

## Docker+K8S+DevOps 微服务架构师

**学神 IT 教育：从零基础到实战，从入门到精通！**

### 版权声明：

本系列文档为《学神 IT 教育》内部使用教材和教案，只允许 VIP 学员个人使用，禁止私自传播。否则将取消其 VIP 资格，追究其法律责任，请知晓！

### 免责声明：

本课程设计目的只用于教学，切勿使用课程中的技术进行违法活动，学员利用课程中的技术进行违法活动，造成的后果与讲师本人及讲师所属机构无关。倡导维护网络安全人人有责，共同维护网络文明和谐。

### 联系方式：

学神 IT 教育官方网站: <http://www.xuegod.cn>

学神 K8S 精英学习 11 群 QQ 群: 957231097



学习顾问：小语老师

学习顾问：边边老师

学神微信公众号

微信扫码添加学习顾问微信，同时扫码关注学神公众号了解最新动态，获取更多学习资料及答疑就业服务！

## 第 1 章 kubernetes 中如何实现 Pod 自动扩缩容

本节所讲内容:

1.1、深入剖析 k8s 应用自动扩缩容的方案

1.2 k8s 中自动扩缩容的方案

1.3 如何实现 k8s 中的应用自动扩缩容?

实战 1: 基于 CPU 指标实现 pod 自动扩缩容

实战 2: 基于内存指标实现 pod 自动扩缩容

实战 3: kubernetes cluster-autoscaler

做实验需要有套 k8s 集群环境, 我们为了节省资源, k8s 只需要一个控制节点一个工作节点即可:

```
[root@xuegod63 ~]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
xuegod63	Ready	control-plane,master	31d	v1.20.4
xuegod64	Ready	worker	31d	v1.20.4

### 1.1 深入剖析 k8s 应用自动扩缩容的方案

#### 1.1.1 为什么要自动扩缩容?

在实际的业务场景中, 我们经常会遇到某个服务需要扩容的场景 (例如: 测试对服务压测、电商平台秒杀、大促活动、或由于资源紧张、工作负载降低等都需要对服务实例数进行扩缩容操作)。

在 k8s 中扩缩容分为两种:

##### 1、Node 层面:

在使用 kubernetes 集群经常问到的一个问题是, 我应该保持多大的节点规模来满足应用需求呢?

cluster-autoscaler 的出现解决了这个问题, 可以通过 cluster-autoscaler 实现节点级别的动态添加与删除, 动态调整容器资源池, 应对峰值流量

##### 2、Pod 层面:

我们一般会使用 Deployment 中的 replicas 参数, 设置多个副本集来保证服务的高可用, 但是这是一个固定的值, 比如我们设置 10 个副本, 就会启 10 个 pod 同时 running 来提供服务。

如果这个服务平时流量很少的时候, 也是 10 个 pod 同时在 running, 而流量突然暴增时, 又可能出现 10 个 pod 不够用的情况。针对这种情况怎么办? 就需要自动扩缩容:

Kubernetes 对 Pod 的扩缩容分为:

手动和自动两种

**1、手动模式:** 通过 kubectl scale 命令, 这样需要每次去手工操作一次, 而且不确定什么时候业务请求量就很大了, 所以如果不能做到自动化的去扩缩容的话, 这也是一个很麻烦的事情

**2、自动模式:** 如果 Kubernetes 系统能够根据 Pod 当前的负载的变化情况来自动的进行扩缩容就好了, 因为这个过程本来就是不固定的, 频繁发生的, 所以纯手工的方式不是很现实

## 1.2 自动扩缩容的方案有哪些?

### 1.2.1 Kubernetes HPA (Horizontal Pod Autoscaling)

通过此功能, 只需简单的配置, 便可以利用监控指标 (cpu 使用率、磁盘、自定义的等) 自动的扩容或缩容服务中 Pod 数量, 当业务需求增加时, 系统将无缝地自动增加适量 pod 容器, 提高系统稳定性。

### 1.2.2 kubernetes KPA (Knative Pod Autoscaler)

基于请求数对 Pod 自动扩缩容, KPA 的主要限制在于它不支持基于 CPU 的自动扩缩容。

