



Raj Saha

cloudwithraj.com



Cloud With Raj



cloudwithraj

Instructor Bio:

Pr. Solutions Architect @aws

Bestselling author

“Top Systems Design” award by LinkedIn

Public speaker at major conferences

Author of multiple official AWS blogs

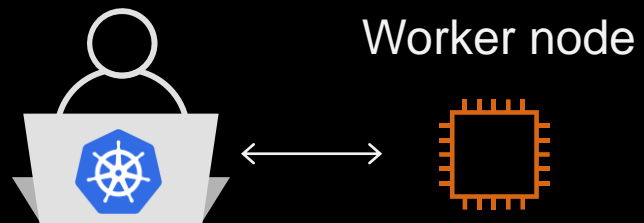
YouTuber with 100K+ subscribers

Previously - Distinguished Cloud Architect @Verizon

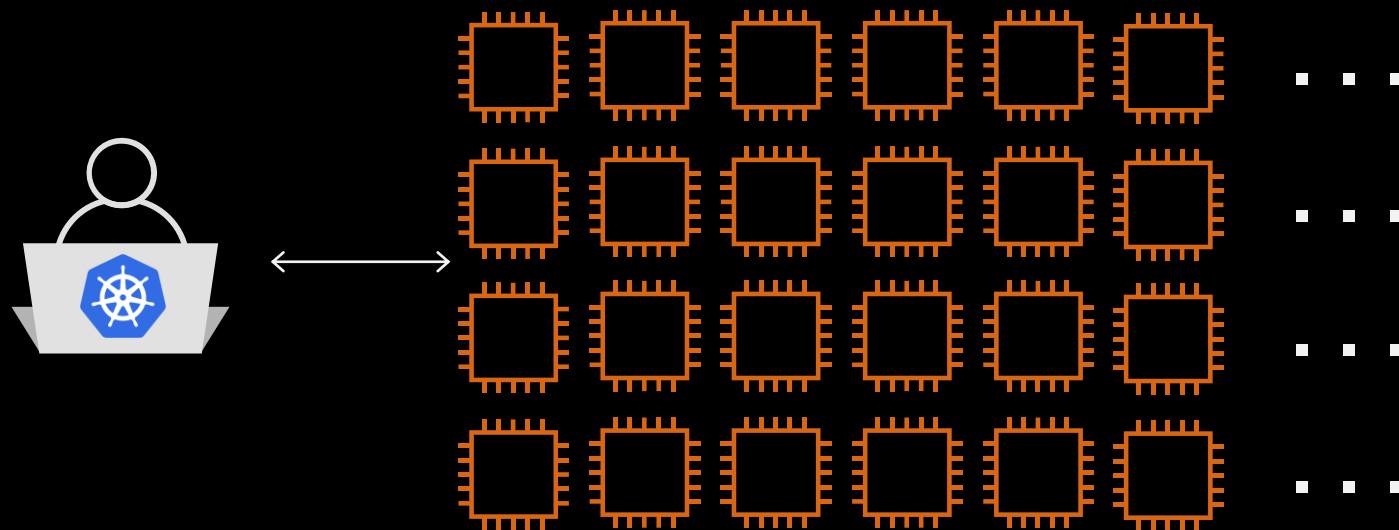
Opinions are my own

How Does Kubernetes Scale?

What is our goal?



This



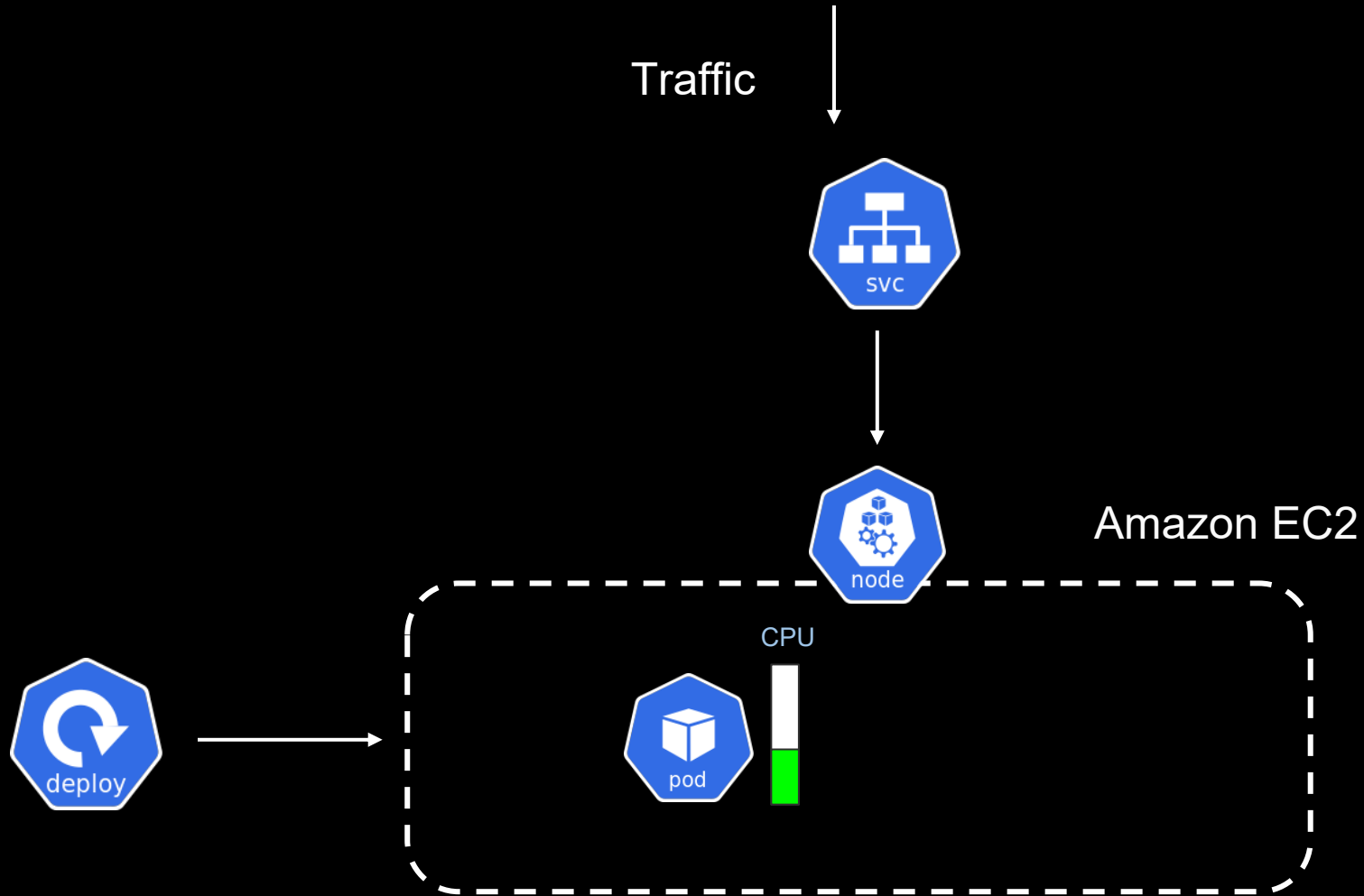
To this

EKS Container Scaling

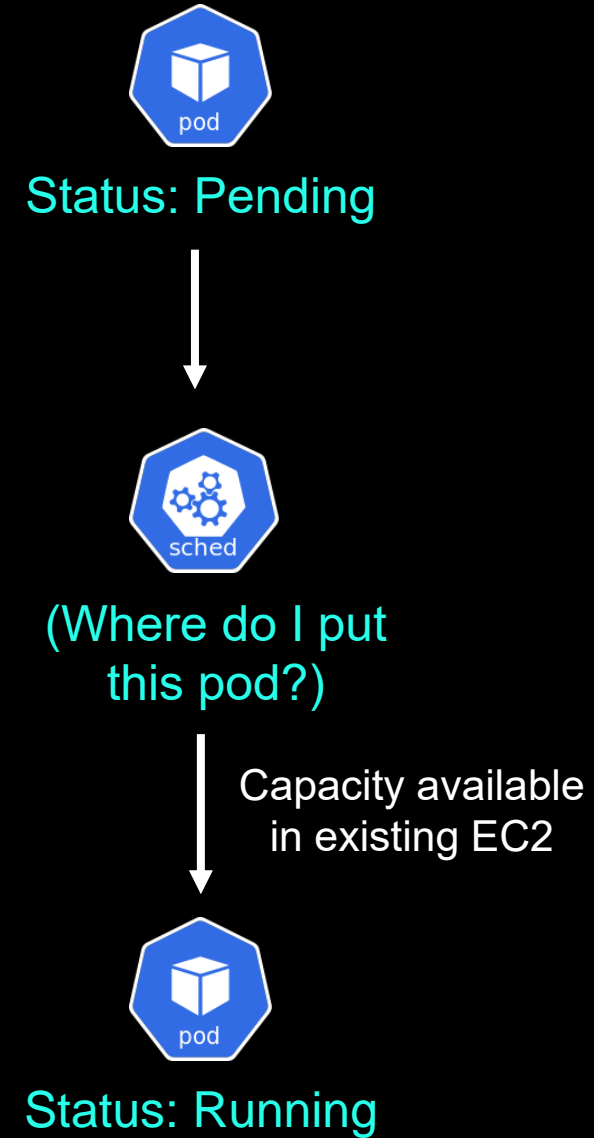
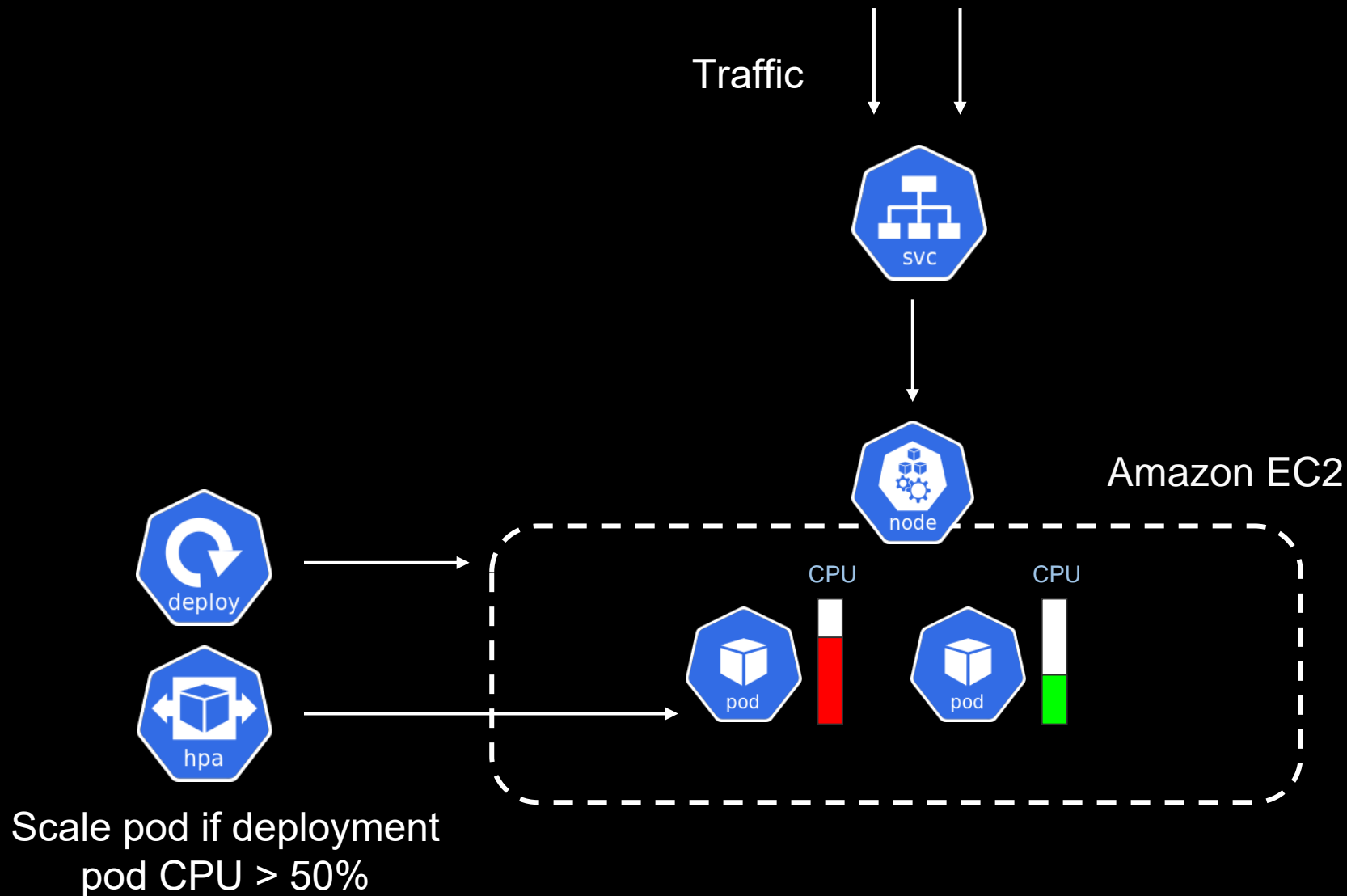


Worker node
(Amazon EC2)

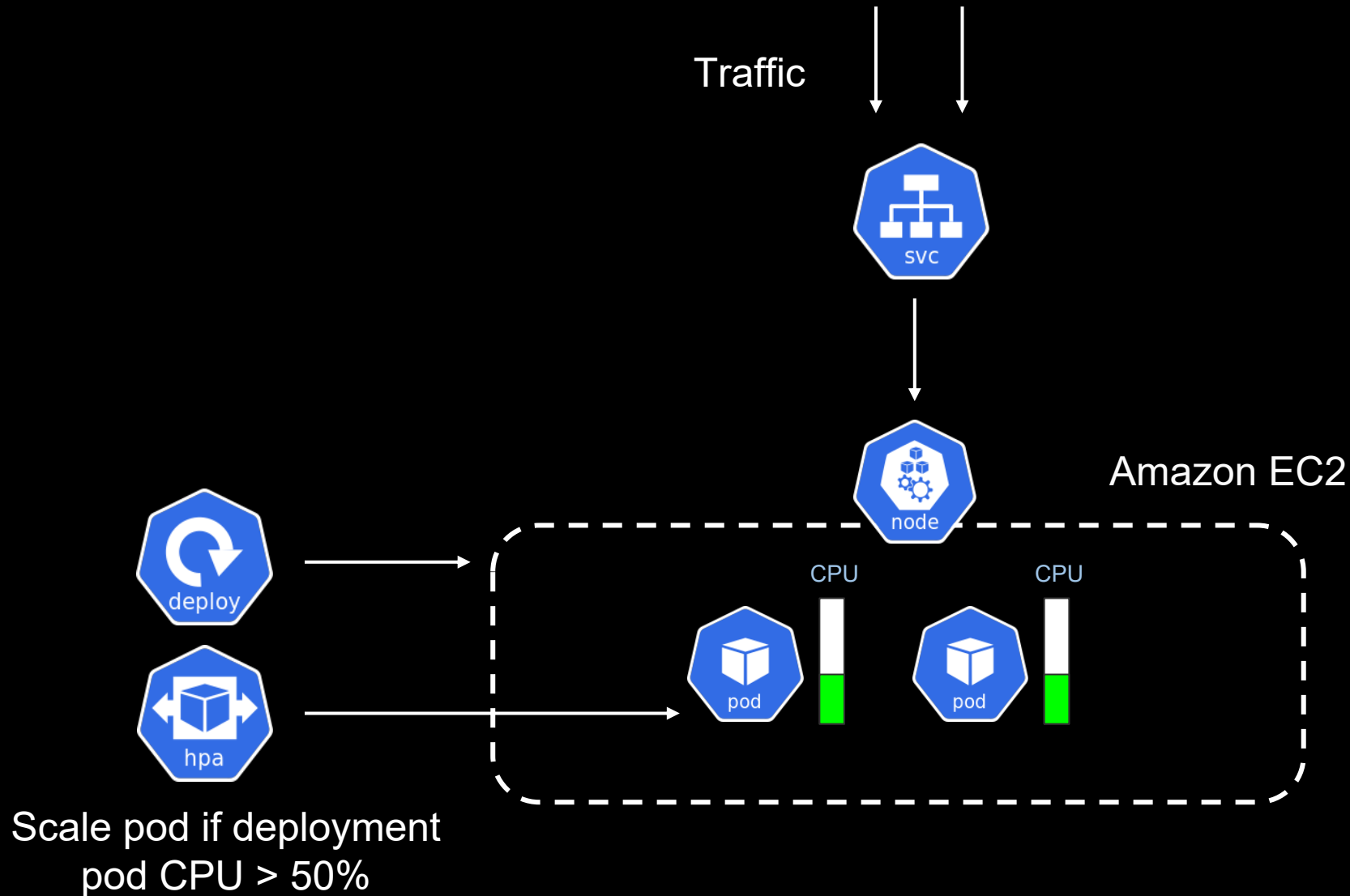
Horizontal Pod Autoscaler (HPA)



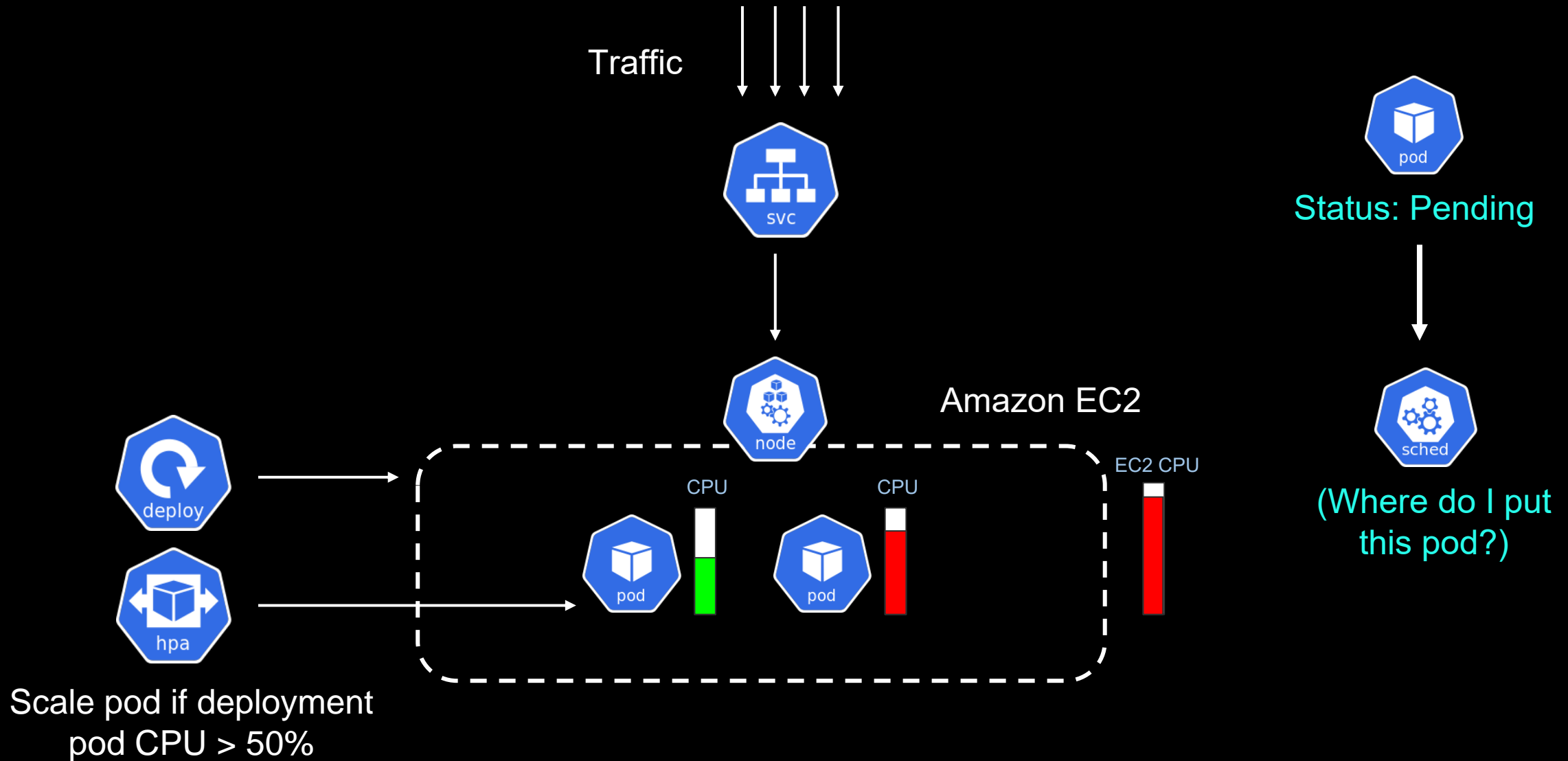
Horizontal Pod Autoscaler (HPA)



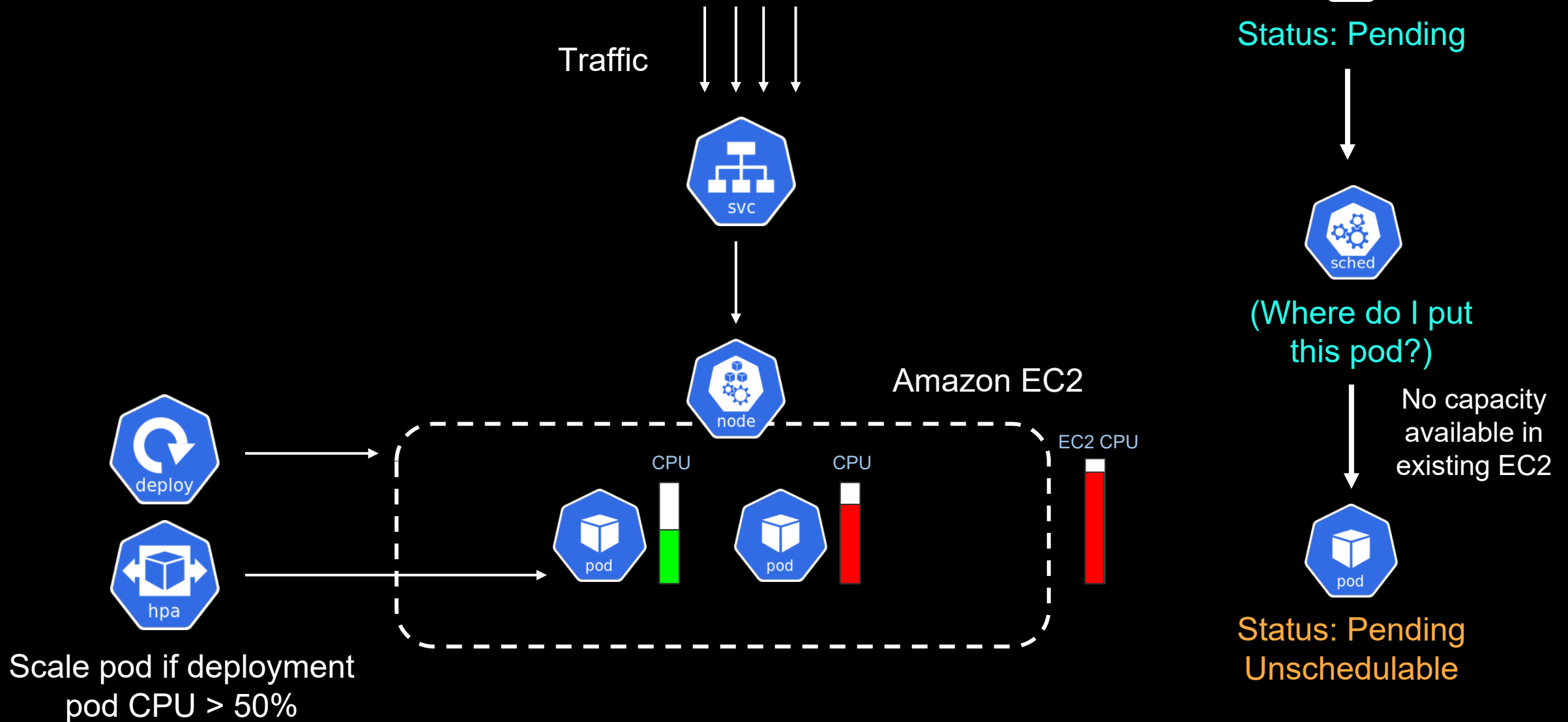
Horizontal Pod Autoscaler (HPA)



