# Node Size Matters

*Running K8s as Cheaply as Possible*


Michael Dresser
Staff Software Engineer
Kubecost

Alex Meijer
Staff Software Engineer
Kubecost

# Overview

- What is overhead?
- How overhead is calculated
- OpenCost 101
- **Demo:** using OpenCost to determine overhead
- Study of big three managed k8s node overhead
  - Key Takeaways
- Node sizing: approaches and challenges
- Incorporating overhead into node sizing decisions
  - Kubecost's new node-sizing algorithm

# Defining Overhead

**Capacity - Allocatable = Overhead**

**What you pay for**

**What you can use**

Overhead includes:

- Kubelet
- Control Plane (if applicable)
- Container Runtime itself
- Node OS
- Any software running directly on node

**Overhead *does not* include:**

- **Prometheus**
- **Calico/Weave/CNI Pods**
- **DNS**
- **Cert Manager**
- **kube-system**
- ***Any pods running in k8s***

# Calculating Overhead (Via Kubectl)

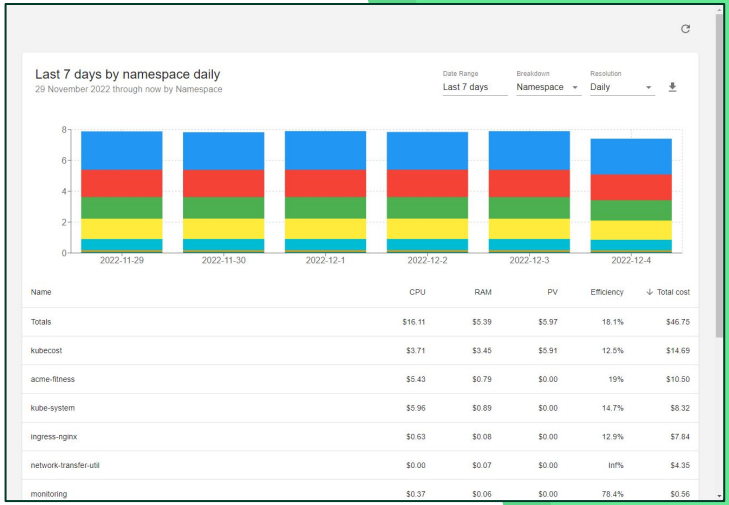
`kubectl describe node`

```
Name:               < redacted >
Labels:             beta.kubernetes.io/instance-type=n1-standard-2
                    cloud.google.com/machine-family=n1
                    node.kubernetes.io/instance-type=n1-standard-2
Capacity:
  attachable-volumes-gce-pd:  127
  cpu:                        2
  ephemeral-storage:          5952044Ki
  hugepages-1Gi:              0
  hugepages-2Mi:              0
  memory:                     7632436Ki
  pods:                       110
Allocatable:
  attachable-volumes-gce-pd:  127
  cpu:                        1930m
  ephemeral-storage:          116694622
  hugepages-1Gi:              0
  hugepages-2Mi:              0
  memory:                     5752372Ki
  pods:                       110
```

