134 Vertical Pod Autoscaling (VPA)

Scaling resources for a workload the manual way

Manually scaling workload vertically.

As an k8s administrator tasked to make sure that there is always sufficient workload to support demand for the application

nginx.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: nginx
        resources:
          requests:
            cpu: "250m"
          limits:
            cpu: "500m"
```

And from the deployment configuration. I see that this pod requests 250 millicores of CPU & has a limit of 500 millicores of CPU, this means that 500 millicores is the max capacity it gets

```
$ kubectl top pod my-app-pod
NAME          CPU (cores)   MEMORY (bytes)
my-app-pod    450m          350Mi
```

we would run 'kubectl top pod' command & monitor the resource consumption of the pod

Remember we need to have metrics server running on the cluster to be able to monitor, the same as before & when it reaches a specific threshold what I'm going to do to scale the pod vertically is to run the kubectl edit deployment command, we then run the kubectl edit deployment command & change the resource requests & limits assigned to the deployment under the pod template, under the Containers section & then save it

```
kubectl edit deployment my-app
```

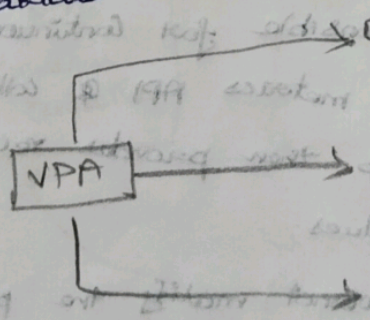
what would happen?

It would kill that pod & create a new pod

This is manual approach.

For automation we have Vertical Pod Autoscaler

Similar to the Horizontal Pod Autoscaler, the Vertical Pod Autoscaler will continuously monitors the metrics & then it automatically increases or decreases the resources assigned to the pods in a deployment & thus balances the workload



Look at it indetail

So unlike the Horizontal Pod Autoscaler, the Vertical Pod Autoscaler do not come builtin.

As such, we must deploy it, So, we first apply the Vertical Pod Autoscaler definition file available in the Github repo.

```
Kubectl apply -f https://github.com/kes/autoscaler/releases/latest/
                                download/vertical-pod-autoscaler.yaml
```

And then

