**Vertical Pod Autoscaling (VPA)**

**What is VPA?**

Vertical Pod Autoscaling (VPA) adjusts the CPU and memory resource requests and limits for your pods. Instead of adding more pods, VPA modifies the resources allocated to existing pods.

**How Does VPA Work?**

VPA operates by:

1. **Monitoring Resource Usage:** VPA continuously monitors the CPU and memory usage of the pods.
2. **Recommending Resources:** Based on the usage patterns, VPA recommends resource adjustments.
3. **Applying Changes:** The VPA can automatically apply the recommended changes to the pod specifications, typically requiring a pod restart.

**Advantages of VPA**

* **Resource Optimization:** Ensures pods have the right amount of resources, preventing over-provisioning or under-provisioning.
* **Simplified Resource Management:** Reduces the need to manually adjust resource requests and limits.
* **Improved Performance:** Helps maintain application performance by dynamically adjusting resources based on actual usage.

**How VPA Adjusts Resources**

1. **Initial Adjustment:**When the VPA first starts managing a pod, it ensures that the pod’s resource requests are within the specified minAllowed and maxAllowed ranges. In your case, the minAllowed CPU is set to 100m, so the VPA initially adjusts the pod's CPU request to at least 100m to ensure it has a minimum baseline of resources.
2. O**ngoing Monitoring and Adjustment:** After the initial adjustment, the VPA continues to monitor the CPU and memory usage of the pods. It collects data over time to understand the resource utilization patterns and make more informed recommendations.
3. **Recommendation Based on Observed Usage:** The VPA makes recommendations based on observed usage. It doesn’t immediately jump to the maximum allowed (maxAllowed) value unless the observed usage justifies it. Instead, it incrementally adjusts the resource requests to align closely with actual usage patterns.

**Practical Example of VPA**

Let’s deploy an example using VPA.

**Step 1: Install VPA**

Before creating VPA objects first we need to install VPA in our cluster using the below steps:

Clone the VPA Source Code: Use Git to clone the VPA source code repository to your local machine. Run the following command:

git clone https://github.com/kubernetes/autoscaler.git

Navigate to the VPA Directory: Change your current directory to the autoscaler directory, which contains the VPA source code:

cd autoscaler

Run the Installation Script: Inside the autoscaler directory, run the installation script vpa-up.sh. This script sets up the necessary components for VPA in your Kubernetes cluster:

./hack/vpa-up.sh

./pkg/admission-controller/gencerts.sh

This script will install the Vertical Pod Autoscaler components, including custom resources, controllers, and other required resources, into your Kubernetes cluster.

Verify Installation: Once the installation script has completed successfully, verify that the VPA components are installed and running in your cluster:

kubectl get pods -n kube-system | grep vpa

You should see pods related to Vertical Pod Autoscaler running in the kube-system namespace.

**Step 2: Create a Deployment**

Create a file named vpa-deployment.yaml with the following content:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: high-cpu-utilization-deployment  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: cpu-utilization-app  
 template:  
 metadata:  
 labels:  
 app: cpu-utilization-app  
 spec:  
 containers:  
 - name: cpu-utilization-container  
 image: ubuntu  
 command: ["/bin/sh", "-c", "apt-get update && apt-get install -y stress-ng && while true; do stress-ng --cpu 1; done"]  
 resources:  
 limits:  
 cpu: "0.05"  
 requests:  
 cpu: "0.05"

This deployment repeatedly runs a CPU stress test using the `stress-ng` tool, consuming a small but continuous amount of CPU (limited to 0.05 cores) to simulate high CPU utilization.

Apply the deployment:

$ kubectl apply -f vpa-deployment.yaml

**Step 3: Create a VPA**

Create a file named vpa.yaml with the following content:

apiVersion: "autoscaling.k8s.io/v1"  
kind: VerticalPodAutoscaler  
metadata:  
 name: stress-vpa  
spec:  
 targetRef:  
 apiVersion: "apps/v1"  
 kind: Deployment  
 name: high-cpu-utilization-deployment  
 updatePolicy:  
 updateMode: Auto  
 resourcePolicy:  
 containerPolicies:  
 - containerName: '\*'  
 minAllowed:  
 cpu: 100m  
 memory: 50Mi  
 maxAllowed:  
 cpu: 200m #maximum vpa will be allocating this many cpus even if demand is higher.  
 memory: 500Mi  
 controlledResources: ["cpu", "memory"]

There are multiple valid options for updateMode in VPA. They are:

* **Off**— VPA will only provide the recommendations, and it will not automatically change resource requirements.
* **Initial**— VPA only assigns resource requests on pod creation and never changes them later.
* **Recreate**— VPA assigns resource requests on pod creation time and updates them on existing pods by evicting and recreating them.
* **Auto mode**— It recreates the pod based on the recommendation.

We increased the CPU metrics in the above demo and then manually applied the changes to scale the pod. We can do this automatically by using the **updateMode: “Auto”** parameter.

