部署管理

helm

基础知识

学习目标

这一节，我们从 基础知识、原理解析、小结 三个方面来学习。

基础知识

需求

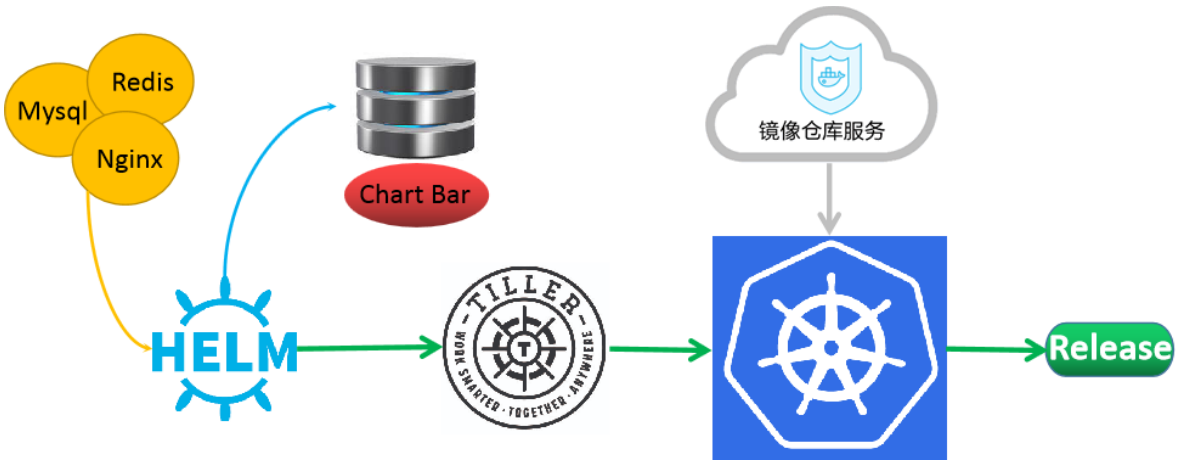
在kubernetes平台上，我们在部署各种各样的应用服务的时候，可以基于手工或者自动的方式对各种资源对象实现伸缩操作，尤其是对于有状态的应用，我们可以结合持久性存储机制实现更大场景的伸缩动作。
但是，无论我们怎么操作各种资源对象，问题最多的就是各种基础配置、镜像等之类的依赖管理操作。在linux平台下，常见的包依赖的就是yum、apt等工具，在kubernetes平台下，同样有类似的解决依赖关系的工具 -- helm。

官方网址：https://v3.helm.sh/
官方地址：https://github.com/helm/helm
官方仓库：https://hub.kubeapps.com/ 和 https://artifacthub.io/
最新版本：3.7.0 | 20211014

V2 简介

helm的功能类似于yum 或 apt，提供应用部署时候所需要的各种配置、资源清单文件，他与yum之类工具不同的是，在k8s中helm是不提供镜像的，这些镜像文件需要由专门的镜像仓库来提供。

例如：k8s平台上的nginx应用部署，对于该应用部署来说，主要需要三类内容：
镜像：nginx镜像
资源定义文件：Deployment、service、hpa等
专用文件：配置文件、证书等
helm管理的主要是：资源定义文件和专用文件。



Tiller Server是一个部署在Kubernetes集群内部的 server，其与 Helm client、Kubernetes API server 进行交互。Tiller server 主要负责如下：

监听来自 Helm client 的请求
通过 chart 及其配置构建一次发布
安装 chart 到Kubernetes集群，并跟踪随后的发布
通过与Kubernetes交互升级或卸载 chart

简单的说，client 管理 charts，而 server 管理发布 release

流程解析

基于helm来成功的部署一个应用服务，完整的工作流程如下：

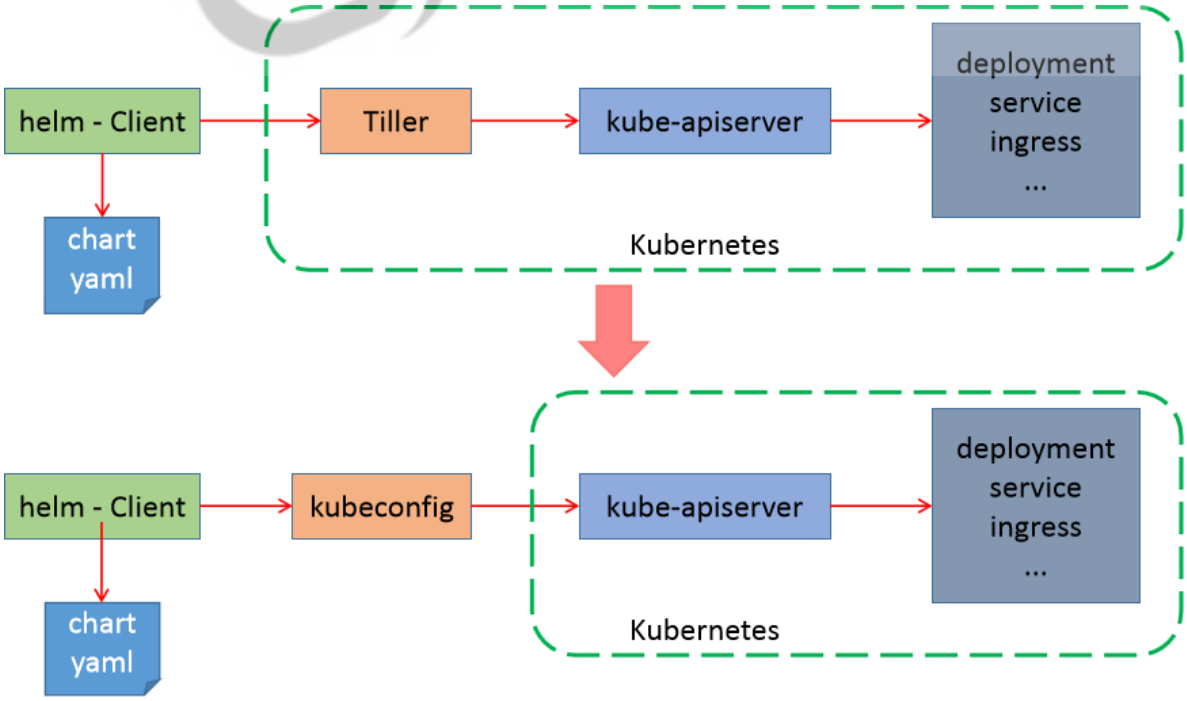
1 部署一个稳定运行的k8s集群，在能管理k8s的主机上部署helm。
2 用户在客户端主机上，定制各种Chart资源和config资源，上传到专用的仓库(本地或者远程)
3 helm客户端向Tiller发出部署请求，如果本地有chart用本地的，否则从仓库获取
4 Tiller与k8s集群的api-server发送请求
5 api-server通过集群内部机制部署应用，需要依赖镜像的时候，从专门的镜像仓库获取。
6 基于helm部署好的应用实例，在k8s集群中，我们称之为release。

v3简介

根据我们对 helm v2 版本的流程解析，我们发现，在客户端上部署tiller来维护 release相关的信息，有些太重量级了，所以在 helm v3 版本的时候，就剔除了专门的Tiller。

根据我们之前对于flannel的理解，flannel的专用数据存储方式可以使用独立的etcd集群来进行数据的存储，也可以借助于apiservice方式将数据存储到etcd里面。
所以helm借鉴与此，也可以通过apiserver来进行chart数据的存储，包括运行出来的release信息，也借助于apiserver存储到etcd里面。

所以：
在 Helm v3 中移除了 Tiller，版本相关的数据直接存储在了 Kubernetes 中.



kustomize vs helm

通过上面对 kustomize 的讲解，可能已经有人注意到它与 Helm 有一定的相似。先来看看 Helm 的定位：Kubernetes 的包管理工具，而 kustomize 的定位是：Kubernetes 原生配置管理。两者定位领域有很大不同，Helm 通过将应用抽象成 Chart 来管理，专注于应用的操作、复杂性管理等，而 kustomize 关注于 k8s API 对象的管理。

总的来说，Helm 有自己一套体系来管理应用，而 kustomize 更轻量级，融入Kubernetes 的设计理念，通过原生 k8s API 对象来管理应用

helm vs operator

helm 仅仅是部署的功能
operator
部署只是他的功能之一，他的核心功能是确保应用程序符合k8s的api的资源管理规范的基础上，通过代码级别的控制，确保应用能够按照用户所定义的期望状态进行执行。

小结

环境安装

学习目标

这一节，我们从 软件部署、简单实践、小结 三个方面来学习。

软件部署

由于helm默认使用别人定义好的配置文件来进行环境的部署，所以，我们可以借助于两种方式来进行helm的部署：
方法1： helm命令行选项传递参数
方法2： 通过值文件传递参数values.yaml