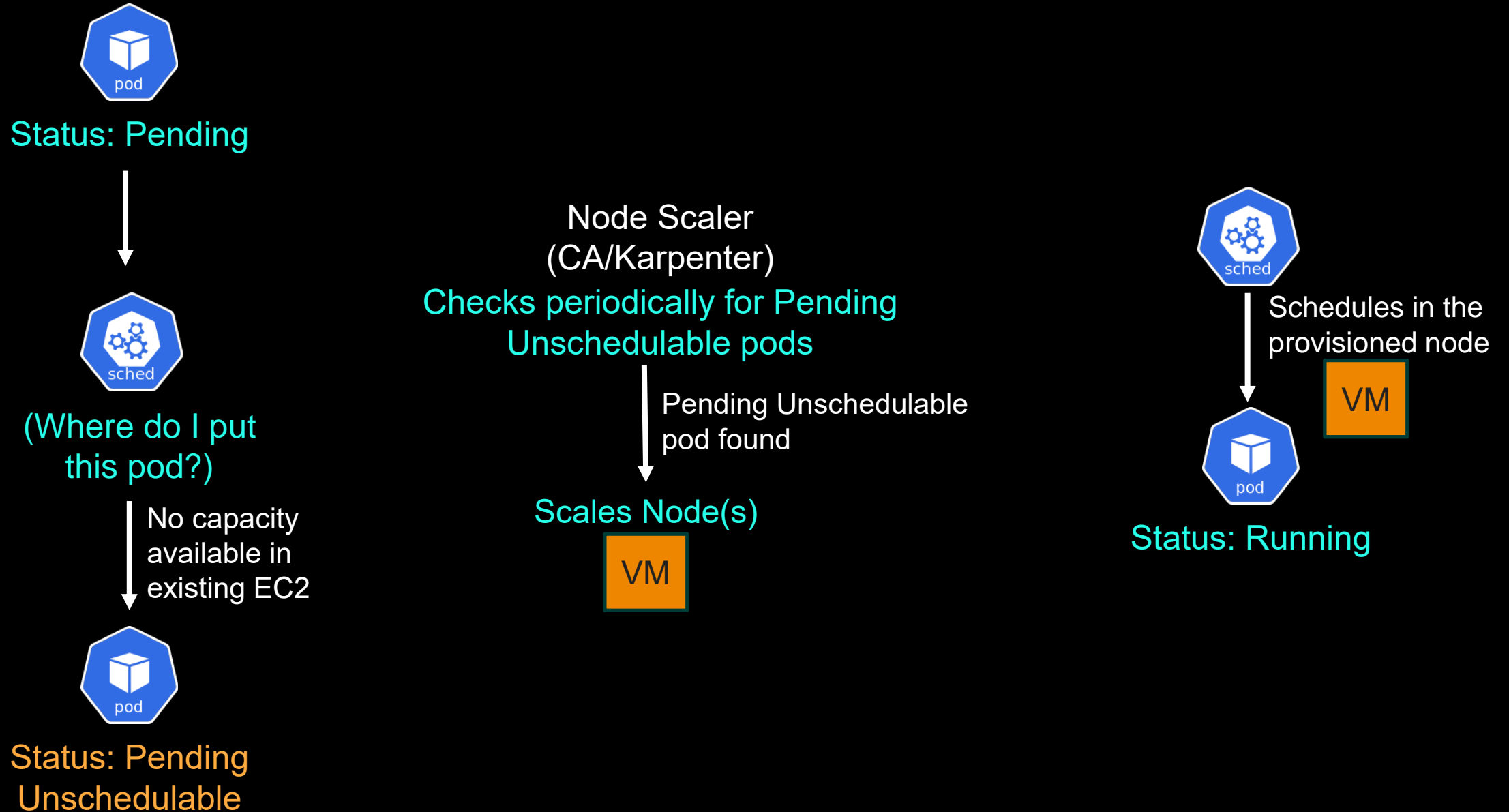
Horizontal Pod Autoscaler (HPA)



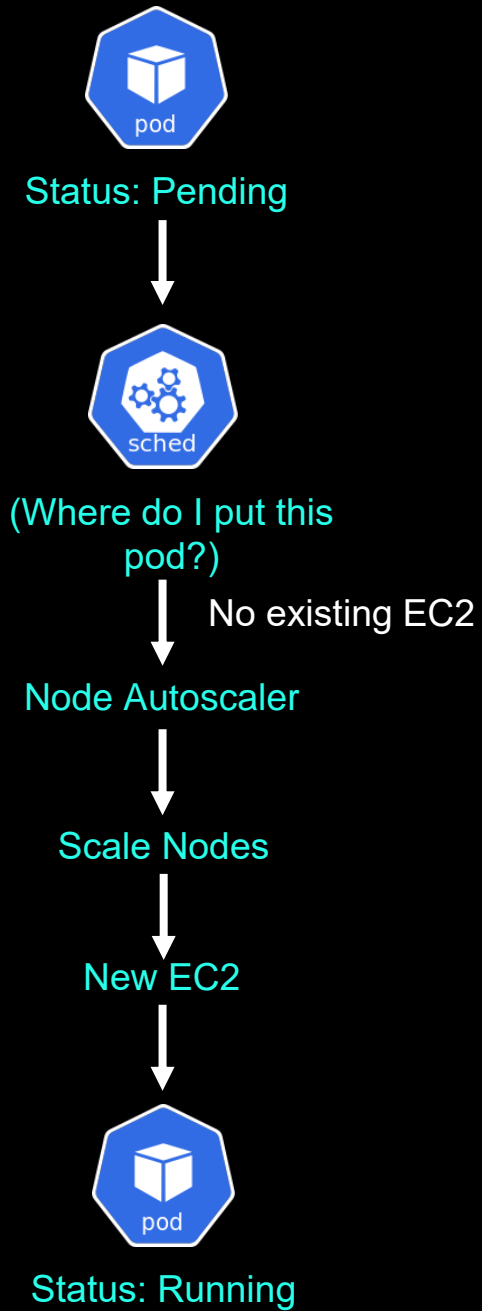
Horizontal Pod Autoscaler (HPA)



How Kubernetes Scale

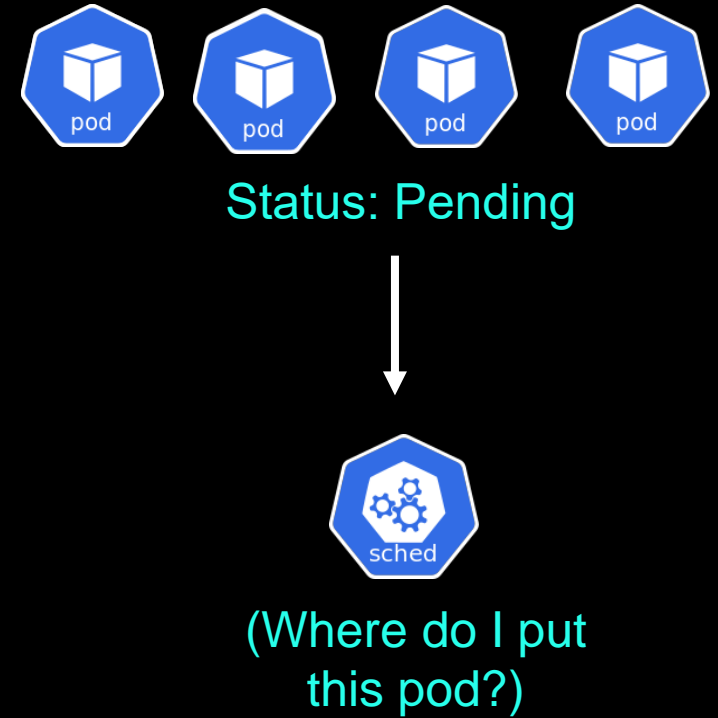
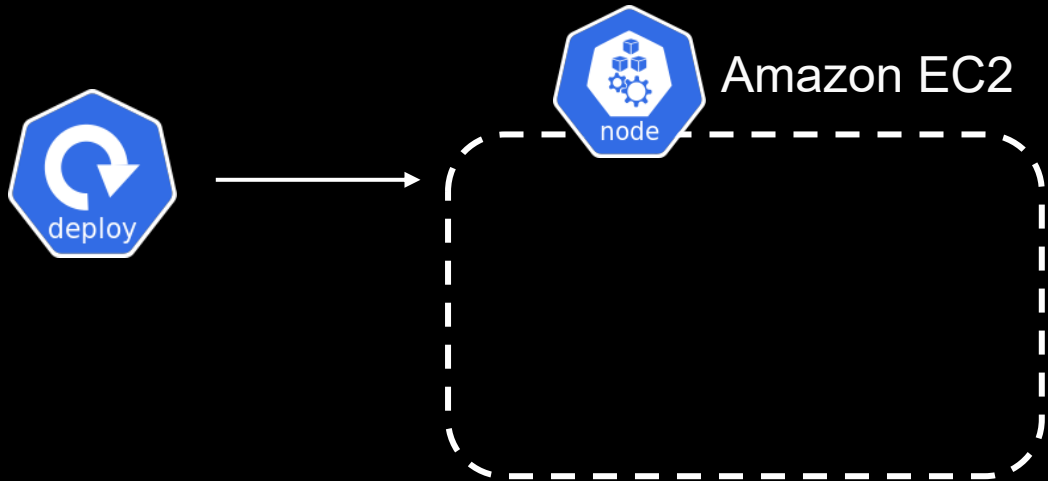


Simplified Flow



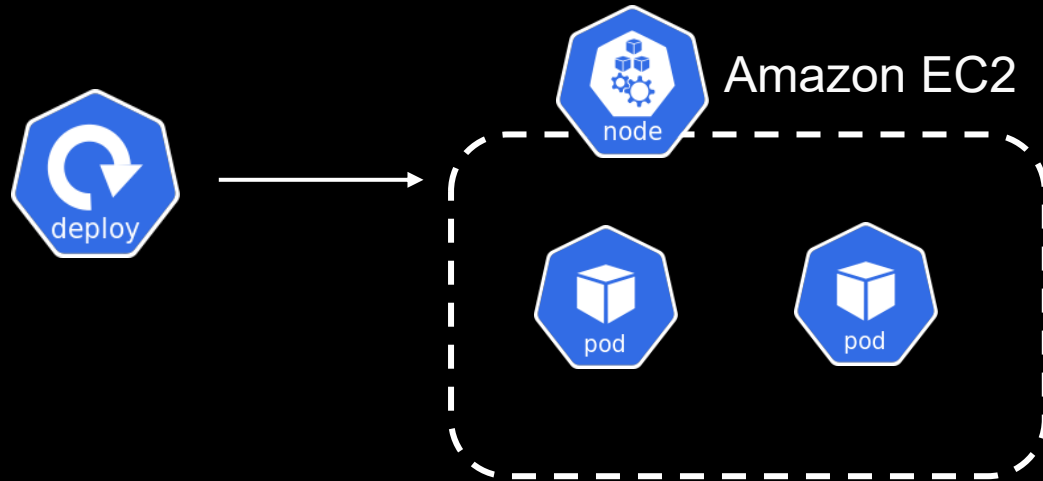
High Number of Replicas

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: batch-job-parallel
  labels:
    app: nightly-payment-processor
spec:
  replicas: 4
```



High Number of Replicas

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: batch-job-parallel
  labels:
    app: nightly-payment-processor
spec:
  replicas: 4
```



Status: Pending



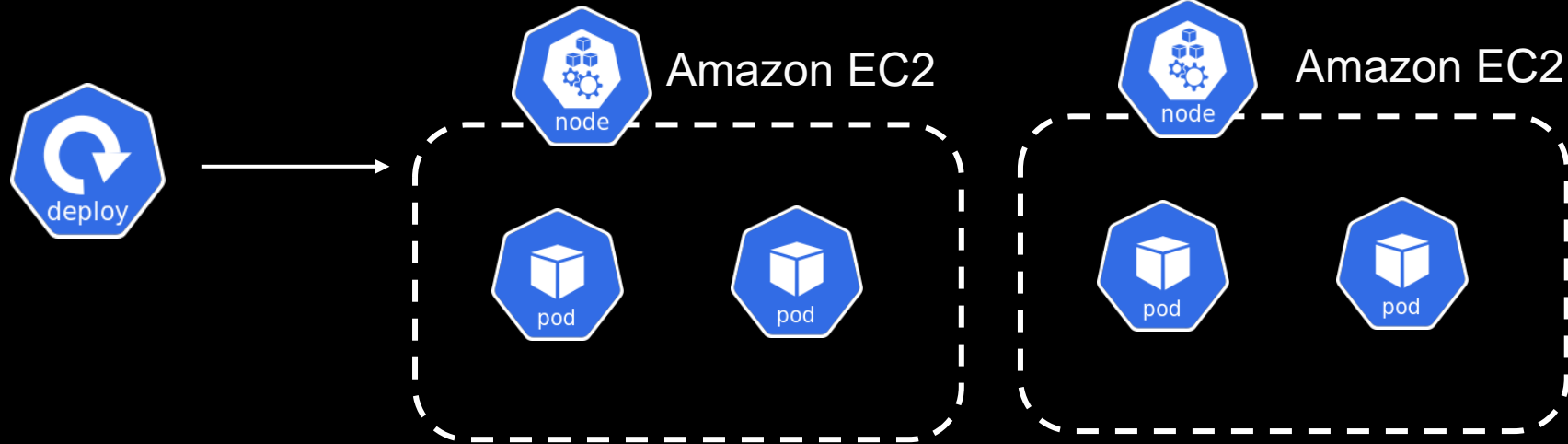
(Where do I put this pod?)



