**helm简介**

很多人都使用过Ubuntu下的ap-get或者CentOS下的yum, 这两者都是Linux系统下的包管理工具。采用apt-get/yum,应用开发者可以管理应用包之间的依赖关系，发布应用；用户则可以以简单的方式查找、安装、升级、卸载应用程序。

我们可以将Helm看作Kubernetes下的apt-get/yum。Helm是Deis (https://deis.com/) 开发的一个用于kubernetes的包管理器。每个包称为一个Chart，一个Chart是一个目录（一般情况下会将目录进行打包压缩，形成name-version.tgz格式的单一文件，方便传输和存储）。

对于应用发布者而言，可以通过Helm打包应用，管理应用依赖关系，管理应用版本并发布应用到软件仓库。

对于使用者而言，使用Helm后不用需要了解Kubernetes的Yaml语法并编写应用部署文件，可以通过Helm下载并在kubernetes上安装需要的应用。

除此以外，Helm还提供了kubernetes上的软件部署，删除，升级，回滚应用的强大功能。

**Helm 组件及相关术语**

**Helm**

Helm 是一个命令行下的客户端工具。主要用于 Kubernetes 应用程序 Chart 的创建、打包、发布以及创建和管理本地和远程的 Chart 仓库。

**Tiller**

Tiller 是 Helm 的服务端，部署在 Kubernetes 集群中。Tiller 用于接收 Helm 的请求，并根据 Chart 生成 Kubernetes 的部署文件（ Helm 称为 Release ），然后提交给 Kubernetes 创建应用。Tiller 还提供了 Release 的升级、删除、回滚等一系列功能。

**Chart**

Helm 的软件包，采用 TAR 格式。类似于 APT 的 DEB 包或者 YUM 的 RPM 包，其包含了一组定义 Kubernetes 资源相关的 YAML 文件。

**Repoistory**

Helm 的软件仓库，Repository 本质上是一个 Web 服务器，该服务器保存了一系列的 Chart 软件包以供用户下载，并且提供了一个该 Repository 的 Chart 包的清单文件以供查询。Helm 可以同时管理多个不同的 Repository。

**Release**

使用 helm install 命令在 Kubernetes 集群中部署的 Chart 称为 Release。

注：需要注意的是：Helm 中提到的 Release 和我们通常概念中的版本有所不同，这里的 Release 可以理解为 Helm 使用 Chart 包部署的一个应用实例。

**Helm工作原理**

Chart Install 过程：

Helm从指定的目录或者tgz文件中解析出Chart结构信息

Helm将指定的Chart结构和Values信息通过gRPC传递给Tiller

Tiller根据Chart和Values生成一个Release

Tiller将Release发送给Kubernetes用于生成Release

Chart Update过程：

Helm从指定的目录或者tgz文件中解析出Chart结构信息

Helm将要更新的Release的名称和Chart结构，Values信息传递给Tiller

Tiller生成Release并更新指定名称的Release的History

Tiller将Release发送给Kubernetes用于更新Release

Chart Rollback过程：

Helm将要回滚的Release的名称传递给Tiller

Tiller根据Release的名称查找History

Tiller从History中获取上一个Release

Tiller将上一个Release发送给Kubernetes用于替换当前Release

**helm部署**

**Helm 客户端安装**

Helm 的安装方式很多，这里采用二进制的方式安装。更多安装方法可以参考 Helm 的官方帮助文档。

方式一：使用官方提供的脚本一键安装

curl https://raw.githubusercontent.com/helm/helm/master/scripts/get > get\_helm.sh

$ chmod 700 get\_helm.sh

$ ./get\_helm.sh

方式二：手动下载安装

#从官网下载最新版本的二进制安装包到本地：https://github.com/kubernetes/helm/releases

tar -zxvf helm-2.9.0.tar.gz # 解压压缩包

# 把 helm 指令放到bin目录下

mv helm-2.9.0/helm /usr/local/bin/helm

helm help # 验证

**Helm 服务端安装Tiller**

注意：先在 K8S 集群上每个节点安装 socat 软件(yum install -y socat )，不然会报如下错误：

E0522 22:22:15.49243624409 portforward.go:331] an error occurred forwarding 38398 -> 44134: error  
forwarding port 44134 to pod  
dc6da4ab99ad9c497c0cef1776b9dd18e0a612d507e2746ed63d36ef40f30174, uid : unable  
to do port forwarding: socat not found.

Error: cannot connect to Tiller

Tiller 是以 Deployment 方式部署在 Kubernetes 集群中的，只需使用以下指令便可简单的完成安装。

$ helm init

由于 Helm 默认会去 storage.googleapis.com 拉取镜像，如果你当前执行的机器不能访问该域名的话可以使用以下命令来安装：

helm init --client-only --stable-repo-url  
https://aliacs-app-catalog.oss-cn-hangzhou.aliyuncs.com/charts/

helm repo add incubator https://aliacs-app-catalog.oss-cn-hangzhou.aliyuncs.com/charts-incubator/

helm repo update

# 创建服务端

helm init --service-account tiller --upgrade -i  
registry.cn-hangzhou.aliyuncs.com/google\_containers/tiller:v2.9.1--stable-repo-url https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts

