DEVCONF.cz

beyond CAS

why the world needs another k8s cluster autoscaler

josephine pfeiffer

what is autoscaling?


nodes together strong.

why does it matter?


UNDERPROVISIONED


OVERPROVISIONED

RIGHT-SIZE :)

what types of autoscaling are there?*

*in k8s

DEV CONF .CZ


hpa


KEDA


vpa


K


hpa


KEDA

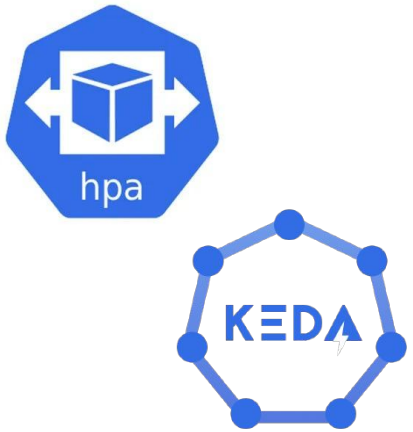

vpa

hpa

KEDA


vpa

K


hpa
KEDA

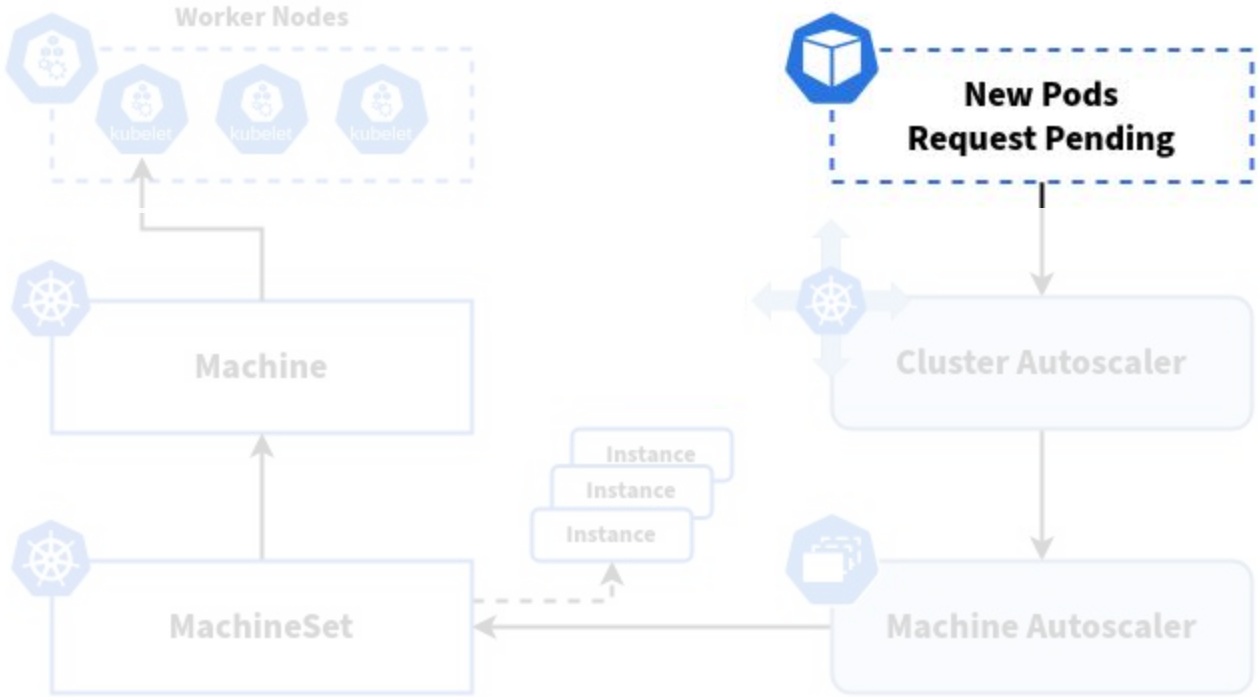
hpa

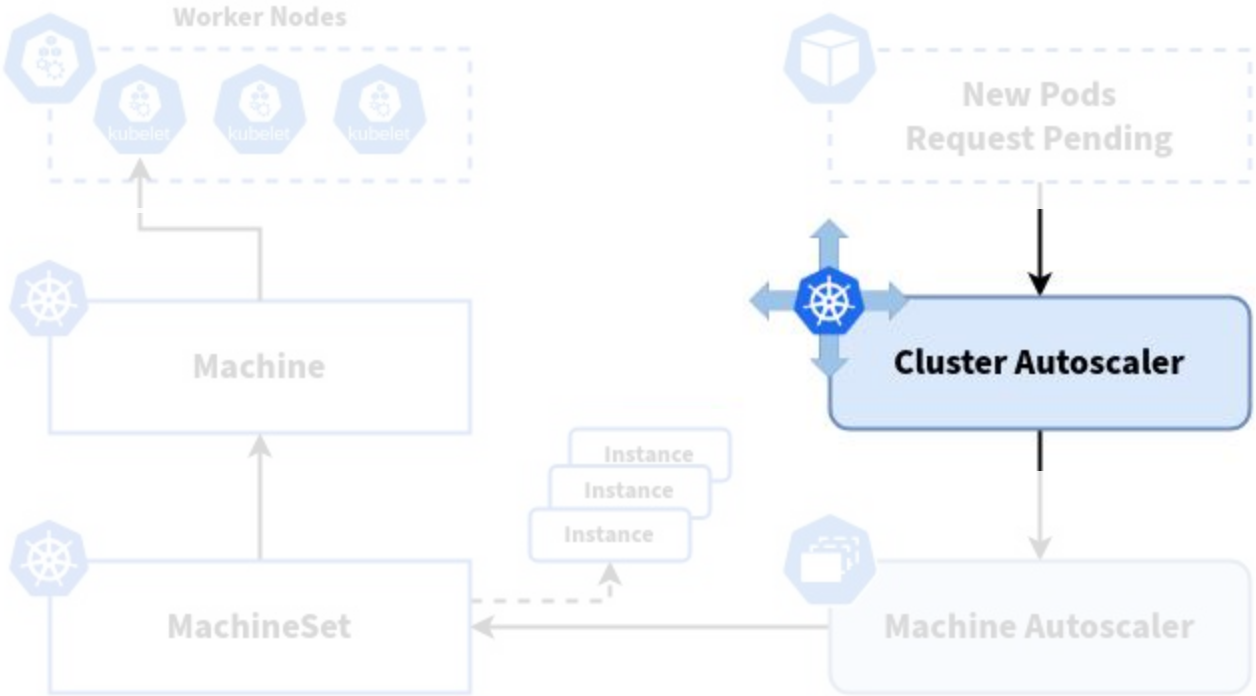
KEDA

vpa

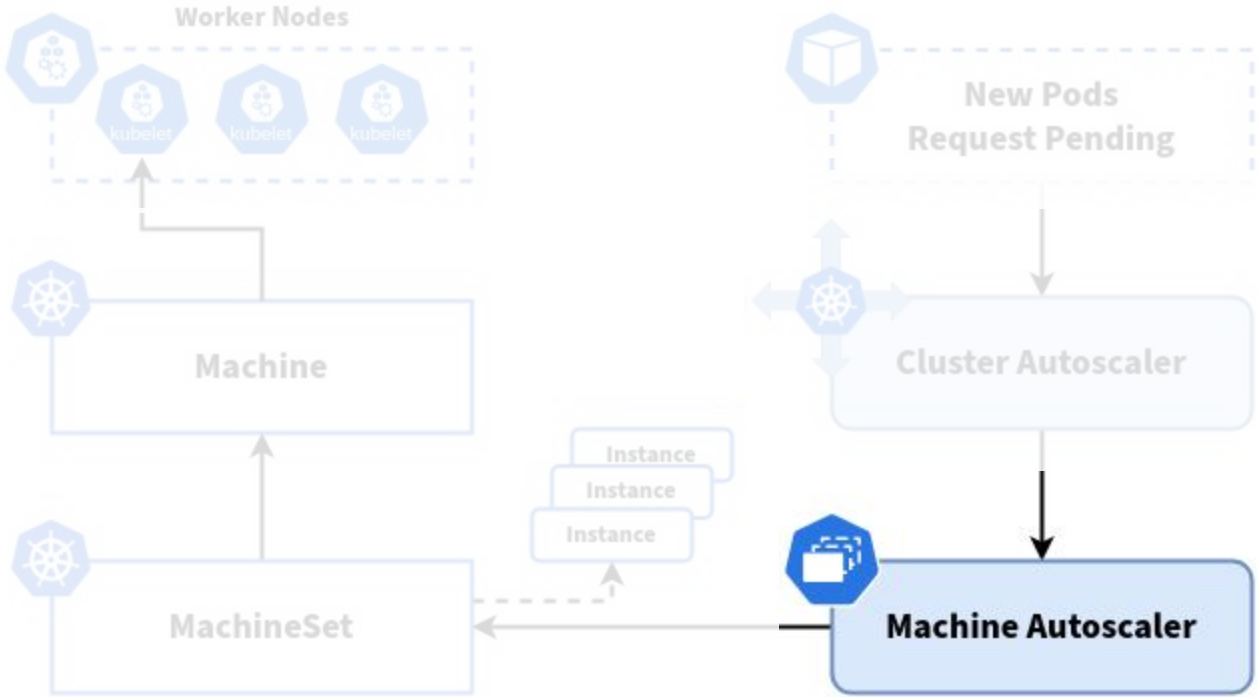

K

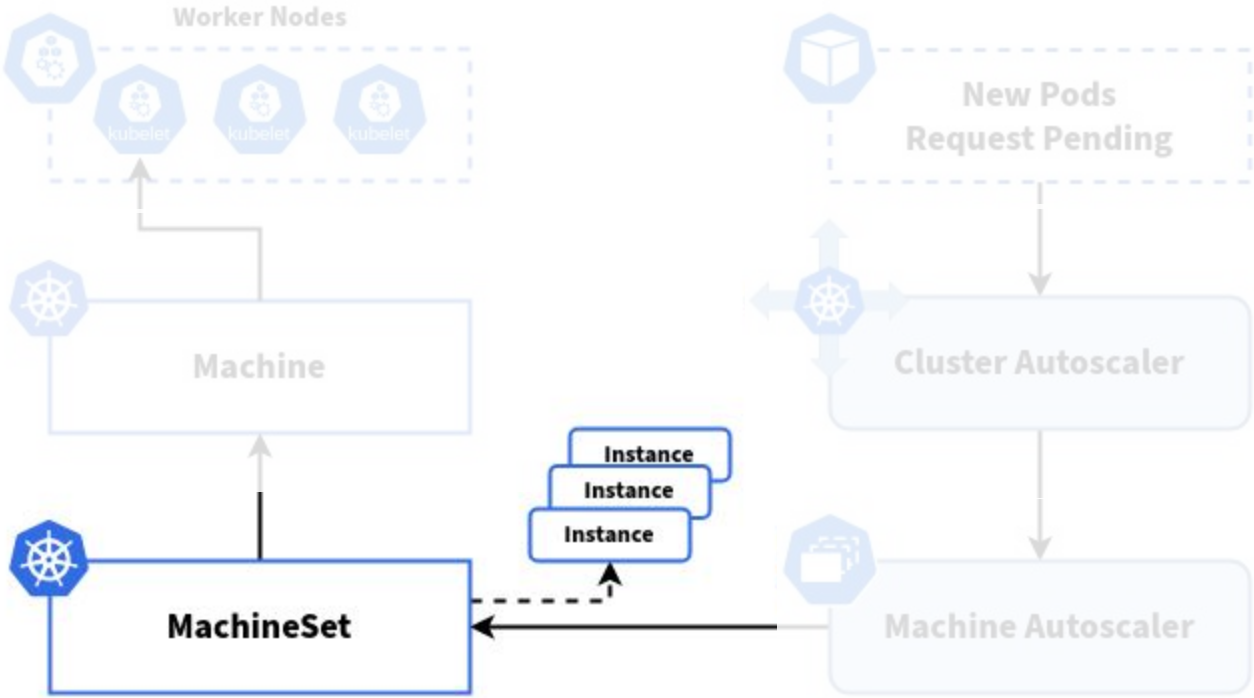
about CAS

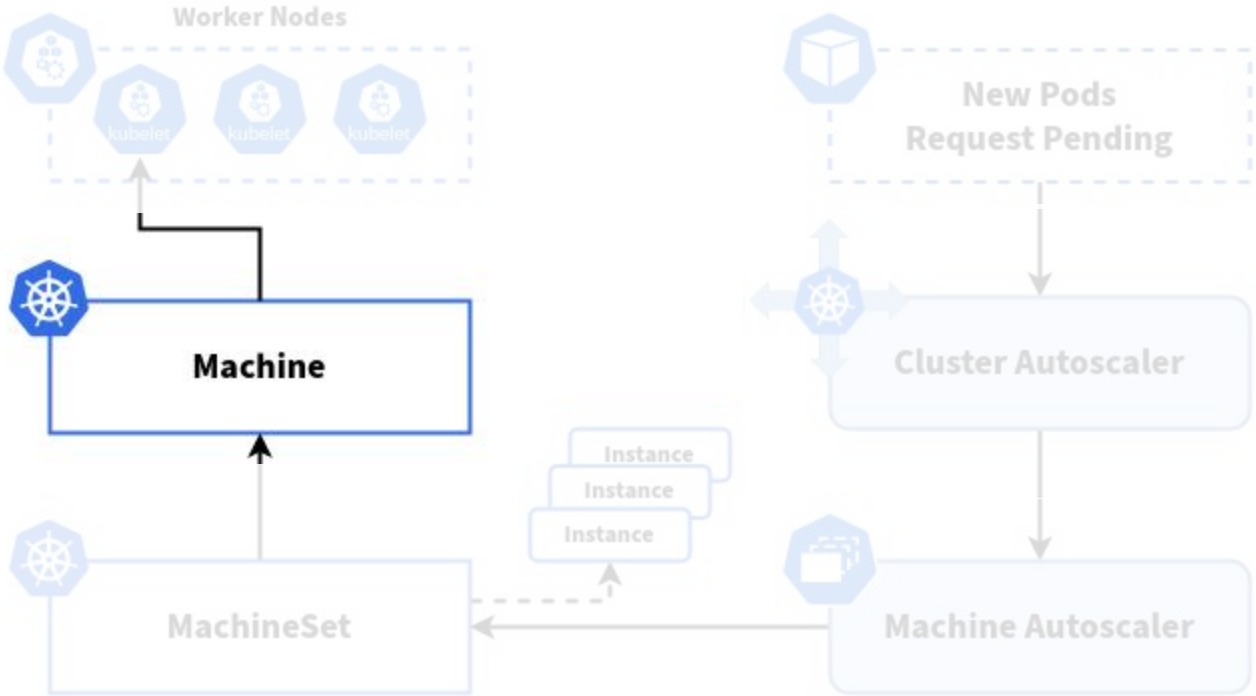

Worker Nodes
kubelet
kubelet
kubelet
New Pods
Request Pending
Machine
Cluster Autoscaler
Instance
Instance
Instance
MachineSet
Machine Autoscaler

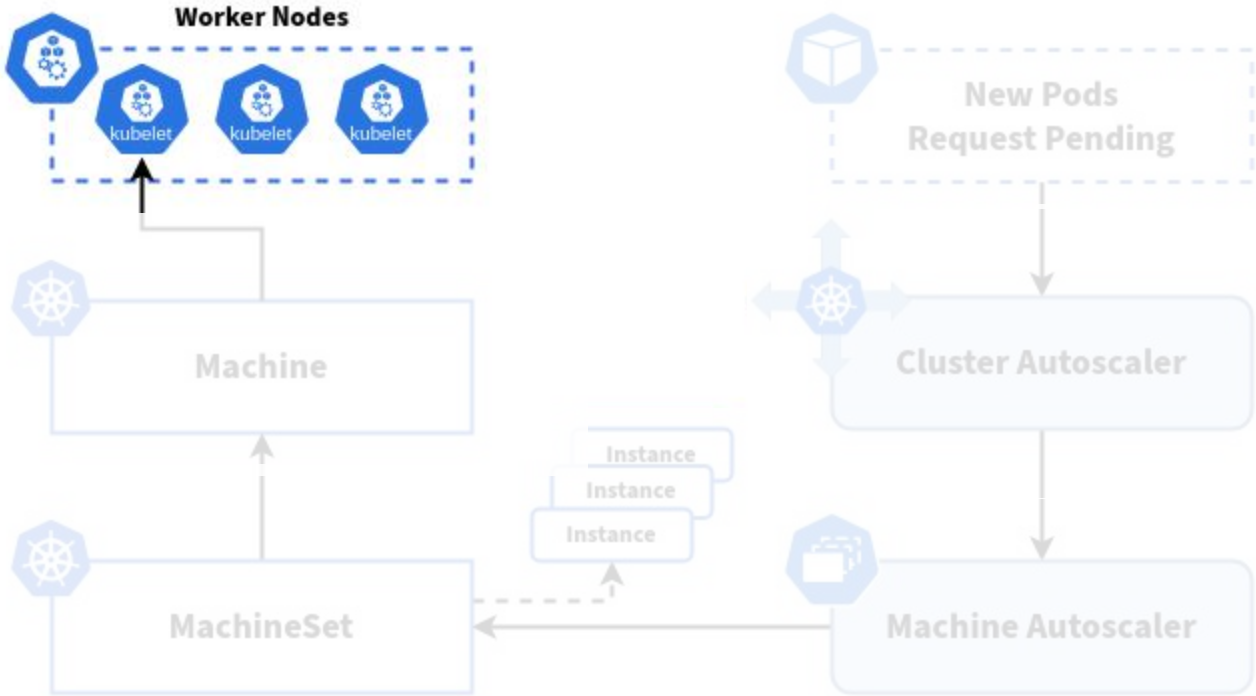