kubecost

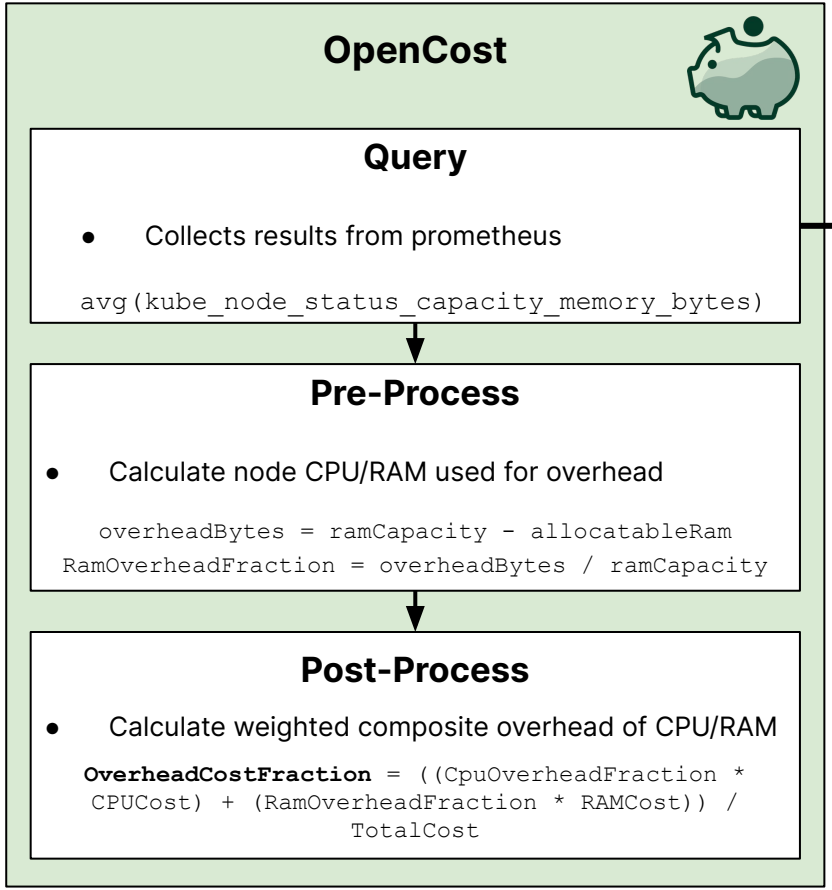
# OpenCost 101

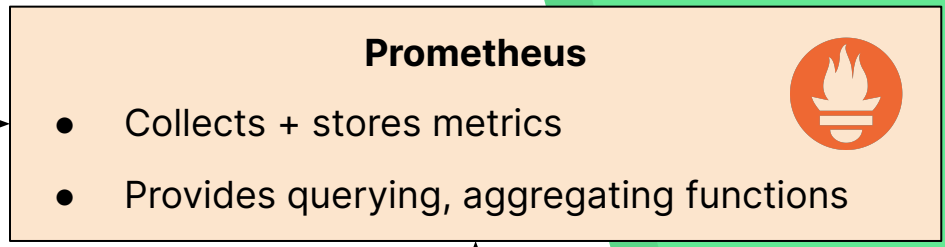
- CNCF sandbox project
- Same core logic powering Kubecost
- Non-Kubecost contributors include
  - Microsoft
  - Grafana Labs
  - Many others
- Powerful REST API
  - Filtering
  - Asset support
  - Many feature flags
    - Govern additional available computation

