云运维 – 中级 – 项目3 Kubernetes HPA实践

1 – 项目导学

1.1 - 项目介绍

我们在实际工作场景中，经常会遇到一些访问量不固定的业务，例如我们上线了秒杀场景类型的业务，在实际环境中，对该业务使用的资源其实是不好估算的，这个时候我们就需要用到K8s的HPA（Horizontal Pod Autoscaler）应用。在K8s集群中，HPA功能和公有云针对ECS开通的弹性伸缩（阿里云）以及Auto Scaling（AWS）类似，可以根据容器的CPU或者内存参数，在资源达到某个阈值，代表着当前Pod的节点压力较大，可以自动新启动一个Pod，来承受当前的业务访问量。HPA实际应用场景有很多，适用于突发类流量场景，或者新上线业务不确定资源，同时也可以作为节约服务器资源的一种方法。

在本项目中，您将从运维工程师的角度，对实际环境的Pod设置HPA策略，来让Pod达到自动扩容、缩容的目的。通过本项目的训练，在后续的生产环境中，您可以根据业务场景对Pod设置HPA策略，将HPA的掌握程度提升到中级水平。

1.2 - 教学目标

1. 掌握HPA的工作原理和实现机制
2. 掌握通过Kubernetes原生能力配置HPA策略
3. 通过实验验证HPA特性，并总结最佳实践

1.3 - 前置技能

1. 熟悉Kubernetes架构
2. 熟悉常见的服务自动扩缩容实现原理

1.4 - 技能提升目标

1. HPA工作原理 了解 --> 掌握
2. HPA策略 了解 --> 掌握

1.5 - 学习周期

总时长	任务1	任务2	任务3
2.5H	0.5H	1H	1H

1.6 - 配套资料

官方文档：https://kubernetes.io/zh/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

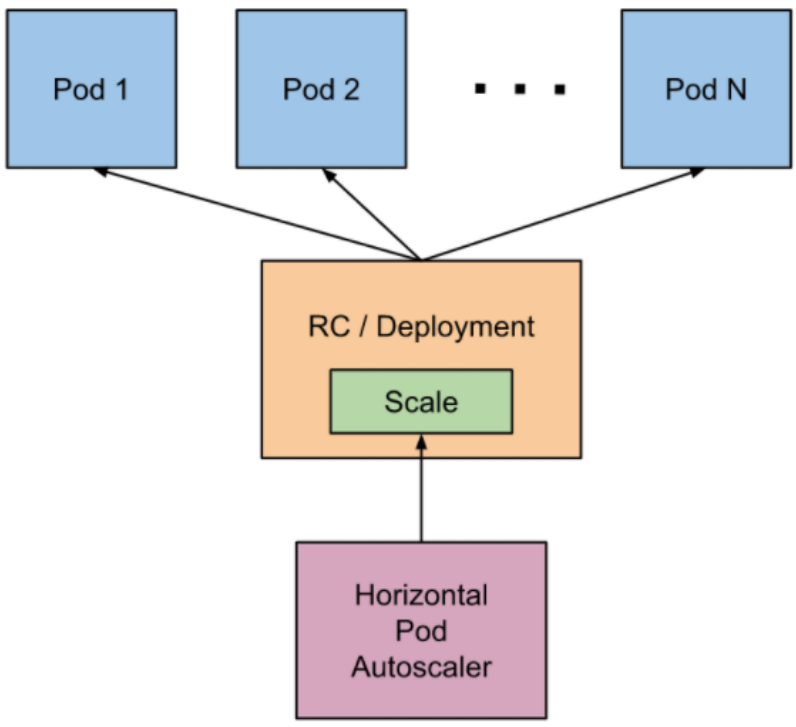
2 – 项目剖析

项目前置学习内容：熟悉虚机开发环境

本项目需要使用centosA 镜像，学员需要将虚拟机环境下载下来，并使用vmware软件打开

2.1 - Kubernetes HPA介绍

HPA的工作模型



HPA被实现为一个控制循环，周期由控制器管理器的HPA Sync Period标志（默认值为15秒）控制。

在每个期间，控制器管理器都会根据每个HPA定义中指定的度量来查询资源利用率。控制器管理器从资源度量API（针对每个Pod的资源度量）或自定义度量API（针对所有其他度量）获取度量。