Worker Nodes
kubelet
kubelet
kubelet
New Pods
Request Pending
Machine
Cluster Autoscaler
Instance
Instance
Instance
MachineSet
Machine Autoscaler

```
// CAS polls every 10 seconds
for {
    time.Sleep(10 * time.Second)
    pods := listPendingPods() // API call every time
    processUnschedulable(pods)
}
```


Worker Nodes
kubelet
kubelet
kubelet
New Pods
Request Pending
Machine
Cluster Autoscaler
Instance
Instance
Instance
MachineSet
Machine Autoscaler

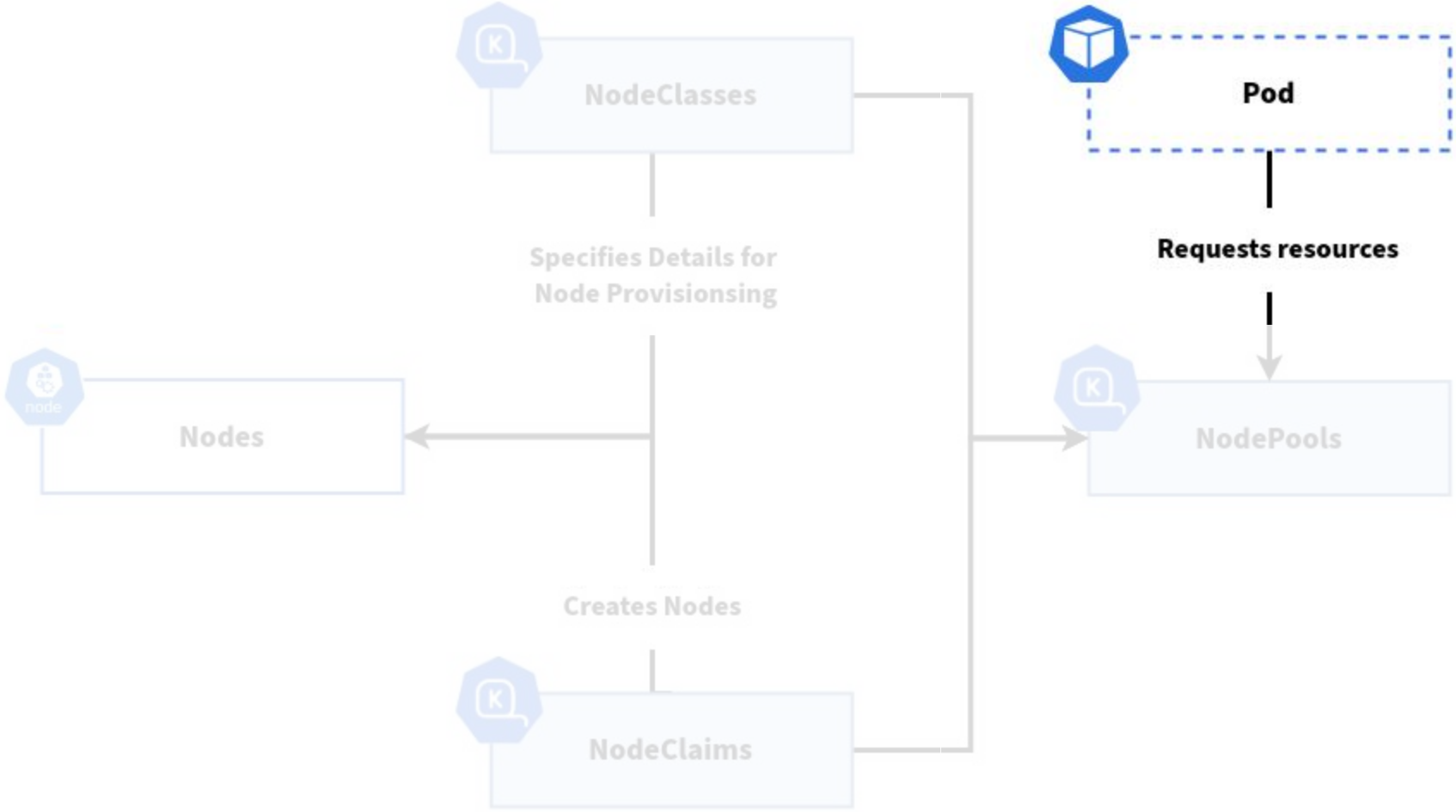

Worker Nodes
kubelet
kubelet
kubelet
New Pods
Request Pending
Machine
Cluster Autoscaler
Instance
Instance
Instance
MachineSet
Machine Autoscaler


Worker Nodes
kubelet
kubelet
kubelet
New Pods
Request Pending
Machine
Cluster Autoscaler
Instance
Instance
Instance
MachineSet
Machine Autoscaler

