 ("@repo-name")
    condition: （可选） 解析为布尔值的yaml路径，用于启用/禁用chart (e.g. subchart1.enabled )
    tags: # （可选）
      - 用于一次启用/禁用 一组chart的tag
    import-values: # （可选）
      - ImportValue 保存源值到导入父键的映射。每项可以是字符串或者一对子/父列表项
    alias: （可选） chart中使用的别名。当你要多次添加相同的chart时会很有用
maintainers: # （可选）
```

```
  - name: 维护者名字 （每个维护者都需要）
    email: 维护者邮箱 （每个维护者可选）
    url: 维护者URL （每个维护者可选）
icon: 用做icon的SVG或PNG图片URL （可选）
appVersion: 包含的应用版本（可选）。不需要是语义化，建议使用引号
deprecated: 不被推荐的chart （可选，布尔值）
annotations:
  example: 按名称输入的批注列表 （可选）.
```

Templates and Values

Helm Chart 模板是按照 Go模板语言书写，values.yaml中的值会替换各个模板中的变量。

模板的Value通过两种方式提供：

- values.yaml文件
- 命令行

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: deis-database
  namespace: deis
  labels:
    app.kubernetes.io/managed-by: deis
spec:
  replicas: 1
  selector:
    app.kubernetes.io/name: deis-database
  template:
    metadata:
      labels:
        app.kubernetes.io/name: deis-database
    spec:
      serviceAccount: deis-database
      containers:
        - name: deis-database
          image: {{ .Values.imageRegistry }}/postgres:{{ .Values.dockerTag }}
          imagePullPolicy: {{ .Values.pullPolicy }}
          ports:
            - containerPort: 5432
          env:
            - name: DATABASE_STORAGE
              value: {{ default "minio" .Values.storage }}
```

范围、依赖和值

Values文件可以声明顶级Chart的值，以及所依赖的Chart的值，定义格式如下（mysql和apache是以来的Chart）：

```yaml
title: "My WordPress Site" # Sent to the WordPress template

mysql:
  max_connections: 100 # Sent to MySQL
  password: "secret"

apache:
  port: 8080 # Passed to Apache
```

高阶Chart具有全局访问能力，低阶chart不能访问高阶数据，低阶访问自己的值可以取消掉名称空间，例如MySQL Chart访问自己的password可以直接通过`{{ .Values.password }}`

通过在values.yaml中定义`global`属性，可在所有chart中生效访问方式`{{ .Values.global.app }}`

架构文件`values.schema.json`

```json
{
  "$schema": "https://json-schema.org/draft-07/schema#",
  "properties": {
    "image": {
      "description": "Container Image",
      "properties": {
        "repo": {
          "type": "string"
        },
        "tag": {
          "type": "string"
        }
      },
      "type": "object"
    },
    "name": {
      "description": "Service name",
      "type": "string"
    },
    "port": {
      "description": "Port",
      "minimum": 0,
      "type": "integer"
    },
    "protocol": {
      "type": "string"
    }
  },
  "required": [
    "protocol",
    "port"
  ],
  "title": "Values",
  "type": "object"
}
```

- helm install

- helm upgrade
- helm lint
- helm template

这些命令会根据架构文件检查values.yaml的相关配置

Chart Hook

可用的Hook

注释值	描述
pre-install	在模板渲染之后，Kubernetes资源创建之前执行
post-install	在所有资源加载到Kubernetes之后执行
pre-delete	在Kubernetes删除之前，执行删除请求
post-delete	在所有的版本资源删除之后执行删除请求
pre-upgrade	在模板渲染之后，资源更新之前执行一个升级请求
post-upgrade	所有资源升级之后执行一个升级请求
pre-rollback	在模板渲染之后，资源回滚之前，执行一个回滚请求
post-rollback	在所有资源被修改之后执行一个回滚请求
test	调用Helm test子命令时执行