### 1.2.3 kubernetes VPA (Vertical Pod Autoscaler)

垂直 Pod 自动扩缩容, VPA 会基于 Pod 的资源使用情况自动为集群设置资源占用的限制, 从而让集群将 Pod 调度到有足够资源的最佳节点上。VPA 也会保持最初容器定义中资源 request 和 limit 的占比。

它会根据容器资源使用率自动设置 pod 的 CPU 和内存的 requests, 从而允许在节点上进行适当的调度, 以便为每个 Pod 提供适当的可用的节点。它既可以缩小过度请求资源的容器, 也可以根据其使用情况随时提升资源不足的容量。

## 1.3 如何实现 k8s 中的应用自动扩缩容?

### 1.3.1 基于 HPA

要想实现自动扩缩容, 需要先考虑如下几点:

#### 1.通过哪些指标决定扩缩容?

HPA v1 版本可以根据 CPU 使用率来进行自动扩缩容:

但是并非所有的系统都可以仅依靠 CPU 或者 Memory 指标来扩容, 对于大多数 Web 应用的后端来说, 基于每秒的请求数量进行弹性伸缩来处理突发流量会更加的靠谱, 所以对于一个自动扩缩容系统来说, 我们不能局限于 CPU、Memory 基础监控数据, 每秒请求数 RPS 等自定义指标也是十分重要。

HPA v2 版本可以根据自定义的指标进行自动扩缩容

**注意:** hpa v1 只能基于 cpu 做扩容所用

hpa v2 可以基于内存和自定义的指标做扩容和缩容

#### 2.如何采集资源指标?

如果我们的系统默认依赖 Prometheus, 自定义的 Metrics 指标则可以从各种数据源或者 exporter 中获取, 基于拉模型的 Prometheus 会定期从数据源中拉取数据。也可以基于 metrics-server 自动获取节点和 pod 的资源指标

#### 3.如何实现自动扩缩容?

K8s 的 HPA controller 已经实现了一套简单的自动扩缩容逻辑, 默认情况下, 每 30s 检测一次指标, 只要检测到了配置 HPA 的目标值, 则会计算出预期的工作负载的副本数, 再进行扩缩容操作。同时, 为了避免过于频繁的扩缩容, 默认在 5min 内没有重新扩缩容的情况下, 才会触发扩缩容。

HPA 本身的算法相对比较保守, 可能并不适用于很多场景。例如, 一个快速的流量突发场景, 如果正处在 5min 内的 HPA 稳定期, 这个时候根据 HPA 的策略, 会导致无法扩容。

### 1.3.2 基于 KPA

1、根据并发请求数实现自动扩缩容

2、设置扩缩容边界实现自动扩缩容

扩缩容边界指应用程序提供服务的最小和最大 Pod 数量。通过设置应用程序提供服务的最小和最大 Pod 数量实现自动扩缩容。

相比 HPA, KPA 会考虑更多的场景, 其中一个比较重要的是流量突发的时候

### 1.3.3 基于 VPA

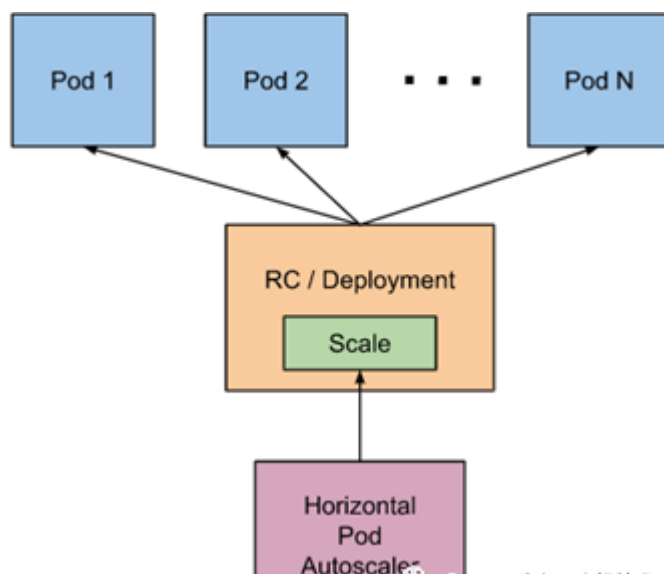
当目前运行 pod 的节点资源达不到 VPA 的推荐值, 就会执行 pod 驱逐, 重新部署新的足够资源的服务。VPA 是 Kubernetes 比较新的功能, 还没有在生产环境大规模实践过, 不建议在线上环境使用自动更新模式, 但是使用推荐模式你可以更好了解服务的资源使用情况。

## 实战 1: 利用 HPA 基于 CPU 指标实现 pod 自动扩缩容

HPA 全称是 Horizontal Pod Autoscaler, 翻译成中文是 POD 水平自动伸缩, HPA 可以基于 CPU 利用率对 deployment 中的 pod 数量进行自动扩缩容 (除了 CPU 也可以基于自定义的指标进行自动扩缩容)。pod 自动缩放不适用于无法缩放的对象, 比如 DaemonSets。

HPA 由 **Kubernetes API 资源和控制器实现**。控制器会周期性的获取平均 CPU 利用率, 并与目标值相比较后调整 deployment 中的副本数量。

### 1、HPA 工作原理



HPA 是根据指标来进行自动伸缩的, 目前 HPA 有两个版本-v1 和 v2beta

HPA 的 API 有三个版本, 通过 `kubectl api-versions | grep autoscal` 可看到  