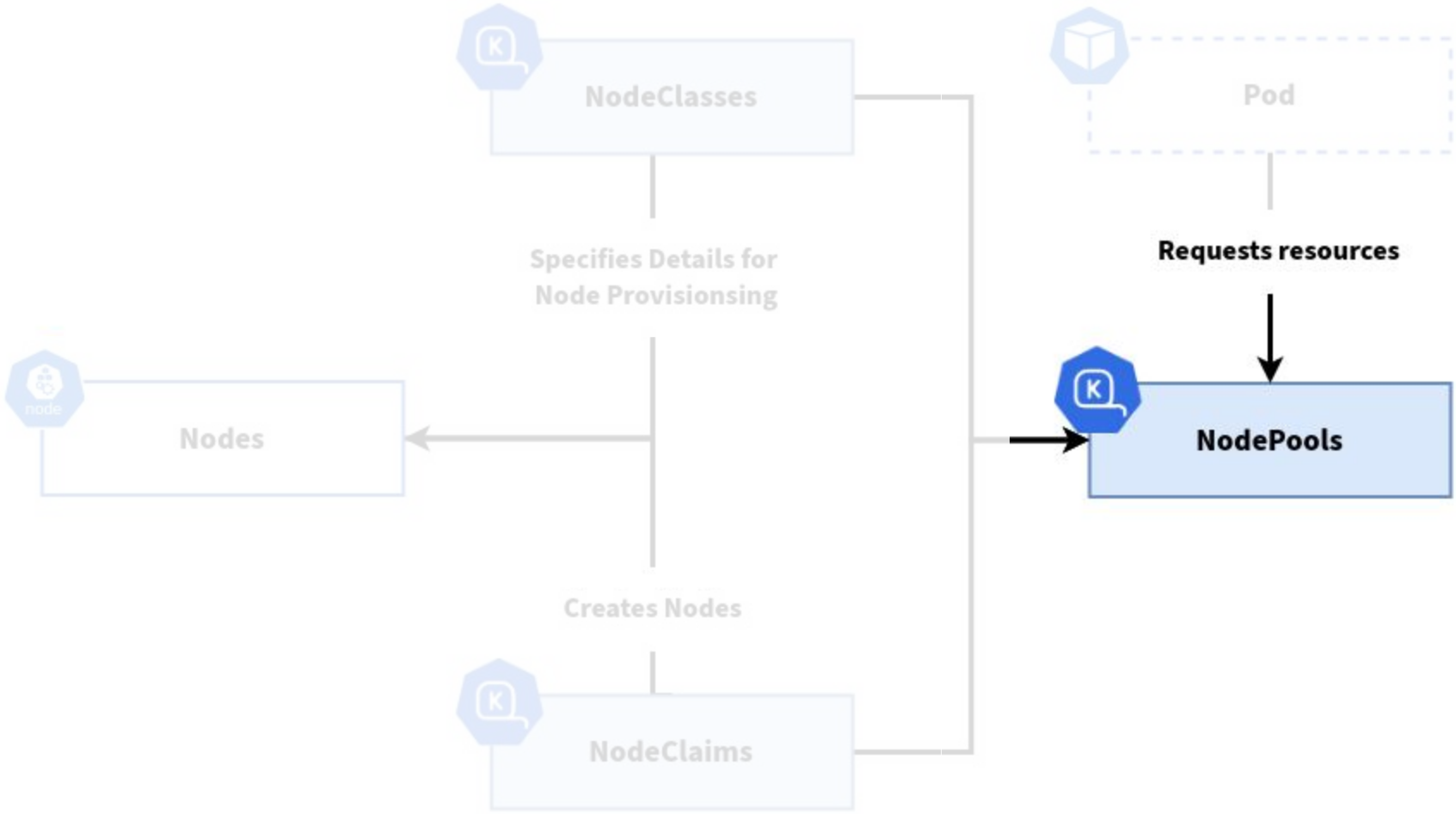

Worker Nodes
kubelet
kubelet
kubelet
New Pods
Request Pending
Machine
Cluster Autoscaler
Instance
Instance
Instance
MachineSet
Machine Autoscaler