软件安装

```
下载软件
cd /data/softs
wget https://get.helm.sh/helm-v3.7.1-linux-amd64.tar.gz

配置环境
mkdir /data/server/helm/bin -p
tar xf helm-v3.7.1-linux-amd64.tar.gz
mv linux-amd64/helm /data/server/helm/bin/

环境变量
# cat /etc/profile.d/helm.sh
#!/bin/bash
# set helm env
export PATH=$PATH:/data/server/helm/bin

chmod +x /etc/profile.d/helm.sh
source /etc/profile.d/helm.sh

确认效果
# helm  version
```

```
version.BuildInfo{Version:"v3.7.1",
GitCommit:"1d11fcb5d3f3bf00dbe6fe31b8412839a96b3dc4", GitTreeState:"clean",
GoVersion:"go1.16.9"}
```

```
查看命令帮助
# helm --help
The Kubernetes package manager

Common actions for Helm:

- helm search:    search for charts
- helm pull:      download a chart to your local directory to view
- helm install:   upload the chart to Kubernetes
- helm list:      list releases of charts
...
Available Commands:
  completion  generate autocompletion scripts for the specified shell
  create      create a new chart with the given name
  dependency  manage a chart's dependencies
  env         helm client environment information
  get         download extended information of a named release
  help        Help about any command
  history     fetch release history
  install     install a chart
  lint        examine a chart for possible issues
  list        list releases
  package     package a chart directory into a chart archive
  plugin      install, list, or uninstall Helm plugins
  pull        download a chart from a repository and (optionally) unpack it in
local directory
  repo        add, list, remove, update, and index chart repositories
  rollback    roll back a release to a previous revision
  search      search for a keyword in charts
  show        show information of a chart
  status      display the status of the named release
  template    locally render templates
  test        run tests for a release
  uninstall   uninstall a release
  upgrade     upgrade a release
  verify      verify that a chart at the given path has been signed and is valid
  version     print the client version information
  ...

命令解析：
    list    查看部署的版本应用
    repo    对chart进行版本操作
```

仓库管理

```
# 添加官方源 - 不推荐
helm repo add stable https://kubernetes-charts.storage.googleapis.com
helm repo add incubator https://kubernetes-charts-
incubator.storage.googleapis.com

# 推荐
helm repo add az-stable http://mirror.azure.cn/kubernetes/charts/
```

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

查看仓库

```
# helm repo list
NAME            URL
az-stable       http://mirror.azure.cn/kubernetes/charts/
bitnami         https://charts.bitnami.com/bitnami
```

更新仓库属性信息

```
# helm repo update
```

搜索chart信息

```
# helm search --help
...
Available Commands:
  hub         search for charts in the Artifact Hub or your own hub instance
  repo        search repositories for a keyword in charts
```

结果显示：

helm 有两种搜索的源地址，官方的在 Artifact，幸运的是，无法访问。

从自定义仓库中获取源信息

```
helm search repo redis
```

```
root@master1:/etc/profile.d# helm search repo redis
NAME                                    CHART VERSION   APP VERSION     DESCRIPTION
az-stable/prometheus-redis-exporter     3.5.1           1.3.4           DEPRECATED Prometheus exporter for Redis metrics
az-stable/redis                         10.5.7          5.0.7           DEPRECATED Open source, advanced key-value stor...
az-stable/redis-ha                      4.4.6           5.0.6           DEPRECATED - Highly available Kubernetes implem...
bitnami/redis                           15.5.1          6.2.6           Open source, advanced key-value store. It is of...
bitnami/redis-cluster                   7.0.7           6.2.6           Open source, advanced key-value store. It is of...
az-stable/sensu                         0.2.5           0.28            DEPRECATED Sensu monitoring framework backed by...
```

查看chart的所有信息

```
helm show all bitnami/redis
```

```
root@master1:/etc/profile.d# helm show all bitnami/redis
annotations:
  category: Database
apiVersion: v2
appVersion: 6.2.6
dependencies:
- name: common
  repository: https://charts.bitnami.com/bitnami
  tags:
  - bitnami-common
  version: 1.x.x
description: Open source, advanced key-value store. It is often referred to as a data
  structure server since keys can contain strings, hashes, lists, sets and sorted
  sets.
home: https://github.com/bitnami/charts/tree/master/bitnami/redis
icon: https://bitnami.com/assets/stacks/redis/img/redis-stack-220x234.png
keywords:
```

使用all功能，可以查看所有的信息，非常多，如果想看部分信息的话，可以将 all更换成以下几条命令：

```
Available Commands:
  all         show all information of the chart
  chart       show the chart's definition
  crds        show the chart's CRDs
  readme      show the chart's README
  values      show the chart's values
```

获取chart文件

```
 helm pull bitnami/redis
```

应用管理

chart的安装主要分为以下两种方式：

安装本地chart
    指定本地chart目录：helm install /path/to/chart_dir
    指定本地chart压缩包：helm install chart_name.tgz

安装chart仓库中的chart
    使用默认的远程仓库：helm install repo/chart_name
    使用指定的仓库：helm install web_address:port/url/to/chart_name.tgz
    注意：
        所谓的远程仓库，本质上是将打包后chart作为静态文件托管到了web服务器上。

简单实践

安装chart
helm install my_helm bitnami/redis
注意：
    这种情况下，helm会视图使用默认配置，把helm里面chart的信息渲染出来，尝试部署redis
    部署出来的应用名称不要使用 _ 等特殊符号

```
9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*')
root@master1:/etc/profile.d# helm install my-helm bitnami/redis
NAME: my-helm
LAST DEPLOYED: Sun Oct 24 17:54:49 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: redis
CHART VERSION: 15.5.1
APP VERSION: 6.2.6

** Please be patient while the chart is being deployed **

Redis&trade; can be accessed on the following DNS names from within your cluster:

    my-helm-redis-master.default.svc.cluster.local for read/write operations (port 6379)
    my-helm-redis-replicas.default.svc.cluster.local for read-only operations (port 6379)



To get your password run:

    export REDIS_PASSWORD=$(kubectl get secret --namespace default my-helm-redis -o jsonpath="{.data.redis-password}" | base64 --de
code)

To connect to your Redis&trade; server:

1. Run a Redis&trade; pod that you can use as a client:

   kubectl run --namespace default redis-client --restart='Never'  --env REDIS_PASSWORD=$REDIS_PASSWORD  --image docker.io/bitnami/
redis:6.2.6-debian-10-r0 --command -- sleep infinity

   Use the following command to attach to the pod:

   kubectl exec --tty -i redis-client \
   --namespace default -- bash

2. Connect using the Redis&trade; CLI:
   redis-cli -h my-helm-redis-master -a $REDIS_PASSWORD
   redis-cli -h my-helm-redis-replicas -a $REDIS_PASSWORD

To connect to your database from outside the cluster execute the following commands:

    kubectl port-forward --namespace default svc/my-helm-redis-master 6379:6379 &
    redis-cli -h 127.0.0.1 -p 6379 -a $REDIS_PASSWORD
root@master1:/etc/profile.d#
```

不同操作权限的 Redis&trade:
    my-helm-redis-master.default.svc.cluster.local for read/write operations (port 6379)
    my-helm-redis-replicas.default.svc.cluster.local for read-only operations (port 6379)

获取密码
    export REDIS_PASSWORD=$(kubectl get secret --namespace default my-helm-redis -o jsonpath="{.data.redis-password}" | base64 --decode)

连接redis应用:

```
1. 运行一个 Redis&trade pod:
   kubectl run --namespace default redis-client --restart='Never'  --env
REDIS_PASSWORD=$REDIS_PASSWORD  --image docker.io/bitnami/redis:6.2.6-debian-10-
r0 --command -- sleep infinity

   连接对应的pod:

   kubectl exec --tty -i redis-client \
   --namespace default -- bash

2. 连接使用的 Redis&trade; CLI:
   redis-cli -h my-helm-redis-master -a $REDIS_PASSWORD
   redis-cli -h my-helm-redis-replicas -a $REDIS_PASSWORD
   连接数据库:
    kubectl port-forward --namespace default svc/my-helm-redis-master 6379:6379
&
    redis-cli -h 127.0.0.1 -p 6379 -a $REDIS_PASSWORD
```

查看安装效果

```
helm list
kubectl get pod
```

```
root@master1:~# helm list
NAME    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
my-helm default         1               2021-10-24 17:54:49.026276721 +0800 CST deployed        redis-15.5.1    6.2.6
root@master1:~# kubectl get pod
NAME                    READY   STATUS    RESTARTS   AGE
my-helm-redis-master-0     0/1     Pending   0          5m38s
my-helm-redis-replicas-0   0/1     Pending   0          5m38s
```

结果显示:

已经部署了一个helm应用，但是因为使用默认的chart相关的配置，而我们本地执行的时候，没有添加，所以这里会创建应用失败。

我们可以通过 helm命令参数 或者 helm values.yaml 文件的方式来进行操作

当我们通过helm 部署应用的时候，会自动在当前用户家目录的.cache 目录下生成缓存文件

```
# ls .cache/helm/repository/
az-stable-charts.txt  az-stable-index.yaml  bitnami-charts.txt  bitnami-
index.yaml  redis-15.5.1.tgz

其中 redis-15.5.1.tgz 就是打包好的chart文件
# tar xf .cache/helm/repository/redis-15.5.1.tgz -C /tmp/
# ls /tmp/redis/
Chart.lock  charts  Chart.yaml  ci  img  README.md  templates
values.schema.json  values.yaml
```