# 创建TLS认证服务端，参考地址：https://github.com/gjmzj/kubeasz/blob/master/docs/guide/helm.md

helm init --service-account tiller --upgrade -i  
registry.cn-hangzhou.aliyuncs.com/google\_containers/tiller:v2.9.1  
--tiller-tls-cert /etc/kubernetes/ssl/tiller001.pem --tiller-tls-key  
/etc/kubernetes/ssl/tiller001-key.pem --tls-ca-cert /etc/kubernetes/ssl/ca.pem  
--tiller-namespace kube-system --stable-repo-url  
https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts

在 Kubernetes 中安装 Tiller 服务，因为官方的镜像因为某些原因无法拉取，使用-i指定自己的镜像，可选镜像：registry.cn-hangzhou.aliyuncs.com/google\_containers/tiller:v2.9.1（阿里云），该镜像的版本与helm客户端的版本相同，使用helm version可查看helm客户端版本。

如果在用helm init安装tiller server时一直部署不成功,检查deployment，根据描述解决问题。

**给 Tiller 授权**

因为 Helm 的服务端 Tiller 是一个部署在 Kubernetes 中 Kube-System Namespace 下 的 Deployment，它会去连接 Kube-Api 在 Kubernetes 里创建和删除应用。

而从 Kubernetes 1.6 版本开始，API Server 启用了 RBAC 授权。目前的 Tiller 部署时默认没有定义授权的 ServiceAccount，这会导致访问 API Server 时被拒绝。所以我们需要明确为 Tiller 部署添加授权。

创建 Kubernetes 的服务帐号和绑定角色

$ kubectl create serviceaccount --namespace kube-system tiller

$ kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin  
--serviceaccount=kube-system:tiller

为 Tiller 设置帐号

# 使用 kubectl patch 更新 API 对象

$ kubectl patch deploy --namespace kube-system tiller-deploy -p  
'{"spec":{"template":{"spec":{"serviceAccount":"tiller"}}}}'

deployment.extensions "tiller-deploy" patched

查看是否授权成功

$ kubectl get deploy --namespace kube-system   tiller-deploy  --output yaml|grep  
 serviceAccount

serviceAccount: tiller

serviceAccountName: tiller

**四、验证 Tiller 是否安装成功**

$ kubectl -n kube-system get pods|grep tiller

tiller-deploy-6d68f5c78f-nql2z 1/1       Running   0  
         5m

$ helm version

Client: &version.Version{SemVer:"v2.9.1",  
GitCommit:"20adb27c7c5868466912eebdf6664e7390ebe710",  
GitTreeState:"clean"}

Server: &version.Version{SemVer:"v2.9.1",  
GitCommit:"20adb27c7c5868466912eebdf6664e7390ebe710",  
GitTreeState:"clean"}

**五、卸载 Helm 服务器端 Tiller**

如果你需要在 Kubernetes 中卸载已部署的 Tiller，可使用以下命令完成卸载。

$ helm reset 或

$helm reset --force

**Helm 的使用（常用命令，template 语法）**

**Helm 命令**

<https://helm.sh/docs/intro/using_helm/>

helm <command> --help.查看帮助

helm get manifest <releasename> 回显所有k8s资源信息，每个资源用---隔开

Helm status [releasename] 跟踪状态，以及一些其他信息

helm repo remove stable移除原先的仓库

helm repo add stable <https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts>添加新的仓库地址

helm repo update更新仓库可用chart列表

helm search 查看在存储库中可用的所有 Helm charts：

helm install [具体chart名] 安装charts. 安装chart的时候默认是根据values.yaml文件渲染的，也可以通过两种方式来传递配置数据

--values (or -f): Specify a YAML file with overrides. This can be specified multiple times and the rightmost file will take precedence

--set: Specify overrides on the command line.

If both are used, --set values are merged into --values with higher precedence. Overrides specified with --set are persisted in a ConfigMap. Values  
that have been --set can be viewed for a given release with helm get values  
<release-name>. Values that have been --set can be cleared by running  
helm upgrade with --reset-values specified.

--set XXX=null 可以删除某个value

--set 设置的键值对可以转换成yaml格式。

|  |  |
| --- | --- |
| --set 写法，省略了--set | 对应的yaml |
| name=value | name: value |
| a=b,c=d | a: b  c: d |
| outer.inner=value | outer:  inner: value |
| name={a, b, c} | name:  - a  - b  - c |
| servers[0].port=80 | servers:  - port: 80 |
| servers[0].port=80,servers[0].host=example | servers:  - port: 80  host: example |
| name=value1\,value2 | name: "value1,value2" |
| nodeSelector."kubernetes\.io/role"=master | nodeSelector:  kubernetes.io/role: master |

helm install --debug --dry-run <chart\_path>该命令会渲染templates并回显在终端。

helm list 查看K8S中已安装的charts：

helm upgrade -f panda.yaml happy-panda stable/mariadb这个名为happy-panda的release 使用panda.yaml文件升级。

helm rollback [RELEASE] [REVISION]回滚release到指定的版本revision

helm get values happy-panda查看新设置的变量

helm uninstall <release>卸载chart，

helm delete xxx删除

helm get XXX获取具体信息

helmshow values [具体chart名]查看chart可配置的项

helm create <chart名>创建chart

helm package <chart名>.打包chart

$ helm show values stable/mariadb

Fetched stable/mariadb-0.3.0.tgz to  
/Users/mattbutcher/Code/Go/src/helm.sh/helm/mariadb-0.3.0.tgz

