How To Enable GPU Access for EKS Cluster

Document Level Classification

200

- Introduction
- Worker Nodes
- Karpenter Nodes
- Enabling GPU Support in Kubernetes
    - Deployment via helm
- Time-Slicing GPUs in Kubernetes

Introduction

This guide will show how to enable GPU instances access from EKS Clusters to enable Graphics Accelerated Applications or Machine Learning projects. The following procedure can be made using two kinds of node management:

Worker Nodes

- Worker nodes require to perform a specific count of g5 nodes addition as worker nodes to your EKS Cluster, this is helpful if you already had dimensioned the count of replicas/pods or GPU you'll require along the service/project is running
- In this mode of worker node management, you add the instance type and a certain quantity of nodes directly to your eks cluster, this mode leaves you to add min, max, and desire count of nodes but autoscaling is not managed dynamically by Karpenter.

- In this mode you require to modify your terraform [substrate/infrastructure]() for your specific EKS Cluster were you want to add these GPU Nodes, you can follow the next example:
- Adding "[aws_eks_node_group]()" to your vpc.tf file in your account for substrate/infrastructure
    - ```
      resource "aws_eks_node_group" "gpu_worker" {
        cluster_name    = "adda-dev-sandbox"
        node_group_name = "adda-dev-sandbox-managed-gpu-worker"
        node_role_arn   = module.eks-sandbox.worker_role_arn
        subnet_ids      = module.vpc_sandbox.nat_subnet_ids
        disk_size       = 250
        instance_types  = ["g5.12xlarge"]
        ami_type        = "AL2_x86_64_GPU"

        scaling_config {
          desired_size = 2
          max_size     = 4
          min_size     = 2
        }

        # taint {}

        labels = {}

        remote_access {
      ```
      ```

```
    ec2_ssh_key = "adda-dev-sandbox-worker"
    source_security_group_ids = [
      module.eks-sandbox.master_sg_id
    ]
  }

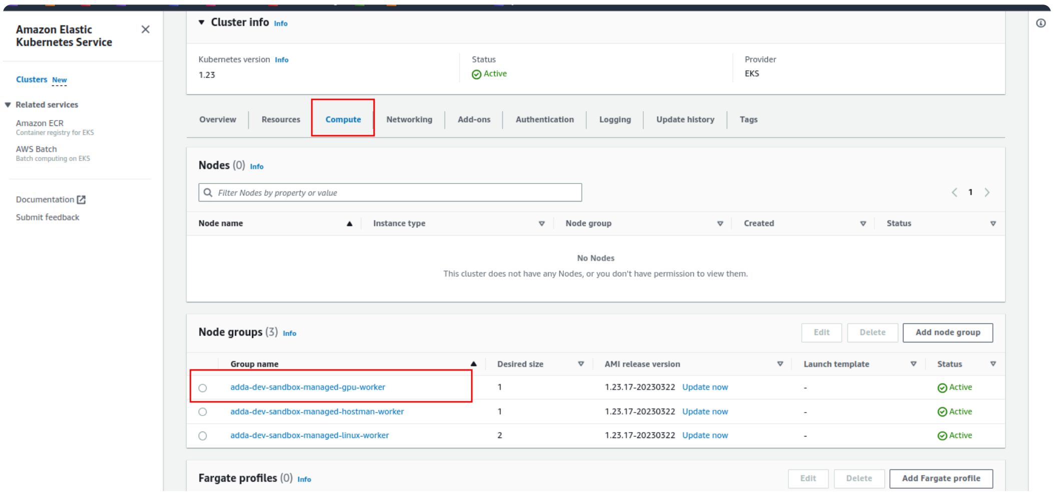
  # Ensure that IAM Role permissions are created before and del
  # Otherwise, EKS will not be able to properly delete EC2 Inst
  /*
  depends_on = [
    aws_iam_role_policy_attachment.worker_node_policy,
    aws_iam_role_policy_attachment.worker_cni_policy,
    aws_iam_role_policy_attachment.worker_registry_policy,
  ]
*/
  lifecycle {
    # create_before_destroy = true
    # prevent_destroy = true

    ignore_changes = [
      scaling_config[0].max_size,
      scaling_config[0].min_size,
      scaling_config[0].desired_size
    ]
  }

  tags = {
    "epic/substrate/eks" = "adda-dev-sandbox"
  }

}
```