文件解析:

values.yaml 就是当前redis应用的各种属性的定制

取消默认的持久化存储方式部署redis

```
# helm uninstall my-helm
# helm install my-helm bitnami/redis --set master.persistence.enabled=false --
set replica.persistence.enabled=false
```

```
root@master1:~#  helm list
NAME    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
my-helm default         1               2021-10-24 18:12:40.614048679 +0800 CST deployed        redis-15.5.1    6.2.6
root@master1:~# kubectl get pod
NAME                     READY   STATUS              RESTARTS   AGE
my-helm-redis-master-0     0/1     ContainerCreating   0          49s
my-helm-redis-replicas-0   0/1     Pending             0          49s
```

结果显示：

应用已经开始创建了，有镜像已经开始拉取了，因为这些镜像来源于外部环境，所以创建的时候非常慢。

简单实践

查看基本操作的信息

```
helm status my-helm
```

获取具备读写权限的主机域名

redis主角色主机： my-helm-redis-master.default.svc.cluster.local

redis从角色主机： my-helm-redis-replicas.default.svc.cluster.local

获取连接密码

```
# export REDIS_PASSWORD=$(kubectl get secret --namespace default my-helm-redis -o jsonpath="{.data.redis-password}" | base64 --decode)
# echo $REDIS_PASSWORD
ID6KzPAZc1
```

创建客户端

```
# kubectl run --namespace default redis-client --restart='Never'  --env REDIS_PASSWORD=$REDIS_PASSWORD  --image docker.io/bitnami/redis:6.2.6-debian-10-r0 --command -- sleep infinity
```

连接redis主角色

```
$ redis-cli -h my-helm-redis-master.default.svc.cluster.local -a ID6KzPAZc1
```

redis操作

```
my-helm-redis-master.default.svc.cluster.local:6379> set a 1
OK
my-helm-redis-master.default.svc.cluster.local:6379> set b 2
OK
my-helm-redis-master.default.svc.cluster.local:6379> keys *
1) "a"
2) "b"
my-helm-redis-master.default.svc.cluster.local:6379> get a
"1"
```

通过values文件方式来定制环境

准备配置文件

```
cd /tmp/redis/
cp values.yaml values-define.yaml
```

定制配置文件

```
# grep -Env '#|^$' values-define.yaml
...
75:image:
76:  registry: 10.0.0.19:80
77:  repository: mykubernetes/redis
78:  tag: 6.2.5
```

```
83:  pullPolicy: IfNotPresent
...
143:master:
...
360:  persistence:
363:    enabled: false
...
444:replica:
...
672:  persistence:
675:    enabled: false
```

基于专用配置文件创建应用

```
helm install my-helm-2 bitnami/redis -f /tmp/redis/values-define.yaml
```

查看效果

```
root@master1:~# helm list
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
my-helm         default         1               2021-10-24 18:23:35.190800463 +0800 CST deployed        redis-15.5.1    6.2.6
my-helm-2       default         1               2021-10-24 18:36:45.915750283 +0800 CST deployed        redis-15.5.1    6.2.6
root@master1:~# kubectl get pod
NAME                           READY   STATUS    RESTARTS   AGE
my-helm-2-redis-master-0       1/1     Running   0          4m14s
my-helm-2-redis-replicas-0     1/1     Running   0          4m14s
my-helm-2-redis-replicas-1     1/1     Running   0          3m38s
my-helm-2-redis-replicas-2     1/1     Running   0          3m12s
my-helm-redis-master-0         1/1     Running   0          17m
my-helm-redis-replicas-0       1/1     Running   0          17m
my-helm-redis-replicas-1       1/1     Running   0          16m
my-helm-redis-replicas-2       1/1     Running   0          10m
redis-client                   1/1     Running   0          13m
root@master1:~# kubectl describe  pod my-helm-2-redis-master-0
Name:         my-helm-2-redis-master-0
Namespace:    default
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  4m24s  default-scheduler  Successfully assigned default/my-helm-2-redis-master-0 to node2
  Normal  Pulled     4m22s  kubelet            Container image "10.0.0.19:80/mykubernetes/redis:6.2.5" already present on machine
  Normal  Created    4m22s  kubelet            Created container redis
  Normal  Started    4m22s  kubelet            Started container redis
```

小结

简单实践

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

> 我们接下来要部署一个应用。就是一个简单的基于nginx的hello world HTTP服务。
> - 该服务通过读取环境变量USERNAME获得用户自己定义的名称，然后监听80端口。
> - 对于任意HTTP请求，返回Hello ${USERNAME}。
> - 如果设置USERNAME=world（默认场景），该服务会返回Hello world。

简单实践

```
进入项目目录
mkdir /data/server/helm_pro -p
cd /data/server/helm_pro

创建项目
helm create nginx-helloworld
```

```
查看效果
root@master1:/data/server/helm_pro# tree nginx-helloworld/
nginx-helloworld/                       - 自动生成的空chart，名称就是nginx-helloworld
├── charts
├── Chart.yaml                          - 声明了当前Chart的名称、版本等基本信息，便于用户在仓库里浏览检索
├── templates                           - 存放各种资源清单文件
|   ├── deployment.yaml
|   ├── _helpers.tpl                    - 定制的模板功能文件，包含各种需要根据值结果进行调整的功能
|   ├── hpa.yaml
|   ├── ingress.yaml
|   ├── NOTES.txt
|   ├── serviceaccount.yaml
|   ├── service.yaml
|   └── tests
|       └── test-connection.yaml
└── values.yaml                         - 存放各种定制的变量值，符合yaml格式语法

3 directories, 10 files
结果显示：
    需要注意的是，Chart里面的my-hello-world名称需要和生成的Chart文件夹名称一致。
    如果修改my-hello-world，则需要做一致的修改。 现在，我们看到Chart的文件夹目录如下
```

文件样式

```
helm创建好的chart文件结构中，包含了大量的k8s的资源清单文件，这些文件都是基于大量模板语言创建的
# cat nginx-helloworld/templates/serviceaccount.yaml
{{- if .Values.serviceAccount.create -}}
    # 如果在项目的根目录下存在Values文件，并且里面的serviceAccount部分的create属性为true的话。执行下面的操作
    # {{- 表达式 -}}，横杠(-)表示去掉表达式输出结果前面和后面的空格,可以使用单个
    # 最左面的点(.)，代表全局作用域-项目根目录，中间的点，是对象中属性的引用方式
apiVersion: v1
kind: ServiceAccount
metadata:
  name: {{ include "nginx-helloworld.serviceAccountName" . }}
    # 加载项目的templates/_helpers.tpl文件里面定义好的 nginx-helloworld.serviceAccountName 内容
  labels:
    {{- include "nginx-helloworld.labels" . | nindent 4 }}
    # 加载项目的templates/_helpers.tpl文件里面定义好的 nginx-helloworld.serviceAccountName 内容
  {{- with .Values.serviceAccount.annotations }}
   # 加载项目的根目录下Values文件中serviceAccount的annotations属性值
   # with语法代表修改内容
  annotations:
    {{- toYaml . | nindent 4 }}
```

```
    # toYaml 表示将数据转为yaml格式，nindent表示不局限于4行，indent 4代表就是格式化转换为4行
  {{- end }}
{{- end }}

helm为了更好的管理chart接口，提供了一个非常重要的数据文件，这个文件里面可以定制各种各样的数据，被yaml文件来采用
# cat nginx-helloworld/values.yaml
...
serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name: ""
...

helm内部定义了很多自定义的模板功能属性值,可以根据属性值来进行调整。
# cat nginx-helloworld/templates/_helpers.tpl
...
{{/*
Create the name of the service account to use
*/}}
{{- define "nginx-helloworld.serviceAccountName" -}}
{{- if .Values.serviceAccount.create }}
{{- default (include "nginx-helloworld.fullname" .) .Values.serviceAccount.name }}
{{- else }}
{{- default "default" .Values.serviceAccount.name }}
{{- end }}
{{- end }}
```

简单实践

需求

```
    因为我们做的是一个nginx网站首页，所以我们需要进行一个简单的调整，而且调整的对象内容是development.yaml 及 service.yaml 文件。
```

修改定制文件

```
查看文件deployment的文件内容，方便调整
# cat nginx-helloworld/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "nginx-helloworld.fullname" . }}
  labels:
    {{- include "nginx-helloworld.labels" . | nindent 4 }}
spec:
  {{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}              # 设定副本数量
```

```
  {{- end }}
  selector:
    matchLabels:
      {{- include "nginx-helloworld.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      ...
    spec:
      ...
      containers:
        - name: {{ .Chart.Name }}                    # 设定容器的名称
          ...
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default
.Chart.AppVersion }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ...

# 根据deployment的内容，我们来修改values.yaml 文件中的相关数据
# vim nginx-helloworld/values.yaml
...
image:
  repository: 10.0.0.19:80/mykubernetes/nginx
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: "1.21.3
```