autoscaling/v1  
autoscaling/v2beta1

autoscaling/v2beta2

autoscaling/v1 只支持基于 CPU 指标的缩放;

autoscaling/v2beta1 支持 Resource Metrics (资源指标, 如 pod 的 CPU, 内存) 和 Custom Metrics (自定义指标) 的缩放;

autoscaling/v2beta2 支持 Resource Metrics (资源指标, 如 pod 的 CPU, 内存) 和 Custom Metrics (自定义指标) 和 ExternalMetrics (额外指标) 的缩放, 但是目前也仅仅是处于 beta 阶段  
指标从哪里来?

K8S 从 1.8 版本开始, CPU、内存等资源的 metrics 信息可以通过 Metrics API 来获取, 用户可以直接获取这些 metrics 信息 (例如通过执行 `kubect top` 命令), HPA 使用这些 metrics 信息来实现动态伸缩。

Metrics server:

- 1、Metrics server 是 K8S 集群资源使用情况的聚合器
- 2、从 1.8 版本开始, Metrics server 可以通过 yaml 文件的方式进行部署
- 3、Metrics server 收集所有 node 节点的 metrics 信息

HPA 如何运作?

HPA 的实现是一个控制循环, 由 controller manager 的 `--horizontal-pod-autoscaler-sync-period` 参数指定周期 (默认值为 15 秒)。每个周期内, controller manager 根据每个 HorizontalPodAutoscaler 定义中指定的指标查询资源利用率。controller manager 可以从 resource metrics API (pod 资源指标) 和 custom metrics API (自定义指标) 获取指标。

然后, 通过现有 pods 的 CPU 使用率的平均值 (计算方式是最近的 pod 使用量 (最近一分钟的平均值, 从 metrics-server 中获得) 除以设定的每个 Pod 的 CPU 使用率限额) 跟目标使用率进行比较, 并且在扩容时, 还要遵循预先设定的副本数限制:  $\text{MinReplicas} \leq \text{Replicas} \leq \text{MaxReplicas}$ 。

计算扩容后 Pod 的个数:  $\text{sum}(\text{最近一分钟内某个 Pod 的 CPU 使用率的平均值}) / \text{CPU 使用上限的整数} + 1$

流程:

- 1、创建 HPA 资源, 设定目标 CPU 使用率限额, 以及最大、最小实例数
- 2、收集一组中 (PodSelector) 每个 Pod 最近一分钟内的 CPU 使用率, 并计算平均值
- 3、读取 HPA 中设定的 CPU 使用限额
- 4、计算: 平均值之和/限额, 求出目标调整的实例个数
- 5、目标调整的实例数不能超过 1 中设定的最大、最小实例数, 如果没有超过, 则扩容; 超过, 则扩容至最大的实例个数
- 6、回到 2, 不断循环

2、安装数据采集组件 metrics-server

metrics-server 是一个集群范围内的资源数据收集和工具, 同样的, metrics-server 也只是显示数据, 并不提供数据存储服务, 主要关注的是资源度量 API 的实现, 比如 CPU、文件描述符、内存、请求延时等指标, metric-server 收集数据给 k8s 集群内使用, 如 kubectl,hpa,scheduler 等

#### 1.部署 metrics-server 组件

##### #通过离线方式获取镜像

需要的镜像是: k8s.gcr.io/metrics-server-amd64:v0.3.6 和 k8s.gcr.io/addon-resizer:1.8.4  
镜像所在地址在课件, 可自行下载, 如果大家机器不能访问外部网络, 可以把镜像上传到 k8s 的各个节点, 按如下方法手动解压:

```
docker load -i metrics-server-amd64-0-3-6.tar.gz
```

```
docker load -i addon.tar.gz
```

##### #部署 metrics-server 服务

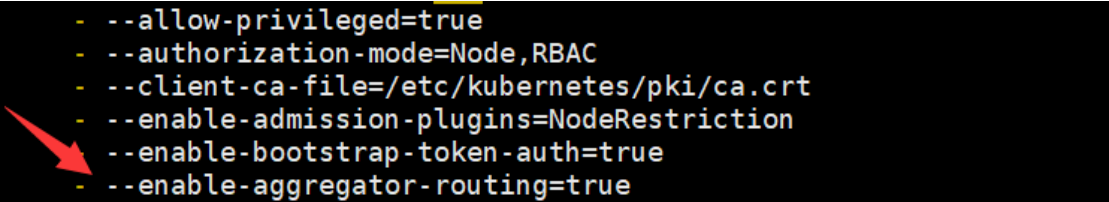
#在/etc/kubernetes/manifests 里面改一下 apiserver 的配置

注意: 这个是 k8s 在 1.7 的新特性, 如果是 1.16 版本的可以不用添加, 1.17 以后要添加。这个参数的作用是 Aggregation 允许在不修改 Kubernetes 核心代码的同时扩展 Kubernetes API。

```
[root@xuegod63 ~]# vim /etc/kubernetes/manifests/kube-apiserver.yaml
```

增加如下内容:

```
- --enable-aggregator-routing=true
```



```
- --allow-privileged=true
- --authorization-mode=Node,RBAC
- --client-ca-file=/etc/kubernetes/pki/ca.crt
- --enable-admission-plugins=NodeRestriction
- --enable-bootstrap-token-auth=true
- --enable-aggregator-routing=true
```

重新更新 apiserver 配置:

```
[root@xuegod63 ~]# kubectl apply -f /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
[root@xuegod63 ~]# kubectl apply -f metrics.yaml
```

##### #验证 metrics-server 是否部署成功

```
[root@xuegod63 ~]# kubectl get pods -n kube-system | grep metrics
```

```
metrics-server-6595f875d6-ml5pc          2/2      Running          0
```

#测试 kubectl top 命令

```
[root@xuegod63 ~]# kubectl top nodes
```

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
xuegod62	282m	4%	1408Mi	24%
xuegod63	673m	11%	2518Mi	44%
xuegod64	286m	4%	1305Mi	22%

