**Types of Auto Scaling**

**1️ Horizontal Scaling (Scale-Out & Scale-In)**

Increases or decreases the **number of pods or containers** based on CPU, memory, or custom metrics.

Used when multiple instances are required for handling increased traffic.

**2️ Vertical Scaling (Scale-Up & Scale-Down)**

Increases or decreases the **CPU and memory resources** allocated to a single pod or container.

Used when a **single instance needs more resources** instead of adding more pods.

**3 Cluster Autoscaler**

Scales the **entire Kubernetes cluster** by adding or removing worker nodes.

Works with **cloud providers** like AWS, GCP, and Azure.

Which Auto Scaling Should You Use?

|  |  |
| --- | --- |
| **Scenario** | **Use** |
| Scale pods based on CPU/memory | HPA (Horizontal Pod Autoscaler) |
| Scale resources of a single pod | VPA (Vertical Pod Autoscaler) |
| Scale entire Kubernetes cluster | Cluster Autoscaler |
| Scale Docker services (manual) | Docker Service Scaling |

**Conclusion**

✅ **Use HPA** for handling **high traffic** with multiple pods.  
✅ **Use VPA** if a **single pod** needs **more CPU/memory**.  
✅ **Use Cluster Autoscaler** if new worker **nodes** are needed.  
✅ **For Docker, Kubernetes is preferred** for **auto-scaling**.

**Understanding Horizontal Pod Autoscaler (HPA) – K8S**

***Deployment.yaml / \*.yaml file for HPA configuration:***

***Eg: 1***

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: my-spring-app-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: my-spring-app

**minReplicas**: 1

**maxReplicas**: 10

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 50 # Scale up when CPU usage exceeds 50%

***Eg: 2***

apiVersion: autoscaling/v2

kind: **HorizontalPodAutoscaler**

metadata:

name: my-app-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: my-app-deployment # Replace with your deployment name

**minReplicas**: 2 # Minimum number of replicas

**maxReplicas**: 10 # Maximum number of replicas

behavior:

scaleUp:

policies:

- type: Percent

value: 50

periodSeconds: 60 # Scale up by 50% every 60 seconds

selectPolicy: Max

stabilizationWindowSeconds: 120 # Stabilization window for scale up

scaleDown:

policies:

- type: Percent

value: 25

periodSeconds: 60 # Scale down by 25% every 60 seconds

selectPolicy: Min

stabilizationWindowSeconds: 180 # Stabilization window for scale down

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 70 # Target CPU utilization percentage

- type: Resource

resource:

name: memory

target:

type: Utilization

averageUtilization: 80 # Target memory utilization percentage

- type: Pods

pods:

metric:

name: requests\_per\_second # Custom metric name. Requires a custom metrics API

target:

type: AverageValue

averageValue: 100 # Target average requests per second per pod

- type: Object

object:

metric:

name: queue\_length # External metric name. Requires an external metrics API

describedObject:

apiVersion: v1

kind: Service

name: my-queue-service # Service name for the queue

target:

type: Value

value: 500 # Target queue length  
  
**Explanation of Key Attributes:**

* **scaleTargetRef:**
  + Specifies the target deployment (or ReplicaSet, StatefulSet) that the HPA will scale.
* **minReplicas and maxReplicas:**
  + Sets the minimum and maximum number of Pod replicas.
* **behavior:**
  + Controls the scaling behavior, including scaling policies and stabilization windows.
  + **scaleUp and scaleDown:** Define the scaling policies for increasing and decreasing replicas.
  + **policies:** Defines the scaling steps.
  + **periodSeconds:** defines the time period for each scaling step.
  + **selectPolicy:** Defines how to choose between multiple policies.
  + **stabilizationWindowSeconds:** Prevents rapid fluctuations in the number of replicas.
* **metrics:**
  + Defines the metrics used for autoscaling.
  + **Resource:** Scales based on resource utilization (CPU, memory).
  + **Pods:** Scales based on custom metrics per pod (requires a custom metrics API).
  + **Object:** Scales based on external metrics (requires an external metrics API).
  + **averageUtilization:** Target average utilization percentage for resource metrics.
  + **averageValue:** Target average value for pod metrics.
  + **value:** Target value for object metrics.

**Important Notes:**

* **Replace Placeholders:** Update my-app-deployment, requests\_per\_second, my-queue-service, and the metric values with your actual values.
* **Metrics Server:** For CPU and memory autoscaling, you need the metrics-server installed in your Kubernetes cluster.
* **Custom and External Metrics:** For Pods and Object metrics, you'll need to configure a custom or external metrics API.
* **Testing:** Thoroughly test your HPA configuration under realistic load conditions to ensure it behaves as expected.
* **Resource Requests and Limits:** Ensure your Pods have appropriate resource requests and limits set.
* **Monitoring:** Monitor the HPA controller and the number of replicas to track autoscaling activity.
* **Adjust values:** The values provided in this yaml file are examples, and should be adjusted to fit your specific applications needs.

***Readiness Probes***

A readiness probe in Kubernetes is a mechanism that helps determine if a container is ready to receive traffic. If a container fails the readiness probe, Kubernetes will temporarily remove it from the list of available endpoints for a service, ensuring that requests are not sent to a potentially unhealthy pod.

***Key Aspects of Readiness Probes***

They help manage traffic routing by ensuring that only healthy containers receive requests.

If a container is not ready, it remains part of the pod but does not receive traffic from the service.

Readiness probes are useful for applications that need time to initialize before handling requests.

***Readiness Probe Parameters***

initialDelaySeconds: Delay before the first probe (default: 0).

periodSeconds: Time between probes (default: 10s).

timeoutSeconds: Timeout for each probe attempt (default: 1s).

successThreshold: Minimum number of consecutive successes required (default: 1).

failureThreshold: Number of failures before the container is marked unready (default: 3).

**Difference between Readiness and Liveness Probes**

| Feature | Readiness Probe | Liveness Probe |
| --- | --- | --- |
| Purpose | Determines if the container is ready to serve traffic | Determines if the container is still running |
| Effect | Unready containers are removed from service endpoints | Failed containers are restarted |
| Use Case | Apps with initialization delays or temporary issues | Detecting stuck or crashed containers |

***Use Case Example***

Imagine a Spring Boot application that loads configurations on startup. While loading, it should not receive traffic. A readiness probe can be used to ensure traffic is only sent after initialization is complete.

Would you like a Spring Boot-specific example?

***Example configuration in yaml file***  
*readinessProbe*:

httpGet:

path: /health

port: 8080

initialDelaySeconds: 5

periodSeconds: 10