**Slide 1: Title Slide** Title: Kubernetes Horizontal Pod Autoscaler (HPA) - Overview & Essentials Subtitle: Key Concepts and Implementation Notes

**Slide 2: What is HPA?**

* Automatically scales number of pods in deployments, replica sets, or stateful sets
* Scaling is based on observed CPU utilization or other metrics (e.g., memory, custom metrics)

**Slide 3: How HPA Works**

* HPA controller checks metrics at regular intervals
* Adjusts the number of pods to meet the defined metric targets
* Ideal for applications with variable workloads

**Slide 4: Prerequisites**

* Metrics Server must be installed in the cluster
* Workloads should have CPU (and optionally memory) requests/limits set
* Kubernetes version 1.6 or higher recommended

**Slide 5: Create HPA via CLI** Command:

kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10

* Scales between 1 to 10 pods based on 50% CPU utilization target

**Slide 6: Check HPA Status** Command:

kubectl get hpa

* Displays current number of pods, metric values, and target utilization

**Slide 7: YAML Configuration Example**

apiVersion: autoscaling/v2  
kind: HorizontalPodAutoscaler  
metadata:  
 name: php-apache  
spec:  
 scaleTargetRef:  
 apiVersion: apps/v1  
 kind: Deployment  
 name: php-apache  
 minReplicas: 1  
 maxReplicas: 10  
 metrics:  
 - type: Resource  
 resource:  
 name: cpu  
 target:  
 type: Utilization  
 averageUtilization: 50

**Slide 8: Supported Metric Types**

* **Resource**: CPU, memory
* **Pods**: Average of custom metric across pods
* **Object**: Metric on another Kubernetes object
* **External**: Non-Kubernetes source (e.g., queue length)

**Slide 9: Best Practices & Limitations**

* Ensure metrics are stable to avoid over-scaling
* Set realistic min/max limits
* Use Cluster Autoscaler with HPA for full elasticity
* Monitor and test frequently for proper scaling behavior

**Slide 10: Thank You**

* Questions?
* Resources: <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>