**How to make a Kubernetes autoscaling HPA with example**

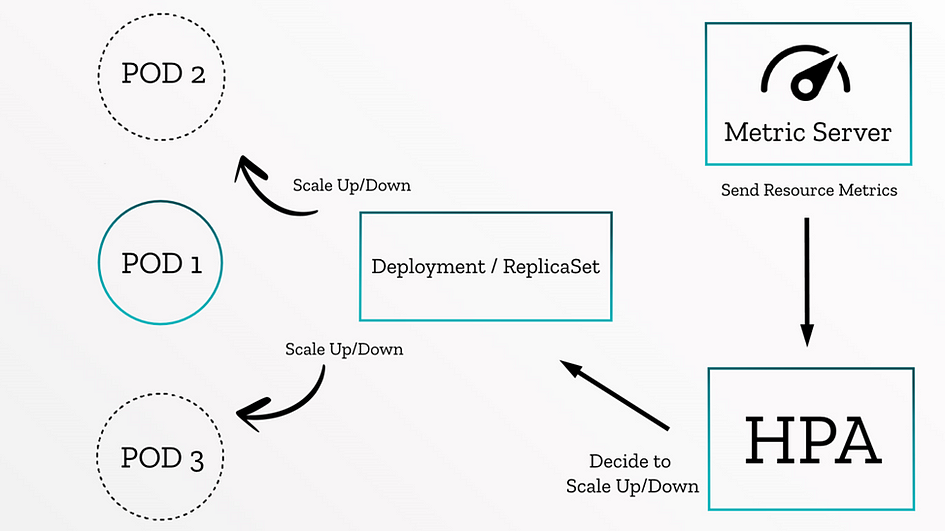
**horizontal pod autoscaling with yaml example file**

Kubernetes Horizontal Pod Autoscaling is another handy resource of kubernetes which helps you in autoscaling and managing your pods when there are overwhelming loads on your pods and the pods reach the defined limits.

How to make a Kubernetes autoscaling HPA with example

As a DevOps engineer you might have experienced exceeding resources like CPU and Ram on your workload and this is considered pretty normal, but things will be messed up when you are absent to take control of pods. Fortunately, HPA perfectly fits this issue and it auto scales your pods when resource usage meets the limits..

This is how Kubernetes HPA work, the metric server sends metrics of resource consumption to HPA and based on the rules you have defined in HPA manifest file, this object decides to scale up or down the pods. For example, if the CPU usage was more than 80 percentage, the HPA order replica Set and deployment to scale up pods and if the usage came below 10 percentage, the additional pods will be removed.



Kubernetes Horizontal autoscaling HPA

Let’s get our hands dirty and deploy a simple project with Kubernetes horizontal auto scaling step by step:

**1. Deploy Metric Server**

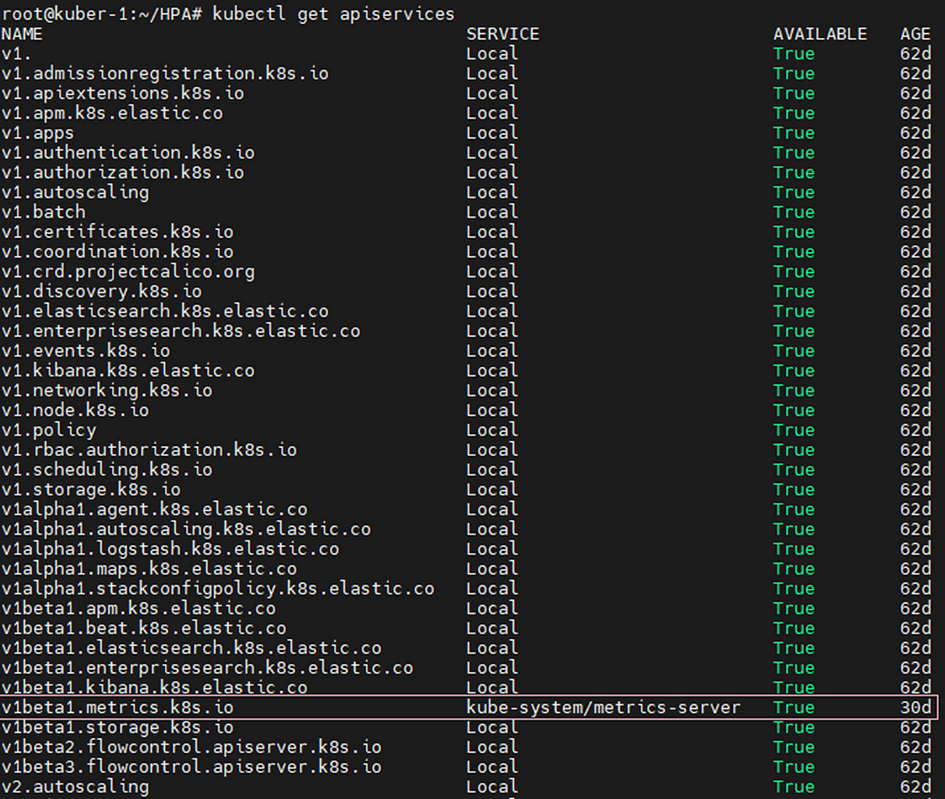
Metric server is an additional module of Kubernetes which is part of [Kubernetes-sigs](https://github.com/kubernetes-sigs) repository. Kubernetes metric server acts like a metric exporter on Kubernetes cluster that expose resource usage of nodes, pods … for many kinds of purposes. As mentioned before, HPA use metric server to observe pods resource usage.