对于每个Pod的资源度量（如CPU），控制器从HPA针对每个Pod的资源度量API获取度量。然后，如果设置了目标利用率值，则控制器将利用率值计算为每个Pod中容器上等效资源请求的百

分比。如果设置了目标原始值，则直接使用原始度量值。然后，控制器获取所有目标Pod的利用率平均值或原始值（取决于指定的目标类型），并生成用于缩放所需副本数量的比率。

官方文档：

- https://kubernetes.io/zh/docs/tasks/run-application/horizontal-pod-autoscale/
- https://kubernetes.io/zh/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

2.2 - 项目背景描述

在实际工作场景中，我们经常会遇到一些访问量不固定的业务，例如上线了秒杀场景类型的业务，在实际环境中对该业务使用的资源其实是不好估算的，这个时候我们就需要用到K8s的HPA应用。HPA可以根据容器的CPU或者内存参数，在资源达到这个阈值时，代表着当前Pod的节点压力较大，可以自动新启动一个Pod来承受当前的业务访问量。HPA的实际应用场景有很多，不仅仅局限于秒杀场景。

下边我们准备启动一个Deployment，配置副本数为1。接着我们配置上HPA自动扩缩容策略，然后对该Pod进行压力测试，来模拟该Pod业务访问变大需要更多的资源，这个时候我们来观察Deployment的副本数变化情况。

2.3 - 项目拆解

任务1：**准备一个K8s集群，如有可复用**

该步骤是前置环境，基于K8s来进行操作。

注意该操作是为了快速搭建起来一个K8s集群，该自建步骤不适用于生产环节。

可参考 https://github.com/easzlab/kubeasz/blob/master/docs/setup/quickStart.md快速安装一个K8s集群。

添加节点可参考：https://github.com/easzlab/kubeasz/blob/master/docs/op/op-node.md

1.2.下载文件

- 下载工具脚本easzup，举例使用kubeasz版本2.2.1

```
export release=2.2.1
curl -C- -fLO --retry 3 https://github.com/easzlab/kubeasz/releases/downloac
chmod +x ./easzup
```

- 使用工具脚本下载

默认下载最新推荐k8s/docker等版本，使用命令 `./easzup` 查看工具脚本的帮助信息

```
# 举例使用 k8s 版本 v1.18.2，docker 19.03.5
./easzup -D -d 19.03.5 -k v1.18.2
```

- 可选下载离线系统包 (适用于无法使用yum/apt仓库情形)

```
./easzup -P
```

上述脚本运行成功后，所有文件（kubeasz代码、二进制、离线镜像）均已整理好放入目录 `/etc/ansible`

- `/etc/ansible` 包含 kubeasz 版本为 ${release} 的发布代码
- `/etc/ansible/bin` 包含 k8s/etcd/docker/cni 等二进制文件
- `/etc/ansible/down` 包含集群安装时需要的离线容器镜像
- `/etc/ansible/down/packages` 包含集群安装时需要的系统基础软件

1.3.安装集群

- 容器化运行 kubeasz，详见文档

```
./easzup -S
```

- 使用默认配置安装 aio 集群

```
docker exec -it kubeasz easzctl start-aio
```

1.4.验证安装

如果提示kubectl: command not found，退出重新ssh登录一下，环境变量生效即可。

```
$ kubectl version          # 验证集群版本
$ kubectl get node         # 验证节点就绪 (Ready) 状态
$ kubectl get pod -A       # 验证集群pod状态，默认已安装网络插件、coredns、metric
$ kubectl get svc -A       # 验证集群服务状态
```