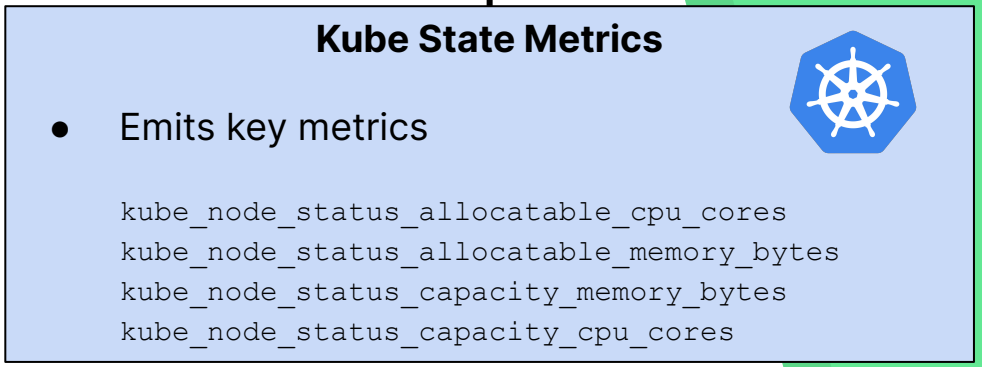



kubecost

# Calculating Overhead In OpenCost




kubecost

# Demo: GKE cluster node overhead with OpenCost

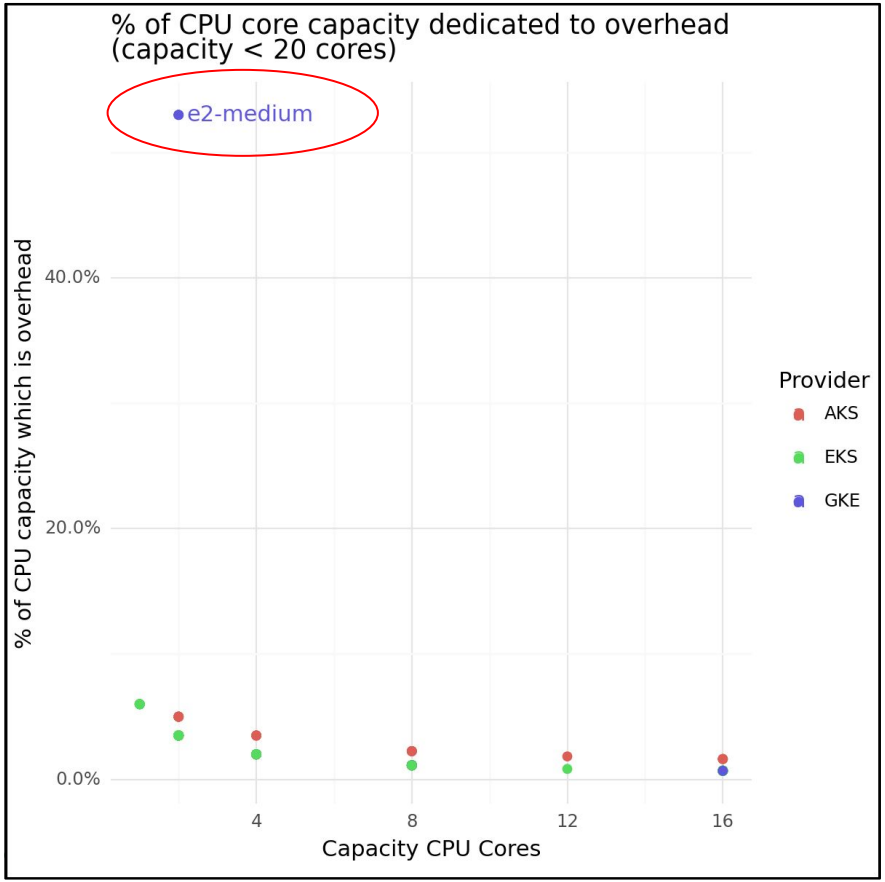
# Overhead Study: Objectives

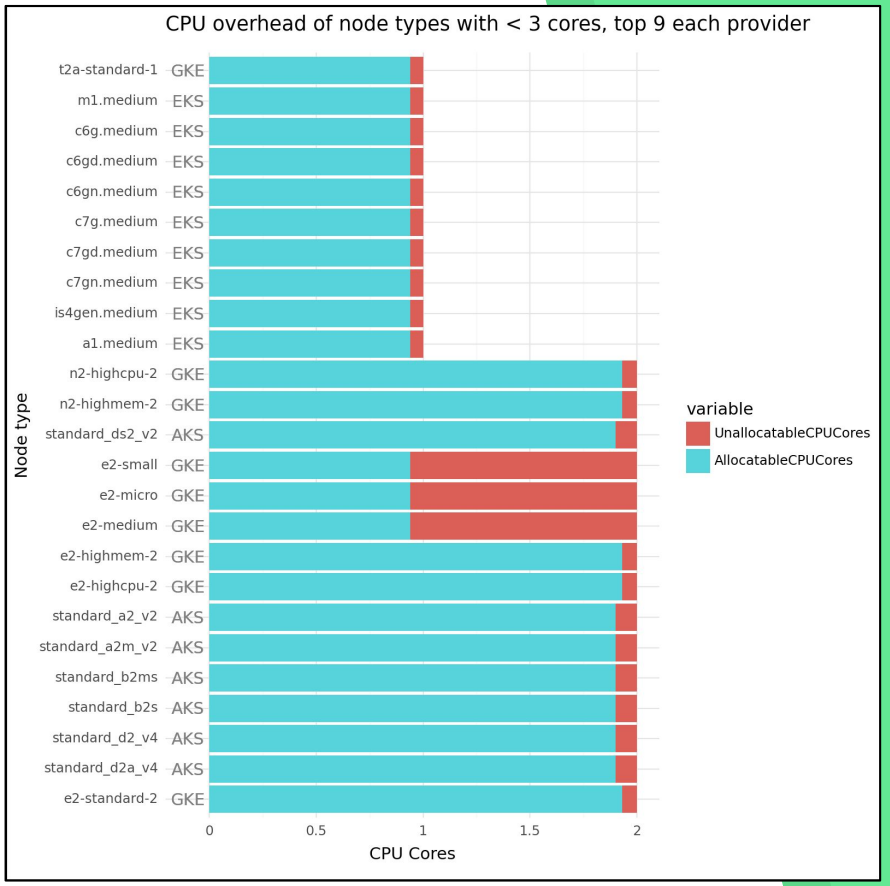
- To gain a better understanding of overhead
- To explore empirically how overhead behaves
  - As node size increases
  - As node family varies
  - As provider varies
- To inform/generate input data for node sizing algorithm modifications

# Overhead Study: Methodology

1. API call to cloud provider price endpoint to obtain instance types, prices
2. Deploy each node type (batches of 10) to cluster using provider CLI
3. Use OpenCost and kubectl to obtain allocatable, capacity, overhead, and node metadata
