Karpenter

CMP405-R

Running efficient Kubernetes clusters on Amazon EC2 with Karpenter

Steve Cole

Principal Specialist Solutions Architect
AWS

Brandon Wagner

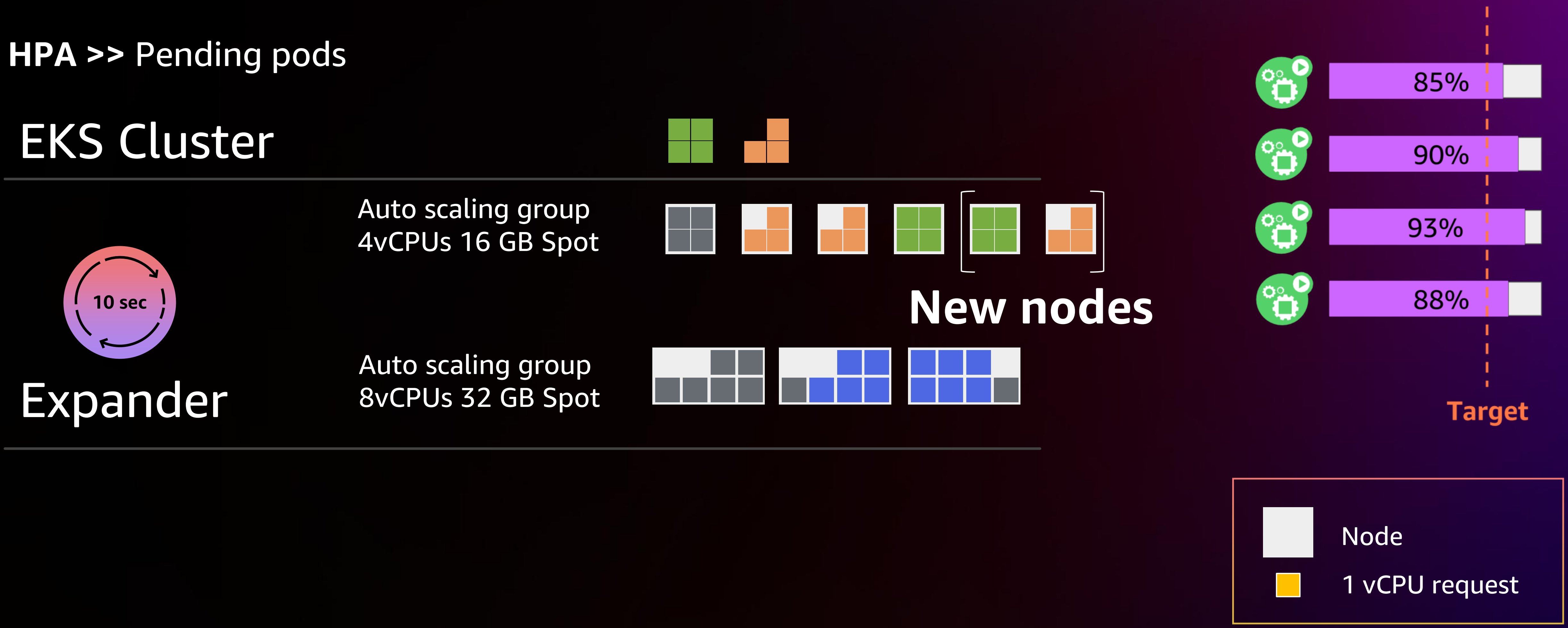
Software Development Engineer
AWS

Agenda

- What is Karpenter
- How Karpenter works
- Karpenter and Flexible Compute
- Best practices with EKS

