Optimizing GPU Costs with GPU Slicing on EKS

# Step 2: Install the NVIDIA GPU Operator

NVIDIA's GPU Operator automates the management of GPU drivers, device plugins, and other necessary components to run GPU workloads on Kubernetes.

## Add the NVIDIA Helm Repository:

```bash  
helm repo add nvidia https://nvidia.github.io/gpu-operator  
helm repo update  
```

## Install the NVIDIA GPU Operator:

```bash  
helm install nvidia-gpu-operator nvidia/gpu-operator --namespace gpu-operator --create-namespace  
```

This will deploy the GPU Operator in your cluster, ensuring that the NVIDIA GPU drivers and related components are managed automatically.

# Step 3: Enable GPU Slicing

## 3.1 Configure GPU Time-Slicing

Time-slicing enables multiple workloads to share the same physical GPU by allocating them time slots for GPU usage. To enable this, create a ConfigMap that specifies the number of virtual GPUs (vGPUs) that each physical GPU can be sliced into.

### Create a ConfigMap for GPU Time-Slicing:

```yaml  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: nvidia-device-plugin  
 namespace: kube-system  
data:  
 any: |-  
 version: v1  
 flags:  
 migStrategy: none  
 sharing:  
 timeSlicing:  
 resources:  
 - name: nvidia.com/gpu  
 replicas: 10  
```

### Apply the ConfigMap:

```bash  
kubectl apply -f nvidia-device-plugin.yaml  
```

This ConfigMap configures each physical GPU to be divided into 10 virtual GPUs.

## 3.2 Upgrade NVIDIA GPU Operator with Time-Slicing

Upgrade the GPU Operator to use the newly created time-slicing configuration:

```bash  
helm upgrade nvidia-gpu-operator nvidia/gpu-operator --namespace gpu-operator --set config.name=nvidia-device-plugin  
```

## 3.3 Verify GPU Slicing Configuration

Check the number of virtual GPUs available on the nodes:

```bash  
kubectl get nodes -o json | jq -r '.items[] | select(.status.capacity."nvidia.com/gpu" != null) | {name: .metadata.name, capacity: .status.capacity}'  
```

This should show the number of available GPUs, including virtual GPUs created by slicing.

# Step 4: Deploy Workloads Using GPU Slicing

Deploy workloads that utilize GPU slicing by specifying a fraction of a GPU in the resource requests. Here's an example deployment:

```yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: gpu-workload  
 labels:  
 app: gpu-app  
spec:  
 replicas: 5  
 selector:  
 matchLabels:  
 app: gpu-app  
 template:  
 metadata:  
 labels:  
 app: gpu-app  
 spec:  
 containers:  
 - name: gpu-container  
 image: <your-image>  
 resources:  
 limits:  
 nvidia.com/gpu: 0.1 # Request 10% of a GPU  
```

## Apply this deployment:

```bash  
kubectl apply -f gpu-workload.yaml  
```

This configuration ensures that each pod requests a fraction (10%) of a GPU, allowing multiple pods to share the same physical GPU.

# Step 5: Leverage Karpenter Autoscaler for Dynamic GPU Provisioning (Optional)

## 5.1 Configure Karpenter Provisioner for GPU Instances

Create a Karpenter provisioner that supports GPU instances and autoscaling:

```yaml  
apiVersion: karpenter.sh/v1alpha5  
kind: Provisioner  
metadata:  
 name: gpu-provisioner  
spec:  
 requirements:  
 - key: "kubernetes.io/arch"  
 operator: In  
 values: ["amd64"]  
 - key: "instance-type"  
 operator: In  
 values: ["g4dn.xlarge", "p3.8xlarge"]  
 - key: "nvidia.com/gpu"  
 operator: Exists  
 limits:  
 resources:  
 cpu: "1000"  
 memory: "4000Gi"  
 providerRef:  
 name: <provider-name>  
```

## Apply the Karpenter provisioner:

```bash  
kubectl apply -f karpenter-gpu-provisioner.yaml  
```

## 5.2 Monitor and Validate Karpenter Scaling

Karpenter will now dynamically provision or deprovision GPU-backed nodes based on the workload’s GPU demands. You can monitor this by checking node provisioning with the following command:

```bash  
kubectl get nodes  
```

# Step 6: Monitoring and Optimization

1. \*\*Monitor GPU Usage\*\*: Use `nvidia-smi` or integrate Prometheus and Grafana to monitor the real-time utilization of GPU resources.  
2. \*\*Adjust Time-Slicing\*\*: Based on your workload demands, adjust the `replicas` in the `ConfigMap` to change the number of virtual GPUs per physical GPU.  
3. \*\*Scaling\*\*: Karpenter will automatically scale GPU instances up or down based on actual GPU usage, helping reduce costs by scaling down unused resources.