```
Kubectl get pods -n kube-system | grep vpa
```

vpa-admission-Controller-xxx	Running
Vpa-Vecommender-xxxx	Running
vpa-updater-xxxx	Running

Search for vpa, we will be able to see that there are multiple components deployed, so there is the admission-controller recommender and the updater service which should be running lets look at what these are

So, the VPA ~~deployment~~ deployment consists of multiple components

* VPA admission controller,
* VPA updater,
* VPA Recommender

⇒ The Recommender is responsible for continuously monitoring resource usage from the K8S metrics API & collects historical and live usage data for pods & then provides recommendations on optimal CPU and memory values

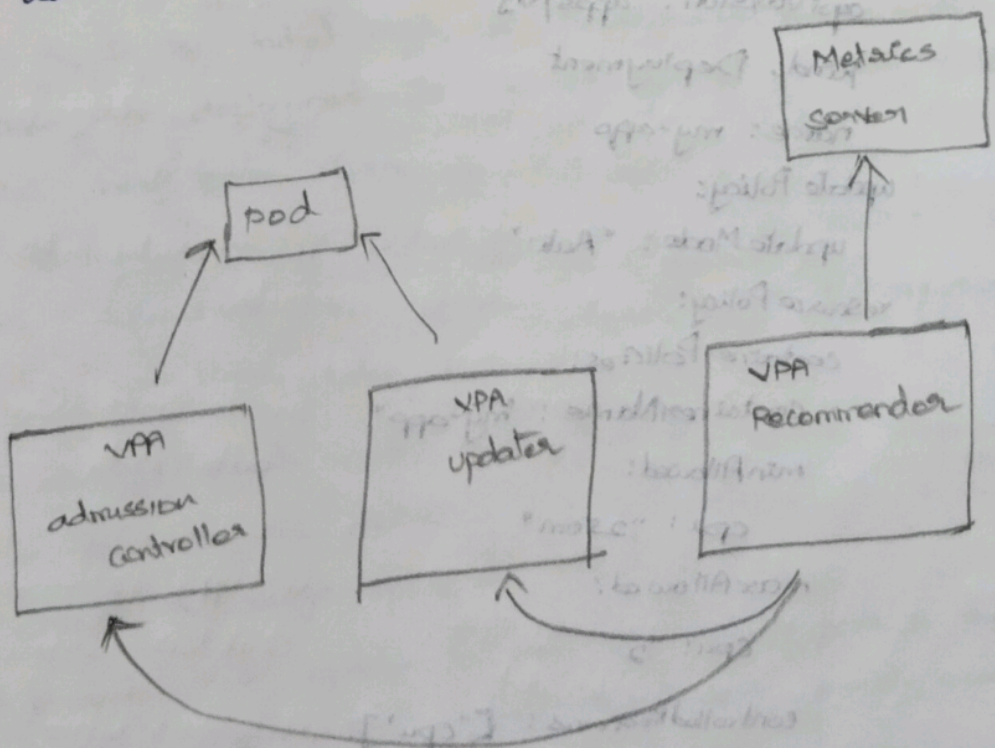
⇒ The Recommender itself doesnot modify the pod directly or it only suggests changes.

The Updater detects pods that are running with suboptimal resources & evicts them when an update is needed.

It gets the information from the Recommender & monitors the pod & if a pod needs to be updated, it evicts them. So that means it just terminates the pod and the admission controller intervene the pod creation process & uses the recommendations from the Recommender again & then mutate the pod spec to apply the recommended CPU & memory values at startup.

This ensures that the newly created pods start with the Coolest resource requests

Basically the VPA Recommender collects information, the updater monitors & get the information from the Recommender, Compares that to the actual pod, & if the pod is beyond the threshold, it kills the pod. and whether it kills or not that depends on the policy that we will talk about in a bit.



But, ideally it would kill the pod, & then the Admission Controller intervenes because when a pod is killed its automatically...

The deployment will automatically recreates the pod. And when it does that, the Admission Controller intervene & updates the resources so that the pod now comes up with new size

Create a VPA service using definition file

NOTE: there is no imperative command to create a VPA because its not a built-in component like the HPA

my-app-vpa.yaml

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app
  updatePolicy:
    updateMode: "Auto"
  resourcePolicy:
    containerPolicies:
    - containerName: "my-app"
      minAllowed:
        cpu: "250m"
      maxAllowed:
        cpu: "2"
      controlledResources: ["cpu"]
```

There are 4 modes that VPA operates in

Mode	Description
Off	Only recommends. Does not change anything
Initial	Only changes at Pod Creation. Not later
Recreate	Evicts pods if usage goes beyond range
Auto	Updates existing pods to recommended numbers. For now this behaves similar to Recreate. But when support for "Inplace update of Pod Resources" is available that mode will be preferred

⇒ The off mode, which only recommends changes but doesnot do anything. In this case, its only the recommender working. The "Updater" and the admission Controllers are not working in case of the off mode.

⇒ The initial mode, that only changes pods on creation, not later

⇒ So in the Initial mode, the Recommender recommends a change & when the deployment is scaled for some other reason and the new pods come up.

⇒ The admission Controller intervenes & changes the pod's resource definitions. The updater in this case does not work to kill as, you know, take down the pod.

⇒ In Case of Recreate mode, thats where the Updater component jumps in & evicts an existing pod, when its resource Consumption goes beyond a range specified

⇒ Auto mode, this updates to existing ~~numbers~~ pods to the recommended numbers. Again, for now, auto mode works exactly as the Recreate mode. But when the support for the in-place update of pod resources is available, then we discussed

in previous lecture then that mode would be proposed over recreate. So, "auto mode" is built for future

⇒ Where the inplace update of pod resources are available in stable version of K8S, that would be the default behaviour. But as of now, the Automode works exactly like Recreate, where if a pod goes beyond a range, it is killed.

