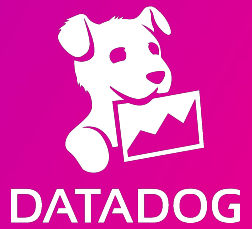
Scaling Smarter Not Harder: How Extending Cluster Autoscaler Saves Millions

KubeCon + CloudNativeCon, London 2025



Rahul Rangith

Software Engineer, Datadog

Ben Hinthorne

Software Engineer, Datadog

DATADOG

Agenda

01 Autoscaling at Datadog

02 Cluster Autoscaler and Expanders

03 Identifying Optimal Instance Types

04 Scaling Optimal Instance Types

Datadog Infrastructure

Kubernetes from scratch in a multi cloud environment

Dozens of clusters

10,000s of nodes

100,000s of pods

Serving trillions of data points per hour for over 30k customers

Compute - Autoscaling

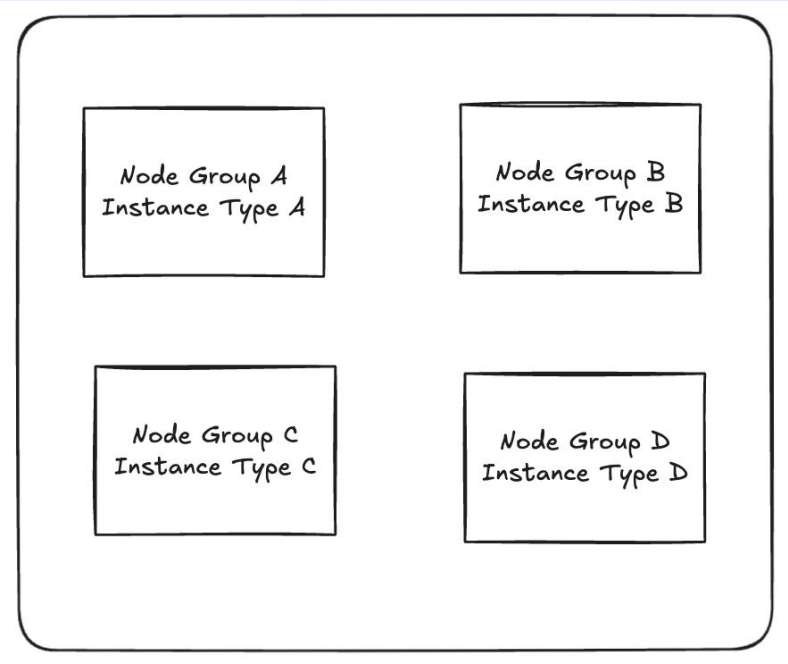
Manage node infrastructure for product teams

Scheduling and scaling efficiency

Bin packing and cost optimizations

Node Group Sets

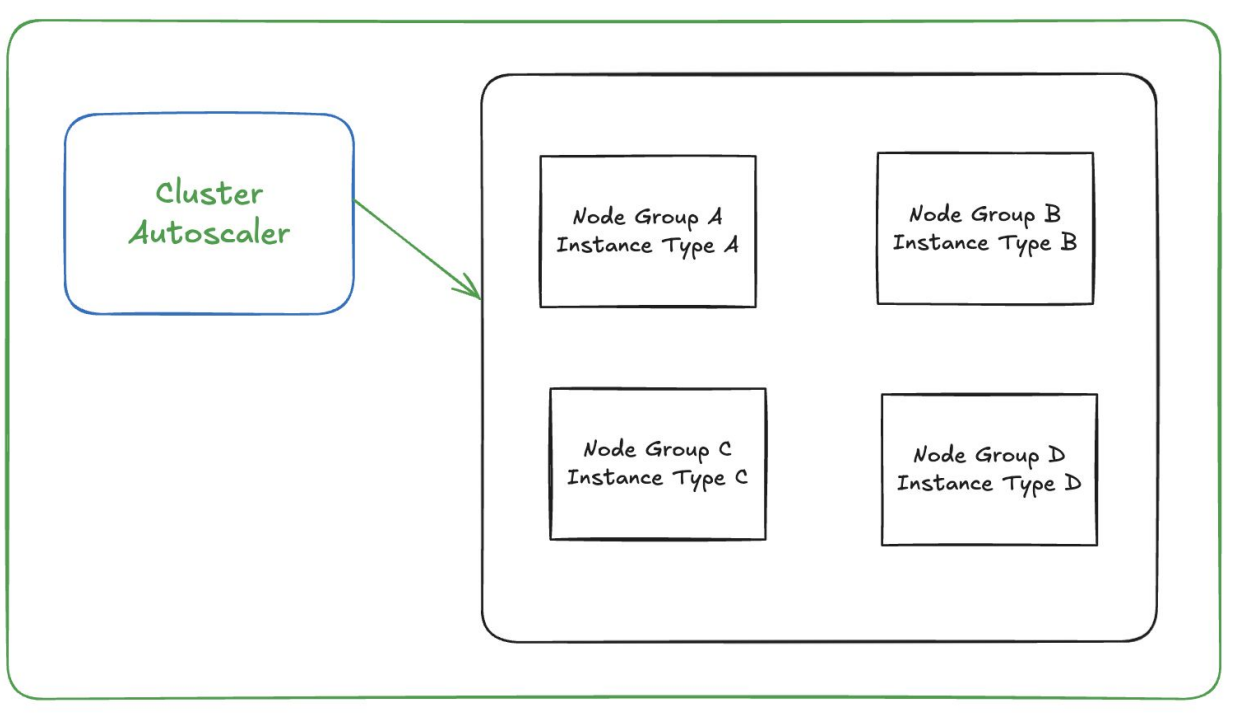
- A **Node Group** is a cloud provider agnostic representation for an ASG, MIG, etc.
- A **Node Group Set** is a set of node groups under the same scheduling domain.
  - Easier to onboard and manage many users
  - Easier to provide multiple instance type options



