Kubernetes Pod 自动扩缩容

前言：

课程名称：Kubernetes Pod 自动扩缩容

实验环境：

本章节 Kubernetes 集群环境如下：

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd	CPU：2vCPU 硬盘：100G 内存：3GB 开启虚拟化
工作节点	192.168.128.21	k8s-node01	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：12GB 开启虚拟化

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

1、自动（弹性）扩缩容背景分析

背景：

弹性伸缩是根据用户的业务需求和策略，自动“调整”其“弹性资源”的管理服务。通过弹性伸缩功能，用户可设置定时、周期或监控策略，恰到好处地增加或减少“弹性资源”，并完成实例配置，保证业务平稳健康运行。

在实际工作中，我们常常需要做一些扩容缩容操作，如：电商平台在 618 和双十一搞秒杀活动。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：

对 K8s 物理节点扩容和缩容，根据业务规模实现物理节点自动扩缩容。

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？就需要扩容和缩容。

2、k8s 中自动伸缩的方案

2.1、HPA

HPA 官网地址：

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

Kubernetes HPA（Horizontal Pod Autoscaling）：Pod 水平自动伸缩。

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。

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。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

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。

2.2、KPA

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、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。

2.3、VPA

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。

3、利用 HPA 基于 CPU 指标实现 pod 自动扩缩容

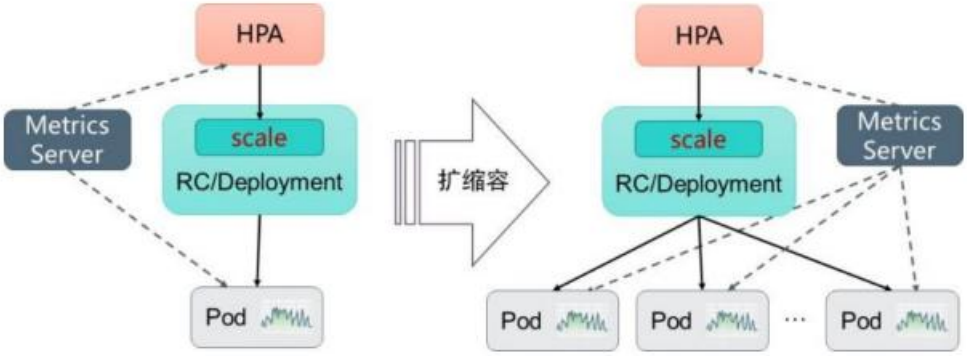
HPA 全称是 Horizontal Pod Autoscaler，翻译成中文是 POD 水平自动伸缩，HPA 可以基于 CPU 利用率对 deployment 中的 pod 数量进行自动扩缩容（除了 CPU

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。

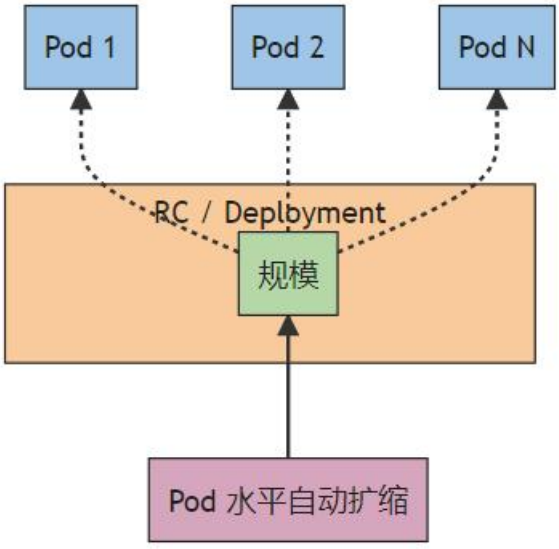
HPA 由 Kubernetes API 资源和控制器实现。控制器会周期性的获取平均 CPU 利用率，并与目标值相比较后调整 deployment 中的副本数量。

3.1、HPA 工作原理

原理图 1：



原理图 2：



HPA 是根据指标来进行自动伸缩的，目前 HPA 有两个版本：v1 和 v2。

HPA 的 API 有两个版本，通过 kubectl api-versions|grep autoscal 可看到如下：

```
[root@k8s-master01 ~]# kubectl api-versions|grep autoscal
autoscaling/v1
autoscaling/v2
```

autoscaling/v1 只支持基于 CPU 指标的缩放。
autoscaling/v2：支持内存、自定义指标、额外指标扩容缩容。

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 metics 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：MinReplicas <= Replicas <= MaxReplicas。