## Bitnami MariaDB image version

## ref: https://hub.docker.com/r/bitnami/mariadb/tags/

##

## Default: none

imageTag: 10.1.14-r3

## Specify a imagePullPolicy

## Default to 'Always' if imageTag is 'latest', else set to 'IfNotPresent'

## ref: https://kubernetes.io/docs/user-guide/images/#pre-pulling-images

##

# imagePullPolicy:

## Specify password for root user

## ref: https://github.com/bitnami/bitnami-docker-mariadb/blob/master/README.md#setting-the-root-password-on-first-run

##

# mariadbRootPassword:

## Create a database user

## ref: https://github.com/bitnami/bitnami-docker-mariadb/blob/master/README.md#creating-a-database-user-on-first-run

##

# mariadbUser:

# mariadbPassword:

## Create a database

## ref: https://github.com/bitnami/bitnami-docker-mariadb/blob/master/README.md#creating-a-database-on-first-run

##

# mariadbDatabase:

# ...

# custom-repos.yaml 内容

cat custom-repos.yaml

api:

config:

repos:

- name: stable

url: https://aliacs-app-catalog.oss-cn-hangzhou.aliyuncs.com/charts

source: https://github.com/kubernetes/charts/tree/master/stable

- name: incubator

url: https://aliacs-app-catalog.oss-cn-hangzhou.aliyuncs.com/charts-incubator

source: https://github.com/kubernetes/charts/tree/master/incubator

- name: monocular

url: https://kubernetes-helm.github.io/monocular

source: https://github.com/kubernetes-helm/monocular/tree/master/charts

**developing templates**

<https://helm.sh/docs/topics/chart_template_guide/>

**helm templates 中的函数**

<https://masterminds.github.io/sprig/>其他常用的函数

<https://godoc.org/text/template>

管道 | 用法跟linux shell中的管道一致

Upper XXX变成大写

Quote XXX加引号

Repeat count string重复count次的string

举例 ： drink: {{.Values.favorite.drink | repeat 5 | quote}}

Empty .Bar如果bar为空则返回true，否则false。 在gotemplates语言中if empty .Foo. 等同于 if .Foo.

default "foo" .Bar 如果bar为空，那么bar=“foo”，如果有则不变

The definition of “empty” depends on type:

Numeric: 0

String: “”

Lists: []

Dicts: {}

Boolean: false

And always nil (aka null)

Default XXX给一个默认值，防止values省略了

举例 ： drink: {{ .Values.favorite.drink | default "tea" | quote }}如果Values中有drink，则使用，如果没有，则使用tea。Defaul值不建议用在静态值，但可以用在需要计算的值。

For templates, the operators (eq, ne, lt, gt, and, or and so on) are all implemented as functions. In pipelines,  
operations can be grouped with parentheses ((, and )).

ternary "foo" "bar" true三元操作。True返回第一个参数，false返回第二个参数

b64enc/b64dec: Encode or decode with Base64

b32enc/b32dec: Encode or decode with Base32

fail "Please accept the end user license agreement"fail函数会使程序失败

**template 流程控制**

* if/else for creating conditional blocks
* with to specify a scope
* range, which provides a “for each”-style loop

In addition to these, it provides a few actions for declaring and using named template segments:

* define declares a new named template inside of your template
* template imports a named template
* block declares a special kind of fillable template area

**if/else**

{{ if PIPELINE }}

# Do something

{{ else if OTHER PIPELINE }}

# Do something else

{{ else }}

# Default case

{{ end }}

A pipeline is evaluated as false if the value is:

a boolean false

a numeric zero

an empty string

a nil (empty or null)

an empty collection (map, slice, tuple, dict, array)

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 }}

**控制空白字符**

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 }}

{{- 表示移除左边的空白符，-}} 表示一直右边的空白符。注意- 跟内容中间必须有一个空白符。回车也是空白符也会被移除

**With**

{{ with PIPELINE }}

# restricted scope

{{ end }}

在global范围中，.表示根目录，.Values表示在values.yaml 中寻值。在with子范围中，重新定义了.，

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 }}

在with模块中，不能再使用.Value.Template 等全局的变量。必要要在end后面才能使用。

**Range 循环**

Range 在helm template 中用作循环

Values.yaml

favorite:

drink: coffee

food: pizza

pizzaToppings:

- mushrooms

- cheese

- peppers

- onions

Templates/config.yaml

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 }}

toppings: |-

{{- range .Values.pizzaToppings }}

- {{ . | title | quote }}

{{- end }}

**.** 一次代表循环的每一项

输出：

apiVersion: v1

kind: ConfigMap

metadata:

name: edgy-dragonfly-configmap

data:

myvalue: "Hello World"

drink: "coffee"

food: "PIZZA"

toppings: |-

- "Mushrooms"

- "Cheese"

- "Peppers"

- "Onions"

|- 表示多行string，在configmap中只能是key/value的键值对，这是k8s configmap 的语法。

也可以用tuple 来直接使用循环。

sizes: |-

{{- range tuple "small" "medium" "large" }}

- {{ . }}

{{- end }}

上面实际会变成

sizes: |-

- small

- medium

- large

**变量**

变量在helm template 中不常用。但是会用在with模块和range模块中

**在with 中使用**

apiVersion: v1

kind: ConfigMap

metadata:

name: {{ .Release.Name }}-configmap

data:

myvalue: "Hello World"