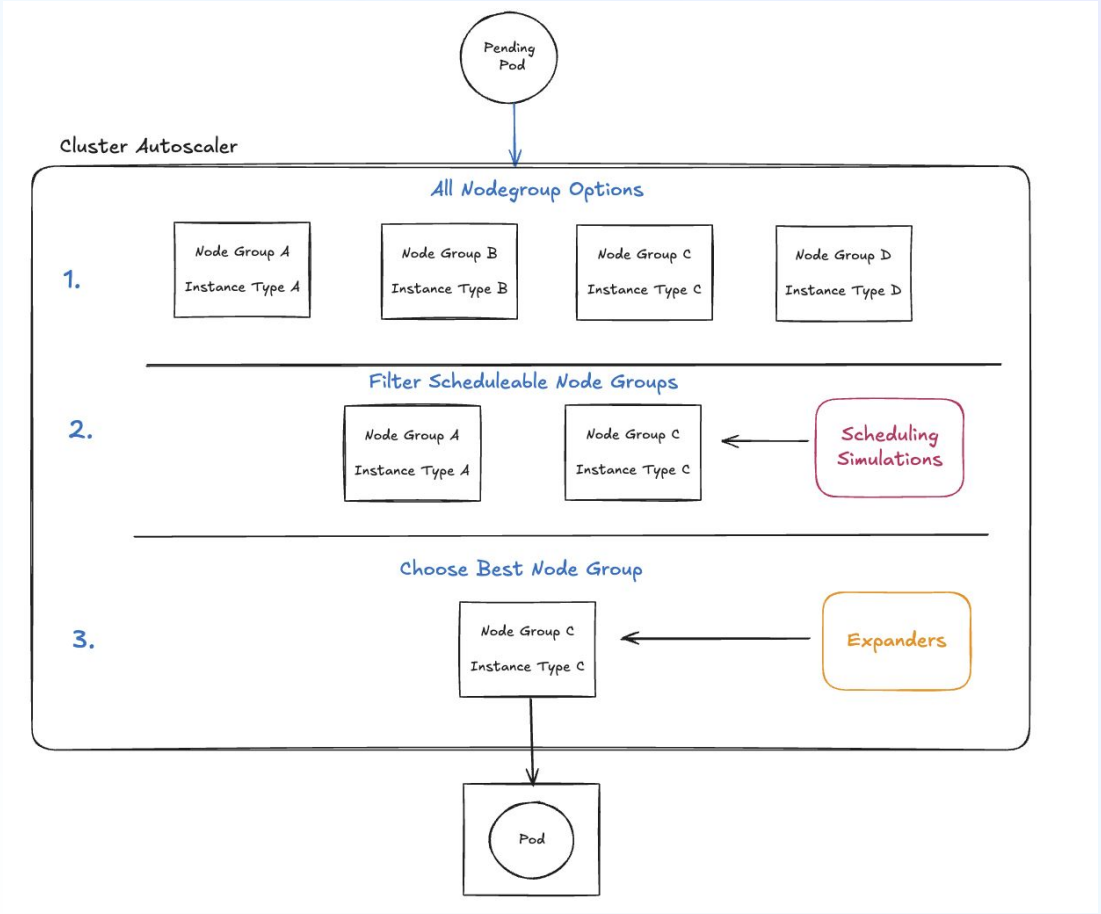
Cluster Autoscaler

We use **Cluster Autoscaler** for node autoscaling opposed to alternatives, primarily because of:

- Multi cloud support
- Happy with the operational experience



Cluster Autoscaler: Instance Type Selection



All Nodegroup Options

Filter Schedulable Node Groups

Expanders pick the best option

DATADOG

Cluster Autoscaler: Built in Expanders

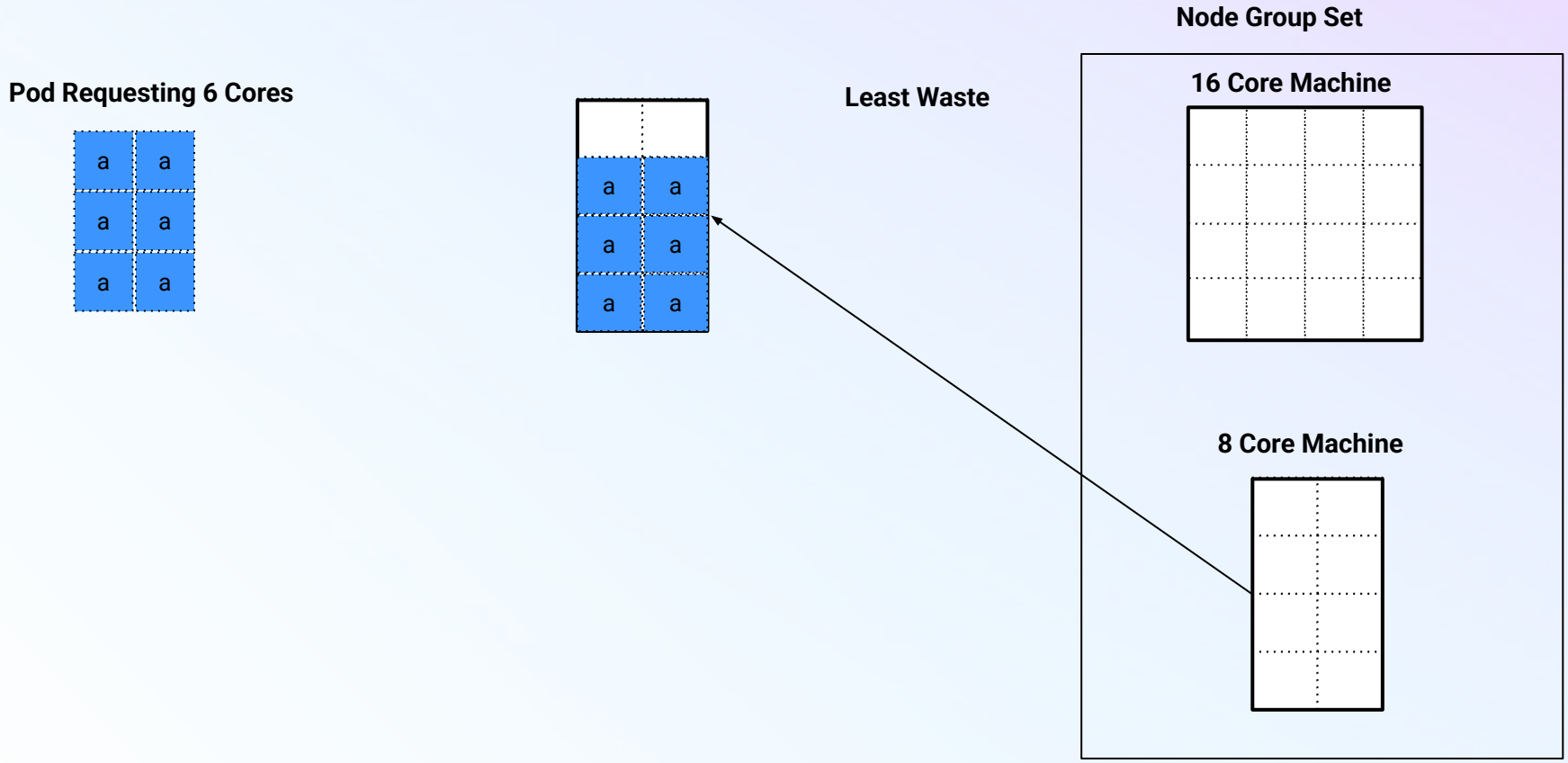