    a. Save to JSON file
4. Run analysis pipeline on JSON files

# Overhead Study: Findings



# Case Study: GCP e2 burstable

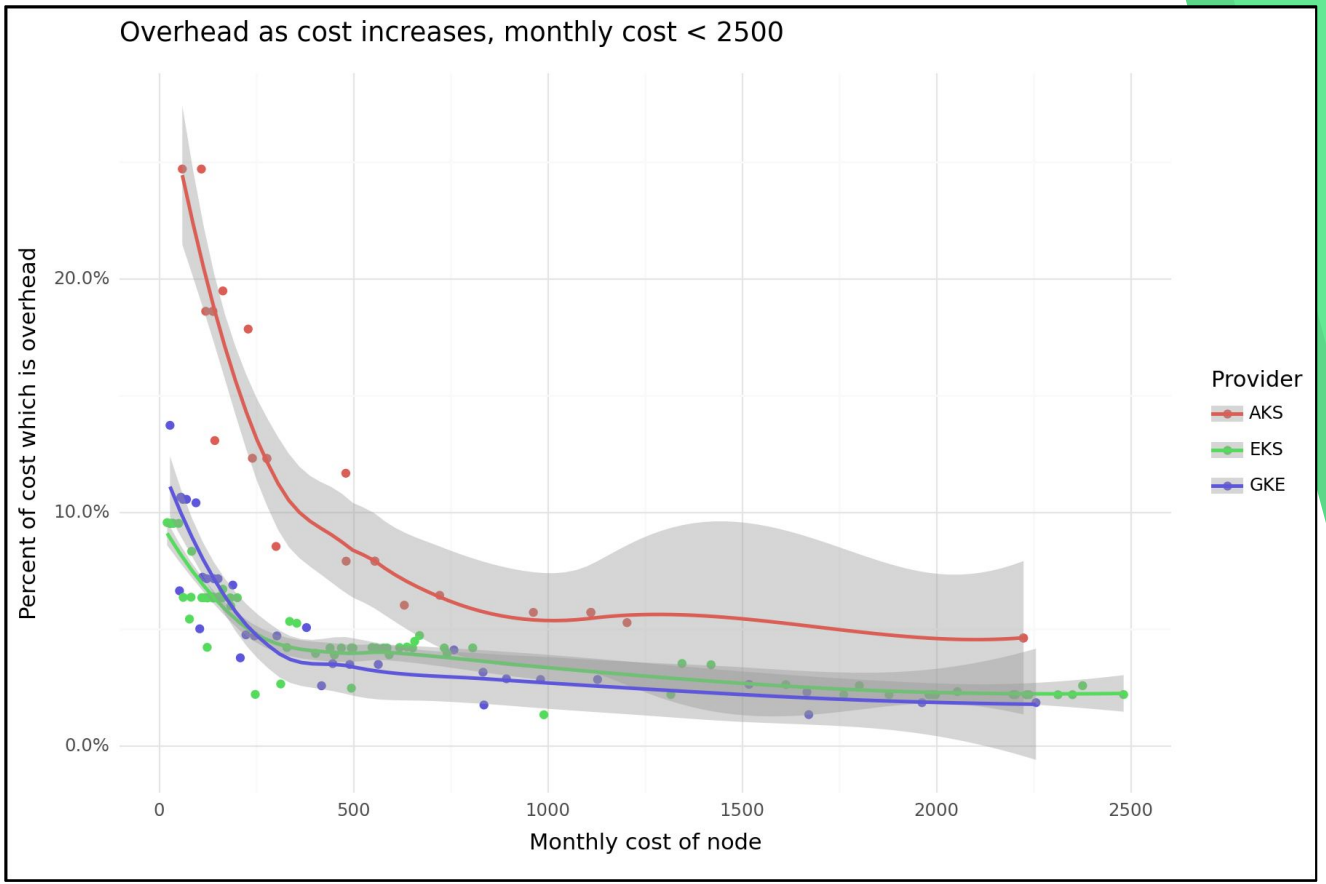
- GCP's e2 class seems to have significant overhead
- GCP Docs for e2-medium
  - "`e2-medium` sustains 2 vCPUs, each at 50% of CPU time, totaling 100% CPU time and effectively consuming 1.0 cores."
  - "Each vCPU can burst up to 100% of CPU time, for short periods, before returning to its prescribed CPU time limitation."
- How do prices stack up?
  - e2-medium: $0.033
  - Custom 1 VCPU/4GB RAM e2 family: $0.035
  - Good deal: Slightly cheaper for a burstable instance
- You aren't quite getting what you think you are
  - How k8s interprets these
  - How these work
- We generally removed these as outliers