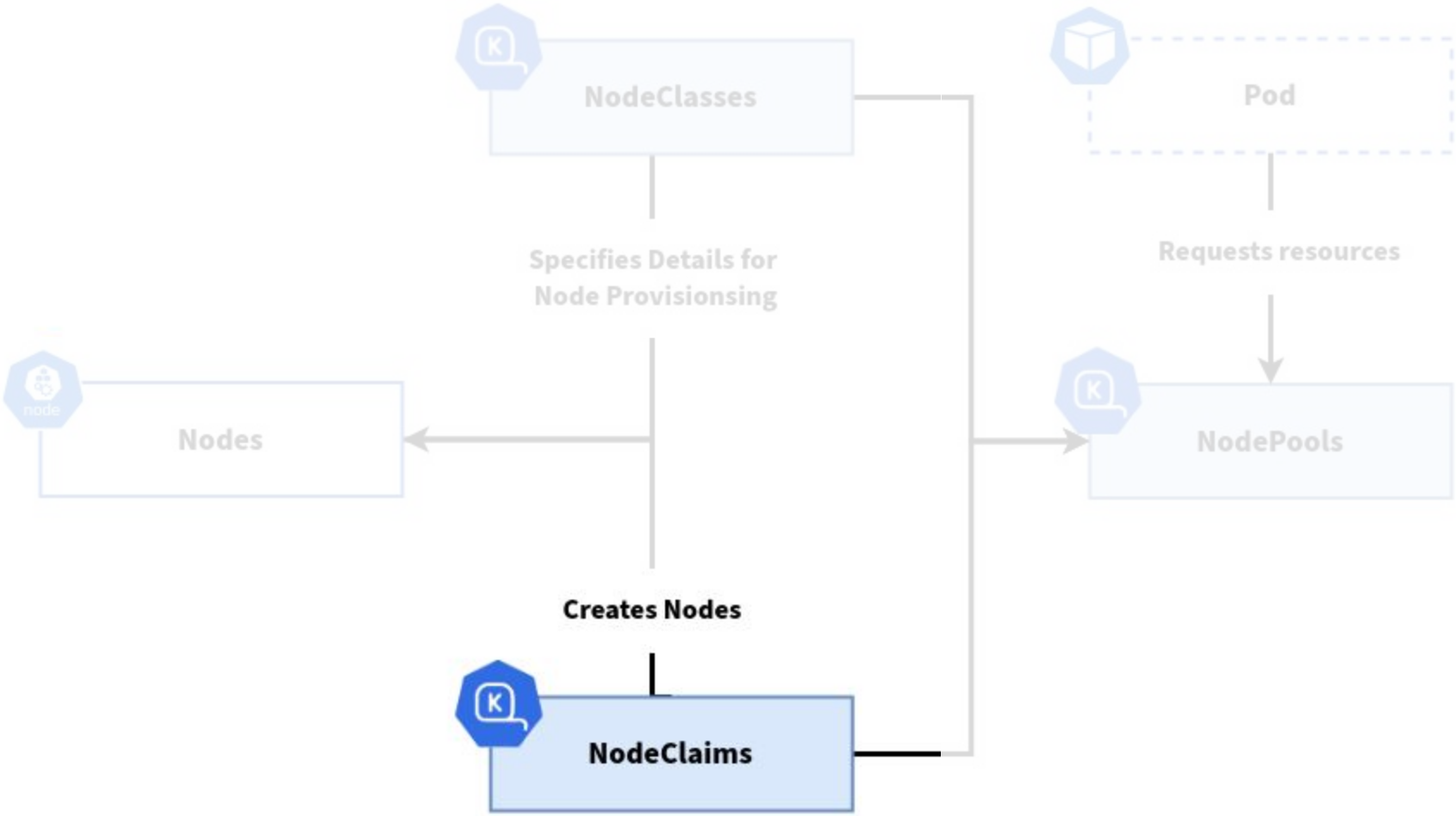
CAS pain points

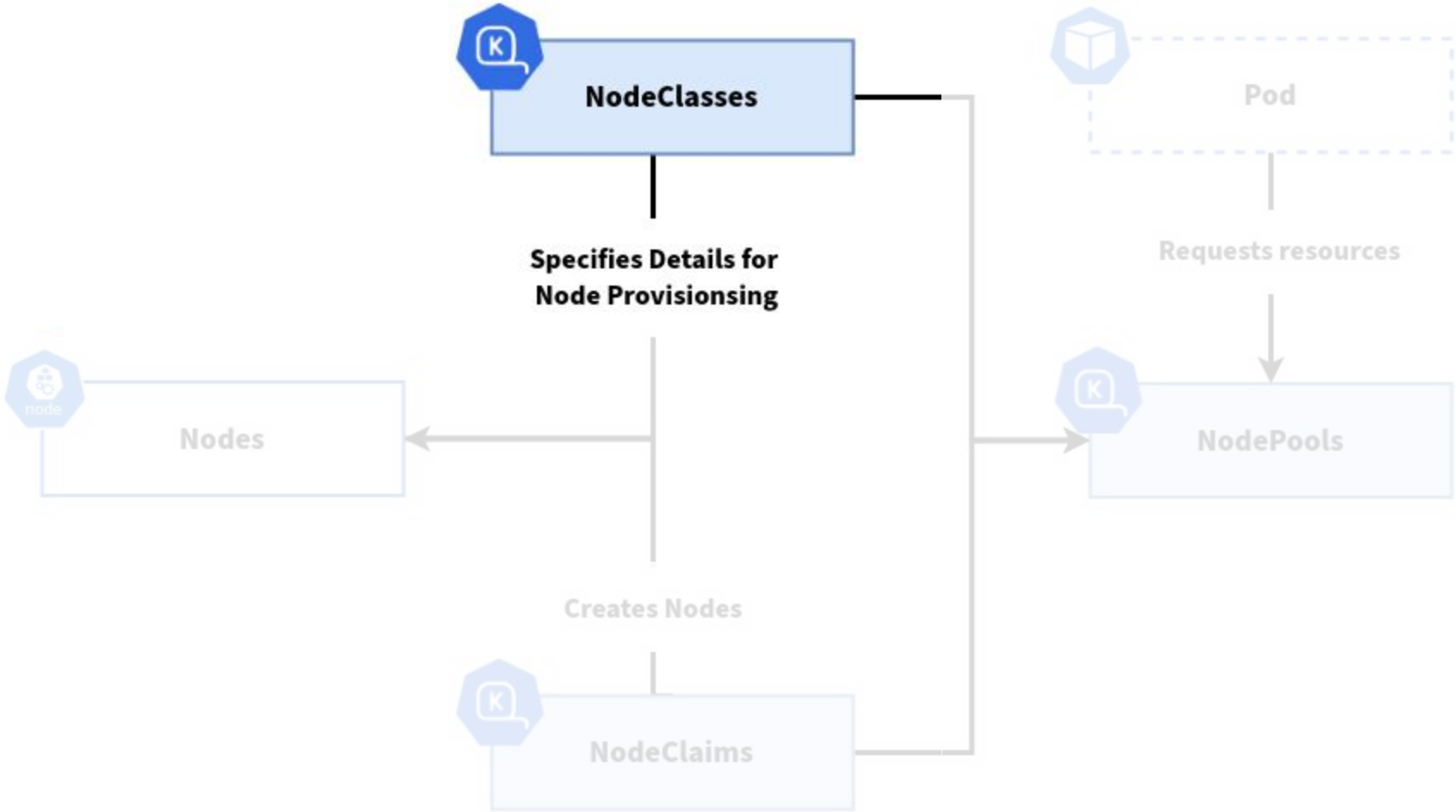
karpenter

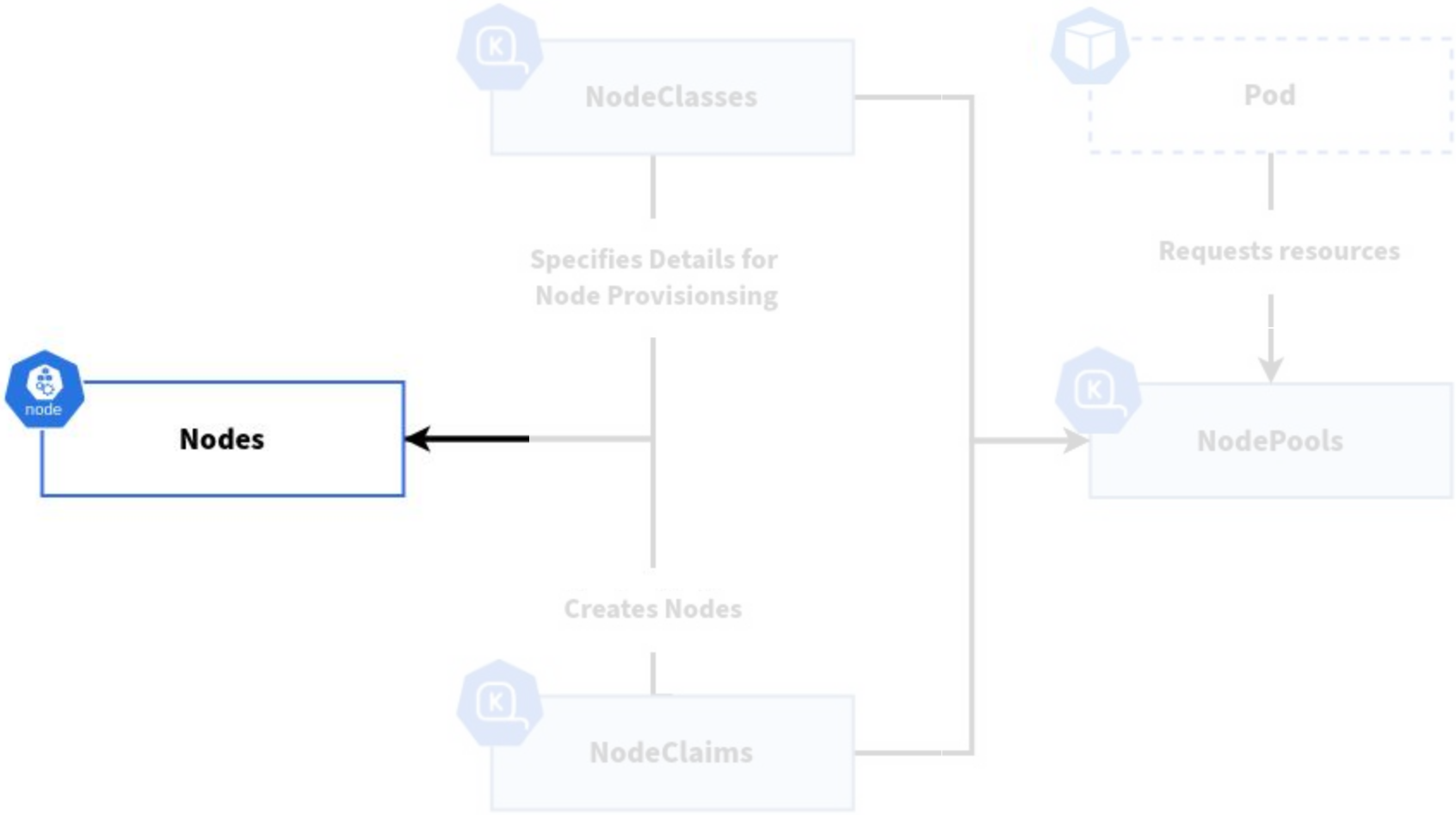

NodeClasses
Pod
Specifies Details for
Node Provisionsing
Requests resources
Nodes
NodePools
Creates Nodes
NodeClaims

```
// Karpenter watches pod events
podInformer.Watch(func(event) {
    if event.Type == "PodUnschedulable" {
        triggerReconcile() // Instant reaction!
    }
})
```


NodeClasses
Pod
Specifies Details for
Node Provisionsing
Requests resources
Nodes
NodePools
Creates Nodes
NodeClaims

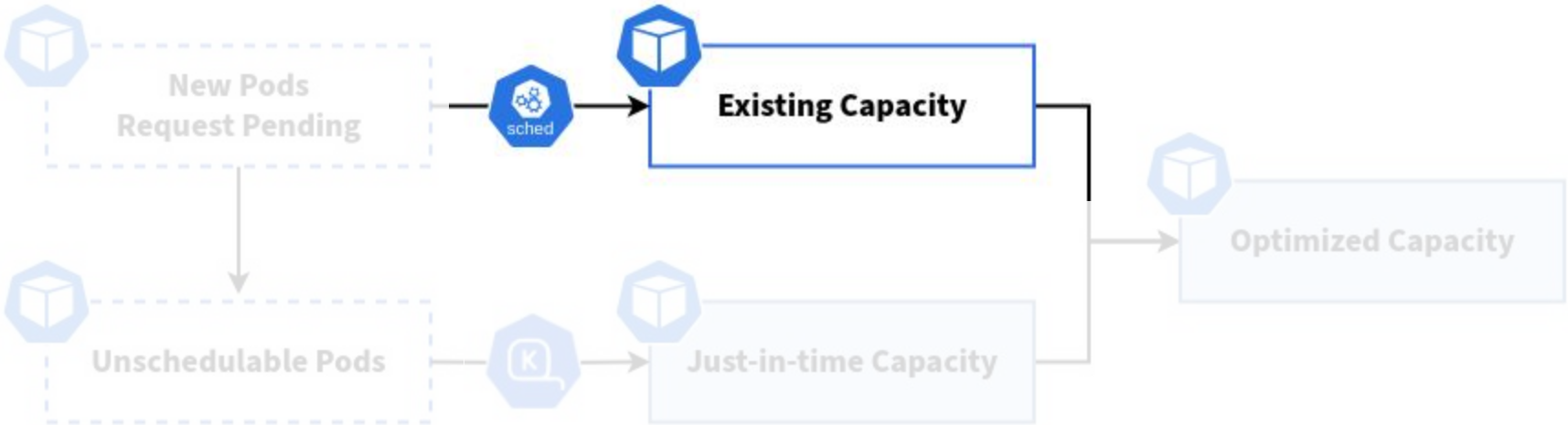

NodeClasses
Pod
Specifies Details for
Node Provisionsing
Requests resources
Nodes
NodePools
Creates Nodes
NodeClaims

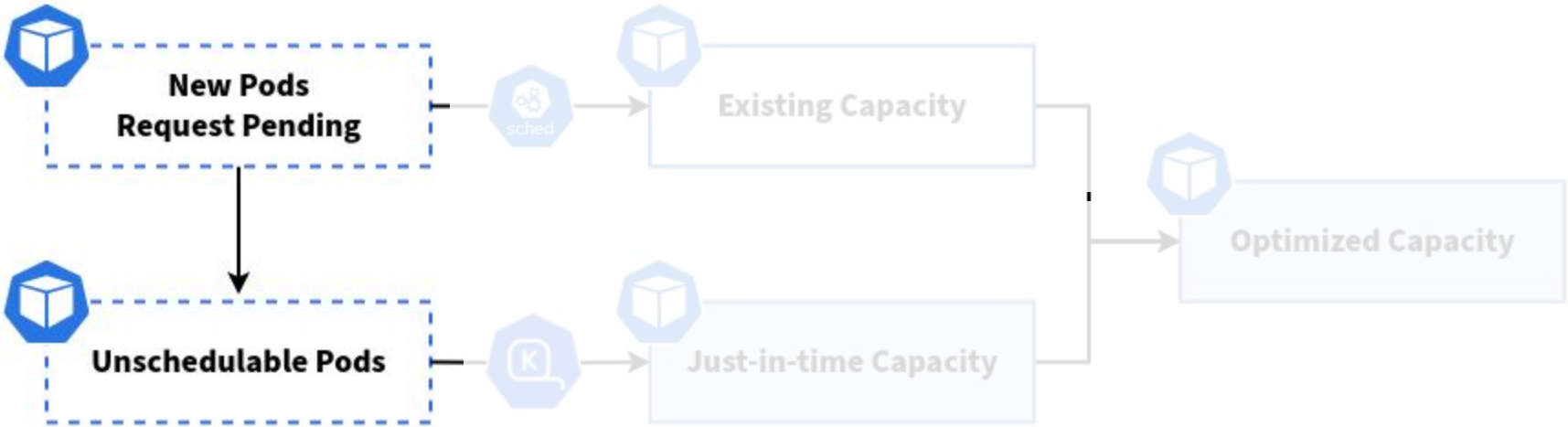

NodeClasses
Pod
Specifies Details for
Node Provisionsing
Requests resources
Nodes
NodePools
Creates Nodes
NodeClaims

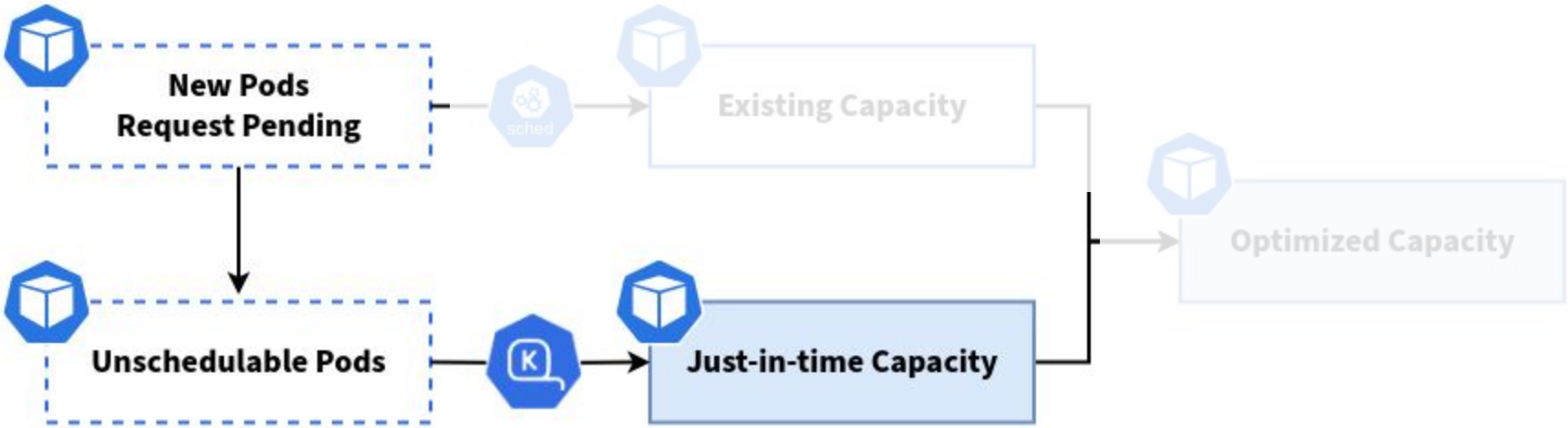

NodeClasses
Pod
Specifies Details for
Node Provisionsing
Requests resources
node
Nodes
NodePools
Creates Nodes
NodeClaims

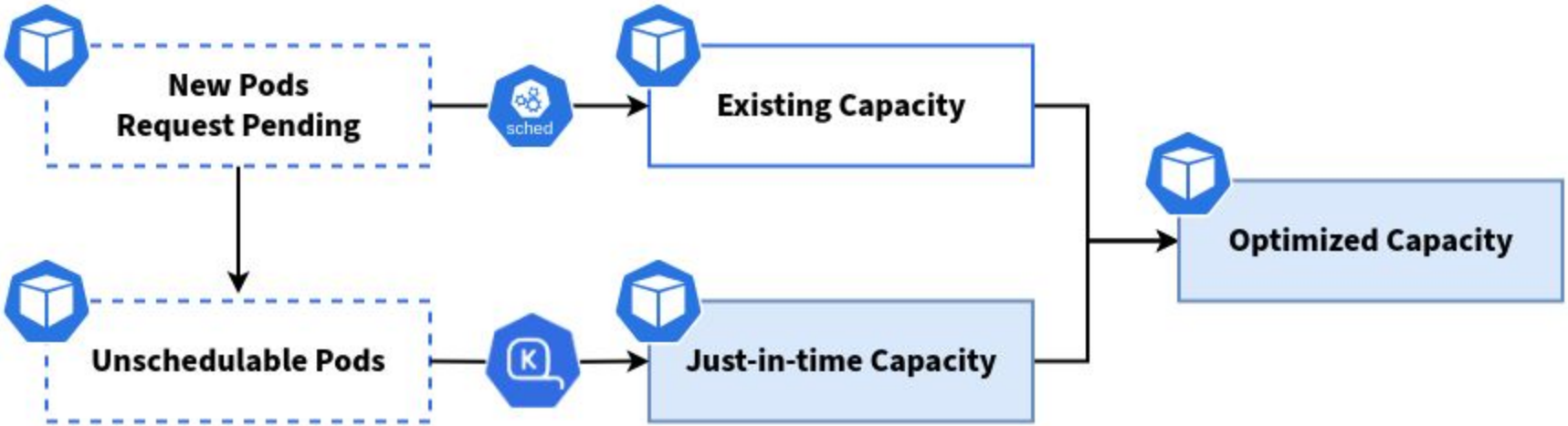
faster.. smarter?