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

{{- with .Values.favorite }}

drink: {{ .drink | default "tea" | quote }}

food: {{ .food | upper | quote }}

release: {{ $relname }}

{{- end }}

可以看到$relname是变量名，:=是赋值符号。本来在with子范围内不能访问.Release.Name,通过在with外面定义变量，我们就能访问了。

# Source: mychart/templates/configmap.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: viable-badger-configmap

data:

myvalue: "Hello World"

drink: "coffee"

food: "PIZZA"

release: viable-badger

**在range中使用**

如果对象是list，则可以获取到index跟value；

toppings: |-

{{- range $index, $topping := .Values.pizzaToppings }}

{{ $index }}: {{ $topping }}

{{- end }}

产出

toppings: |-

0: mushrooms

1: cheese

2: peppers

3: onions

如果是dict，则可以获取到key和value。

apiVersion: v1

kind: ConfigMap

metadata:

name: {{ .Release.Name }}-configmap

data:

myvalue: "Hello World"

{{- range $key, $val := .Values.favorite }}

{{ $key }}: {{ $val | quote }}

{{- end }}

Now on the first iteration, $key will be drink and $val will be coffee, and on the second, $key will be food and $val  
will be pizza. Running the above will generate this:

# Source: mychart/templates/configmap.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: eager-rabbit-configmap

data:

myvalue: "Hello World"

drink: "coffee"

food: "pizza"

变量的作用范围：

变量在哪个scope定义，则作用范围仅限于该scope。如后面两个例子。在第一个例子红，$relname在全局定义，则全局有效。但是有一个变量 $ ,总是指向跟“root context”

{{- range .Values.tlsSecrets }}

apiVersion: v1

kind: Secret

metadata:

name: {{ .name }}

labels:

# Many helm templates would use `.` below, but that will not work,

# however `$` will work here

app.kubernetes.io/name: {{ template "fullname" $ }}

# I cannot reference .Chart.Name, but I can do $.Chart.Name

helm.sh/chart: "{{ $.Chart.Name }}-{{ $.Chart.Version }}"

app.kubernetes.io/instance: "{{ $.Release.Name }}"

# Value from appVersion in Chart.yaml

app.kubernetes.io/version: "{{ $.Chart.AppVersion }}"

app.kubernetes.io/managed-by: "{{ $.Release.Service }}"

type: kubernetes.io/tls

data:

tls.crt: {{ .certificate }}

tls.key: {{ .key }}

---

{{- end }}

**Named template**

Named template 是定义在一个文件中的template，并命名。

注意：这个名称是全局的，如果重名，后加载的覆盖前面加载的。这个全局的意思是不同chart也会导致名字冲突。所有传统的命名是加上chart名 {{ define “mychart.labels”}}

**Helm Chart 结构**

**Chart 目录结构**

examples/

Chart.yaml# Yaml文件，用于描述Chart的基本信息，包括名称版本等

LICENSE# [可选] 协议

README.md# [可选] 当前Chart的介绍

values.yaml# Chart的默认配置文件

requirements.yaml# [可选] 用于存放当前Chart依赖的其它Chart的说明文件

charts/# [可选]: 该目录中放置当前Chart依赖的其它Chart

templates/# [可选]: 部署文件模版目录，模版使用的值来自values.yaml和由Tiller提供的值

templates/NOTES.txt # [可选]: 放置Chart的使用指南

**Chart.yaml 文件**

name: [必须] Chart的名称

version: [必须] Chart的版本号，版本号必须符合 SemVer 2：http://semver.org/

description: [可选] Chart的简要描述

keywords:

-[可选] 关键字列表

home: [可选] 项目地址

sources:

- [可选] 当前Chart的下载地址列表

maintainers: # [可选]

- name: [必须] 名字

email: [可选] 邮箱

engine: gotpl # [可选] 模版引擎，默认值是gotpl

icon: [可选] 一个SVG或PNG格式的图片地址

**requirements.yaml 和 charts目录**

requirements.yaml 文件内容：

dependencies:

- name: example

version: 1.2.3

repository: http://example.com/charts

- name: Chart名称

version: Chart版本

repository: 该Chart所在的仓库地址

Chart支持两种方式表示依赖关系，可以使用requirements.yaml或者直接将依赖的Chart放置到charts目录中。

**templates 目录**

templates目录中存放了Kubernetes部署文件的模版。

例如：

# db.yaml

apiVersion: v1

kind: ReplicationController

metadata:

name: deis-database

namespace: deis

labels:

heritage: deis

spec:

replicas: 1

selector:

app: deis-database

template:

metadata:

labels:

app: 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}}

模版语法扩展了 golang/text/template的语法：

# 这种方式定义的模版，会去除test模版尾部所有的空行

{{- define "test"}}

模版内容

{{- end}}

# 去除test模版头部的第一个空行

{{- template "test" }}

用于yaml文件前置空格的语法：

# 这种方式定义的模版，会去除test模版头部和尾部所有的空行

{{- define "test" -}}

模版内容

{{- end -}}

# 可以在test模版每一行的头部增加4个空格，用于yaml文件的对齐

{{ include "test" | indent 4}}

**创建自己的chart**

我们创建一个名为mongodb的chart，看一看chart的文件结构。

$ helm create mongodb

$ tree mongodb

mongodb

├── Chart.yaml #Chart本身的版本和配置信息

├── charts #依赖的chart

├── templates #配置模板目录

│├── NOTES.txt #helm提示信息

│├── \_helpers.tpl #用于修改kubernetes objcet配置的模板

│├── deployment.yaml #kubernetes Deployment object

│└── service.yaml #kubernetes Serivce

└── values.yaml #kubernetes object configuration

2 directories, 6 files

**模板**

Templates目录下是yaml文件的模板，遵循Go template语法。使用过Hugo的静态网站生成工具的人应该对此很熟悉。

我们查看下deployment.yaml文件的内容。

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: {{ template "fullname" . }}

labels:

chart: "{{ .Chart.Name }}-{{ .Chart.Version | replace "+" "\_" }}"

spec:

replicas: {{ .Values.replicaCount }}

template:

metadata:

labels:

app: {{ template "fullname" . }}

spec:

containers:

- name: {{ .Chart.Name }}

image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"

imagePullPolicy: {{ .Values.image.pullPolicy }}

ports:

- containerPort: {{ .Values.service.internalPort }}

livenessProbe:

httpGet:

path: /

port: {{ .Values.service.internalPort }}

readinessProbe:

httpGet:

path: /

port: {{ .Values.service.internalPort }}

resources:

{{ toyaml .Values.resources | indent 12 }}

这是该应用的Deployment的yaml配置文件，其中的双大括号包扩起来的部分是Go template，其中的Values是在values.yaml文件中定义的：

# Default values for mychart.

# This is a yaml-formatted file.

# Declare variables to be passed into your templates.

replicaCount: 1

image:

repository: nginx

tag: stable

pullPolicy: IfNotPresent

service:

name: nginx

type: ClusterIP

externalPort: 80

internalPort: 80

resources:

limits:

cpu: 100m

memory: 128Mi

requests:

cpu: 100m

memory: 128Mi

比如在Deployment.yaml中定义的容器镜像image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"其中的：

.Values.image.repository就是nginx

.Values.image.tag就是stable

以上两个变量值是在create chart的时候自动生成的默认值。

我们将默认的镜像地址和tag改成我们自己的镜像harbor-001.jimmysong.io/library/nginx:1.9。

**检查配置和模板是否有效**

当使用kubernetes部署应用的时候实际上讲templates渲染成最终的kubernetes能够识别的yaml格式。

使用helm install --dry-run --debug <chart\_dir>命令来验证chart配置。该输出中包含了模板的变量配置与最终渲染的yaml文件。

$ helm install --dry-run --debug mychart

Created tunnel using local port: '58406'

SERVER: "localhost:58406"

CHART PATH: /Users/jimmy/Workspace/github/bitnami/charts/incubator/mean/charts/mychart

NAME:filled-seahorse

REVISION: 1

RELEASED: Tue Oct 24 18:57:13 2017

CHART: mychart-0.1.0

USER-SUPPLIED VALUES:

{}

COMPUTED VALUES:

image:

pullPolicy: IfNotPresent

repository: harbor-001.jimmysong.io/library/nginx

tag: 1.9

replicaCount: 1

resources:

limits:

cpu: 100m

memory: 128Mi

requests:

cpu: 100m

memory: 128Mi

service:

externalPort: 80

internalPort: 80

name: nginx

type: ClusterIP

HOOKS:

MANIFEST:

---

# Source: mychart/templates/service.yaml

apiVersion: v1

kind: Service

metadata:

name: filled-seahorse-mychart

labels:

chart: "mychart-0.1.0"

spec:

type: ClusterIP

ports:

- port: 80

targetPort: 80

protocol: TCP

name: nginx

selector:

app: filled-seahorse-mychart

---

# Source: mychart/templates/deployment.yaml

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: filled-seahorse-mychart

labels:

chart: "mychart-0.1.0"

spec:

replicas: 1

template:

metadata:

labels:

app: filled-seahorse-mychart

spec:

containers:

- name: mychart

image: "harbor-001.jimmysong.io/library/nginx:1.9"

imagePullPolicy: IfNotPresent

ports:

- containerPort: 80

livenessProbe:

httpGet:

path: /

port: 80

readinessProbe:

httpGet:

path: /

port: 80

resources:

limits:

cpu: 100m

memory: 128Mi

requests:

cpu: 100m

memory: 128Mi

我们可以看到Deployment和Service的名字前半截由两个随机的单词组成，最后才是我们在values.yaml中配置的值。

**部署到kubernetes**

在mychart目录下执行下面的命令将nginx部署到kubernetes集群上。

helm install .

