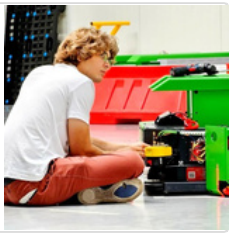
Kubernetes node Autoscaling: Cluster Autoscaler vs Karpenter

Check GitHub for helpful DevOps tools:

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics



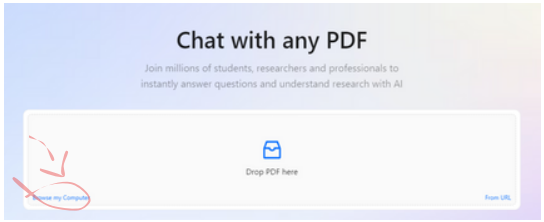
Ask Personal AI Document assistant to learn interactively (FASTER)!

1. Download PDF
2. Go to website
3. Browse file
4. Chat with Document

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesAutoscaleP2.pdf

Click there to go to ChatPdf website 2

3



Ask questions about document! 4

Complety new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

https://www.hackthebox.com/

What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

Kubernetes Autoscaling: Cluster Autoscaler

1) What is Cluster Autoscaler (CA)?

The Cluster Autoscaler automatically adjusts the size of a Kubernetes cluster by scaling the number of worker nodes. It ensures that the cluster has enough nodes to meet workload demands while minimizing costs by removing underutilized nodes.

2) Key Benefits of Cluster Autoscaler

- **Efficient Resource** Utilization: Automatically adds or removes nodes based on workload needs, ensuring efficient use of infrastructure.
- **Cost Optimization:** Reduces costs by scaling down unused nodes when workloads decrease.
- **Node Group Flexibility:** Supports scaling for multiple node groups, enabling workload-specific optimizations.

3) How Does Cluster Autoscaler Work?

Cluster Autoscaler works in conjunction with cloud providers (e.g., AWS, GCP, Azure) to dynamically provision and terminate virtual machines, ensuring the cluster maintains the desired state for workloads.

4) Setup EKS cluster

Quickest way to create EKS cluster is by use of eksctl. Install AWS CLI on your machine and log into your AWS account. Then create cluster:

```
eksctl create cluster --name my-cluster --version 1.30 --managed --asg-access
```

Cluster will use nearest region, use its VPC, create large EC2 instances. Specify version 1.30. Paramtere --asg-access is used to grant IAM permissions to allow the EKS cluster to manage Auto Scaling Groups (ASGs). Eksctl will automatically create nodegroup.

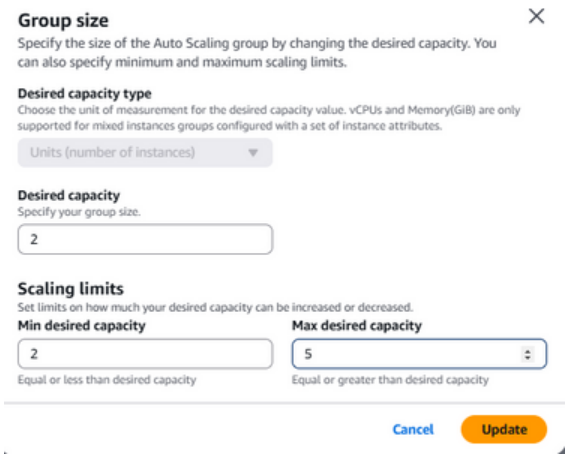
downlaad autoscaler:

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/ClusterAutoscaler/cluster-autoscaler-autodiscover.yaml
```

Deploy it

```
kubectl apply -f cluster-autoscaler-autodiscover.yaml
```

Go to EC2 service in AWS, then to Auto scaling groups and change max nodes number in group, so Cluster Autoscaler can actually scale more than default limit:



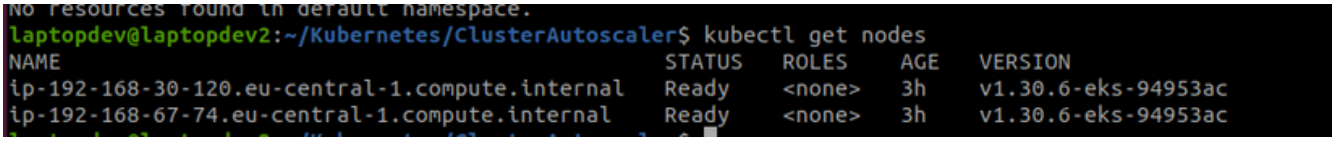
5) Test Cluster Autoscaler

Download deployment file