Cluster Autoscaler scale-up

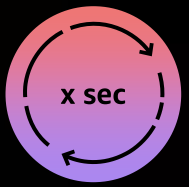
HPA >> Pending pods



Karpenter scale-up

HPA >> Pending pods

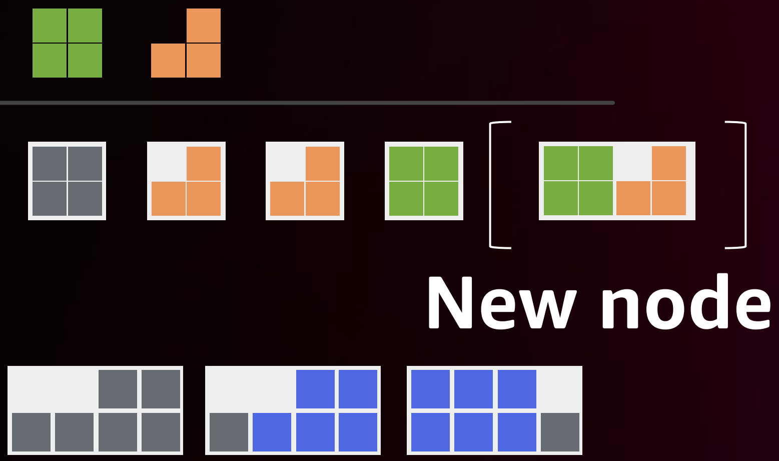
EKS Cluster

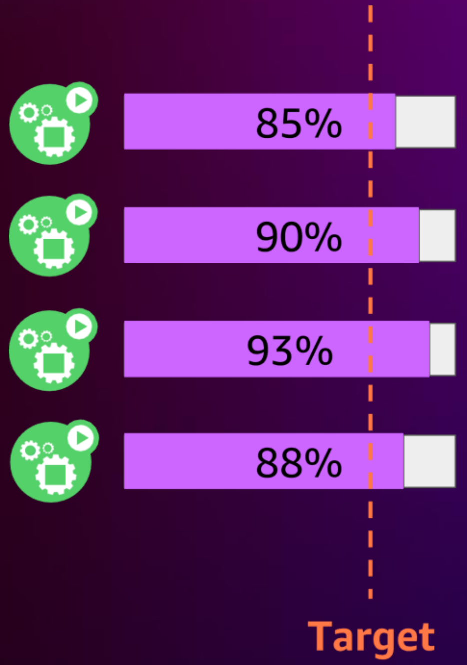


Default: all standard instance types

OR

instanceFamilies: [m5, m5a, m6i, …]





⬜ Node

🟨 1 vCPU request

Provisioning and scheduling decisions

- Works with kube-scheduler to provision the right set of nodes
- Supports all scheduling constraints: Topology Spread, Node/Pod Affinity and Anti-Affinity, etc.

Karpenter

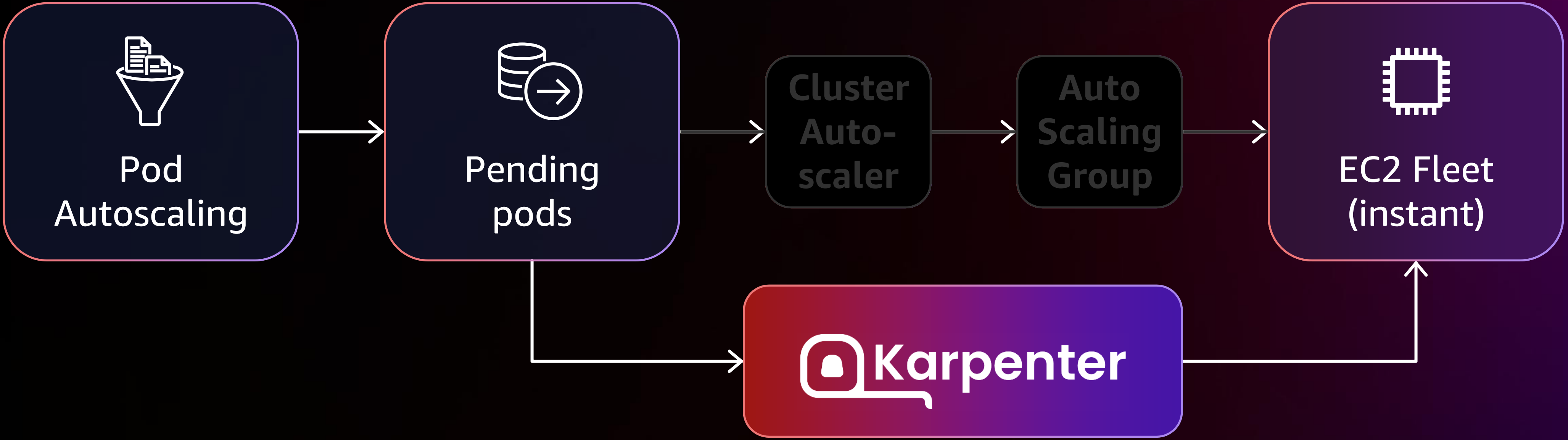
GROUPLESS PROVISIONING AND AUTO SCALING

What if we remove the concept of node groups?

