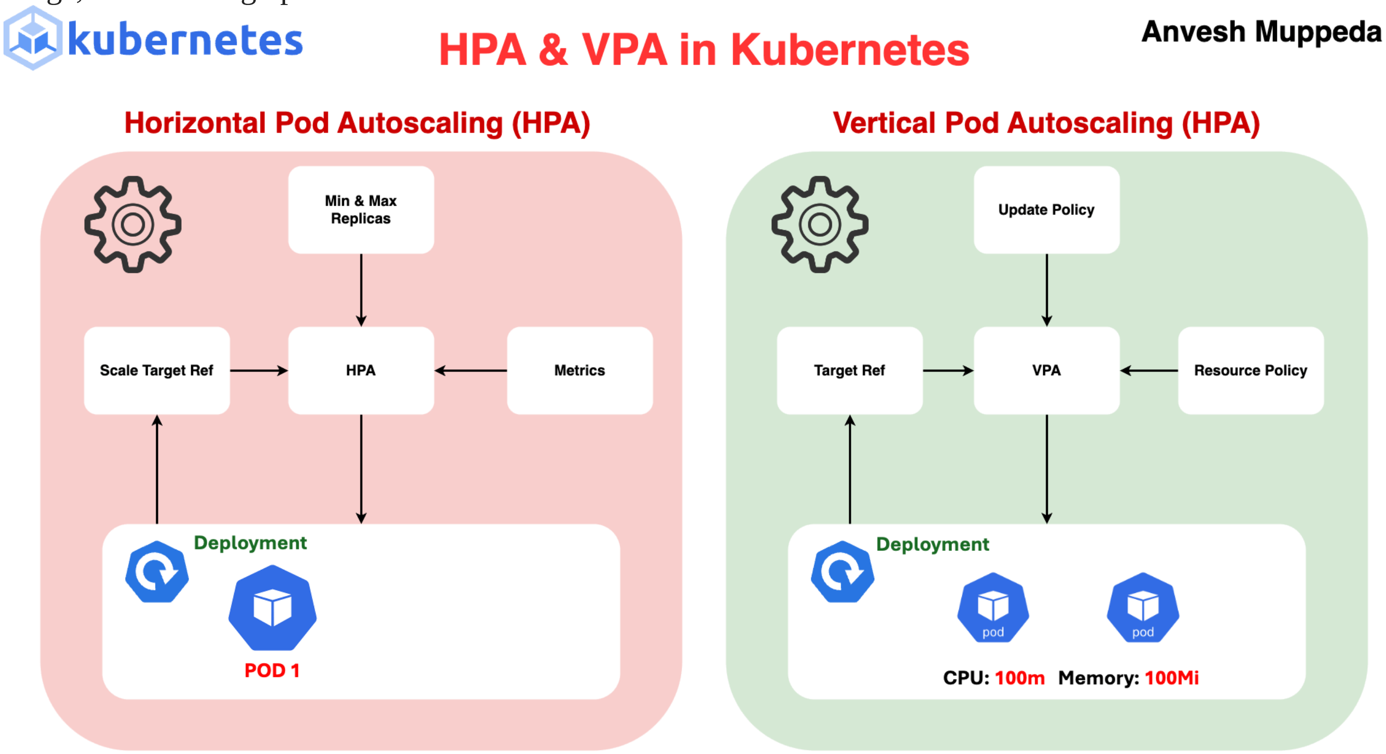
Configure autoscaling in your cluster (Horizontal scaling)

the world of container orchestration, Kubernetes has emerged as the de facto standard. One of its powerful features is autoscaling, which helps maintain application performance and optimize resource usage. In this post, we'll dive into two main types of autoscaling in Kubernetes: Horizontal Pod Autoscaling (HPA) and Vertical Pod Autoscaling (VPA).

Introduction to Kubernetes Autoscaling

Autoscaling in Kubernetes ensures that your application can handle varying loads by automatically adjusting the number of running pods or the resources allocated to them. This dynamic adjustment is crucial for maintaining application performance, optimizing resource usage, and reducing operational costs.



HPA & VPN in Kubernetes by Anvesh Muppeda

Metric Server Setup

Before testing Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA), it's essential to have the Metrics Server installed in your Kubernetes cluster. The Metrics Server collects resource usage metrics from the cluster's nodes and pods, which are necessary for autoscaling decisions.

