Kubernetes Horizontal Pod Autoscaler Implementation

Project Overview

I developed a Kubernetes deployment with automatic scaling capabilities using Horizontal Pod Autoscaler (HPA) to demonstrate container orchestration and resource management skills.

 # Deployment Configuration :

- **Container** : nginx image with memory requests/limits
- **Availability** : Minimum 2 pod replicas for high availability

 # HPA Configuration :

- **Replicas** : 2-5 pods (min-max scaling range)
- **Scaling Metric** : Memory utilization at 75% threshold
- **Behavior** : Automatically scales up/down based on memory usage

 # Project Deliverables :

1. **YAML Manifests** : Deployment, HPA, and Service configurations
2. **Documentation** : HPA functionality, memory-based scaling mechanics, and resource management best practices
3. **Implementation Evidence** : kubectl outputs and live monitoring screenshots

 # Key Learning Outcomes :

- Kubernetes resource management and optimization
- Automated scaling strategies for production workloads
- Cost-effective cloud resource allocation
- Performance monitoring and troubleshooting

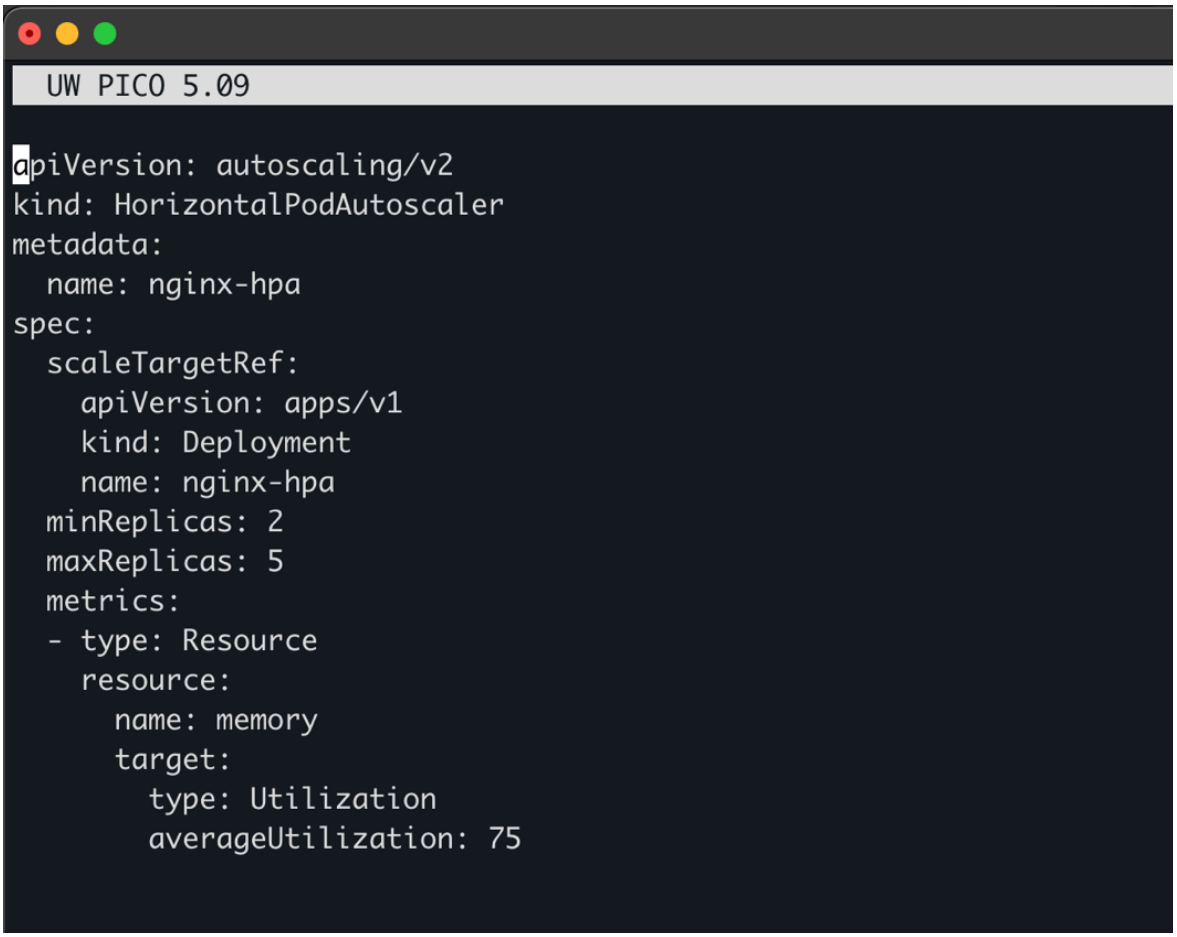
Technologies Used : Kubernetes, Docker, nginx, HPA, YAML

1. Create YAML manifests for Deployment and **Horizontal Pod Autoscalers** :

deployment1.yml

- **Replicas** : The number of pods running at the start.
- **Resources.requests.memory** : “64Mi”
The minimum memory requested.
- **Resources.limits.memory** : “128Mi”
The maximum memory requested/allowed.

hpa.yml



How it works is simple :

- If the average memory usage is **>75%**, the **HPA** will add pods (scale up).
- If the average usage is **<75%**, the **HPA** will reduce pods (scale down).
- But it remains within the minimum limit of 2 pods, maximum 5 pods.