- Improve the efficiency and cost of running workloads
- Simplification of configuration
- Kubernetes native
- Flexible compute built in

- Provision capacity directly with “instant” EC2 Fleets
- Choose instance types from pod resource requests
- Provision nodes using K8s scheduling constraints
- Track nodes using native Kubernetes labels

How Karpenter works

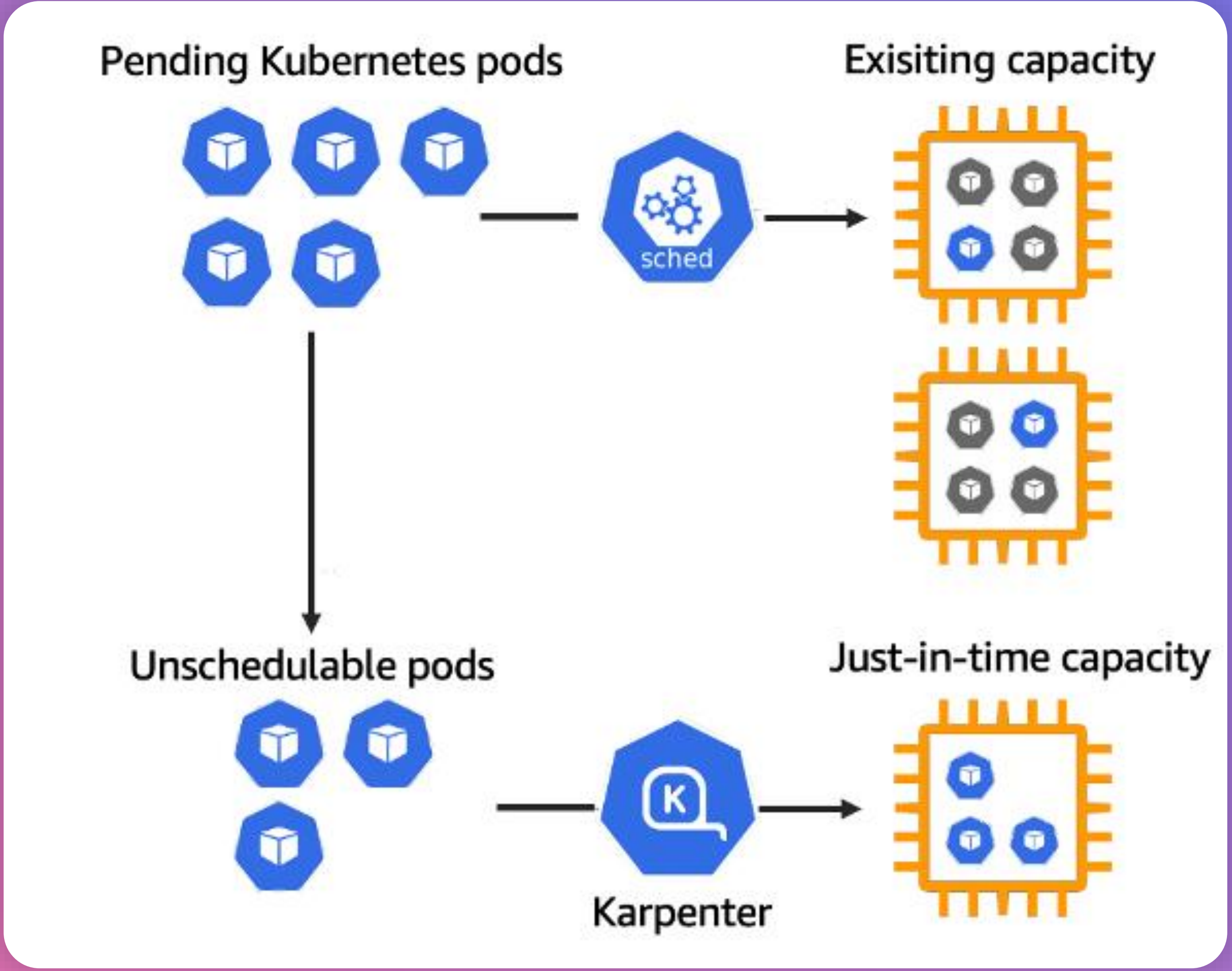


Consolidates instance orchestration responsibilities within a single system

Karpenter

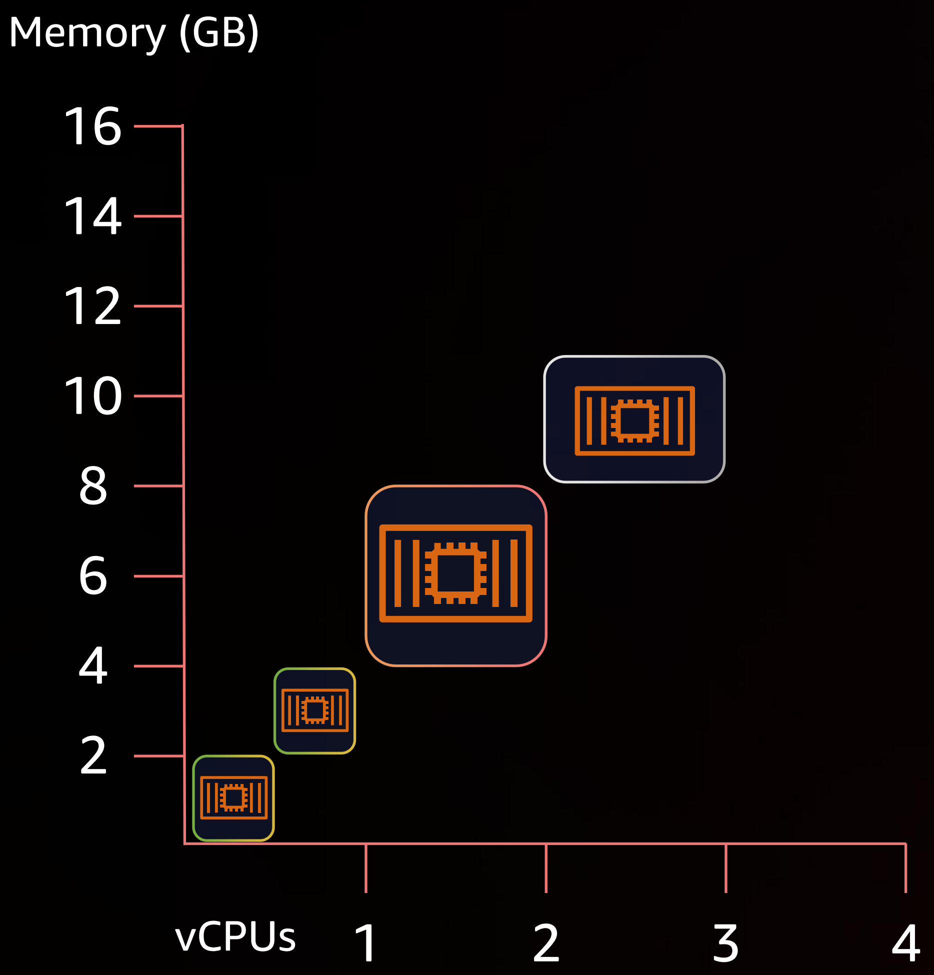
NODE PROVISIONING