Status: Pending
Unschedulable

High Number of Replicas

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: batch-job-parallel
  labels:
    app: nightly-payment-processor
spec:
  replicas: 4
```



Node Autoscaler



Status: Pending

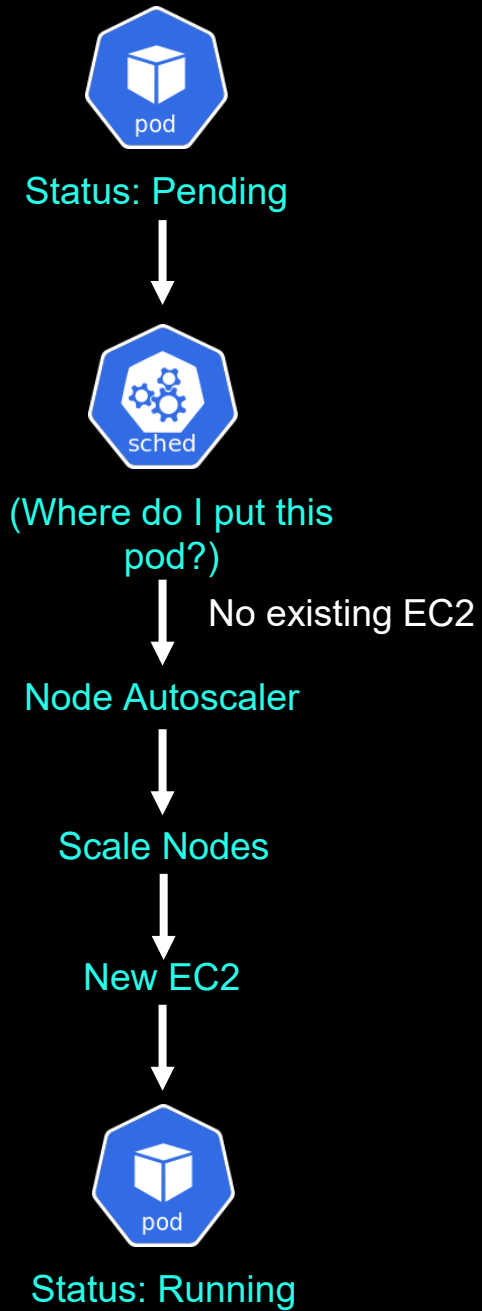


(Where do I put this pod?)



Status: Pending
Unschedulable

Simplified Flow

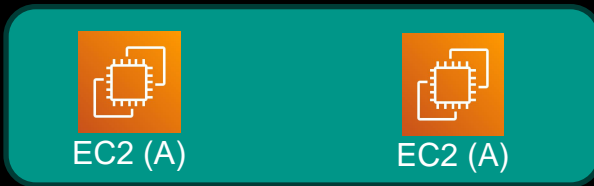


Cluster Autoscaler Challenges (Enter Karpenter)

Cluster Autoscaler (CA)

Cluster Autoscaler

Node Group A



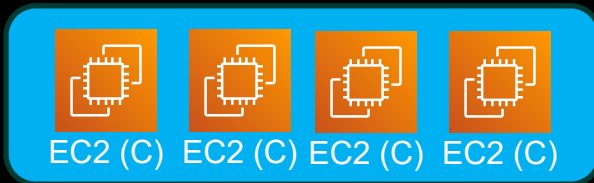
Auto Scaling Group A

Node Group B



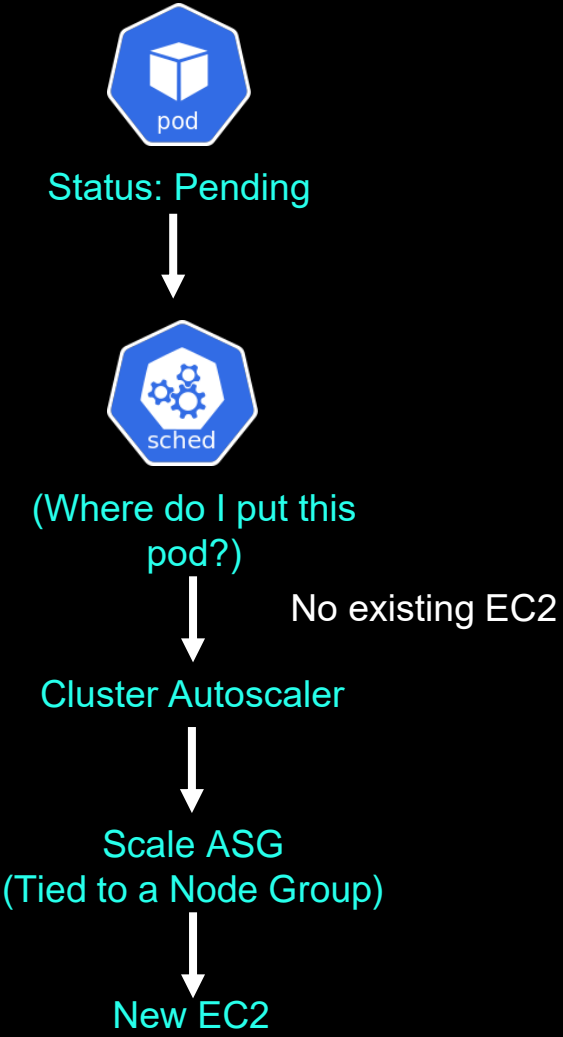
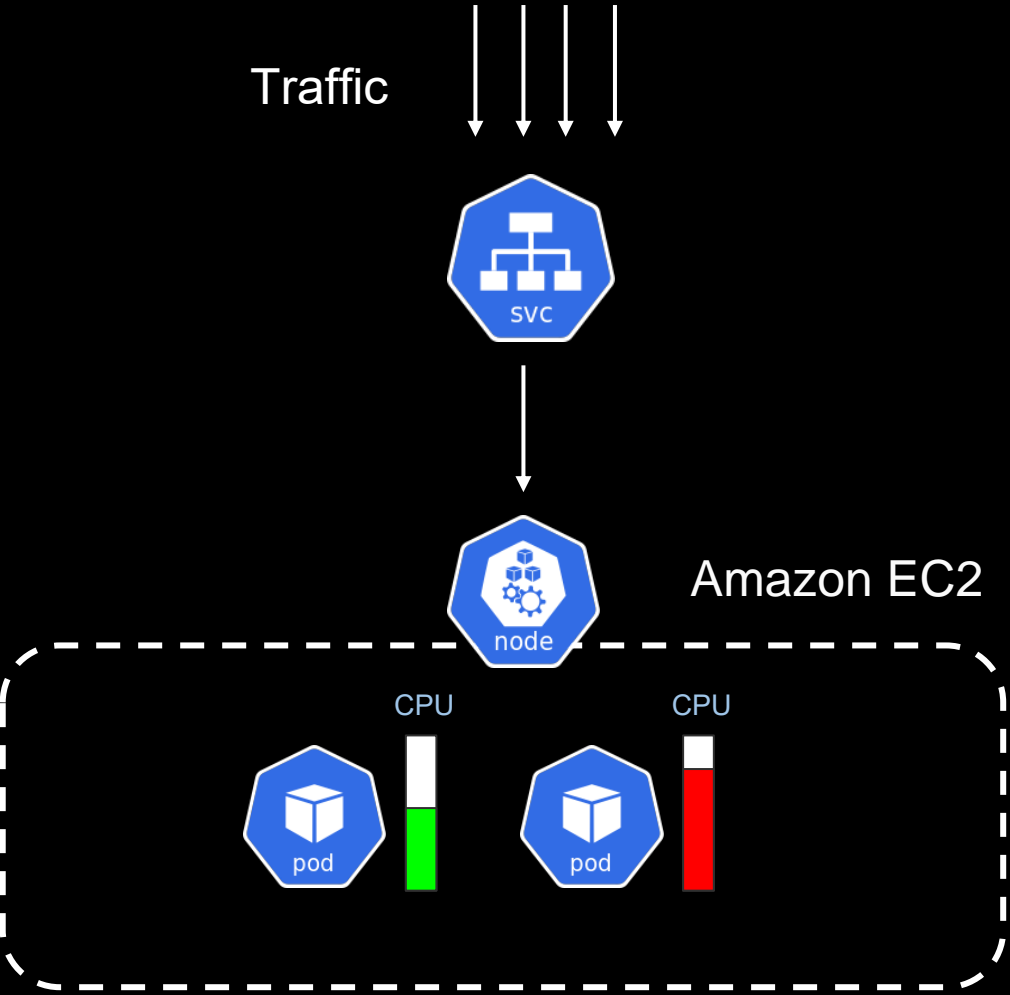
Auto Scaling Group B

Node Group C

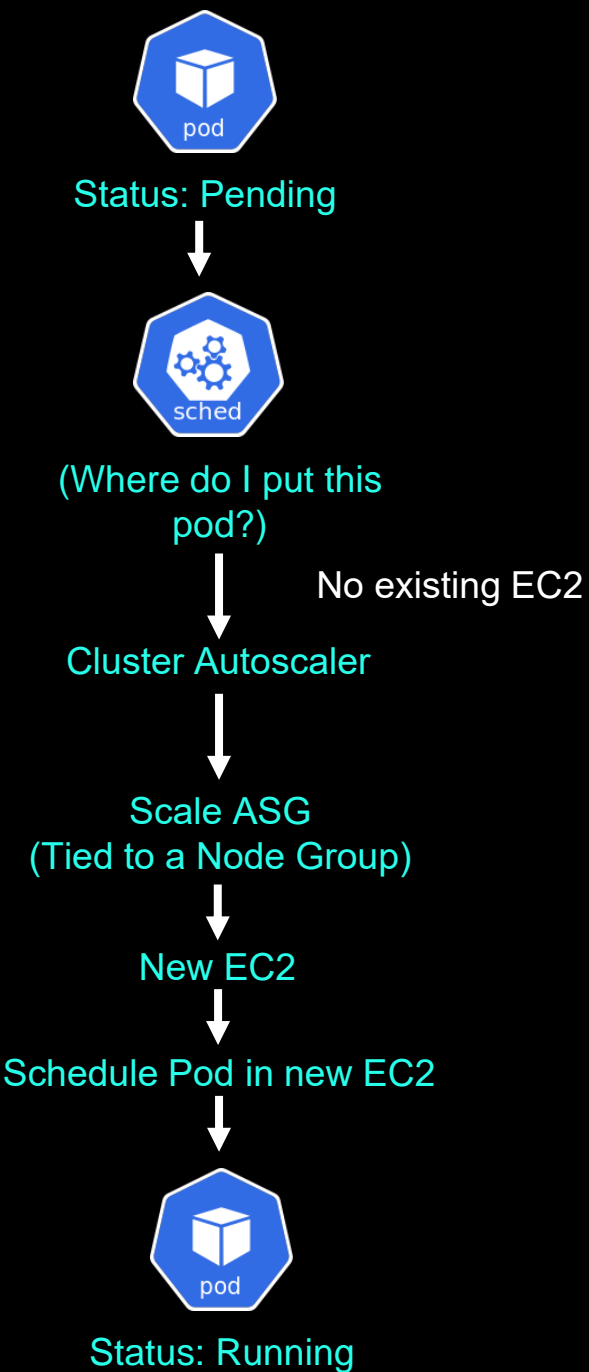
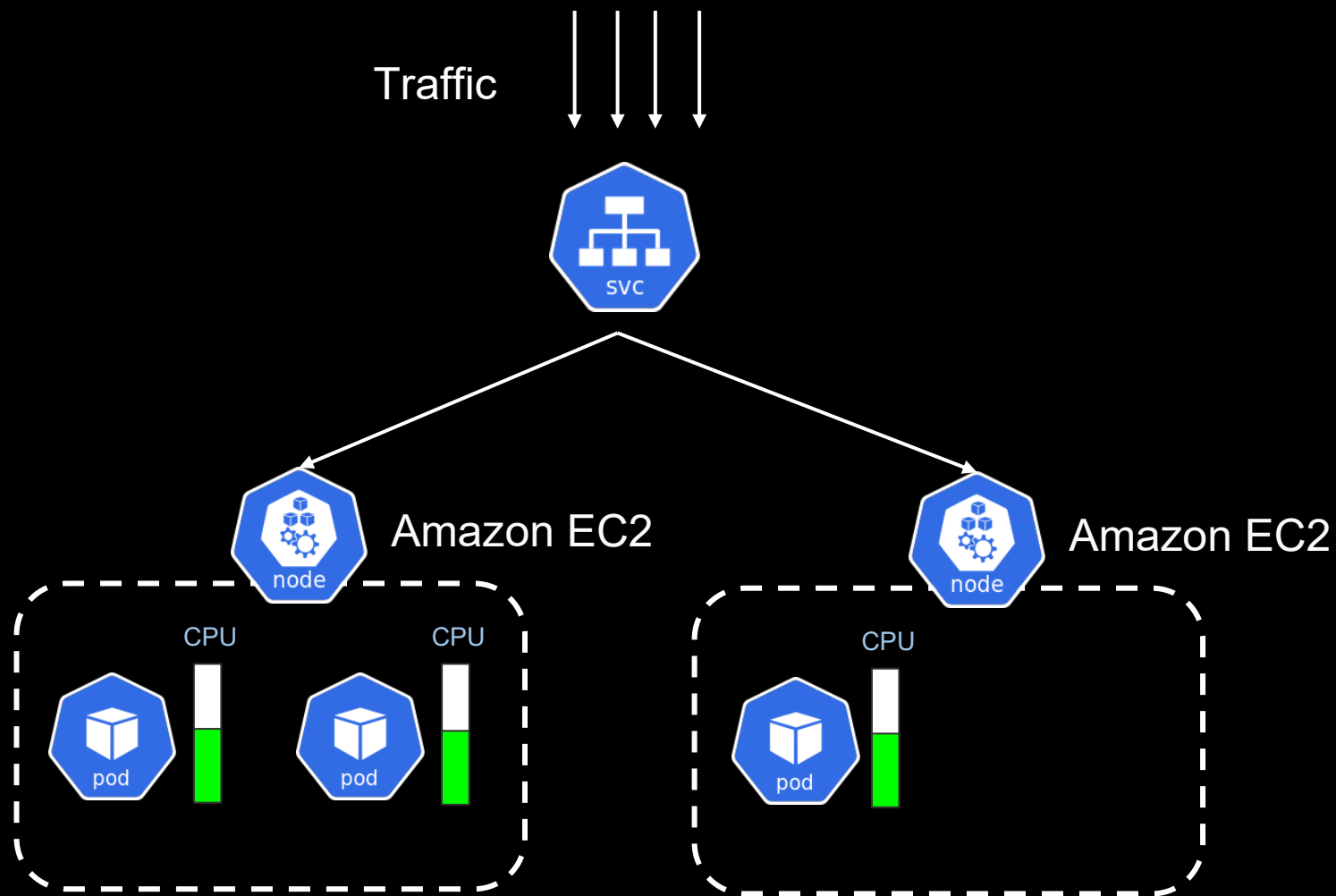


Auto Scaling Group C

Autoscaling Flow



Autoscaling Flow



Cluster Autoscaler Challenges

Cluster Autoscaler

Node Group A



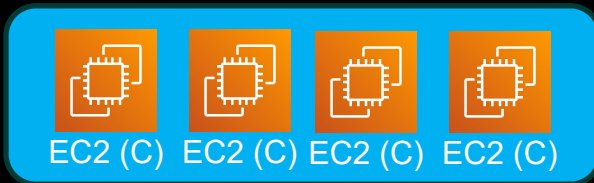
Auto Scaling Group A

Node Group B



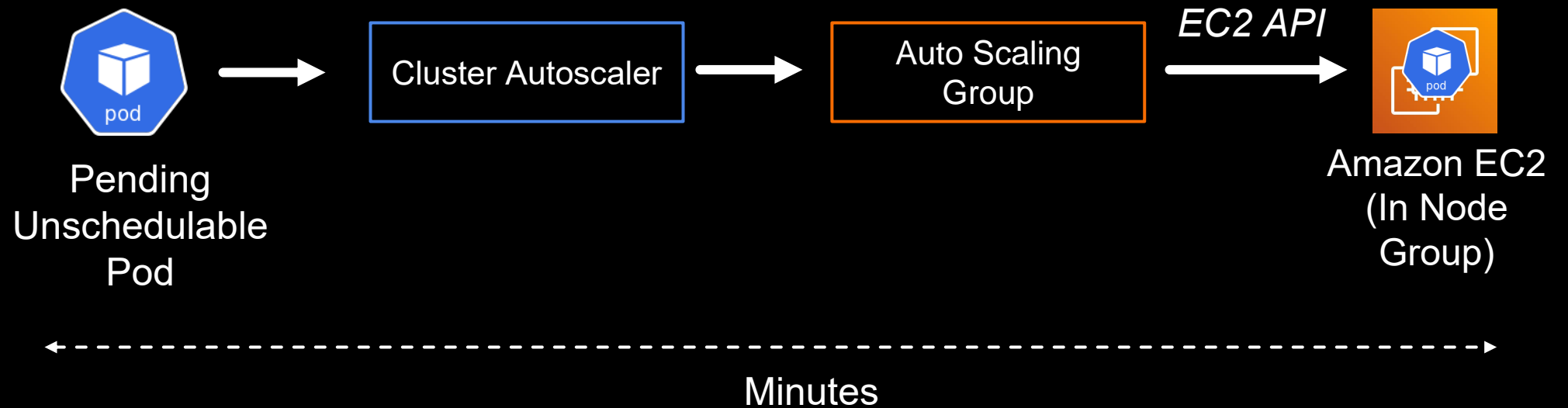
Auto Scaling Group B

Node Group C

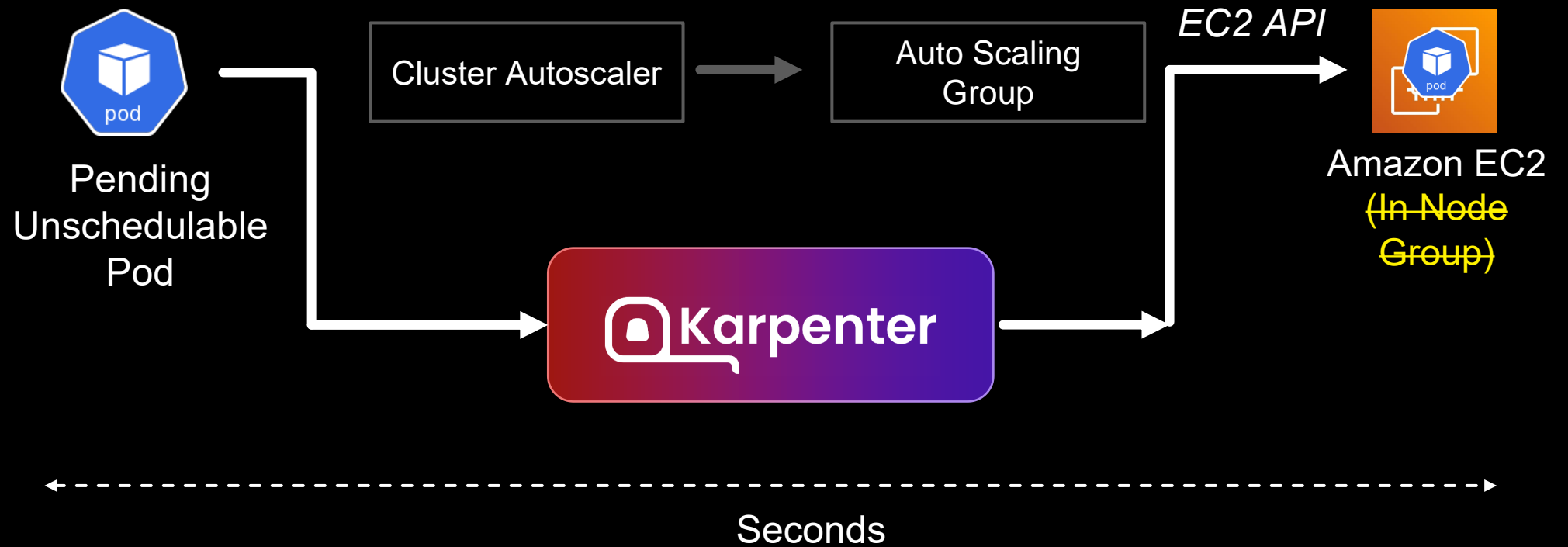


Auto Scaling Group C

Node Provision Latency



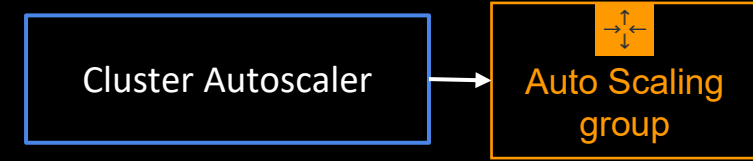
Enter Karpenter



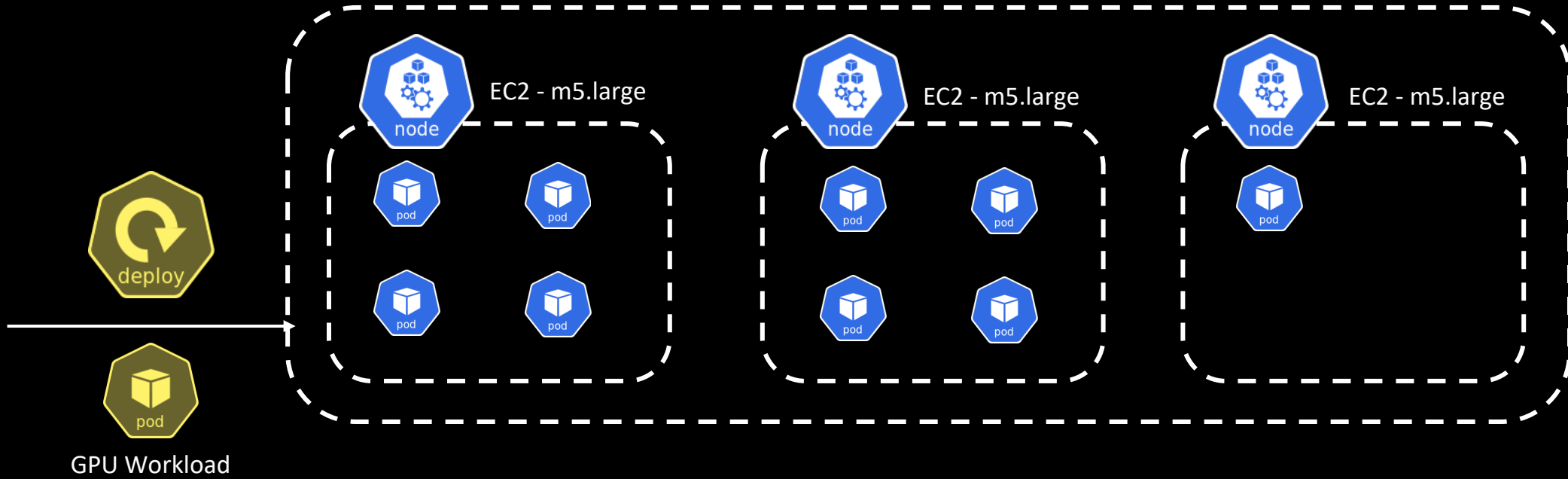
Enter Karpenter



Node Group Management



Compute Node Group

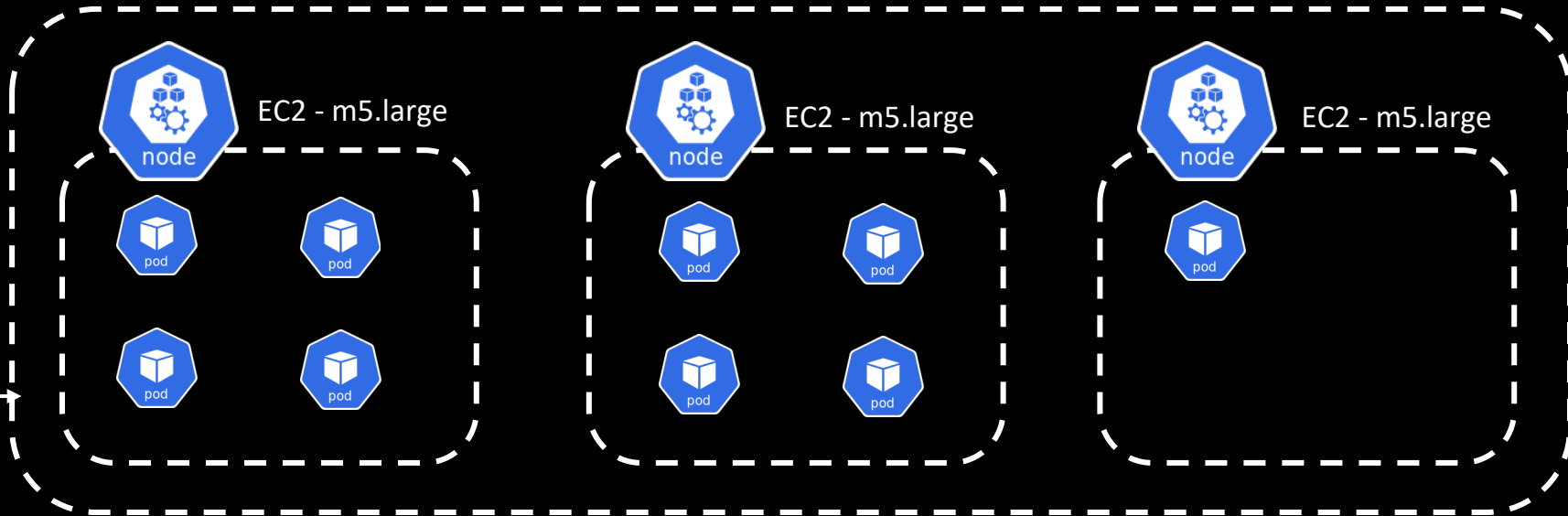


Node Group Management

Cluster Autoscaler

Auto Scaling group

Compute Node Group



GPU Node Group



GPU workload

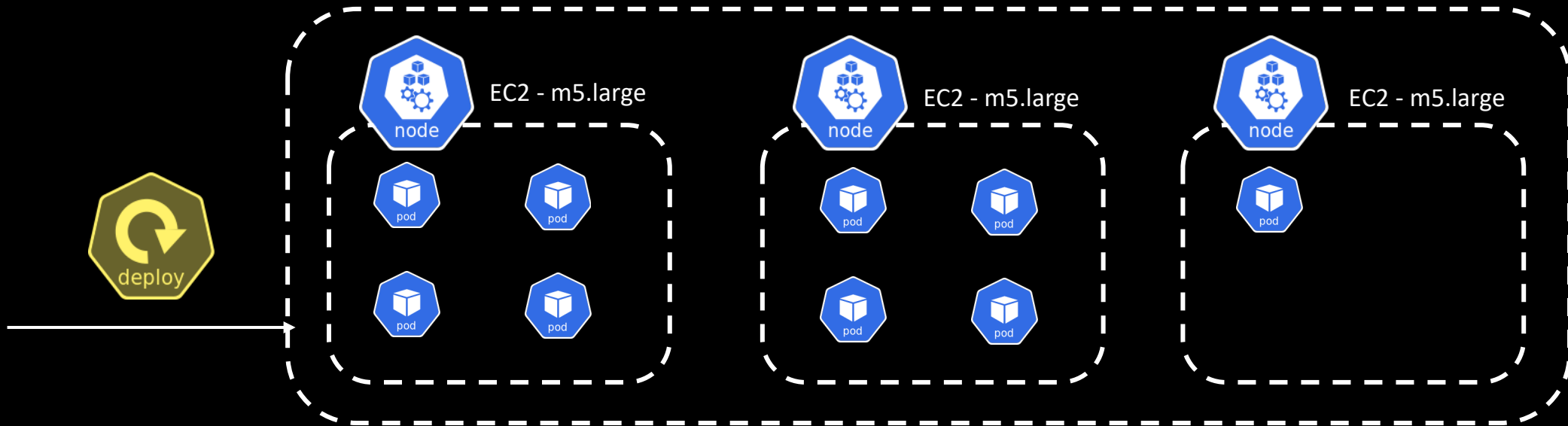


Node Group Management

Cluster Autoscaler

Auto Scaling group

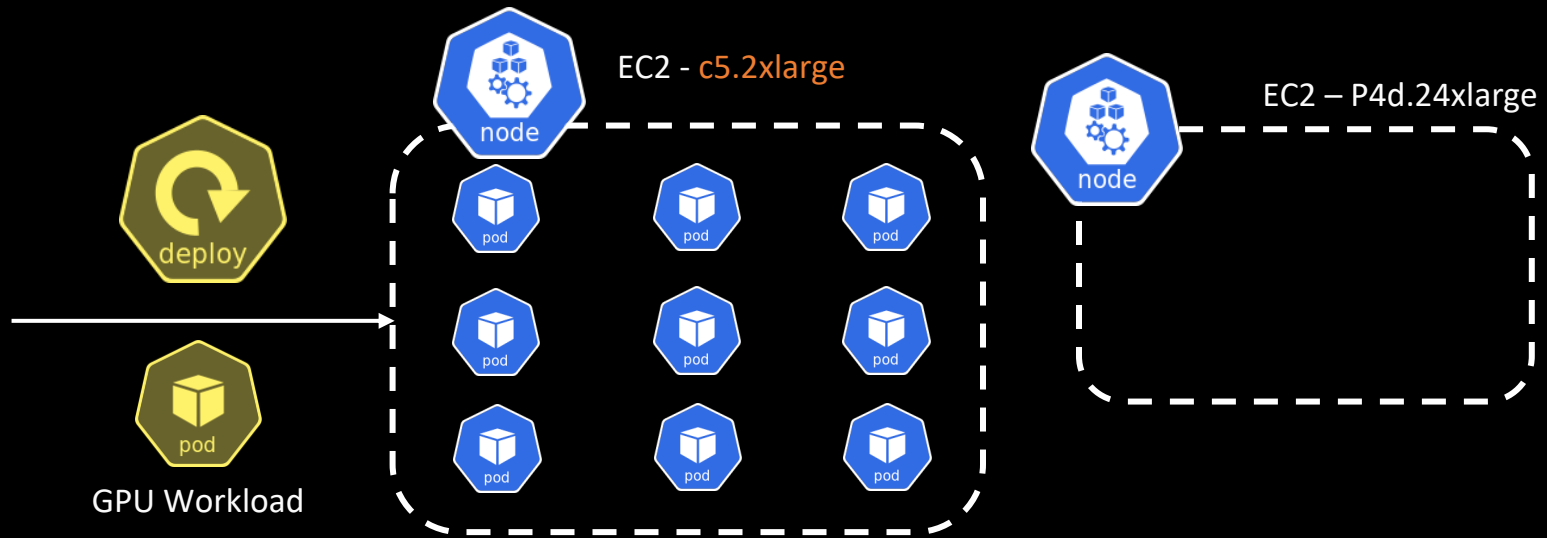
Compute Node Group



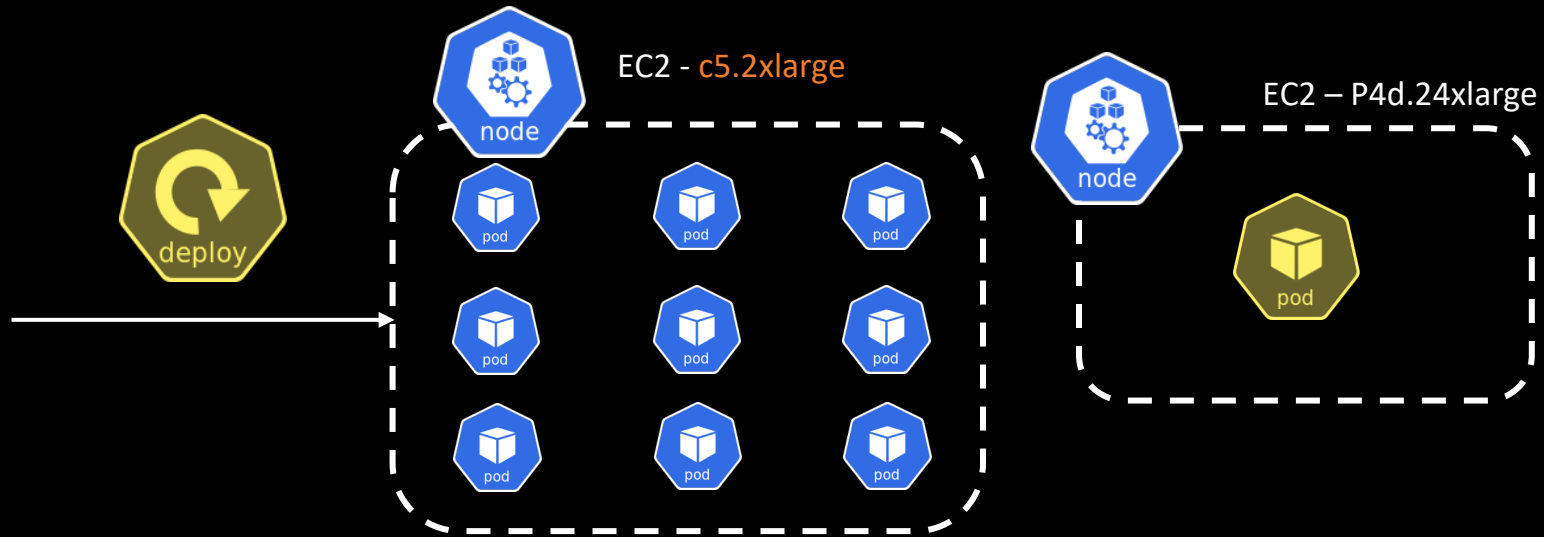
GPU Node Group



Karpenter



Karpenter



- Provision appropriate instances based on podspec without separate nodegroups
- Note – with ASG nodegroup, only compatible instance types can be used

Karpenter Proper Introduction!

What is Karpenter

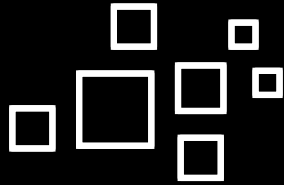


- Efficient Node Autoscaler for Kubernetes
- Automatically launches appropriate worker nodes without Node Groups
- Created by AWS
- Open-sourced
 - AWS Donated to CNCF (SIG-Autoscaling)
 - Adopted by Azure
- Karpenter is Kubernetes native
 - YAML support
 - Respects Kubernetes scheduling

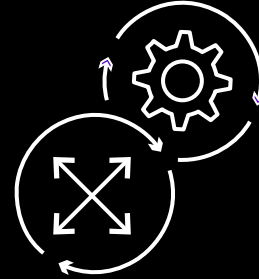
Karpenter does more than scaling



Cost optimization



Supports diverse
workloads including
ML and generative AI



Helps upgrade and
patching



Kubernetes native

Total data plane implementation

Karpenter is part of Kubernetes (OSS)

HPA Vs. Karpenter

HPA Vs. Karpenter



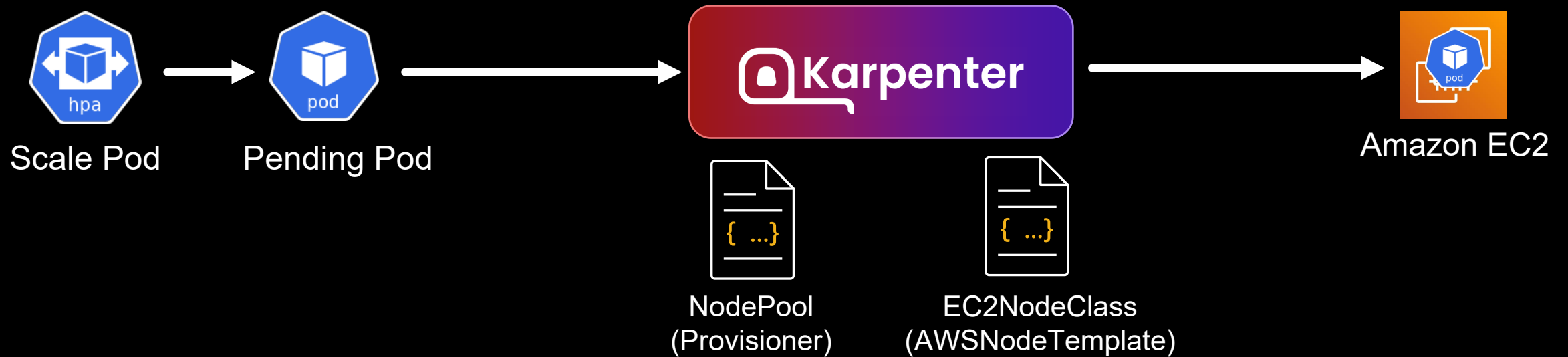
- Scales Pod
- Triggered by Pod Metric threshold – CPU, Memory, or Custom Metrics
- Part of Kubernetes distribution



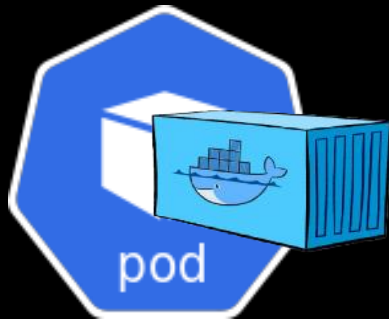
- Scales Node
- Triggered when kube-scheduler can't schedule pending pods in existing worker nodes
- CNCF project, need to be installed to the Kubernetes cluster
 - Created by AWS
- Karpenter does more than scaling – cost optimization, upgrade etc.

NodePool YAML

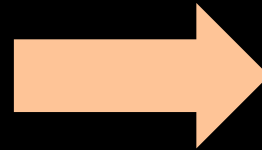
Karpenter YAMLs



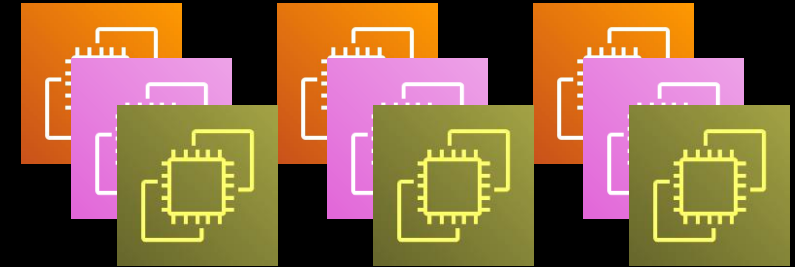
Containers scaling requirements



CPU
Memory
Custom metrics

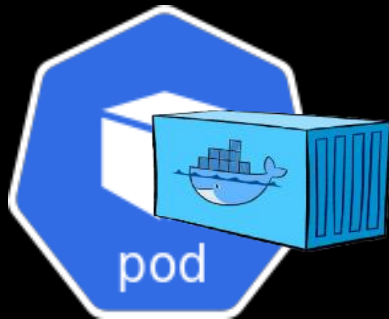


Amazon EC2

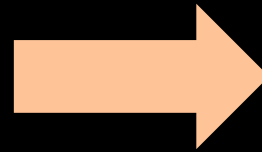


300+ instance types

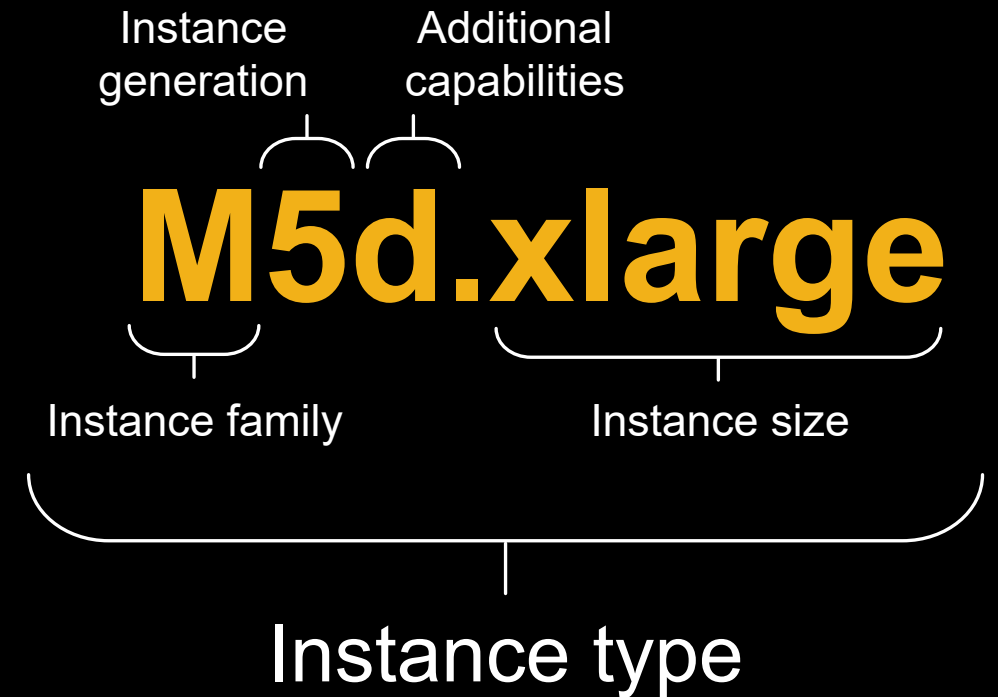
Containers scaling requirements



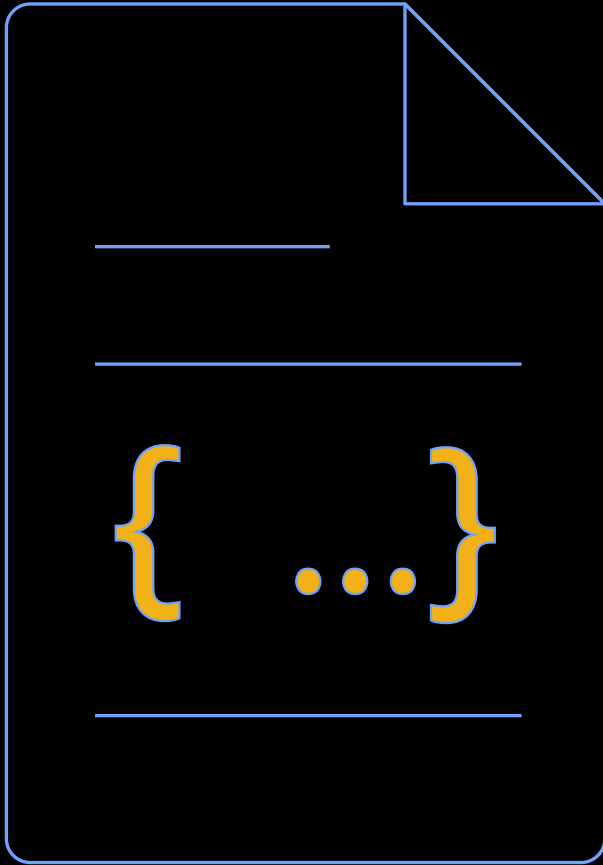
CPU
Memory
Custom metrics



Amazon EC2

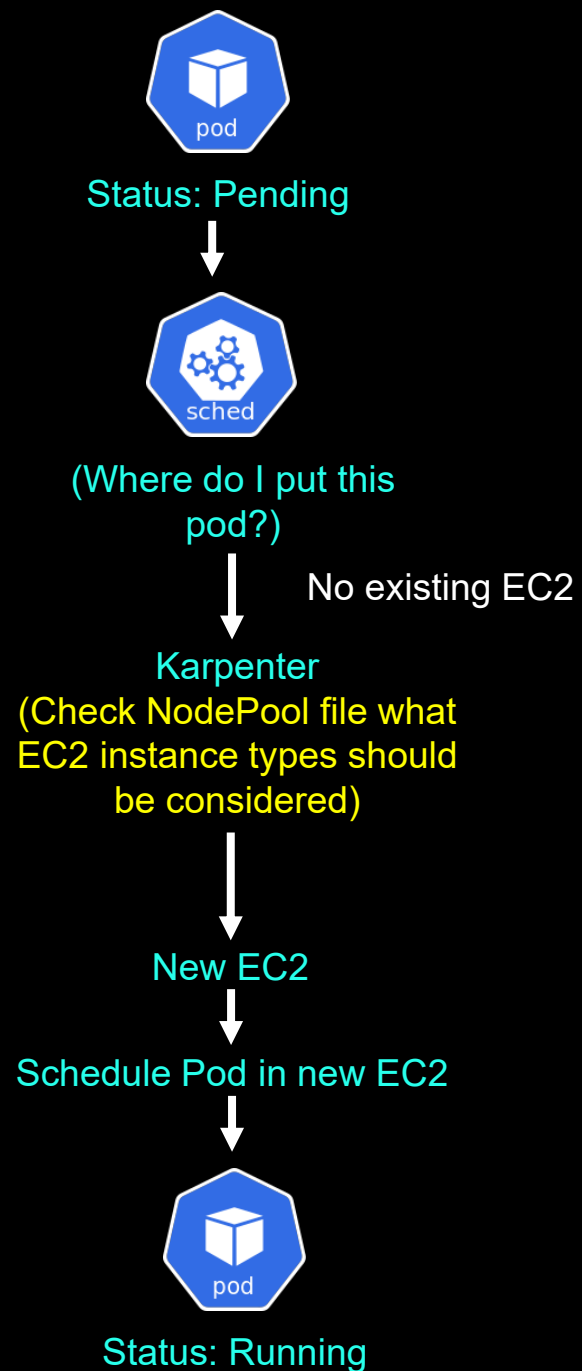
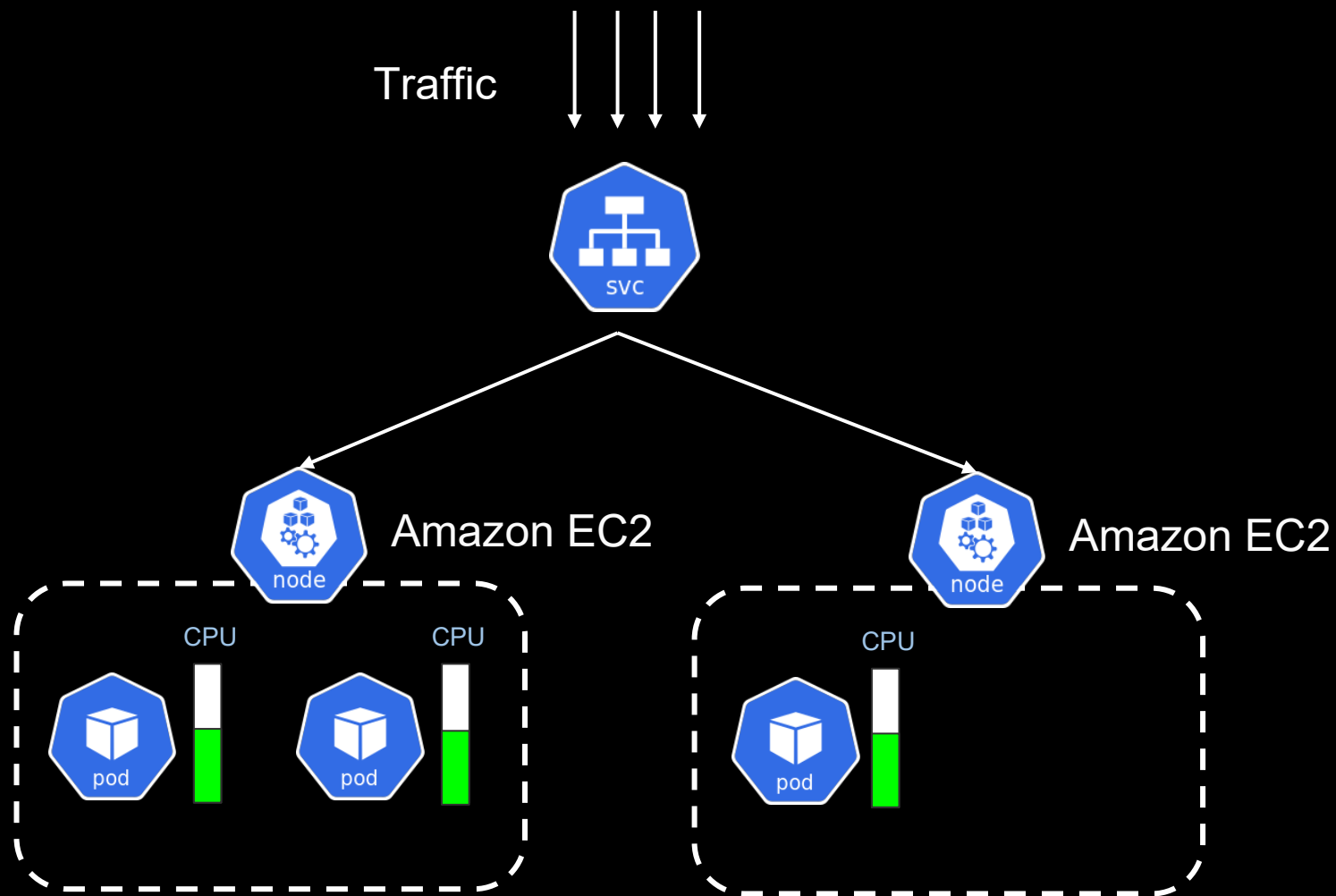


NodePool



- YAML file
- Defines what kind of nodes Karpenter will create
- Define instance types, CPU architecture, number of cores, certain AZs for nodes that Karpenter will respect
- One or multiple NodePools per Karpenter installation (More on this later)

Autoscaling Flow



NodePool YAML

Instance type flexibility

- Attribute-based requirements → sizes, families, generations, CPU architectures
- No list → picks from all instance types in EC2 universe, excluding metal
- Limits how many EC2 instances this NodePool can provision

AZ flexibility

- Provision in any AZ
- Provision in specified AZs