New Pods
Request Pending
sched
Existing Capacity
Optimized Capacity
Unschedulable Pods
Just-in-time Capacity

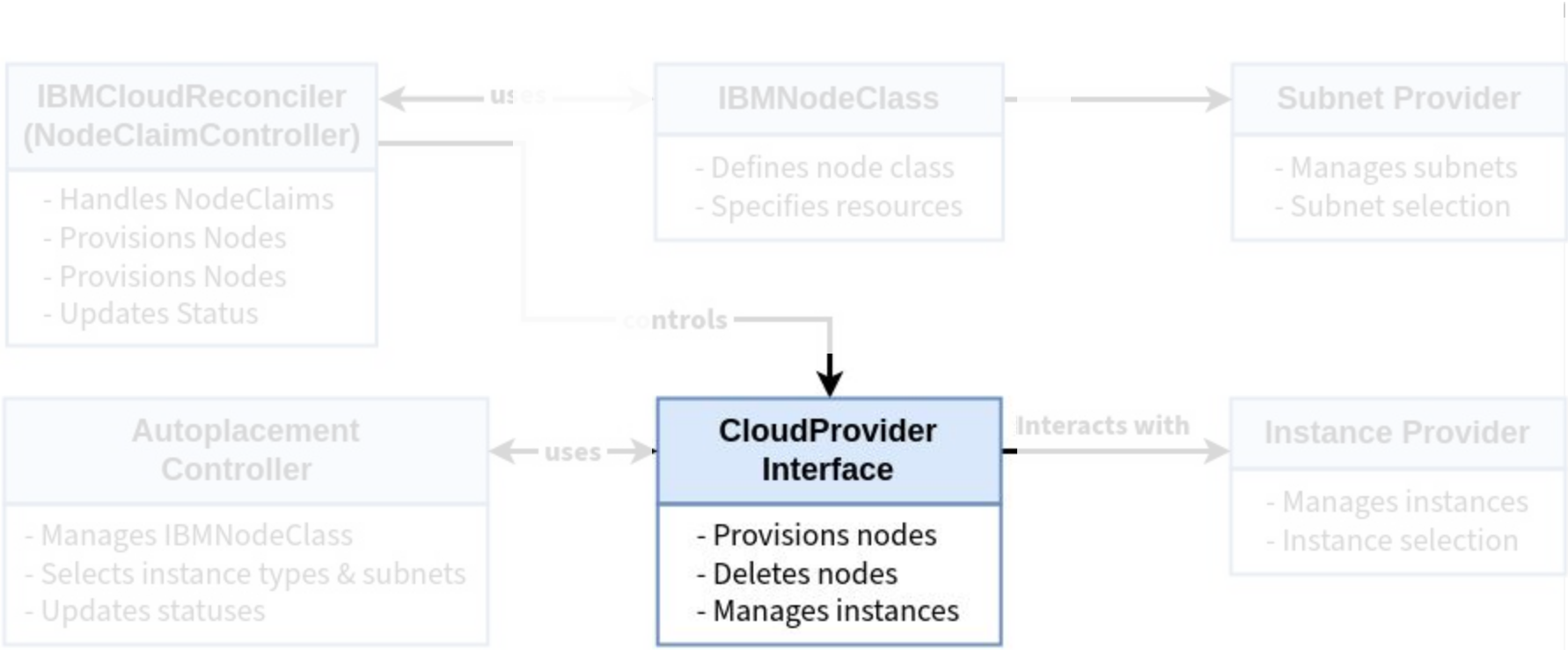

New Pods
Request Pending
sched
Existing Capacity
Optimized Capacity
Unschedulable Pods
Just-in-time Capacity

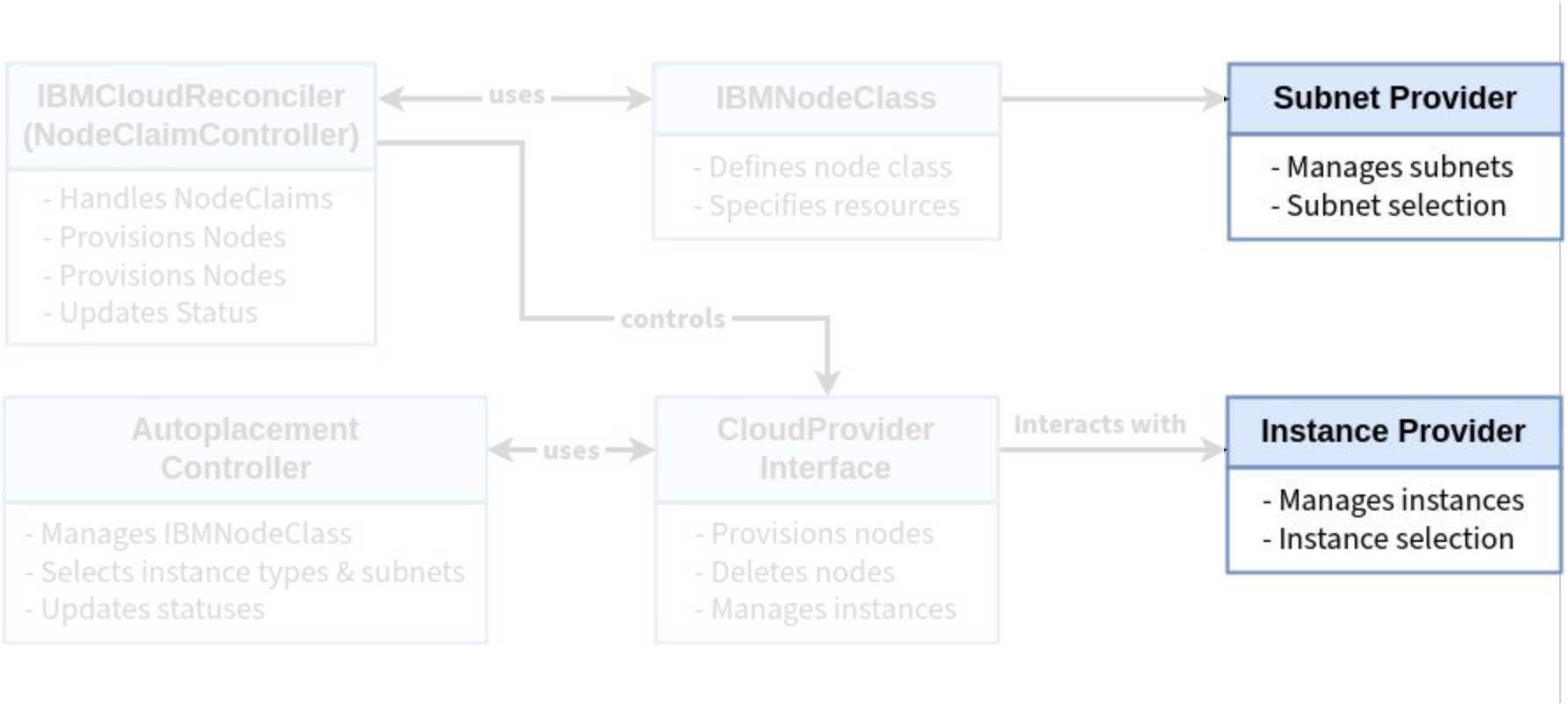

New Pods
Request Pending
Unschedulable Pods
sched
Existing Capacity
Just-in-time Capacity
Optimized Capacity

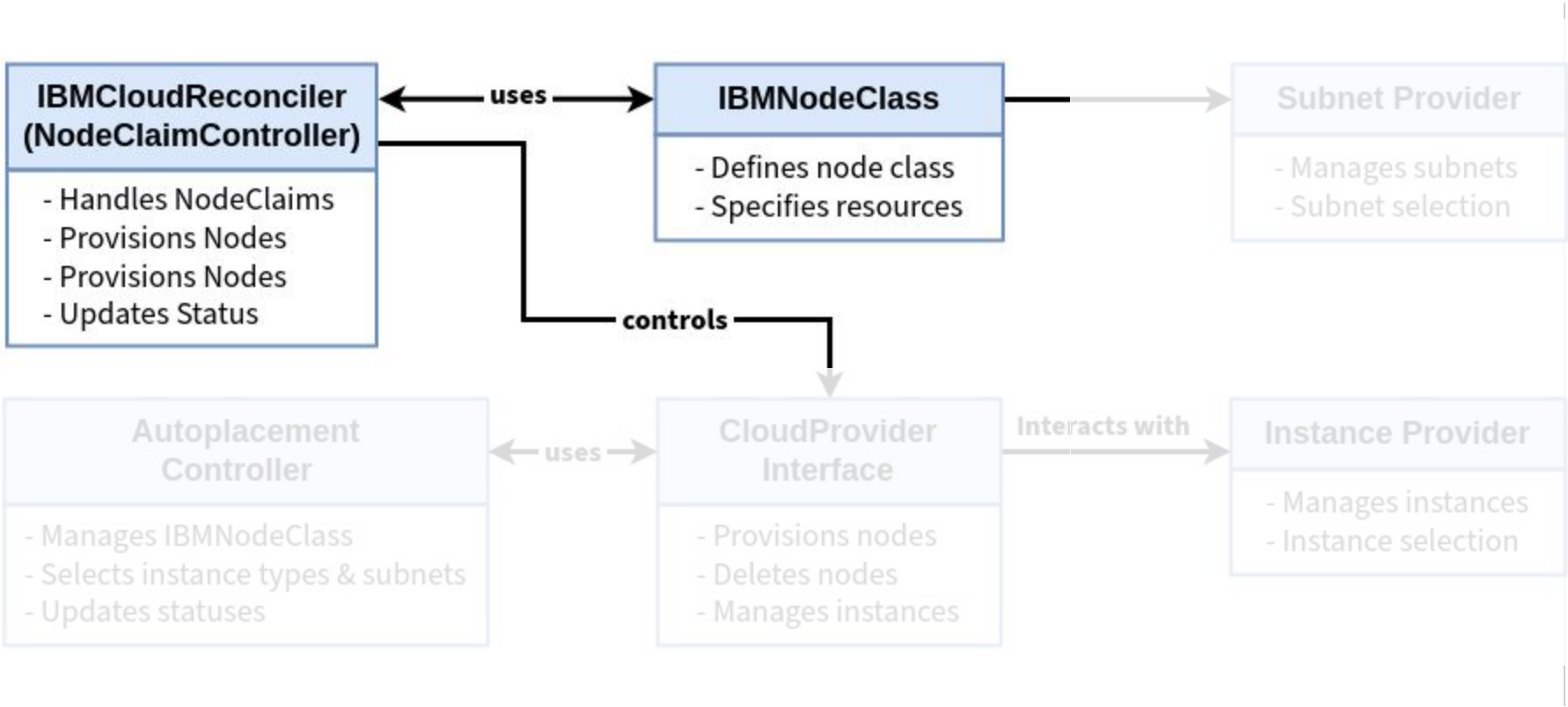

New Pods
Request Pending
sched
Existing Capacity
Optimized Capacity
Unschedulable Pods
K
Just-in-time Capacity