改造定制文件支持我们的需求

```
改造deployment.yaml文件，让其支持定制首页功能
# vim nginx-helloworld/templates/deployment.yaml
...
      containers:
        - name: {{ .Chart.Name }}
          ...
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default
.Chart.AppVersion }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          # 添加以下内容
          env:
          - name: USERNAME
            value: {{ .Values.Username }}
          lifecycle:
            postStart:
              exec:
                command: ["/bin/sh","-c","echo $USERNAME >
/usr/share/nginx/html/index.html"]

改造values.yaml文件，添加定制的内容
# vim nginx-helloworld/values.yaml
Username: "hello world"    # 添加一个定制的环境变量名称
replicaCount: 1
...
```

定制打包文件

```
回到helm的项目根目录下，进行项目的检查效果
```

```
root@master1:/data/server/helm_pro# helm lint --strict nginx-helloworld/
==> Linting nginx-helloworld/
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, 0 chart(s) failed
```

结果显示：

我们的chart包没有任何问题。

进行项目的打包功能

```
# helm package nginx-helloworld/
Successfully packaged chart and saved it to: /data/server/helm_pro/nginx-helloworld-0.1.0.tgz
# ls
nginx-helloworld  nginx-helloworld-0.1.0.tgz
```

结果显示：

在helm项目目录下，chart的根目录下，创建了一个同名的压缩包文件，压缩包的版本号来自于Chart.yaml文件中的属性

本地安装定制的镜像文件

注意：

必须在k8s的集群中来部署相关的应用，如果当前主机没有k8s的话，可以通过环境变量来进行指定k8s主机的地址位置。

```
export KUBERNETES_MASTER=http://10.0.0.200:6443
```

安装helm压缩包

```
root@master1:/data/server/helm_pro# helm install nginx-helloworld nginx-helloworld-0.1.0.tgz
NAME: nginx-helloworld
LAST DEPLOYED: Thu Oct 28 11:35:16 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=nginx-helloworld,app.kubernetes.io/instance=nginx-helloworld" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
```

注意：

这里的注释是由Chart中的 templates/NOTES.txt，提供的。

如果希望提示自定义的内容的话，可以自己修改NOTES.txt 文件的显示格式

测试效果

```
kubectl get pod -o wide
kubectl get svc
curl 10.101.47.91
```

```
root@master1:/data/server/helm_pro# kubectl get pod -o wide
NAME                              READY   STATUS    RESTARTS   AGE   IP            NODE    NOMINATED NODE   READINESS GATES
nginx-helloworld-cb7dff758-hbj44   1/1     Running   0          59s   10.244.4.20   node1   <none>           <none>
root@master1:/data/server/helm_pro# kubectl get svc
NAME              TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes        ClusterIP   10.96.0.1      <none>        443/TCP   9d
nginx-helloworld   ClusterIP   10.101.47.91   <none>        80/TCP    78s
root@master1:/data/server/helm_pro# curl 10.101.47.91
hello world
```

进阶使用

values.yaml只是Helm install参数的默认设置，我们可以通过命令参数，向里面传递一些属性信息，效果如下

```
helm install nginx-1 nginx-helloworld-0.1.0.tgz --set Username="Nihao Nginx"
```

查看效果

```
helm list
```

```
root@master1:/data/server/helm_pro# helm install nginx-1 nginx-helloworld-0.1.0.tgz --set Username="Nihao Nginx"
NAME: nginx-1
LAST DEPLOYED: Thu Oct 28 11:39:30 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=nginx-helloworld,app.kubernetes.io/instance=nginx-1" -o jsonp
ath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
root@master1:/data/server/helm_pro# helm list
NAME                    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                   APP VERSI
ON
nginx-1                 default         1               2021-10-28 11:39:30.173717131 +0800 CST deployed        nginx-helloworld-0.1.0  1.16.0

nginx-helloworld        default         1               2021-10-28 11:35:16.963207856 +0800 CST deployed        nginx-helloworld-0.1.0  1.16.0

root@master1:/data/server/helm_pro# kubectl get pod
NAME                                     READY   STATUS    RESTARTS   AGE
nginx-1-nginx-helloworld-67858b7794-9vdrk   1/1     Running   0          64s
nginx-helloworld-cb7dff758-hbj44            1/1     Running   0          5m15s
root@master1:/data/server/helm_pro# kubectl get svc
NAME                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes                ClusterIP   10.96.0.1       <none>        443/TCP   9d
nginx-1-nginx-helloworld  ClusterIP   10.109.204.18   <none>        80/TCP    68s
nginx-helloworld          ClusterIP   10.101.47.91    <none>        80/TCP    5m21s
root@master1:/data/server/helm_pro# curl 10.109.204.18
Nihao Nginx
```

监控部署

学习目标

这一节，我们从 环境搭建、指标实践、小结 三个方面来学习。

环境搭建

环境准备

获取文件

```
# cd ~/mykubernetes/prometheus/helm
# helm pull prometheus-community/prometheus
# ls
prometheus-14.11.0.tgz
```

解压文件

```
# tar xf prometheus-14.11.0.tgz
# ls prometheus
Chart.lock  charts  Chart.yaml  README.md  templates  values.yaml
# mv prometheus-adapter/values.yaml ./prometheus-14.11.0-values.yaml
```

准备镜像

```
# grep 19:80 prometheus-14.11.0-values.yaml
    repository: 10.0.0.19:80/helm/alertmanager
      repository: 10.0.0.19:80/helm/configmap-reload
      repository: 10.0.0.19:80/helm/configmap-reload
    repository: 10.0.0.19:80/helm/node-exporter
    repository: 10.0.0.19:80/helm/prometheus
    repository: 10.0.0.19:80/helm/pushgateway
```

关联镜像 - 在所有的node节点上进行

```
docker pull 10.0.0.19:80/helm/kube-state-metrics:v2.2.0
```

```
docker tag 10.0.0.19:80/helm/kube-state-metrics:v2.2.0 k8s.gcr.io/kube-state-metrics/kube-state-metrics:v2.2.0
```

修改参数值文件

```
root@master1:/data/softs# grep -Env '#|^$' prometheus-14.11.0-values.yaml
...
30:alertmanager:
...
49:  image:
50:    repository: 10.0.0.19:80/helm/alertmanager
51:    tag: v0.22.2
...
105:  ingress:
108:    enabled: true
116:    annotations:
117:      kubernetes.io/ingress.class: nginx
122:    extraLabels: {}
127:    hosts:
128:      - alert.localprom.com
...
186:  persistentVolume:
190:    enabled: false
...
368:configmapReload:
369:  prometheus:
        ...
381:      repository: 10.0.0.19:80/helm/configmap-reload
382:      tag: v0.5.0
...
407:  alertmanager:
...
419:      repository: 10.0.0.19:80/helm/configmap-reload
420:      tag: v0.5.0
...
456:nodeExporter:
...
479:  image:
480:    repository: 10.0.0.19:80/helm/node-exporter
481:    tag: v1.1.2
...
611:server:
...
654:  image:
655:    repository: 10.0.0.19:80/helm/prometheus
656:    tag: v2.26.0
...
783:  ingress:
786:    enabled: true
794:    annotations:
795:      kubernetes.io/ingress.class: nginx
800:    extraLabels: {}
805:    hosts:
806:      - prom.localprom.com
...
870:  persistentVolume:
874:    enabled: false
```

```
...
1098:pushgateway:
...
1115:    repository: 10.0.0.19:80/helm/pushgateway
1116:    tag: v1.3.1
...
1281:  persistentVolume:
1284:    enabled: false
1290:    ...
```

创建专用域名

```
kubectl create ns monitoring
```

创建ingress controller

```
kubectl apply -f ~/mykubernetes/ingress/deploy.yaml
```

安装应用

```
helm install my-prom prometheus-14.11.0.tgz -f prometheus-14.11.0-values.yaml --namespace monitoring
```

```
root@master1:/data/softs# helm install my-prom prometheus-14.11.0.tgz -f prometheus/values.yaml --namespace monitoring
NAME: my-prom
LAST DEPLOYED: Sun Oct 31 13:33:52 2021
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Prometheus server can be accessed via port 80 on the following DNS name from within your cluster:
my-prom-prometheus-server.monitoring.svc.cluster.local

From outside the cluster, the server URL(s) are:
http://prom.localprom.com
#################################################################################
######   WARNING: Persistence is disabled!!! You will lose your data when   #####
######            the Server pod is terminated.                             #####
#################################################################################


The Prometheus alertmanager can be accessed via port 80 on the following DNS name from within your cluster:
my-prom-prometheus-alertmanager.monitoring.svc.cluster.local

From outside the cluster, the alertmanager URL(s) are:
http://alert.localprom.com
#################################################################################
######   WARNING: Persistence is disabled!!! You will lose your data when   #####
######            the AlertManager pod is terminated.                       #####
#################################################################################
#################################################################################
######   WARNING: Pod Security Policy has been moved to a global property.  #####
######            use .Values.podSecurityPolicy.enabled with pod-based      #####
######            annotations                                               #####
######            (e.g. .Values.nodeExporter.podSecurityPolicy.annotations) #####
#################################################################################


The Prometheus PushGateway can be accessed via port 9091 on the following DNS name from within your cluster:
my-prom-prometheus-pushgateway.monitoring.svc.cluster.local


Get the PushGateway URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace monitoring -l "app=prometheus,component=pushgateway" -o jsonpath="{.it
ems[0].metadata.name}")
  kubectl --namespace monitoring port-forward $POD_NAME 9091

For more information on running Prometheus, visit:
https://prometheus.io/
```