| Machine type | vCPUs | Memory | Price (USD) |
|---|---|---|---|
| e2-micro | 2 | 1GB | $0.008376 |
| e2-small | 2 | 2GB | $0.016751 |
| e2-medium | 2 | 4GB | $0.033503 |

| Item | On-demand price |
|---|---|
| Custom vCPUs | $0.02289 / vCPU hour |
| Custom Memory | $0.003067 / GB hour |

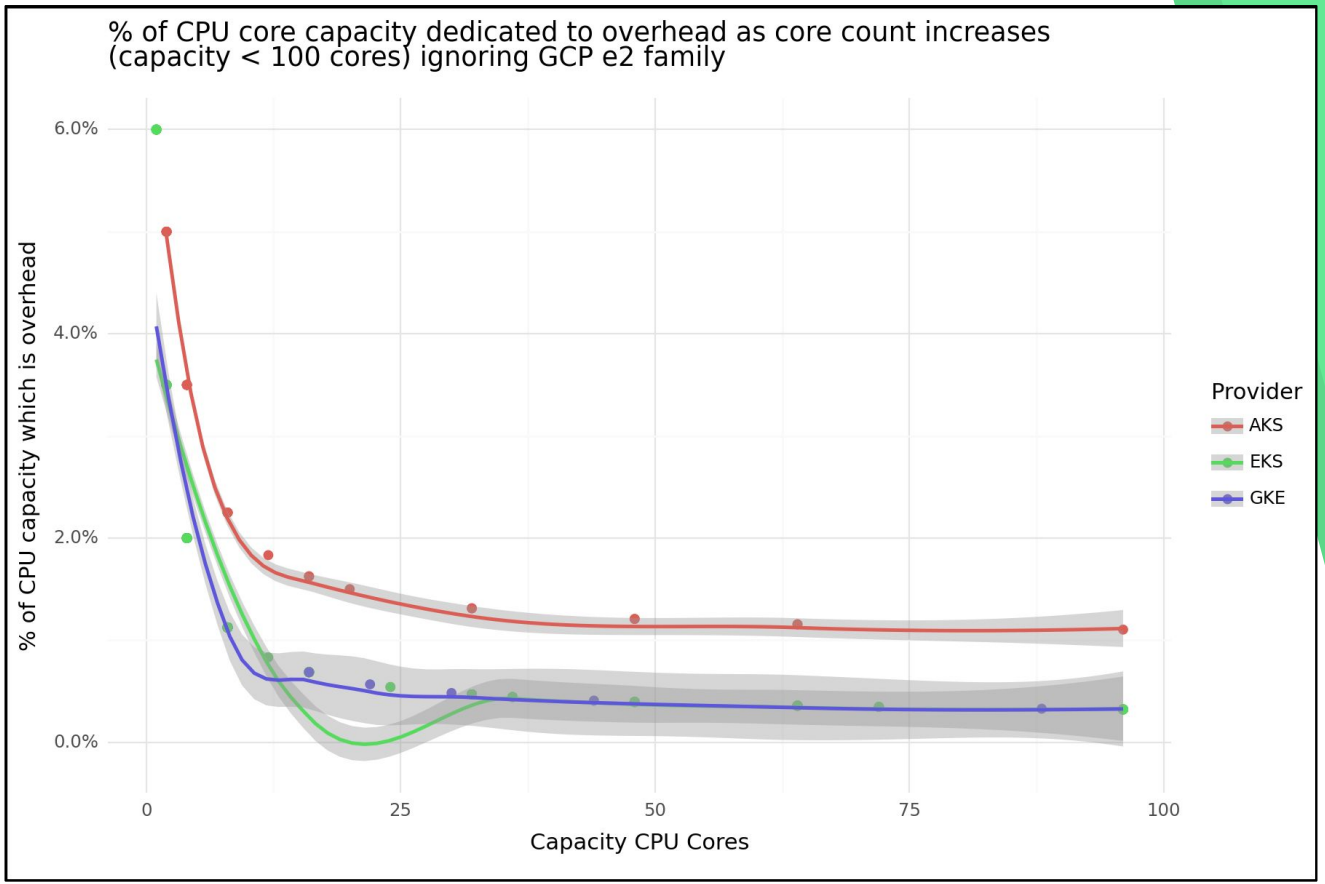
kubecost

# Overhead Study: Composite Trends



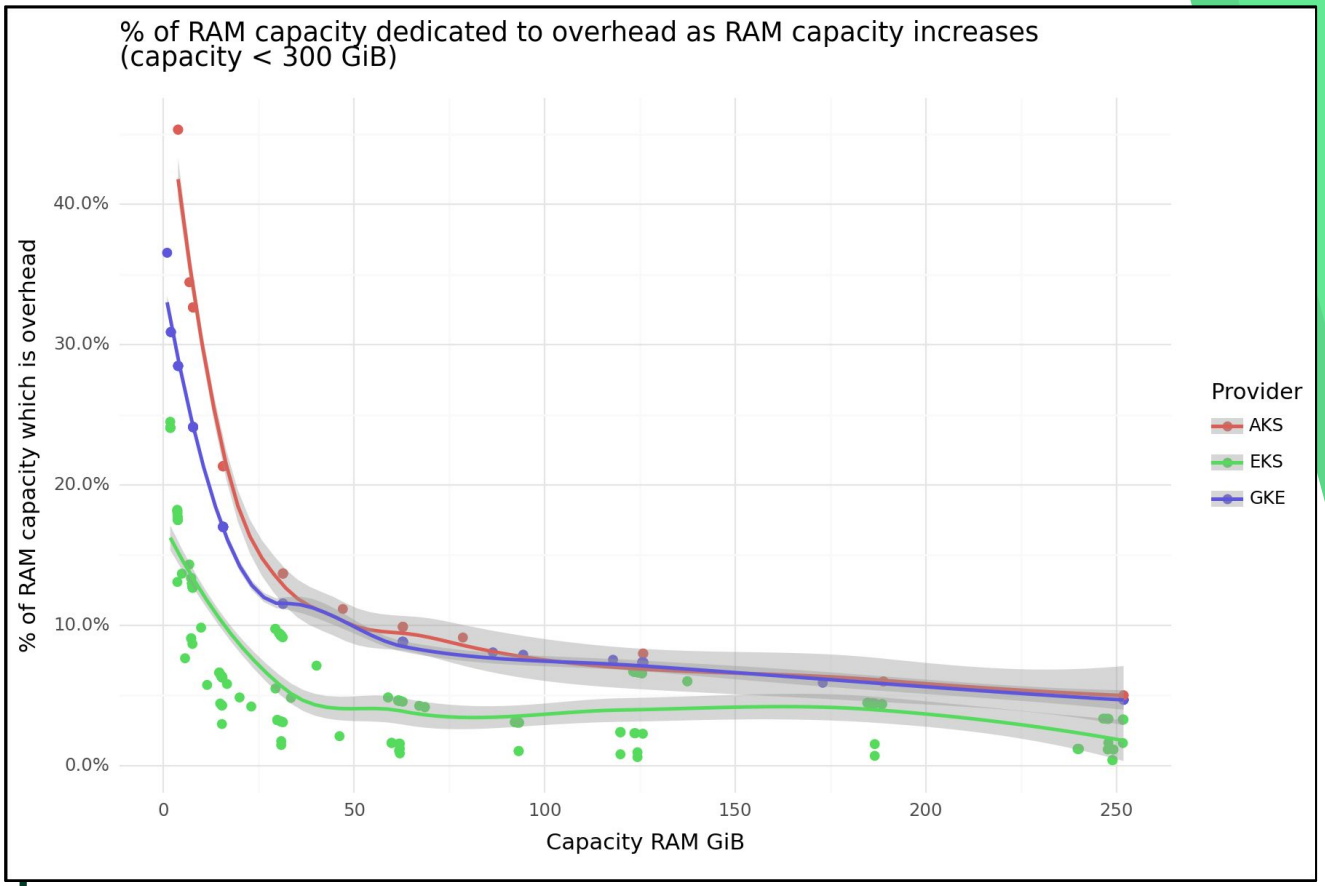

kubecost