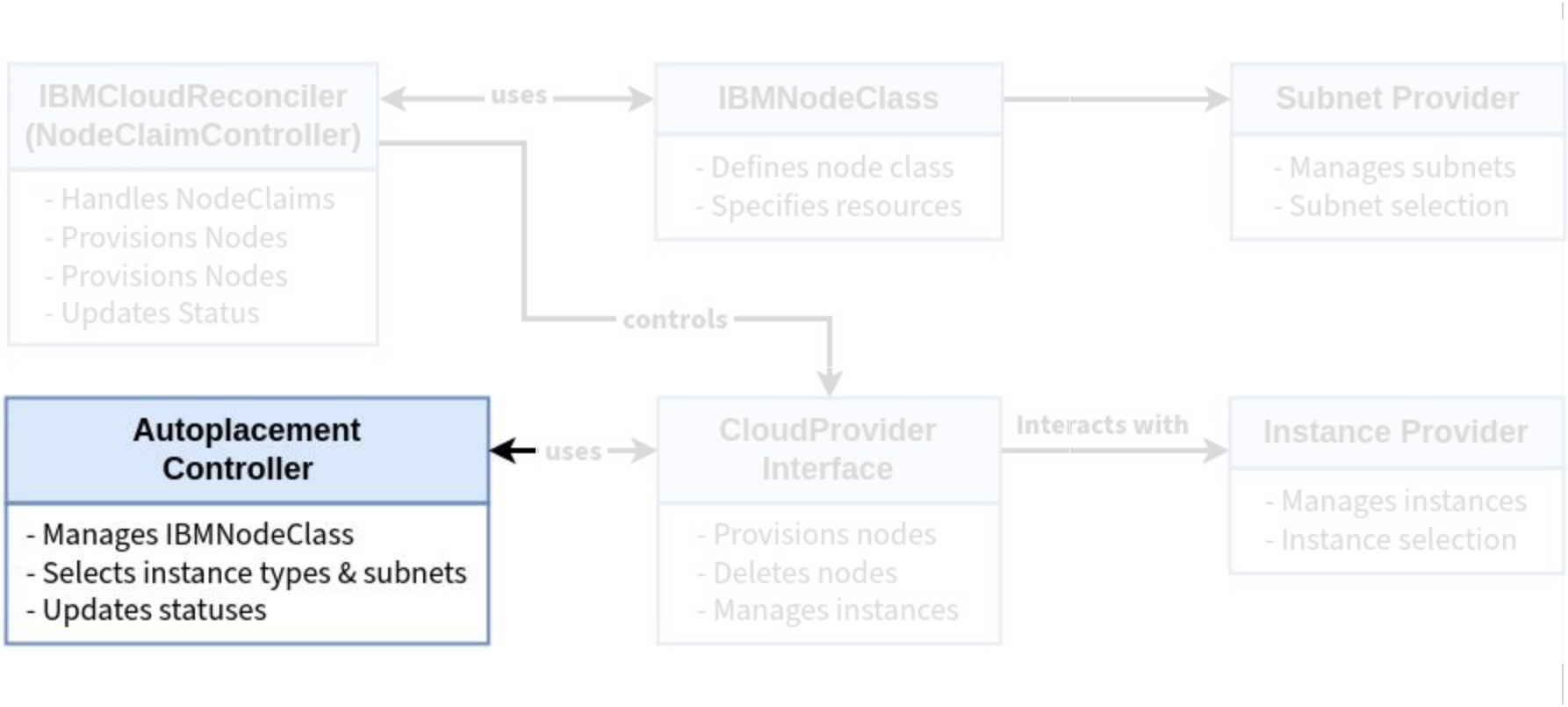

New Pods
Request Pending
sched
Existing Capacity
Optimized Capacity
Unschedulable Pods
Just-in-time Capacity

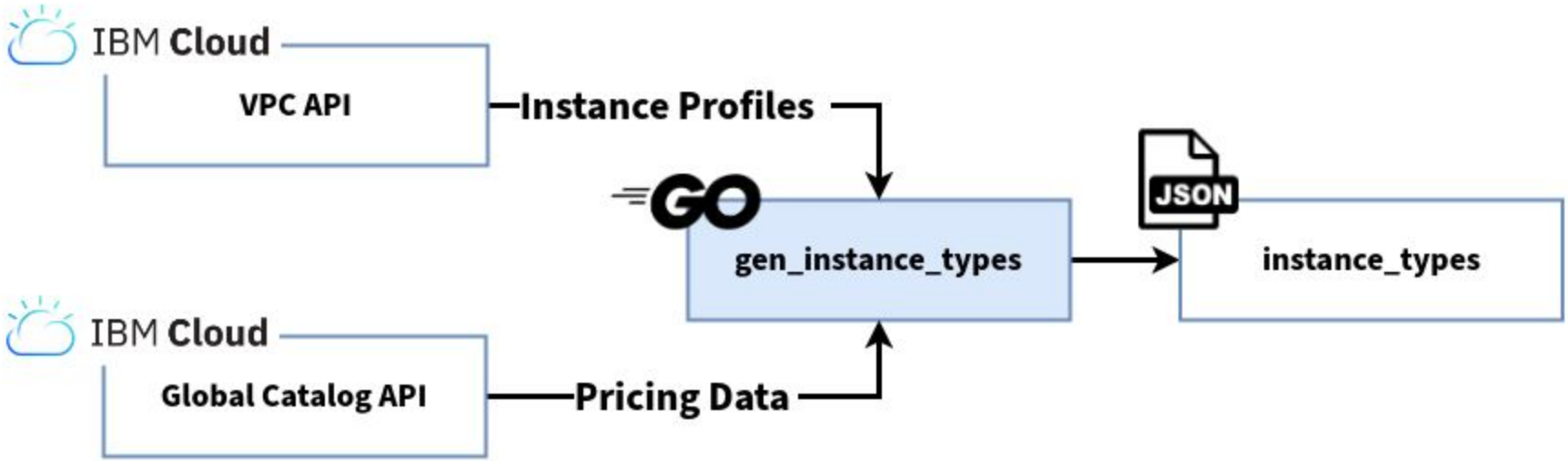
developing a karpenter provider

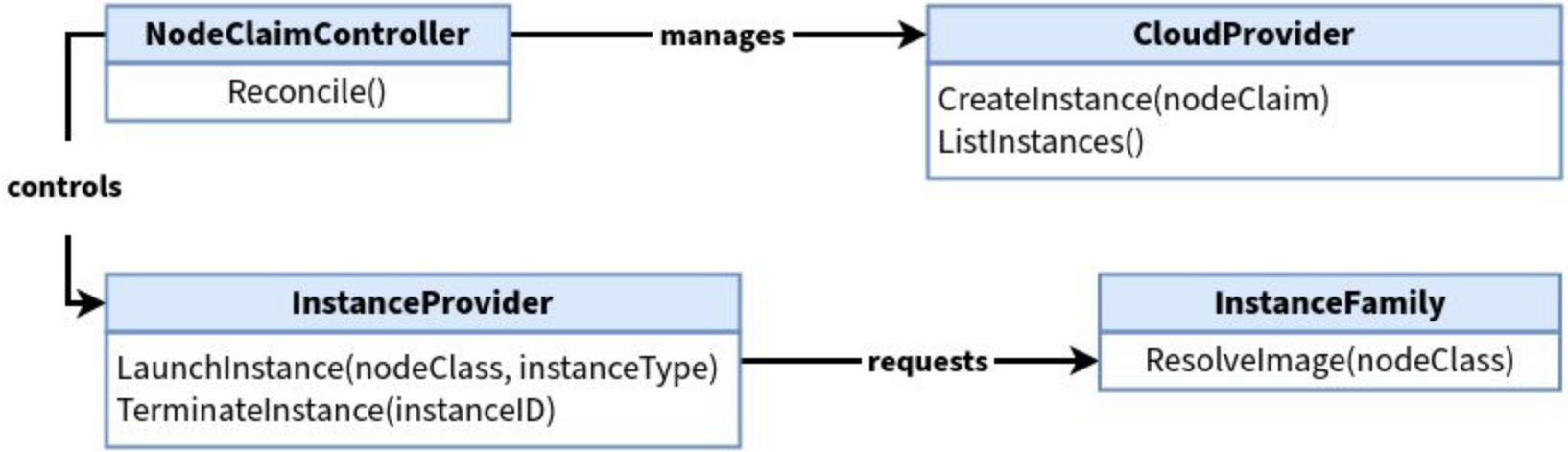

IBMCloudReconciler
(NodeClaimController)
- Handles NodeClaims
- Provisions Nodes
- Provisions Nodes
- Updates Status
uses
IBMNodeClass
- Defines node class
- Specifies resources
Subnet Provider
- Manages subnets
- Subnet selection
controls
Autoplacement
Controller
- Manages IBMNodeClass
- Selects instance types & subnets
- Updates statuses
uses
CloudProvider
Interface
- Provisions nodes
- Deletes nodes
- Manages instances
Interacts with
Instance Provider
- Manages instances
- Instance selection

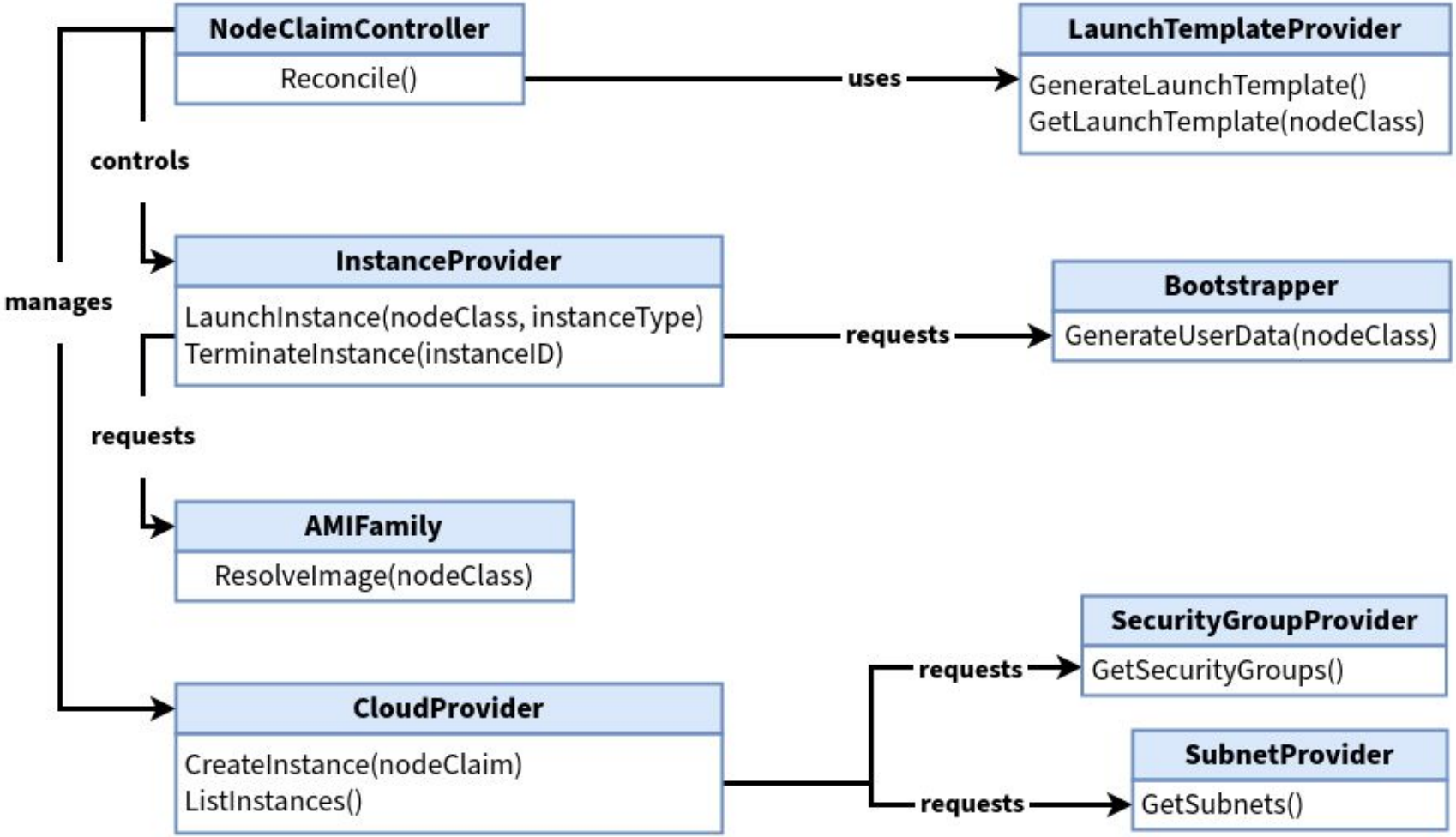

IBMCloudReconciler
(NodeClaimController)
- Handles NodeClaims
- Provisions Nodes
- Provisions Nodes
- Updates Status
uses
IBMNodeClass
- Defines node class
- Specifies resources
Subnet Provider
- Manages subnets
- Subnet selection
controls
Autoplacement
Controller
- Manages IBMNodeClass
- Selects instance types & subnets
- Updates statuses
uses
CloudProvider
Interface
- Provisions nodes
- Deletes nodes
- Manages instances
Interacts with
Instance Provider
- Manages instances
- Instance selection


