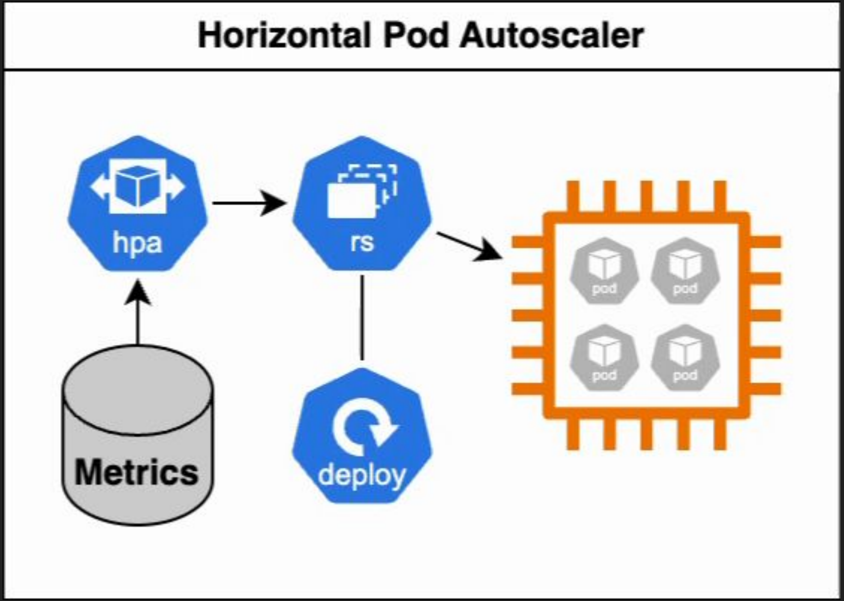
Scaling Your Kubernetes Clusters Without Going Broke