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

工作流程：
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。

3.2、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

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

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、修改 kube-apiserver

```
    # 在/etc/kubernetes/manifests 里面改一下 apiserver 的配置。
    # 注意：这个是 k8s 在 1.17 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@k8s-master01 ~]# vi /etc/kubernetes/manifests/kube-apiserver.yaml
    添加如下内容：
        - --enable-aggregator-routing=true
```

```
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.43.101
    - --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
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
```

```
    [root@k8s-master01 ~]# kubectl apply -f /etc/kubernetes/manifests/kube-apiserver.yaml
    pod/kube-apiserver created
    [root@k8s-master01 ~]# kubectl get pods -n kube-system

    # 把这个 Eroor 的删除掉
    [root@k8s-master01 ~]# kubectl delete pods kube-apiserver -n kube-system
    pod "kube-apiserver" deleted
```

```
[root@k8s-master01 ~]# kubectl get pods -n kube-system
NAME                                       READY   STATUS    RESTARTS   AGE
calico-kube-controllers-659bd7879c-hlhwf   1/1     Running   0          64m
calico-node-7k27r                          1/1     Running   0          64m
calico-node-wrx8s                          1/1     Running   0          64m
calico-node-z95wk                          1/1     Running   0          64m
coredns-7f89b7bc75-62q2z                   1/1     Running   0          65m
coredns-7f89b7bc75-l8hj9                   1/1     Running   0          65m
etcd-k8s-master01                          1/1     Running   0          65m
kube-apiserver                             0/1     Error     2          67s
kube-apiserver-k8s-master01                1/1     Running   0          72s
kube-controller-manager-k8s-master01       0/1     Running   1          65m
kube-proxy-4zq74                           1/1     Running   0          65m
kube-proxy-mfhfz                           1/1     Running   0          65m
kube-proxy-vbfz7                           1/1     Running   0          65m
kube-scheduler-k8s-master01                1/1     Running   1          65m
```

2、部署 metrics-server 服务

```
# 上传 yaml 文件到服务器，部署
[root@k8s-master01 ~]# kubectl apply -f components-v0.6.1.yaml
[root@k8s-master01 ~]# kubectl get pods -n kube-system -l
k8s-app=metrics-server
```

```
[root@k8s-master01 ~]# kubectl get pods -n kube-system -l k8s-app=metrics-server
NAME                              READY   STATUS    RESTARTS   AGE
metrics-server-5cfd988496-cbc29   1/1     Running   0          36s
[root@k8s-master01 ~]#
```

```
[root@k8s-master01 ~]# kubectl top pods -n kube-system
```

```
[root@k8s-master01 ~]# kubectl top pods -n kube-system
NAME                                       CPU(cores)   MEMORY(bytes)
coredns-5bbd96d687-gxhbp                   2m           28Mi
coredns-5bbd96d687-h74qj                   2m           18Mi
etcd-k8s-master01                          28m          78Mi
kube-apiserver-k8s-master01                47m          373Mi
kube-controller-manager-k8s-master01       27m          57Mi
kube-proxy-5gknd                           5m           21Mi
kube-proxy-r2vtg                           5m           33Mi
kube-scheduler-k8s-master01                4m           22Mi
metrics-server-5cfd988496-cbc29            6m           20Mi
tigera-operator-5d6845b496-969bv           3m           31Mi
[root@k8s-master01 ~]#
```

```
[root@k8s-master01 ~]# kubectl top nodes
```

```
[root@k8s-master01 ~]# kubectl top nodes
NAME           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
k8s-master01   202m         5%     1312Mi          35%
k8s-node01     131m         2%     1032Mi          18%
[root@k8s-master01 ~]#
```

3.3、创建测试 pod

编写用于测试的 Deployment：

```
[root@k8s-master01 ~]# vi test.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hpatest
spec:
  replicas: 1
  selector:
```

```
      matchLabels:
        app: hpatest
    template:
      metadata:
        labels:
          app: hpatest
      spec:
        containers:
        - name: hpatest
          image: nginx
          imagePullPolicy: IfNotPresent
          command: ["/bin/sh"]
          args: ["-c","/usr/sbin/nginx; while true;do echo `hostname
-I` > /usr/share/nginx/html/index.html; sleep 120;done"]
          ports:
          - containerPort: 80
          resources:
            limits:
              cpu: 500m
            requests:
              cpu: 200m
---
apiVersion: v1
kind: Service
metadata:
  name: hpatest-svc
spec:
  selector:
    app: hpatest
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
```

创建 pod

```
[root@k8s-master01 ~]# kubectl apply -f test.yaml
deployment.apps/hpatest created
service/hpatest-svc created

[root@k8s-master01 ~]# kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
hpatest-6f589f96-ngb7j     1/1     Running   0          119s
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 ~]# kubectl get svc|grep hpatest-svc
hpatest-svc    ClusterIP    10.10.109.115    <none>    80/TCP   63s
```

3.4、创建 HPA

Web 服务正在运行，使用 kubectl autoscale 创建自动缩放器，实现对 Web 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 top pods 看到的每个 pod 如果是 200 毫核，这意味着平均 CPU 利用率为 100 毫核）