确认效果

```
kubectl get pod -n monitoring
```

```
root@master1:/data/softs# kubectl get pod -n monitoring
NAME                                             READY   STATUS    RESTARTS   AGE
my-prom-kube-state-metrics-94d97bd87-d48th       1/1     Running   0          15m
my-prom-prometheus-alertmanager-7ccbbf7b8c-kz9hx 2/2     Running   0          15m
my-prom-prometheus-node-exporter-ldv6x           1/1     Running   0          15m
my-prom-prometheus-node-exporter-wdvjf           1/1     Running   0          15m
my-prom-prometheus-pushgateway-fcb569bcd-v22ds   1/1     Running   0          15m
my-prom-prometheus-server-5f5f46fd59-ftj85       2/2     Running   0          15m
root@master1:/data/softs# kubectl get all -n monitoring
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/my-prom-kube-state-metrics-94d97bd87-d48th       1/1     Running   0          15m
pod/my-prom-prometheus-alertmanager-7ccbbf7b8c-kz9hx 2/2     Running   0          15m
pod/my-prom-prometheus-node-exporter-ldv6x           1/1     Running   0          15m
pod/my-prom-prometheus-node-exporter-wdvjf           1/1     Running   0          15m
pod/my-prom-prometheus-pushgateway-fcb569bcd-v22ds   1/1     Running   0          15m
pod/my-prom-prometheus-server-5f5f46fd59-ftj85       2/2     Running   0          15m

NAME                                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/my-prom-kube-state-metrics            ClusterIP   10.105.154.15   <none>        8080/TCP   15m
service/my-prom-prometheus-alertmanager       ClusterIP   10.104.239.99   <none>        80/TCP     15m
service/my-prom-prometheus-node-exporter      ClusterIP   None            <none>        9100/TCP   15m
service/my-prom-prometheus-pushgateway        ClusterIP   10.102.106.69   <none>        9091/TCP   15m
service/my-prom-prometheus-server             ClusterIP   10.96.179.168   <none>        80/TCP     15m
```

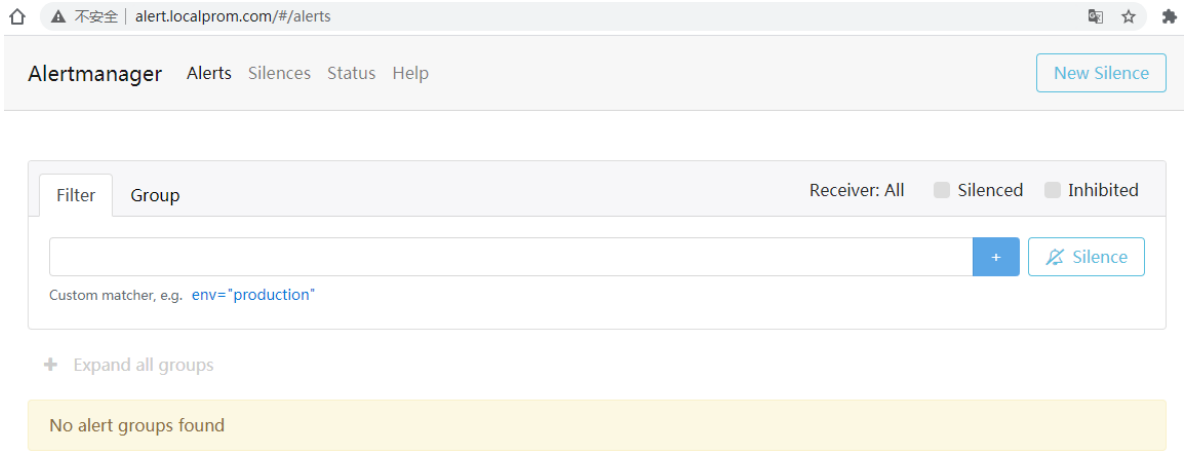
```
NAME                                         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
AGE
daemonset.apps/my-prom-prometheus-node-exporter   2         2         2       2            2           <none>
15m

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-prom-kube-state-metrics    1/1     1            1           15m
deployment.apps/my-prom-prometheus-alertmanager   1/1     1            1           15m
deployment.apps/my-prom-prometheus-pushgateway    1/1     1            1           15m
deployment.apps/my-prom-prometheus-server     1/1     1            1           15m

NAME                                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/my-prom-kube-state-metrics-94d97bd87    1         1         1       15m
replicaset.apps/my-prom-prometheus-alertmanager-7ccbbf7b8c   1         1         1       15m
replicaset.apps/my-prom-prometheus-pushgateway-fcb569bcd    1         1         1       15m
replicaset.apps/my-prom-prometheus-server-5f5f46fd59        1         1         1       15m
```

浏览器访问prometheus效果

浏览器访问alertmanager的效果



指标实践

监控指标

在k8s的系统上包含了各种各样的指标数据，早期的k8s系统，为kubelet集成了一个CAdvsior工具可以获取kubelet所在节点上的相关指标，包括容器指标。但是CAdvsior的缺陷在于，我们仅能够获取，指定节点上的指标信息，而无法获取集群管理的统一指标。

比如，在k8s集群中，提供了一些监控用的命令，比如top，通过它可以汇总节点上的相关统计信息。由于没有默认情况下，k8s没有提供专用的metrics接口，所以这个命令无法正常使用。

```
# kubectl top node
error: Metrics API not available
```

虽然k8s已经内嵌了CAdvsior，但是没有集群级别的资源对象能够汇总这些所有的指标数据，并通过相关资源对象的api接口暴露出去。

```
kubectl api-resources | grep metrics
```

指标类型

早期的k8s提供了一个专用的metrics的指标服务器heapster，用于采集所有的监控数据，只不过这个软件因为某些原因被弃用了。

其中一部分原因就是，k8s的指标分化成了两种不同的类别：

核心资源指标

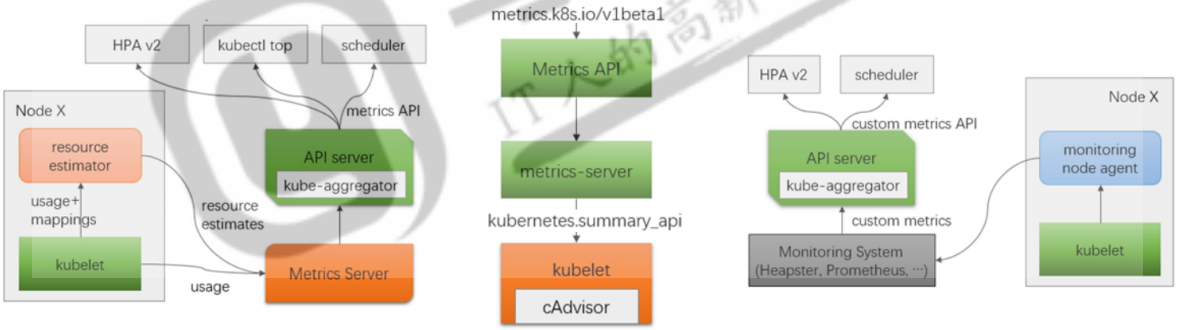
- k8s系统内部默认就应用的指标
- 它通过metrics-server服务来进行采集所有kubelet的数据，进而通过/metrics接口输出数据

  /api/metrics.k8s.io/v1bata1

  该服务默认情况下，是仅仅获取指标数据，然后保存到内存中。

自定义指标

- 用户根据情况自定义的指标
- 通过专用的监控平台软件来实现，或者通过对项目接口改造，暴露相关数据



注意：

默认情况下，k8s的指标格式与prometheus的指标格式不兼容。

如果我们需要通过prometheus来监控k8s的话，需要添加适配器

官方地址：https://github.com/kubernetes-sigs/metrics-server
最新版本：3.6.0 | 20211018

数据采集汇总

方式	解析
监控代理程序	如node_exporter，收集标准的主机指标数据，包括平均负载、CPU、Memory、Disk、Network及诸多其他维度的数据
kubelet	收集容器指标数据，它们也是Kubernetes“核心指标”，每个容器的相关指标数据主要有CPU利用率（user和system）及限额、文件系统读/写/限额、内存利用率及限额、网络报文发送/接收/丢弃速率等。
API Server	收集API Server的性能指标数据，包括控制工作队列的性能、请求速率与延迟时长、etcd缓存工作队列及缓存性能、普通进程状态（文件描述符、内存、CPU等）、Golang状态（GC、内存和线程等）
etcd	收集etcd存储集群的相关指标数据，包括领导节点及领域变动速率、提交/应用/挂起/错误的提案次数、磁盘写入性能、网络与gRPC计数器等
kube-state-metrics	该组件用于根据Kubernetes API Server中的资源派生出多种资源指标，它们主要是资源类型相关的计数器和元数据信息，包括指定类型的对象总数、资源限额、容器状态（ready/restart/running/terminated/waiting）以及Pod资源的标签系列等