默认生命周期：

```
helm install -->
helm库调用安装API -->
验证，渲染模板 -->
加载资源to K8S -->
返回发布对象to Client -->
客户端退出
```

以pre-install 和 post-install定义为例的生命周:

```
helm install -->
helm库调用安装API -->
crds/目录中的CRD安装 -->
验证，渲染模板 -->
库准备执行pre-install钩子 -->
以权重顺序执行钩子 -->
库等待HOOK Ready -->
加载资源to K8S -->
库准备执post-install钩子 -->
```

```
以权重顺序执行钩子 -->
库等待HOOK Ready -->
返回发布对象to Client -->
客户端退出
```

钩子的定义同样放在模板中，通过Job资源的annotations生命HOOK属性：

```
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Release.Name }}"
  labels:
    app.kubernetes.io/managed-by: {{ .Release.Service | quote }}
    app.kubernetes.io/instance: {{ .Release.Name | quote }}
    app.kubernetes.io/version: {{ .Chart.AppVersion }}
    helm.sh/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,post-upgrade
    "helm.sh/hook-weight": "-5"
    "helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded,hook-failed
spec:
  template:
    metadata:
      name: "{{ .Release.Name }}"
      labels:
        app.kubernetes.io/managed-by: {{ .Release.Service | quote }}
        app.kubernetes.io/instance: {{ .Release.Name | quote }}
        helm.sh/chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
    spec:
      restartPolicy: Never
      containers:
      - name: post-install-job
        image: "alpine:3.3"
        command: ["/bin/sleep","{{ default "10" .Values.sleepyTime }}"]
```

Chart Test

```
{{- if .Values.mariadb.enabled }}
apiVersion: v1
kind: Pod
metadata:
  name: "{{ .Release.Name }}-credentials-test"
  annotations:
    "helm.sh/hook": test
spec:
  containers:
    - name: {{ .Release.Name }}-credentials-test
      image: {{ template "wordpress.image" . }}
      imagePullPolicy: {{ .Values.image.pullPolicy | quote }}
      {{- if .Values.securityContext.enabled }}
      securityContext:
```

```
        runAsUser: {{ .Values.securityContext.runAsUser }}
      {{- end }}
      env:
        - name: MARIADB_HOST
          value: {{ template "mariadb.fullname" . }}
        - name: MARIADB_PORT
          value: "3306"
        - name: WORDPRESS_DATABASE_NAME
          value: {{ default "" .Values.mariadb.db.name | quote }}
        - name: WORDPRESS_DATABASE_USER
          value: {{ default "" .Values.mariadb.db.user | quote }}
        - name: WORDPRESS_DATABASE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: {{ template "mariadb.fullname" . }}
              key: mariadb-password
      command:
        - /bin/bash
        - -ec
        - |
                    mysql --host=$MARIADB_HOST --port=$MARIADB_PORT --
user=$WORDPRESS_DATABASE_USER --password=$WORDPRESS_DATABASE_PASSWORD
  restartPolicy: Never
{{- end }}
```

测试安装：

```
$ helm install quirky-walrus wordpress --namespace default
$ helm test quirky-walrus
```

阿里云APPHUB

```
$ helm repo add apphub https://apphub.aliyuncs.com
```

模板实践

1. 创建mychart Chart

```
[root@kube01 ~]# helm create mychart
Creating mychart
[root@kube01 ~]# tree mychart/
mychart/
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
```

```
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 10 files
```

2. 清理模板

```
rm -rf mychart/templates/*
```

3. 创建configmap模板`mychart/templates/configmap.yaml`

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mychart-configmap
data:
  myvalue: "Hello World"
```

4. 尝试部署chart

```
[root@kube01 ~]# helm install helmstu ./mychart
NAME: helmstu
LAST DEPLOYED: Fri Jul 23 02:05:08 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
[root@kube01 ~]# helm list
NAME    NAMESPACE   REVISION    UPDATED                                 STATUS      CHART
APP VERSION
helmstu default     1           2021-07-23 02:05:08.477789184 -0400 EDT deployed
mychart-0.1.0   1.16.0
```

查看创建的资源