（1）查看 pods cpu 利用率

```
[root@k8s-master01 ~]# kubectl top pods
NAME                      CPU(cores)   MEMORY(bytes)
hpatest-6f589f96-ngb7j    0m           4Mi
```

（2）给上面 hpatest 这个 deployment 创建 HPA

```
[root@k8s-master01 ~]# kubectl autoscale deployment hpatest
--cpu-percent=50 --min=1 --max=10
horizontalpodautoscaler.autoscaling/baidu-web autoscaled

参数解释：
kubectl autoscale deployment hpatest-svc:hpatest-svc 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
```

也可以使用 yaml 文件进行创建：

编写 HPA，用于水平扩展，当 cpu 达到 50%的利用率的时候开始扩展：

```
[root@k8s-master01 ~]# vi hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpatest
spec:
  minReplicas: 1
  maxReplicas: 10
```

```
      scaleTargetRef:
        apiVersion: apps/v1
        kind: Deployment
        name: hpatest
      metrics:
      - type: Resource
        resource:
          name: cpu
          target:
            averageUtilization: 50
            type: Utilization
```

yaml 文件说明：

```
    apiVersion: autoscaling/v2
    kind: HorizontalPodAutoscaler
    metadata:
      name: haptest
    spec:
      minReplicas: 1  #最小扩容到 1 个 pod
      maxReplicas: 10  #最大扩容到 10 个 pod
      scaleTargetRef:
        apiVersion: apps/v1
        kind: Deployment
        name: hpatest
      metrics:
      - type: Resource
        resource:
          name: cpu
          target:
            averageUtilization: 50  #cpu 平均资源使用率达到 50%就开始扩容，低于 50%就开始缩容
            type: Utilization
```

（3）验证 HPA 是否创建成功

```
    [root@k8s-master01 ~]# kubectl get hpa
[root@k8s-master01 yaml]# kubectl get hpa
NAME      REFERENCE            TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
haptest   Deployment/hpatest   0%/50%    1         10        1          20s
```

提示：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）。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

3.5、压测 Pod 服务，实现 Pod 自动扩缩容

（1）启动一个容器，并将无限查询循环发送到 baidu-web 服务

```
    [root@k8s-master01 ~]# kubectl run test -it --image=busybox /bin/sh
    / # while true; do wget -q -O-
http://hpatest-svc.default.svc.cluster.local; done
```

（2）压力测试 hpa，使 pod 扩容

另开一个终端，在一分钟左右的时间内，我们通过执行以下命令来查看 hpa 的 CPU 负载

```
    [root@k8s-master01 ~]# kubectl get hpa
[root@k8s-master01 yaml]# kubectl get hpa
NAME      REFERENCE            TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
haptest   Deployment/hpatest   74%/50%   1         10        2          2m42s

    [root@k8s-master01 ~]# kubectl get pods
    NAME                       READY   STATUS    RESTARTS   AGE
    hpatest-6f589f96-6879f     1/1     Running   0          37s
    hpatest-6f589f96-ngb7j     1/1     Running   0          9m25s
```

上面可以看到，CPU 消耗已经达到 74%，每个 pod 的目标 cpu 使用率是 50%，所以，hpatest 这个 deployment 创建的 pod 副本数将调整为 2 个副本，为什么是 2 个副本，因为 74/50=2

（3）停止压力测试 hpa，使 pod 缩容

停止对 web 服务压测，HPA 会自动对 hpatest 这个 deployment 创建的 pod 做缩容。

停止向 hpatest 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（大约五分钟后）：

```
    [root@k8s-master01 ~]# kubectl get hpa
[root@k8s-master01 ~]# kubectl get hpa
NAME      REFERENCE            TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
haptest   Deployment/hpatest   0%/50%    1         10        1          9m35s

    [root@k8s-master01 ~]# kubectl get pods
    NAME                       READY   STATUS        RESTARTS   AGE
    hpatest-6f589f96-6879f     0/1     Terminating   0          7m20s
    hpatest-6f589f96-ngb7j     1/1     Running       0          16m

    [root@k8s-master01 ~]# kubectl get pods
    NAME                       READY   STATUS    RESTARTS   AGE
    hpatest-6f589f96-ngb7j     1/1     Running   0          16m
```

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。

4、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@k8s-master01 ~]# vi 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:latest
        imagePullPolicy: IfNotPresent
        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

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
```

2、更新资源清单文件，并查看创建的 pod

```
[root@k8s-master01 ~]# kubectl apply -f nginx.yaml
deployment.apps/nginx-hpa created
service/nginx created

[root@k8s-master01 ~]# kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
nginx-hpa-76584f6d49-r6rn6   1/1     Running   0          10s

[root@k8s-master01 ~]# kubectl get svc
```

```
[root@k8s-master01 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        30d
nginx        NodePort    10.99.145.153   <none>        80:32134/TCP   24s
```

3、创建 HPA