Random: Select a random node group

Least Waste: Select the node group that will waste the fewest resources

Price: Select the cheapest node group option

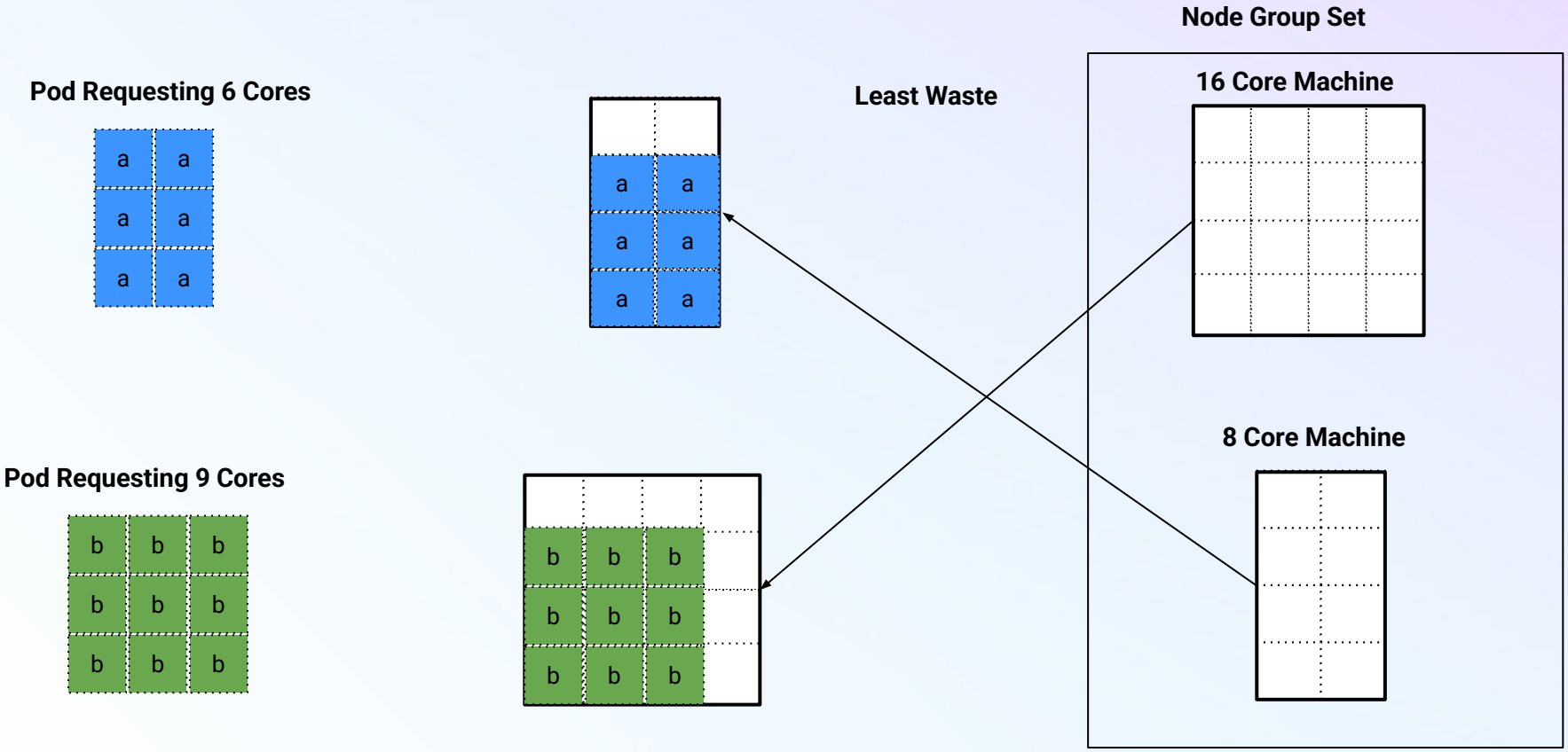
Priority: Select node groups based on user assigned priorities