# Overhead Study: CPU Trends



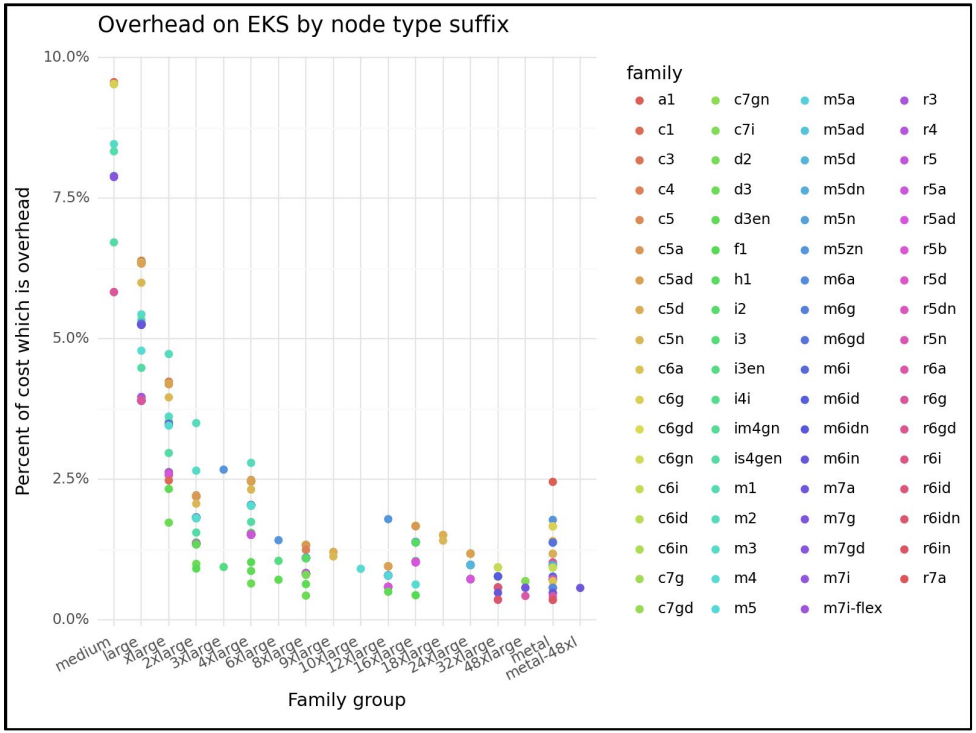

kubecost

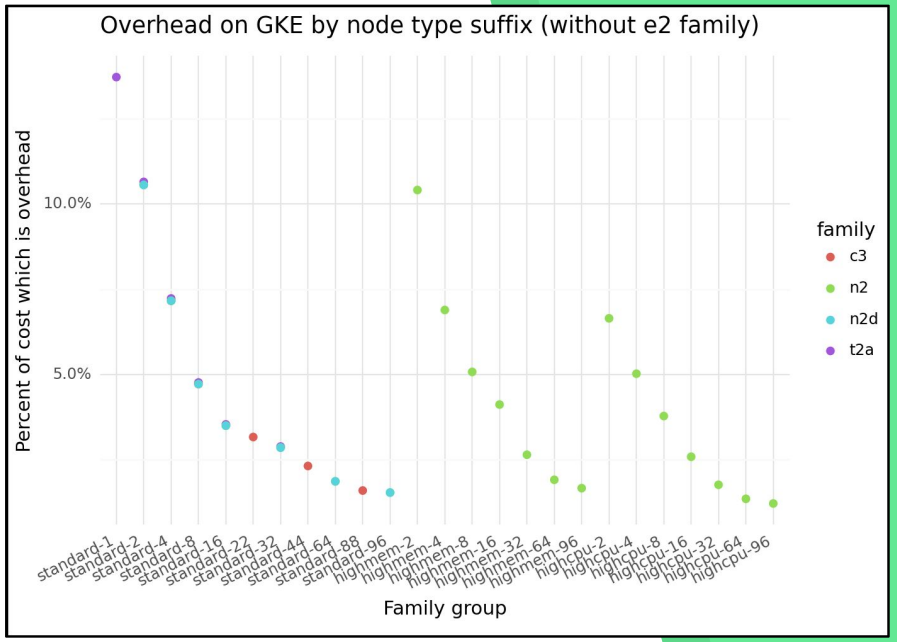
# Overhead Study: RAM Trends




kubecost

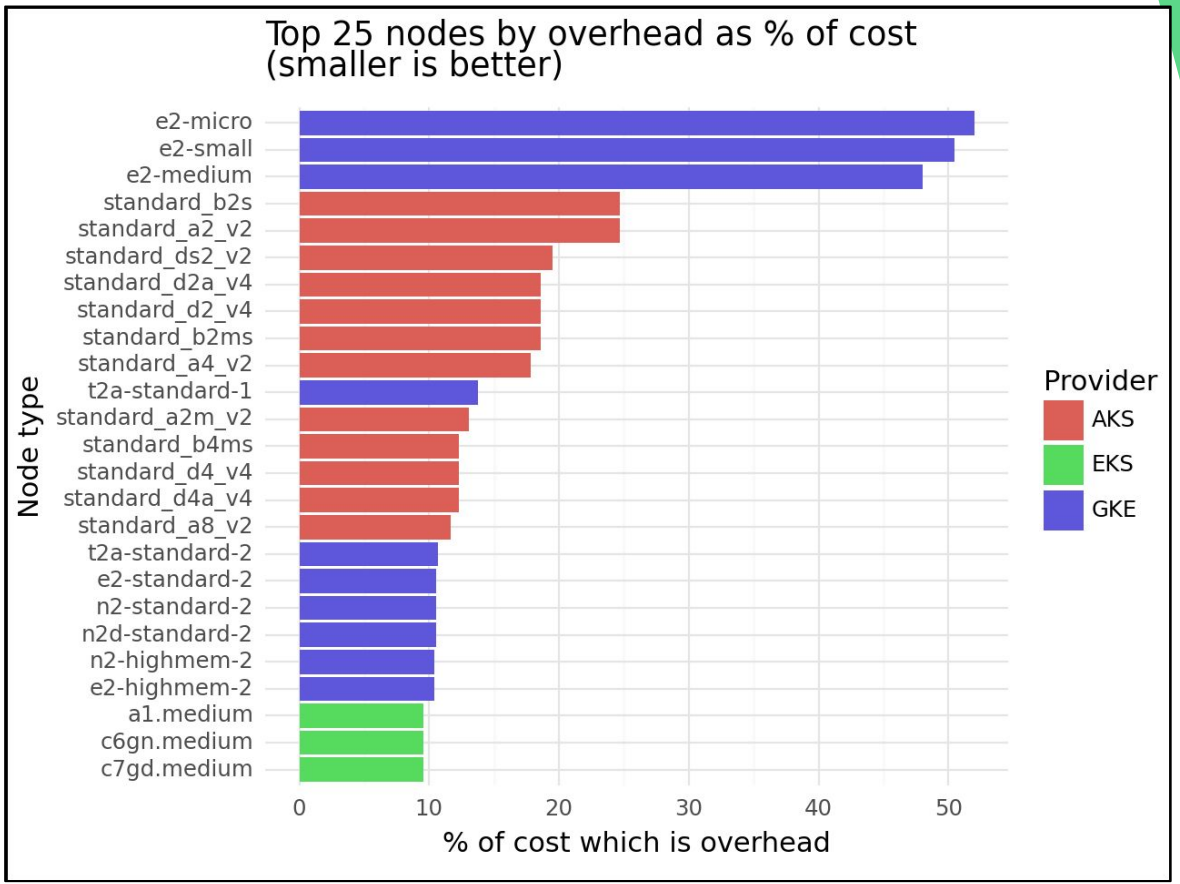
# Overhead Study: By Family



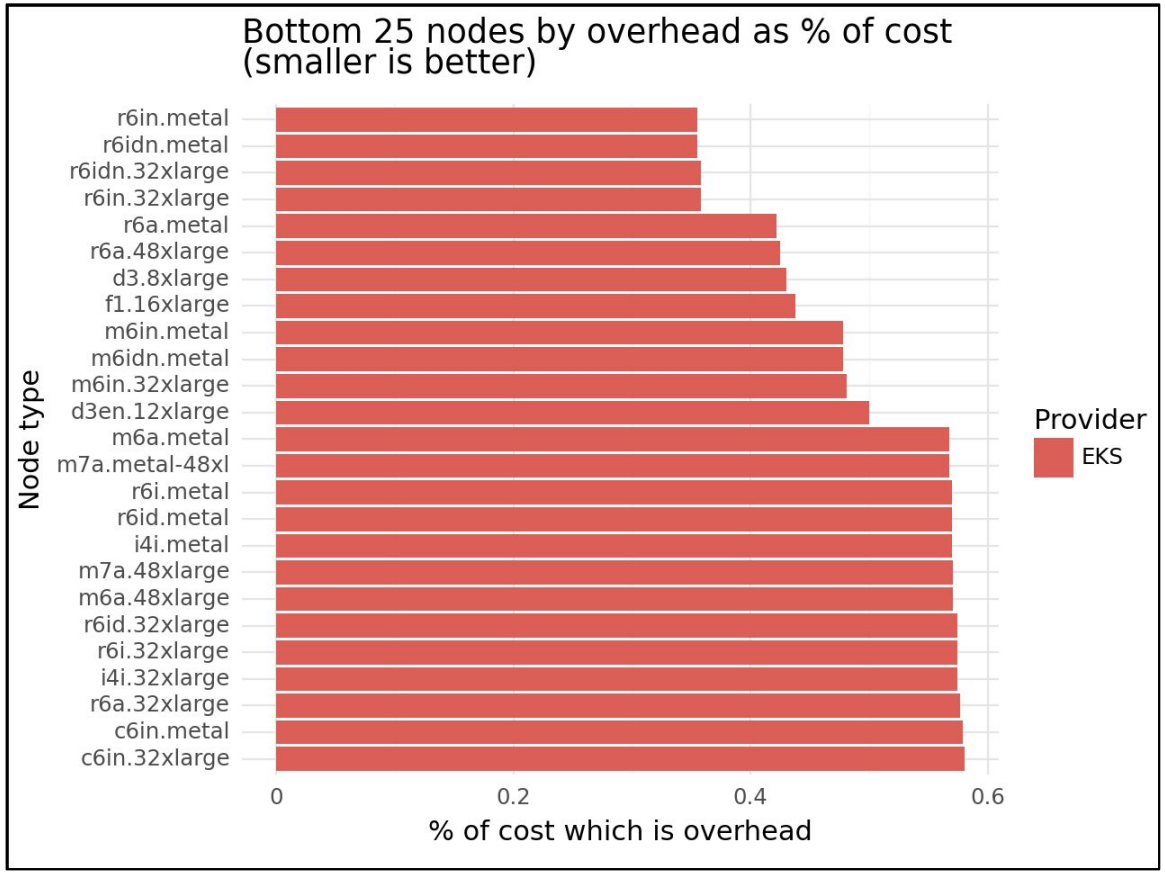

kubecost

# Overhead Study: Highest




kubecost

# Overhead Study: Lowest




kubecost

# Overhead Study: Key Takeaways

- **The larger the node, the lower the overhead**
  - Overhead converges toward 0% asymptotically
- Significant number of nodes with 10%+ overhead
- **Total overhead cost largely driven by RAM**
  - CPU Overhead generally 6% or less
- K8s has an interesting way of using shared core instances
- AKS nodes generally exhibit highest overhead
- EKS nodes generally exhibit lowest overhead
  - Especially for small instance sizes
- Node boot disk, GPU overhead is not supported yet