```
[root@k8s-master01 ~]# vi hpa-nginx.yaml
apiVersion: autoscaling/v2
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
            target:
              averageUtilization: 60
              type: Utilization
```

Yaml 文件说明：

```
    apiVersion: autoscaling/v2
    kind: HorizontalPodAutoscaler
    metadata:
      name: nginx-hpa    #hpa 名称
    spec:
        maxReplicas: 10    #最大 pod 数量
        minReplicas: 1    #最小 pod 数量
        scaleTargetRef:    #指明要缩放的资源
          apiVersion: apps/v1  #要缩放资源的 api 版本
          kind: Deployment      #要缩放资源的类型
          name: nginx-hpa       #要缩放资源的名称
        metrics:           #度量指标
        - type: Resource    #type 是度量的类型，可选 External、Object、
Pod、Resource。
          resource:
            name: memory    #资源名称，可选 cpu、memory
            target:  #指定给定度量指标目标值
              averageUtilization: 60  #averageUtilization 表示资源平均
利用率，60 为给定值。同级别可选：averageValue 平均值、value 度量的目标
值。
              type: Utilization    #可选：Utilization 表示是利用率类型、
value 表示值、AverageValue 表示平均值
```

4、更新资源清单文件

```
    [root@k8s-master01 ~]# kubectl apply -f hpa-nginx.yaml
    horizontalpodautoscaler.autoscaling/nginx-hpa created
```

```
[root@k8s-master01 ~]# kubectl get hpa nginx-hpa
```

```
[root@k8s-master01 yaml]# kubectl get hpa nginx-hpa
NAME        REFERENCE              TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
nginx-hpa   Deployment/nginx-hpa   22%/60%   1         10        1          17s
```

5、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

```
[root@k8s-master01 ~]# kubectl exec -it nginx-hpa-76584f6d49-r6rn6 -- /bin/sh
# dd if=/dev/zero of=/tmp/a     #压力测试

root@nginx-hpa-5f5f7bb65b-gb9m9:/# apt-get update
root@nginx-hpa-5f5f7bb65b-gb9m9:/# apt-get install stress
# 启动 5 个进程，分配 3G 内存，分配后不释放，持续时长 100s
root@nginx-hpa-5f5f7bb65b-gb9m9:/# stress --vm 3 --vm-bytes 1G --vm-hang 100 --timeout 100s

[root@k8s-master01 ~]# kubectl get hpa
```

```
[root@k8s-master01 ~]# kubectl get hpa
NAME        REFERENCE              TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
nginx-hpa   Deployment/nginx-hpa   237%/60%   1         10        1          4m14s
```

上面的 targets 列可看到 237%/60%，237%表示当前内存使用率，60%表示所有 pod 的内存使用率维持在 60%，现在内存使用率达到 200%，所以 pod 增加到 5 个

```
[root@k8s-master01 ~]# kubectl get deployment
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
nginx-hpa   5/5     5            5           23h

[root@k8s-master01 ~]# kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
nginx-hpa-76584f6d49-2pr8q   1/1     Running   0          55s
nginx-hpa-76584f6d49-h256b   1/1     Running   0          55s
nginx-hpa-76584f6d49-kt726   1/1     Running   0          24s
nginx-hpa-76584f6d49-r6rn6   1/1     Running   1          23h
nginx-hpa-76584f6d49-wh26r   1/1     Running   0          55s
```

6、取消对 nginx 内存的压测，hpa 会对 pod 自动缩容

```
在 nginx-hpa 容器中删除/tmp/a 这个文件
# rm -rf /tmp/a
```

```
[root@k8s-master01 ~]# kubectl get hpa
```

```
[root@k8s-master01 ~]# kubectl get hpa
NAME        REFERENCE              TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
nginx-hpa   Deployment/nginx-hpa   18%/60%   1         10        5          7m45s
```

可以看到内存使用率已经降下来了，副本数还没有降下来，再等会。

Pod 数量以及恢复正常：

```
[root@k8s-master01 ~]# kubectl get hpa
```

```
[root@k8s-master01 ~]# kubectl get hpa -w
NAME        REFERENCE              TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
nginx-hpa   Deployment/nginx-hpa   17%/60%   1         10        1          18m
```

5、VPA 实现 Pod 自动扩缩容

Vertical Pod Autoscaler（VPA）：垂直 Pod 自动扩缩容，用户无需为其 pods 中的容器设置最新的资源 request。配置后，它将根据使用情况自动设置 request，从而允许在节点上进行适当的调度，以便为每个 pod 提供适当的资源量。

VPA 有以下四种更新策略：

- Initial：仅在 Pod 创建时修改资源请求，以后都不再修改。
- Auto：默认策略，在 Pod 创建时修改资源请求，并且在 Pod 更新时也会修改。
- Recreate：类似 Auto，在 Pod 的创建和更新时都会修改资源请求，不同的是，只要 Pod 中的请求值与新的推荐值不同，VPA 都会驱逐该 Pod，然后使用新的推荐值重新启一个。因此，一般不使用该策略，而是使用 Auto，除非你真的需要保证请求值是最新的推荐值。
- Off：不改变 Pod 的资源请求，不过仍然会在 VPA 中设置资源的推荐值。

5.1、安装 vpa

Github 地址：https://github.com/kubernetes/autoscaler

1、下载并上传软件到 k8s-master01 节点：

