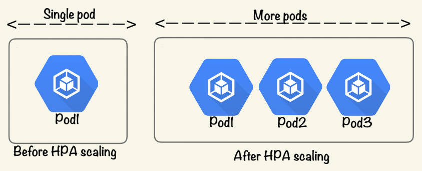
Autoscaling

Autoscaling refers to the ability of the Kubernetes cluster to automatically adjust the number of running instances of a specific workload or application based on the current demand or load. Autoscaling helps to ensure that there are enough resources available to handle the workload while also optimizing resource utilization.
Kubernetes provides two types of autoscaling mechanisms:



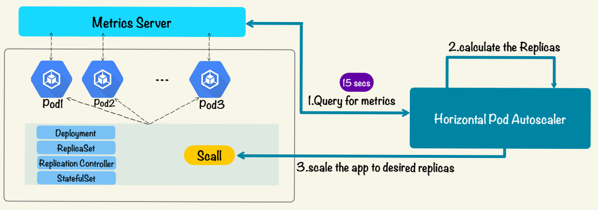
Horizontal Pod Autoscaling (HPA) allows you to automatically scale out your application by adding or removing replicas based on resource utilization metrics such as CPU utilization or custom metrics. This ensures that you have the necessary resources to handle increased traffic or load without over-provisioning resources and incurring additional costs

Scaling out, also known as horizontal scaling, is the process of adding more replicas of a Deployment or ReplicaSet to handle an increase in traffic or load

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: php-apache
  namespace: dev
spec:
  scaleTargetRef:
    kind: Deployment
    name: php-apache
    apiVersion: apps/v1
  minReplicas: 3
  maxReplicas: 20
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
    - type: Resource
      resource:
        name: memory
        target:
          type: Utilization
          averageUtilization: 40
  behavior:
    scaleUp:
      policies:
        - type: Pods
          value: 5
          periodSeconds: 30
        - type: Percent
          value: 100
          periodSeconds: 30
      selectPolicy: Max
      stabilizationWindowSeconds: 40
    scaleDown:
      policies:
        - type: Pods
          value: 4
          periodSeconds: 10
        - type: Percent
          value: 10
          periodSeconds: 10
      selectPolicy: Min
      stabilizationWindowSeconds: 5
```

This specifies the target Deployment for the HPA

minimum and maximum number of replicas for the Deployment



HPA uses the metrics server to collect the metrics data and then uses the scaling algorithm to calculate the new number of replicas needed based on the current load

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
  namespace: dev
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
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
----
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  namespace: dev
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
```

The metrics section defines the metrics that the HPA uses to scale the Deployment. In this case, two metrics are specified: CPU utilization and memory utilization. For each metric, the HPA calculates the average utilization across all pods over a certain period of time and compares it to the target utilization. If the actual utilization exceeds the target utilization, the HPA increases the number of replicas. If the actual utilization falls below the target utilization, the HPA decreases the number of replicas. By using multiple metrics, the HPA can make more informed scaling decisions

The behavior section defines the scaling behavior for the HPA. In this case, the HPA uses two policies for scaling up and two policies for scaling down. The Pods policy specifies the number of replicas to add or remove, while the Percent policy specifies the percentage of replicas to add or remove. By using both policies, the HPA can scale up or down more quickly or slowly, depending on the workload. The selectPolicy field specifies how the HPA should choose between the Pods and Percent policies. In this case, it's set to Max, which means that the highest value of the two policies will be used for scaling up, and the lowest value will be used for scaling down. The stabilizationWindowSeconds field specifies the number of seconds that the HPA should wait before it starts scaling again after a scaling event. This helps to prevent rapid scaling, which can cause instability in the cluster

The Pods policy specifies that the HPA should add 5 replicas every 30 seconds, while the Percent policy specifies that the HPA should add 100% replicas every 30 seconds

K get -n dev hpa

```
NAME         REFERENCE               TARGETS             MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   40%/40%,  20%/50%      3         20        3        1d
```

Max(5,3)
5pod
100%[3pod]

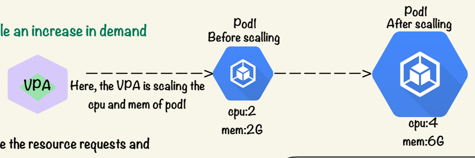
```
NAME         REFERENCE               TARGETS             MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   30%/40%,  15%/50%      3         20        8        1d
```

The Pods policy specifies that the HPA should remove 4 replicas every 10 seconds, while the Percent policy specifies that the HPA should remove 10% replicas every 10 seconds

HPA is designed to automatically scale the number of replicas of a deployment or a replica set based on observed CPU utilization, memory utilization, or custom metrics. This makes it well-suited for stateless workloads that can be easily scaled horizontally by adding more replicas