What is a **Horizontal Pod Autoscaler** (**HPA**) ?

Horizontal Pod Autoscaler (**HPA**) is a Kubernetes component that automatically increases or decreases the number of pods in a **Deployment/ReplicaSet** based on specific metrics, such as CPU or memory usage.

How **Horizontal Pod Autoscalers** (**HPA**) work :

- The **HPA** reads memory usage metrics from pods.

If the average memory usage of a pod is **>75%** of requests.memory, then :

- **HPA** will add pods (scale out)

- If memory usage is **<75%**, then :

- **HPA** will reduce pods (scale in), to a minimum (2 pods)

What happens if memory usage exceeds or falls short of the target :

- **Memory usage >75%** : Increase the number of pods (no greater than maxReplicas)

- **Memory usage <75%** : Reduce the number of pods (no less than minReplicas)

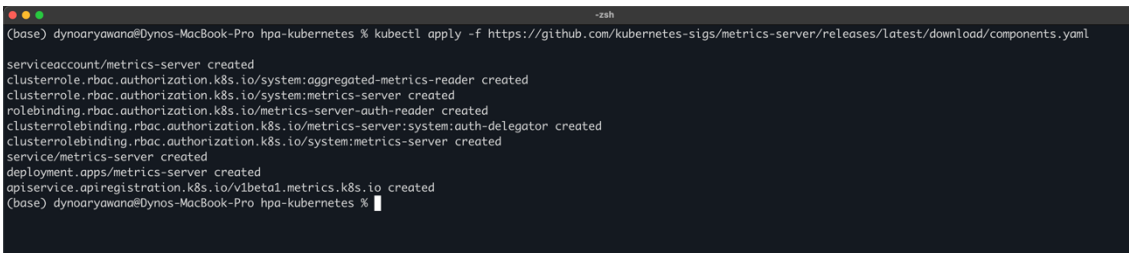
Real-life example :

I have a news website that receives a lot of visitors in the morning :

- When traffic increases, memory usage increases, **HPA** will automatically scale out

- When traffic decreases, memory usage decreases, **HPA** will scale in, the benefit of which is resource and cost savings in a cloud environment.