Case Study: Least Waste Expander



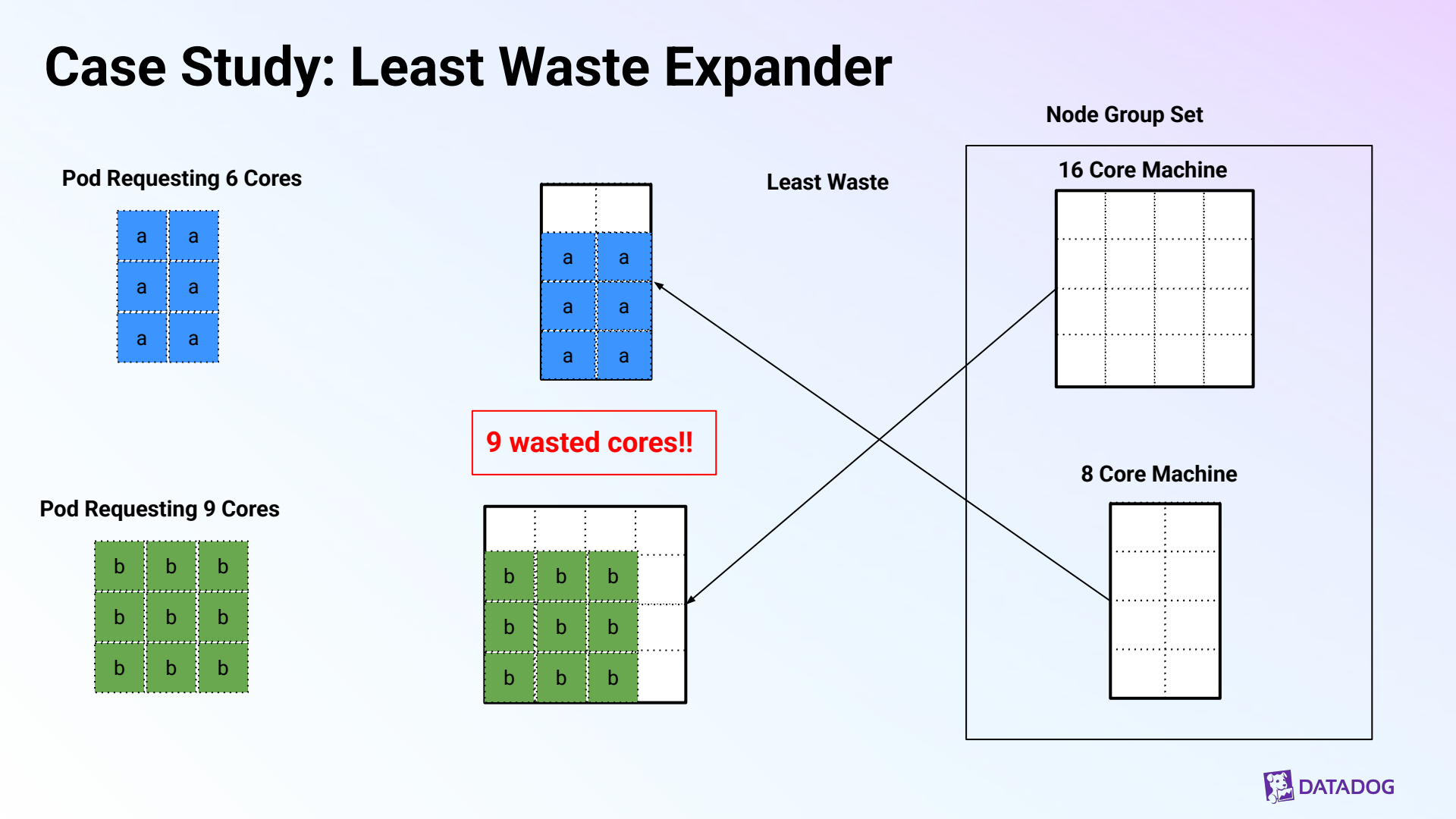
DATADOG

Case Study: Least Waste Expander



DATADOG

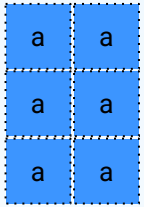
Case Study: Least Waste Expander



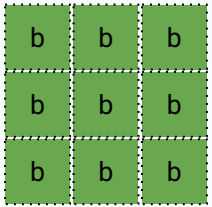
DATADOG

Case Study: Least Waste Expander

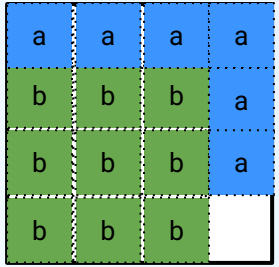
Pod Requesting 6 Cores



Pod Requesting 9 Cores



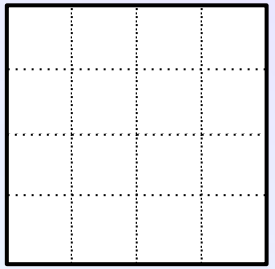
1 wasted core!!



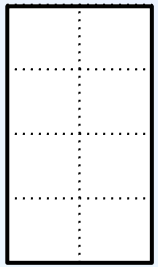
Is this theoretical optimization applicable?