软件安装

```
获取文件
wget https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

修改文件 - 仅需要修改镜像即可
# grep image: components.yaml
        image: 10.0.0.19:80/helm/metrics-server:v0.5.1

   镜像来源： registry.aliyuncs.com/google_containers/metrics-server:v0.5.1

安装软件
kubectl apply -f components.yaml
注意：
    默认情况下，该服务会因为无法与k8s进行认证，导致服务无法正常运行。
    所以需要在pod启动的时候，添加一条属性
      containers:
      - args:
        - --cert-dir=/tmp
        - --secure-port=443
        - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
        - --kubelet-use-node-status-port
        - --metric-resolution=15s
        - --kubelet-insecure-tls      # 添加这一条配置属性

查看效果
```

```
kubectl get pod -n kube-system | egrep 'NAME|metr'
```

```
root@master1:~/mykubernetes/prometheus/metrics# kubectl get pod -n kube-system | egrep 'NAME|metr'
NAME                                READY   STATUS    RESTARTS        AGE
metrics-server-8cd965c6-gd9m8       1/1     Running   0               51s
root@master1:~/mykubernetes/prometheus/metrics# kubectl get all -n kube-system | egrep 'NAME|metr'
NAME                                READY   STATUS    RESTARTS        AGE
pod/metrics-server-8cd965c6-gd9m8   1/1     Running   0               56s
NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                  AGE
service/metrics-server   ClusterIP   10.111.52.203   <none>        443/TCP                  57s
NAME                                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/metrics-server      1/1     1            1           57s
NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/metrics-server-8cd965c6        1         1         1       57s
```

资源创建完毕后，会生成如下几个资源

```
# kubectl api-resources | egrep 'metr|NAME'
NAME       SHORTNAMES   APIVERSION                NAMESPACED   KIND
nodes                   metrics.k8s.io/v1beta1    false        NodeMetrics
pods                    metrics.k8s.io/v1beta1    true         PodMetrics
```

确认效果

```
kubectl top node
kubectl top pod -n kube-system
```

结果显示：

这个时候，我们就可以通过 top命令来进行相关信息的获取

```
root@master1:~/mykubernetes/prometheus/metrics# kubectl top node
NAME      CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
master1   375m         18%    1696Mi          59%
node1     152m         15%    1531Mi          53%
node2     136m         13%    1406Mi          49%
root@master1:~/mykubernetes/prometheus/metrics# kubectl top pod -n kube-system
NAME                                       CPU(cores)   MEMORY(bytes)
calico-kube-controllers-6d97d7c96f-m9knk   5m           23Mi
calico-node-dxvhn                          41m          122Mi
calico-node-lbgbm                          33m          121Mi
calico-node-s7tlh                          50m          116Mi
coredns-fd5877b89-8dr2b                    2m           17Mi
coredns-fd5877b89-kpmfb                    2m           19Mi
etcd-master1                               24m          77Mi
kube-apiserver-master1                     88m          421Mi
kube-controller-manager-master1            24m          64Mi
kube-proxy-2tff8                           1m           17Mi
kube-proxy-7hzrt                           1m           17Mi
kube-proxy-bg294                           1m           18Mi
kube-scheduler-master1                     4m           25Mi
metrics-server-8cd965c6-gd9m8              4m           16Mi
root@master1:~/mykubernetes/prometheus/metrics#
```

- 简单实践

实践1 - 手工自动调整

定制基础的应用对象 01-hpa-deployment-core.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-web
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-web
  template:
    metadata:
      labels:
        app: flask-web
    spec:
      containers:
      - name: flask-web
```

```
        image: 10.0.0.19:80/mykubernetes/pod_test:v0.1
        ports:
        - containerPort: 80
          name: http
        resources:
          requests:
            memory: "256Mi"
            cpu: "50m"
          limits:
            memory: "256Mi"
            cpu: "50m"
---
apiVersion: v1
kind: Service
metadata:
  name: flask-service
spec:
  selector:
    app: flask-web
  ports:
  - name: http
    port: 80
    targetPort: 80
```

检查效果

```
# kubectl get pod,deployment,svc
```

```
root@master1:~/mykubernetes/prometheus/hpa# kubectl get pod,deployment,svc
NAME                              READY   STATUS    RESTARTS   AGE
pod/flask-web-6679fd954c-cgnx2    1/1     Running   0          15s
pod/flask-web-6679fd954c-xd5tb    1/1     Running   0          10s

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/flask-web    2/2     2            2           117s

NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/flask-service    ClusterIP   10.98.208.32    <none>        80/TCP    117s
service/kubernetes       ClusterIP   10.96.0.1       <none>        443/TCP   10d
```

在k8s中有一个自动的扩缩容命令 autoscale

```
kubectl autoscale deployment flask-web --min=2 --max=5 --cpu-percent=40
```

属性解析：

```
        --min=2              最少的pod数量
        --max=5              最多的pod数量
        --cpu-percent=40     调整的标准，以cpu为例
```

查看效果

```
#  kubectl get hpa -w
NAME        REFERENCE               TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
flask-web   Deployment/flask-web    2%/40%     2         5         2          57s
```

注意：

由于这里查询的是核心指标，所以它是从metrics-server中获取到的

尝试对pod进行压测

```
# kubectl run pod-test --image=10.0.0.19:80/mykubernetes/admin-box:v0.1 --rm -it
--command -- /bin/bash
root@pod-test /# while true; do curl http://flask-service.default.svc; sleep
0.001; done
...
```

查看hpa的动态调整效果

```
root@master1:~/mykubernetes/prometheus/hpa# kubectl get hpa -w
NAME        REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
flask-web   Deployment/flask-web   8%/40%    2         5         2          5m40s
flask-web   Deployment/flask-web   2%/40%    2         5         2          5m46s
flask-web   Deployment/flask-web   19%/40%   2         5         2          6m17s
flask-web   Deployment/flask-web   70%/40%   2         5         2          6m32s
flask-web   Deployment/flask-web   69%/40%   2         5         4          6m47s
flask-web   Deployment/flask-web   66%/40%   2         5         4          7m2s
flask-web   Deployment/flask-web   65%/40%   2         5         4          7m17s
flask-web   Deployment/flask-web   53%/40%   2         5         5          7m32s
flask-web   Deployment/flask-web   55%/40%   2         5         5          7m47s
flask-web   Deployment/flask-web   30%/40%   2         5         5          8m2s
flask-web   Deployment/flask-web   2%/40%    2         5         5          8m17s
flask-web   Deployment/flask-web   2%/40%    2         5         5          12m
flask-web   Deployment/flask-web   2%/40%    2         5         3          13m
flask-web   Deployment/flask-web   2%/40%    2         5         2          13m
```

结果显示：

可以实现资源的动态调整了

注意：

扩缩容的时候，为了防止pod抖动，一般持续的时间稍微长一点，我们这里离持续了5分钟才缩容了

实践2-核心指标调整

定制flask-web的动态调整配置 02-hpa-deployment-core-autoscale.yaml

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: flask-web
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: flask-web
  minReplicas: 2
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 30
  - type: Resource
    resource:
      name: memory
      target:
        type: AverageValue
        averageValue: 30Mi
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 120
```

应用配置文件

```
kubectl apply -f 02-hpa-deployment-core-autoscale.yaml
```

检查效果

```
# kubectl get hpa

在测试终端尝试对hpa进行压测
while true; do curl http://flask-service.default.svc; sleep 0.001; done
```

```
root@master1:~/mykubernetes/prometheus/hpa# kubectl get hpa -w
NAME        REFERENCE              TARGETS                MINPODS   MAXPODS   REPLICAS   AGE
flask-web   Deployment/flask-web   17408k/30Mi, 2%/30%    2         5         2          55s
flask-web   Deployment/flask-web   17659904/30Mi, 48%/30%   2         5         2          75s
flask-web   Deployment/flask-web   17684480/30Mi, 74%/30%   2         5         4          90s
flask-web   Deployment/flask-web   18016256/30Mi, 58%/30%   2         5         5          105s
flask-web   Deployment/flask-web   17448960/30Mi, 40%/30%   2         5         5          2m
flask-web   Deployment/flask-web   17474355200m/30Mi, 45%/30%   2         5         5          2m15s
flask-web   Deployment/flask-web   17480089600m/30Mi, 42%/30%   2         5         5          2m31s
flask-web   Deployment/flask-web   17495654400m/30Mi, 43%/30%   2         5         5          2m46s
flask-web   Deployment/flask-web   17498112/30Mi, 37%/30%       2         5         5          3m1s
flask-web   Deployment/flask-web   17498112/30Mi, 23%/30%       2         5         5          3m16s
flask-web   Deployment/flask-web   17498112/30Mi, 2%/30%        2         5         5          3m31s
flask-web   Deployment/flask-web   17498112/30Mi, 2%/30%        2         5         5          5m1s
flask-web   Deployment/flask-web   17408k/30Mi, 2%/30%          2         5         4          5m16s
flask-web   Deployment/flask-web   17679701333m/30Mi, 2%/30%    2         5         3          5m31s
flask-web   Deployment/flask-web   17679701333m/30Mi, 2%/30%    2         5         3          7m16s
flask-web   Deployment/flask-web   18112512/30Mi, 2%/30%        2         5         2          7m32s
```