```
apiVersion: karpenter.sh/v1beta1
```

```
kind: NodePool
```

```
metadata:
```

```
  name: team-a
```

```
spec:
```

```
  template:
```

```
    spec:
```

```
      requirements:
```

```
- key: karpenter.k8s.aws/instance-family
```

```
  operator: In
```

```
  values: ["c5","m5","r5"]
```

```
- key: karpenter.k8s.aws/instance-size
```

```
  operator: NotIn
```

```
  values: ["nano","micro","small"]
```

```
- key: topology.kubernetes.io/zone
```

```
  operator: In
```

```
  values: ["us-west-2a","us-west-2b"]
```

```
- key: kubernetes.io/arch
```

```
  operator: In
```

```
  values: ["amd64","arm64"]
```

```
- key: karpenter.sh/capacity-type
```

```
  operator: In
```

```
  values: ["spot","on-demand"]
```

```
limits:
```

```
  cpu: 100
```

```
  memory: 1000Gi
```


NodePool YAML

CPU architecture flexibility

- Amd64 (x86)
- Arm64 (Graviton)

Purchase options flexibility

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

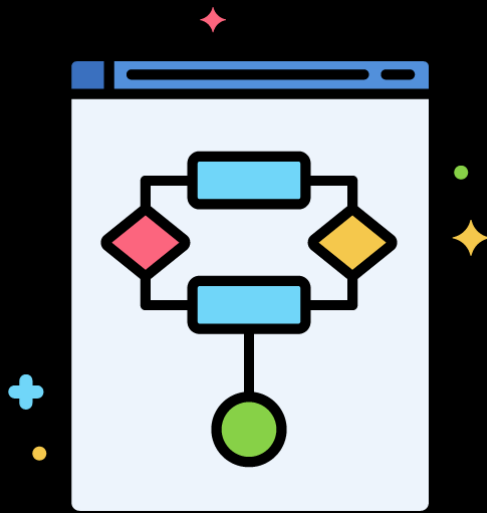
```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  template:
    spec:
      requirements:
        - key: karpenter.k8s.aws/instance-family
          operator: In
          values: ["c5", "m5", "r5"]
        - key: karpenter.k8s.aws/instance-size
          operator: NotIn
          values: ["nano", "micro", "small"]
        - key: topology.kubernetes.io/zone
          operator: In
          values: ["us-west-2a", "us-west-2b"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64", "arm64"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"]
      limits:
        cpu: 100
        memory: 1000Gi
```

NodePool YAML

Specify instance types directly

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  template:
    spec:
      requirements:
        - key: "node.kubernetes.io/instance-type"
          operator: In
          values: ["m5d.xlarge", "c6a.large"]
```

Spot interruption handling with Karpenter



Spot notification

- 2-minute Spot Instance interruption notice via Amazon EventBridge event
- Set as environment variables in Karpenter controller Deployment object

- NodePools can be configured for a mix of On-Demand and Spot (Spot is prioritized)
- Karpenter has built-in Spot interruption handler
- Not required to use Node Termination Handler

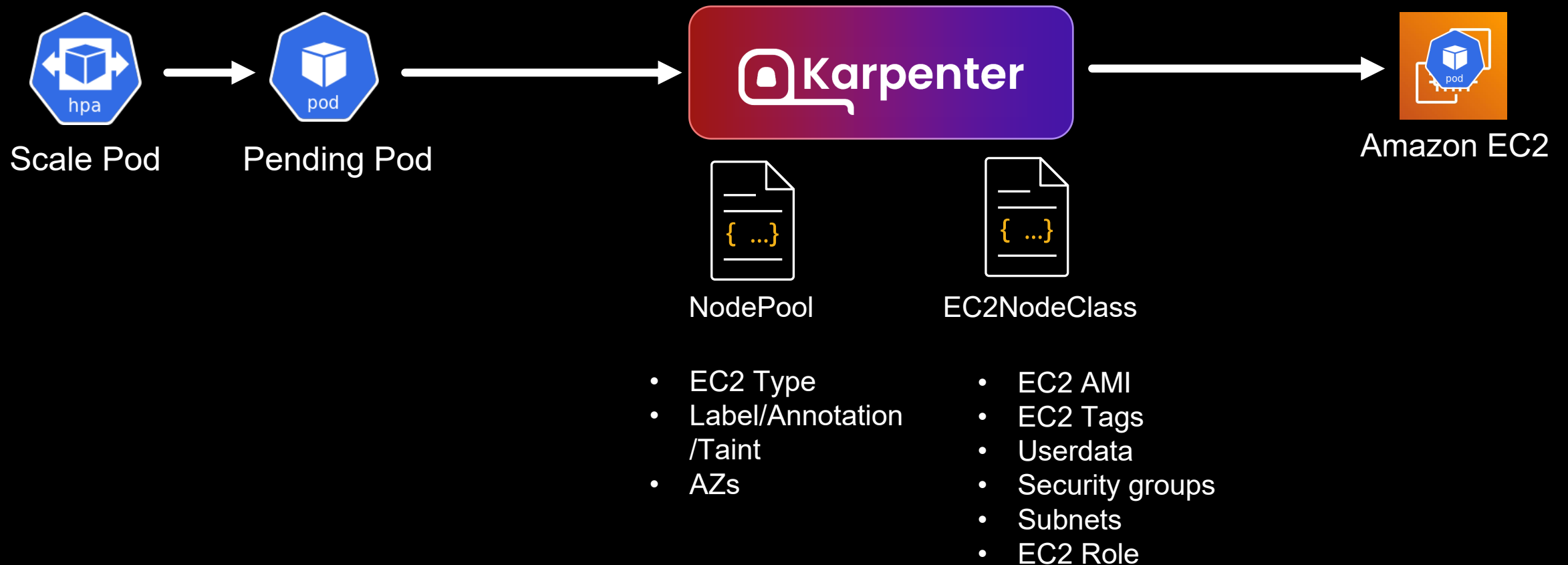
Spot and On-Demand Capacity with Karpenter



- On demand allocation strategy – lowest price
- Spot allocation strategy – price capacity optimized
 - Lowest cost but small capacity pool is not desirable
 - Balance between cost and probability of interruption

NodeClasses

Karpenter YAMLs



EC2NodeClass

- NodeClasses enable configuration of AWS specific settings
- Each NodePool must reference an EC2NodeClass
- Multiple NodePools can point to same EC2NodeClass

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  template:
    spec:
      nodeClassRef:
        apiVersion: karpenter.k8s.aws/v1beta1
        kind: EC2NodeClass
        name: app-a-nodeclass
```

```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: app-a-nodeclass
spec:
...
```

EC2NodeClass

Required

One of these two required

```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  amiFamily: AL2
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
      id: subnet-09fa4a0a8f233a921
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
      name: my-security-group
      id: sg-063d7acfb4b06c82c
  role: "KarpenterNodeRole-${CLUSTER_NAME}"
  instanceProfile: "KarpenterNodeInstanceProfile-${CLUSTER_NAME}"
```

If conditions NOT satisfied, no EC2 will be provisioned
(We can check if conditions are NOT satisfied using *status* field)

EC2NodeClass - amiFamily

- AMIs that Karpenter launch EC2s with
- Latest EKS-Optimized AMIs (Unless `amiSelectorTerms` specified)
- Values: AL2, Bottlerocket, Ubuntu, Windows2019, Windows2022, Custom

```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  amiFamily: AL2
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
    - id: subnet-09fa4a0a8f233a921
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
    - name: my-security-group
    - id: sg-063d7acfb4b06c82c
  role: "KarpenterNodeRole-${CLUSTER_NAME}"
  instanceProfile: "KarpenterNodeInstanceProfile-${CLUSTER_NAME}"
```

EC2NodeClass - amiSelectorTerms

- All requirements within a single selector (e.g. tags, id) are AND, each selector is OR
- Tags allows wildcard
- AMIs that Karpenter launch EC2s with
- If multiple AMIs selected, latest one is used

```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  amiFamily: AL2
  amiSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
      - name: my-ami
      - id: ami-123
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
      - id: subnet-09fa4a0a8f233a921
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
      - name: my-security-group
      - id: sg-063d7acfb4b06c82c
  role: "KarpenterNodeRole-${CLUSTER_NAME}"
```

EC2NodeClass - amiSelectorTerms

```
amiSelectorTerms:  
  - name: my-ami  
    owner: self  
  - name: my-ami  
    owner: 0123456789
```

```
spec:  
  amiSelectorTerms:  
    - name: "*EKS*"
```

```
amiSelectorTerms:  
  - tags:  
    karpenter.sh/discovery/MyClusterName: '*'
```

EC2NodeClass - subnetSelectorTerms

- Karpenter chooses from these subnets to launch EC2s in
- All requirements within a single selector (e.g. tags, id) are AND, each selector is OR
- Wildcards allowed in tags

```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  amiFamily: AL2
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
      - id: subnet-09fa4a0a8f233a921
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
      - name: my-security-group
      - id: sg-063d7acfb4b06c82c
  role: "KarpenterNodeRole-${CLUSTER_NAME}"
  instanceProfile: "KarpenterNodeInstanceProfile-${CLUSTER_NAME}"
```

EC2NodeClass - subnetSelectorTerms

```
spec:
  subnetSelectorTerms:
    - tags:
        Name: "my-subnet-1"
    - tags:
        Name: "my-subnet-2"
```

```
spec:
  subnetSelectorTerms:
    - tags:
        Name: "*Public*"
```

```
spec:
  subnetSelectorTerms:
    - id: "subnet-09fa4a0a8f233a921"
    - id: "subnet-0471ca205b8a129ae"
```

```
spec:
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery/MyClusterName: '*'
```

EC2NodeClass - securityGroupSelectorTerms

- Karpenter attaches the security groups to the provisioned EC2s
- All requirements within a single selector (e.g. tags, id) are AND, each selector is OR
- Wildcards allowed in tags
- All the discovered security groups are attached to the EC2s

```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  amiFamily: AL2
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
      id: subnet-09fa4a0a8f233a921
    securityGroupSelectorTerms:
      - tags:
          karpenter.sh/discovery: "${CLUSTER_NAME}"
          environment: test
        - name: my-security-group
        - id: sg-063d7acfb4b06c82c
  role: "KarpenterNodeRole-${CLUSTER_NAME}"
  instanceProfile: "KarpenterNodeInstanceProfile-${CLUSTER_NAME}"
```

EC2NodeClass – Notable Fields

Others

- blockDeviceMappings
- userData

```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: payment-app-nodeclass
spec:
  amiFamily: AL2
  amiSelectorTerms:
    - id: $AMI_OLD
  role: karpenterNodeRole-$CLUSTER_NAME
  securityGroupSelectorTerms:
    - tags:
        alpha.eksctl.io/cluster-name: $CLUSTER_NAME
  subnetSelectorTerms:
    - tags:
        alpha.eksctl.io/cluster-name: $CLUSTER_NAME
  tags:
    app: payment
    managed-by: karpenter
```

NodePool Vs. EC2NodeClass

NodePool Vs. EC2NodeClass

kind: NodePool

- Required
- YAML file in the Kubernetes cluster where Karpenter is installed
- Tells Karpenter what type of EC2 instance to be launched – instance family, spot/on-demand, graviton/x86, number of cores etc.

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: team-a
spec:
  template:
    metadata:
      labels:
        appteam: team-a
  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: ["spot", "on-demand"]
```

kind: EC2NodeClass

- Required
- YAML file in the Kubernetes cluster where Karpenter is installed
- Tells Karpenter about actual cloud (AWS) constructs related to the instance – AMI, security group, subnet, user data, EBS volume, instance role etc.

