Just-in-time Nodes for Any AWS EKS Cluster - Auto Scaling with Karpenter

Samuel Baruffi

Sr. Solutions Architect @ aws

Agenda

- EKS Overview
- Kubernetes Autoscaling Overview
- Customer Challenges
- Karpenter Overview
- Karpenter Demo

EKS Overview

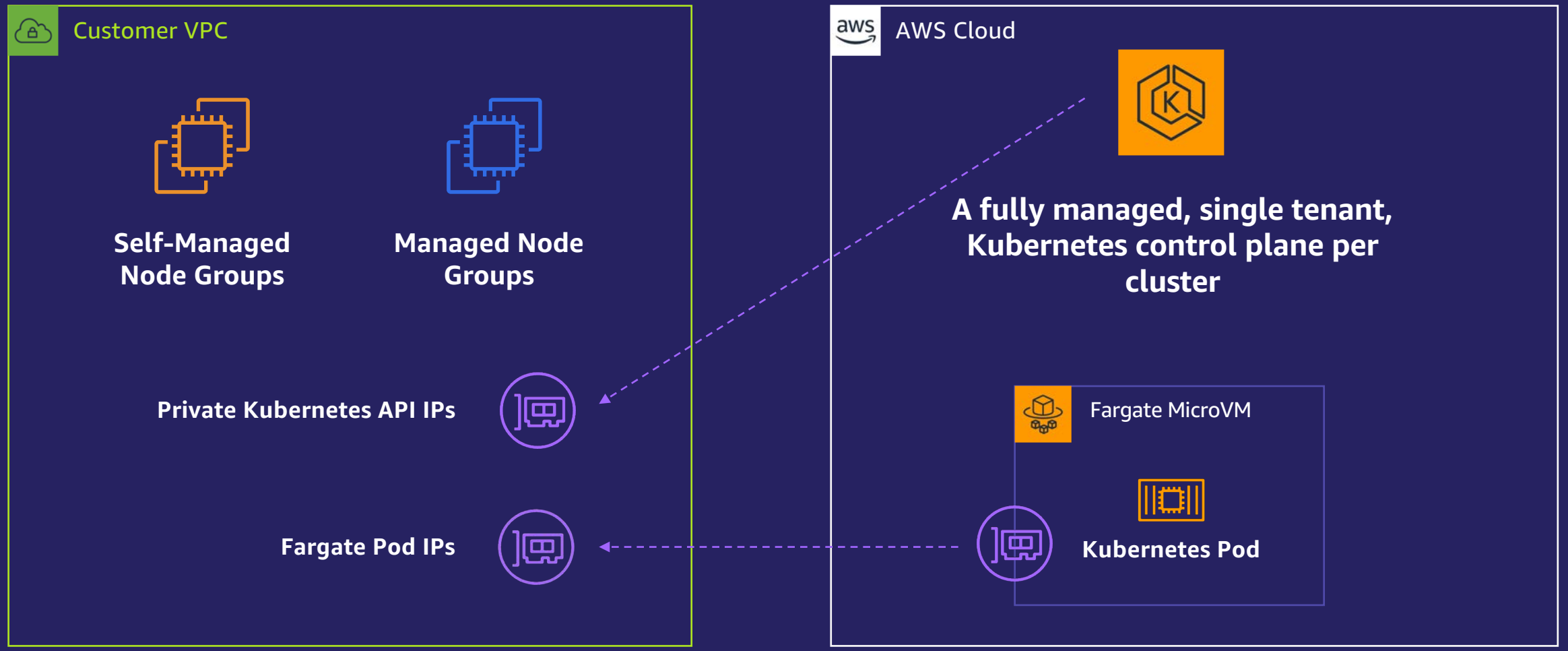
Amazon EKS is the most trusted and secure way to run Kubernetes

Amazon **EKS**

- EKS runs vanilla Kubernetes. EKS is upstream and certified conformant version of Kubernetes (with backported security fixes)
- EKS supports 4 versions of Kubernetes, giving customers time to test and roll out upgrades.
- EKS provides a managed Kubernetes experience for performant, reliable, and secure Kubernetes.
- EKS makes Kubernetes operations, administration, and management simple and boring.

Amazon EKS enables you to build reliable, stable, and secure applications in any environment.