小结

监控进阶

学习目标

这一节，我们 从 适配器、简单实践、小结 三个方面来学习。

适配器

简介

我们之前通过对指标数据的了解，k8s中主要包括两类指标，核心指标和自定义指标，幸运的是，这两类指标默认情况下都与prometheus不兼容，所以我们需要有一种机制能够，让k8s和prometheus实现兼容的效果。从而实现，prometheus抓取的指标数据，能够暴露给k8s上，并且为k8s使用。

而这就是 资源指标适配器。目前常用的适配器主要有两种：

k8s-prometheus-adapter 和 k8s-metrics-adapter

环境部署

获取代码

```
cd ~/mykubernetes/prometheus/helm
helm pull prometheus-community/prometheus-adapter

解压文件后修改values文件
# tar xf prometheus-adapter-3.0.0.tgz
# mv prometheus-adapter/values.yaml ./prometheus-adapter-3.0.0-values.yaml

修改配置文件
# grep -Env '#|^$' prometheus-adapter-3.0.0-values.yaml
2:affinity: {}
4:image:
5:  repository: 10.0.0.19:80/helm/prometheus-adapter
6:  tag: v0.9.1
...
29:prometheus:
31:  url: http://my-prom-prometheus-server.monitoring.svc   # 修改为prometheus的地址
32:  port: 80
注意：
```

```
    prometheus的地址
    # kubectl get svc -n monitoring | egrep 'NAME|server'
    NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
...
    my-prom-prometheus-server  ClusterIP   10.96.179.168   <none>        80/TCP
...
```

安装环境

```
helm install prometheus-adapter prometheus-adapter-3.0.0.tgz -f prometheus-adapter-3.0.0-values.yaml --namespace kube-system
```

```
root@master1:~/mykubernetes/prometheus/helm# helm install prometheus-adapter prometheus-adapter-3.0.0.tgz -f prometheus-adapter-3.0.0-values.yaml --namespace kube-system
NAME: prometheus-adapter
LAST DEPLOYED: Sun Oct 31 17:06:47 2021
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
prometheus-adapter has been deployed.
In a few minutes you should be able to list metrics using the following command(s):

  kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1
```

结果显示：

我们可以通过 kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1 来获取自定义资源

抓取自定义指标

```
# kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1 | jq
{"kind":"APIResourceList","apiVersion":"v1","groupVersion":"custom.metrics.k8s.io/v1beta1","resources":[]}
```

注意：

这里会有几秒钟的时间，从prometheus中获取相关指标数据，然后给k8s来使用

前提是prometheus的连接地址是正确的

检查效果

```
helm list -n kube-system
kubectl get all -n kube-system | egrep 'NAME|adap'
```

```
root@master1:~/mykubernetes/prometheus/helm# helm list -n kube-system
NAME                    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART
                        APP VERSION
prometheus-adapter      kube-system     1               2021-10-31 17:06:47.976374558 +0800 CST deployed        prometheus-adapter-3.0.0
                        v0.9.1
root@master1:~/mykubernetes/prometheus/helm# kubectl get all -n kube-system | egrep 'NAME|adap'
NAME                                       READY   STATUS    RESTARTS         AGE
pod/prometheus-adapter-656ddbdb95-flngb    1/1     Running   0                2m1s
NAME                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                  AGE
service/prometheus-adapter   ClusterIP   10.100.99.199   <none>        443/TCP                  2m1s
NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/prometheus-adapter       1/1     1            1           2m1s
NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/prometheus-adapter-656ddbdb95    1         1         1       2m1s
```

简单实践

实践1-自定义资源实践

创建资源清单文件 03-hpa-deployment-define.yaml

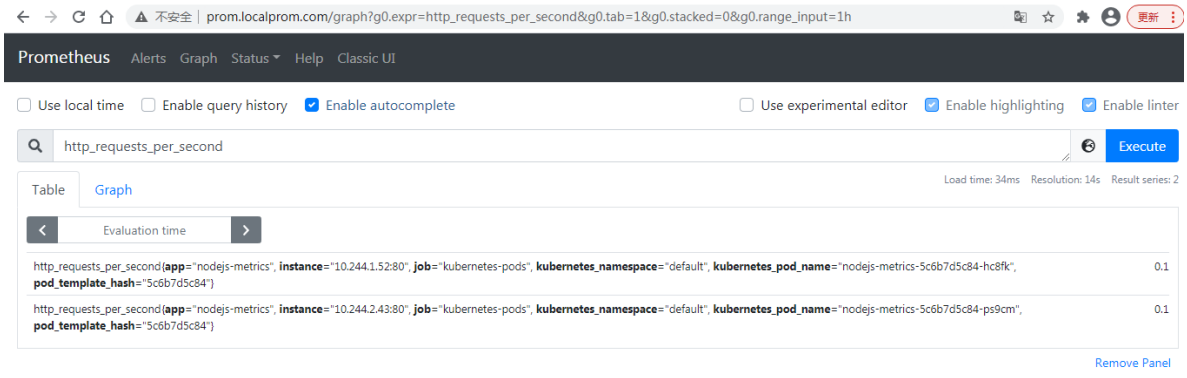
```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: nodejs-metrics
spec:
  replicas: 2
  selector:
    matchLabels:
```

```
      app: nodejs-metrics
  template:
    metadata:
      labels:
        app: nodejs-metrics
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "80"
        prometheus.io/path: "/metrics"
        # 这里的annotations告诉prometheus来metrics里抓数据
    spec:
      containers:
      - image: 10.0.0.19:80/mykubernetes/nodejs_pod:v0.1
        name: nodejs-metrics
        ports:
        - name: web
          containerPort: 80
        resources:
          requests:
            memory: "256Mi"
            cpu: "500m"
          limits:
            memory: "256Mi"
            cpu: "500m"
---
apiVersion: v1
kind: Service
metadata:
  name: nodejs-service
spec:
  type: NodePort
  ports:
  - name: web
    port: 80
    targetPort: 80
  selector:
    app: nodejs-metrics
```

应用资源配置文件

```
kubectl apply -f 03-hpa-deployment-define.yaml
```

确认自定义指标的采集



结果显示：
    可以看到自定义的指标内容

小结

综合实践

学习目标

这一节，我们从 环境准备、简单实践、小结 三个方面来学习。

简单实践

需求

    虽然metrics-server提供了很多的k8s指标，我们也可以通过 adapter方式将相关的指标返回到k8s中进行使用，但是仍然有很多场景需要自己定义一些指标。
    比如说，我们可以根据用户访问的流量来对某些资源对象进行自动调整，而这就涉及到了指标的定制。根据我们对prometheus的指标定制流程，这里我们可以对adapter的指标机制进行丰富。

```
adapter默认的配置信息在cm配置文件里面
# kubectl get cm -n kube-system | egrep 'NAME|ada'
NAME                                 DATA   AGE
prometheus-adapter                   1      48m

root@master1:~/mykubernetes/prometheus/helm# kubectl get cm prometheus-adapter -
n kube-system -o yaml
apiVersion: v1
data:
  config.yaml: |
    rules:
    - seriesQuery:
'{__name__=~"^container_.*",container!="POD",namespace!="",pod!=""}'
      seriesFilters: []
      resources:
        overrides:
          namespace:
            resource: namespace
          pod:
            resource: pod
      name:
        matches: ^container_(.*)_seconds_total$
        as: ""
      metricsQuery: sum(rate(<<.Series>>{<<.LabelMatchers>>,container!="POD"}
[5m]))
        by (<<.GroupBy>>)
    ...
```

```
导出指标配置文件
kubectl get cm prometheus-adapter -n kube-system -o yaml > prometheus-adapter-
3.0.0-cm.yaml

定制指标配置
# vim prometheus-adapter-3.0.0-cm.yaml
```

```
...
    - seriesQuery:
'http_requests_total{kubernetes_namespace!="",kubernetes_pod_name!=""}'
      resources:
        overrides:
          kubernetes_namespace: {resource: "namespace"}
          kubernetes_pod_name: {resource: "pod"}
      name:
        matches: "^(.*)_total"
        as: "${1}_per_second"
      metricsQuery: 'rate(<<.Series>>{<<.LabelMatchers>>}[2m])'
    existing:
    external: []
kind: ConfigMap
metadata:
  annotations:
    meta.helm.sh/release-name: prometheus-adapter
    meta.helm.sh/release-namespace: kube-system
  labels:
    app.kubernetes.io/component: metrics
    app.kubernetes.io/instance: prometheus-adapter
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: prometheus-adapter
    app.kubernetes.io/part-of: prometheus-adapter
    app.kubernetes.io/version: v0.9.1
    helm.sh/chart: prometheus-adapter-3.0.0
  name: prometheus-adapter
  namespace: kube-system
配置解析:
    1 添加一个定制的指标内容
    2 删除cm内部无效的属性信息

重载cm配置文件
kubectl apply -f prometheus-adapter-3.0.0-cm.yaml
```

简单实践

实践1- 自定义资源调整

```
定制资源清单配置文件 04-hpa-deployment-define-autoscale.yaml
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2beta2
metadata:
  name: nodejs-metrics-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nodejs-metrics
  minReplicas: 2
  maxReplicas: 6
  metrics:
  - type: Pods
    pods:
      metric:
        name: http_requests_per_second
      target:
```

```
        type: AverageValue
        averageValue: 5
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 120

应用资源配置文件
kubectl apply -f 04-hpa-deployment-define-autoscale.yaml

对该资源对象进行压测
while true; do curl http://nodejs-service.default.svc; sleep 0.001; done
```