Node Group Set

16 Core Machine



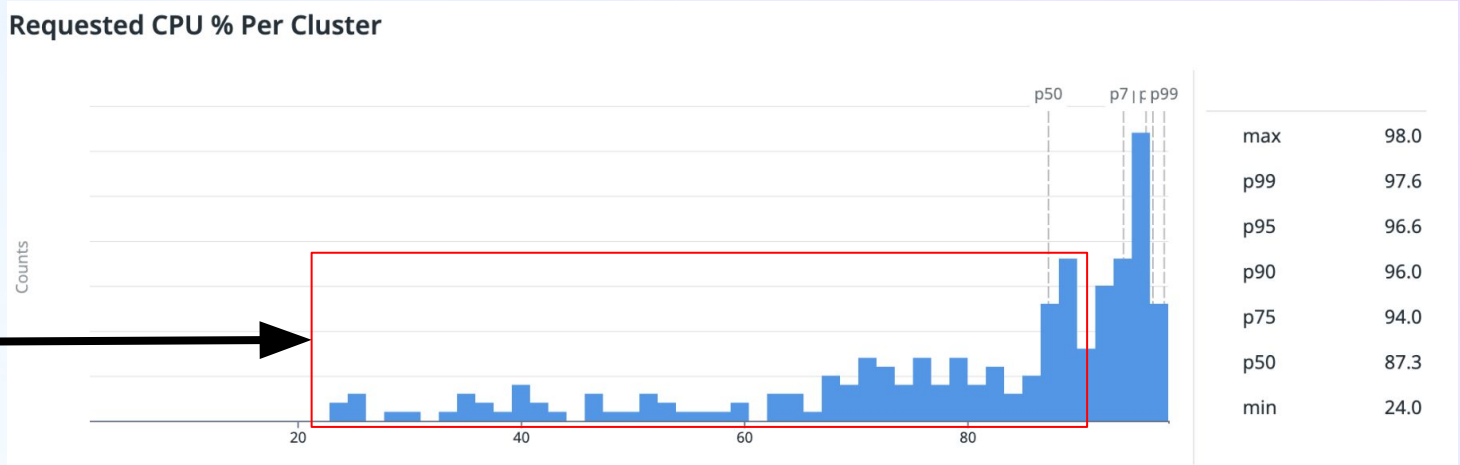
8 Core Machine



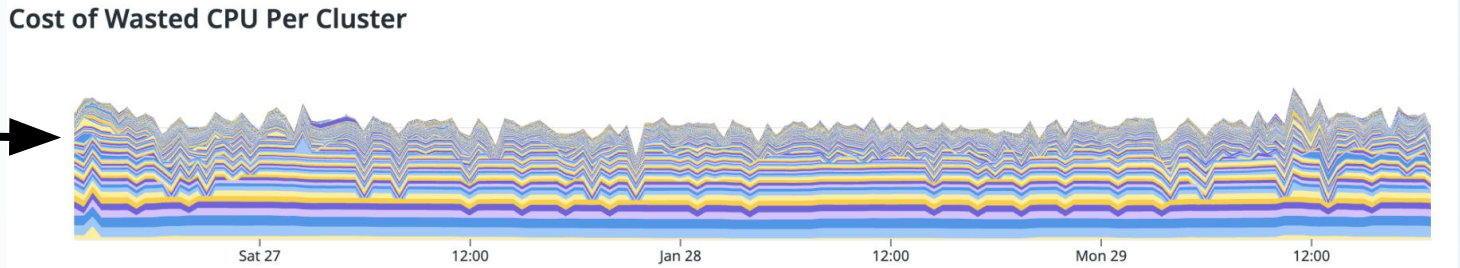
DATADOG

In Practice: Least Waste Expander

Clear opportunity for bin packing improvements

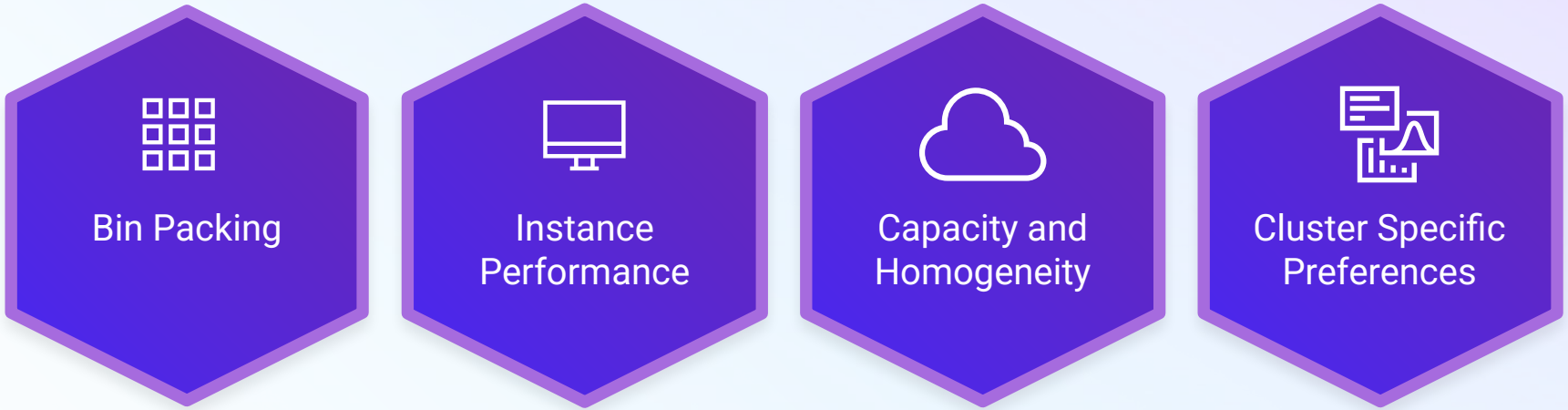


Translates to significant cost impact

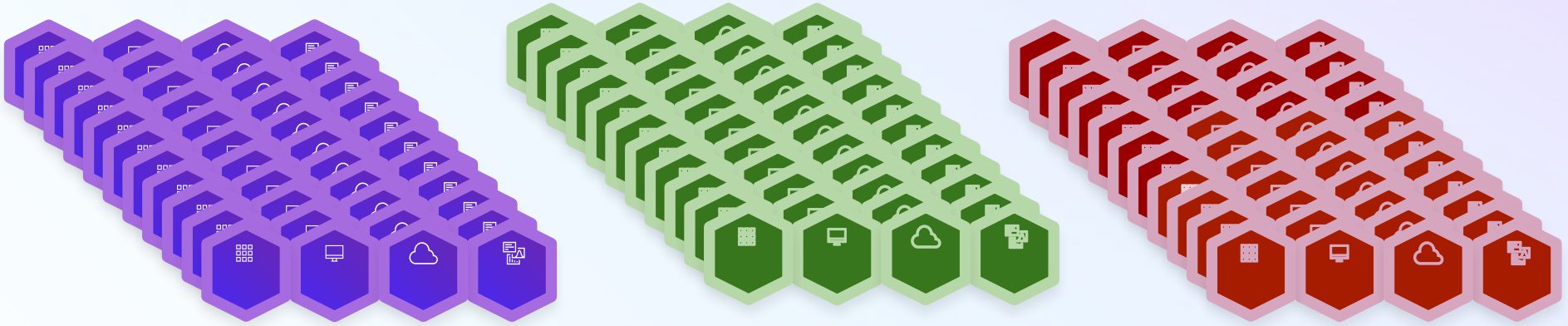


DATADOG

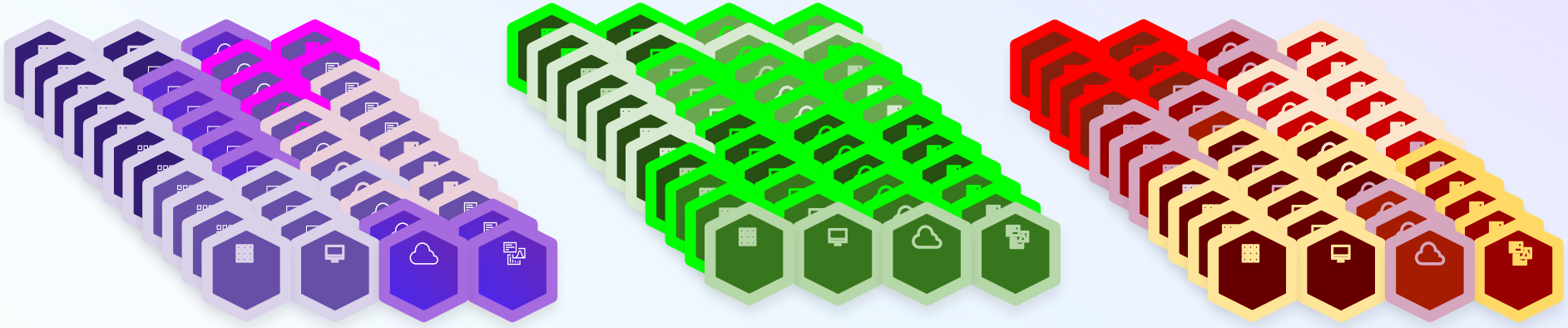
Instance Selection: Beyond Least Waste



Instance Selection: At Scale



Instance Selection: At Scale



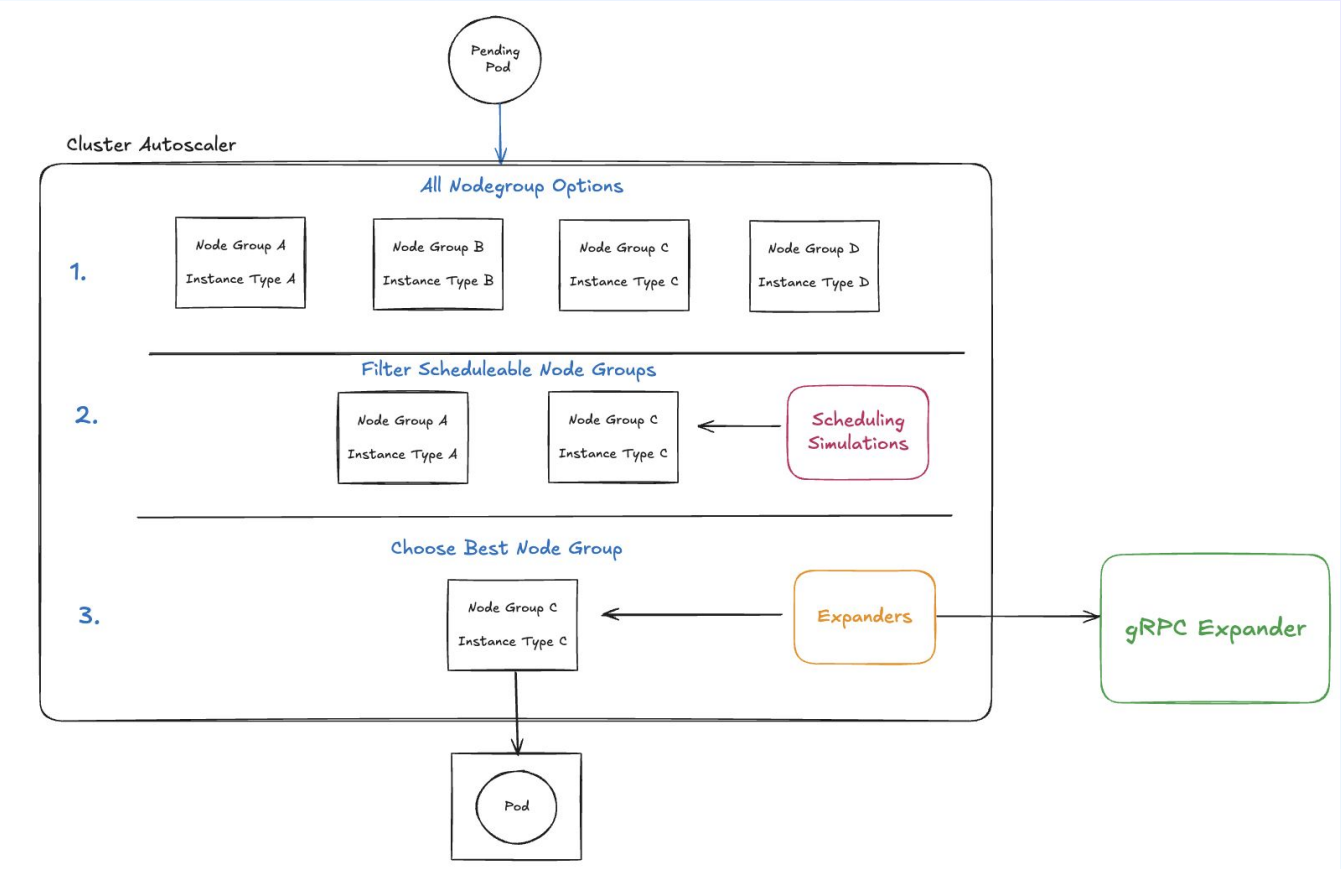
DATADOG

There is no “one expander fits all” solution.

There is no “one expander fits all” solution.

Unless…

Cluster Autoscaler: gRPC Expander



Cluster Autoscaler supports a **gRPC Expander!**