```
[root@k8s-master01 ~]# ll autoscaler-vertical-pod-autoscaler-0.12.0.tar.gz
-rw-r--r-- 1 root root 67944702 Jan  4 02:02 autoscaler-vertical-pod-autoscaler-0.12.0.tar.gz
```

2、解压

```
[root@k8s-master01 ~]# tar xf autoscaler-vertical-pod-autoscaler-0.12.0.tar.gz
```

3、修改 yaml 文件

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    [root@k8s-master01 ~]# cd autoscaler-vertical-pod-autoscaler-0.12.0/vertical-pod-autoscaler/deploy/

    # 修改 admission-controller-deployment.yaml 文件的 image 镜像
    [root@k8s-master01 deploy]# vi admission-controller-deployment.yaml
                image: giantswarm/vpa-admission-controller:0.12.0
                imagePullPolicy: IfNotPresent

    # 修改 recommender-deployment.yaml 文件的 image 镜像
    [root@k8s-master01 deploy]# vi recommender-deployment.yaml
            image: giantswarm/vpa-recommender:0.12.0
            imagePullPolicy: IfNotPresent

    # 修改 updater-deployment.yaml 文件的 image 镜像
    [root@k8s-master01 deploy]# vi updater-deployment.yaml
                image: giantswarm/vpa-updater:0.12.0
                imagePullPolicy: IfNotPresent
```

4、安装 vpa

```
    [root@k8s-master01 ~]# cd autoscaler-vertical-pod-autoscaler-0.12.0/vertical-pod-autoscaler/hack/
    [root@k8s-master01 hack]# ./vpa-up.sh
  -nameopt arg    - various certificate name options
  -reqopt arg     - various request text options

ERROR: Failed to create CA certificate for self-signing. If the error is "unknown option -addext", update your openssl version
  or deploy VPA from the vpa-release-0.8 branch.
deployment.apps/vpa-admission-controller created
service/vpa-webhook created
[root@k8s-master01 hack]#
```

报了一个错误：

ERROR: Failed to create CA certificate for self-signing. If the error is "unknown option -addext", update your openssl version or deploy VPA from the vpa-release-0.8 branch.

意思大概为：错误：无法为自签名创建 CA 证书。错误为“未知选项-addext”，请更新 openssl 版本或从 VPA-release-0.8 分支部署 VPA。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
clusterrolebinding.rbac.authorization.k8s.io/system:vpa-status-reader-binding created
serviceaccount/vpa-updater created
deployment.apps/vpa-updater created
serviceaccount/vpa-recommender created
deployment.apps/vpa-recommender created
Generating certs for the VPA Admission Controller in /tmp/vpa-certs.
Generating RSA private key, 2048 bit long modulus
.....................................................................................+++
.........................+++
e is 65537 (0x10001)
unknown option -addext
req [options] <infile >outfile
where options  are
 -inform arg    input format - DER or PEM
 -outform arg   output format - DER or PEM
 -in arg        input file
 -out arg       output file
 -text          text form of request
 -pubkey        output public key
 -noout         do not output REQ
 -verify        verify signature on REQ
 -modulus       RSA modulus
 -nodes         don't encrypt the output key
 -engine e      use engine e, possibly a hardware device
 -subject       output the request's subject
 -passin        private key password source
 -key file      use the private key contained in file
 -keyform arg   key file format
 -keyout arg    file to send the key to
 -rand file:file:...
                load the file (or the files in the directory) into
                the random number generator
 -newkey rsa:bits generate a new RSA key of 'bits' in size
 -newkey dsa:file generate a new DSA key, parameters taken from CA in 'file'
 -newkey ec:file generate a new EC key, parameters taken from CA in 'file'
 -[digest]      Digest to sign with (see openssl dgst -h for list)
```

在上面确实发现提示 unknown option -addext。

安装 openssl 版本，首先要确保本机没有 openssl，若存在 openssl，请先卸载。

卸载本机的 openssl：

```
[root@k8s-master01 hack]# yum -y remove openssl
```

Openssl 下载地址：https://github.com/openssl/openssl/tags

#将下载好的 openssl 上传到服务器

```
[root@k8s-master01 ~]# ll openssl-openssl-3.0.7.tar.gz
```

```
[root@k8s-master01 ~]# ll openssl-openssl-3.0.7.tar.gz
-rw-r--r-- 1 root root 15255577 Jan  4 02:28 openssl-openssl-3.0.7.tar.gz
```