- Kube Scheduler gets the first crack at scheduling pending pods. Tries to schedule on existing capacity
- Karpenter observes aggregate resource requests of unschedulable pods (set by kube scheduler) to make decisions on what instances to launch

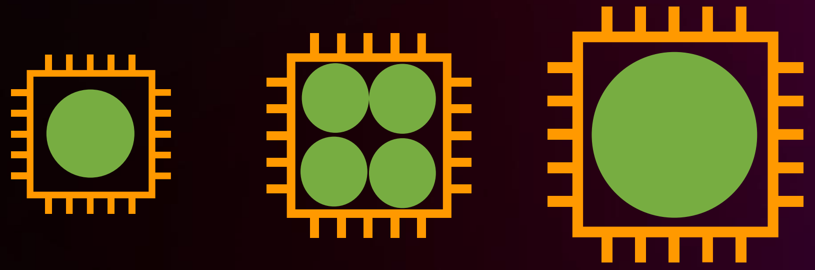


Karpenter

BINPACKING



Online binpacking



Well-known labels

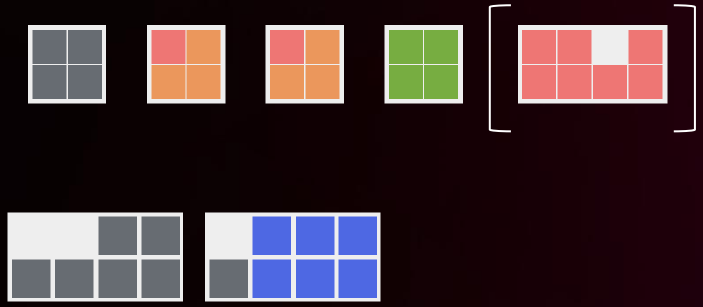
- karpenter.sh/capacity-type=spot
- karpenter.k8s.aws/instance-family=m6i
- kubernetes.io/arch=arm64
- topology.Kubernetes.io/zone=us-west-2a