NAME:eating-hound

LAST DEPLOYED: Wed Oct 25 14:58:15 2017

NAMESPACE: default

STATUS: DEPLOYED

RESOURCES:

==> v1/Service

NAMECLUSTER-IPEXTERNAL-IPPORT(S)AGE

eating-hound-mychart10.254.135.68<none>80/TCP0s

==> extensions/v1beta1/Deployment

NAMEDESIREDCURRENTUP-TO-DATEAVAILABLEAGE

eating-hound-mychart11100s

NOTES:

1. Get the application URL by running these commands:

export POD\_NAME=$(kubectl get pods --namespace default -l "app=eating-hound-mychart" -o  
jsonpath="{.items[0].metadata.name}")

echo "Visit http://127.0.0.1:8080 to use your application"

kubectl port-forward $POD\_NAME 8080:80

现在nginx已经部署到kubernetes集群上，本地执行提示中的命令在本地主机上访问到nginx实例。

export POD\_NAME=$(kubectl get pods --namespace default -l  
"app=eating-hound-mychart" -o  
jsonpath="{.items[0].metadata.name}")

echo "Visit http://127.0.0.1:8080 to use your application"

kubectl port-forward $POD\_NAME 8080:80

在本地访问http://127.0.0.1:8080即可访问到nginx。

查看部署的relaese

$ helm list

NAMEREVISIONUPDATEDSTATUSCHARTNAMESPACE

eating-hound1Wed Oct 25 14:58:15 2017DEPLOYEDmychart-0.1.0default

删除部署的release

$ helm delete eating-hound

release "eating-hound" deleted

**打包分享**

我们可以修改Chart.yaml中的helm chart配置信息，然后使用下列命令将chart打包成一个压缩文件。

helm package .

打包出mychart-0.1.0.tgz文件。

**将应用发布到 Repository**

虽然我们已经打包了 Chart 并发布到了 Helm 的本地目录中，但通过 helm search 命令查找，并不能找不到刚才生成的 mychart包。

$ helm search mychart

No results found

这是因为 Repository 目录中的 Chart 包还没有被 Helm 管理。通过 helm repo list 命令可以看到目前 Helm 中已配置的 Repository 的信息。

$ helm repo list

NAME URL

stable https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts

注：新版本中执行 helm init 命令后默认会配置一个名为 local 的本地仓库。

我们可以在本地启动一个 Repository Server，并将其加入到 Helm Repo 列表中。Helm Repository 必须以 Web 服务的方式提供，这里我们就使用 helm serve 命令启动一个 Repository Server，该 Server 缺省使用 $HOME/.helm/repository/local 目录作为 Chart 存储，并在 8879 端口上提供服务。

$ helm serve &

Now serving you on 127.0.0.1:8879

默认情况下该服务只监听 127.0.0.1，如果你要绑定到其它网络接口，可使用以下命令：

$ helm serve --address 192.168.100.211:8879 &

如果你想使用指定目录来做为 Helm Repository 的存储目录，可以加上 --repo-path 参数：

$ helm serve --address 192.168.100.211:8879 --repo-path /data/helm/repository/ --url  
http://192.168.100.211:8879/charts/

通过 helm repo index 命令将 Chart 的 Metadata 记录更新在 index.yaml 文件中:

# 更新 Helm Repository 的索引文件

$ cd /home/k8s/.helm/repository/local

$ helm repo index --url=http://192.168.100.211:8879 .

完成启动本地 Helm Repository Server 后，就可以将本地 Repository 加入 Helm 的 Repo 列表。

$ helm repo add local http://127.0.0.1:8879

"local" has been added to your repositories

现在再次查找 mychart 包，就可以搜索到了。

$ helm repo update

$ helm search mychart

NAME CHART VERSION APP VERSION DESCRIPTION

local/mychart 0.1.0         1.0         A Helm chart  
for Kubernetes

**依赖**

我们可以在requirement.yaml中定义应用所依赖的chart，例如定义对mariadb的依赖：

dependencies:

- name: mariadb

version: 0.6.0

repository: https://kubernetes-charts.storage.googleapis.com

使用helm lint .命令可以检查依赖和模板配置是否正确。

**helm升级和回退一个应用**

从上面 helm list 输出的结果中我们可以看到有一个 Revision（更改历史）字段，该字段用于表示某一个 Release 被更新的次数，我们可以用该特性对已部署的 Release 进行回滚。

修改 Chart.yaml 文件

将版本号从 0.1.0 修改为 0.2.0, 然后使用 helm package 命令打包并发布到本地仓库。

$ cat mychart/Chart.yaml

apiVersion: v1

appVersion: "1.0"

description: A Helm chart for Kubernetes

name: mychart

version: 0.2.0

$ helm package mychart

Successfully packaged chart and saved it to: /home/k8s/mychart-0.2.0.tgz

查询本地仓库中的 Chart 信息

我们可以看到在本地仓库中 mychart 有两个版本。

$ helm search mychart -l

NAME CHART VERSION APP VERSION DESCRIPTION

local/mychart 0.2.0         1.0         A Helm chart  
for Kubernetes

local/mychart 0.1.0         1.0         A Helm chart  
for Kubernetes

**升级一个应用**

现在用 helm upgrade 命令将已部署的 mike-test 升级到新版本。你可以通过 --version 参数指定需要升级的版本号，如果没有指定版本号，则缺省使用最新版本。

$ helm upgrade mike-test local/mychart

Release "mike-test" has been upgraded. Happy Helming!