You can install the Metrics Server using either a YAML manifest or the official Helm chart. To install the latest release of the Metrics Server from the YAML manifest, follow these steps:

1. **Download the Components Manifest:** Use kubectl apply to download and apply the Components manifest directly from the latest release of the Metrics Server:

```
$ kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

This command fetches the YAML manifest for the latest release of the Metrics Server from its GitHub repository and applies it to your Kubernetes cluster.

2. **Verify Installation:** After applying the manifest, verify that the Metrics Server pods are running successfully. You can check the pods in the kube-system namespace:

```
kubectl get pods -n kube-system | grep metrics-server
```

You should see pods related to the Metrics Server running and ready.

3. **Confirm Metrics Collection:** Once the Metrics Server is up and running, you can confirm that it’s collecting metrics by querying the API. For example, you can retrieve the CPU and memory usage metrics for nodes and pods:

```
$ kubectl top nodes kubectl top pods --all-namespaces
```

If the Metrics Server is properly installed and functioning, you should see CPU and memory usage metrics for nodes and pods in your cluster.

With the Metrics Server installed and collecting metrics, you can proceed to test the Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA) functionalities in your Kubernetes cluster.

Horizontal Pod Autoscaling (HPA)

What is HPA?

Horizontal Pod Autoscaling (HPA) automatically scales the number of pods in a replication controller, deployment, or replica set based on observed CPU utilization (or other select metrics). This allows your application to scale out (add more pods) or scale in (reduce the number of pods) in response to load changes.

How Does HPA Work?

HPA operates by:

1. **Monitoring Metrics:** HPA continuously monitors the specified metrics (e.g., CPU usage, memory usage, custom metrics).

2. **Evaluating Thresholds:** It compares the current metric values against predefined thresholds.

3. S**caling Pods:** If the metric values exceed the thresholds, HPA increases the number of pods. Conversely, if the values drop below the thresholds, it decreases the number of pods.

Advantages of HPA

- **Elasticity**: Automatically adjusts the number of pods to meet current demand.

- **Cost-Efficiency:** Optimizes resource usage by adding or removing pods as needed.

- **Performance**: Helps maintain application performance during high traffic periods.

HPA Practical Example

To set up HPA, you need to define an `HPA` object in your Kubernetes cluster.

Let’s deploy an example using `HPA`.

Step 1: Create a Deployment

Create a file named `hpa-deployment.yaml` with the following content:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hpa-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        resources:
          requests:
            cpu: "25m"
          limits:
            cpu: "200m"
```

Apply the deployment:

```
kubectl apply -f hpa-deployment.yaml
```

Step 2: Create a Service

Create a file named `nginx-service.yaml` with the following content:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
  labels:
    app: nginx
spec:
  ports:
  - port: 80
  selector:
    app: nginx
```

Apply the service:

```
kubectl apply -f nginx-service.yaml
```

Step 3: Create an HPA

Create a file named `hpa.yaml` with the following content:

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

Apply the HPA:

```
kubectl apply -f hpa.yaml
```

With this setup, the HPA will automatically scale the number of `nginx` pods between 1 and 10 based on the CPU utilization, aiming to keep it around 50%.

Step 4: Testing

Now, let’s test the HPA by increasing the load on the hpa-deploy deployment. We'll use the following command to create a new pod that generates load on the target deployment and observe if the number of pods increases accordingly.

Pod status before generate load:

```
$ kubectl top po
NAME                          CPU(cores)   MEMORY(bytes)
hpa-deploy-695d6d995c-lb7f2   0m           2Mi
```

Run this command to generate load:

```
$ kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- /bin/sh -c "while
sleep 0.01; do wget -q -O- http://nginx-svc; done"
```

This command will continuously generate requests to the nginx-svc service, thereby increasing the CPU utilization of the nginx pods.

To observe the CPU utilization and the status of the HPA, use the following commands:

Check the CPU utilization of the pods:

```
$ kubectl top po
NAME                          CPU(cores)   MEMORY(bytes)
hpa-deploy-695d6d995c-5smnw   13m          3Mi
```

```
hpa-deploy-695d6d995c-lb7f2   17m         2Mi
load-generator                133m        0Mi
```

Monitor the HPA status:

```
$ kubectl get hpa -w
NAME        REFERENCE               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
nginx-hpa   Deployment/hpa-deploy   0%/50%    1         10        1          101s
nginx-hpa   Deployment/hpa-deploy   80%/50%   1         10        1          105s
nginx-hpa   Deployment/hpa-deploy   68%/50%   1         10        2          2m
nginx-hpa   Deployment/hpa-deploy   54%/50%   1         10        2          2m15s
nginx-hpa   Deployment/hpa-deploy   42%/50%   1         10        2          2m30s
```

These commands will help you monitor the CPU usage and see if the HPA adjusts the number of pods in response to the increased load.

By following these steps, you can validate the functionality of the HPA and ensure it dynamically scales your application based on CPU utilization.