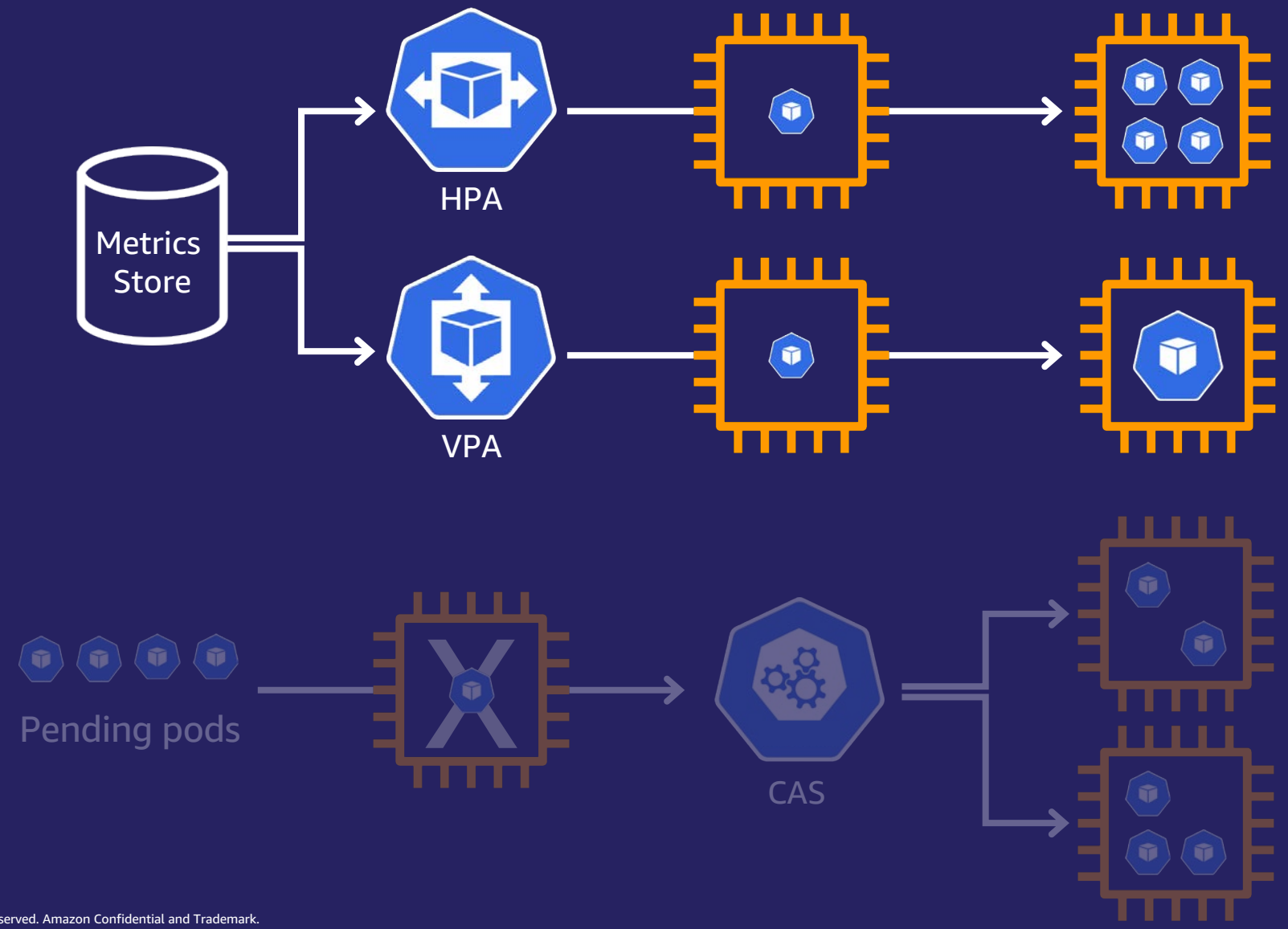
Amazon EKS High Level Architecture



Kubernetes Autoscaling

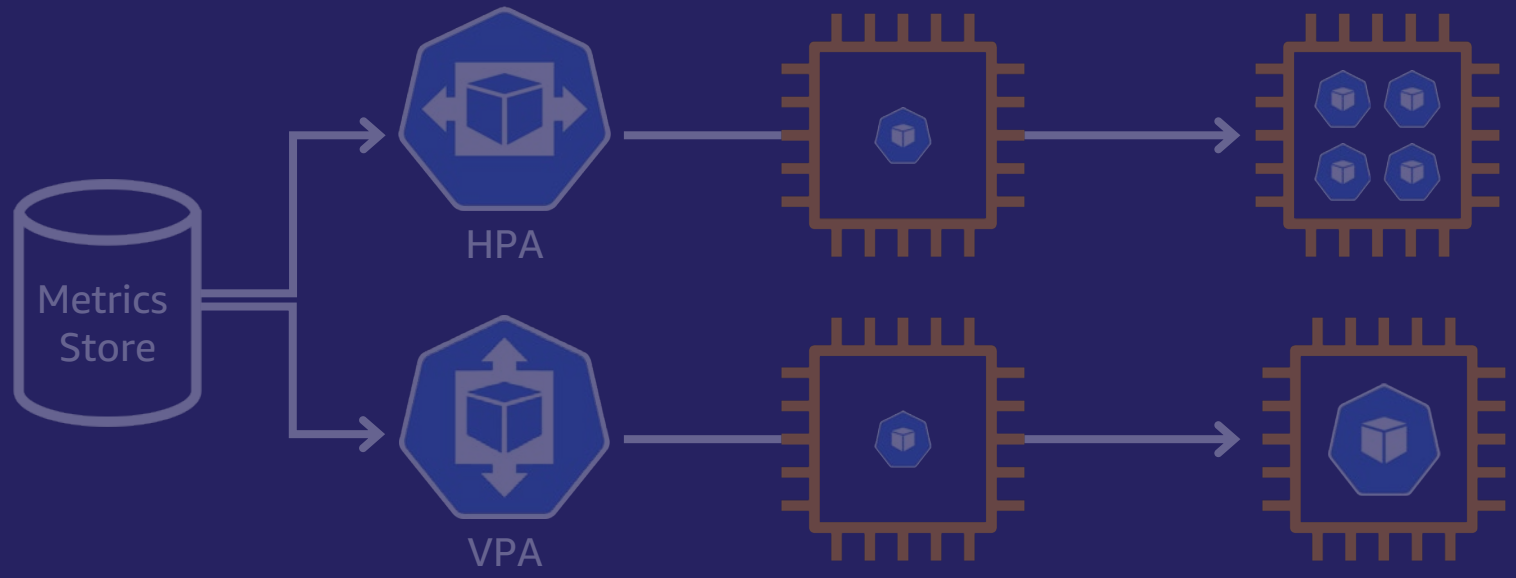
Kubernetes Autoscaling

1. Horizontal Pod Autoscaling (HPA)
2. Vertical Pod Autoscaling (VPA)
3. Cluster Autoscaler (CAS)

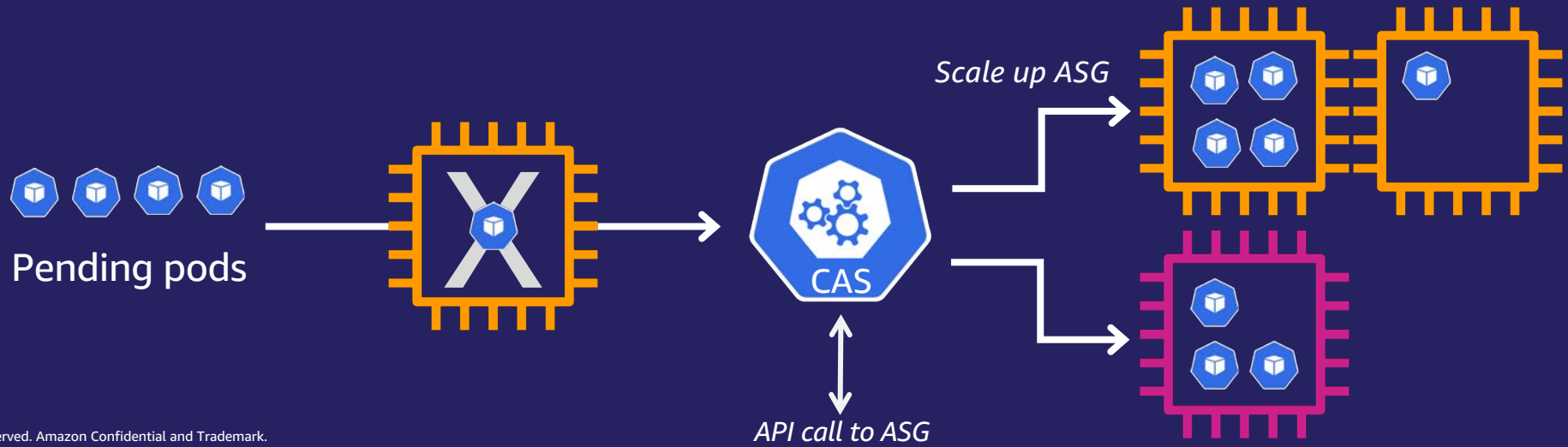


Kubernetes Autoscaling

1. Horizontal Pod Autoscaling (HPA)
2. Vertical Pod Autoscaling (VPA)



3. **Cluster Autoscaler (CAS)**

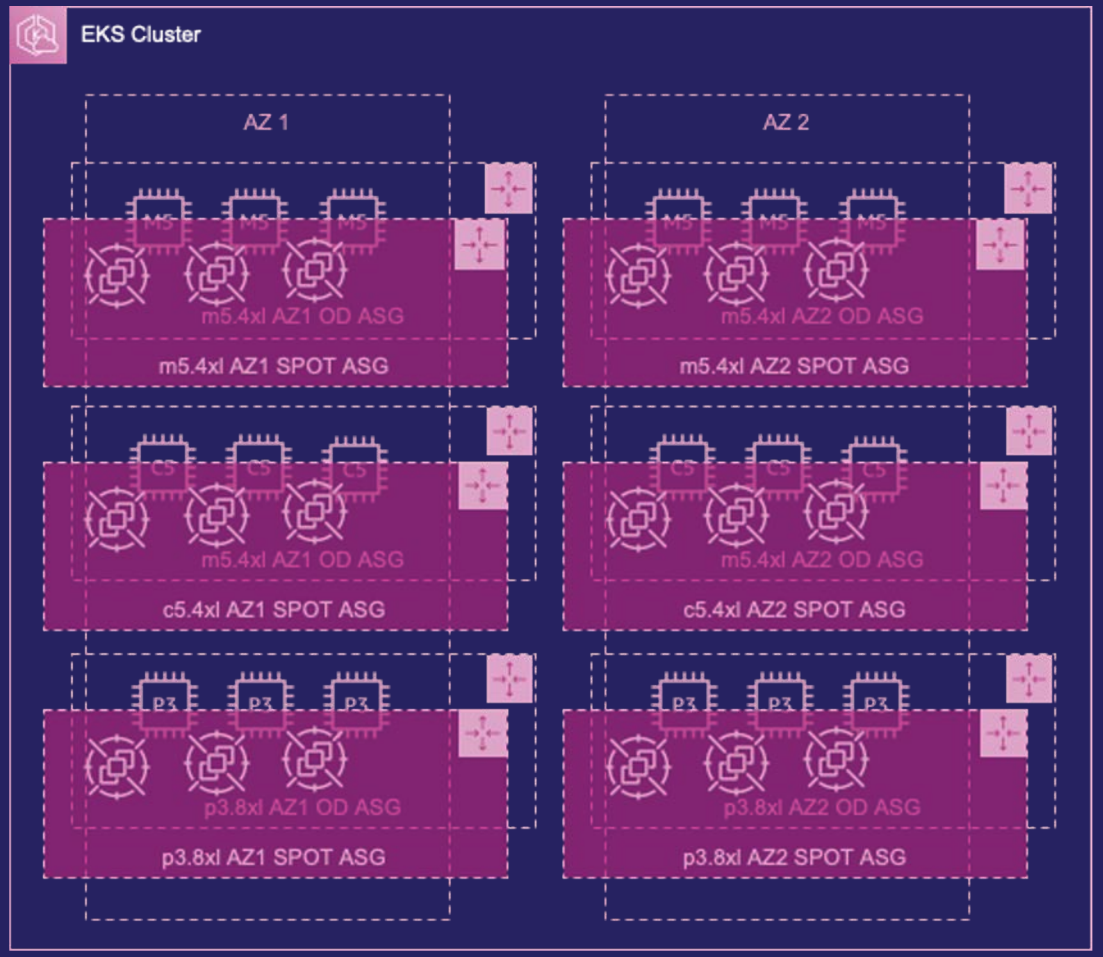


Customer Challenges

Nearly *half* of AWS Kubernetes customers tell us that configuring cluster autoscaling is challenging

Node Group and Auto Scaling Group Sprawl

- Different workloads need different compute resources
- Not all workloads need to be isolated in separate clusters
- Balancing the needs of different workloads adds complexity



Slow response to capacity needs for spiky workloads

- Quickly iterate and experiment, each time spinning up many expensive accelerated instances
- Waiting to acquire and deploy capacity slows pace of innovation
- Slow scale-down performance decreases cluster efficiency

Hard to balance utilization, availability, and cost

- Hard to get high cluster utilization and efficiency simply
- Overhead of multiple ASGs across AZs for high availability
- Added overhead increases cost of operations

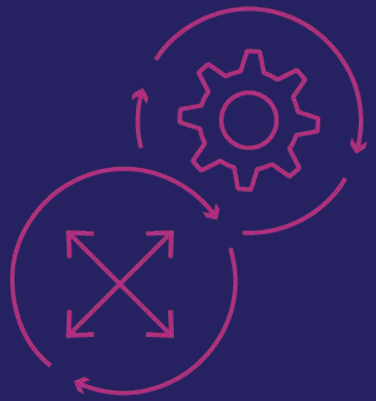
What Is Karpenter?

What is Karpenter?

Karpenter is an ***open-source***, ***flexible***, and ***high-performance*** Kubernetes cluster autoscaler.