```
[root@xuegod63 ~]# kubectl top pods -n kube-system
```

NAME	CPU(cores)	MEMORY(bytes)
calico-kube-controllers-6949477b58-xbt82	3m	13Mi
calico-node-4t7bs	118m	55Mi
calico-node-hbx75	105m	88Mi
calico-node-hdr45	92m	95Mi
coredns-7f89b7bc75-prt8x	9m	13Mi
coredns-7f89b7bc75-xncd5	8m	11Mi



etcd-xuegod63	62m	44Mi
kube-apiserver	0m	0Mi
kube-apiserver-xuegod63	204m	332Mi
kube-controller-manager-xuegod63	64m	48Mi
kube-proxy-6hwqz	1m	17Mi
kube-proxy-jhlhv	1m	19Mi
kube-proxy-wx64x	2m	12Mi
kube-scheduler-xuegod63	10m	22Mi
metrics-server-6595f875d6-ml5pc	3m	16Mi

### 3、创建 php-apache 服务，利用 HPA 进行自动扩缩容。

#基于 dockerfile 构建一个 PHP-apache 项目

#### 1) 创建并运行一个 php-apache 服务

使用 dockerfile 构建一个新的镜像，在 k8s 的 xuegod63 节点构建

```
[root@xuegod63 ~]# mkdir php
```

```
[root@xuegod63 ~]# cd php/
```

```
[root@xuegod63 php]# cat dockerfile
```

```
FROM php:5-apache
```

```
ADD index.php /var/www/html/index.php
```

```
RUN chmod a+rx index.php
```

```
[root@xuegod63 php]# cat index.php
```

```
<?php
```

```
$x = 0.0001;
```

```
for ($i = 0; $i <= 1000000;$i++) {
```

```
$x += sqrt($x);
```

```
}
```

```
echo "OK!";
```

```
?>
```

#构建镜像

```
[root@xuegod63 php]# docker build -t k8s.gcr.io/hpa-example:v1 .
```

#打包镜像

```
docker save -o hpa-example.tar.gz k8s.gcr.io/hpa-example:v1
```

#解压镜像

可以把镜像传到 k8s 的各个工作节点，通过 docker load -i hpa-example.tar.gz 进行解压

#通过 deployment 部署一个 php-apache 服务

```
[root@xuegod63 ~]# cat php-apache.yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: php-apache
```

```
spec:
```

```
  selector:
```

```
    matchLabels:
```

```
      run: php-apache
```

```
replicas: 1
template:
  metadata:
    labels:
      run: php-apache
  spec:
    containers:
      - name: php-apache
        image: k8s.gcr.io/hpa-example:v1
        ports:
          - containerPort: 80
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
```

---

```
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
    - port: 80
  selector:
    run: php-apache
```

**#更新资源清单文件**

```
[root@xuegod63 ~]# kubectl apply -f php-apache.yaml
```

deployment.apps/php-apache created

service/php-apache created

**#验证 php 是否部署成功**

```
[root@xuegod63 ~]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
php-apache-7d8fdb687c-bq8c7	1/1	Running	0	31s

#### 4、创建 HPA

php-apache 服务正在运行, 使用 kubectl autoscale 创建自动缩放器, 实现对 php-apache 这个 deployment 创建的 pod 自动扩缩容, 下面的命令将会创建一个 HPA, HPA 将会根据 CPU, 内存等资源指标增加或减少副本数, 创建一个可以实现如下目的的 hpa:

- 1) 让副本数维持在 1-10 个之间 (这里副本数指的是通过 deployment 部署的 pod 的副本数)
- 2) 将所有 Pod 的平均 CPU 使用率维持在 50% (通过 kubectl run 运行的每个 pod 如果是 200



毫核, 这意味着平均 CPU 利用率为 100 毫核)