Karpenter Scale-In

HPA << Pending pods

Karpenter

Consolidation actively seeks out opportunities to make the cluster more cost efficient



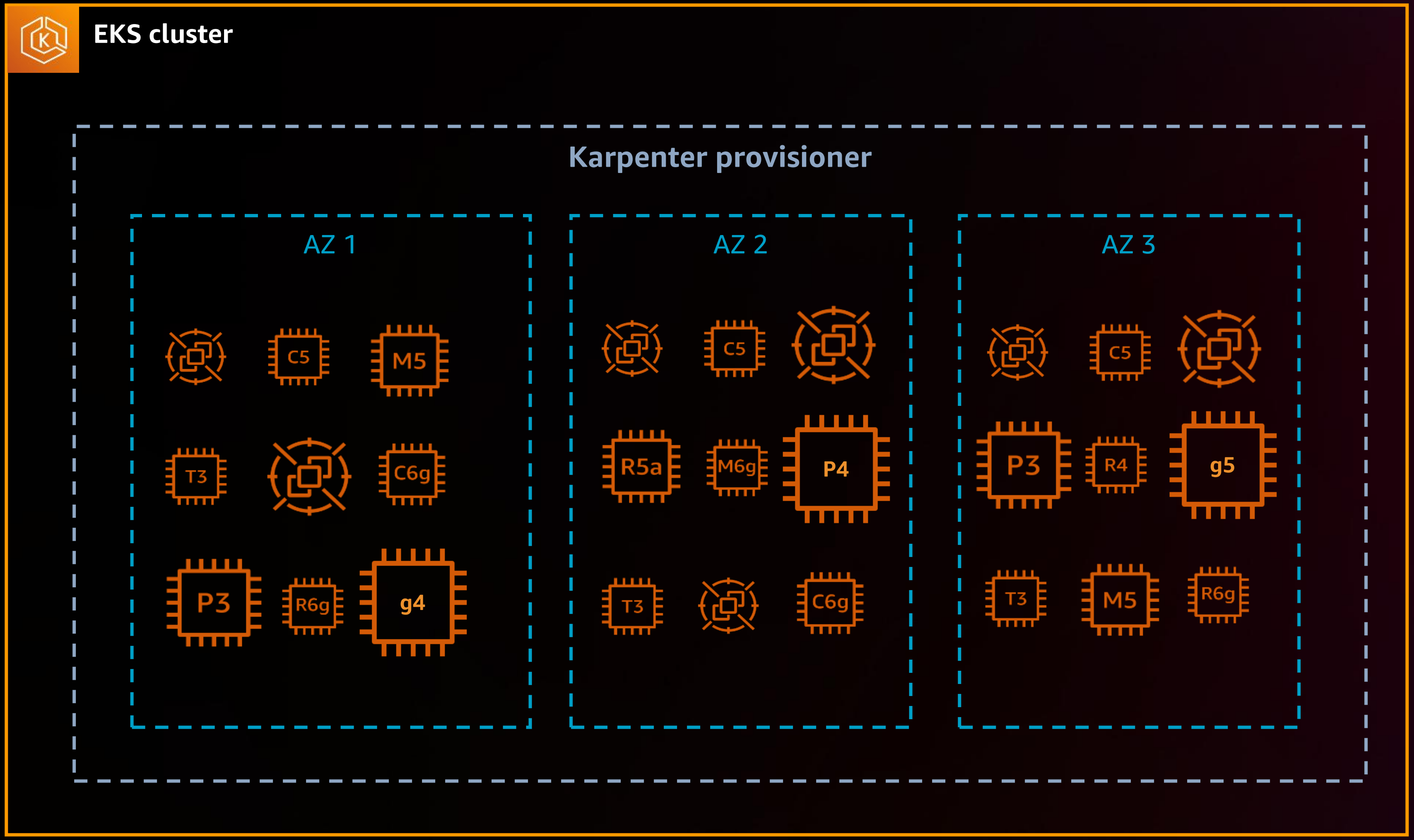
Terminations

- Replace underutilized nodes with more efficient compute
- Node Expiration TTL
- kubectl delete node with graceful draining