```
curl -O
https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/ClusterAutoscaler/cluster-autoscaler-test.yaml
```

Check number of node before deployment:

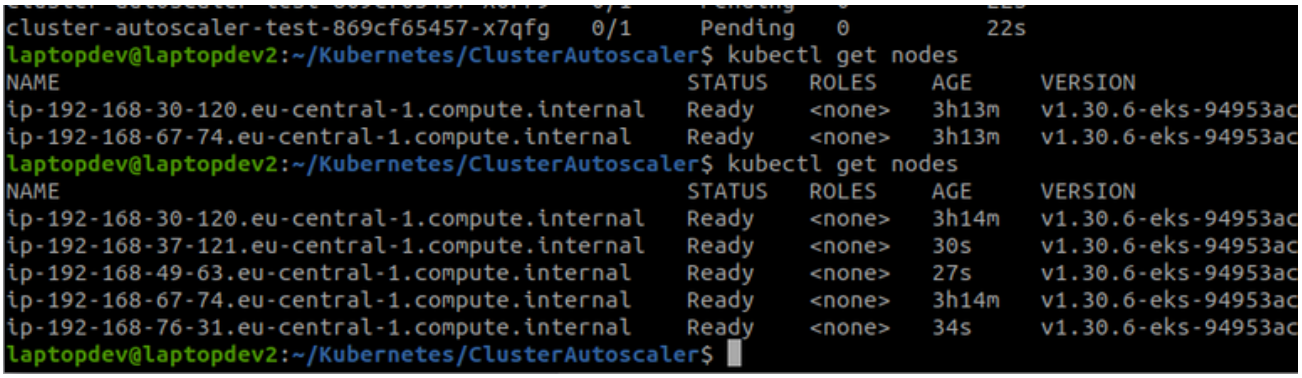
```
kubectl get nodes
```



apply deployment manifest

```
kubectl apply -f cluster-autoscaler-test.yaml
```

Number of nodes should go up:



Delete deployment so node number will go to normal

```
kubectl delete deployment cluster-autoscaler-test
```

Process of node deletion will take some time. To delete EKS cluster:

```
eksctl delete cluster --name=my-cluster
```

Thats it! You've testes CA functionality.

Kubernetes Autoscaling: Karpenter

1) What is Karpenter?

Karpenter is an open-source Kubernetes node autoscaler designed to simplify and optimize scaling by dynamically launching the right compute resources to meet workload demands.

2) Key Benefits of Karpenter

- **Fast Scaling:** Quickly launches nodes to meet sudden workload spikes, reducing scheduling delays.
- **Cost Efficiency:** Optimizes cost by selecting the most efficient instance types and consolidating workloads.
- **Flexibility:** Supports a wide range of instance types and sizes, including Spot Instances, to maximize resource flexibility.
- **Simplified Management:** Reduces configuration complexity by automatically managing node groups and scaling policies.

3) How Does Karpenter Works?

Karpenter directly interacts with cloud provider APIs (e.g., AWS EC2) to provision and decommission instances, leveraging diverse compute options like Spot Instances and Reserved Instances for optimal performance and cost efficiency.

Kubernetes Autoscaling: Karpenter vs CA comparison

1) Key Benefits of Karpenter Over Cluster Autoscaler

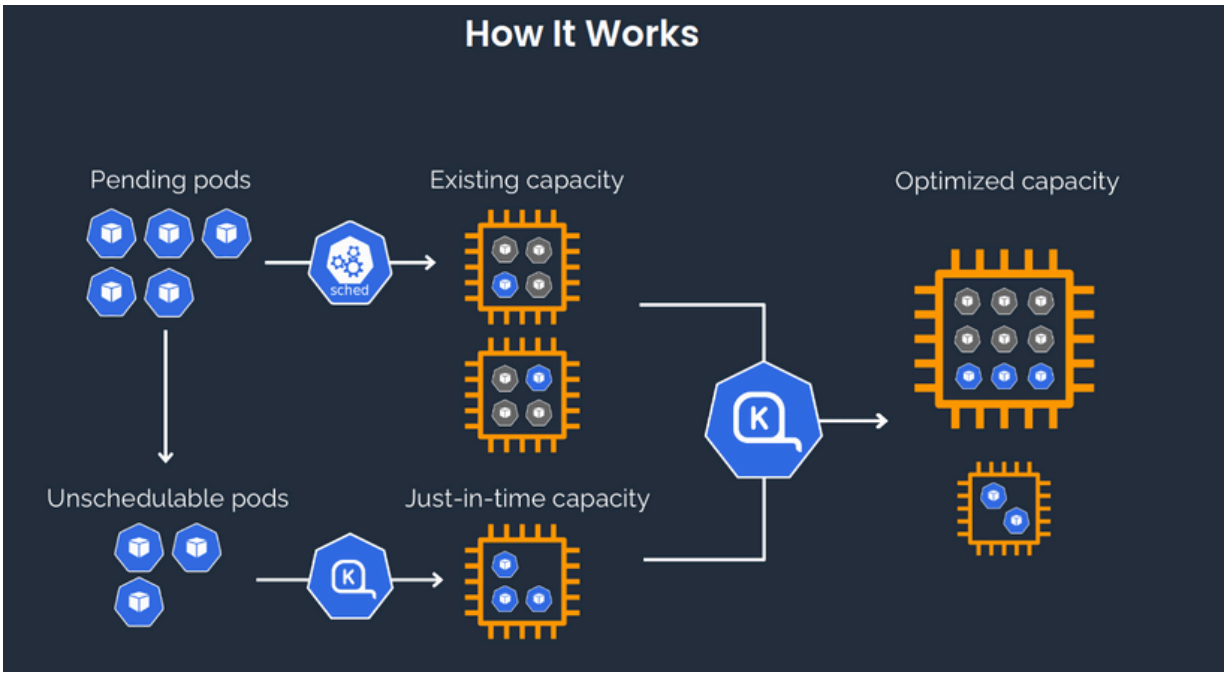
- **Faster Scaling:** Karpenter responds to workload demands more quickly, reducing pod scheduling delays.
- **Cost Optimization:** Automatically chooses the most cost-effective instance types, using Spot and Reserved Instances seamlessly.
- **Improved Flexibility:** Eliminates the need for preconfigured node groups, dynamically adapting to workload-specific requirements.
- **Modern Cloud Integration:** Direct API interactions allow Karpenter to use advanced cloud features, improving scalability and reliability.

2) When to Use Karpenter vs. Cluster Autoscaler?

- **Use Cluster Autoscaler if:**
  - You have a stable, predictable workload.
  - You are using traditional scaling with predefined node groups.
  - You prefer a well-established solution with broad cloud provider support.

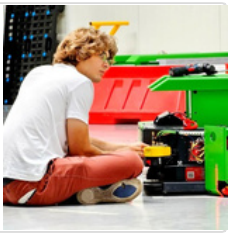
- **Use Karpenter if:**
  - You need rapid scaling for dynamic or bursty workloads.
  - You want to minimize costs with smarter instance selection, especially for Spot Instances.
  - You seek simplified operations without the overhead of managing node groups.

This image from official Karpenter website represent difference well:



3) Create EKS

Navigate towards my PDF about EKS deployment. I deployed EKS prepared for Karpenter.



4) Install karpenter

Download Karpenter installation script:

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/Karpenter/karpenter-install.sh
```

navigate to directory with script and execute:

```
bash karpenter-install.sh
```

5) Create NodePool and test Karpenter

A single Karpenter NodePool is capable of handling many different pod shapes. Karpenter makes scheduling and provisioning decisions based on pod attributes such as labels and affinity. In other words, Karpenter eliminates the need to manage many different node groups.