# Node Sizing Algorithm (Single)

1. Compute requirements for a given k8s cluster
   a. Max Individual pod CPU/RAM
   b. Total Pod Requirement CPU/RAM
   c. DaemonSet CPU/RAM
   d. Max pods per node

2. For each node type a given cloud provider offers
   a. Ensure the node type can schedule the largest pod (capacity)
   b. Determine the number of nodes needed to run the workload
      i. Take any minimum availability requirements into account

```
cpuCoeff = math.Ceil(vCPURequired / (VCPUsPerNode - daemonSetVCPUs))
ramCoeff = math.Ceil(gbRAMRequired / (RAMGBPerNode - daemonSetRAMGB))
count = math.Max(cpuCoeff, ramCoeff)
```

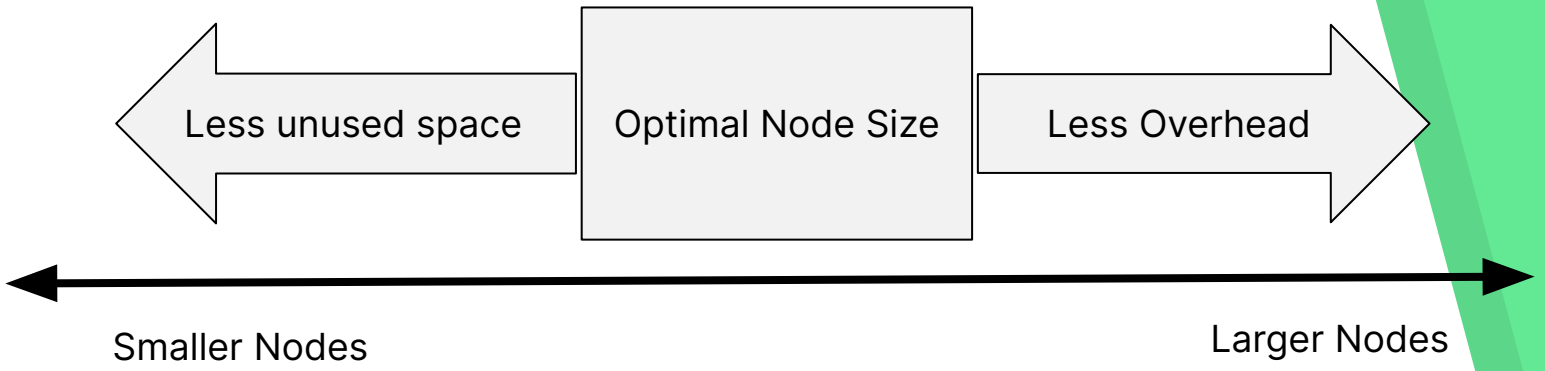
   c. Compute total cost

```
totalCost = count * pricePerNode
```

3. Select node size with lowest total cost to run cluster workload

kubecost

# Node Sizing: Challenges




- Going to an 800% larger node for 10% overhead savings doesn't make sense
  - *Unless capacity can be used*
  - More smaller nodes consolidated to fewer larger nodes?
- A few % savings can make huge impact on bottom line
- Might be worthwhile to go larger nodes to have ready space for scaling
  - Extra capacity that is usable > capacity wasted in overhead

kubecost

# How Kubecost Sizes Nodes (Overhead edition)

1. Obtain trend line equation of overhead vs resource size
2. Compute requirements for a given k8s cluster
   - Max Individual pod CPU/RAM
   - Total Pod Requirement CPU/RAM
   - DaemonSet CPU/RAM
3. For each node type a given cloud provider offers
   - Compute resource discount

```
overheadRAM = overheadRAM(resourceRAM) * overheadPenalty
overheadCPU = overheadCPU(resourceCPU) * overheadPenalty
```

   - Reduce node resources by overhead
   - Ensure the node type can schedule the largest pod (allocatable)
   - As before, determine the minimum number of nodes needed to run all pods
     i. Incorporate availability concerns here
   - Compute total cost
4. Select node size with lowest total cost to run cluster workload
5. Surface total overhead of cluster with specified node size

kubecost

# Closing Thoughts

- Trying to raise awareness of node overhead
  - Nothing is free
- Can sap 50%+ (more typically 10-20%) of compute
- Fewer larger nodes can result in more compute usable
  - Subject to availability requirements
- Use revised node sizing algorithm to factor in overhead when picking a node
- Kubecost Enterprise
  - Does this continuously, automatically
  - Renders recommendations

# Thank You!

Come talk with Michael and Alex at Kubecost's booth M10

Visit OpenCost maintainers at F34 Kiosk in the Project Pavilion

*ameijer@kubecost.com* *michael@kubecost.com*

kubecost