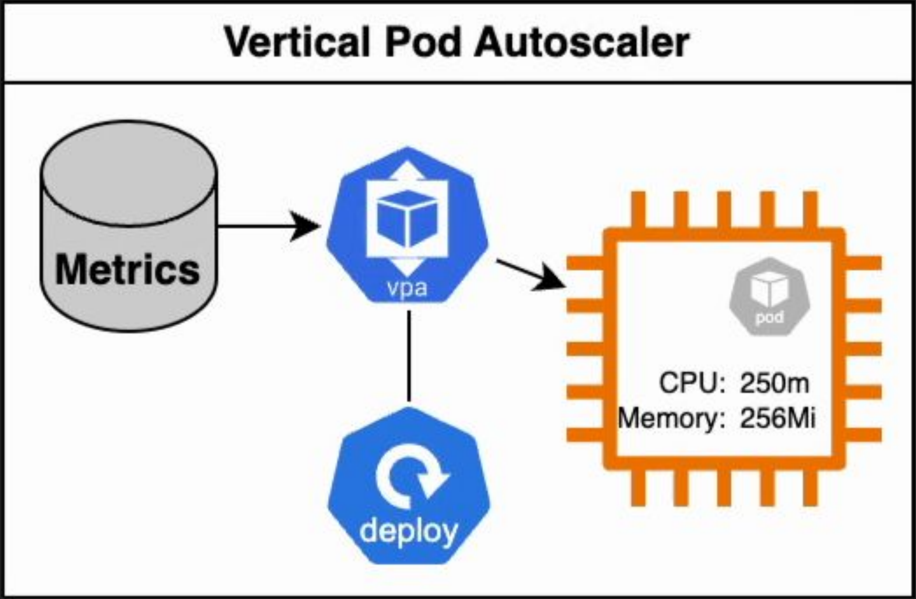
Joe Allen

Staff Site Reliability Engineer at Subsplash

Agenda

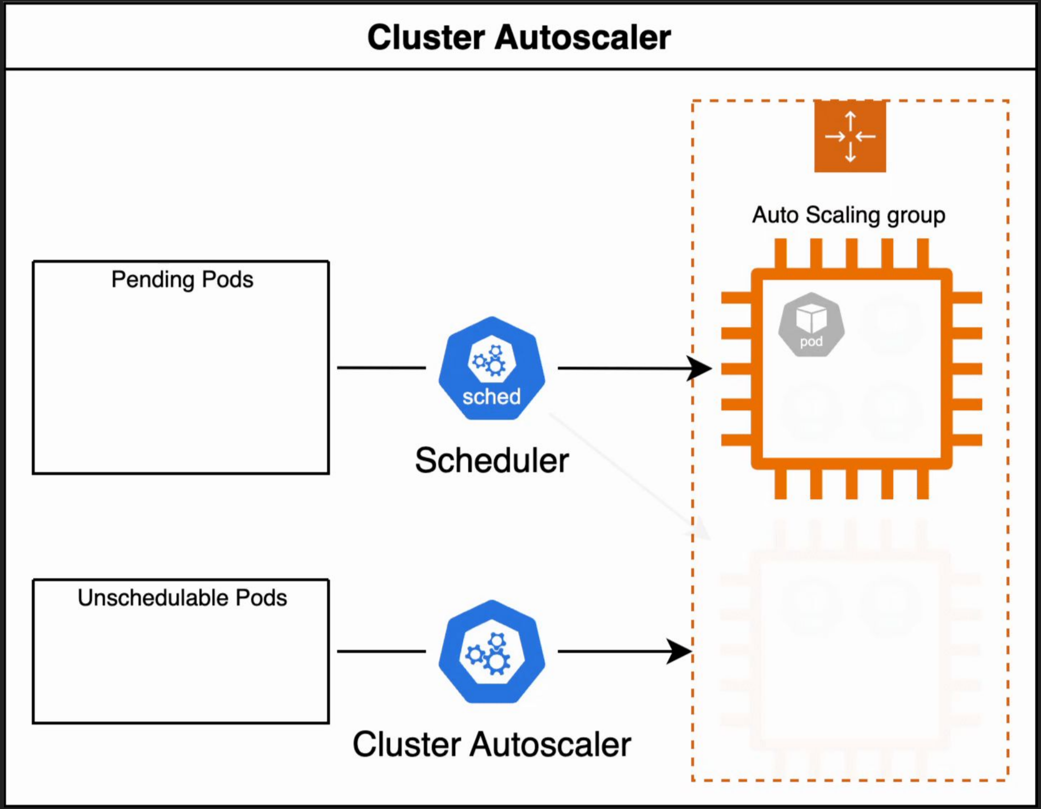
Kubernetes Autoscaling HPA/VPA



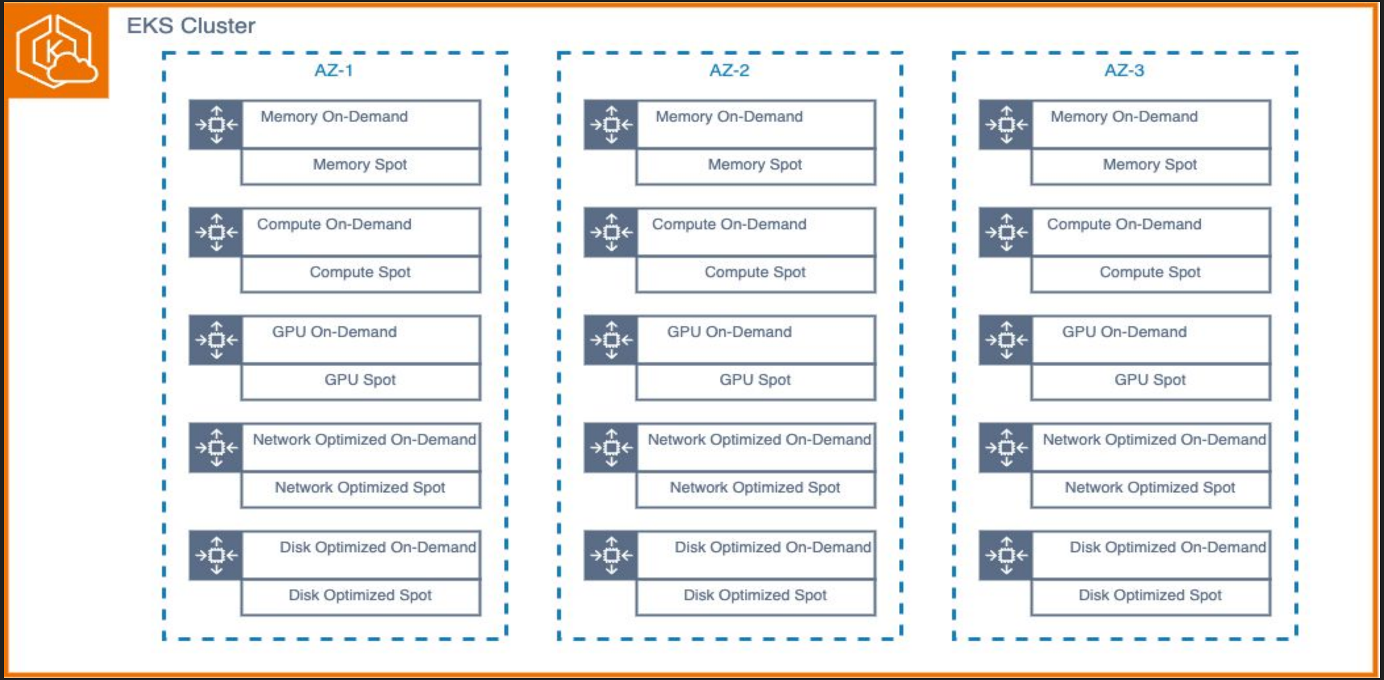


Kubernetes Autoscaling CAS

- Challenge to configure and maintain
  - Multiple Availability Zones
  - Instance types are inflexible
  - Spot capacity
  - Low cluster utilization
  - Slow to scale
- VPA may break a working deployment
- Slows innovation



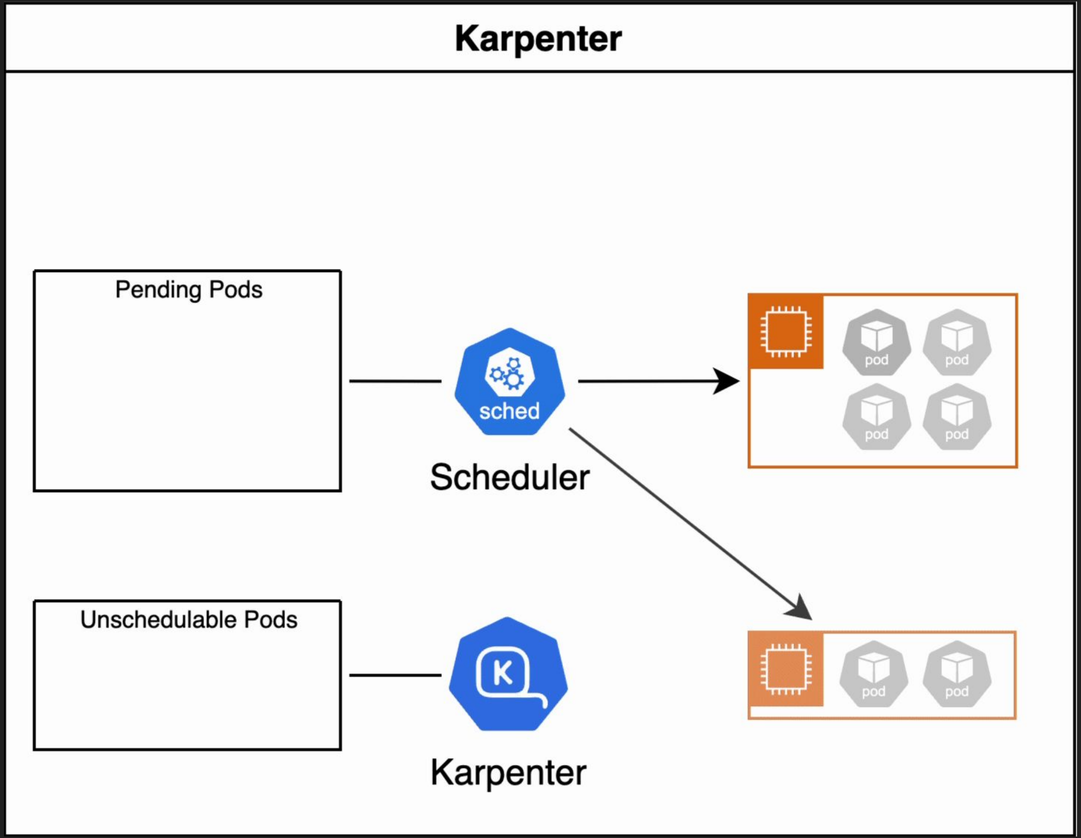
Kubernetes Autoscaling CAS (AZ)



The Solution - Karpenter

- High-Performance Autoscaler
- Can be ran standalone or alongside CAS
- Simplifies configuration, giving you the right node at the right time
- Dynamically chooses the best-suited node for unschedulable pods
- Automatically consolidates nodes and removes nodes that are no longer needed
- Open-Source - Donated to CNCF in 2023
- Built for AWS, but designed to work with other cloud providers
    - Provider for Azure is in Beta
    - Provider for GCP is in development, but still pre-Alpha