准备一个K8s集群，该步骤需要用到自定义镜像，所以我们用单节点K8s集群进行模拟。

```
[root@10-55-20-3 ~]# kubectl get nodes
NAME          STATUS                     ROLES    AGE    VERSION
10.55.20.3    Ready                      master   292d   v1.16.3-tke.7
10.55.30.10   Ready,SchedulingDisabled   <none>   292d   v1.14.3-tke.14
10.55.30.16   Ready,SchedulingDisabled   <none>   292d   v1.14.3-tke.14
10.55.30.5    Ready                      master   292d   v1.16.3-tke.7
10.55.30.8    Ready,SchedulingDisabled   <none>   292d   v1.14.3-tke.14
10.55.40.13   Ready,SchedulingDisabled   <none>   292d   v1.14.3-tke.14
10.55.40.14   Ready                      master   292d   v1.16.3-tke.7
10.55.40.4    Ready                      <none>   292d   v1.14.3-tke.14
[root@10-55-20-3 ~]#
```

不同的K8s集群，红框部分显示nodename不同，接下来我们登录到该节点构建镜像。

任务2：构建一个Docker镜像，用于启动Deployment

该步骤我们需要对容器进行压力测试，单纯的nginx可以承受很高的并发量，所以我们需要构建一个Docker镜像。
首先创建一个用于模拟的压力测试的PHP页面。

```
vim index.php

<?php
  $x = 0.0001;
  for ($i = 0; $i <= 1000000; $i++) {
    $x += sqrt($x);
  }
  echo "OK!";
?>
```

然后保存，我们在同目录创建一个Dockerfile进行镜像构建。

```
vim Dockerfile

FROM php:5-apache
ADD index.php /var/www/html/index.php
RUN chmod a+rx index.php

```

接下来构建Docker镜像。此时目录有

```
[root@VM_40_4_centos docker]# ls
Dockerfile  index.php
```

```
docker build ./ -t php-apache:hpatest   # -t 指定镜像名字
```

```
Dockerfile  index.php
[root@VM_40_4_centos docker]# docker build ./ -t php-apache:hpatest
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM php:5-apache
5-apache: Pulling from library/php
5e6ec7f28fb7: Pull complete
cf165947b5b7: Pull complete
7bd37682846d: Pull complete
99daf8e838e1: Pull complete
ae320713efba: Pull complete
ebcb99c48d8c: Pull complete
9867e71b4ab6: Pull complete
936eb418164a: Pull complete
bc298e7adaf7: Pull complete
ccd61b587bcd: Pull complete
b2d4b347f67c: Pull complete
56e9dde34152: Pull complete
9ad99b17eb78: Pull complete
Digest: sha256:0a40fd273961b99d8afe69a61a68c73c04bc0caa9de384d3b2dd9e7986eec86d
Status: Downloaded newer image for php:5-apache
 ---> 24c791995c1e
Step 2/3 : ADD index.php /var/www/html/index.php
 ---> d02daca7a514
Step 3/3 : RUN chmod a+rx index.php
 ---> Running in cf6b553ae552
Removing intermediate container cf6b553ae552
 ---> 54f6ca698e6e
Successfully built 54f6ca698e6e
Successfully tagged php-apache:hpatest
[root@VM_40_4_centos docker]# FROM php:5-apache
```