LAST DEPLOYED: Mon Jul 23 10:50:25 2018

NAMESPACE: default

STATUS: DEPLOYED

RESOURCES:

==> v1/Pod(related)

NAME READY  STATUS   RESTARTS  AGE

mike-test-mychart-6d56f8c8c9-d685v 1/1    Running  0         9m

==> v1/Service

NAME TYPE       CLUSTER-IP    
   EXTERNAL-IP  PORT(S)  AGE

mike-test-mychart ClusterIP  10.254.120.177  <none>        
80/TCP   9m

==> v1beta2/Deployment

NAME DESIRED  CURRENT  UP-TO-DATE  
 AVAILABLE  AGE

mike-test-mychart 1        1        1      
      1          9m

NOTES:

1. Get the application URL by running these commands:

export POD\_NAME=$(kubectl get pods --namespace default -l  
"app=mychart,release=mike-test" -o  
jsonpath="{.items[0].metadata.name}")

echo "Visit http://127.0.0.1:8080 to use your application"

kubectl port-forward $POD\_NAME 8080:80

完成后，可以看到已部署的 mike-test 被升级到 0.2.0 版本。

$ helm list

NAME REVISION  UPDATED                
    STATUS    CHART         NAMESPACE

mike-test 2         Mon Jul 23 10:50:25 2018  DEPLOYED  
 mychart-0.2.0 default

**回退一个应用**

如果更新后的程序由于某些原因运行有问题，需要回退到旧版本的应用。首先我们可以使用 helm history 命令查看一个 Release 的所有变更记录。

$ helm history mike-test

REVISION UPDATED                    
STATUS      CHART         DESCRIPTION

1 Mon Jul 23 10:41:20 2018  SUPERSEDED  mychart-0.1.0  
Install complete

2 Mon Jul 23 10:50:25 2018  DEPLOYED    
 mychart-0.2.0 Upgrade complete

其次，我们可以使用下面的命令对指定的应用进行回退。

$ helm rollback mike-test 1

Rollback was a success! Happy Helming!

注：其中的参数 1 是 helm history 查看到 Release 的历史记录中 REVISION 对应的值。

最后，我们使用 helm list 和 helm history 命令都可以看到 mychart 的版本已经回退到 0.1.0 版本。

$ helm list

NAME REVISION  UPDATED                
    STATUS    CHART         NAMESPACE

mike-test 3         Mon Jul 23 10:53:42 2018  DEPLOYED  
 mychart-0.1.0 default

$ helm history mike-test

REVISION UPDATED                    
STATUS      CHART         DESCRIPTION

1 Mon Jul 23 10:41:20 2018  SUPERSEDED  mychart-0.1.0  
Install complete

2 Mon Jul 23 10:50:25 2018  SUPERSEDED  mychart-0.2.0  
Upgrade complete

3 Mon Jul 23 10:53:42 2018  DEPLOYED    
 mychart-0.1.0 Rollback to 1

**删除一个应用**

如果需要删除一个已部署的 Release，可以利用 helm delete 命令来完成删除。

$ helm delete mike-test

release "mike-test" deleted

确认应用是否删除，该应用已被标记为 DELETED 状态。

$ helm ls -a mike-test

NAME REVISION  UPDATED                
    STATUS  CHART         NAMESPACE

mike-test 3         Mon Jul 23 10:53:42 2018  DELETED mychart-0.1.0  
default

也可以使用 --deleted 参数来列出已经删除的 Release

$ helm ls --deleted

NAME REVISION  UPDATED                
    STATUS  CHART         NAMESPACE

mike-test 3         Mon Jul 23 10:53:42 2018  DELETED mychart-0.1.0  
default

从上面的结果也可以看出，默认情况下已经删除的 Release 只是将状态标识为 DELETED 了 ，但该 Release 的历史信息还是继续被保存的。

$ helm hist mike-test

REVISION UPDATED                    
STATUS      CHART         DESCRIPTION

1 Mon Jul 23 10:41:20 2018  SUPERSEDED  mychart-0.1.0  
Install complete

2 Mon Jul 23 10:50:25 2018  SUPERSEDED  mychart-0.2.0  
Upgrade complete

3 Mon Jul 23 10:53:42 2018  DELETED      
mychart-0.1.0 Deletion complete

如果要移除指定 Release 所有相关的 Kubernetes 资源和 Release 的历史记录，可以用如下命令：

$ helm delete --purge mike-test

release "mike-test" deleted

再次查看已删除的 Release，已经无法找到相关信息。

$ helm hist mike-test

Error: release: "mike-test" not found

# helm ls 命令也已均无查询记录。

$ helm ls --deleted

$ helm ls -a mike-test

**使用Helm 部署 Wordpress应用实例**

**部署wordpress**

以Wordpress 为例，包括 MySQL、PHP 和 Apache。

由于测试环境没有可用的 PersistentVolume（持久卷，简称 PV），这里暂时将其关闭。

**$ helm install --name wordpress-test --set "persistence.enabled=false,mariadb.persistence.enabled=false,serviceType=NodePort"  
 stable/wordpress**

NAMESPACE: default

STATUS: DEPLOYED

RESOURCES:

==> v1beta1/Deployment

NAME DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE

wordpress-test-mariadb 1        1        1    
        1          26m

wordpress-test-wordpress 1        1        1      
      1          26m

==> v1/Pod(related)