⬜ Node

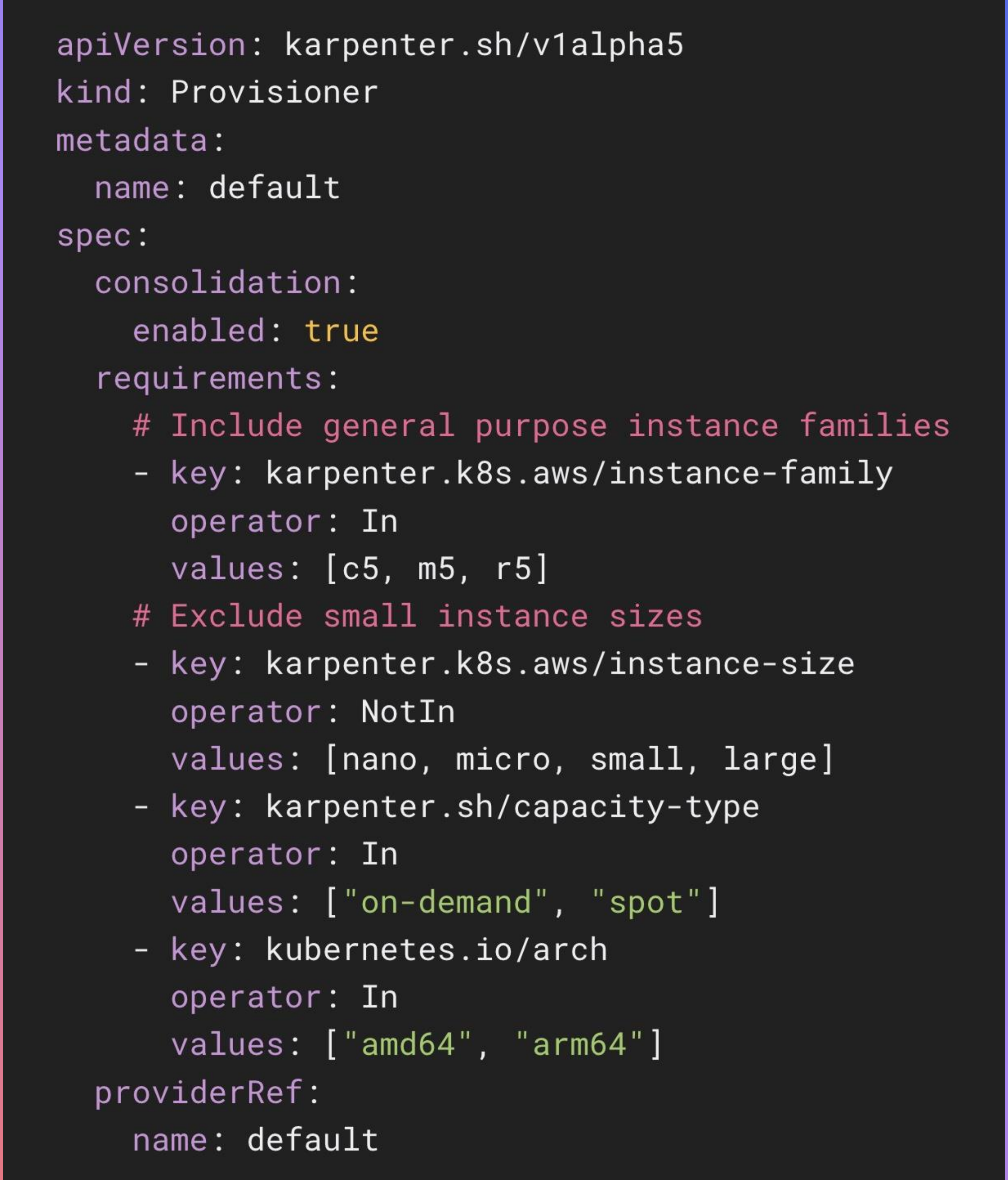
🟨 1 vCPU request

Compute flexibility with Karpenter



Provisioner CRD

- **Provisioner** – custom resource to provision nodes with a set of attributes (taints, labels, requirements, TTL)
- Single provisioner can manage compute for multiple teams and workloads
- Can also have multiple provisioners for isolating compute for different needs



```yaml
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
  consolidation:
    enabled: true
  requirements:
    # Include general purpose instance families
    - key: karpenter.k8s.aws/instance-family
      operator: In
      values: [c5, m5, r5]
    # Exclude small instance sizes
    - key: karpenter.k8s.aws/instance-size
      operator: NotIn
      values: [nano, micro, small, large]
    - key: karpenter.sh/capacity-type
      operator: In
      values: ["on-demand", "spot"]
    - key: kubernetes.io/arch
      operator: In
      values: ["amd64", "arm64"]
  providerRef:
    name: default
```