To deploy the metric server, get the latest manifest file of metric server from here and add these parameters on the metric server deployment file and run.

spec:  
 hostNetwork: true  
 containers:  
 - args:  
 - --cert-dir=/tmp  
 - --secure-port=4443  
 - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname  
 - --kubelet-use-node-status-port  
 - --metric-resolution=15s  
 - --kubelet-insecure-tls

kubectl apply -f metric-server.yaml file

After deploying the manifest file, check the availability of metric server with the below command. Remember the marked API must be visible and True.



Now you can see the pods and nodes metrics with these two commands

kubectl top pods

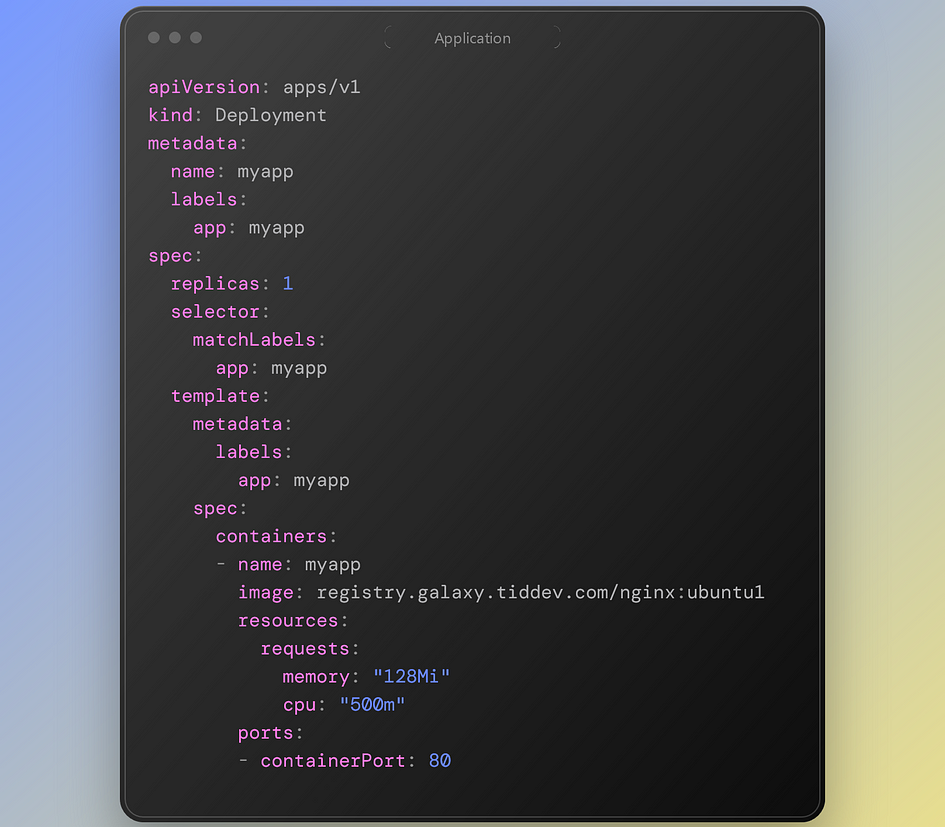
kubectl top nodes

**2. Deploy an application**

Deploying a containerized application is a must to test HPA which can be anything you want but you have to determine resource limitation or request parameters on the manifest Yaml file.

resources:  
 limits:  
 memory: "128Mi"  
 cpu: "500m"

In my example I have a simple web server and defined limits on my resources.

