GPU Slicing

GPU slicing is a virtualization technique that lets multiple workloads share a GPU by dividing its processing time into slices. It's useful when many jobs need to run on limited hardware

To help optimize GPU costs for AI workloads on Amazon Elastic Kubernetes Service (EKS), enabling GPU slicing is an effective strategy. This approach allows multiple workloads to share a single GPU, maximizing utilization and reducing expenses. Integrating this with Karpenter, an open-source Kubernetes cluster autoscaler, can further enhance cost efficiency by dynamically managing node scaling based on real-time demand.

Implementing GPU Slicing on EKS Clusters:

GPU slicing can be achieved through two primary methods: Time-Slicing and Multi-Instance GPU (MIG)

1. **Time-Slicing**: This method allows multiple processes to share a GPU by allocating time slots for each process. It's particularly useful when workloads have intermittent GPU demands.

2. **Multi-Instance GPU (MIG)**: Available on NVIDIA A100 GPUs, MIG partitions a single GPU into multiple isolated instances, each with dedicated resources. This ensures predictable performance for concurrent workloads.

Steps to Enable GPU Slicing on EKS:

1. **Deploy the NVIDIA GPU Operator**: This operator automates the management of NVIDIA GPU resources in Kubernetes, including driver installation and device plugin deployment.

- **Installation**: Use Helm to install the GPU Operator in your EKS cluster

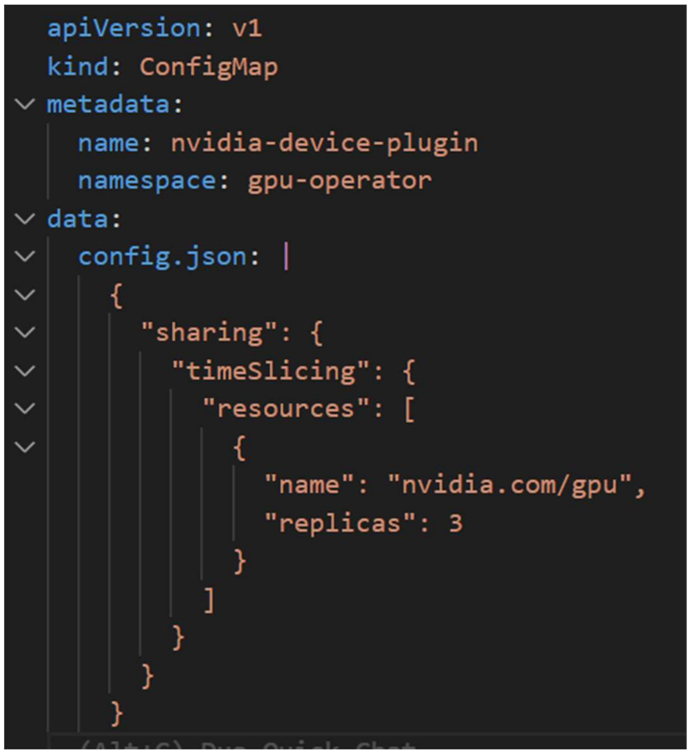
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia

helm repo update

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

- **Configuration**: After installation, configure the operator for time-slicing or MIG based on your GPU model and workload requirements

Time-Slicing: Create a ConfigMap to define the number of GPU replicas.



```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nvidia-device-plugin
  namespace: gpu-operator
data:
  config.json: |
    {
      "sharing": {
        "timeSlicing": {
          "resources": [
            {
              "name": "nvidia.com/gpu",
              "replicas": 3
            }
          ]
        }
      }
    }
```

setting replicas to 3 allows three pods to share a single GPU. Apply the ConfigMap and restart the NVIDIA device plugin to enable time-slicing.

MIG: Label your GPU nodes to define MIG strategies.

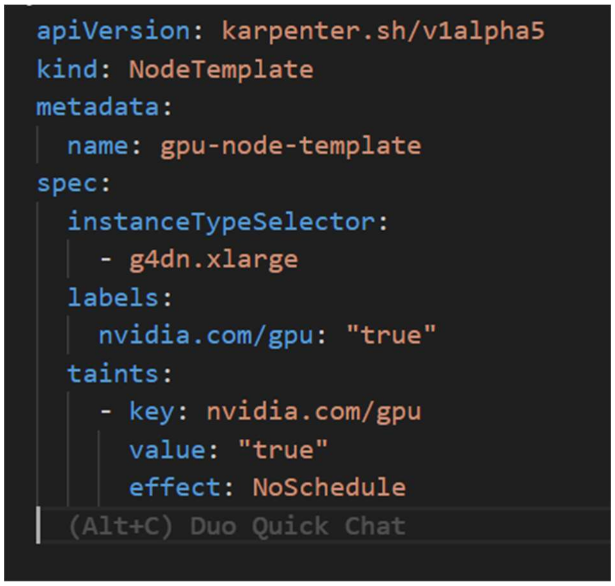
kubectl label node <node-name> nvidia.com/mig.config=all-1g.5gb

This command configures each GPU to have multiple MIG instances of a specified size.