#给上面 php-apache 这个 deployment 创建 HPA

```
[root@xuegod63 ~]# kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

#上面命令解释说明

kubectl autoscale deployment php-apache (php-apache 表示 deployment 的名字) --cpu-percent=50 (表示 cpu 使用率不超过 50%) --min=1 (最少一个 pod) --max=10 (最多 10 个 pod)

#验证 HPA 是否创建成功

```
[root@xuegod63 ~]# kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	0%/50%	1	10	1	53m

**注:** 由于我们没有向服务器发送任何请求, 因此当前 CPU 消耗为 0% (TARGET 列显示了由相应的 deployment 控制的所有 Pod 的平均值)。

##### 5、压测 php-apache 服务, 只是针对 CPU 做压测

#把 busybox.tar.gz 和 nginx-1-9-1.tar.gz 上传到 xuegod64 上, 手动解压:

```
docker load -i busybox.tar.gz
```

```
docker load -i nginx-1-9-1.tar.gz
```

启动一个容器, 并将无限查询循环发送到 php-apache 服务 (复制 k8s 的 master 节点的终端, 也就是打开一个新的终端窗口):

```
kubectl run v1 -it --image=busybox /bin/sh
```

登录到容器之后, 执行如下命令

```
while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
```

在一分钟左右的时间, 我们通过执行以下命令来看到更高的 CPU 负载

```
kubectl get hpa
```

显示如下:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
php-apache	Deployment/php-apache	231%/50%	1	10	4

上面可以看到, CPU 消耗已经达到 256%, 每个 pod 的目标 cpu 使用率是 50%, 所以, php-apache 这个 deployment 创建的 pod 副本数将调整为 5 个副本, 为什么是 5 个副本, 因为  $256/50=5$

```
kubectl get pod
```

显示如下:

NAME	READY	STATUS	RESTARTS	AGE
php-apache-5694767d56-b2kd7	1/1	Running	0	18s
php-apache-5694767d56-f9vzm	1/1	Running	0	2s
php-apache-5694767d56-hpgb5	1/1	Running	0	18s
php-apache-5694767d56-mm8r	1/1	Running	0	4h13m
php-apache-5694767d56-zljkd	1/1	Running	0	18s

```
kubectl get deployment php-apache
```

显示如下:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
php-apache	5/5	5	5	2h1m

注意: 可能需要几分钟来稳定副本数。由于不以任何方式控制负载量, 因此最终副本数可能会与此示例不同。

停止对 php-apache 服务压测, HPA 会自动对 php-apache 这个 deployment 创建的 pod 做缩容

停止向 php-apache 这个服务发送查询请求, 在 busybox 镜像创建容器的终端中, 通过 <Ctrl> + C 把刚才 while 请求停止, 然后, 我们将验证结果状态 (大约一分钟后):

kubectl get hpa

显示如下:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	0%/50%	1	10	1	53m

kubectl get deployment php-apache

显示如下:

php-apache	1/1	1	1	5s
------------	-----	---	---	----

通过上面可以看到, CPU 利用率下降到 0, 因此 HPA 自动将副本数缩减到 1。

注意: 自动缩放副本可能需要几分钟。

## 实战 2: 利用 HPA 基于内存指标实现 pod 自动扩缩容

1、创建一个 nginx 的 pod

[root@xuegod63 ~]# cat nginx.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-hpa

spec:

selector:

matchLabels:

app: nginx

replicas: 1

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.9.1

ports:

- containerPort: 80

name: http

```
    protocol: TCP
  resources:
    requests:
      cpu: 0.01
      memory: 25Mi
    limits:
      cpu: 0.05
      memory: 60Mi
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  selector:
    app: nginx
  type: NodePort
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30080
#更新资源清单文件
[root@xuegod63 ~]# kubectl apply -f nginx.yaml
deployment.apps/nginx-hpa created
service/nginx created
2、验证 nginx 是否运行
[root@xuegod63 ~]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-hpa-fb74696c-6m6st           1/1     Running   0           28s
```