- This procedure leaves you to enable GPU TimeSlicing sharing, as we will discuss further in this guide

- Once your terraform modification it's approved, merge & applied by #cloud-ops-support-ext, you'll be able to see a new worker node group attached to your EKS Compute
    - 

- To take advantage of this configuration, you require to define "nodegroup" which pod/job will be assigned as part of your NodeSelector specification as part of EKS Chart manifiest
    - Single pod example
        - ```
          apiVersion: v1
          kind: Pod
          metadata:
            name: nvidia-smi
          spec:
            nodeSelector:
              eks.amazonaws.com/nodegroup: "adda-dev-sandbox-managed-
            restartPolicy: OnFailure
            containers:
            - name: nvidia-smi
              image: nvidia/cuda:12.1.0-devel-ubuntu22.04
              args:
              - "nvidia-smi"
              resources:
                limits:
                  nvidia.com/gpu: 2
          ```
          ```

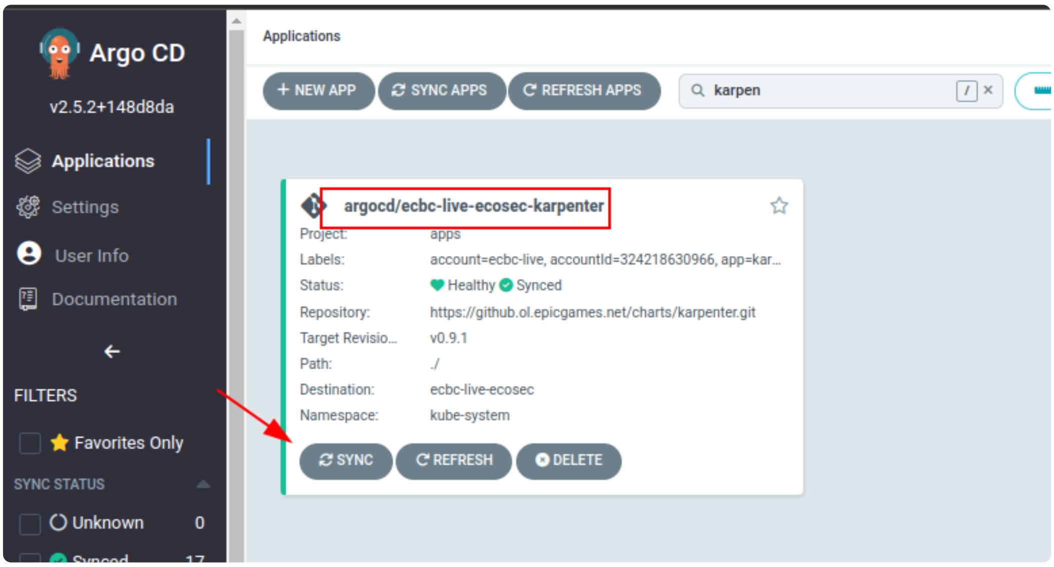
- deployment/service example
  - ```
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: nvidia-smi
      labels:
        app: nvidia-smi
    spec:
      replicas: 4
       nodeSelector:
        eks.amazonaws.com/nodegroup: "adda-dev-sandbox-managed-
      template:
        metadata:
          name: nvidia-smi
          labels:
            app: nvidia-smi
        spec:
          containers:
          - name: nvidia-smi
            image: nvidia/cuda:12.1.0-devel-ubuntu22.04
            args:
            - "nvidia-smi"
            resources:
              limits:
                nvidia.com/gpu: 4
    ```
- epic-app helm example
  - ```
    epic-app:
      nameOverride: "hmc-canary-sandbox"
      fullnameOverride: "hmc-canary-sandbox"

      nodeSelector:
        eks.amazonaws.com/nodegroup: "adda-dev-sandbox-managed-
    ```