Download Karpenter NodePool yaml:

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/Karpenter/node-pool.yml
```

This Karpenter NodePool provisions on-demand AWS Linux (amd64) nodes from instance families c, m, or r, designed for specific workloads: c (compute-optimized), m (general-purpose), and r (memory-optimized). CPU capacity is capped at 1000, meaning the combined vCPUs of all nodes in the NodePool cannot exceed this limit, preventing over-provisioning.

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: default
spec:
  template:
    spec:
      requirements:
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
        - key: kubernetes.io/os
          operator: In
          values: ["linux"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["c", "m", "r"]
[...]
```

Deploy NodePool

```
envsubst < node-pool.yml | kubectl apply -f -
```

Download script for grafana monitoring

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/Karpenter/grafana-karpenter.sh
```

Script creates grafana dashobards automatically:

```
bash grafana-karpenter.sh
```

Now port forward grafana to port 3000. Download script:

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/Karpenter/grafana-port.sh
```

Execute. To enter grafana, type in your browser 127.0.0.1:3000

```
bash grafana-port.sh
```

Login is: admin. To get password, type:

```
kubectl get secret --namespace monitoring grafana -o jsonpath="{.data.admin-password}" | base64 --decode
```

Download deployment file:

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/Karpenter/Karpenter-deployment.yaml
```

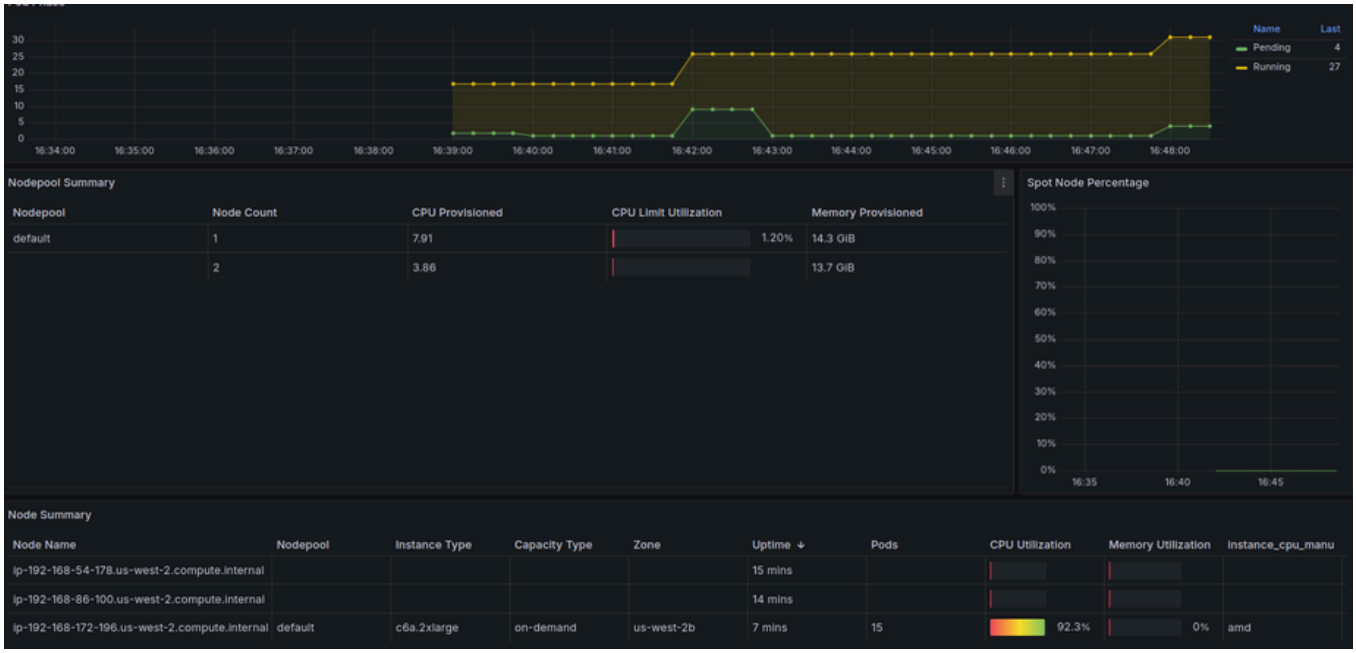
Deploy and inflate (set deployment at 5 replicas):

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

wait couple minutes, then:

```
kubectl scale deployment inflate --replicas 5
```

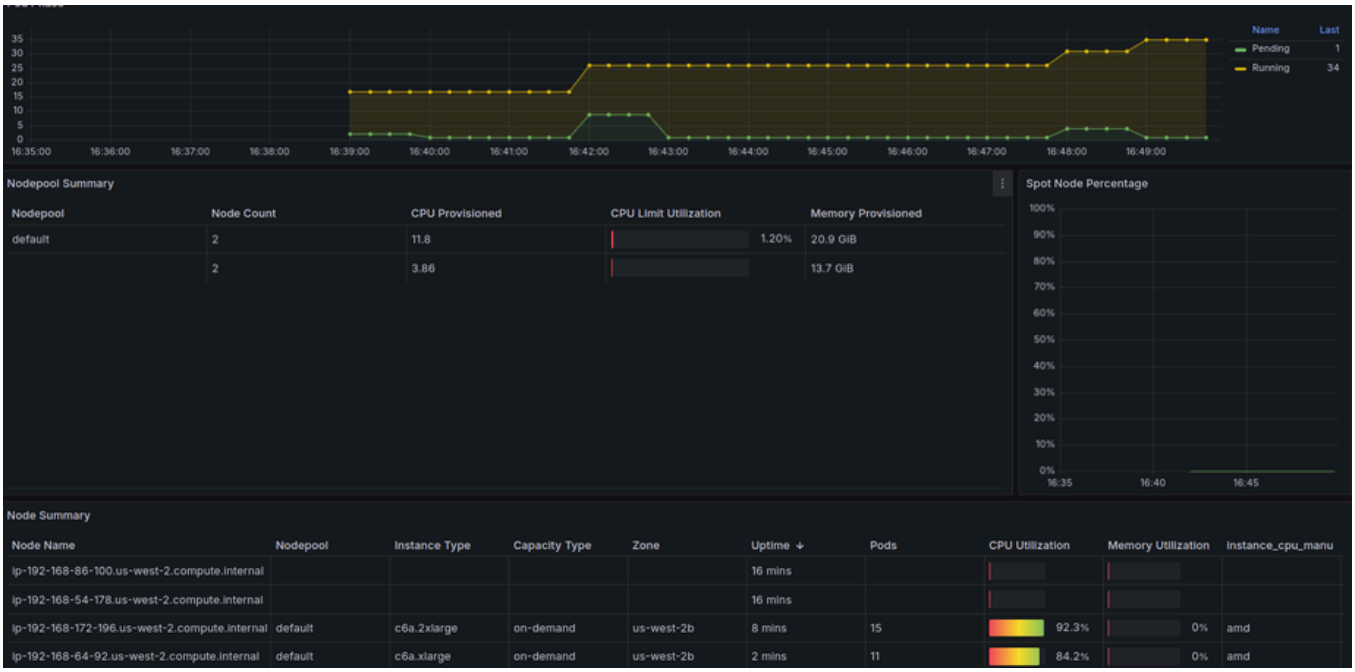
Check grafana:



At 16:39 deployment was created, at 16:42 deployment got inflated and Karpenter created third node.

Now inflate to 10:

```
kubectl scale deployment inflate --replicas 10
```

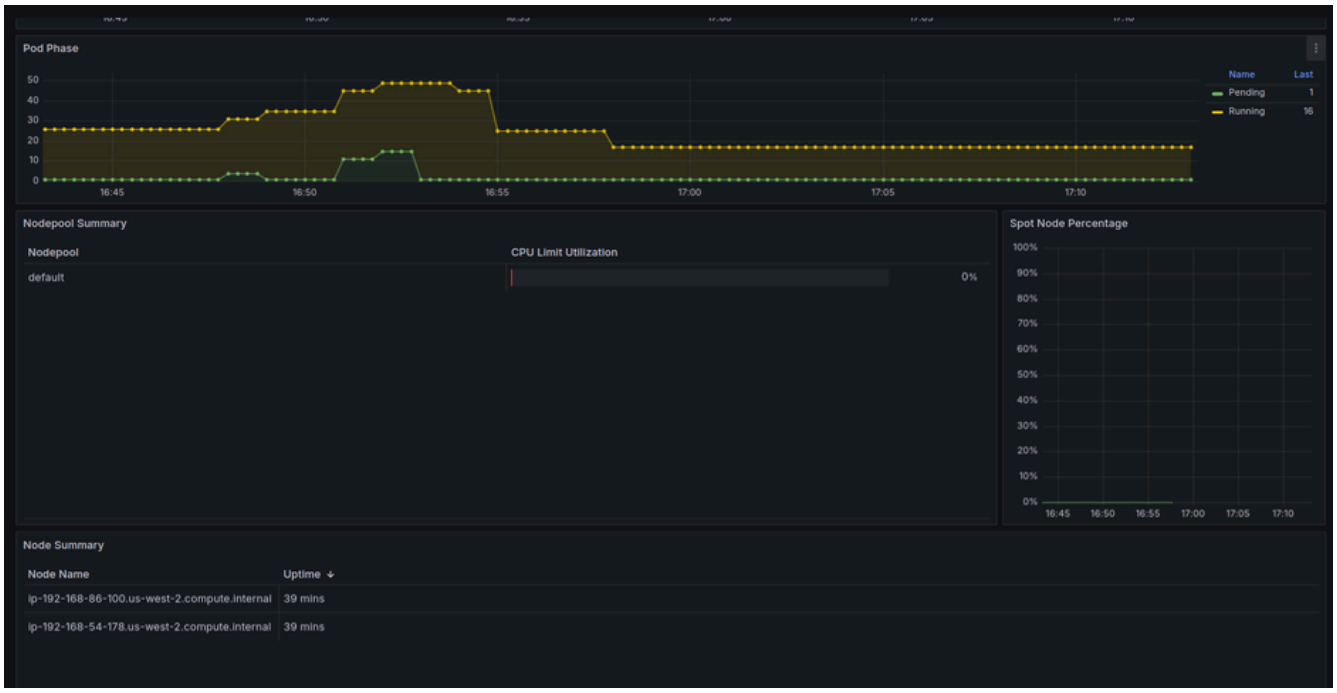


Karpenter added additional node. Inflate to 20:

```
kubectl scale deployment inflate --replicas 20
```

Delete deployment. Karpenter should destroy unused nodes:

```
kubectl delete deployment inflate
```



6) Destroy EKS cluster

Run in terminal:

```
helm uninstall karpenter --namespace "${KARPENTER_NAMESPACE}"
aws cloudformation delete-stack --stack-name "Karpenter-${CLUSTER_NAME}"
aws ec2 describe-launch-templates --filters
"Name=tag:karpenter.k8s.aws/cluster,Values=${CLUSTER_NAME}" |
    jq -r ".LaunchTemplates[].LaunchTemplateName" |
    xargs -I{} aws ec2 delete-launch-template --launch-template-name {}
eksctl delete cluster --name "${CLUSTER_NAME}"
```

common troubleshooting

1) Karpenter Not Launching Nodes

Cause: Misconfigured provisioners or insufficient AWS IAM permissions.
Solution: Verify the provisioner configuration (e.g., instance types, constraints) using kubectl describe provisioner <provisioner-name>. Check IAM permissions to ensure Karpenter can create nodes by reviewing the associated IAM role policies.

2) Cluster Autoscaler Not Scaling Up

Cause: Cluster Autoscaler is unable to find a compatible node group for the pending pods.
Solution: Confirm the instance types in the node group can support the pod resource requests. Use kubectl describe pod <pod-name> to verify requirements and ensure the node group configuration matches the Cluster Autoscaler policies.

3) Karpenter Scaling Delays

Cause: Issues with the event-driven architecture or webhook communication.
Solution: Inspect Karpenter logs using kubectl logs -n karpenter <karpenter-pod-name>. Check the webhook and controller configurations for errors.

4) Check my Kubernetes Troubleshooting series:

Learn more about Kubernetes

Check Kubernetes and piyushsachdeva - great docs!

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27

Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

https://kubernetes.io/docs/setup/

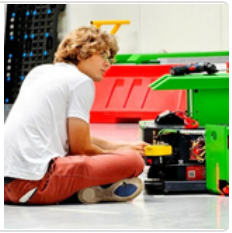
Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics



PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!