### 注意:

nginx 的 pod 里需要有如下字段, 否则 hpa 会采集不到内存指标

```
resources:
  requests:
    cpu: 0.01
    memory: 25Mi
  limits:
    cpu: 0.05
    memory: 60Mi
```

### 3、创建一个 hpa

```
[root@xuegod63 ~]# cat hpa-v1.yaml
```

```
apiVersion: autoscaling/v2beta1
```

```
kind: HorizontalPodAutoscaler
```

```
metadata:
```

```
  name: nginx-hpa
```

```
spec:
```

```
  maxReplicas: 10
```

```
  minReplicas: 1
```

```
  scaleTargetRef:
```

```
    apiVersion: apps/v1
```

```
    kind: Deployment
```

```
    name: nginx-hpa
```

```
  metrics:
```

```
  - type: Resource
```

```
    resource:
```

```
      name: memory
```

```
      targetAverageUtilization: 60
```

```
#更新资源清单文件
```

```
[root@xuegod63 ~]# kubectl apply -f hpa-v1.yaml
```

```
horizontalpodautoscaler.autoscaling/nginx-hpa created
```

```
#查看创建的 hpa
```

```
[root@xuegod63 ~]# kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
nginx-hpa	Deployment/nginx-hpa	5%/60%	1	10	1

4、压测 nginx 的内存, hpa 会对 pod 自动扩缩容

登录到上面通过 pod 创建的 nginx, 并生成一个文件, 增加内存

```
kubectl exec -it nginx-hpa-fb74696c-6m6st -- /bin/sh
```

```
#压测
```

```
dd if=/dev/zero of=/tmp/a
```

```
#打开新的终端
```

```
[root@xuegod63 ~]# kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
nginx-hpa	Deployment/nginx-hpa	200%/60%	1	10	1	

上面的 targets 列可看到 200%/60%, 200%表示当前 cpu 使用率, 60%表示所有 pod 的 cpu 使用率维持在 60%, 现在 cpu 使用率达到 200%, 所以 pod 增加到 4 个

```
[root@xuegod63 ~]# kubectl get deployment
```

```
显示如下:
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-hpa	4/4	4	4	25m

```
[root@xuegod63 ~]# kubectl get pods
```

```
显示如下:
```

NAME	READY	STATUS	RESTARTS	AGE
------	-------	--------	----------	-----

nginx-hpa-bb598885d-j4kcp	1/1	Running	0	25m
nginx-hpa-bb598885d-rj5hk	1/1	Running	0	63s
nginx-hpa-bb598885d-twv9c	1/1	Running	0	18s
nginx-hpa-bb598885d-v9ft5	1/1	Running	0	63s

5、取消对 nginx 内存的压测, hpa 会对 pod 自动缩容

```
[root@xuegod63 ~]# kubectl exec -it nginx-hpa-fb74696c-6m6st -- /bin/sh
```

删除/tmp/a 这个文件

```
rm -rf /tmp/a
```

```
[root@xuegod63 ~]# kubectl get hpa
```

显示如下,可看到内存使用率已经降到 5%:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS
REPLICAS	AGEngine-hpa	Deployment/nginx-		
hpa 5%/60%	1	10	1	26m

```
[root@xuegod63 ~]# kubectl get deployment
```

显示如下, deployment 的 pod 又恢复到 1 个了:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-hpa	1/1	1	1	38m

扩展: 查看 v2 版本的 hpa 如何定义?