Compute flexibility

Instance type flexibility

- No list → picks from all instance types in EC2 universe, **excluding metal**
- Attribute-based requirements → sizes, families, generations, CPU architectures

AZ flexibility

- Provision in any AZ
- Provision in specified AZs

```yaml
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
  requirements:
    - key: karpenter.k8s.aws/instance-family
      operator: In
      values: [c5, m5, r5]
    - key: topology.kubernetes.io/zone
      operator: In
      values: ["us-west-2a", "us-west-2b"]
    - key: karpenter.sh/capacity-type
      operator: In
      values: ["on-demand", "spot"]
    - key: kubernetes.io/arch
      operator: In
      values: ["amd64", "arm64"]
  providerRef:
    name: default
```

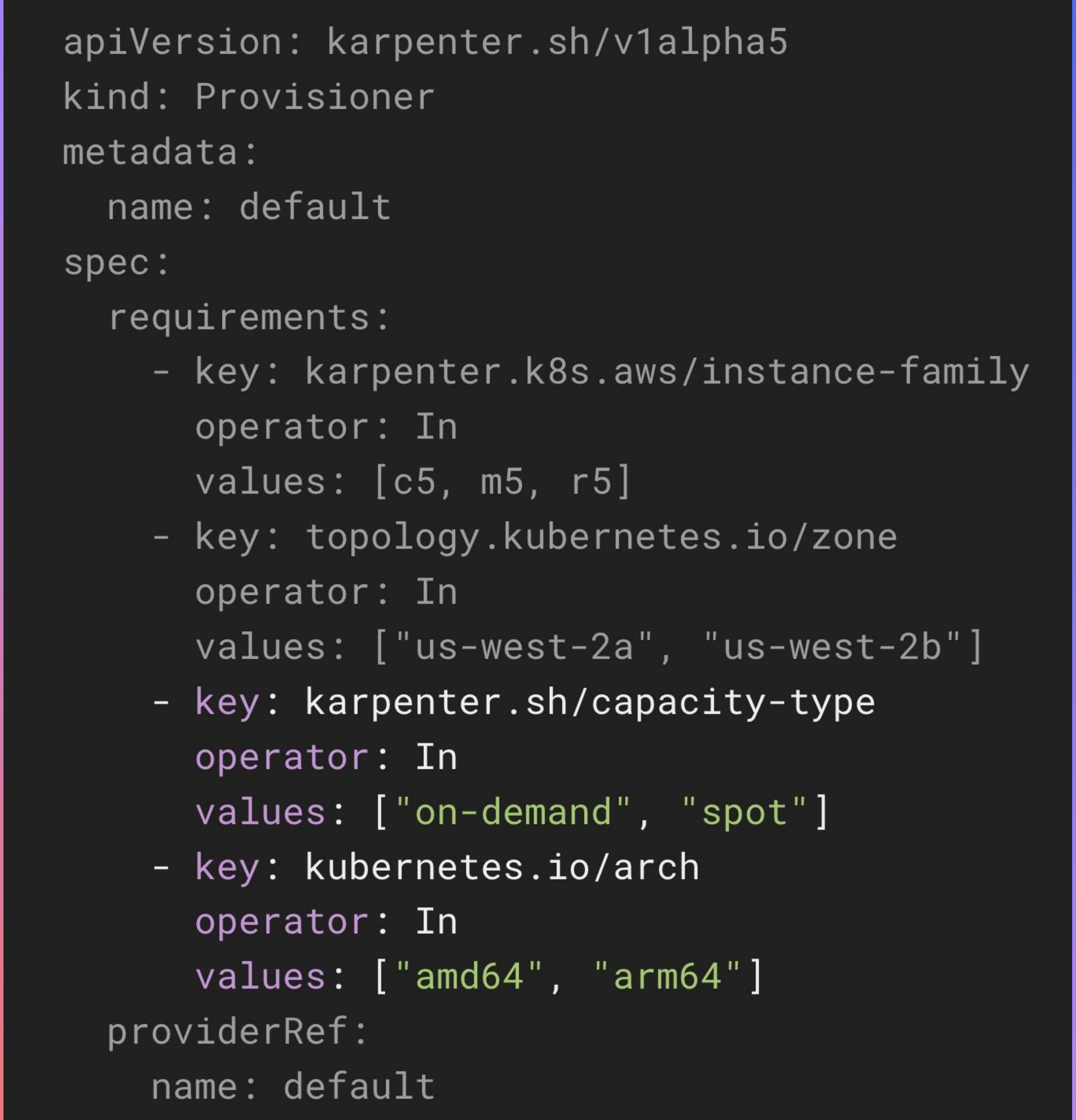
Compute flexibility

Purchase options flexibility

- On-demand, if nothing specified
- Prioritizes Spot if flexible to both capacity types

CPU architecture flexibility

- x86-64
- Arm64



```yaml
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
  requirements:
    - key: karpenter.k8s.aws/instance-family
      operator: In
      values: [c5, m5, r5]
    - key: topology.kubernetes.io/zone
      operator: In
      values: ["us-west-2a", "us-west-2b"]
    - key: karpenter.sh/capacity-type
      operator: In
      values: ["on-demand", "spot"]
    - key: kubernetes.io/arch
      operator: In
      values: ["amd64", "arm64"]
  providerRef:
    name: default
```

Node template CRD

- **AWS node template** – configures cloud provider-specific parameters, such as tags, subnets, and AMIs

- Supports custom user-data and AMIs without launch templates

```
apiVersion: karpenter.k8s.aws/v1alpha1
kind: AWSNodeTemplate
metadata:
  name: default
spec:
  amiFamily: AL2
  subnetSelector:
    karpenter.sh/discovery: my-cluster
  securityGroupSelector:
    karpenter.sh/discovery: my-cluster
  blockDeviceMappings:
  - deviceName: /dev/xvda
    ebs:
      volumeSize: 100Gi
      volumeType: gp3
  tags:
    team: dev
```

Compute per workload scheduling requirements

Workloads may be required to run

In certain AZs

On certain types of processors or hardware (AWS Graviton, GPUs)