小结

helm仓库

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

简介

```
我们知道，helm chart 能够很好的封装和管理我们的 kubernetes 应用，可以实现中间件、数据库、公共组件等快速发布。虽然互联网上有很多helm仓库，但是我们知道，互联网中的helm仓库的文件有两个方面的问题；
    1 内容仅仅是通用的，不满足定制的需求
    2 文件往往是最新的，不满足环境的需求

    因此，当我们的项目内部应用多了以后，彼此产生互相的依赖，之前直接使用yaml文件的管理方式已经不再适应新的环境需求，因此我们有必要构建自己的chart仓库，来存放自己的文件。
```

```
为什么我们要做一个一个私有的helm仓库呢
    日常工作中helm可以提高部署效率
    工作中的部署环境很多，需要定制的模板很多
    旧有版本需要定制化管理
注意：
    helm2 原本是带了本地仓库功能，helm3 移除了这部分，将他变成了一个纯粹的应用管理工具。从这个层面上来说，helm的chart本身对仓库的要求并不是太高。
    也就是说，仓库本身只需要提供yaml文件和tar文件的web服务即可。
```

实现方式

```
常见的实现方式很多：
    兼容helm仓库功能：harbor，JFrog Artifactory
    版本控制功能：github 或 gitlab
    开源工具：chartmuseum
```

chartmuseum简介

Chartmuseum 除了给我们提供一个类似于web服务器的功能之外，还提供了其他有用的功能，便于日常我们私有仓库的管理。

根据chart文件自动生成index.yaml
helm push的插件，可以在helm命令之上实现将chart文件推送到chartmuseum上
相应的tls配置，Basic认证，JWT认证
提供了Restful的api和可以使用的cli命令行工具
提供了各种后端存储的支持
提供了Prometheus的集成，对外提供自己的监控信息。
没有用户的概念，但是基于目录实现了一定程度上的多租户的需求。

官方源码：https://github.com/helm/chartmuseum.git
最新版本：v0.13.1

常见接口

```
Helm Chart 仓库
GET /index.yaml                          - retrieved when you run helm repo add
chartmuseum http://xxx:8080/
GET /charts/mychart-0.1.0.tgz            - retrieved when you run helm install
chartmuseum/mychart
GET /charts/mychart-0.1.0.tgz.prov       - retrieved when you run helm install
with the --verify flag

Chart 管理
POST /api/charts                         - upload a new chart version
POST /api/prov                           - upload a new provenance file
DELETE /api/charts/<name>/<version>      - delete a chart version (and
corresponding provenance file)
GET /api/charts                          - list all charts
GET /api/charts/<name>                   - list all versions of a chart
GET /api/charts/<name>/<version>         - describe a chart version
HEAD /api/charts/<name>                  - check if chart exists (any versions)
HEAD /api/charts/<name>/<version>        - check if chart version exists

Server 信息
GET /                                    - HTML welcome page
GET /info                                - returns current ChartMuseum version
GET /health                              - returns 200 OK
```

简单实践

环境部署

```
获取镜像
docker pull chartmuseum/chartmuseum:latest
docker tag chartmuseum/chartmuseum:latest 10.0.0.19:80/helm/chartmuseum:v0.13.1
docker push 10.0.0.19:80/helm/chartmuseum:v0.13.1

准备数据目录
mkdir /data/server/charts

创建仓库
```

```
docker run -d -p 8080:8080 -e DEBUG=1 -e STORAGE=local -e
STORAGE_LOCAL_ROOTDIR=/charts -v /data/server/charts:/charts
10.0.0.19:80/helm/chartmuseum:v0.13.1
```

测试仓库接口

```
apt install -y jq
curl localhost:8080/api/charts | jq
```

注意：

只要我们把charts文件存放到指定的目录下就可以直接进行使用了

准备数据文件

```
mkdir /data/server/charts
chmod 777 /data/server/charts
```

创建部署文件

```
# cat 01-chartmuseum-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: chartmuseum
  name: chartmuseum
  namespace: kube-system
spec:
  replicas: 1
  selector:
    matchLabels:
      app: chartmuseum
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: chartmuseum
    spec:
      containers:
      - image: 10.0.0.19:80/helm/chartmuseum:v0.13.1
        name: chartmuseum
        ports:
        - containerPort: 8080
          protocol: TCP
        env:
        - name: DEBUG
          value: "1"
        - name: STORAGE
          value: local
        - name: STORAGE_LOCAL_ROOTDIR
          value: /charts
        resources:
          limits:
            cpu: 500m
            memory: 256Mi
          requests:
```

```
            cpu: 100m
            memory: 64Mi
        volumeMounts:
        - mountPath: /charts
          name: charts-volume
      nodeSelector:
        kubernetes.io/hostname: node1
      volumes:
      - name: charts-volume
        hostPath:
          path: /data/server/charts
          type: DirectoryOrCreate
      restartPolicy: Always

创建访问文件
# cat 02-chartmuseum-service-yaml
apiVersion: v1
kind: Service
metadata:
  name: chartmuseum
  namespace: kube-system
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
    nodePort: 30080
  type: NodePort
  selector:
    app: chartmuseum

创建仓库环境
kubectl apply -f 01-chartmuseum-deployment.yaml
kubectl apply -f 02-chartmuseum-service-yaml

确认效果
kubectl get pod -n kube-system | grep chart
kubectl get svc -n kube-system | grep chart
curl http://10.109.189.53:8080
curl http://10.0.0.12:30080
```

仓库实践

```
在当前的helm环境中，获取一些测试chart工具
helm repo add az-stable http://mirror.azure.cn/kubernetes/charts/

把 charts 文件直接下载到 chartmuseum 指定的本地目录
cd /data/server/charts
helm pull az-stable/mysql
helm pull az-stable/tomcat

测试效果
curl localhost:8080/api/charts -s | jq
```

仓库准备

```
如果需要 helm push的话，需要安装 helm push 插件
helm plugin install https://github.com/chartmuseum/helm-push.git

添加本地仓库
helm repo add localrepo http://10.0.0.12:30080
helm repo update

查看效果
# helm repo list
NAME                    URL
...
localrepo               http://10.0.0.12:8080

提交文件 - 这种方式不受限制
curl --data-binary "@nginx-helloworld-0.1.0.tgz"
http://10.0.0.12:8080/api/charts

确认效果
# helm search repo nginx-helloworld
NAME                          CHART VERSION   APP VERSION DESCRIPTION
localrepo/nginx-helloworld  0.1.0           1.16.0      A Helm chart for
Kubernetes

安装效果
helm install mynginx localrepo/nginx-helloworld
```

```
注意：如果helm-push安装失败
wget https://github.com/chartmuseum/helm-push/releases/download/v0.10.1/helm-
push_0.10.1_linux_amd64.tar.gz
tar xf helm-push_0.10.1_linux_amd64.tar.gz
mkdir -p /root/.local/share/helm/plugins/helm-push.git/bin/
cp bin/helmpush /root/.local/share/helm/plugins/helm-push.git/bin/
```

通过ingress向外提供服务

```
获取资源文件
```

```
wget  https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.0.3/deploy/static/provider/baremetal/deploy.yaml

修改基础镜像
# grep image: deploy.yaml
          image: 10.0.0.19:80/google_containers/ingress-nginx-controller:v1.0.0
          image: 10.0.0.19:80/google_containers/ingress-nginx-kube-webhook-certgen:v1.0
          image: 10.0.0.19:80/google_containers/ingress-nginx-kube-webhook-certgen:v1.0

开放访问入口地址
# vim deploy.yaml
261 apiVersion: v1
262 kind: Service
...
273   namespace: ingress-nginx
274 spec:
275   type: NodePort
276   externalIPs: ['10.0.0.12']            # 增加一个外网访问的入口ip
277   ports:
278     - name: http
279       port: 80

应用资源配置文件
kubectl apply -f deploy.yaml

确认效果
kubectl get ns
kubectl get svc -n ingress-nginx

测试访问页面
# curl 10.0.0.12:30531
# curl -I -o /dev/null -s -w %{http_code}"\n" 10.0.0.12:30531
```

```
创建 chartmuseum ingress.yaml 文件
# vim 03-chartmuseum-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: chartmuseum
  namespace: kube-system
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host: sswang.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: chartmuseum
            port:
```

```
              number: 8080

应用资源文件
kubectl apply -f 03-chartmuseum-ingress.yaml

查看效果
kubectl get ingress -n kube-system
kubectl describe ingress ingress-test


添加主机解析记录
# vi /etc/hosts
10.244.0.10 sswang.example.com

测试访问效果
curl sswang.example.com
```

```
确认效果
helm repo remove localrepo
helm repo add myrepo http://sswang.example.com
helm repo update

搜索chart
helm search repo nginx-helloworld

安装应用
helm install myrepo/nginx-helloworld --generate-name
注意：
    如果仓库的配置是域名方式需要添加  --generate-name 参数
```

小结