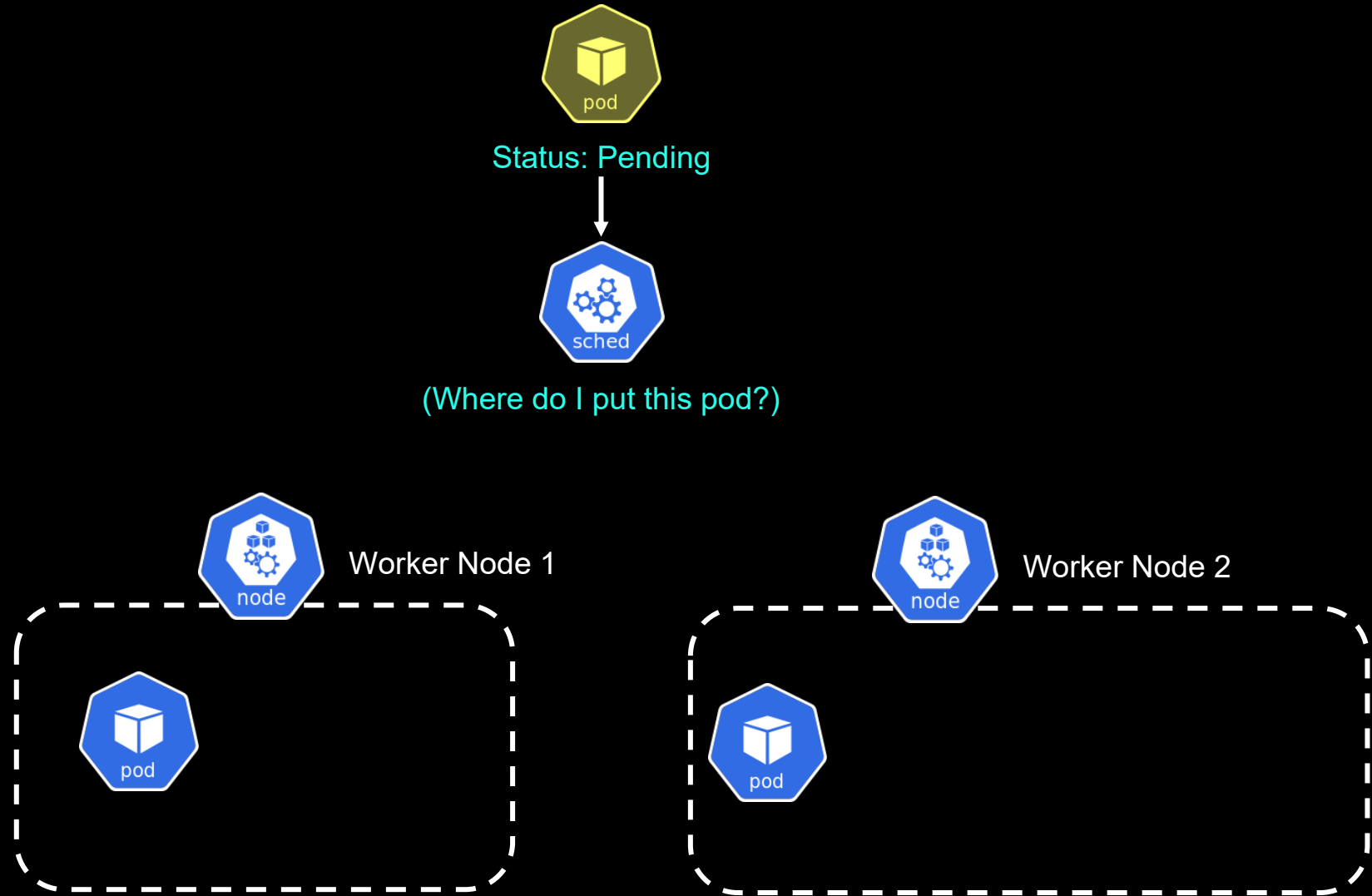
```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  amiFamily: AL2
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
      id: subnet-09fa4a0a8f233a921
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
        environment: test
      name: my-security-group
      id: sg-063d7acfb4b06c82c
  role: "KarpenterNodeRole-${CLUSTER_NAME}"
  instanceProfile: "KarpenterNodeInstanceProfile-${CLUSTER_NAME}"
```

- Multiple NodePool can point to a single EC2NodeClass (`nodeClassRef` field in NodePool)
- Both NodePool and EC2NodeClass must be satisfied for Karpenter launching EC2s (`status` field)

Demo – NodePool and Ec2NodeClass

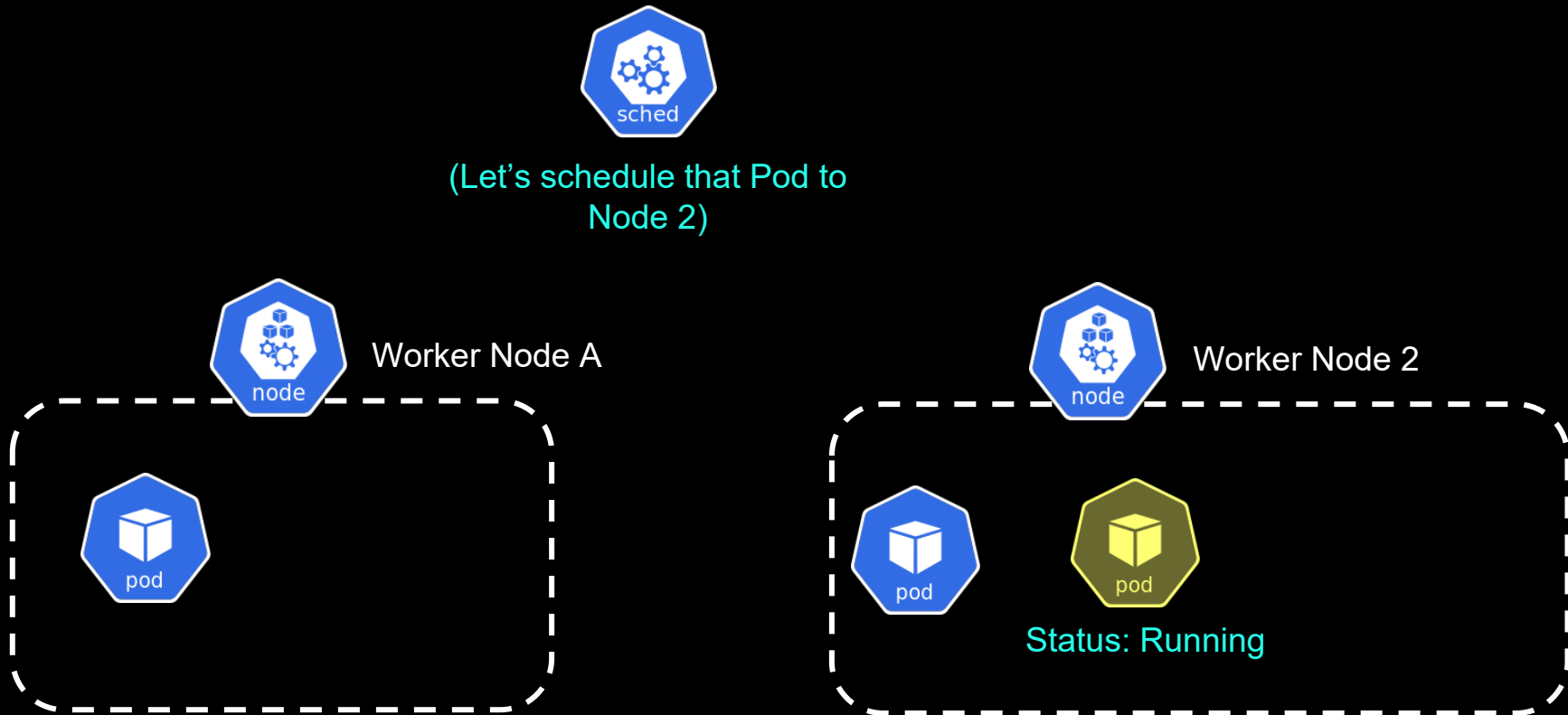
Kubernetes Scheduling with Karpenter

Kubernetes Scheduling



Kubernetes Scheduling

The Kubernetes scheduler is a control plane process which assigns Pods to Nodes. The scheduler determines which Nodes are valid placements for each Pod in the scheduling queue according to constraints and available resources. The scheduler then ranks each valid Node and binds the Pod to a suitable Node.



Why Control Kubernetes Scheduling?



- Run certain workloads in certain EC2 instance types
 - Spot, GPU, Graviton etc.
- Make workload highly-available
 - Atleast two replica, with each one in EC2 in unique AZ (using topology spread)
- Run certain workloads for one team in separate worker nodes

Scheduling Using nodeSelector

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  nodeSelector:
    team: team-a
```



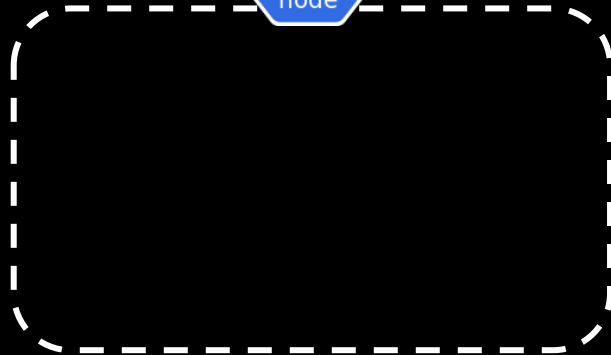
Use labels to schedule pods
for different apps

(FYI Kubernetes Labels are NOT Tags)

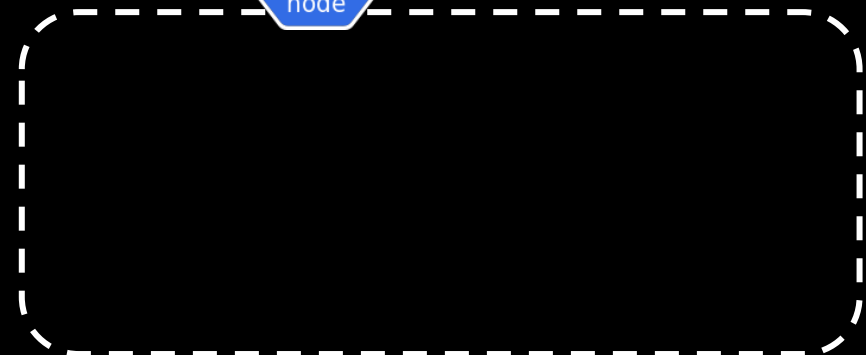
```
kubectl get nodes --show-labels
```



team=team-a
Worker Node 1



Worker Node 2

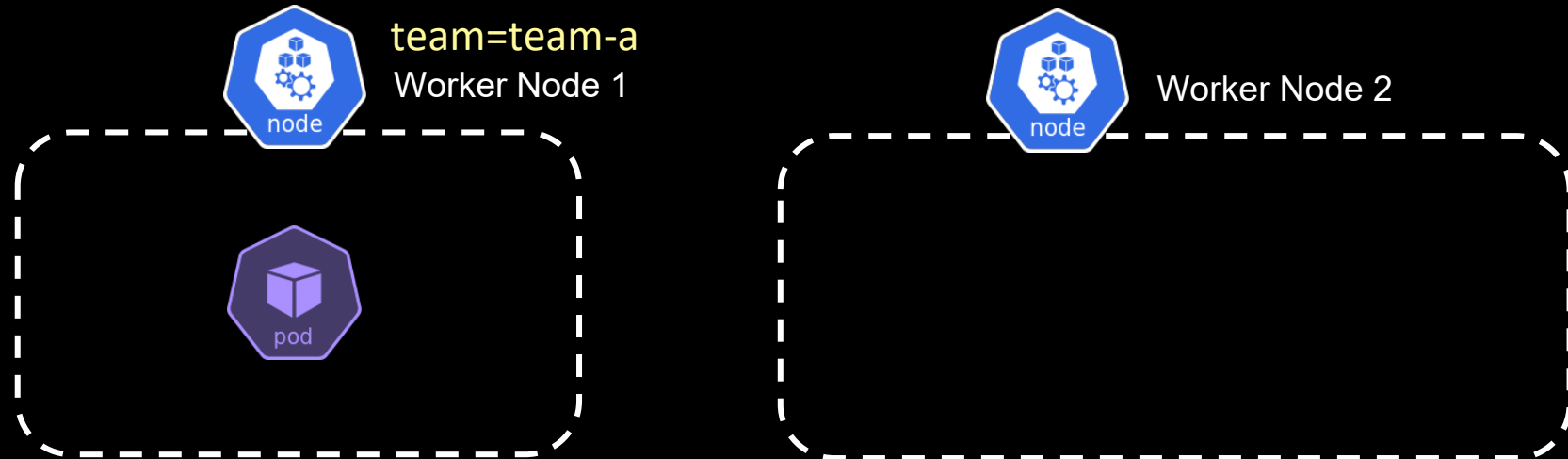


Scheduling Using nodeSelector

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  nodeSelector:
    team: team-a
```

Use labels to schedule pods for different apps

```
kubectl get nodes --show-labels
```



Who puts labels on the node?

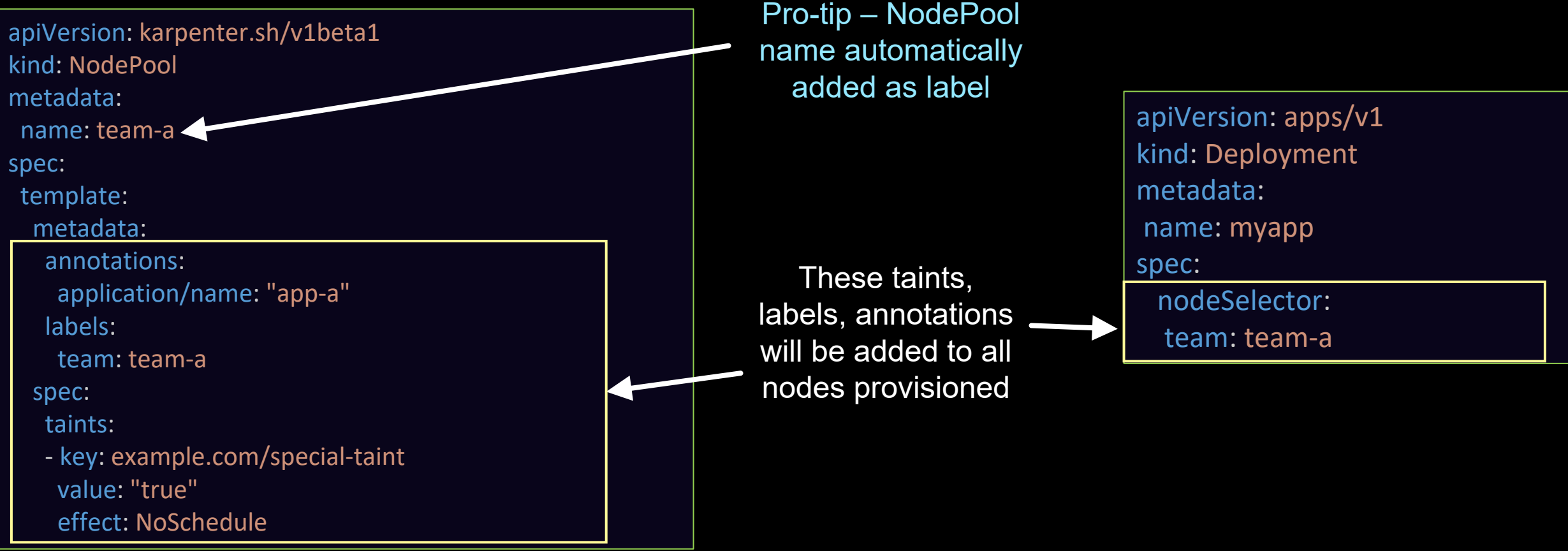
Who puts labels on the node?
Who puts annotations on the node?
Who puts taints on the node?

Should be during Node provision

Karpenter NodePool

User-defined annotation, labels, taints

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: team-a
```



```
spec:
  template:
    metadata:
      annotations:
        application/name: "app-a"
      labels:
        team: team-a
    spec:
      taints:
        - key: example.com/special-taint
          value: "true"
          effect: NoSchedule
```

Pro-tip – NodePool
name automatically
added as label

These taints,
labels, annotations
will be added to all
nodes provisioned

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  nodeSelector:
    team: team-a
```

Use labels to schedule pods for different apps

Sample well-known labels added to nodes

Label	Example
topology.kubernetes.io/zone	us-east-2a
node.kubernetes.io/instance-type	g4dn.8xlarge
kubernetes.io/os	linux
kubernetes.io/arch	amd64
karpenter.sh/capacity-type	spot
karpenter.k8s.aws/instance-hypervisor	nitro
karpenter.k8s.aws/instance-encryption-in-transit-supported	true
karpenter.k8s.aws/instance-category	g

And more . . .

AZ Scheduling

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
```

```
  nodeSelector:
    topology.Kubernetes.io/zone: us-west-2a
```

This pod spec will make NodePool to create an EC2 in AZ us-west-2a and add that as label

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: team-a
spec:
  template:
    spec:
      requirements:
        - key: karpenter.k8s.aws/instance-family
          operator: In
          values: ["c5", "m5", "r5"]
        - key: karpenter.k8s.aws/instance-size
          operator: NotIn
          values: ["nano", "micro", "small"]
        - key: topology.kubernetes.io/zone
          operator: In
          values: ["us-west-2a", "us-west-2b"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64", "arm64"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"]
      limits:
        cpu: 100
        memory: 1000Gi
```

Karpenter respects scheduling constraints

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  template:
    spec:
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64", "arm64"]
```

These
labels
added to
the nodes

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: karpenter.sh/capacity-type
                operator: In
                values:
                  - spot
```

- Adds label karpenter.sh/capacity-type: spot, karpenter.io/arch: amd64 to nodes
- Utilize nodeSelector, nodeAffinity to schedule pods into appropriate nodes

Karpenter Supports GPUs

```
spec:
  template:
    spec:
      containers:
      - resources:
          limits:
            nvidia.com/gpu: "1"
```

GPU values supported

- nvidia.com/gpu
- amd.com/gpu
- aws.amazon.com/neuron
- habana.ai/audi

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: gpu
spec:
  template:
    spec:
      requirements:
      - key: karpenter.k8s.aws/instance-family
        operator: In
        values:
          - p3
```

Additional Steps Required: <https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/#deploying-amd-gpu-device-plugin>

Taint and Toleration with Karpenter

```
apiVersion: v1
kind: Pod
metadata:
  name: mygpupod
spec:
  containers:
  - name: gpuapp
    resources:
      requests:
        nvidia.com/gpu: 1
      limits:
        nvidia.com/gpu: 1
    image: mygpucontainer
  tolerations:
  - key: "nvidia.com/gpu"
    operator: "Exists"
    effect: "NoSchedule"
```

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: gpu
spec:
  template:
    spec:
      requirements:
      - key: karpenter.k8s.aws/instance-family
        operator: In
        values:
        - p3
      taints:
      - key: nvidia.com/gpu
        value: true
        effect: "NoSchedule"
```

TopologySpread with Karpenter

```
spec:
  topologySpreadConstraints:
    - maxSkew: 1
      topologyKey: "topology.kubernetes.io/zone"
      whenUnsatisfiable: ScheduleAnyway
      labelSelector:
        matchLabels:
          appteam: team-a
```

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: team-a
spec:
  template:
    metadata:
      labels:
        appteam: team-a
    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: ["spot","on-demand"]
```


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 and
on-demand capacity

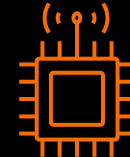
Standard K8s pod scheduling mechanisms



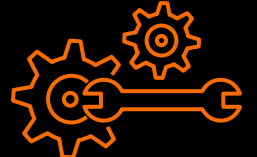
Node
selectors



Node
affinity



Taints and
tolerations



Topology
spread

**Pod scheduling constraints must fall
within a provisioner's constraints**

Karpenter is Kubernetes Native



One or Many NodePool?



Managed by
AWS

Control Plane

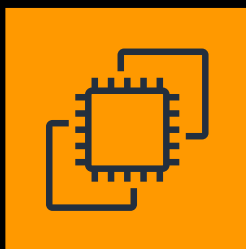


Amazon Elastic Kubernetes
Service

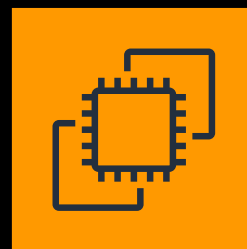
Karpenter



Amazon EC2



Amazon EC2



Amazon EC2

Data Plane

One Karpenter installation cover entire cluster.
It can support multi-tenancy

Strategies for defining NodePools

Single

A single NodePool can manage compute for multiple teams and workloads

Example use cases:

- Single NodePool for a mix of Graviton and x86, while a pending pod has a requirement for a specific processor type

Multiple

Isolating compute for different purposes

Example use cases:

- Expensive hardware
- Security isolation
- Team separation
- Different AMI
- Tenant isolation due to noisy neighbor

Weighted

Define order across your NodePools so that the node scheduler will attempt to schedule with one NodePool before another

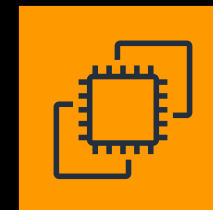
Example use cases:

- Prioritize RI and Savings Plan ahead of other instance types
- Default clusterwide configuration
- Ratio split – Spot/OD, x86/Graviton

Single NodePool

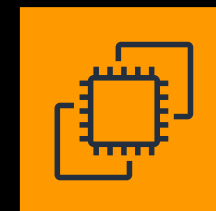
```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: team-all
spec:
  template:
    spec:
      requirements:
        - key: karpenter.k8s.aws/instance-family
          operator: In
          values: ["c5", "m5", "r5", "g5", "p5"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64", "arm64"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"]
      limits:
        cpu: 600
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-team-a
spec:
  nodeSelector:
    Kubernetes.io/arch: amd64
    karpenter.sh/capacity-type: spot
```



Spot X86 EC2

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-team-b
spec:
  nodeSelector:
    Kubernetes.io/arch: arm64
    karpenter.sh/capacity-type: on-demand
```



On-demand
Graviton
EC2

Multiple NodePools

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: team-a
spec:
  template:
    spec:
      requirements:
        - key: karpenter.k8s.aws/instance-family
          operator: In
          values: ["c5", "m5", "r5"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot"]
      limits:
        cpu: 500
```

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: team-b
spec:
  template:
    spec:
      requirements:
        - key: karpenter.k8s.aws/instance-family
          operator: In
          values: ["g5", "p5"]
        - key: kubernetes.io/arch
          operator: In
          values: ["arm64"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
      limits:
        cpu: 100
```

- If Pod scheduling constraints matches both NodePools, alphabetically ascending NodePool will be picked

User-defined annotation, labels, taints

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: team-a
spec:
  template:
    metadata:
      labels:
        team: team-a
```

Pro-tip – NodePool
name automatically
added as label

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  nodeSelector:
    team: team-a
```

Use labels to schedule pods for different apps

Weighted NodePools

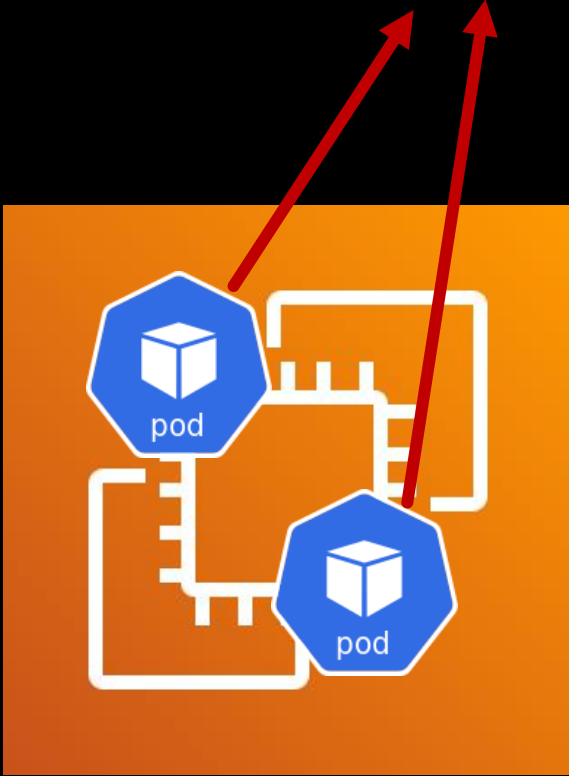
```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: compute-savings-plan-nodepool
spec:
  template:
    requirements:
      - key: "karpenter.k8s.aws/instance-category"
        operator: In
        values: ["c", "r"]
      - key: "karpenter.k8s.aws/instance-cpu"
        operator: In
        values: ["16", "32"]
      - key: "karpenter.k8s.aws/instance-hypervisor"
        operator: In
        values: ["nitro"]
    limits:
      cpu: "1000"
      memory: 1000Gi
  weight: 60
```