Open source and Kubernetes-native

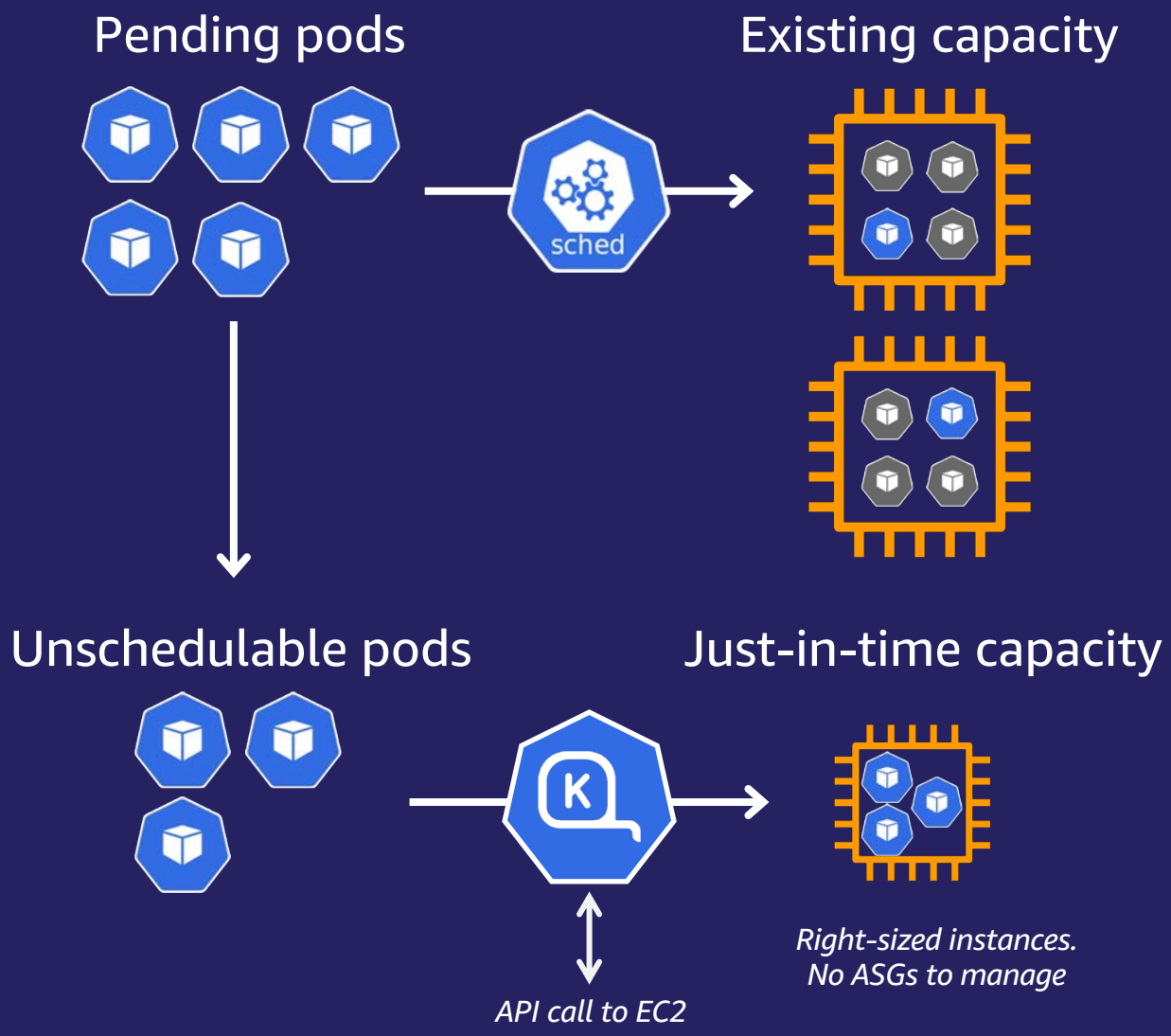
Dynamic, group-less node provisioning



Automatic node sizing

High-performance at scale

How Karpenter Works

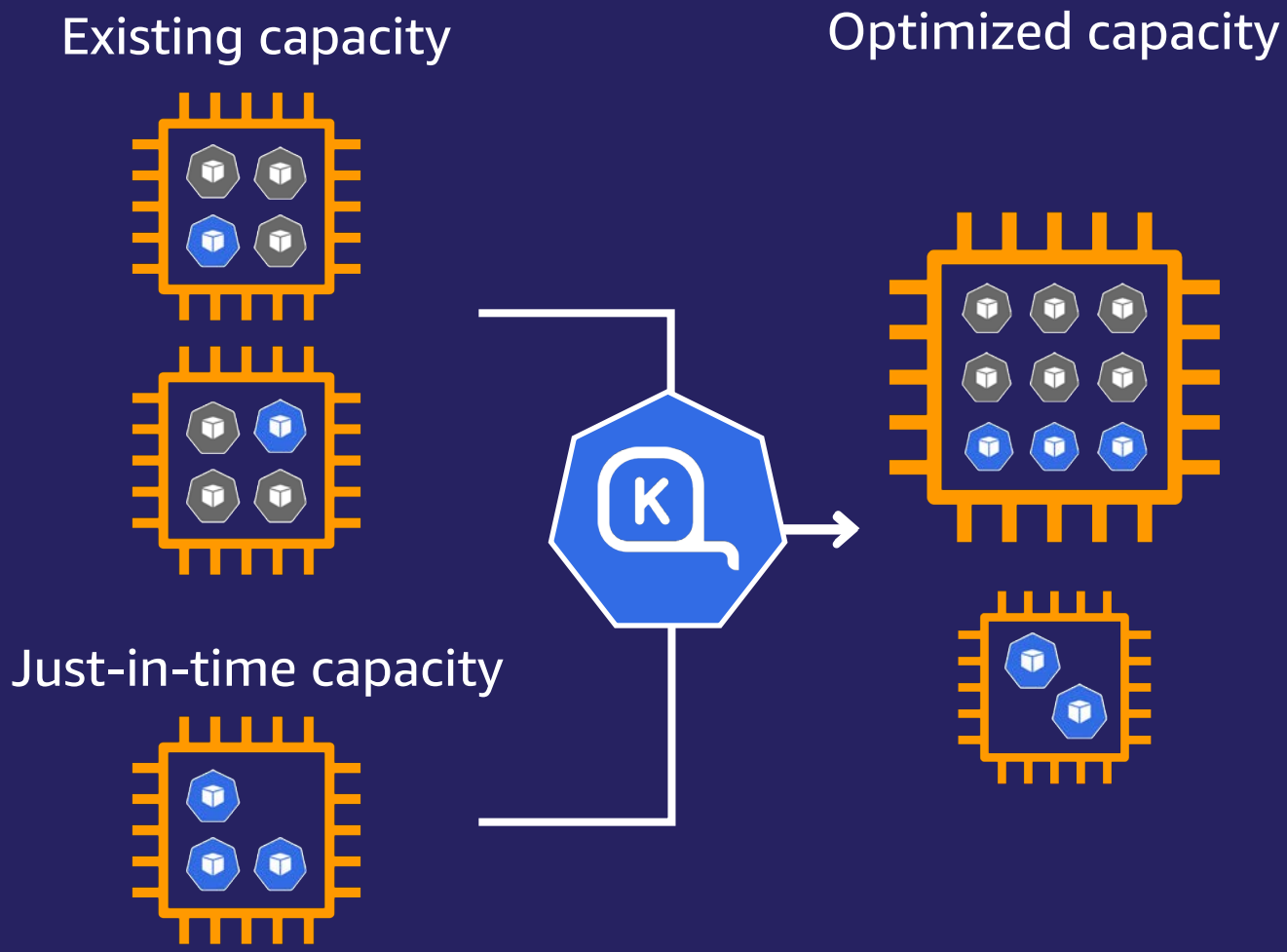


- Deeply integrated with EC2
  - EC2 Fleet API, no ASGs
- Deeply Kubernetes native
  - Watch API, Labels, Finalizers
- Automated instance selection
  - Matches workload needs to instance type

