# Karpenter: Efficient scaling of Kubernetes clusters

Viktor Vedmich

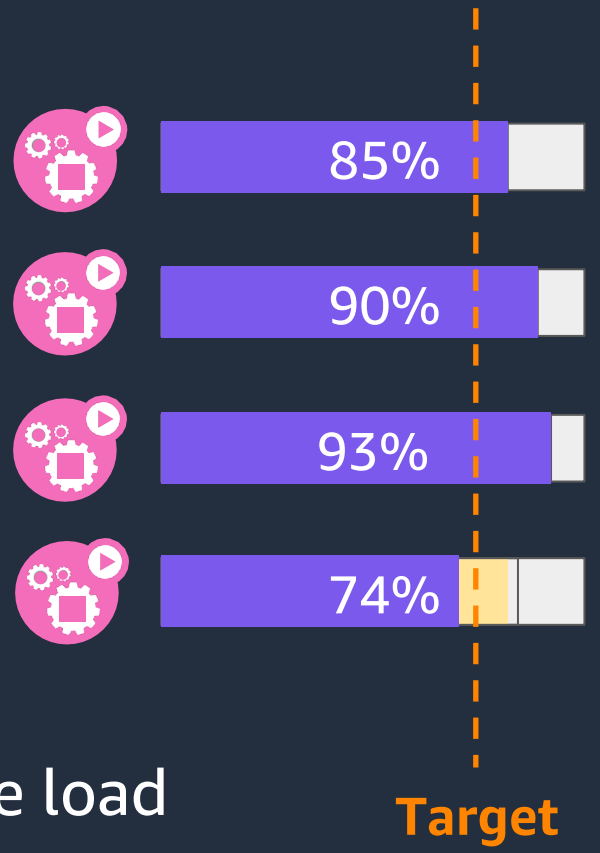
Sr. Developer Advocate
AWS

# How can we scale use Kubernetes?

# Horizontal and Vertical Pod Autoscalers

## HPA and VPA

# HPA - Concept


Specify the target for the load

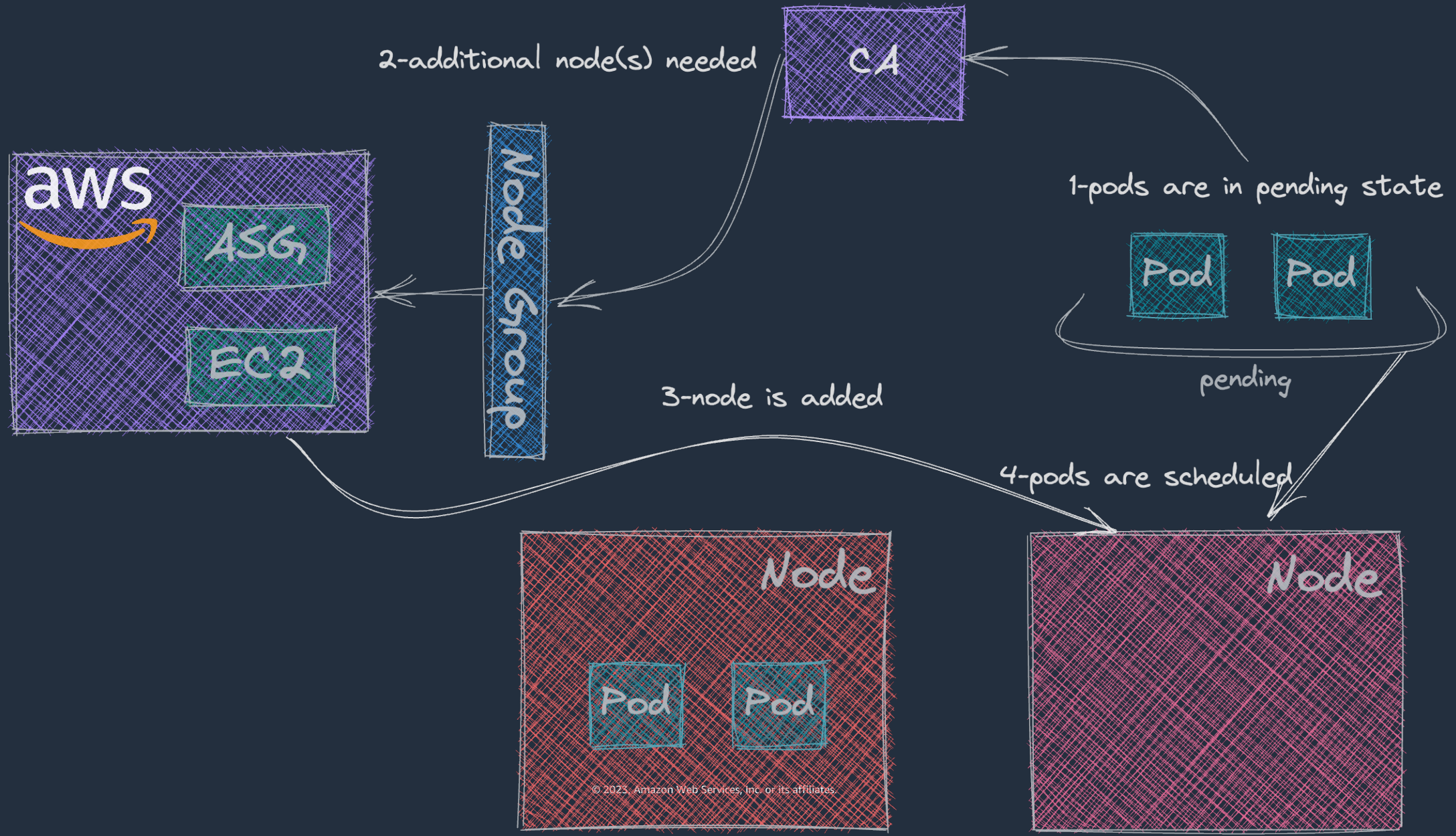
Ex: Target = CPU utilization 70%

# VPA - Concept

- Frees the user from the necessity of setting up to date **resource limits** and **requests**.