This Vertical Pod Autoscaler (VPA) automatically adjusts the CPU and memory requests and limits for the `high-cpu-utilization-deployment` to ensure efficient resource usage, within specified bounds (100m to 200m CPU and 50Mi to 500Mi memory). It targets all containers in the deployment for resource management.

Apply the VPA:

$ kubectl apply -f vpa.yaml

**Step 4: Testing**

First, let’s check the initial CPU utilization of the target pods:

$ kubect top po  
NAME CPU(cores) MEMORY(bytes)   
high-cpu-utilization-deployment-78cc948dfb-fqbq9 50m 13Mi   
high-cpu-utilization-deployment-78cc948dfb-qtbt8 50m 9Mi

Verify the CPU requests for the pods:

$ kubectl get po -o jsonpath='{.items[\*].spec.containers[\*].resources.requests.cpu}'  
50m 50m

As seen, the two target pods are consuming their maximum CPU request since we are generating CPU load on them.

Next, check the VPA status:

$ kubectl get vpa   
NAME MODE CPU MEM PROVIDED AGE  
stress-vpa Auto 100m 262144k True 2m5s

Wait for some time and then check the pod status again:

$ kubectl get po   
NAME READY STATUS RESTARTS AGE  
high-cpu-utilization-deployment-78cc948dfb-wvfkm 1/1 Running 0 66s  
high-cpu-utilization-deployment-78cc948dfb-z7f87 1/1 Running 0 2m6s

Check the updated CPU requests:

$ kubectl get po -o jsonpath='{.items[\*].spec.containers[\*].resources.requests.cpu}'  
100m 100m%

Monitor the updated CPU utilization:

$ kubectl top po   
NAME CPU(cores) MEMORY(bytes)   
high-cpu-utilization-deployment-78cc948dfb-wvfkm 101m 47Mi   
high-cpu-utilization-deployment-78cc948dfb-z7f87 100m 49Mi

As observed, the pods were restarted and their CPU request values increased from 50m to 100m due to the VPA’s minAllowed setting. The pods will continue generating load until they reach their limits.

The VPA will then increase the CPU request values to the recommended value of i.e., max of 200m as needed.

$ kubectl get po -o jsonpath='{.items[\*].spec.containers[\*].resources.requests.cpu}'  
126m 126m  
  
$ kubectl top po   
NAME CPU(cores) MEMORY(bytes)   
high-cpu-utilization-deployment-78cc948dfb-7pqln 127m 48Mi   
high-cpu-utilization-deployment-78cc948dfb-8tn82 127m 47Mi

**Why the CPU Request Increased to 126m Instead of 200m**

* **Observed Usage:** The VPA’s recommendation of 126m for the CPU is based on the actual usage observed from the container. The VPA aims to provide just enough resources to meet the demand without over-provisioning. The 126m recommendation suggests that the current usage pattern of the container indicates that 126m is sufficient for its needs at this time.
* **Incremental Adjustments:** The VPA makes incremental adjustments to avoid sudden large changes that might disrupt the application. It gradually increases the resource requests based on the monitored usage, ensuring stability and efficiency.
* **Efficiency and Limits:** While the maxAllowed is set to 200m, this value is a ceiling rather than a target. The VPA only increases the CPU request to 200m if the observed usage indicates that such an increase is necessary. In above case, the observed usage suggested that 126m is adequate, so the VPA set the request to this value.

After some time:

$ kubectl get po -o jsonpath='{.items[\*].spec.containers[\*].resources.requests.cpu}'  
163m 163m

With this setup, the VPA will automatically adjust the CPU and memory requests for the high-cpu-utilization-deployment deployment, aiming to keep the resource usage within the specified limits.

**Exclude scaling for a specific container**

To exclude scaling for a specific container within a pod managed by the VerticalPodAutoscaler (VPA), you can set the mode for that container to “Off”. This ensures that the VPA does not adjust the resource requests for that particular container. Here’s an example configuration:

apiVersion: autoscaling.k8s.io/v1  
kind: VerticalPodAutoscaler  
 .  
 .  
 .  
 resourcePolicy:  
 containerPolicies:  
 - containerName: <container\_name>  
 mode: "Off"

In this example:

* <container\_name> refers to the name of the container within the pod for which you want to exclude scaling.
* mode: "Off" specifies that the VPA should not apply any autoscaling actions to this particular container.

By configuring the VPA in this way, you can ensure that only specific containers within your pods are subject to autoscaling, while others remain unaffected. This flexibility allows you to fine-tune the scaling behavior according to the requirements of your workload.

**HPA vs. VPA: Which One to Use?**

Choosing between HPA and VPA depends on your application’s requirements:

* **HPA** is ideal for stateless applications that can easily scale horizontally by adding more pods. It’s useful when you need to handle variable workloads and ensure high availability.
* **VPA** is suitable for applications where scaling horizontally is not feasible or when you need to optimize resource usage for individual pods. It’s beneficial for stateful applications or those with varying resource needs.