How Karpenter Works, continued…

Karpenter's **consolidation** feature looks for opportunities to improve cluster utilization over time by:

- Rescheduling running pods onto existing, under-utilized cluster capacity
- Launching new, more cost-efficient cluster capacity to replace nodes that have become under-utilized



Karpenter

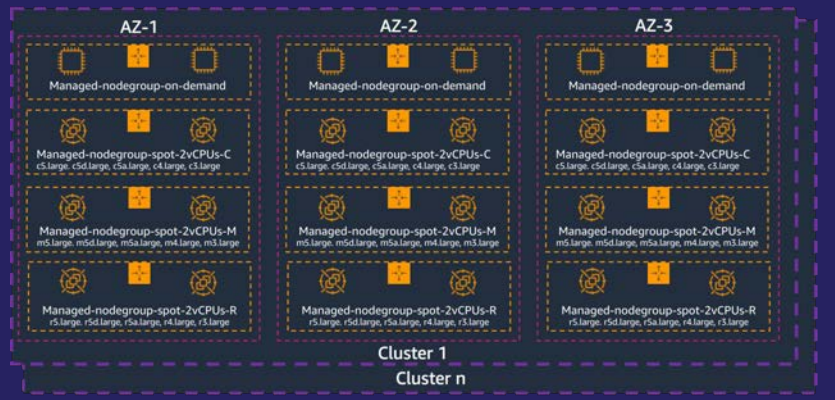
Karpenter simplifies Kubernetes infrastructure with the right nodes at the right time.

Application first infrastructure
Node provisioning based on Pod requirements

Simplified Configuration
Single configuration with On-demand and Spot purchasing options and diverse instance types

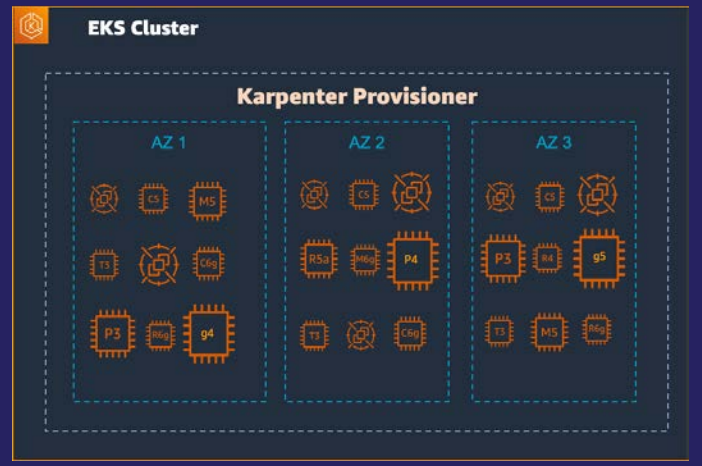
Faster Scheduling
Node provisioned faster using EC2 “Instant” fleet / Reduced cloud provider API load

Launch template support
Custom configuration and custom AMIs for your Kubernetes nodes