2. Integrating GPU Slicing with Karpenter Autoscaler

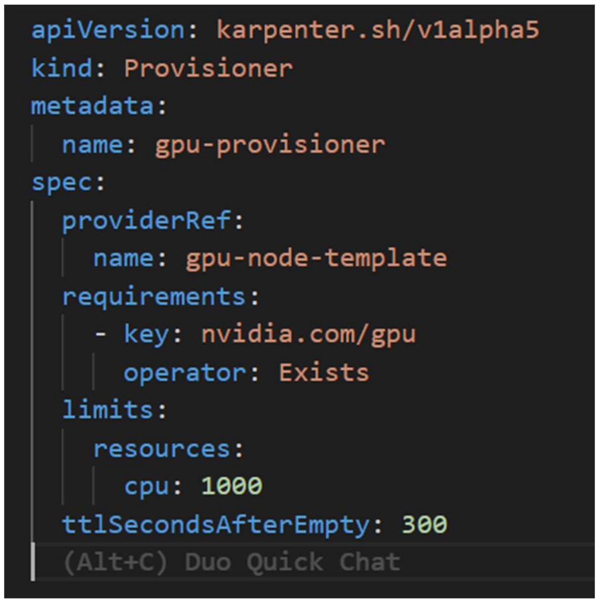
Karpenter is a Kubernetes cluster autoscaler that provisions nodes based on workload demands.
To leverage GPU slicing with Karpenter:

1. **Define Node Templates**: Create a NodeTemplate that specifies GPU instance types and includes necessary labels and taints.



```yaml
apiVersion: karpenter.sh/v1alpha5
kind: NodeTemplate
metadata:
  name: gpu-node-template
spec:
  instanceTypeSelector:
    - g4dn.xlarge
  labels:
    nvidia.com/gpu: "true"
  taints:
    - key: nvidia.com/gpu
      value: "true"
      effect: NoSchedule
(Alt+C) Duo Quick Chat
```

2. **Configure Provisioners**: Set up a Provisioner that references the NodeTemplate and defines scaling parameters.



```yaml
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: gpu-provisioner
spec:
  providerRef:
    name: gpu-node-template
  requirements:
    - key: nvidia.com/gpu
      operator: Exists
  limits:
    resources:
      cpu: 1000
  ttlSecondsAfterEmpty: 300
(Alt+C) Duo Quick Chat
```

This configuration ensures that Karpenter provisions GPU-enabled nodes as needed and scales them down when idle.

3. Deploying GPU-Accelerated Workloads

When deploying workloads that require GPU resources:

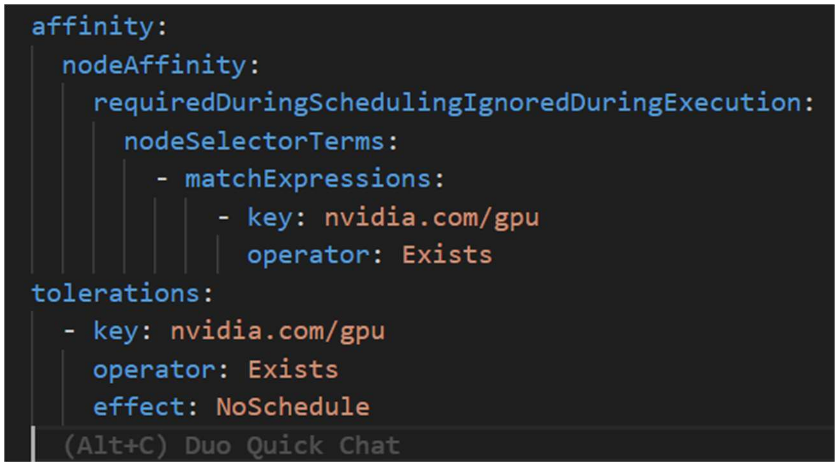
- **Resource Requests**: Specify GPU resource requests in your pod specifications. (*i.e for the GPU time-slicing setup*)

resources:

  limits:

    nvidia.com/gpu: 1

- **Node Affinity and Tolerations**: Ensure pods are scheduled on GPU-enabled nodes by setting appropriate node affinity and tolerations. (i.e for MIG)



```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: nvidia.com/gpu
              operator: Exists
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

By implementing GPU slicing and integrating it with Karpenter, you can achieve efficient GPU utilization and cost savings on your EKS clusters.

Benefits of GPU Slicing

- **Cost Savings**: More workloads can run on the same GPU, reducing the number of GPUs required.
- **Improved Utilization**: Prevents GPU underutilization by efficiently allocating resources.
- **Workload Isolation**: MIG ensures that workloads don't interfere with each other.
- **Scalability**: Supports dynamic resource allocation based on demand