⇒ Now, VPA will monitor resource usage & suggest adjustments, so we can check the recommendations with the

```
Kubectl describe vpa my-app-vpa
  Recommendations:
    Target:
      CPU: 1.5
```

you can see the recommendation. So VPA suggest increasing CPU to 1.5

When to use VPA / HPA

Feature	VPA Vertical Scaling	HPA Horizontal Scaling
Scaling Method	Increases CPU & memory of existing pod	Adds/removes pods based on loads
Pod Behaviour	Restarts pods to apply new resource values	Keeps existing pods running
Handles Traffic spikes	No, because scaling requires a pod restart	yes, instantly adds more pods
optimizes costs	prevents over-provisioning of CPU/MEMORY	Avoids unnecessary idle pods
Best for	stateful workloads, CPU/memory-heavy apps (DBs, ML workloads)	webapps, microservices, Stateless services.
Example usecase	Databases (MYSQL, PostgresSQL) JVM based apps, AI/ML workloads	web services (Nginx, API services) message queues, microservices.

Summary

⇒ VPA is about optimizing resource allocation for individual pods

⇒ HPA is about scaling no.of. pods dynamically based on demand

⇒ Hence choosing the right autoscaler depends on the workload type & how you need to scale your apps.

VPA quick notes

what is VPA

VPA automatically adjusts CPU and memory requests | limits for pods based on usage over time

* Helps avoid under-provisioning (OOM/crash) & over-provisioning (wasted resources)

* VPA adjusts pod size, not the no.of. pods.

what VPA adjusts

Resource Field	Adjusted by VPA	Notes
resources.requests	yes	Key input for scheduling
resources.limits	Sometimes	optional - may be configured
Replica Count	NO	use HPA for this

VPA modes

Mode	what it does
Off	Only shows recommendations; no change to running pods
Initial	Applies resources only when pod is created initially
Auto	updates pods anytime by evicting & restarting them

Internal VPA Components

Component	Role
Recommender	Watches usage, Calculates resource suggestions
Updater	Evicts pods if they fall outside recommended ranges
Admission Controllers	Injects recommended values into new pods

Typical VPA workflow

1) App runs with set requests / limit
2) Recommender observes resource usage
3) Recommendations appear in kubectl describe vpa
4) Updater (in Auto) evicts pods to apply new settings
5) New pods start with updated resources

VPA YAML Example

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: myapp-vpa
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  updatePolicy:
    updateMode: Auto    # or Initial, off
```

Common Problems + Fixes

Symptom	Likely Cause	Fix
No recommendations	metrics-server not running	Install metrics-server
Pods restart unexpectedly	VPA in Auto mode	Use Initial (or Off)
HPA + VPA conflict	Both control CPU requests	Use HPA for CPU & VPA for memory
Recommendations too high	No resource limits	Use maxAllowed field in VPA
short-lived pods, no data	Not enough runtime for learning	Use long-running apps

VPA vs HPA vs HPA + VPA

Feature	VPA	HPA	Both Together
Adjusts CPU	yes	yes	only one should handle it
Adjusts memory	yes	no	safe to combine
Restarts pods	yes	No	N/A
Adjusts pod count	No	yes	Combine with care.

Safe VPA + HPA Usage Strategy

⇒ Use HPA for scaling replicas (e.g. based on CPU)

⇒ Use VPA for memory tuning

⇒ Prevents conflicts

```
resourcePolicy:
  containerPolicies:
  - containerName: my-app
    controlledResources: ["memory"]  # not CPU
```

Key Commands

Task	Command
View VPA status	kubectl describe vpa <name>
Lists all VPAs	kubectl get VPA
Create VPA from yaml	kubectl apply -f vpa-yaml
View vpa component pods	kubectl get pods -n kube-system

Final summary

⇒ VPA makes pods smarter by tuning their CPU/memory

⇒ It learns from metrics server

⇒ It can restart pods (Auto) or just suggest off

⇒ Use with HPA carefully - avoid overlapping control of same resource.