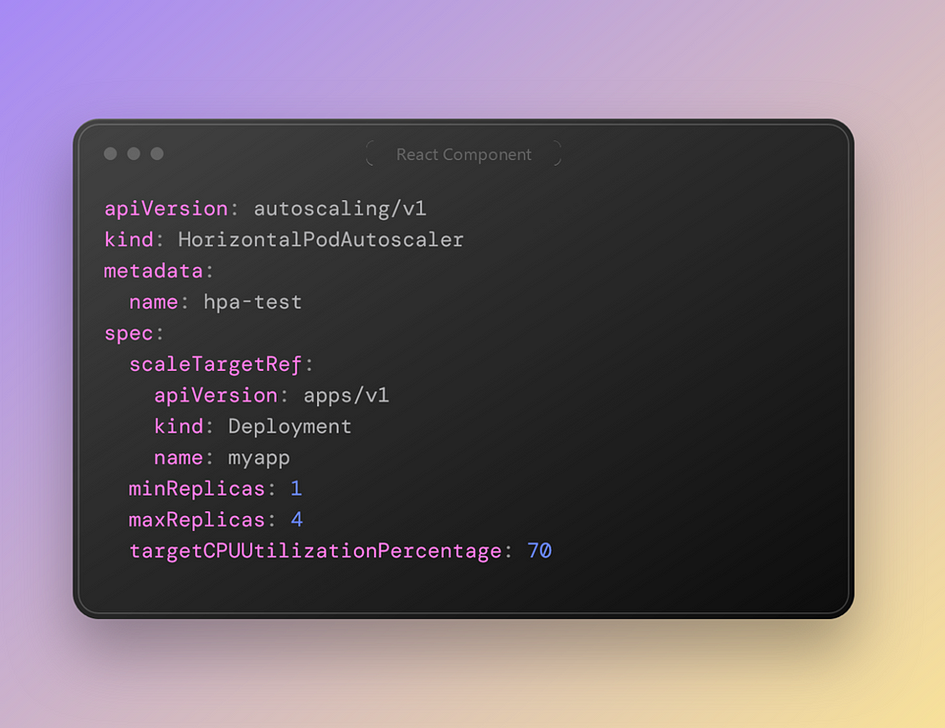


**3. Creating HPA**

After deploying your application, it is time for creating HPA manifest.

In this article I have mentioned both versions of autoscaling/v1 and autoscaling/v2

This is the manifest file for v1



Kubernetes HPA example v1

In this approach we have defined the API Version of our deployment object and the name of it.

We have demonstrated the minimum replicas of the pointed deployment and the maximum replicas, although the target CPU usage is 70 percent. It means if the CPU usage was higher than 70% the replicas would be scaled up to 4 replicas and after decreasing CPU usage under 70%, the replicas numbers will be 1 again.

*In my example, I have added “stress” package to the base image and increased CPU usage but you can do it with http request or any other way based on your application. I have entered the container environment and ran this command “stress -c 10” which simulates like 10 core CPU.*