```
[root@kube01 ~]# helm get manifest helmstu
---
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mychart-configmap
data:
  myvalue: "Hello World"
```

```
------

[root@kube01 ~]# kubectl get configmaps
NAME                DATA   AGE
mychart-configmap   1      113s
```

5. 添加模板调用，变更`configmap.yaml`文件

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
```

6. 测试执行，查看渲染效果

```
[root@kube01 ~]# helm install --debug --dry-run ttt ./mychart/
install.go:173: [debug] Original chart version: ""
install.go:190: [debug] CHART PATH: /root/mychart

NAME: ttt
LAST DEPLOYED: Fri Jul 23 02:10:49 2021
NAMESPACE: default
STATUS: pending-install
REVISION: 1
TEST SUITE: None
USER-SUPPLIED VALUES:
{}

COMPUTED VALUES:
affinity: {}
autoscaling:
  enabled: false
  maxReplicas: 100
  minReplicas: 1
  targetCPUUtilizationPercentage: 80
fullnameOverride: ""
image:
  pullPolicy: IfNotPresent
  repository: nginx
  tag: ""
imagePullSecrets: []
ingress:
  annotations: {}
  className: ""
  enabled: false
  hosts:
  - host: chart-example.local
    paths:
    - path: /
      pathType: ImplementationSpecific
```

```
  tls: []
nameOverride: ""
nodeSelector: {}
podAnnotations: {}
podSecurityContext: {}
replicaCount: 1
resources: {}
securityContext: {}
service:
  port: 80
  type: ClusterIP
serviceAccount:
  annotations: {}
  create: true
  name: ""
tolerations: []

HOOKS:
MANIFEST:
---
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ttt-configmap
data:
  myvalue: "Hello World"
```

内置对象

- `Release`： 该对象描述了版本发布本身。包含了以下对象：
  - Release.Name： release名称
  - Release.Namespace： 版本中包含的命名空间(如果manifest没有覆盖的话)
  - Release.IsUpgrade： 如果当前操作是升级或回滚的话，需要将该值设置为true
  - Release.IsInstall： 如果当前操作是安装的话，需要将该值设置为true
  - Release.Revision： 此次修订的版本号。安装时是1，每次升级或回滚都会自增
  - Release.Service： 该service用来渲染当前模板。Helm里一般是Helm
- `Values`： Values是从values.yaml文件和用户提供的文件传进模板的。Values默认为空
- `Chart`： Chart.yaml文件内容。 Chart.yaml里的任意数据在这里都可以可访问的。比如 {{ .Chart.Name }}-{{ .Chart.Version }} 会打印出 mychart-0.1.0
- `Files`： 在chart中提供访问所有的非特殊文件。当你不能使用它访问模板时，你可以访问其他文件。 请查看这个 文件访问部分了解更多信息
  - Files.Get 通过文件名获取文件的方法。 （.Files.Getconfig.ini）
  - Files.GetBytes 用字节数组代替字符串获取文件内容的方法。 对图片之类的文件很有用
  - Files.Glob 用给定的shell glob模式匹配文件名返回文件列表的方法
  - Files.Lines 逐行读取文件内容的方法。迭代文件中每一行时很有用
  - Files.AsSecrets 使用Base 64编码字符串返回文件体的方法
  - Files.AsConfig 使用YAML格式返回文件体的方法

- `Capabilities`： 提供关于Kubernetes集群支持功能的信息
  - Capabilities.APIVersions 是一个版本集合
  - Capabilities.APIVersions.Has $version 说明集群中的版本 (e.g., batch/v1) 或是资源 (e.g., apps/v1/Deployment) 是否可用
  - Capabilities.KubeVersion 和 Capabilities.KubeVersion.Version 是Kubernetes的版本号
  - Capabilities.KubeVersion.Major Kubernetes的主版本
  - Capabilities.KubeVersion.Minor Kubernetes的次版本
- `Template`： 包含了已经被执行的当前模板信息
  - Template.Name: 当前模板的命名空间文件路径 (e.g. mychart/templates/mytemplate.yaml)
  - Template.BasePath: 当前chart模板目录的路径 (e.g. mychart/templates)