Vertical Pod Autoscaling (VPA) allows you to automatically scale up or down the resource requests and limits of containersin a Pod based on actual resource usage. This ensures that each Pod has the necessary resources to handle the workload efficiently without wasting resources

Scaling up, also known as vertical scaling, is the process of increasing the resources available to each replica of a Deployment or ReplicaSet to handle an increase in demand



```
apiVersion: "autoscaling.k8s.io/v1"
kind: VerticalPodAutoscaler
metadata:
  name: php-apache
  namespace: dev
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: php-apache
  updatePolicy:
    updateMode: "Auto"
  resourcePolicy:
    containerPolicies:
      - containerName: '*'
        mode: "Auto"
        controlledValues: "RequestsAndLimits"
        minAllowed:
          cpu: 10m
          memory: 5Mi
        maxAllowed:
          cpu: 200m
          memory: 500Mi
        controlledResources: ["cpu", "memory"]
```

there are two ways to trigger VPA in k8s: automatic and manual,you can set the updatePolicy field to Auto for automatic scaling or Off for manual scaling

1 Because of the updateMode field in is set to "Auto", the VPA Updater component will automatically update the resource requests and limits of the containers in the pods.

targetRef: The reference to the target workload object that the VPA should adjust. In this case, it's a deployment with the name php-apache

updatePolicy determines how frequently the pod resource requests and limits should be updated. In this case, it's set to "Auto", which means the VPA will automatically update the resource requests and limits based on the pod's usage

resourcePolicy defines the resource requests and limits for the containers in the target workload. In this case, there is one container policy defined

containerName: The name of the container to apply the policy to. In this case, it's set to *, which means the policy applies to all containers in the target workload

mode determines how the resource requests and limits are set. In this case, it's set to "Auto", which means the VPA will automatically adjust the resource requests and limits based on the pod's usage.

controlledValues: The values that the VPA is allowed to set for the resource requests and limits. In this case, it's set to "RequestsAndLimits", which means the VPA can adjust both the resource requests and limits.

The minimum resource request and limit values allowed for the container are set to 10 milliCPU and 5 MiB of memory. The maximum resource request and limit values allowed for the container are set to 200 milliCPU and 500 MiB of memory

controlledResources: The resources that the VPA is allowed to adjust. In this case, it's set to both CPU and memory.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
  namespace: dev
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
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: “20m”
            memory: “200Mi”
          limits:
            cpu: “500m”
            memory: “1Gi”
```

```
apiVersion: "autoscaling.k8s.io/v1"
kind: VerticalPodAutoscaler
metadata:
  name: php-apache
  namespace: dev
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: php-apache
  updatePolicy:
    updateMode: "off"
```

2 Because of the updateMode field in is set to "Off", the VPA Updater component will not automatically update the resource requests and limits of the containers in the pods. In this case, you will need to manually update the resource requests and limits of the pods when necessary
To manually adjust the resource requests and limits, you can update the deployment or statefulset object that the VPA is targeting

kubectl describe vpa command can provide recommendations for the resource requests and limits of containers based on the resource usage metrics collected by the VPA controller

VPA is well-suited for stateful workloads

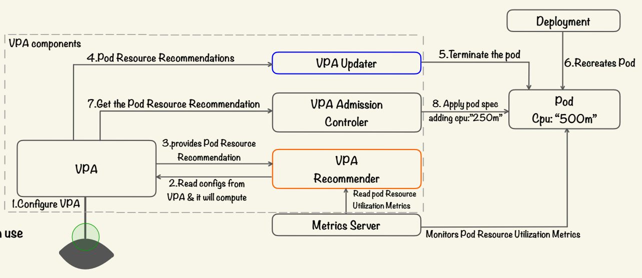
```
kubectl -n dev get vpa
NAME         MODE     CPU      MEM       PROVIDED    AGE
php-apache   off      163m     262144k   True        2m7s
```

```
kubectl describe vpa -n kube-system
...
container Recommendations.
  Container Name: php-apache
  Lower Bound:
    Cpu:      25m
    Memory: 262144k
  Target:
    Cpu:      163m
    Memory: 262144k
  Upper Bound:
     Cpu:      10173m
     Memory: 2770366988
```

Lower Bound: The minimum amount of CPU and memory that the container should have to meet the resource utilization targets

Target: The target amount of CPU and memory that the container should have to achieve the desired resource utilization levels

Upper Bound: The maximum amount of CPU and memory that the container can use


Deployment
VPA components
4.Pod Resource Recommendations
VPA Updater
5.Terminate the pod
6.Recreates Pod
7.Get the Pod Resource Recommendation
VPA Admission Controler
8. Apply pod spec adding cpu:"250m"
Pod
Cpu: “500m”
3.provides Pod Resource Recommendation
VPA
VPA Recommender
2.Read configs from VPA & it will compute
Read pod Resource Utilization Metrics
1.Configure VPA
Metrics Server
Monitors Pod Resource Utilization Metrics