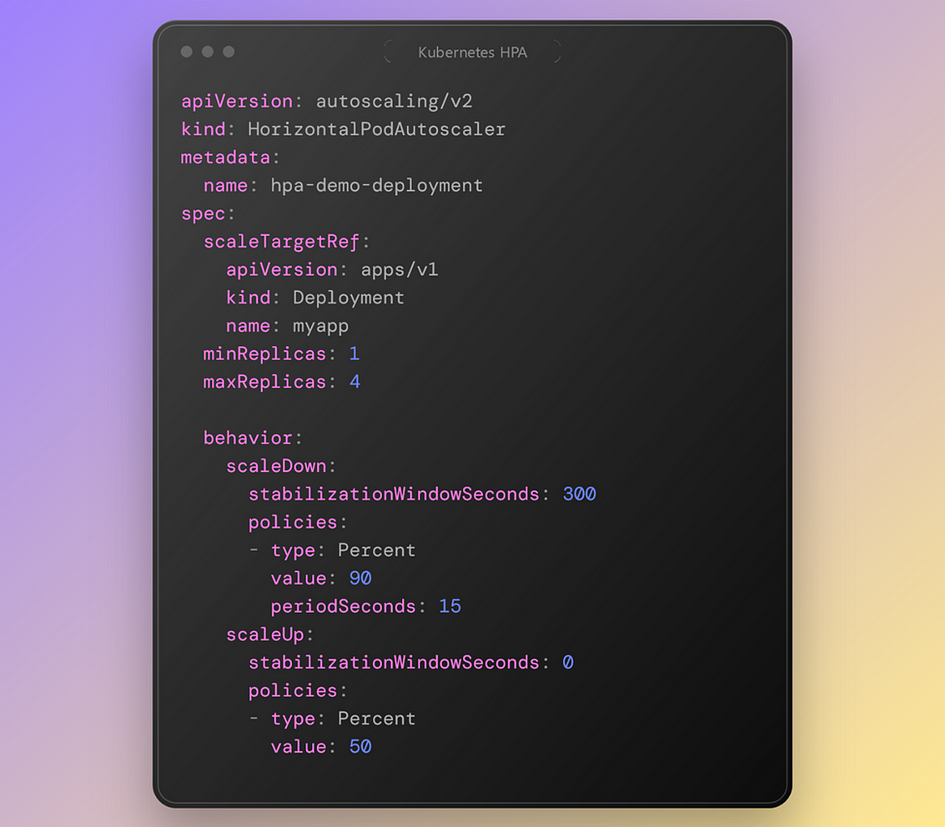
https://miro.medium.com/v2/resize:fit:945/1*Kk3oymf5JELFz-FfFnTulg.png

Kubernetes Horizontal Pod Scaling

This is the result after some seconds and the pods scaled up until 4. The average container resource is like 70% and after reaching the threshold, the replicas would scale up to 4 pods.

Autoscaling version 2 is the new and better approach which gives you more accessibility on pods and lets you assign different policy on the HPA.

On the autoscaling/v2 you can still see min and max replicas keys and they have same behavior. But the most important feature is behavior. Behavior is divided into two scale down and scale up section which lets you define policies for both up and down scale.



Kubernetes HPA example v2

As it seems in the scale up policy section If the pod`s CPU usage became higher that 50 percentage, after 0 seconds the pods will be scaled up to 4 replicas. But in the scale down part if the CPU usage would be lower than 90% and yet after 15 seconds and staying the usage stable for 300 seconds, the pods would scale down to one replica.

**4. Extra points**

There are some other points in the v2 API version which I should mention.

Look at this example:

behavior:  
 scaleDown:  
 policies:  
 - type: Percent  
 value: 10  
 periodSeconds: 60  
 - type: Pods  
 value: 5  
 periodSeconds: 60  
 selectPolicy: Min

We have two policies here, to ensure that no more than 5 Pods are removed per minute, you can add a second scale-down policy with a fixed size of 5, and set selectPolicy to minimum. Setting selectPolicy to Min means that the autoscaler chooses the policy that affects the smallest number of Pods.

**selectPolicy: Min**

If you set selectPolicy to “Min,” the HPA will select the policy with the minimum value among the matching policies during a scaling decision.

Example: If you have two policies for scaling up based on CPU utilization, one with a percentage increase of 150% and another with a pod count increase of 5, the policy with the minimum increase (5 in this case) will be selected.

**selectPolicy: Max**

Conversely, if you set selectPolicy to “Max,” the HPA will select the policy with the maximum value among the matching policies during a scaling decision.

Example: Using the same scenario as above, if you have two policies for scaling up based on CPU utilization, one with a percentage increase of 150% and another with a pod count increase of 5, the policy with the maximum increase (150% in this case) will be selected.

The selectPolicy value of Disabled turns off scaling the given direction. So to prevent downscaling policy would be used.