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: general-instances-nodepool
spec:
  template:
    requirements:
      - key: "karpenter.k8s.aws/instance-cpu"
        operator: In
        values: ["16", "32"]
  weight: 40
```

- Karpenter prioritizes NodePool with higher weight

Karpenter Disruption

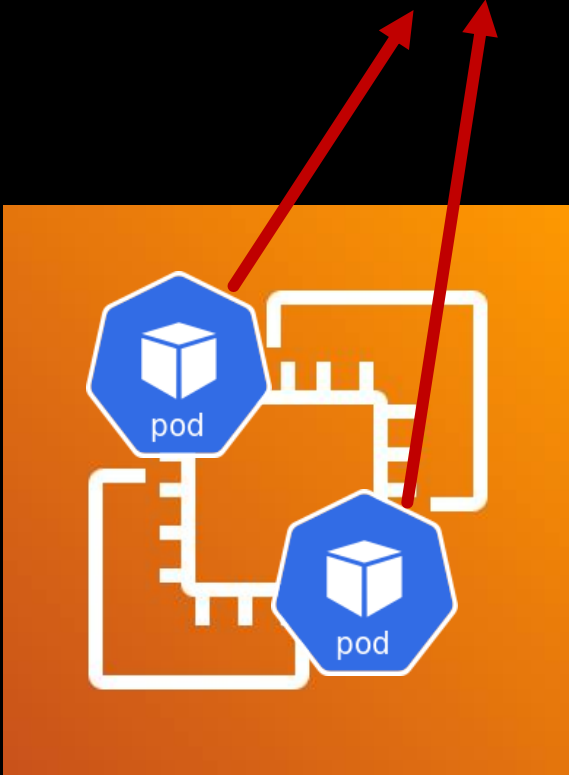
Karpenter Disruption



EC2 To Be Deleted

- Karpenter disrupts (deletes) nodes appropriately
- Voluntary disruption
 - Expiration
 - Drift
 - Consolidation
- Involuntary disruption
 - Spot interruption
 - EC2 Health events
 - Instance stop/termination events

Karpenter Disruption

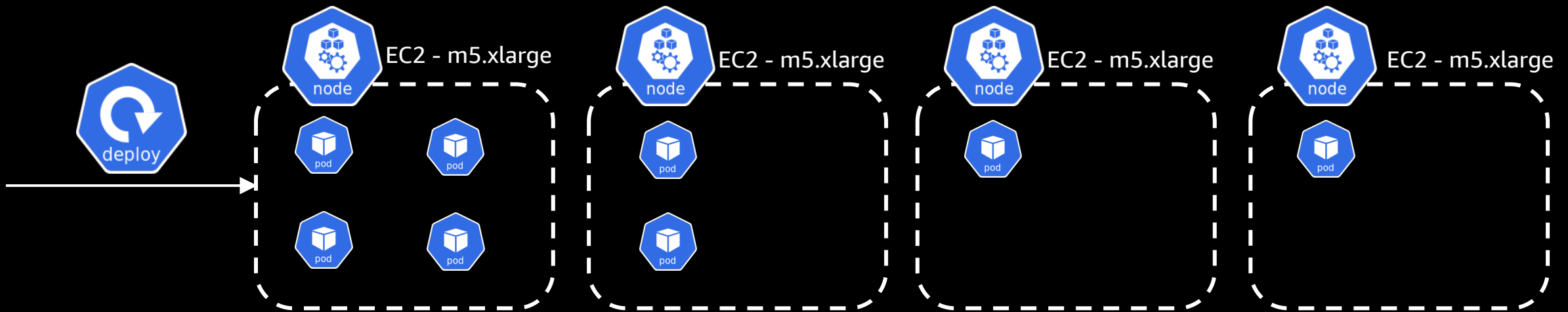


EC2 To Be Deleted

- Execute scheduling simulation and provisions replacement node if needed
- Karpenter prevents pods to schedule in disrupted nodes by adding taint `karpenter.sh/disruption:NoSchedule`
- Eviction happens through Kubernetes eviction API
- Disruption respects Pod Disruption Budgets (PDBs)
 - *May be violated for involuntary disruption*
- Karpenter terminates disrupted node

Consolidation

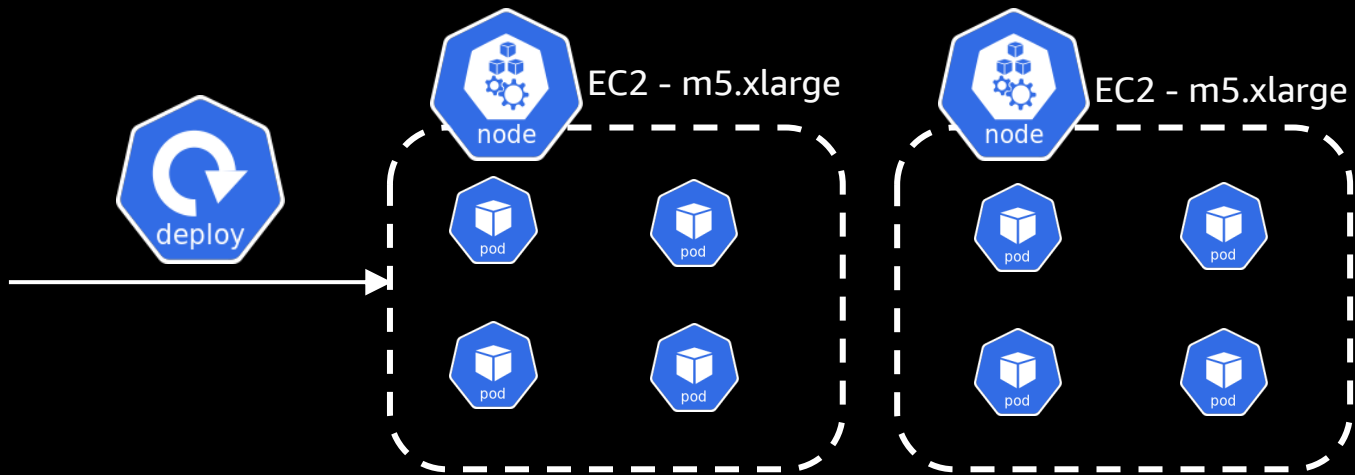
Karpenter optimization



Enable consolidation

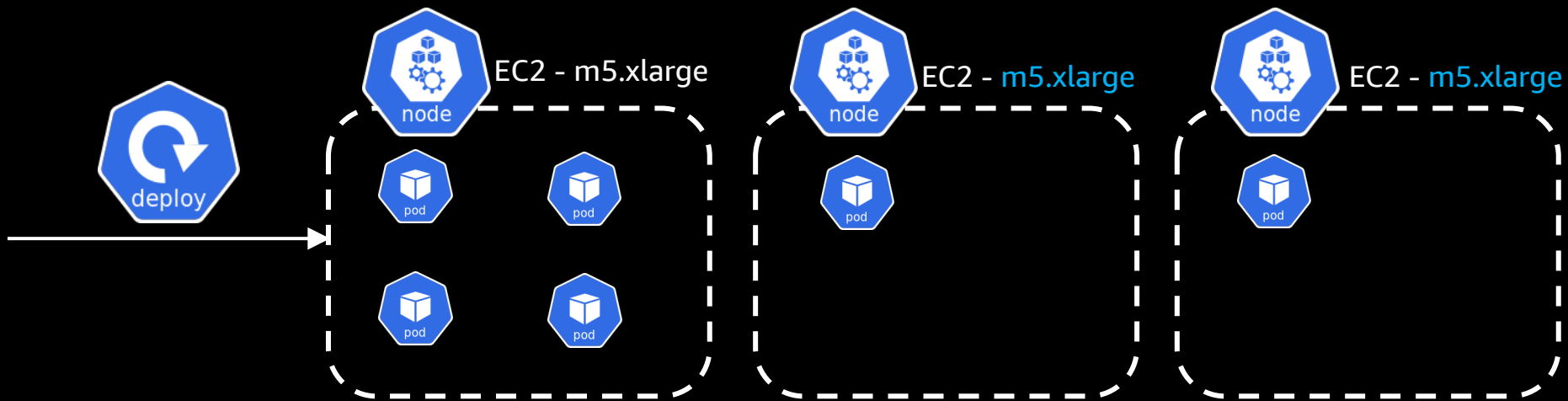
```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
spec:
  disruption:
    consolidationPolicy: WhenUnderutilized
```

Karpenter optimization



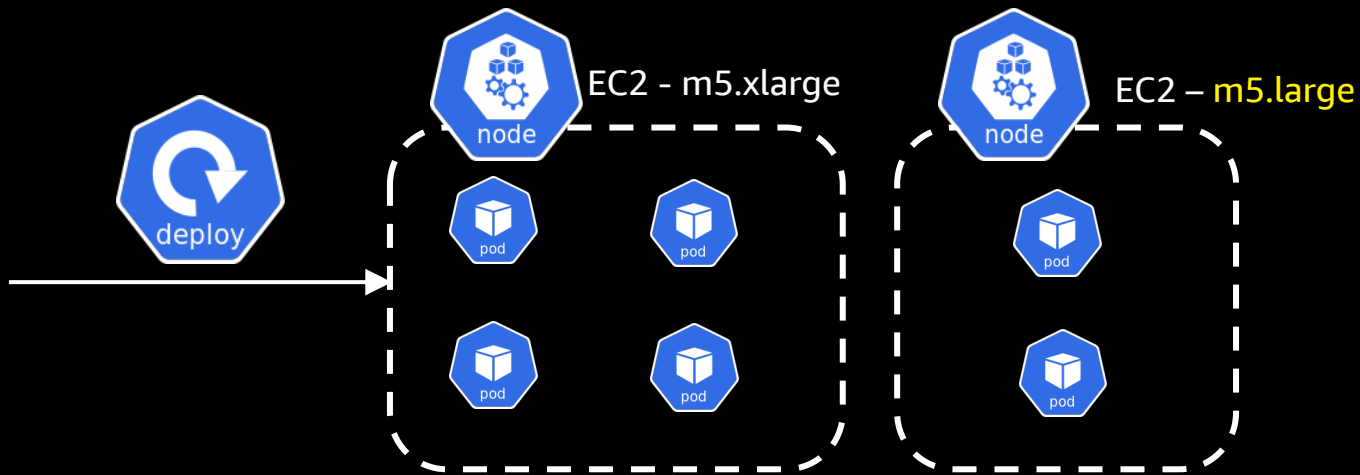
Better utilization of worker nodes – reduced cost

Karpenter optimization



Enable consolidation

Karpenter optimization – Pick cheaper nodes



Better selection of worker nodes – reduced cost

Consolidation

- Karpenter works to reduce cluster cost by:
 - Removing empty nodes
 - Removing node(s) by moving the pods to another underutilized node(s)
 - Replacing nodes with cheaper variants
- `consolidateAfter` - How many seconds to wait to scale nodes down due to low utilization
- `consolidationPolicy` - `WhenEmpty` | `WhenUnderutilized`
 - `WhenEmpty` – Karpenter will only consider nodes for consolidation that contain no workload pods
 - `WhenUnderutilized` – Karpenter will attempt to remove or replace nodes when a node is underutilized and could be changed to reduce cost

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  disruption:
    consolidateAfter: 30s
    consolidationPolicy: WhenEmpty
    expireAfter: Never
  limits:
    cpu: "10"
  template:
    metadata:
      labels:
        eks-immersion-team: my-team
    spec:
      requirements:
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["c", "m", "r"]
```

Consolidation – YAML Variation

- Karpenter works to reduce cluster cost by:
 - Removing empty nodes
 - Removing node(s) by moving the pods to another underutilized node(s)
 - Replacing nodes with cheaper variants
- `consolidateAfter` - How many seconds to wait to scale nodes down due to low utilization
- `consolidationPolicy` - `WhenEmpty` | `WhenUnderutilized`
 - `WhenEmpty` – Karpenter will only consider nodes for consolidation that contain no workload pods
 - `WhenUnderutilized` – Karpenter will attempt to remove or replace nodes when a node is underutilized and could be changed to reduce cost

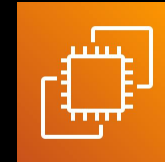
```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  disruption:
    consolidationPolicy: WhenUnderutilized
  limits:
    cpu: "10"
  template:
    metadata:
      labels:
        eks-immersion-team: my-team
    spec:
      requirements:
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["c", "m", "r"]
```

Upgrading/Patching with Drift

Karpenter Drift

- Karpenter CRD values do not match running machines
- Triggered by machine/node/instance changes or *NodePool/EC2NodeClass* changes
- Enable Drift in *karpenter-global-settings* configmap (auto enabled after v0.32.x)

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: reserved-instance
spec:
  template:
    spec:
      requirements:
        - key: node.kubernetes.io/instance-type
          operator: In
          values: ["m5.large"]
```



Amazon Elastic Compute
Cloud (Amazon EC2)
(m5.large)

Patching and upgrading with Drift

- Karpenter CRD values do not match running machines
- Triggered by machine/node/instance changes or *NodePool/EC2NodeClass* changes
- Enable Drift in *karpenter-global-settings* configmap (auto enabled after v0.32.x)

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: reserved-instance
spec:
  template:
    spec:
      requirements:
      - key: node.kubernetes.io/instance-type
        operator: In
        values: ["c5.large"]
```

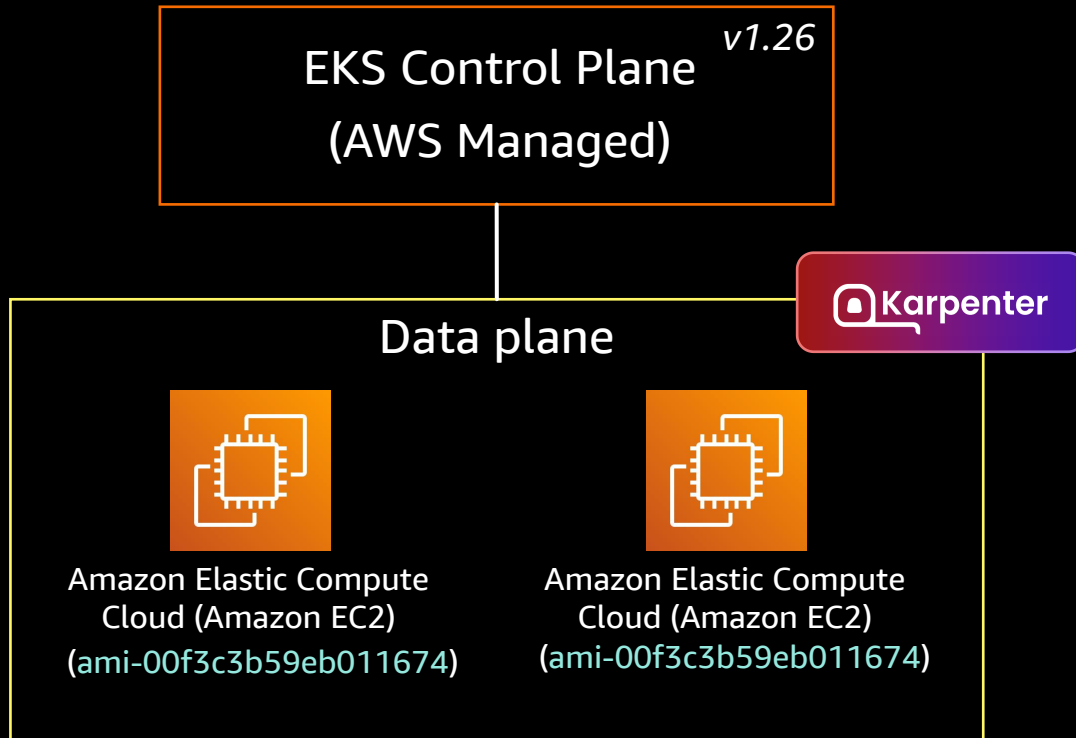


Selecting AMI and more with EC2NodeClass

```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
  amiFamily: AL2
  amiSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}"
  userData: |
    echo "Hello world"
  role: "KarpenterNodeRole-${CLUSTER_NAME}"
  tags:
    team: team-a
  blockDeviceMappings: {...}
  metadataOptions: {...}
```

- Select which subnets are eligible to launch EC2
- Attach security groups to EC2
- Select AMI for EC2
- By default, AL2 is prepopulated
- Supported AMIs: AL2, BR, Ubuntu, Win 19/22, custom
- Run userData, define EBS, attach tags, and more

Patching/upgrading with Drift – Default AMI

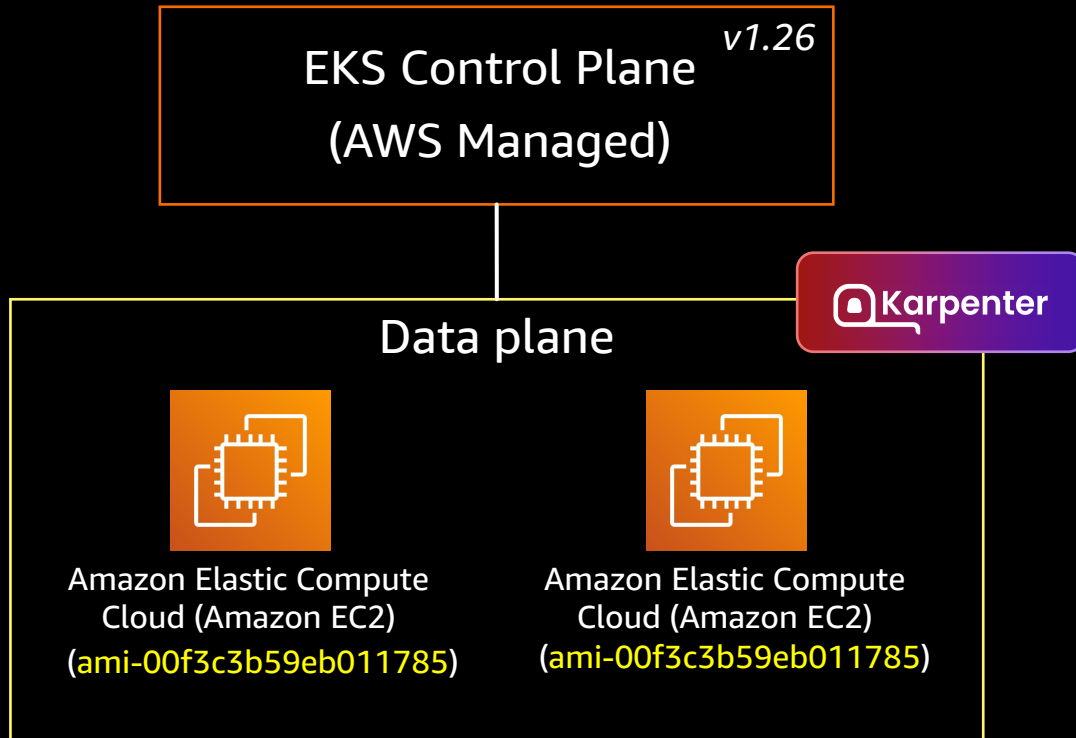


```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  amiFamily: AL2
```

EC2NodeClass default behavior

- Amazon EKS Optimized Linux AMI is used (for v1.26)
- AWS publishes recommended AMIs for each version in AWS Systems Manager
- Karpenter selects the AMI from the parameter store

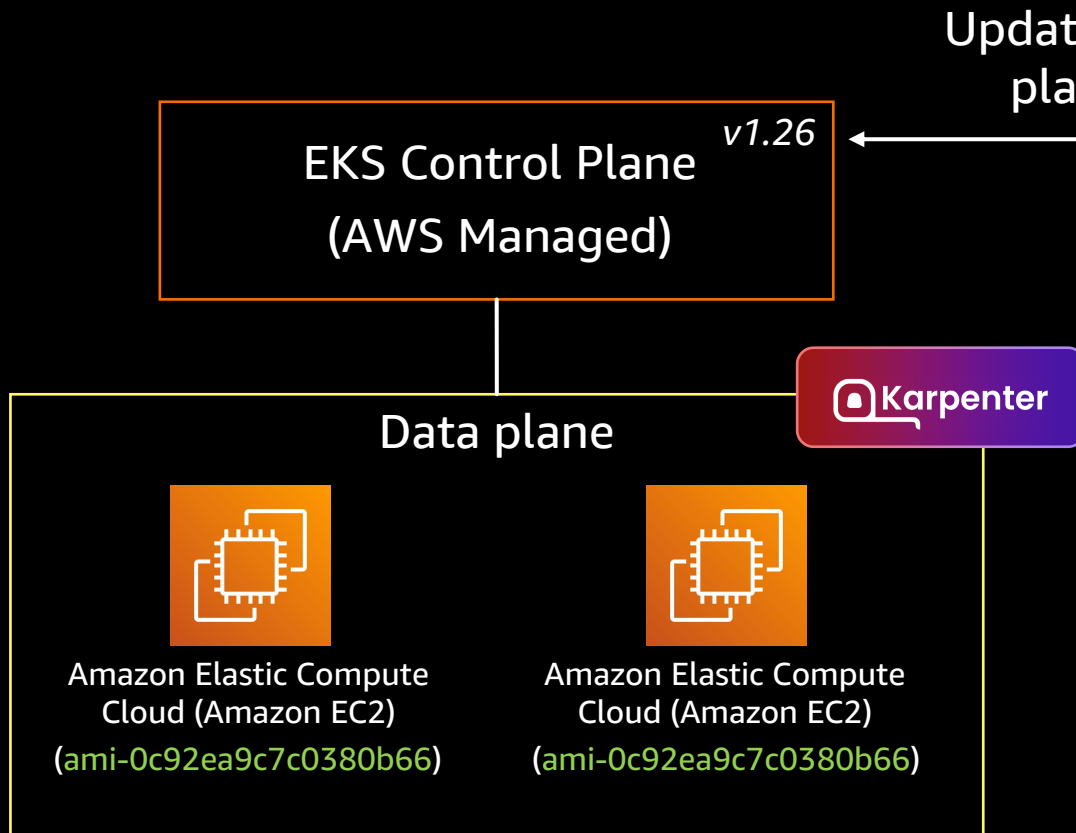
Patching/upgrading with Drift – Default AMI



EC2NodeClass default behavior

- AWS releases a new AMI for the same EKS version
- AWS updates recommended AMI in AWS Systems Manager
- Karpenter monitors the parameter store
- Karpenter updates the worker nodes AMIs automatically
- Done in rolling deployment fashion

Patching/upgrading with Drift – Default AMI



EC2NodeClass default behavior

- Karpenter monitors the parameter store
- Karpenter selects the recommended AMI for the new EKS version
- Karpenter updates the worker nodes AMIs automatically
- Done in rolling deployment fashion

Zero touch and secure – always latest EKS optimized AMI

Patching/upgrading with Drift – Custom AMI

- Custom AMIs are used in your application
- Select AMIs using *amiSelector* field in *EC2NodeClass*
- If multiple AMIs found, Karpenter will use the latest one
- If no AMIs found, no nodes will be provisioned
- *EC2NodeClass status.amis* field lists discovered AMIs

```
amiSelectorTerms:
```

```
- tags:
```

```
  Name: my-ami
```

```
amiSelectorTerms:
```

```
- name: my-ami
```

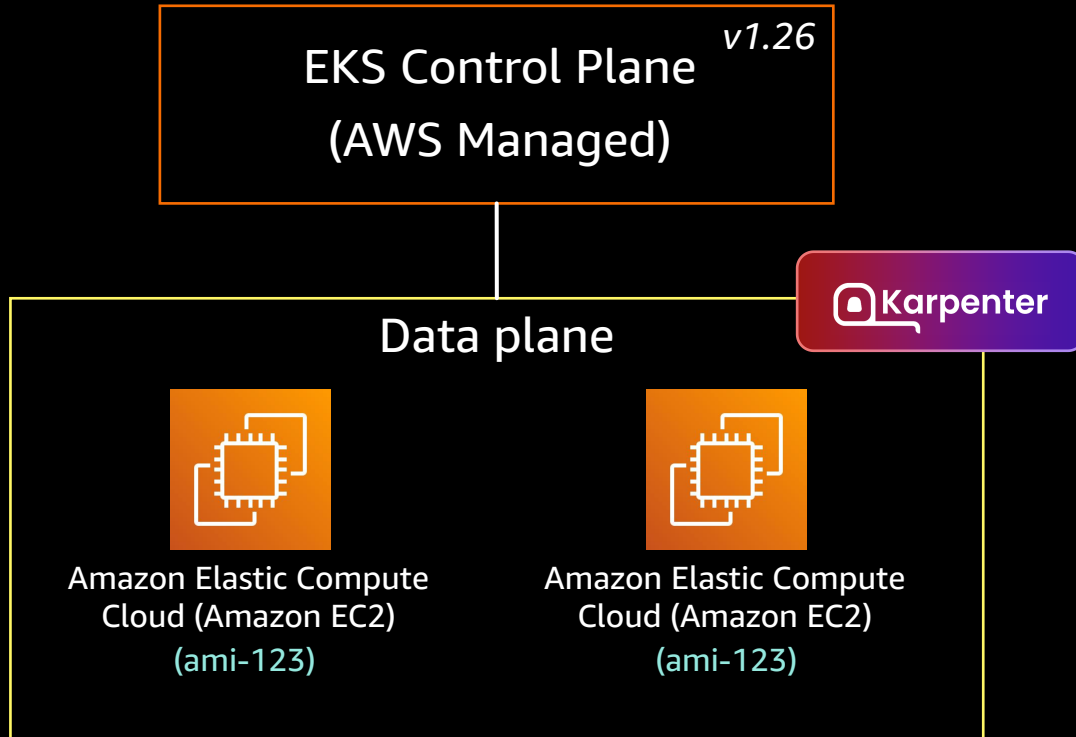
```
  owner: 0123456789
```

```
amiSelectorTerms:
```

```
- id: "ami-123"
```