镜像构建完成，接下来我们创建Deployment并部署到K8s集群中。

任务3： 创建Deployment和Service并部署到K8s集群

我们回到master节点创建Deployment，该步骤我们资源限制配置的CPU资源较低，为了更好地体现出来HPA策略的时效性。

```
vim php-apache.yaml

apiVersion: apps/v1       #指定API版本
kind: Deployment          #指定资源类型为deployment
metadata:                 #元数据
  name: php-apache        # 名字
spec:                     #pod中容器的详细定义
  selector:               #有基于等式和基于集合的两种表达方式
    matchLabels:           #一个键值对的映射
      run: php-apache     #标签
  replicas: 1              #指定pod数量为1
  template:                 #用于创建pod的模板
    metadata:               #元数据
      labels:               #元数据标签列表
        run: php-apache     #标签
    spec:                   #pod中容器的详细定义
      containers:           #pod中的容器列表，可以有多个容器
      - name: php-apache     #pod名字
        image: php-apache:hpatest     #镜像
        ports:                        #端口
```

```
        - containerPort: 80           #端口
        resources:                    #资源限制
          limits:                    # 资源限制的设置
            cpu: 200m
          requests:                   #资源请求的设置
            cpu: 200m

```

接下来我们创建Service。

```
vim php-apache-svc.yaml

apiVersion: v1                      # 版本号
kind: Service                      #指定资源类型为 Service
metadata:                          #元数据
  name: php-apache                  # 名字
  labels:                         #标签
    run: php-apache               #标签
spec:                          #service的详细定义
  ports:                      #端口
  - port: 80                  #对外提供端口
  selector:                   #有基于等式和基于集合的两种表达方式
    run: php-apache          #关联这个标签的po
```

然后我们把它部署到K8s集群中

```
 kubectl apply -f php-apache.yaml
 kubectl apply -f php-apache-svc.yaml
```

```
[root@10-55-20-3 yml]# kubectl apply -f php-apache.yaml
deployment.apps/php-apache created
service/php-apache created
```

接着我们来验证一下。

该步骤红框的IP，是我们刚刚登陆node节点的IP，K8s默认拉取镜像的是`IfNotPresent`。本地有则使用本地镜像，不拉取。

```
[root@10-55-20-3 yml]# kubectl get pod -o wide |grep php-apache
php-apache-7c848f56c4-bp9kh                  2/2     Running   0          85s     10.150.9.188   10.55.40.4   <none>           <none>
[root@10-55-20-3 yml]# kubectl get service |grep php-apache
php-apache                 ClusterIP      10.150.148.14    <none>          80/TCP           88s
[root@10-55-20-3 yml]#
```

接下来我们配置HPA策略。

任务4：配置HPA策略并验证是否生效

现在，php-apache服务器已经运行，我们将通过kubectl autoscale命令创建HPA。以下命令将创建一个HPA用于控制我们上一步骤中创建的Deployment，使Pod的副本数量在维持在1到10之间。大致来说，HPA将通过增加或者减少Pod副本的数量（通过Deployment）以保持所有Pod的平均CPU利用率在50%以内（由于每个Pod通过yaml文件申请了200 milli-cores CPU，所以50%的CPU利用率意味着平均CPU利用率为100 milli-cores）。

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

```
[root@10-55-20-3 yml]# kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
horizontalpodautoscaler.autoscaling/php-apache autoscaled
[root@10-55-20-3 yml]#
```

```
kubectl get hpa #查看HPA创建状态
```

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    0%/50%    1         10        1          6m12s
```

接下来我们测试访问。来将当前业务Pod资源压力加大。
首先使用kubectl get svc得到Service地址。

```
kubectl get service |grep php
```

```
php-apache   ClusterIP   172.26.12.254   <none>        80/TCP    9m16s
```

然后我们在主机执行命令，来将当前业务Pod资源压力加大，注意红框部分是SVC地址，不同的集群IP不一样。

```
while true; do wget -q -O- http://svcIP; done
while true; do wget -q -O- http://172.26.12.254; done
```

```
[root@iZbp12nqeth7qkzk8tmczwZ test]# while true; do wget -q -O- http://172.26.12.254; done
OK!OK!OK!OK!
```

然后我们查看HPA负载，该步骤需要稍微等待几分钟，可以查看到负载逐渐上升。

```
kubectl get hpa
```

```
[root@iZbp1h8rg4tw6a3ll7s7syZ yml]# kubectl get  hpa
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    17%/50%   1         10        1          10m
```

然后我们再查看HPA。