Best practice K8s node groups with HA

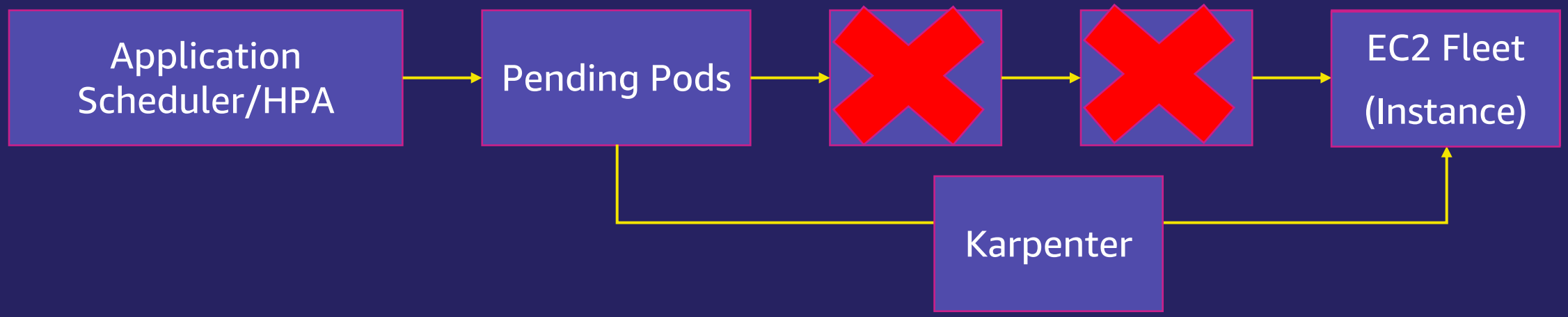
Mix of Spot + On-Demand



Karpenter right-sized groupless provisioned nodes

Mix of Spot + On-Demand

How Karpenter provisions nodes on AWS

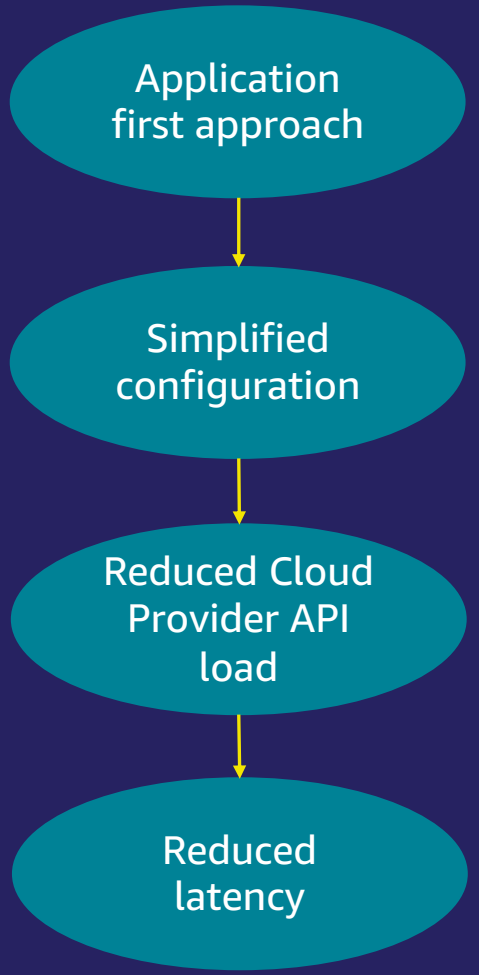


consolidates instance orchestration responsibilities within a single system

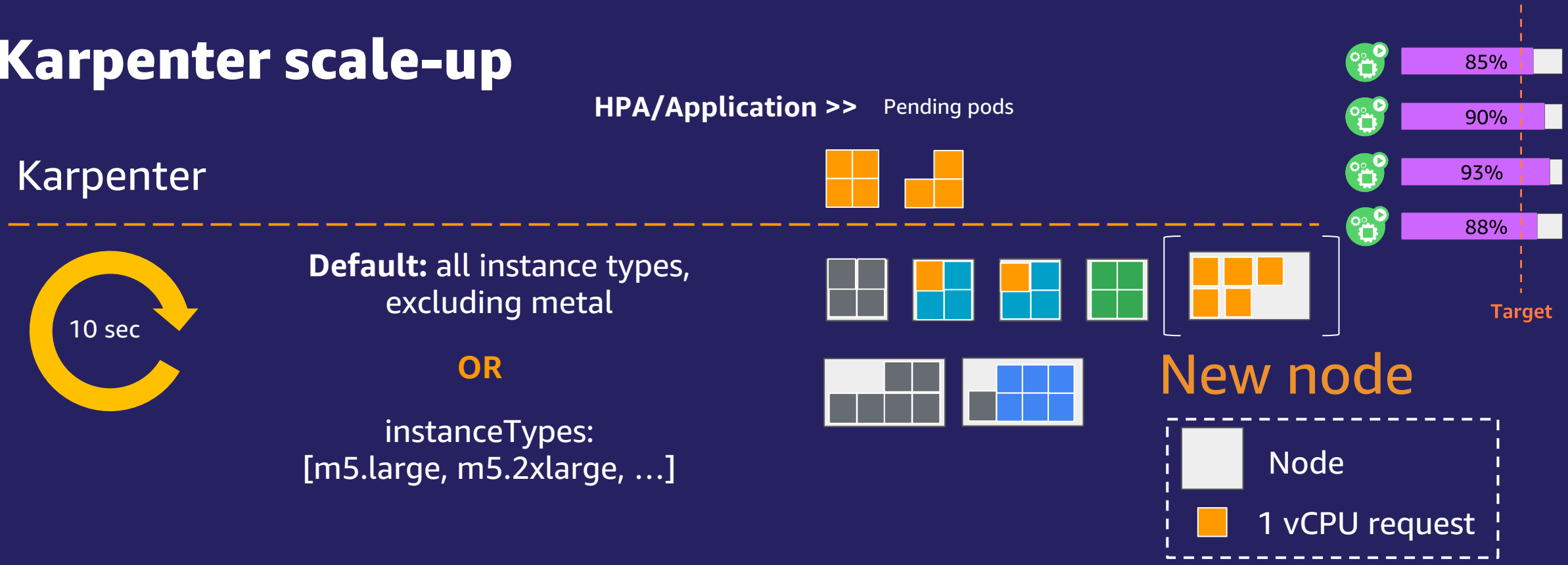
Karpenter for Groupless provisioning and Autoscaling

What if we remove the concept of node groups?

- Choose instance types from pod resource requests
- Choose nodes as per pod scheduling constraints
- Provision capacity directly - EC2 “Instant” Fleets
- Track nodes using native Kubernetes labels
- Bind pods early to provisioned nodes



Karpenter scale-up



Provisioning and scheduling decisions

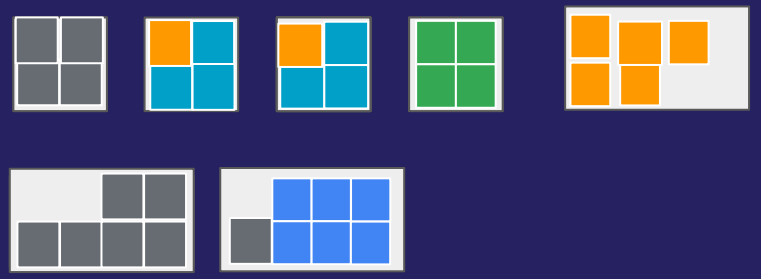
- Early binding to provisioned nodes vs. placeholder instances
- Remove scheduler version dependency

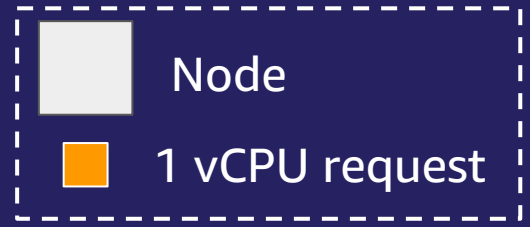
Karpenter scale-in

Karpenter

HPA/Application << Pending pods

***ttlSecondsAfterEmpty:**
seconds the controller will wait before attempting to delete a node, measured from when the node is detected to be empty