Values文件

1.在values.yaml中定义一个参数

```
favoriteDrink: coffee
```

2.在configmap填入模板引用

```
[root@kube01 ~]# vim mychart/values.yaml
[root@kube01 ~]# vim mychart/templates/configmap.yaml
[root@kube01 ~]# helm install --debug --dry-run ttt ./mychart/
install.go:173: [debug] Original chart version: ""
install.go:190: [debug] CHART PATH: /root/mychart

NAME: ttt
LAST DEPLOYED: Fri Jul 23 02:21:30 2021
NAMESPACE: default
STATUS: pending-install
REVISION: 1
TEST SUITE: None
USER-SUPPLIED VALUES:
{}

COMPUTED VALUES:
favoriteDrink: coffee

HOOKS:
MANIFEST:
---
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ttt-configmap
data:
  myvalue: "Hello World"
  drink: coffee
```

3. 命令行中覆盖values

```
helm install ttt ./mychart --dry-run --debug --set favoriteDrink=water
```

--set比values.yaml拥有更高的优先级

4. 变更values.yaml

```
favorite:
  drink: coffee
  food: pizza
```

变更configmap.yaml引入变量

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink }}
  food: {{ .Values.favorite.food }}
```

模板函数

模板函数遵循的语法是 `functionName arg1 arg2...`

quote

字符串引用

default

管道符 |

lookup

查找资源

命令	Lookup 函数
kubectl get pod mypod -n mynamespace	lookup "v1" "Pod" "mynamespace" "mypod"
kubectl get pods -n mynamespace	lookup "v1" "Pod" "mynamespace" ""

命令	Lookup 函数
kubectl get pods --all-namespaces	lookup "v1" "Pod" "" ""
kubectl get namespace mynamespace	lookup "v1" "Namespace" "" "mynamespace"
kubectl get namespaces	lookup "v1" "Namespace" "" ""

查找的资源可以进一步获取其属性

```
(lookup "v1" "Namespace" "" "mynamespace").metadata.annotations
```

当 `lookup` 返回一个对象列表时，可以循环迭代

```
{{ range $index, $service := (lookup "v1" "Service" "mynamespace" "").items }}
    {{/* do something with each service */}}
{{ end }}
```

逻辑运算符

`and` , `coalesce` , `default` , `empty` , `eq` , `fail` , `ge` , `gt` , `le` , `lt` , `ne` , `not` , `or`

函数清单链接地址

`https://helm.sh/docs/chart_template_guide/function_list/`

流程控制

if/else

```
{{ if PIPELINE }}
  # Do something
{{ else if OTHER PIPELINE }}
  # Do something else
{{ else }}
  # Default case
{{ end }}
```

举例:

```
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 -}}
```

注意通过 `{{-`, `-}}` 来控制换行和空格

with

```
{{ with PIPELINE }}
  # restricted scope
{{ end }}
```

举例:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  {{- end }}
```

range

```
favorite:
  drink: coffee
  food: pizza
pizzaToppings:
  - mushrooms
  - cheese
  - peppers
  - onions
```

举例:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
```

```
  food: {{ .food | upper | quote }}
  toppings: |-
    {{- range $.Values.pizzaToppings }}
    - {{ . | title | quote }}
    {{- end }}
  {{- end }}
```

变量的使用

```
toppings: |-
    {{- range $index, $topping := .Values.pizzaToppings }}
      {{ $index }}: {{ $topping }}
    {{- end }}
```

效果

```
  toppings: |-
      0: mushrooms
      1: cheese
      2: peppers
      3: onions
```