```
[root@k8s-master01 ~]# tar xf openssl-OpenSSL_1_1_1s.tar.gz
[root@k8s-master01 ~]# cd openssl-OpenSSL_1_1_1s
[root@k8s-master01 openssl-OpenSSL_1_1_1s]# ./config
--prefix=/usr/local/openssl
[root@k8s-master01 openssl-OpenSSL_1_1_1s]# make && make install
[root@k8s-master01 ~]# ln -s /usr/local/openssl/bin/openssl
/usr/local/bin/openssl

# 加载一下 lib 库
[root@k8s-master01 hack]# ln -s /usr/local/openssl/include/openssl
/usr/include/openssl/
[root@k8s-master01 hack]# echo "/usr/local/openssl/lib/" >
/etc/ld.so.conf
[root@k8s-master01 hack]# ldconfig -v

# 查看 openssl 版本
[root@k8s-master01 ~]# openssl version
```

```
    OpenSSL 1.1.1s  1 Nov 2022

    # 重新部署 vpa
    [root@k8s-master01 ~]# cd
autoscaler-vertical-pod-autoscaler-0.12.0/vertical-pod-autoscaler/hac
k/
    [root@k8s-master01 hack]# ./vpa-down.sh
    [root@k8s-master01 hack]# ./vpa-up.sh

    # 验证 vpa 是否部署成功
    [root@k8s-master01 ~]# kubectl get pods -n kube-system | grep vpa
[root@k8s-master01 ~]# kubectl get pods -n kube-system | grep vpa
vpa-admission-controller-6bcf678c99-trjtd   1/1     Running   0          58s
vpa-recommender-5bf7b746b4-px2cf            1/1     Running   0          58s
vpa-updater-66c84d8bf9-gvjk9                1/1     Running   0          57s
    部署完成
```

5.2、测试 VPA 实现 pod 自动扩缩容

1、部署一个 nginx 服务,部署到 namespace: vpa 名称空间下

```
    [root@k8s-master01 ~]# kubectl create ns vpa
    namespace/vpa created

    [root@k8s-master01 ~]# vi vpa-nginx.yaml
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      labels:
        app: nginx
      name: nginx
      namespace: vpa
    spec:
      replicas: 2
      selector:
        matchLabels:
          app: nginx
      template:
        metadata:
          labels:
            app: nginx
        spec:
          containers:
          - image: nginx
```

```
            imagePullPolicy: IfNotPresent
            name: nginx
            resources:
              requests:
                cpu: 10m
                memory: 100Mi
```

创建并查看 pod

```
[root@k8s-master01 vpa]# kubectl apply -f vpa-nginx.yaml
deployment.apps/nginx created

[root@k8s-master01 ~]# kubectl get pods -n vpa
NAME                     READY   STATUS    RESTARTS   AGE
nginx-6b5f57674f-8rbkf   1/1     Running   0          72s
nginx-6b5f57674f-lcwqf   1/1     Running   0          72s
```

查看当前 pods requests

```
[root@k8s-master01 ~]# kubectl describe pods nginx-6b5f57674f-8rbkf
-n vpa | egrep "Requests|cpu|memory"
        Requests:
          cpu:        10m
          memory:     100Mi

[root@k8s-master01 ~]# kubectl describe pods nginx-6b5f57674f-lcwqf
-n vpa | egrep "Requests|cpu|memory"
        Requests:
          cpu:        10m
          memory:     100Mi
```

可以看到现在至少需要 10m cpu 和 100Mi 内存

2、在 nginx 管理的 pod 前端创建四层代理 Service

```
[root@k8s-master01 vpa]# vi vpa-nginx-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: vpa
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
```

```
    selector:
      app: nginx

[root@k8s-master01 vpa]# kubectl apply -f vpa-nginx-service.yaml
service/nginx created

[root@k8s-master01 vpa]# kubectl get svc -n vpa
NAME    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)      AGE
nginx   NodePort   10.100.41.138   <none>        80:32461/TCP   5s

[root@k8s-master01 vpa]# curl -I 192.168.43.201:32461
HTTP/1.1 200 OK
Server: nginx/1.21.5
Date: Sat, 22 Jan 2022 13:41:27 GMT
Content-Type: text/html
Content-Length: 615
Last-Modified: Tue, 28 Dec 2021 15:28:38 GMT
Connection: keep-alive
ETag: "61cb2d26-267"
Accept-Ranges: bytes
```

3、创建 VPA

使用 updateMode: "Auto"模式，这种模式可以获取资源推荐值，也会动态更新 Pod。

updateMode:"Off"模式,这种模式可以获取资源推荐值,不会动态更新 Pod。