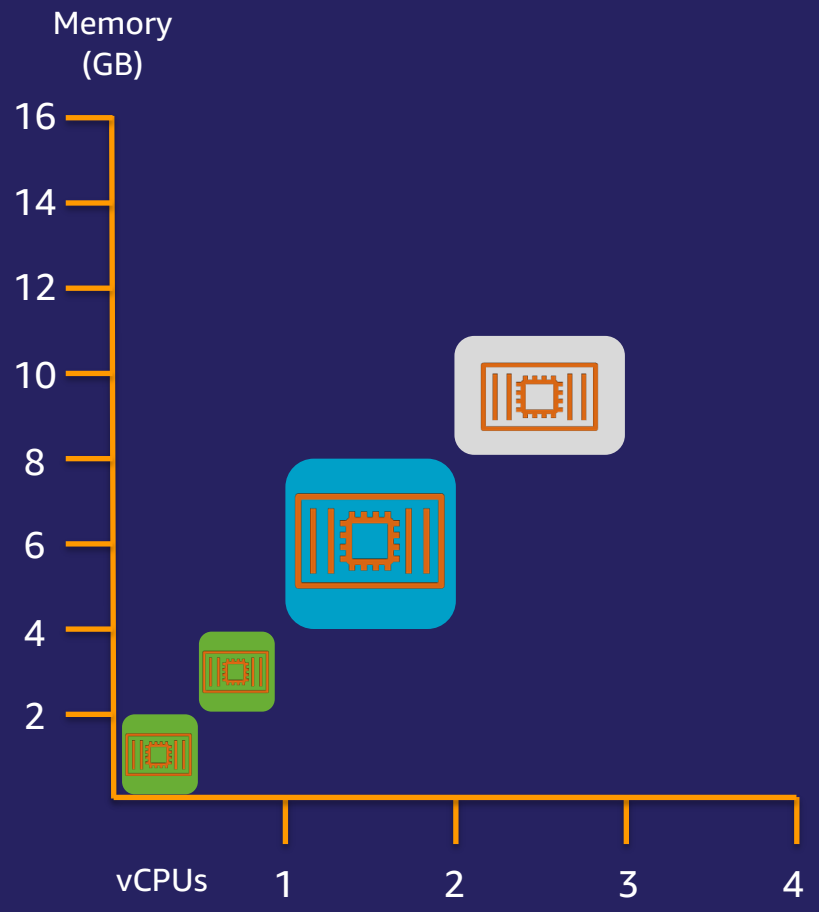


Terminations

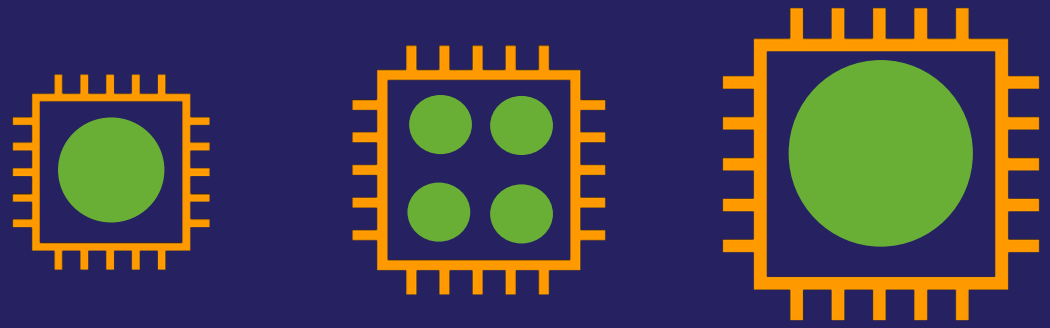
- Remove underutilized nodes (empty nodes)
- Node TTL

**If not specified, the feature is disabled and nodes will never scale down*

Karpenter Binpacking



Online binpacking while scaling up

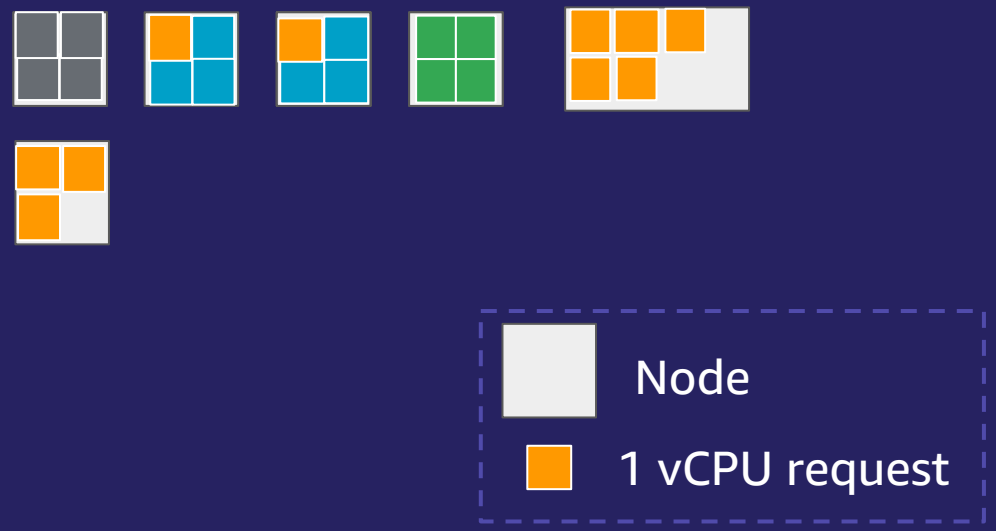


Well-known labels

karpenter.sh/capacity-type=spot
kubernetes.io/arch=arm64
topology.Kubernetes.io/zone=us-west-2a
node.kuberenetes.io/instance-type=m5.large

Karpenter Consolidation

```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: my-provisioner
spec:
  consolidation:
    enabled: true
```



Consolidation

- Deletes a node – When pods can run on free capacity of other nodes in the cluster
- Deletes a node – When node is empty (no need of setting ttSecondsAfterEmpty with Consolidation)
- Replaces a node – When pods can run on a combination of free capacity of other nodes in the cluster + more efficient replacement node

Compute flexibility – Purchase Options and CPUs

Purchase options

- Default is on-demand
- Configure on-demand and Spot purchase options
- When on-demand and Spot are configured – Spot prioritized
- Provisions on-demand when Spot constrained

```
spec:
  requirements:
    - key: karpenter.sh/capacity-type
      operator: In
      values: ["spot", "on-demand"]
```

CPU architecture

- Default is x86 instances only (amd64)
- Diversify across x86 and ARM architecture instances

```
spec:
  requirements:
    - key: node.kubernetes.io/arch
      operator: In
      values: ["arm64", "amd64"]
```

Compute flexibility – Instance types and AZs

Instance type

- Defaults to all EC2 instance types excluding metal and GPU
- Only restrict instance types if required
- Instance diversification across
  - Sizes
  - Families
  - Generations
  - CPUs