- Will set resource limits and requests according to the pod's actual usage.
- Supports scale-down and scale-up
- Will most commonly be used for a deployment object

# Cluster Autoscaler
# CA

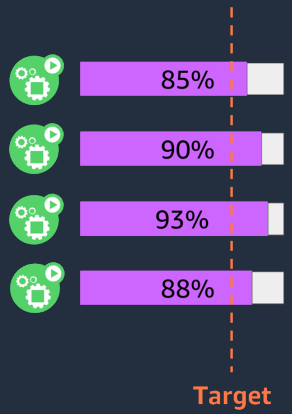
# Cluster Autoscaler: steps to add new node(s)



# Cluster Autoscaler scale-up

Pending pods

Cluster Autoscaler

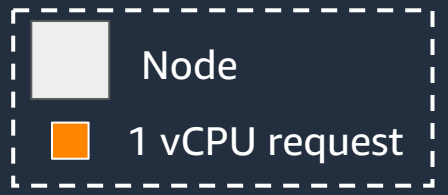




Expander

Autoscaling group
4vCPUs 16 GB Spot

New nodes



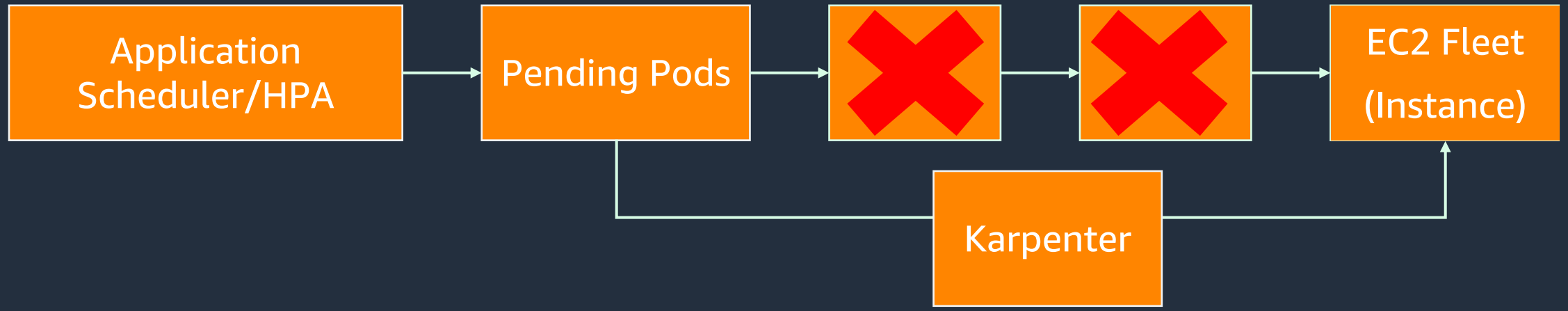

Cluster AutoScaler RunOnce:

- Reconciliation and filtering
- Scale up (simulation and expander logic)
- Scale down and filtering

aws

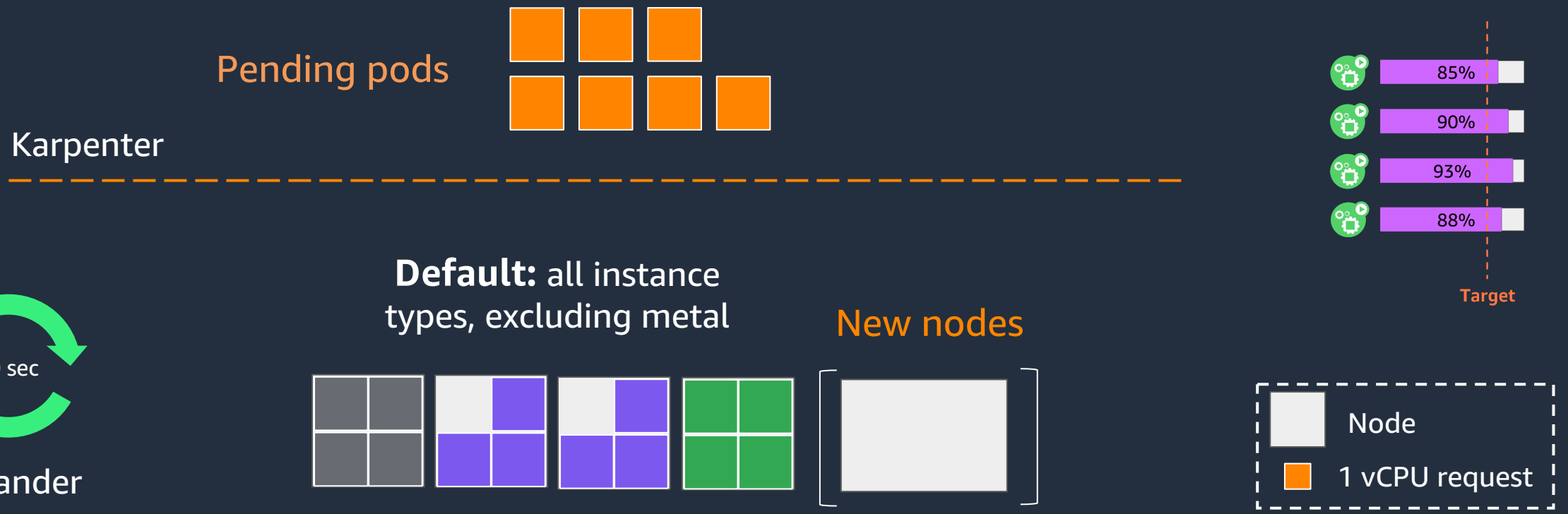
# Karpenter Cluster Autoscaler

# How Karpenter provisions nodes on AWS




consolidates instance orchestration responsibilities within a single system

# Karpenter scale-up




## Provisioning and scheduling decisions

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

# Karpenter scale-in

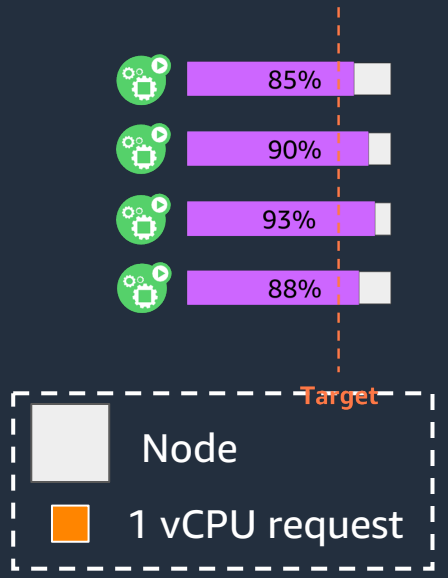
Karpenter

10 sec

Expander

**Default:** all instance types, excluding metal

New nodes




## Terminations

- Remove underutilized nodes (empty nodes)
- Node TTL
- Consolidation

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

# Karpenter Consolidation

Karpenter simulates all pods being evicted from a candidate node

```
spec:
  consolidation:
    enabled: true
```

Reduce the overall cost in two ways:

- Node Deletion
- Node

Protecting an Application with a **PodDisruptionBudget**

# Compute provisioning with Provisioner CRD

- Provisioner – Custom Resource to provision nodes with a set of optional attributes (Taints, Labels, Requirements, TTL)
- A single provisioner can manage compute for multiple teams and workloads
- Create a default provisioner (named “default”) for common scenarios
- Multiple provisioners for isolating compute for different needs

```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
  labels:
    intent: apps
  ttlSecondsAfterEmpty: 30
  ttlSecondsUntilExpired: 2592000

  requirements:
    - key: karpenter.sh/capacity-type
      operator: In
      values: ["spot“,”on-demand”]
    - key: "topology.kubernetes.io/zone"
      operator: In
      values: ["us-west-2a", "us-west-2b"]
taints:
    - key: example.com/special-taint
      effect: NoSchedule

  limits:
    resources:
      cpu: 1000

  provider:
    securityGroupSelector:
      karpenter.sh/discovery: ${CLUSTER_NAME}
```

# 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

## CPU architecture

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

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

```
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-type
      operator: In
      values: ["m5.large", "m5.2xlarge"]
```

## 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"]
```

# Taints

## Startup Taints

- Temporary nodes start with the taint
- DaemonSet will delete it (networking)

```
startupTaints:
    - key: example.com/another-taint
      effect: NoSchedule
```

## Taints

- Prevent pods from scheduling

```
taints:
    - key: example.com/special-taint
      effect: NoSchedule
```

# Scheduling

## Node Selector

- Ask for a node that matches selected key-value pairs
- well-known labels or custom labels

```
nodeSelector:
  topology.kubernetes.io/zone: us-west-2a
  karpenter.sh/capacity-type: spot
```

## Node Affinity

- requiredDuringSchedulingIgnoredDuringExecution:
  - hard rule that must be met.
- preferredDuringSchedulingIgnoredDuringExecution:
  - preference, pod can run on a node where it is not guaranteed.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
          - key: "topology.kubernetes.io/zone"
            operator: "In"
            values: ["us-west-2a, us-west-2b"]
          - key: "topology.kubernetes.io/zone"
            operator: ”NotIn"
            values: ["us-west-2b"]
```

# Custom User Data and AMI

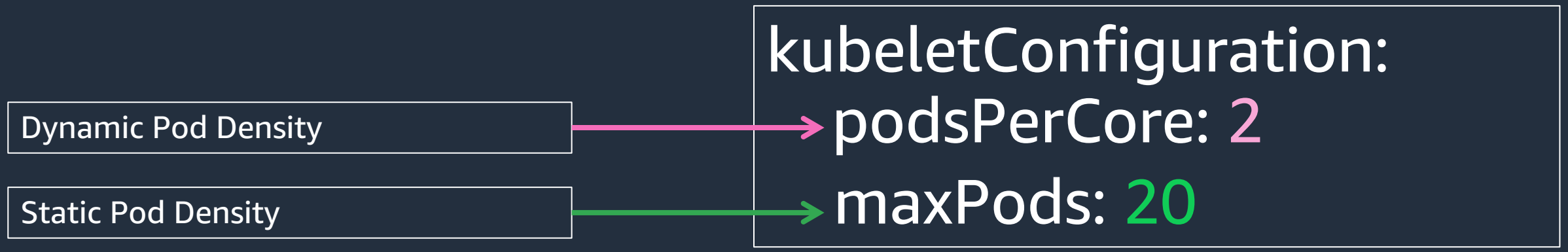
```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
  providerRef:
    name: bottlerocket-example
  ...
```

```
apiVersion: karpenter.k8s.aws/v1alpha1
kind: AWSNodeTemplate
metadata:
  name: bottlerocket-example
spec:
    amiFamily: Bottlerocket
    instanceProfile: MyInstanceProfile
    subnetSelector:
        karpenter.sh/discovery: my-cluster
    securityGroupSelector:
        karpenter.sh/discovery: my-cluster
    userData:  |
        [settings.kubernetes]
        kube-api-qps = 30
        [settings.kubernetes.eviction-hard]
        "memory.available" = "20%"
    amiSelector:
        karpenter.sh/discovery: my-cluster
```

# Control Pod Density

## Networking Limitations

- Number of networking interfaces (ENIs)
- Number of IP addresses that can be assigned to each ENI




## Limit Pod Density

- Topology Spread
- Restrict Instance Types

# Demo time

# Demo

## Env karpenter-01

1. Review how karpenter requests resources
2. Consolidate resources – check efficiency usage of resources
3. Deploy app 50% on spot instances and 50% on on-demand instances (multiple profiles)

## Env karpenter high load

1. Deploy 3000 pods : 500 pods per deployment
2. Check efficiency usage of resources
3. How quickly karpenter scale down

# Thank you!

Viktor Vedmich

**https://karpenter.sh/**