```
kubectl get hpa
```

```
[root@iZbp1h8rg4tw6a3ll7s7syZ yml]# kubectl get  hpa
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    35%/50%   1         10        2          14m
```

可以查看到Replicas已经扩容了一个Pod。
然后我们get pod查看。

```
kubectl get pod |grep php
```

```
[root@iZbp1h8rg4tw6a3ll7s7syZ yml]# kubectl get pod |grep php
php-apache-7c848f56c4-hn75x   1/1     Running   0          20m
php-apache-7c848f56c4-qtql7   1/1     Running   0          7m3s
[root@iZbp1h8rg4tw6a3ll7s7syZ yml]#
```

然后我们在稍微等一下，再get hpa

```
kubectl get hpa
```

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    57%/50%   1         10        3          19m
[root@iZbp1h8rg4tw6a3ll7s7syZ yml]#
```

因为我们的压力任务并未停止，可以看到又扩容了一个Pod。
然后我们get pod查看，可以查看到又启动了一个Pod。

```
[root@iZbp1h8rg4tw6a3ll7s7syZ yml]# kubectl get pod  |grep php
php-apache-7c848f56c4-5czrj   1/1     Running   0          84s
php-apache-7c848f56c4-hn75x   1/1     Running   0          26m
php-apache-7c848f56c4-qtql7   1/1     Running   0          13m
[root@iZbp1h8rg4tw6a3ll7s7syZ yml]#
```

此时我们停掉压力访问任务，按ctrl + c终结掉。
此时访问已经没有，Pod的压力会逐步降下来。过一会在查看Pod，会发现他自动缩成一个了。

```
[root@iZbp1h8rg4tw6a3ll7s7syZ yml]# kubectl get hpa
NAME         REFERENCE               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          46m
[root@iZbp1h8rg4tw6a3ll7s7syZ yml]# kubectl get pod  |grep php
php-apache-7c848f56c4-hn75x   1/1     Running   0          49m
[root@iZbp1h8rg4tw6a3ll7s7syZ yml]#
```

3 – 项目总结

通过完成此项目，您可以掌握在生产环境中HPA的自动扩容。在实际的工作中，HPA有很多应用场景，可应用于突发流量或秒杀类业务，或者新上线对Pod所需资源不确定等业务，都可以使用HPA来自动扩缩容资源。在该项目中，您掌握了以下技能：

1. 掌握HPA的工作原理和实现机制；
2. 掌握通过Kubernetes原生能力配置HPA策略；
3. 通过实验验证HPA特性，并总结最佳实践。

4 – 作业

作业提交规范：
1. 实现X项任务中的所有要求。
2. 需要为每一个任务创建文件夹，命名为任务1，任务2等
3. 将每个任务要求提交的素材，放入创建好的文件夹，并且整体打包上传

作业评分标准：

任务名称	任务总分	作业提交内容与评分标准	作业提交要点
任务2	30	提交Docker构建任务运行成功截图	创建Docker镜像
任务3	30	提交任务运行成功截图准备具有一个节点可用的K8s集群	部署Deployment到K8s集群
任务4	40	测试HPA自动扩缩容并提交运行成功截图	配置HPA策略并查看扩缩容是否成功

例:

重要提示：

交付前请您按文档的流程跑一遍代码， 提出疑问和有问题的地方用红色标记修正在文档中！（非常重要）

参考文档:
https://shimo.im/docs/WJOWzx3lPGoMTz4E

验收清单：

1. 导学视频一份 （留下选项即可）？ 已提供/未提供/不需要
2. 项目文档一份（word），极客提供模板，在该模板中直接填写内容
3. 是否自己跑过代码（留下选项即可）？ 是/否
4. 项目源代码 已提供/未提供/不需要
5. 作业答案一份（与学生提交样式一致，留下选项即可）？ 是/否