IBMCloudReconciler
(NodeClaimController)
- Handles NodeClaims
- Provisions Nodes
- Provisions Nodes
- Updates Status
uses
IBMNodeClass
- Defines node class
- Specifies resources
Subnet Provider
- Manages subnets
- Subnet selection
controls
Autoplacement
Controller
- Manages IBMNodeClass
- Selects instance types & subnets
- Updates statuses
uses
CloudProvider
Interface
- Provisions nodes
- Deletes nodes
- Manages instances
Interacts with
Instance Provider
- Manages instances
- Instance selection

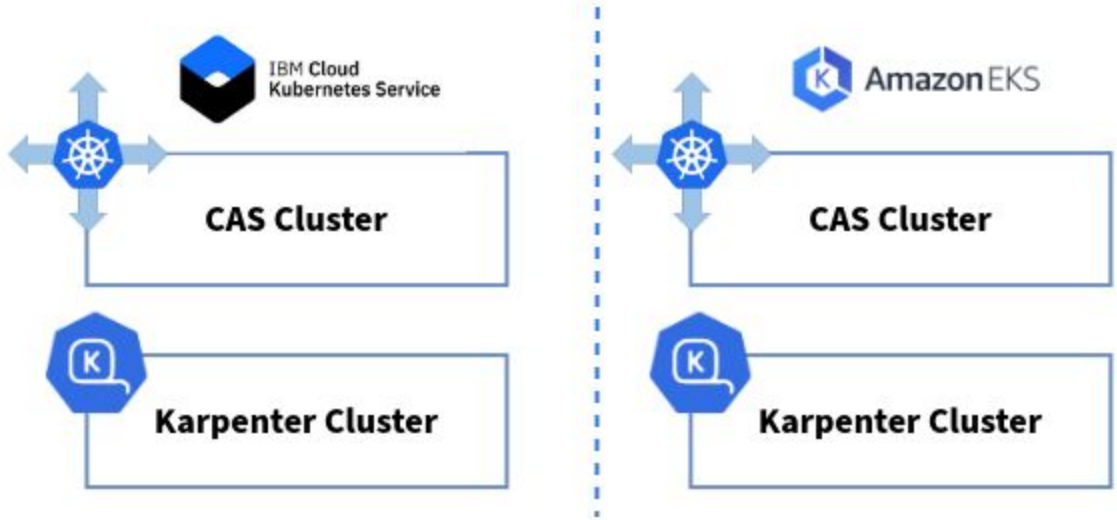

IBMCloudReconciler
(NodeClaimController)
- Handles NodeClaims
- Provisions Nodes
- Provisions Nodes
- Updates Status
uses
IBMNodeClass
- Defines node class
- Specifies resources
Subnet Provider
- Manages subnets
- Subnet selection
controls
Autoplacement
Controller
- Manages IBMNodeClass
- Selects instance types & subnets
- Updates statuses
uses
CloudProvider
Interface
- Provisions nodes
- Deletes nodes
- Manages instances
Interacts with
Instance Provider
- Manages instances
- Instance selection

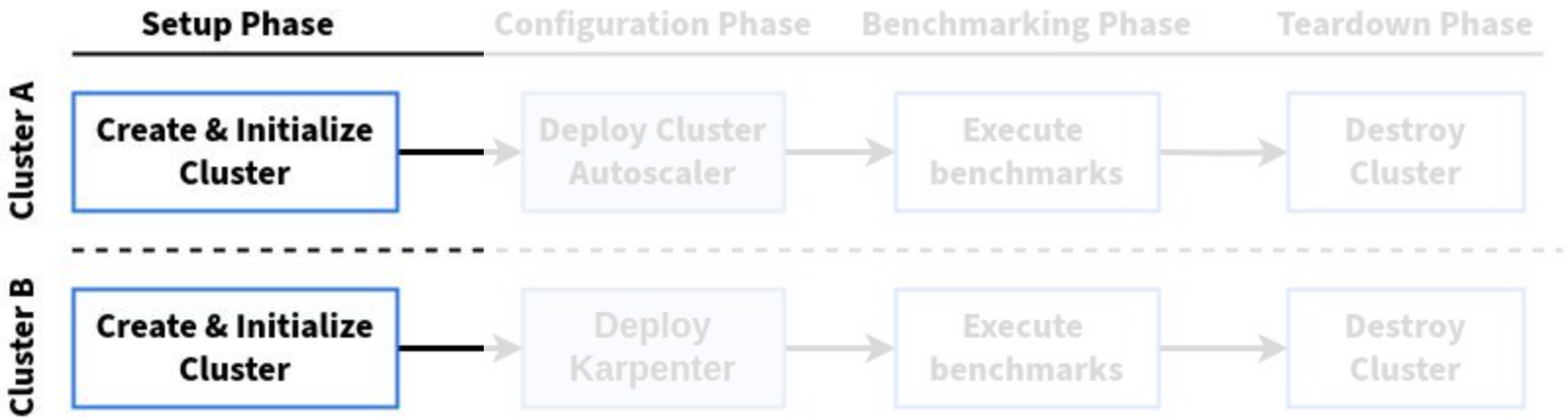

IBM Cloud
VPC API
Instance Profiles
GO
gen_instance_types
JSON
instance_types
IBM Cloud
Global Catalog API
Pricing Data

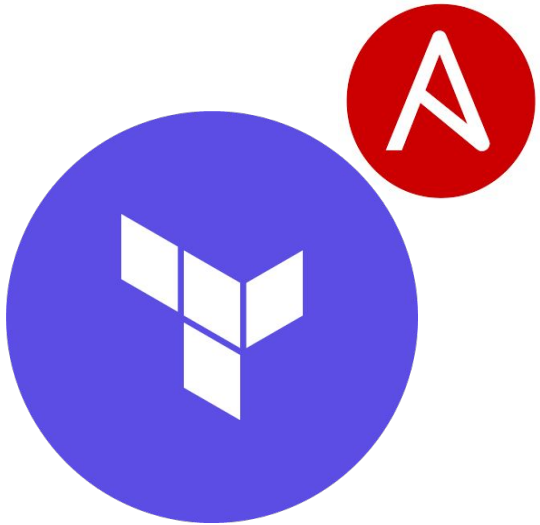

NodeClaimController
Reconcile()
manages
CloudProvider
CreateInstance(nodeClaim)
ListInstances()
controls
InstanceProvider
LaunchInstance(nodeClass, instanceType)
TerminateInstance(instanceID)
requests
InstanceFamily
ResolveImage(nodeClass)


NodeClaimController
Reconcile()
uses
LaunchTemplateProvider
GenerateLaunchTemplate()
GetLaunchTemplate(nodeClass)
controls
manages
InstanceProvider
LaunchInstance(nodeClass, instanceType)
TerminateInstance(instanceID)
requests
Bootstrapper
GenerateUserData(nodeClass)
requests
AMIFamily
ResolveImage(nodeClass)
SecurityGroupProvider
requests
GetSecurityGroups()
CloudProvider
CreateInstance(nodeClaim)
ListInstances()
SubnetProvider
requests
GetSubnets()

benchmarking CAS vs karpenter

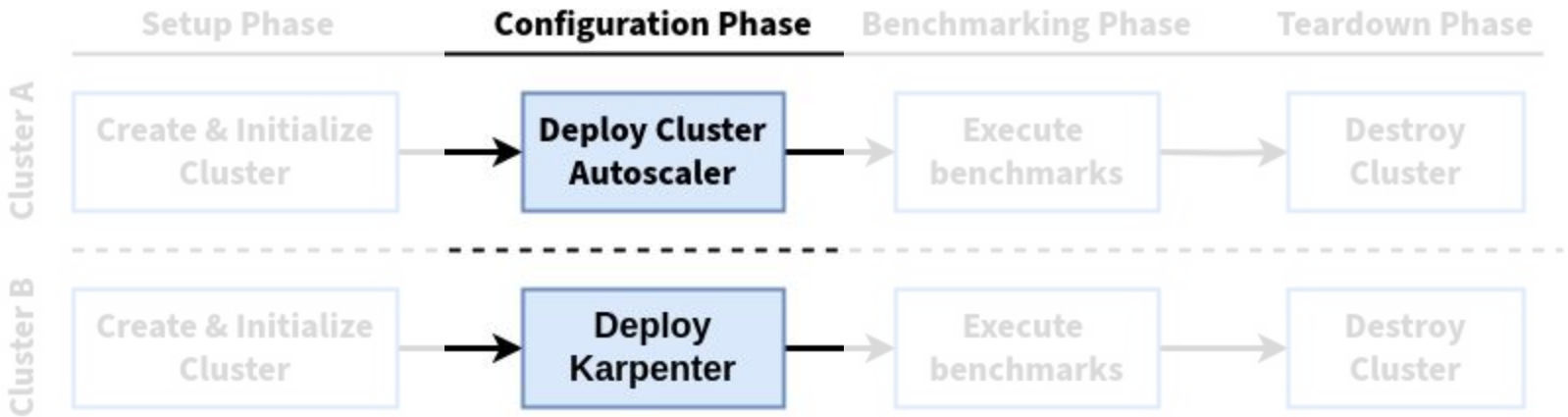

IBM Cloud
Kubernetes Service
CAS Cluster
Karpenter Cluster
Amazon EKS
CAS Cluster
Karpenter Cluster

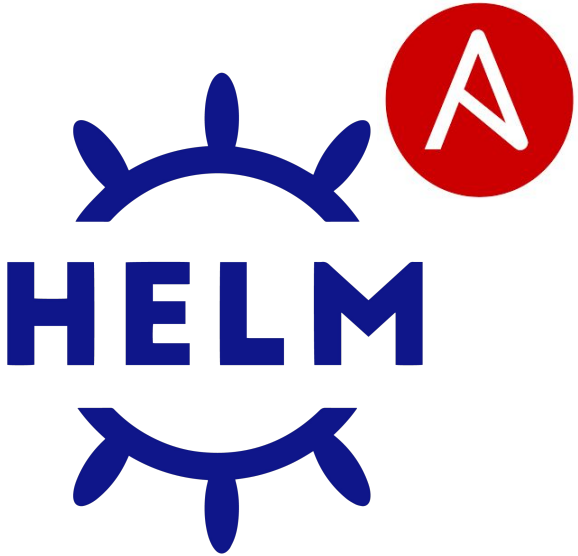

Setup Phase
Configuration Phase
Benchmarking Phase
Teardown Phase
Cluster A
Create & Initialize Cluster
Deploy Cluster Autoscaler
Execute benchmarks
Destroy Cluster
Cluster B
Create & Initialize Cluster
Deploy Karpenter
Execute benchmarks
Destroy Cluster