On Spot or on-demand capacity

Standard K8s pod scheduling mechanisms

- Node selectors
- Node affinity
- Taints and tolerations
- Topology spread

Pod scheduling constraints must fall within a provisioner’s constraints

Karpenter

SPOT DEPLOYMENT

Allocation strategy

- Price Capacity Optimized
    - Reduce the frequency of Spot terminations
    - Reduce the cost of the instances

Diversify and don’t constrain

Spot interruptions

The work you are doing to make your applications fault-tolerant also enabled Spot

Spot notification

- 2-minute Spot Instance interruption notice via instance metadata or Event Bridge event

Spot is optimized for stateless, fault-tolerant, or flexible workloads

What do you do?

- Implement node termination handling for interruptions, thus **increasing chance of completing work**
- Capture the spot termination signal and implementing graceful termination and **best effort checkpointing.**

Best practices with EKS

Karpenter

- Use Karpenter for workloads with changing capacity needs
- Do not run Karpenter on a node that is managed by Karpenter
- Karpenter controller on EKS Fargate or on an OD worker node (1 node nodegroup)
- Exclude instance types that do not fit your workload

Provisioner

- Enable Consolidation
- Use Node Expiration TTLs to rotate nodes
- Use a diverse set of instance types

Scheduling Pods

- Follow EKS best practices for high availability
- Use nodeSelectors and taints for colocation
- Create billing alarms to monitor your compute spend on top of Resource Limits
- Use the do-not-evict annotation for critical nodes
- Use LimitRanges to configure defaults for resource requests and limits on a namespace

Recap

- Karpenter provides asynchronous infrastructure management
- Karpenter is compatible with native K8S scheduling
- Karpenter is open-source and evolving quickly
- Karpenter offers compute flexibility and cost optimization

Now let’s experience Karpenter first-hand in our workshop

Workshop link and hash

https://ec2spotworkshops.com/

5. EKS and Karpenter … at an AWS event

1b69-1209dfdbe4-3f

aws

Thank you!

Please complete the session survey in the **mobile app**