Karpenter Nodes

- Karpenter nodes are provisioned under demand, which means along your service/application is not running that computer/gpu power is not in use, the node is destroyed to avoid costs, it's more flexible if you require an increase or decrease dynamically the count of pods/replicas accessing GPU power, and removes those nodes which processing power is not being used.
- in this mode you require to modify your Karpenter configuration at ArgoCD repository, to leave substrate management systems to apply and manage EPIC systems management services, which control Karpenter configurations.
- add the [following GPU block](#)in your cluster configuration file for [ArgoCD](#)
    - ```
      gpu:
       enabled: true
       labels:
         use: gpu
       providerRef:
         name: default
       requirements:
         arch:
           values:
           - amd64
         capacityType:
           values:
           - on-demand
         instanceGPUManufacturer:
           values:
           - nvidia
         instanceType:
           values:
           - g5.12xlarge
         zone:
           values: {}
       taints:
      ```
      ```

```
    - effect: NoExecute
      key: epicgames.com/gpu
    - effect: NoSchedule
      key: nvidia.com/gpu
    ttlSecondsAfterEmpty: 300
```

- For more specific information about Karpenter nodes addition and management, you can review the following guides:
  - https://confluence-epicgames.atlassian.net/wiki/spaces/IE/pages/82150715
- Once your PR be approved & merged by [#cloud-ops-support-ext](#), you'll require to sync/apply new configuration in your cluster's Karpenter, please go to [ArgoCD GUI Page](#) and sync your Karpenter app.
  - 

- To take advantage of these configurations you require to perform the following additions to your EKS Manifest or chart, here we'll explore some examples that may adjust to your specific project requirements.
- The most important pieces of these configurations to be taken for Karpenter and provision your requested node properly and your pod/service being assigned to your provisioned node are
  - The affinity matches:
    -
```
spec:
  affinity:
```

```
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: use
            operator: In
            values:
            - gpu
```

- taint/tolerations
  - ```
     spec:
    ...
       tolerations:
      - effect: NoExecute
        key: epicgames.com/gpu
        operator: Exists
      - effect: NoSchedule
        key: nvidia.com/gpu
        operator: Exists
    ```
- Another less important but that will leave you to manage how many GPUs will be accessed by your pod/service, are the resources limits
  - ```
     spec:
    ...
       containers:
       ...
          resources:
          requests:
            nvidia.com/gpu: 4
          limits:
            nvidia.com/gpu: 4
    ```

- This is an example if you want to run a single pod (containerized application with specific arguments)
  - ```
    apiVersion: v1
    kind: Pod
    metadata:
      name: nvidia-smi
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: use
                operator: In
                values:
                - gpu
      tolerations:
      - effect: NoExecute
        key: epicgames.com/gpu
        operator: Exists
      - effect: NoSchedule
        key: nvidia.com/gpu
        operator: Exists
      restartPolicy: OnFailure
      containers:
      - name: nvidia-smi
        image: nvidia/cuda:12.1.0-devel-ubuntu22.04
        args:
        - "nvidia-smi"
        resources:
          requests:
            nvidia.com/gpu: 4
          limits:
            nvidia.com/gpu: 4
    ```
    ```

- This is an example if you want to perform a Service Deployment (a service that requires one or more pods, long-term running application/service)
  - Here you add "replicas" value, which means how many times GPU Cores you'll need, each replica will access the count of GPU Core you specify in your resource.requests/resource.limits, (if you specify resource.requests: 2 and replica: 2, you'll require at least 4 GPU Cores to satisfy the GPU Cores demand by your Deployment)
  - Your EKS manifest should look like these pieces of code, it's probably required to adjust it depending if your service uses HELM Chart or any other kind of chart management tool.
  - ```
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: nvidia-smi
      labels:
        app: nvidia-smi
    spec:
      replicas: 4
      selector:
        matchLabels:
          app: nvidia-smi
      template:
        metadata:
          name: nvidia-smi
          labels:
            app: nvidia-smi
        spec:
          affinity:
            nodeAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
                nodeSelectorTerms:
                - matchExpressions:
                  - key: use
    ```
    ```

```
            operator: In
            values:
            - gpu
  tolerations:
  - effect: NoExecute
    key: epicgames.com/gpu
    operator: Exists
  - effect: NoSchedule
    key: nvidia.com/gpu
    operator: Exists
  containers:
  - name: nvidia-smi
    image: nvidia/cuda:12.1.0-devel-ubuntu22.04
    args:
    - "nvidia-smi"
    resources:
      limits:
        nvidia.com/gpu: 4
```