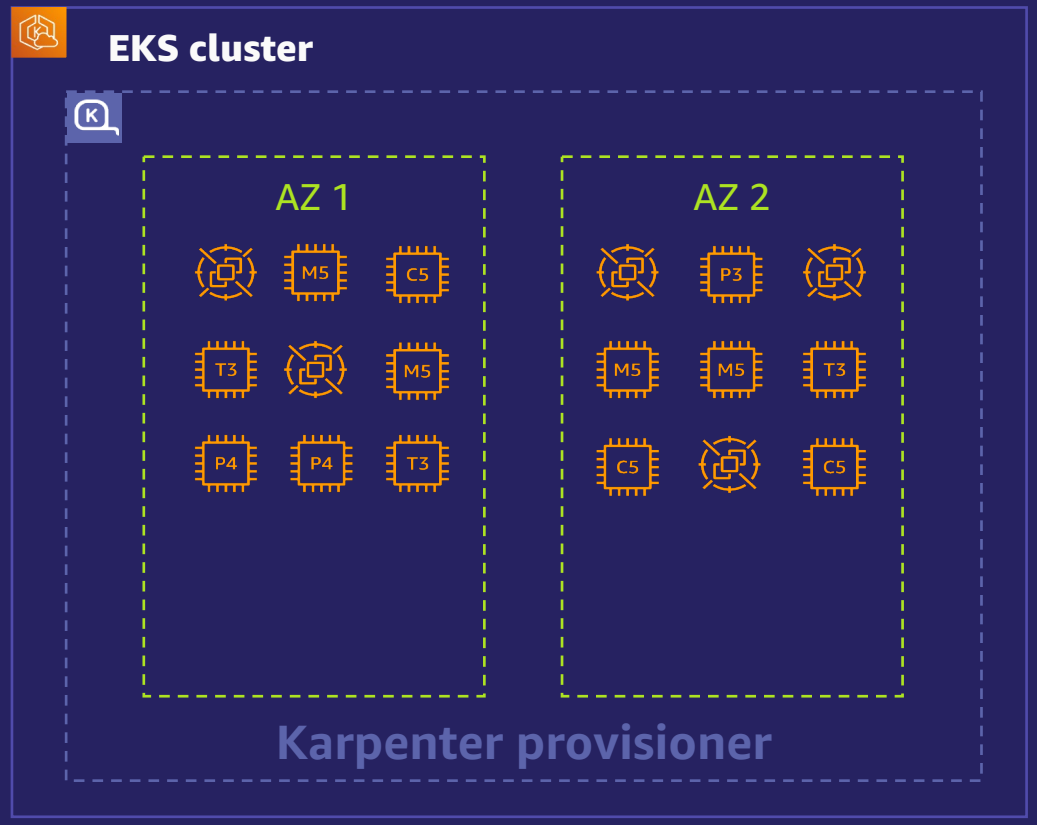
```
spec:
  requirements:
    - key: node.kubernetes.io/instance-size
      operator: NotIn
      values: [nano, tiny, small, large]
```

Availability Zone

- Defaults to all AZs
- Only restrict AZs if required

```
spec:
  requirements:
    - key: topology.kubernetes.io/zone
      operator: In
      values: ["us-west-2a", "us-west-2b"]
```

Flexible Cluster Auto Scaling with Karpenter



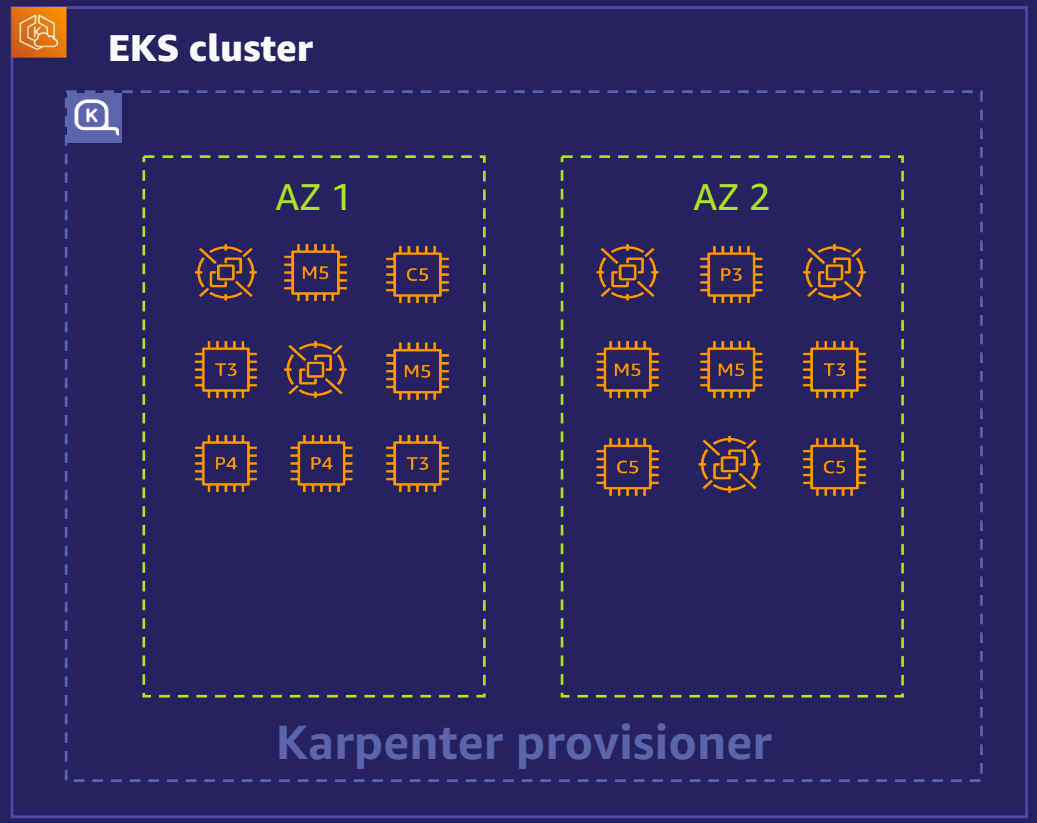
Compute is best fit to cluster workloads automatically

Example 1: Limits nodes to specific zones using on-demand and Spot

```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
          name: westzones
spec:
          requirements:
            - key: "topology.kubernetes.io/zone"
              operator: In
              values: ["us-west-2a", "us-west-2b", "us-west-2c"]
            - key: "karpenter.sh/capacity-type"
              operator: In
                values: ["spot", "on-demand"]
```

Declarative k8s custom resource definition (CRD) configuration

Flexible Cluster Auto Scaling with Karpenter



Compute is best fit to cluster workloads automatically

Example 2: Isolating Expensive Hardware

```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: gpu
spec:
  ttlSecondsAfterEmpty: 60
  requirements:
  - key: node.kubernetes.io/instance-type
    operator: In
    values: ["p3.8xlarge", "p3.16xlarge"]
  taints:
  - key: nvidia.com/gpu
    value: true
    effect: “NoSchedule”
```

Declarative k8s custom resource definition (CRD) configuration

Main takeaways

- Schedule pods to EC2 Spot Instances to optimize cost.
- Use Provisioners to ensure you are scaling nodes using Spot best practices.
- Use default Provisioner with diverse Instance Types and Availability Zones
- Use additional Provisioners for different compute constraints
- Control scheduling of your application Pods with Node Selector, topologySpreadConstraints, Taints, Tolerations and Provisioners
- Use Horizontal Pod Autoscaler and Karpenter to scale

Resources

- Karpenter Webpage and Documentation: https://karpenter.sh/
- Karpenter Github: https://github.com/aws/karpenter
- Karpenter Workshop: https://ec2spotworkshops.com/karpenter.html
- EKS Workshop Karpenter Section: https://www.eksworkshop.com/docs/autoscaling/compute/karpenter/
- Karpenter Video on Container from the Couch: https://www.youtube.com/watch?v=qawuLEMyeEw

Demo