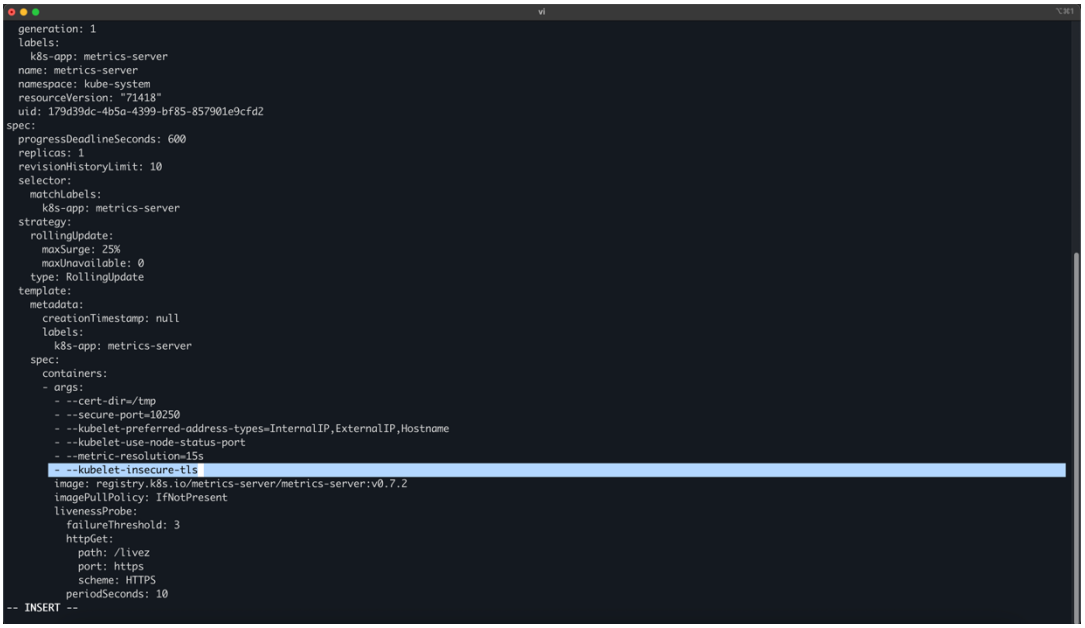
Installing **metrics-server**



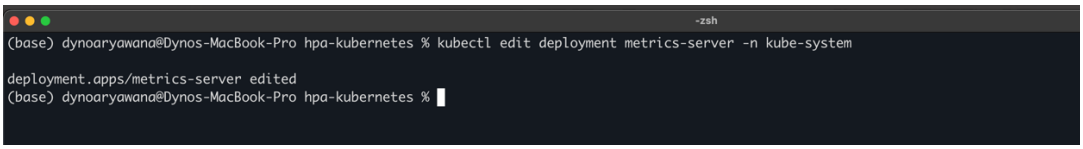
Modify **metrics-server** to work locally :

For **Horizontal Pod Autoscaler (HPA)** to function properly, Kubernetes requires a **metrics-server** capable of reading resource usage (such as CPU and memory) from pods. However, in on-premises environments like a Macbook running **Docker Desktop**, there's a problem because the kubelet uses a self-signed **TLS certificate**, which is invalid according to the **metrics-server**. To address this issue, modify the **metrics-server** deployment by adding the following argument to the args section :

- Add **- --kubelet-insecure-tls** in this line:

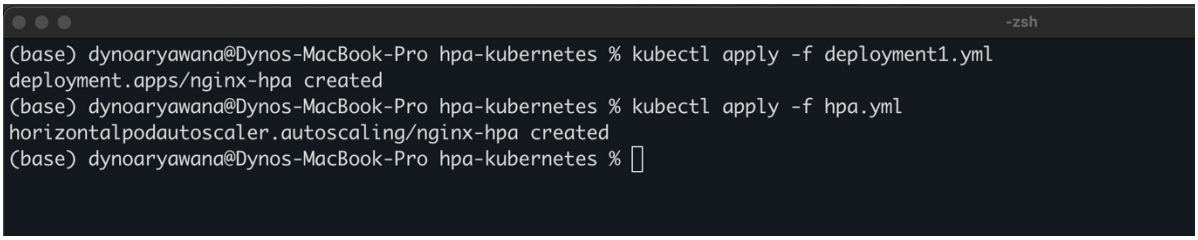


(save file and exit)

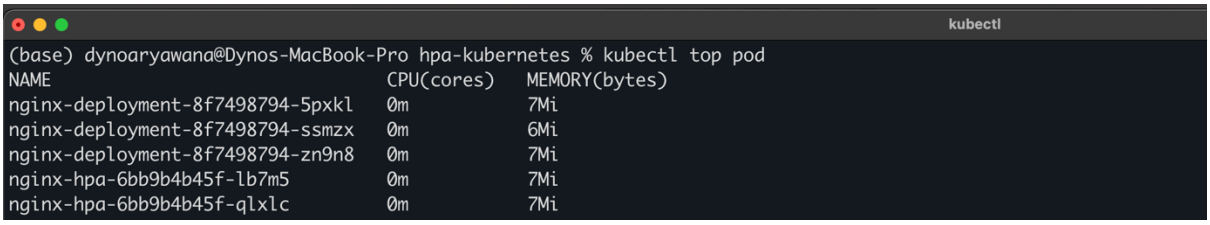


This argument instructs metrics-server to **Ignore TLS certificate verification when trying to access the kubelet API**, so it can still read metrics from the node even if the certificate is not trusted.

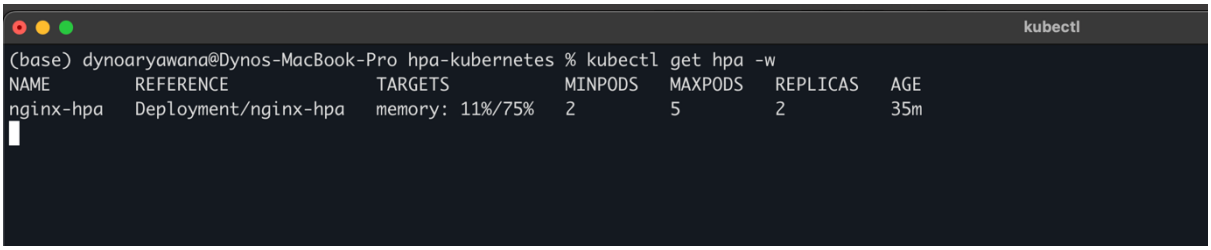
- Running YAML configuration with **kubectl apply -f <filename>**



- Monitoring CPU metrics across multiple pods **kubectl top pod**



- Monitoring active Autoscalers HPA metrics running in real time with **-w**



THANK YOU