How Does It Work



Karpenter Concepts

```yaml
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
 name: default
spec:
 amiFamily: AL2
 amiSelectorTerms:
  - alias: al2@latest
```

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
 name: default
spec:
 weight: 100
 template:
   spec:
     requirements:
       - key: karpenter.k8s.aws/instance-category
         operator: In
         values: [c, m, r]
       - key: karpenter.k8s.aws/instance-generation
         operator: Gt
         values: ['4']
       - key: kubernetes.io/arch
         operator: In
         values: [amd64]
```

Strategies For Defining NodePools

Single

A single NodePool for all workloads

Notes:
- Simplest use-case
- All workloads must be compatible with cpu architecture or specify pod requirements

Multiple

Isolate workloads for different purposes

Notes:
- Isolate for security or stability
- Different AMI
- Team Separation

Weighted

Define preference across NodePools

Notes:
- Prioritize RI and Savings Plans
- Can be use as an alternative to taints/tolerations when isolation isn't a concern

Optimizing For Cost (RI/Savings Plan)

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
 name: savings-plan
spec:
 weight: 100
 limits:
   cpu: "20"
 template:
   spec:
     requirements:
       - key: karpenter.k8s.aws/instance-family
         operator: In
         values: [c4]
       - key: kubernetes.io/arch
         operator: In
         values: [amd64]
       - key: karpenter.sh/capacity-type
         operator: In
         values: [on-demand]
```

weight: 100 → Pools for savings plans or reserved instances need to weighted higher than other pools

cpu: "20" → Limit should match your plan commitment

values: [c4] → Must be set if using an EC2 Instance Savings Plan

values: [on-demand] → Must be set to on-demand

Optimizing For Cost (On-Demand/Spot)

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
 name: default
spec:
 weight: 50
 limits:
   cpu: "200"
 template:
   spec:
     requirements:
       - key: kubernetes.io/arch
         operator: In
         values: [amd64, arm64]
       - key: karpenter.k8s.aws/instance-category
         operator: NotIn
         values: [t]
       - key: karpenter.k8s.aws/instance-generation
         operator: Gt
         values: ['3']
       - key: karpenter.sh/instance-size
         operator: NotIn
         values: [nano, micro, small, medium]
       - key: karpenter.sh/instance-family
         operator: NotIn
         values: [c4, m4, r4]
       - key: karpenter.sh/capacity-type
         operator: In
         values: [on-demand, spot]
```

`values: [amd64, arm64]` → Support multi-arch to ensure the lowest cost possible

`values: [t]` → Restrict instances that use credits

`values: [nano, micro, small, medium]` → Avoid smaller instances

`values: [on-demand, spot]` → If splitting pools on capacity type, the spot pool should have a higher weight if using taints

Disruption

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
...
spec:
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 10m
    budgets:
    - nodes: '20%'
      reasons:
        - Empty
    - nodes: '1'
      schedule: '@daily'
      duration: 60m
      reasons:
        - Drifted
        - Underutilized
    template:
      spec:
        expireAfter: 720h
        terminationGracePeriod: 24h
```

Controls deleting nodes, budgets can provide more restrictions

Allows terminating 20% of the nodes at a time if they are empty

Limits removing underutilized or drifted nodes from being consolidated

Make sure you set terminationGracePeriod if you use expireAfter

How Karpenter Uses Affinity

```yaml
spec:
 affinity:
   nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
       nodeSelectorTerms:
         - matchExpressions:
           - key: kubernetes.io/arch
             operator: In
             values: [arm64]
     preferredDuringSchedulingIgnoredDuringExecution:
       - weight: 1
         preference:
           matchExpressions:
           - key: karpenter.sh/capacity-type
             operator: In
             values: [spot]
```

Karpenter will respect this requirement

Karpenter will treat this as a requirement, but the scheduler will still treat this as a preference

Handling Bin Packing When Time-slicing

```yaml
spec:
 affinity:
   podAffinity:
     preferredDuringSchedulingIgnoredDuringExecution:
     - labelSelector:
         matchExpressions:
         - key: app
           operator: In
           values: [my-app]
       topologyKey: kubernetes.io/hostname
       weight: 100
```

Q&A Time