```
[root@k8s-master01 ~]# vi vpa-nginx-1.yaml
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: nginx-vpa
  namespace: vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: nginx
  updatePolicy:
    updateMode: "Auto"
  resourcePolicy:
    containerPolicies:
    - containerName: "nginx"
      minAllowed:
```

```
            cpu: "10m"
            memory: "100Mi"
          maxAllowed:
            cpu: "30m"
            memory: "2600Mi"

yaml 文件说明：
    apiVersion: autoscaling.k8s.io/v1
    kind: VerticalPodAutoscaler
    metadata:
      name: nginx-vpa
      namespace: vpa
    spec:
      targetRef:   #匹配资源
        apiVersion: "apps/v1"
        kind: Deployment
        name: nginx
      updatePolicy:
        updateMode: "Auto"
      resourcePolicy:
        containerPolicies:
        - containerName: "nginx"  #containers 容器名称，注意部署 pod 名称
          minAllowed:     #下限值
            cpu: "10m"
            memory: "100Mi"
          maxAllowed:     #上限值
            cpu: "30m"
            memory: "2600Mi"

    [root@k8s-master01 vpa]# kubectl apply -f vpa-nginx-1.yaml
    verticalpodautoscaler.autoscaling.k8s.io/nginx-vpa created

    [root@k8s-master01 test]#  kubectl get vpa -n vpa
    NAME        MODE   CPU   MEM       PROVIDED   AGE
    nginx-vpa   Auto   20m   262144k   True       3m22s

    查看 vpa 详细信息：
    [root@k8s-master01 test]# kubectl describe vpa nginx-vpa -n vpa
    Name:         nginx-vpa
    Namespace:    vpa
    Labels:       <none>
```

```
  Annotations:  <none>
  API Version:  autoscaling.k8s.io/v1
  Kind:         VerticalPodAutoscaler
  Metadata:
    Creation Timestamp:  2023-01-04T09:34:09Z
    Generation:          5
    Managed Fields:
      API Version:  autoscaling.k8s.io/v1beta2
      Fields Type:  FieldsV1
      fieldsV1:
        f:metadata:
          f:annotations:
            .:
            f:kubectl.kubernetes.io/last-applied-configuration:
        f:spec:
          .:
          f:resourcePolicy:
            .:
            f:containerPolicies:
          f:targetRef:
            .:
            f:apiVersion:
            f:kind:
            f:name:
          f:updatePolicy:
            .:
            f:updateMode:
      Manager:      kubectl-client-side-apply
      Operation:    Update
      Time:         2023-01-04T09:34:09Z
      API Version:  autoscaling.k8s.io/v1
      Fields Type:  FieldsV1
      fieldsV1:
        f:status:
          .:
          f:conditions:
          f:recommendation:
            .:
            f:containerRecommendations:
      Manager:         recommender
      Operation:       Update
      Time:            2023-01-04T09:34:24Z
```

```
    Resource Version:  5933
    UID:               f9451838-123a-49f3-a177-34f08796eeab
  Spec:
    Resource Policy:
      Container Policies:
        Container Name:  nginx
        Max Allowed:
          Cpu:     20m
          Memory:  2600Mi
        Min Allowed:
          Cpu:     10m
          Memory:  100Mi
    Target Ref:
      API Version:  apps/v1
      Kind:         Deployment
      Name:         nginx
    Update Policy:
      Update Mode:  Auto
  Status:
    Conditions:
      Last Transition Time:  2023-01-04T09:34:23Z
      Status:                True
      Type:                  RecommendationProvided
    Recommendation:
      Container Recommendations:
        Container Name:  nginx
        Lower Bound:
          Cpu:     20m
          Memory:  262144k
        Target:
          Cpu:     20m
          Memory:  262144k
        Uncapped Target:
          Cpu:     109m
          Memory:  262144k
        Upper Bound:
          Cpu:     20m
          Memory:  842269230
  Events:          <none>

  Lower Bound：下限值
  Target：推荐值
```

Upper Bound：上限值
Uncapped Target：如果 minAllowed 和 maxAllowed 都未设置的情况下建议的目标值。

上面结果表示，推荐的 Pod 的 CPU 请求为 20m，推荐的内存请求为 262144k 字节。

4、压测 nginx cpu

```
[root@k8s-master01 ~]# kubectl run test -it --image=busybox /bin/sh
/ # while true; do wget -q -O- http://nginx.vpa.svc.cluster.local; done
```

当 CPU 大于 20m 时，就会执行 pod 驱逐，重新部署新的足够资源的服务。

5、来看下 event 事件