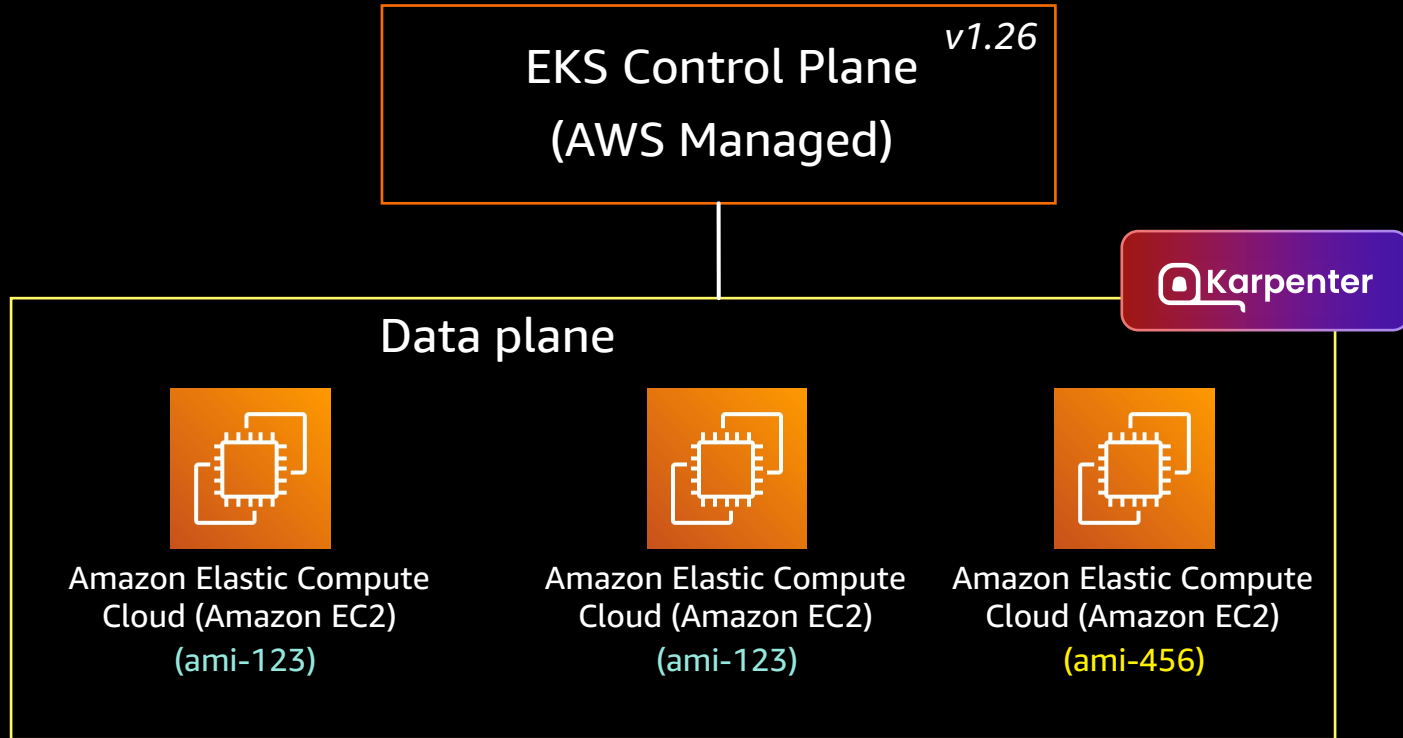
```
- id: "ami-456"
```

Patching/upgrading with Drift – Custom AMI



```
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  amiFamily: AL2
  amiSelectorTerms:
  - id: ami-123
```

Patching/upgrading with Drift – Custom AMI

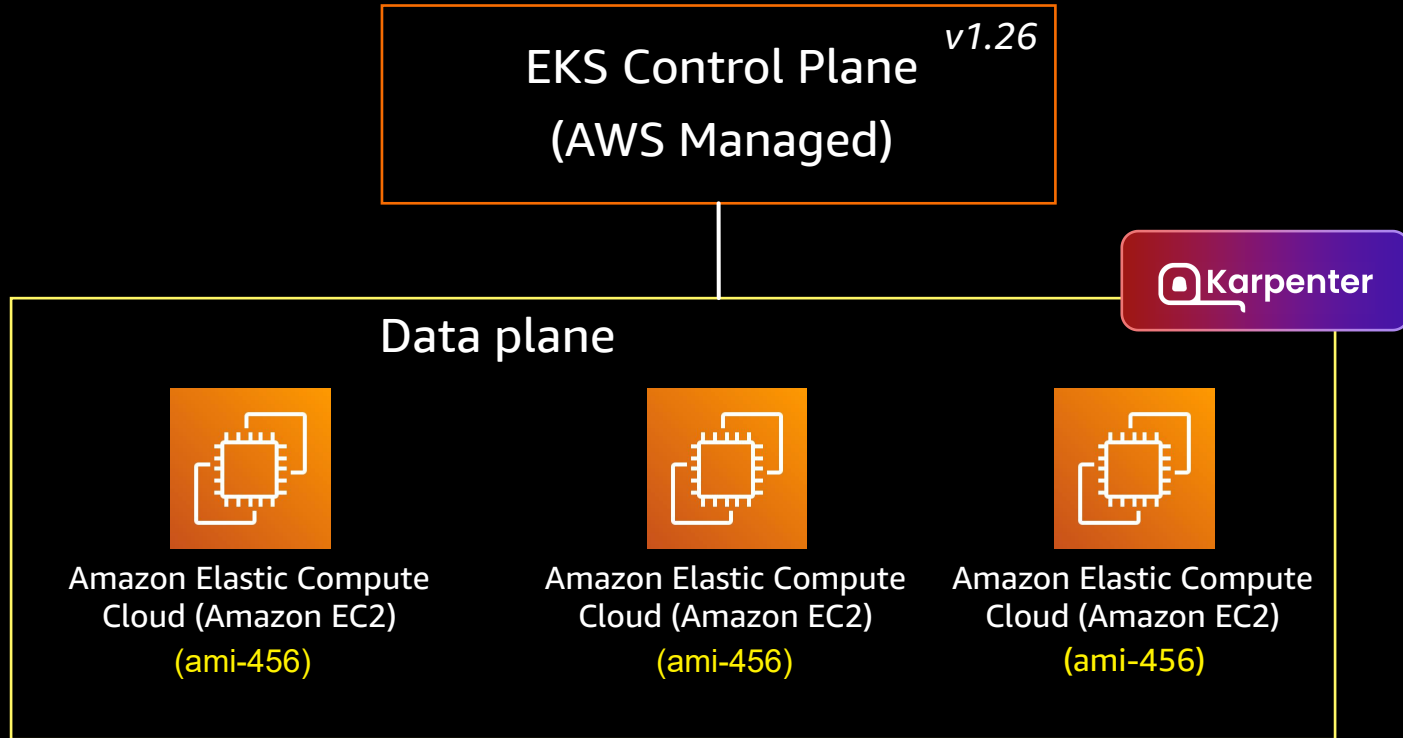


`amiSelectorTerms:`

- id: "ami-123"
- id: "ami-456"

- New custom AMI “ami-456” included
- New nodes provisioned with “ami-456”
- `amiSelectorTerms` chooses the latest AMI
- Old nodes will drift

Patching/upgrading with Drift – Custom AMI

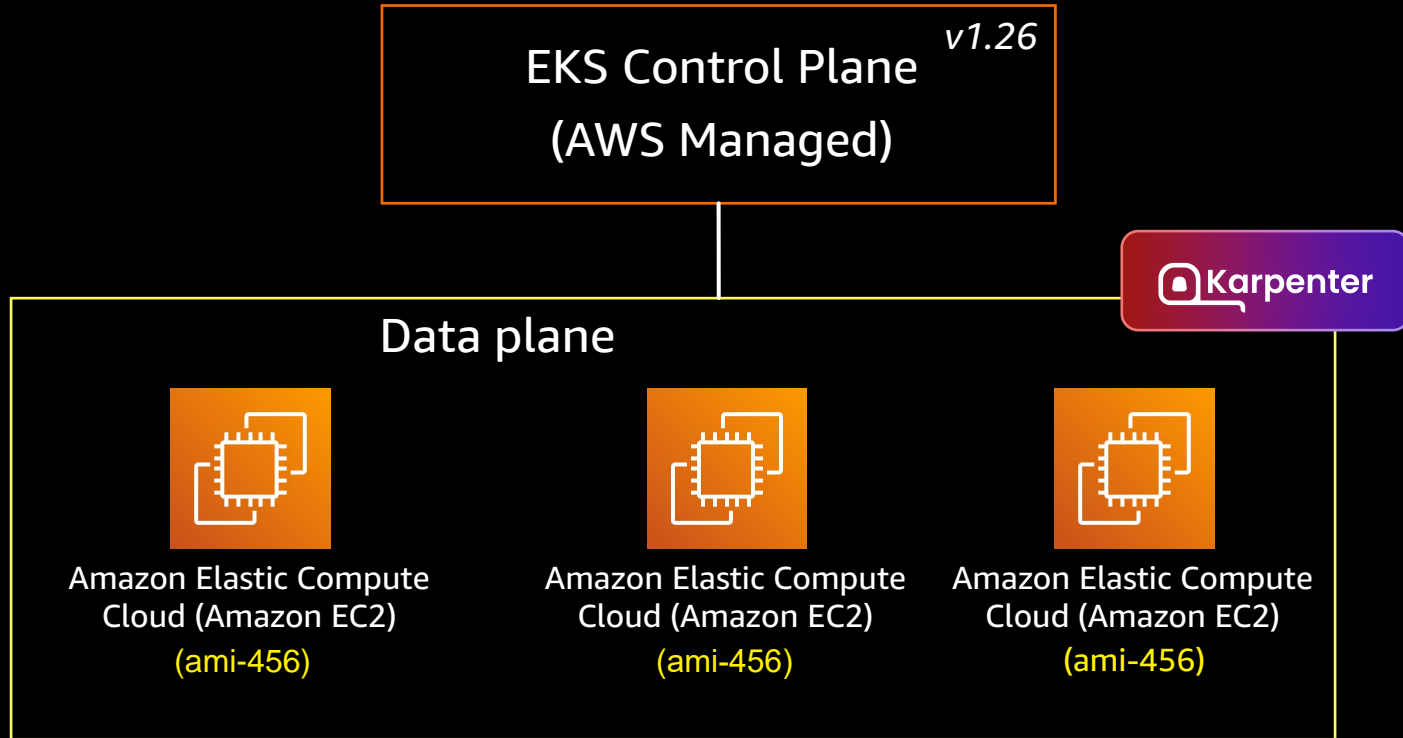


`amiSelectorTerms:`

- id: "ami-123"
- id: "ami-456"

- New custom AMI “ami-456” included
- New nodes provisioned with “ami-456”
- `amiSelectorTerms` chooses the latest AMI
- Old nodes will drift, and recycle with new ami “ami-456”
- Updating EKS control plane to newer version will NOT trigger drift

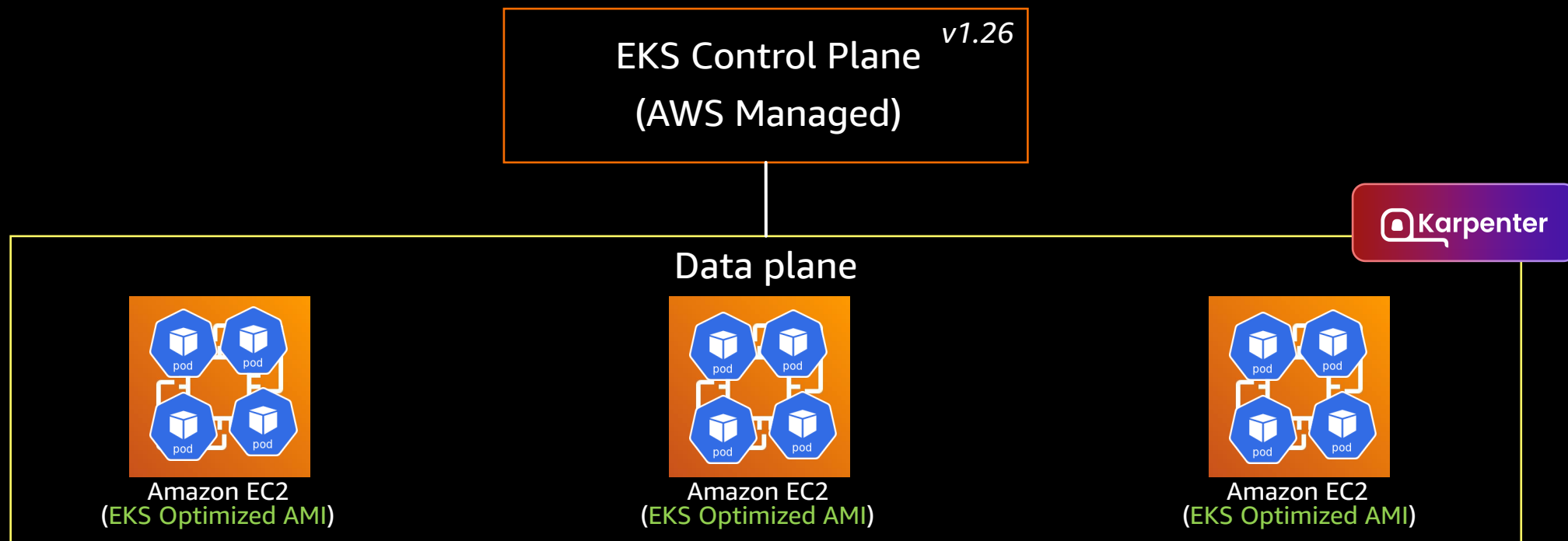
Patching/upgrading with Drift – Custom AMI



```
amiSelectorTerms:  
  - id: "ami-456"
```

- New custom AMI “ami-456” included
- New nodes provisioned with “ami-456”
- amiSelectorTerms chooses the latest AMI
- Old nodes will drift, and recycle with new ami “ami-456”
- Updating EKS control plane to newer version will NOT trigger drift

The power of Drift and Consolidation



- Used Karpenter Drift for automatic upgrade of the nodes
- Achieved ~50% optimization by consolidating thousands of nodes in prod

Disruption - Expiration

Expiration



- Karpenter will expire nodes (when you tell it to do so!) after a certain amount of time
- Useful for force AMI refresh, or recycle nodes for security concerns
- Provisions replacement node if needed
- Adds *karpenter.sh/disruption:NoSchedule* taint to the disrupted node preventing pods to schedule in it
- Evict the pods (respects PDB)
 - Evicted workload pods will get scheduled in either an existing node or a new node
- Terminate node

Expiration

- `expireAfter` - `Never` | `xh` (x=numeric, h=hours) | `yd` (y=numeric, d=days)
- Defaults:

```
spec:  
  disruption:  
    consolidationPolicy: WhenUnderutilized  
    expireAfter: 720h
```

```
apiVersion: karpenter.sh/v1beta1  
kind: NodePool  
metadata:  
  name: default  
spec:  
  disruption:  
    consolidateAfter: 30s  
    consolidationPolicy: WhenEmpty  
    expireAfter: Never
```

```
apiVersion: karpenter.sh/v1beta1  
kind: NodePool  
metadata:  
  name: default  
spec:  
  disruption:  
    consolidateAfter: 30s  
    consolidationPolicy: WhenEmpty  
    expireAfter: 720h
```

Controlling Disruption

No Disruption for Certain Pods/Nodes

To be not Disrupted, annotate Pods with:

```
kind: Pod
metadata:
  annotations:
    karpenter.sh/do-not-disrupt: "true"
```

To be not Disrupted, annotate Nodes with:

```
kind: Node
metadata:
  annotations:
    karpenter.sh/do-not-disrupt: "true"
```

Disable Disruption on a NodePool:

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  template:
    metadata:
      annotations: # will be applied to all nodes
        karpenter.sh/do-not-disrupt: "true"
```

Karpenter Respects PDB (Pod Disruption Budget)

- Karpenter respects Pod Disruption Budget
- Configure Pod Disruption Budget to control the rate of pod disruption
- Mindful of certain PDBs blocking node update

```
kind: PodDisruptionBudget
metadata:
  name: app-a
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: app-a
```

NodePool Disruption Budget

NodePool Disruption Budget is NOT Pod Disruption Budget

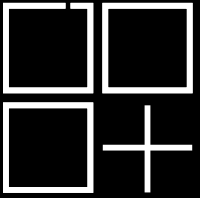
(But there is similarity in concept)

- Controls how many nodes can be disrupted at a time
- All the budgets are cumulative (AND)
- Only allow 20% of the nodes to be disrupted at one time
- Max 5 nodes disruption allowed, even if 20% comes higher
- “0” disruptions allowed the first 10 minutes of every day
- `schedule` is a Cron Job schedule

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  disruption:
    consolidationPolicy: WhenUnderutilized
    expireAfter: 720h # 30 * 24h = 720h
    budgets:
      - nodes: "20%"
      - nodes: "5"
      - nodes: "0"
        schedule: "@daily"
        duration: 10m
```

Revisiting Karpen Superpowers

Total Data Plane Implementation



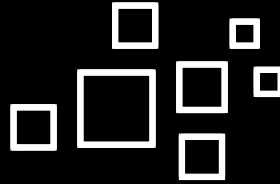
Efficient Scaling

- No Node Groups needed
- Provisions nodes with low latency
- Automatically select appropriate EC2s (Or control using NodePool)



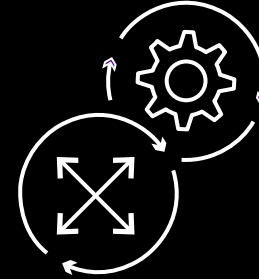
Cost optimization

- Cheapest allocation strategy (on-demand)
- Spot handling
- Consolidation



Supports diverse workloads including ML and generative AI

- Easily provision GPUs
- KEDA+Karpenter, ML workloads
- Kubeflow, ML framework integration



Helps upgrade and patching

- Expiration
- Drift
- Work with EKS-Optimized AMI and custom AMI



Kubernetes native

- Respects Kubernetes scheduling constraints
- Respects PDBs
- Powerful YAML
- CNCF project

Total Data Plane Implementation



Karpenter combines:

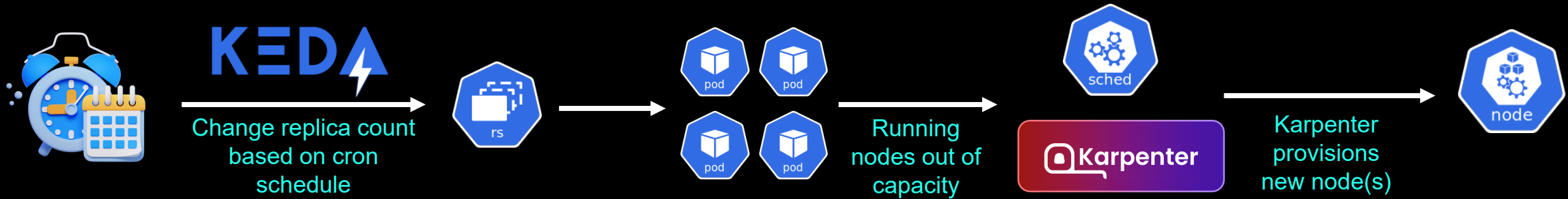
- Cluster Autoscaler
- Node groups
- Node Termination Handler
- Descheduler



- Karpenter is fast
- Karpenter is simple yet powerful
- Karpenter is cost effective
- Karpenter is secure
- Karpenter is Kubernetes native
- Karpenter is part of SIG Autoscaling (OSS)
- Karpenter is AWSome!

Scheduled and Pre-emptive Autoscaling

Scheduled Autoscaling Flow



Scheduled Autoscaling – KEDA + Karpenter



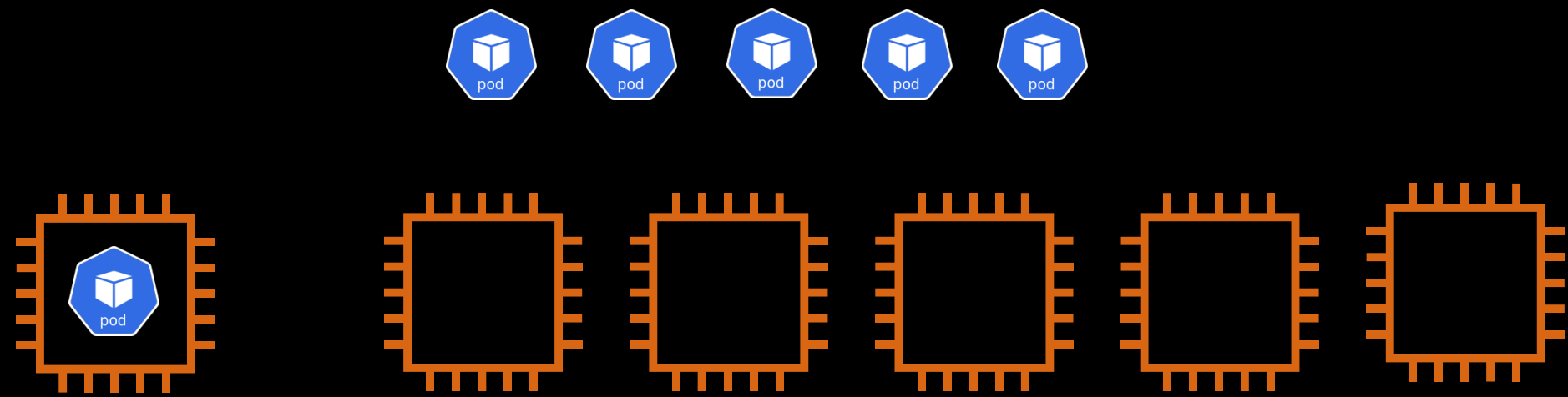
```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: cron-scaledobject
  namespace: default
spec:
  scaleTargetRef:
    name: my-deployment
  triggers:
  - type: cron
    metadata:
      timezone: Asia/Kolkata
      start: 30 * * * *
      end: 45 * * * *
      desiredReplicas: "10"
```

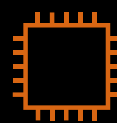
Why Scheduled Autoscaling?