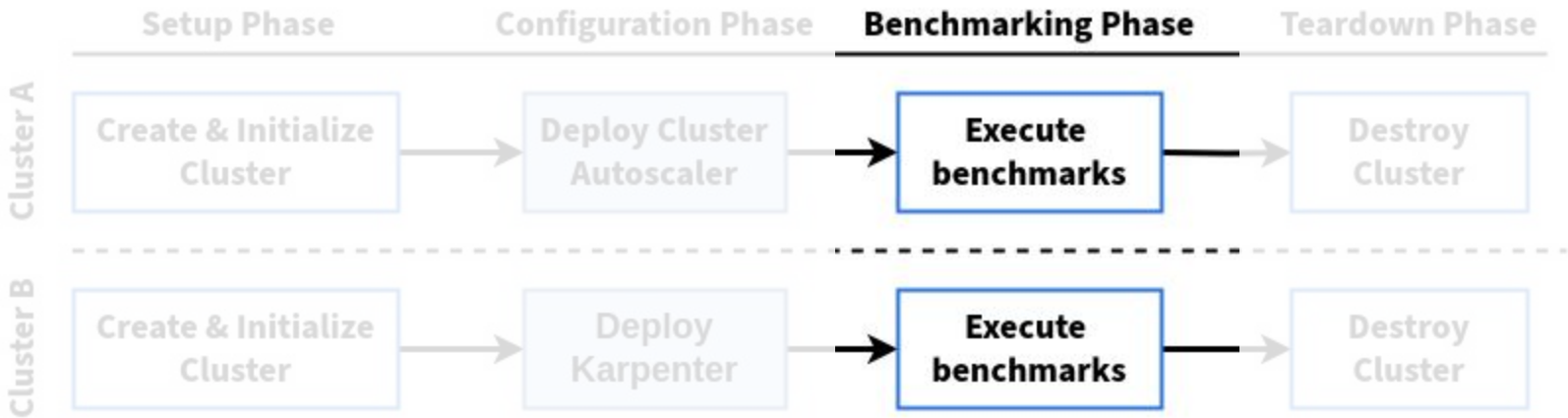
```hcl
# AWS EKS cluster
resource "aws_eks_cluster" "cas_cluster" {
  name    = "cas-eks"
  tags = {
    tag = "cas-${var.tag_uuid}"  # Unique experiment ID
  }
}

# IBM IKS cluster
resource "ibm_container_vpc_cluster" "cas_cluster" {
  name     = "cas-iks"
  vpc_id   = var.vpc_id
  tags     = ["cas-${var.tag_uuid}"]  # Same experiment ID
  timeouts {
    create = "150m"
  }
}
```


Setup Phase
Configuration Phase
Benchmarking Phase
Teardown Phase
Cluster A
Create & Initialize Cluster
Deploy Cluster Autoscaler
Execute benchmarks
Destroy Cluster
Cluster B
Create & Initialize Cluster
Deploy Karpenter
Execute benchmarks
Destroy Cluster

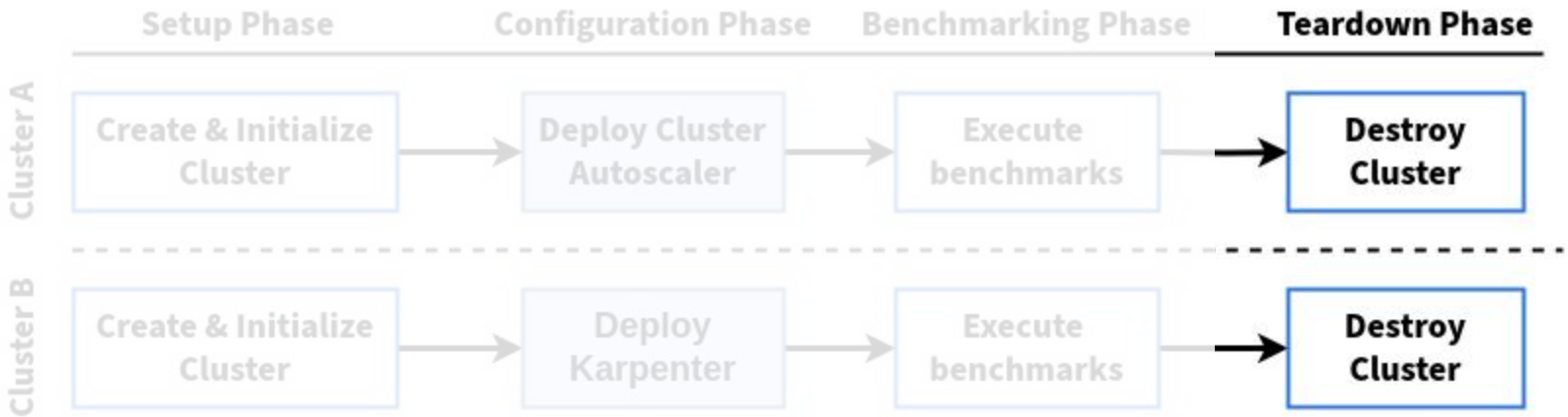

HELM

```
# Deploy to both autoscalers simultaneously
loop:
  - karpenter
  - cas
environment:
  KUBECONFIG: "{{ lookup('vars', 'KUBECONFIG_EKS_' ~ (item | upper)) }}"
```


Setup Phase
Configuration Phase
Benchmarking Phase
Teardown Phase
Cluster A
Create & Initialize Cluster
Deploy Cluster Autoscaler
Execute benchmarks
Destroy Cluster
Cluster B
Create & Initialize Cluster
Deploy Karpenter
Execute benchmarks
Destroy Cluster

kube-burner

```yaml
# Kube-burner heterogeneous workload configuration
jobs:
  - name: cpu-intensive
    jobIterations: 50
    qps: 40
    burst: 50
    objects:
      - objectTemplate: deployment-cpu-l
        replicas: 20      # Large CPU pods
      - objectTemplate: deployment-memory-l
        replicas: 30      # Large memory pods
    podWait: false        # Create scaling pressure
    waitWhenFinished: false  # Concurrent stress
```

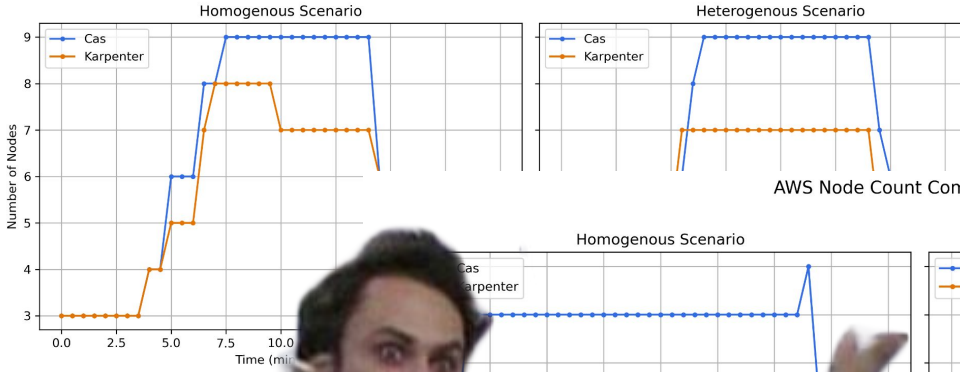

Setup Phase
Configuration Phase
Benchmarking Phase
Teardown Phase
Cluster A
Create & Initialize Cluster
Deploy Cluster Autoscaler
Execute benchmarks
Destroy Cluster
Cluster B
Create & Initialize Cluster
Deploy Karpenter
Execute benchmarks
Destroy Cluster

```
# Collect core performance metrics
- name: Query node provisioning latency
  uri:
    url: "{{ PROM_ENDPOINT }}/api/v1/query_range"
    headers:
      Authorization: "Bearer {{ PROM_TOKEN }}"
    body_format: form-urlencoded
    body:
      query: "cluster_autoscaler_nodes_count"
      start: "{{ experiment_start_time }}"
      end: "{{ experiment_end_time }}"
      step: 30s
  register: node_latency_metrics
```

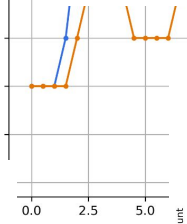
quiz time!

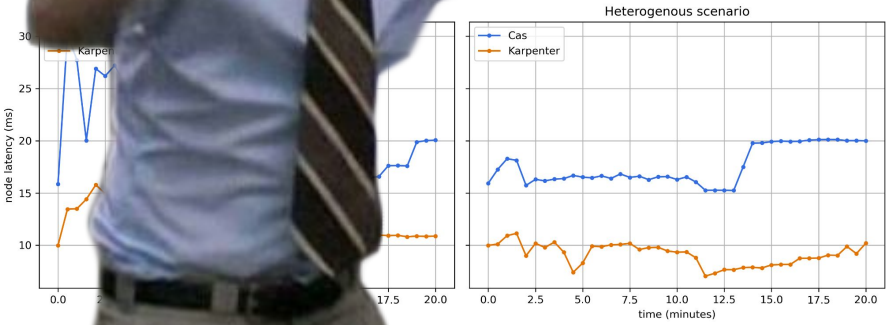
what did we find out?

IBM Node Count Comparison

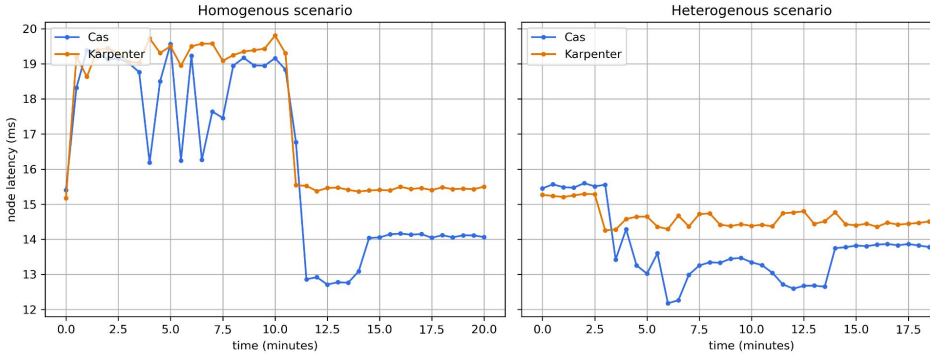


AWS Node Count Con

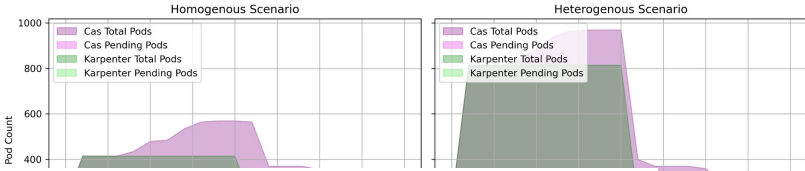




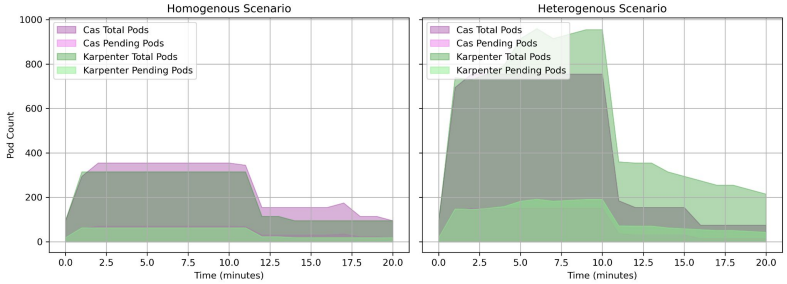
IBM node latency comparison



IBM Total vs Pending Pods



AWS Total vs Pending Pods



33%

fewer nodes required vs. CAS

47%

faster than CAS in
Node Provisioning Latency

platform	autoscaler	$/hr	$/mo
homogenous			
AWS	CAS	0.87	626.40
AWS	karpenter	0.58	417.60
IBM	CAS	0.71	511.20
IBM	karpenter	0.57	410.40
heterogenous			
AWS	CAS	0.75	540.00
AWS	karpenter	0.71	511.20
IBM	CAS	0.73	525.60
IBM	karpenter	0.56	403.20

33%

monthly cost reduction vs. CAS

does the world really need another k8s cluster autoscaler?

it depends..

further reading

kube-burner
go.josie.lol/kb

experiment infra git repo
go.josie.lol/dcgit

IBM cloud karpenter provider git
go.josie.lol/ibmgit

kube-burner

IBM Cloud

https://josie.lol
josie@redhat.com

q&a