- Notice that the minimum and important part to assign the job to a g5 instance is the affinity and tolerance settings:
    - ```
      spec:
        affinity:
          nodeAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              nodeSelectorTerms:
              - matchExpressions:
                - key: use
                  operator: In
                  values:
                  - gpu
        tolerations:
        - effect: NoExecute
          key: epicgames.com/gpu
          operator: Exists
        - effect: NoSchedule
      ```
      ```

```
        key: nvidia.com/gpu
        operator: Exists
```

- Have in mind that the Deployment manifest already specify quantity of replicas, which means how many times pod specifications will be duplicated, which means that the count of GPU Core you request to access by resources.requests in your pod will be multiplied by count of replicas specified in the Deployment manifest
  - ```
    spec:
    ...
      replicas: 4
      template:
      spec:
      ....
           containers:
           ...
             resources:
              requests:
                nvidia.com/gpu: 4
               limits:
                nvidia.com/gpu: 4
    ```

- In this example you're requesting 16 GPU cores (4 replicas of 4 GPU Cores each)
  - This is an epic-app helm chart example:
  - ```
    epic-app:
      nameOverride: "hmc-canary-sandbox"
      fullnameOverride: "hmc-canary-sandbox"

      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
    ```

```
        nodeSelectorTerms:
        - matchExpressions:
          - key: gpu
            operator: In
            values:
            - "true"
  tolerations:
    - effect: NoExecute
      key: epicgames.com/gpu
      operator: Exists
    - effect: NoSchedule
      key: nvidia.com/gpu
      operator: Exists
```

Enabling GPU Support in Kubernetes

Once you have configured the options above on all the GPU nodes in your cluster, you can enable GPU support by deploying the following Daemonset:

```
$ kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device

Note: This is a simple static daemonset meant to demonstrate the basic
```

Deployment via `helm`

The preferred method to deploy the device plugin is as a daemonset using `helm`. Instructions for installing `helm` can be found here.

Begin by setting up the plugin's `helm` repository and updating it at follows:

```
$ helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
$ helm repo update
```

Then verify that the latest release (`v0.14.0`) of the plugin is available:

```
$ helm search repo nvdp --devel
NAME                          CHART VERSION   APP VERSION    DESCRIPTIO
nvdp/nvidia-device-plugin     0.14.0          0.14.0          A Helm cha
```

Once this repo is updated, you can begin installing packages from it to deploy the `nvidia-device-plugin` helm chart.

The most basic installation command without any options is then:

```
helm upgrade -i nvdp nvdp/nvidia-device-plugin --namespace nvidia-device-p

Source: https://github.com/NVIDIA/k8s-device-plugin
```

Time-Slicing GPUs in Kubernetes

The NVIDIA GPU Operator enables oversubscription of GPUs through a set of extended options for the NVIDIA Kubernetes Device Plugin. GPU time-slicing enables workloads that are scheduled on oversubscribed GPUs to interleave with one another.

This mechanism for enabling time-slicing of GPUs in Kubernetes enables a system administrator to define a set of replicas for a GPU, each of which can be handed out independently to a pod to run workloads on. Unlike Multi-Instance GPU (MIG), there is no memory or fault-isolation between replicas, but for some workloads this is better than not being able to share at all. Internally, GPU time-slicing is used to multiplex workloads from replicas of the same underlying GPU.

Create your time-slicing-config.yaml file with following content

- time-slicing-config.yaml
  - ```
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: time-slicing-config
    data:
      any: |-
        version: v1
        flags:
          migStrategy: none
        sharing:
          timeSlicing:
            renameByDefault: false
            failRequestsGreaterThanOne: false
            resources:
              - name: nvidia.com/gpu
                replicas: 4
    ```
  - helm upgrade -i nvdp nvdp/nvidia-device-plugin --namespace nvidia-device-plugin --create-namespace --version 0.14.0 --set-file config.map.config=time-slicing-config.yaml

```
```

- So far we've tested this timeSlicing configuration only works properly in gpu worker-nodes method for node provisioned by karpenter it's not working yet, they're showing only the physical GPU Cores provisioned with the instance.
- For more specific information you can look at the following links
  - https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/gpu-sharing.html
  - https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/archive/1.11.0/gpu-sharing.html
```