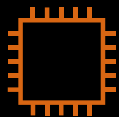
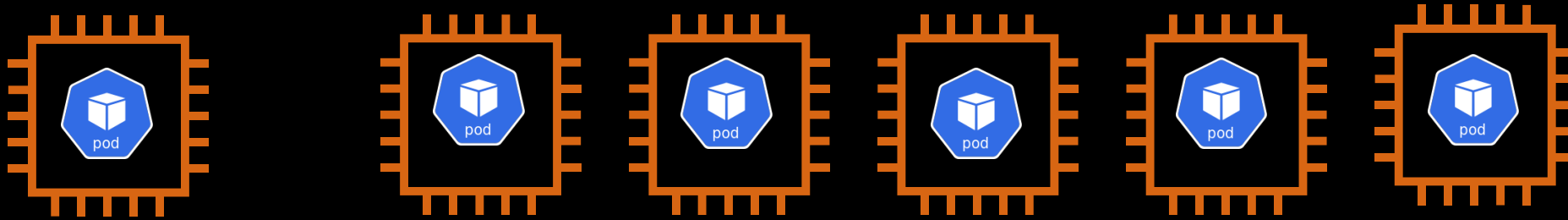
- Increased load at certain times
- Before a major product release
- Bring up instances using cluster overprovisioner pre-emptively
 - Optional - Keep provisioning more headroom EC2s using proportional autoscaler

Normal Scenario



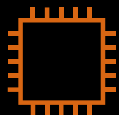
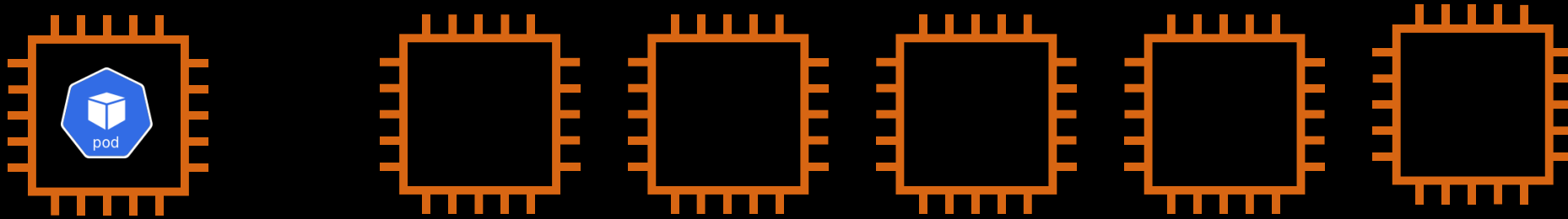
 Amazon EC2 Instance

Normal Scenario



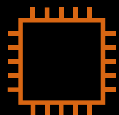
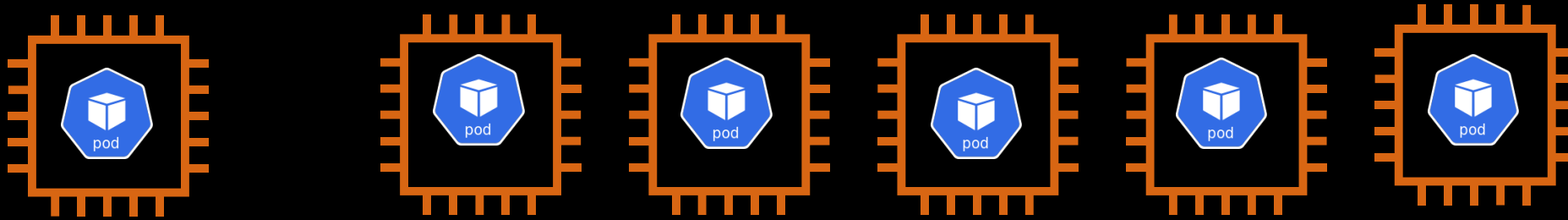
Amazon EC2 Instance

Pre-Warmed Scenario



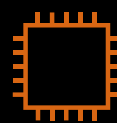
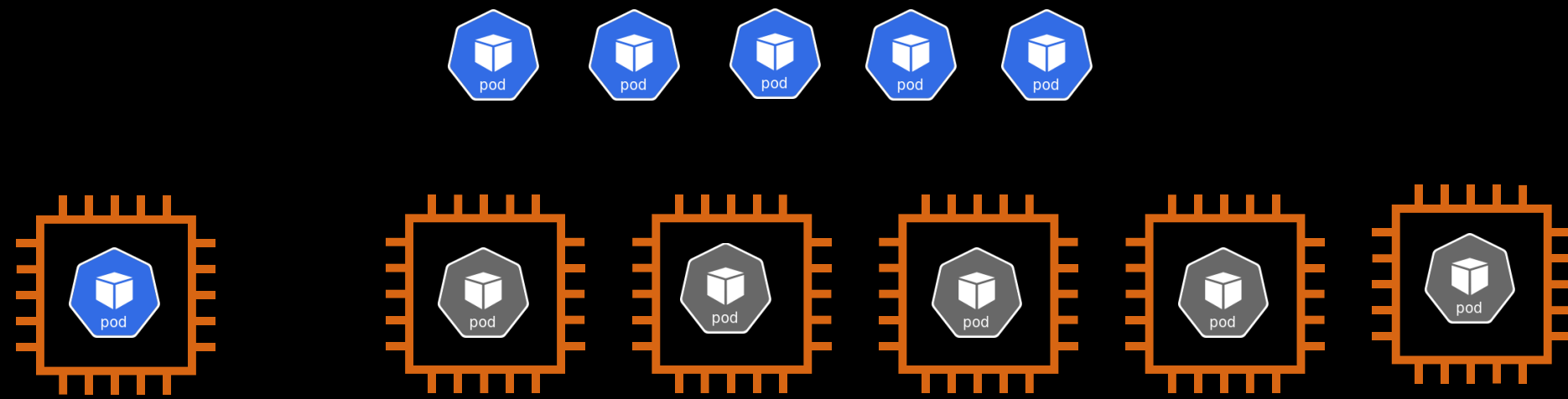
Amazon EC2 Instance

Pre-Warmed Scenario



Amazon EC2 Instance

Pre-Warmed Scenario – Cluster Overprovisioner

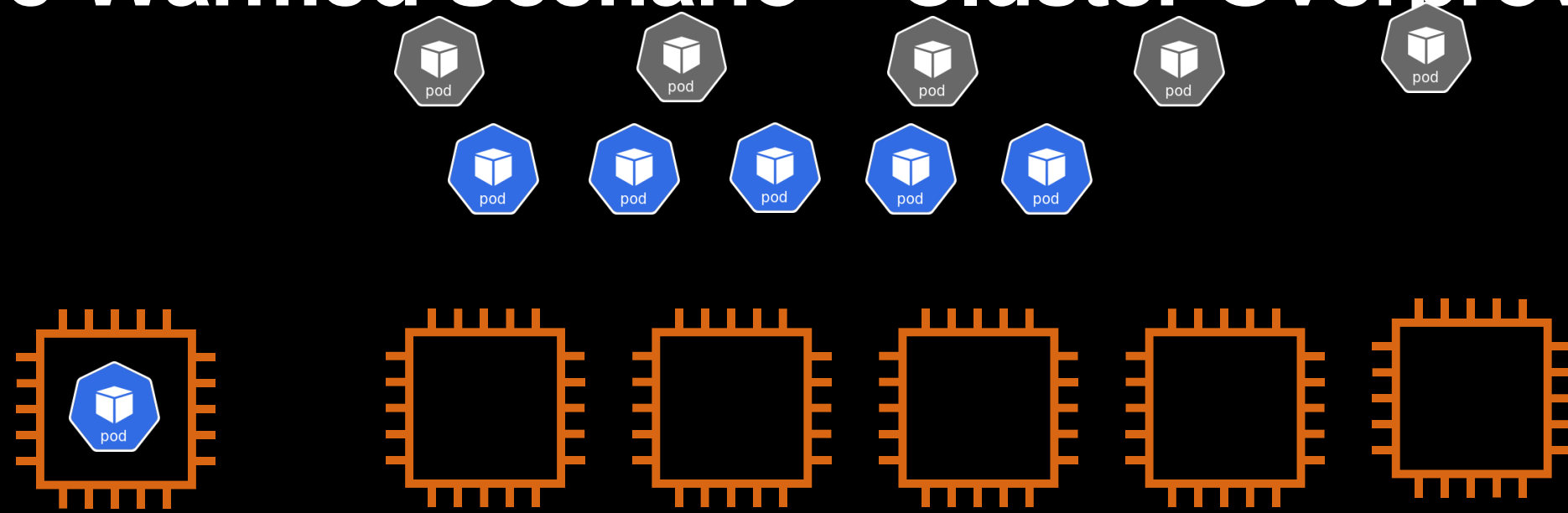


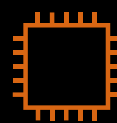
Amazon EC2 Instance




Pause/Dummy Pods (Gets evicted as soon as actual application pod is pending)

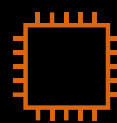
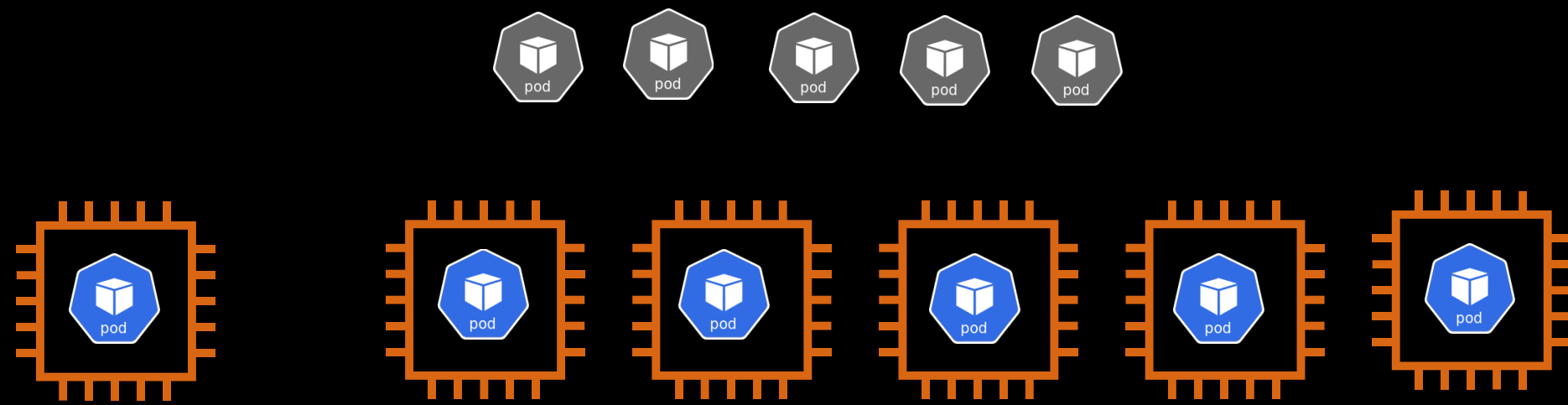
Pre-Warmed Scenario – Cluster Overprovisioner



 Amazon EC2 Instance

 Pause/Dummy Pods (Gets evicted as soon as actual application pod is pending)

Pre-Warmed Scenario – Cluster Overprovisioner

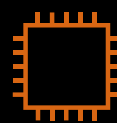
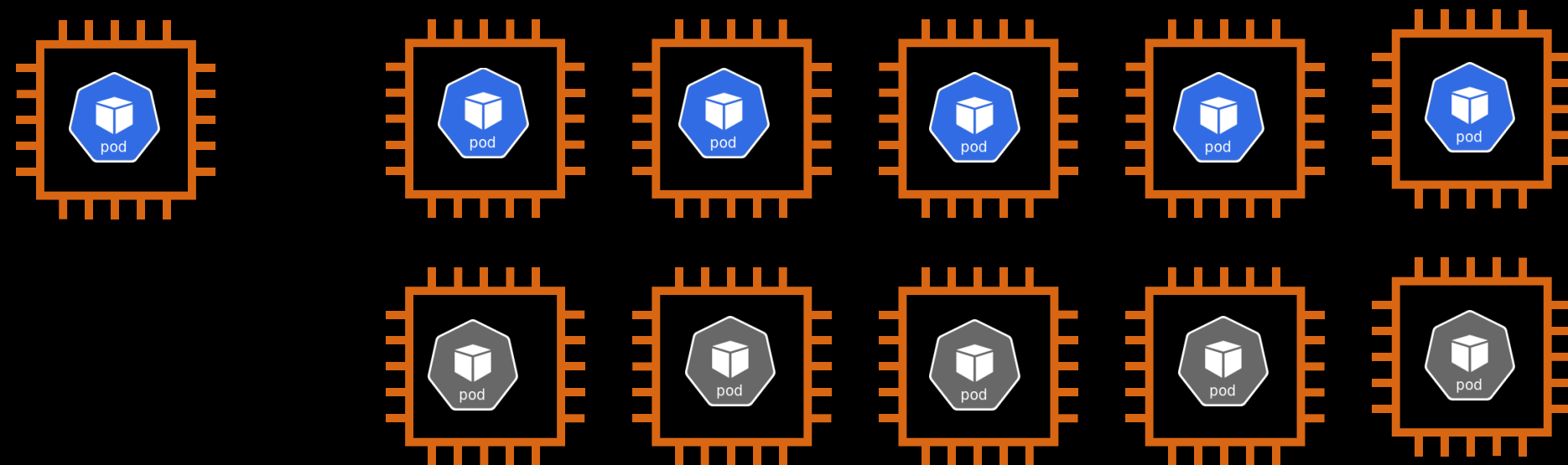


Amazon EC2 Instance



Pause/Dummy Pods (Gets evicted as soon as actual application pod is pending)

Pre-Warmed Scenario – Cluster Overprovisioner

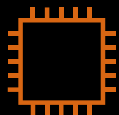
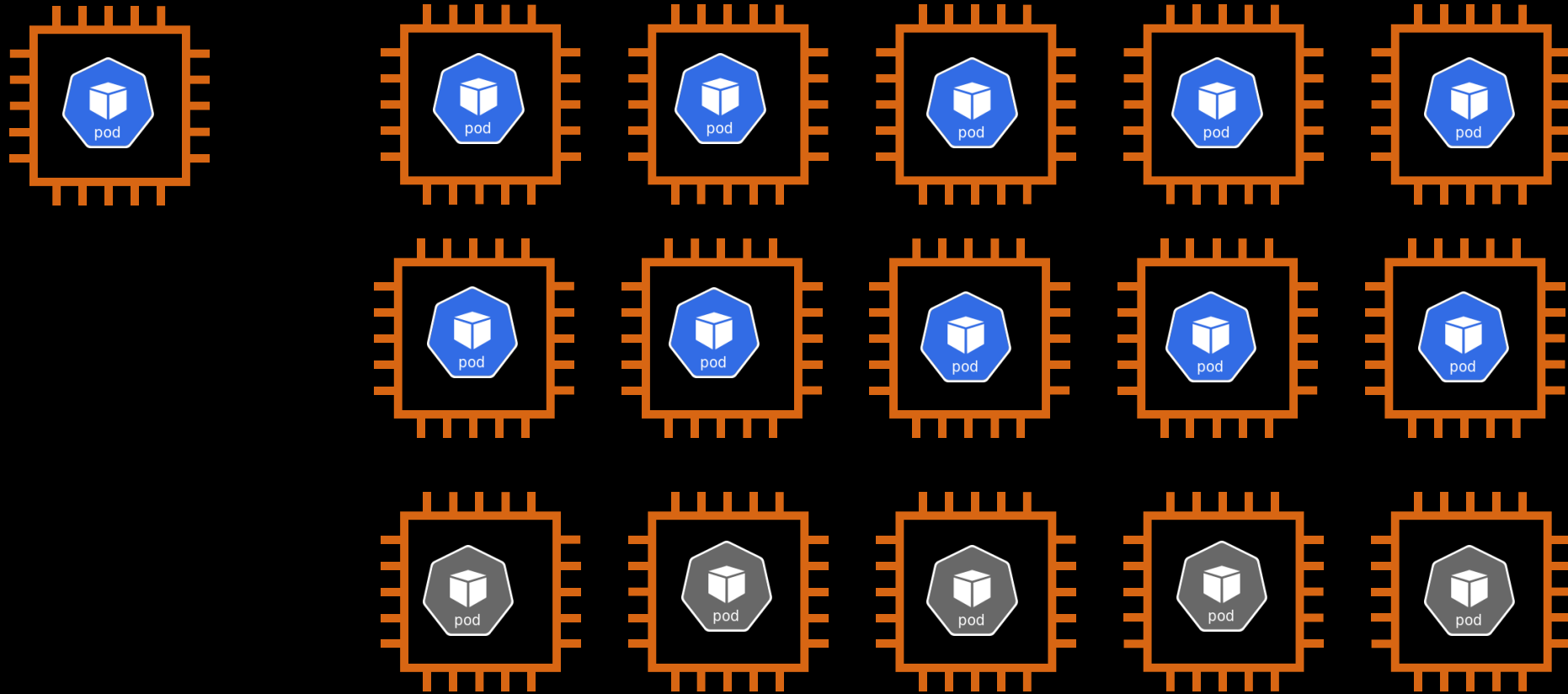


Amazon EC2 Instance



Pause/Dummy Pods (Gets evicted as soon as actual application pod is pending)

Pre-Warmed Scenario – Cluster Overprovisioner with Proportional Autoscaler

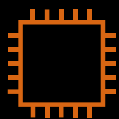
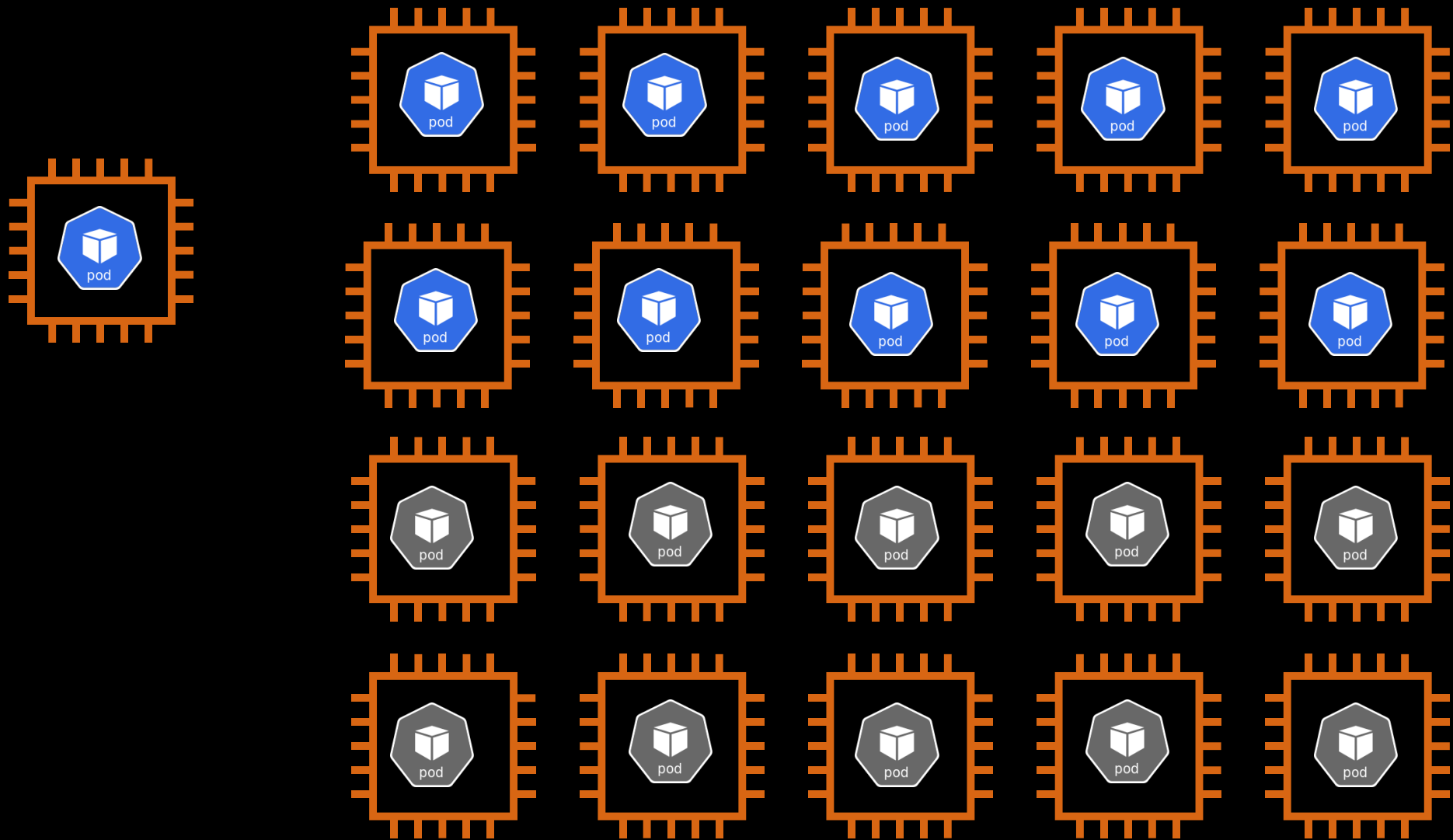


Amazon EC2 Instance



Pause/Dummy Pods (Gets evicted as soon as actual application pod is pending)

Pre-Warmed Scenario – Cluster Overprovisioner with Proportional Autoscaler



Amazon EC2 Instance



Pause/Dummy Pods (Gets evicted as soon as actual application pod is pending)

Controlling Pod Density

Onboarding Karpenter

Onboarding Karpenter

- Install Karpenter Helm chart from AWS public ECR
- Do not run Karpenter on a node that is managed by Karpenter
- Karpenter controller on Amazon EKS Fargate or on a worker node (2 node nodegroup)
- Migrating from CA
 - Step-by-step guide: [Migrating from Cluster Autoscaler](#)
 - Reuse custom AMI pipeline – update EC2NodeClass instead of ASG Launch Template

Migrating from CA to Karpenter

Karpenter per Namespace – Possible?

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: nodepool-a
spec:
  disruption:
    consolidationPolicy: WhenUnderutilized
    expireAfter: 720h
  limits:
    cpu: 100
  template:
    metadata:
      labels:
        app-team: payments
    spec:
```

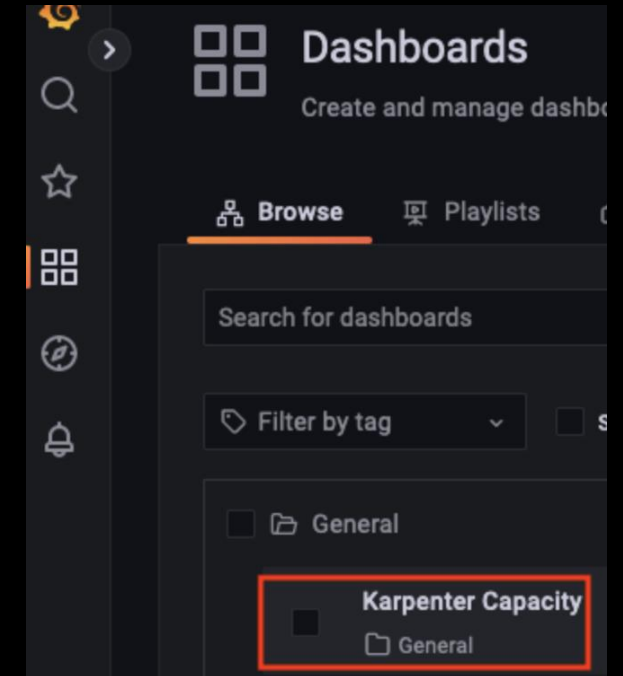
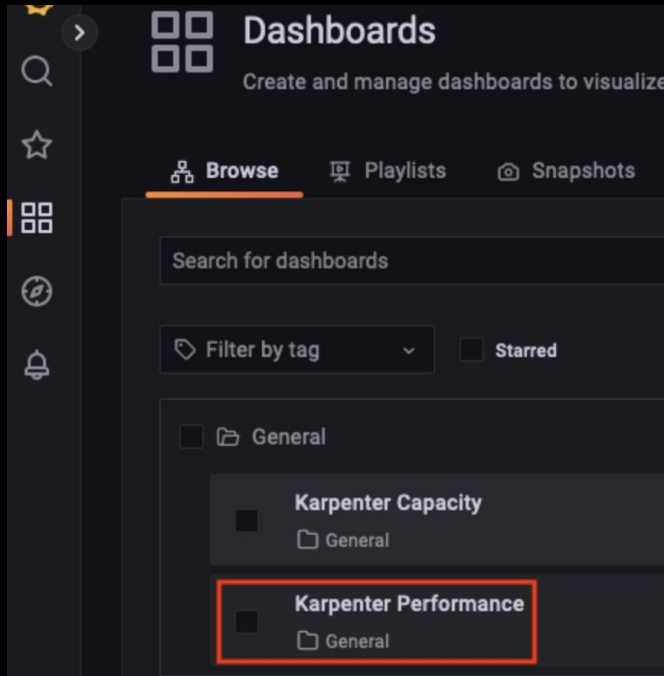
```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: nodepool-b
spec:
  disruption:
    consolidationPolicy: WhenUnderutilized
    expireAfter: 720h
  limits:
    cpu: 50
  template:
    metadata:
      labels:
        app-team: reporting
    spec:
```

Karpenter Observability

Karpenter observability – Logs/metrics

- Inspect Karpenter logs similar to pod logs
- Karpenter emits Prometheus metrics
 - Install Prometheus and Grafana for dashboards
 - Monitor pod startup latency, node utilization, and so on

```
kubectl logs deployment/karpenter -c controller -n karpenter
```



Karpenter Best Practices

Karpenter Best Practices



- Run Karpenter controller on EKS Fargate or a 2 node nodegroup not managed by Karpenter
- Exclude instance types that do not fit your workload
- Use Karpenter for workloads which are:
 - Changing capacity needs
 - Need to work with Kubernetes scheduling constraints

```
- key: node.kubernetes.io/instance-type
  operator: NotIn
  values:
    'm6g.16xlarge'
    'm6gd.16xlarge'
    'r6g.16xlarge'
    'r6gd.16xlarge'
    'c6g.16xlarge'
```

Karpenter Best Practices



- Create multiple NodePools and EC2NodeClasses when:
 - Different teams have different node (Type, AMI) requirements in same cluster
- Create mutually exclusive NodePools, unless using weight
- Use drift and/or expiration to recycle nodes to latest (and secure) AMI
- Avoid overly constraining instance types for Spot

Pod Best Practices Related To Karpenter



- Utilize limits in NodePool for cost control
 - Export logs to CloudWatch and create alarm on this if threshold breached
- Use `karpenter.sh/do-not-disrupt: "true"` annotation to prevent disruption of an applicable pod/node
- Use resource requests for pods
 - Use tools to rightsize requests (Kubecost, Stormforge, Goldilocks, CloudWatch Container Insights)
 - Configure default requests using limit ranges for namespace
 - Configure requests=limit for non-CPU resources



Raj Saha

cloudwithraj.com



Cloud With Raj



cloudwithraj

Instructor Bio:

Pr. Solutions Architect @aws

Bestselling author

“Top Systems Design” award by LinkedIn

Public speaker at major conferences

Author of multiple official AWS blogs

YouTuber with 100K+ subscribers

Previously - Distinguished Cloud Architect @Verizon

Opinions are my own