NAME READY  STATUS  
  RESTARTS  AGE

wordpress-test-mariadb-84b866bf95-n26ff 1/1    Running  1         26m

wordpress-test-wordpress-5ff8c64b6c-sgtvv 1/1    Running  6         26m

==> v1/Secret

NAME TYPE    DATA  AGE

wordpress-test-mariadb Opaque  2     26m

wordpress-test-wordpress Opaque  2     26m

==> v1/ConfigMap

NAME DATA  AGE

wordpress-test-mariadb 1     26m

wordpress-test-mariadb-tests 1     26m

==> v1/Service

NAME TYPE       CLUSTER-IP     EXTERNAL-IP  
 PORT(S)                    
AGE

wordpress-test-mariadb ClusterIP  10.254.99.67   <none>      
  3306/TCP                  
 26m

wordpress-test-wordpress NodePort   10.254.175.16  <none>        
80:8563/TCP,443:8839/TCP  26m

NOTES:

1. Get the WordPress URL:

Or running:

export NODE\_PORT=$(kubectl get --namespace default -o  
jsonpath="{.spec.ports[0].nodePort}" services  
wordpress-test-wordpress)

export NODE\_IP=$(kubectl get nodes --namespace default -o  
jsonpath="{.items[0].status.addresses[0].address}")

echo http://$NODE\_IP:$NODE\_PORT/admin

2. Login with the following credentials to see your blog

echo Username: user

echo Password: $(kubectl get secret --namespace default wordpress-test-wordpress -o  
jsonpath="{.data.wordpress-password}" | base64 --decode)

**访问 Wordpress**

部署完成后，我们可以通过上面的提示信息生成相应的访问地址和用户名、密码等相关信息。

# 生成 Wordpress 管理后台地址

$ export NODE\_PORT=$(kubectl get --namespace default -o  
jsonpath="{.spec.ports[0].nodePort}" services  
wordpress-test-wordpress)

$ export NODE\_IP=$(kubectl get nodes --namespace default -o  
jsonpath="{.items[0].status.addresses[0].address}")

$ echo http://$NODE\_IP:$NODE\_PORT/admin

http://192.168.100.211:8433/admin

# 生成 Wordpress 管理帐号和密码

$ echo Username: user

Username: user

$ echo Password: $(kubectl get secret --namespace default wordpress-test-wordpress -o  
jsonpath="{.data.wordpress-password}" | base64 --decode)

Password: 9jEXJgnVAY

给一张访问效果图吧：

**Helm 其它使用技巧**

**如何设置 helm 命令自动补全？**

为了方便 helm 命令的使用，Helm 提供了自动补全功能，如果使用 ZSH 请执行：

$ source <(helm completion zsh)

如果使用 BASH 请执行：

$ source <(helm completion bash)

**如何使用第三方的 Chart 存储库？**

helm repo add monocular https://kubernetes-helm.github.io/monocular

随着 Helm 越来越普及，除了使用预置官方存储库，三方仓库也越来越多了（前提是网络是可达的）。你可以使用如下命令格式添加三方 Chart 存储库。

$ helm repo add 存储库名 存储库URL

$ helm repo update

一些三方存储库资源:

# Prometheus Operator

https://github.com/coreos/prometheus-operator/tree/master/helm

# Bitnami Library for Kubernetes

https://github.com/bitnami/charts

# Openstack-Helm

https://github.com/att-comdev/openstack-helm

https://github.com/sapcc/openstack-helm

# Tick-Charts

https://github.com/jackzampolin/tick-charts

**Helm 如何结合 CI/CD ？**

采用 Helm 可以把零散的 Kubernetes 应用配置文件作为一个 Chart 管理，Chart 源码可以和源代码一起放到 Git 库中管理。通过把 Chart 参数化，可以在测试环境和生产环境采用不同的 Chart 参数配置。

下图是采用了 Helm 的一个 CI/CD 流程

**Helm 如何管理多环境下 (Test、Staging、Production) 的业务配置？**

Chart 是支持参数替换的，可以把业务配置相关的参数设置为模板变量。使用 helm install 命令部署的时候指定一个参数值文件，这样就可以把业务参数从 Chart 中剥离了。例如： helm install --values=values-production.yaml wordpress。

**Helm 如何解决服务依赖？**

在 Chart 里可以通过 requirements.yaml 声明对其它 Chart 的依赖关系。如下面声明表明 Chart 依赖 Apache 和 MySQL 这两个第三方 Chart。

dependencies:

- name: mariadb

version: 2.1.1

repository: https://kubernetes-charts.storage.googleapis.com/

condition: mariadb.enabled

tags:

- wordpress-database

- name: apache

version: 1.4.0

repository: https://kubernetes-charts.storage.googleapis.com/

**如何让 Helm 连接到指定 Kubernetes 集群？**

Helm 默认使用和 kubectl 命令相同的配置访问 Kubernetes 集群，其配置默认在 ~/.kube/config 中。

**如何在部署时指定命名空间？**

helm install 默认情况下是部署在 default 这个命名空间的。如果想部署到指定的命令空间，可以加上 --namespace 参数，比如：

$ helm install local/mychart --name mike-test --namespace mynamespace

**如何查看已部署应用的详细信息？**

$ helm get wordpress-test

默认情况下会显示最新的版本的相关信息，如果想要查看指定发布版本的信息可加上 --revision 参数。

$ helm get  --revision 1  wordpress-test