```
[root@k8s-master01 vpa]# kubectl get event -n vpa
```

```
[root@k8s-master01 ~]# kubectl get event -n vpa
LAST SEEN   TYPE     REASON              OBJECT                          MESSAGE
22s         Normal   Scheduled           pod/nginx-6b5f57674f-27zjb      Successfully assigned vpa/nginx-6b5f57674f-27zjb to k8s-node01
21s         Normal   Pulled              pod/nginx-6b5f57674f-27zjb      Container image "nginx" already present on machine
21s         Normal   Created             pod/nginx-6b5f57674f-27zjb      Created container nginx
21s         Normal   Started             pod/nginx-6b5f57674f-27zjb      Started container nginx
5m28s       Normal   Scheduled           pod/nginx-6b5f57674f-8rbkf      Successfully assigned vpa/nginx-6b5f57674f-8rbkf to k8s-node01
5m28s       Normal   Pulled              pod/nginx-6b5f57674f-8rbkf      Container image "nginx" already present on machine
5m28s       Normal   Created             pod/nginx-6b5f57674f-8rbkf      Created container nginx
5m28s       Normal   Started             pod/nginx-6b5f57674f-8rbkf      Started container nginx
22s         Normal   Killing             pod/nginx-6b5f57674f-8rbkf      Stopping container nginx
22s         Normal   EvictedByVPA        pod/nginx-6b5f57674f-8rbkf      Pod was evicted by VPA Updater to apply resource recommendation.
5m28s       Normal   Scheduled           pod/nginx-6b5f57674f-lcwqf      Successfully assigned vpa/nginx-6b5f57674f-lcwqf to k8s-node02
5m29s       Normal   Pulled              pod/nginx-6b5f57674f-lcwqf      Container image "nginx" already present on machine
5m28s       Normal   Created             pod/nginx-6b5f57674f-lcwqf      Created container nginx
5m28s       Normal   Started             pod/nginx-6b5f57674f-lcwqf      Started container nginx
83s         Normal   Killing             pod/nginx-6b5f57674f-lcwqf      Stopping container nginx
82s         Normal   EvictedByVPA        pod/nginx-6b5f57674f-lcwqf      Pod was evicted by VPA Updater to apply resource recommendation.
82s         Normal   Scheduled           pod/nginx-6b5f57674f-t7hqp      Successfully assigned vpa/nginx-6b5f57674f-t7hqp to k8s-node02
82s         Normal   Pulled              pod/nginx-6b5f57674f-t7hqp      Container image "nginx" already present on machine
82s         Normal   Created             pod/nginx-6b5f57674f-t7hqp      Created container nginx
82s         Normal   Started             pod/nginx-6b5f57674f-t7hqp      Started container nginx
5m29s       Normal   SuccessfulCreate    replicaset/nginx-6b5f57674f     Created pod: nginx-6b5f57674f-8rbkf
5m29s       Normal   SuccessfulCreate    replicaset/nginx-6b5f57674f     Created pod: nginx-6b5f57674f-lcwqf
82s         Normal   SuccessfulCreate    replicaset/nginx-6b5f57674f     Created pod: nginx-6b5f57674f-t7hqp
22s         Normal   SuccessfulCreate    replicaset/nginx-6b5f57674f     Created pod: nginx-6b5f57674f-27zjb
5m29s       Normal   ScalingReplicaSet   deployment/nginx                Scaled up replica set nginx-6b5f57674f to 2
```

从输出信息可以看到，vpa 执行了 EvictedByVPA，自动停掉了 nginx，然后使用 VPA 推荐的资源启动了新的 nginx

我们查看下 nginx 的 pod 可以得到确认

```
[root@k8s-master01 ~]# kubectl get pods -n vpa
NAME                     READY   STATUS    RESTARTS   AGE
nginx-6b5f57674f-27zjb   1/1     Running   0          46s
nginx-6b5f57674f-t7hqp   1/1     Running   0          106s
```

6、再次查看 pods requests

```
[root@k8s-master01 ~]# kubectl describe pods nginx-6b5f57674f-27zjb -n vpa | egrep "Requests|cpu|memory"
        Requests:
          cpu:        20m
```

```
          memory:     262144k

    [root@k8s-master01 ~]# kubectl describe pods nginx-6b5f57674f-t7hqp
-n vpa | egrep "Requests|cpu|memory"
        Requests:
          cpu:        20m
          memory:     262144k
```

可以看到现在至少需要 20m cpu 和 262144k 内存，这个值其实就是 vpa 的 Target 推荐值。

7、说明

现在可以知道 VPA 做了哪些事了吧。当然，随着服务的负载的变化，VPA 的推荐值也会不断变化。当目前运行的 pod 的资源达不到 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 使用限制：

1、不能与 HPA（Horizontal Pod Autoscaler）一起使用

2、Pod 比如使用副本控制器，例如属于 Deployment 或者 StatefulSet

VPA 有啥好处：

1、Pod 资源用其所需，所以集群节点使用效率高。

2、Pod 会被安排到具有适当可用资源的节点上。

3、不必运行基准测试任务来确定 CPU 和内存请求的合适值。

4、VPA 可以随时调整 CPU 和内存请求，无需人为操作，因此可以减少维护时间。