```
kubectl get hpa.v2beta2.autoscaling -o yaml > 1.yaml
```

## 实战 3: kubernetes cluster-autoscaler

### 1、什么是 cluster-autoscaler

Cluster Autoscaler (CA)是一个独立程序, 是用来弹性伸缩 kubernetes 集群的。它可以自动根据部署应用所请求的资源量来动态的伸缩集群。当集群容量不足时, 它会自动去 Cloud Provider (支持 GCE、GKE 和 AWS) 创建新的 Node, 而在 Node 长时间资源利用率很低时自动将其删除以节省开支。

项目地址: <https://github.com/kubernetes/autoscaler>

### 2、Cluster Autoscaler 什么时候伸缩集群?

在以下情况下, 集群自动扩容或者缩放:

扩容: 由于资源不足, 某些 Pod 无法在任何当前节点上进行调度

缩容: Node 节点资源利用率较低时, 且此 node 节点上存在的 pod 都能被重新调度到其他 node 节点上运行

### 3、什么时候集群节点不会被 CA 删除?

1) 节点上有 pod 被 PodDisruptionBudget 控制器限制。

2) 节点上有命名空间是 kube-system 的 pods。

3) 节点上的 pod 不是被控制器创建, 例如不是被 deployment, replica set, job, stateful set 创建。

- 4) 节点上有 pod 使用了本地存储
- 5) 节点上 pod 驱逐后无处可去, 即没有其他 node 能调度这个 pod
- 6) 节点有注解: "cluster-autoscaler.kubernetes.io/scale-down-disabled": "true"(在 CA 1.0.3 或更高版本中受支持)

#### 扩展: 什么是 PodDisruptionBudget?

通过 PodDisruptionBudget 控制器可以设置应用 POD 集群处于运行状态最低个数, 也可以设置应用 POD 集群处于运行状态的最低百分比, 这样可以保证在主动销毁应用 POD 的时候, 不会一次性销毁太多的应用 POD, 从而保证业务不中断

#### 4、Horizontal Pod Autoscaler 如何与 Cluster Autoscaler 一起使用?

Horizontal Pod Autoscaler 会根据当前 CPU 负载更改部署或副本集的副本数。如果负载增加, 则 HPA 将创建新的副本, 集群中可能有足够的空间, 也可能没有足够的空间。如果没有足够的资源, CA 将尝试启动一些节点, 以便 HPA 创建的 Pod 可以运行。如果负载减少, 则 HPA 将停止某些副本。结果, 某些节点可能变得利用率过低或完全为空, 然后 CA 将终止这些不需要的节点。

#### 扩展: 如何防止节点被 CA 删除?

节点可以打上以下标签:

"cluster-autoscaler.kubernetes.io/scale-down-disabled": "true"

可以使用 kubectl 将其添加到节点(或从节点删除):

```
$ kubectl annotate node <nodename> cluster-autoscaler.kubernetes.io/scale-down-disabled=true
```

#### 5、Cluster Autoscaler 支持那些云厂商?

GCE <https://kubernetes.io/docs/concepts/cluster-administration/cluster-management/>

GKE <https://cloud.google.com/container-engine/docs/cluster-autoscaler>

AWS (微软) <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/aws/README.md>

Azure (微软) <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/azure/README.md>

Alibaba Cloud <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/alicloud/README.md>

OpenStack Magnum <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/magnum/README.md>

DigitalOcean <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/digitalocean/README.md>

CloudStack <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/cloudstack/README.md>

Exoscale <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/exoscale/README.md>

Packet <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/packet/README.md>

OVHcloud <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/ovhcloud/README.md>

Linode <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/linode/README.md>

Hetzner <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/hetzner/README.md>

Cluster API <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/clusterapi/README.md>

## 扩展: kubernetes KPA

Github: <https://knative.dev/docs/install/>

安装参考:

<https://knative.dev/docs/install/install-serving-with-yaml/>

## 总结:

- 1.1、深入剖析 k8s 应用自动扩缩容的方案
- 1.2 k8s 中自动扩缩容的方案
- 1.3 如何实现 k8s 中的应用自动扩缩容?
- 实战 1: 基于 CPU 指标实现 pod 自动扩缩容
- 实战 2: 基于内存指标实现 pod 自动扩缩容
- 实战 3: kubernetes cluster-autoscaler