Allows users to build a service to make custom node group decisions!

DATADOG

gRPC Expander: Upstream Example

```go
// BestOptions method filters out the best options of all options passed from the gRPC Client in CA, according to the defined strategy.
func (ServerImpl *ExpanderServerImpl) BestOptions(ctx context.Context, req *protos.BestOptionsRequest) (*protos.BestOptionsResponse, error) {
        opts := req.GetOptions()
        log.Printf("Received BestOption Request with %v options", len(opts))

        // This strategy simply chooses the Option with the longest NodeGroupID name, but can be replaced with any arbitrary logic
        longest := 0
        var choice *protos.Option
        for _, opt := range opts {
                log.Println(opt.NodeGroupId)
                if len(opt.NodeGroupId) > longest {
                        choice = opt
                }
        }

        log.Print("returned bestOptions with option: ", choice.NodeGroupId)

        // Return just one option for now
        return &protos.BestOptionsResponse{
                Options: []*protos.Option{choice},
        }, nil
}
```

https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/expander/grpcplugin/example/fake_grpc_server.go

DATADOG

What we know so far

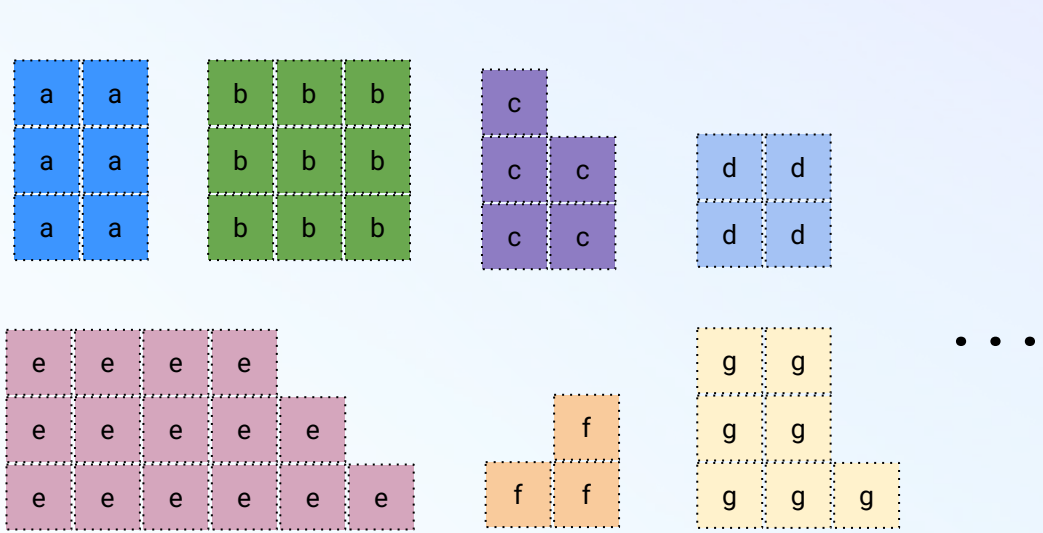
1. We have an opportunity to improve our fleet's efficiency via instance type selection.
2. The **gRPC Expander** gives us ultimate flexibility in instance type selection.

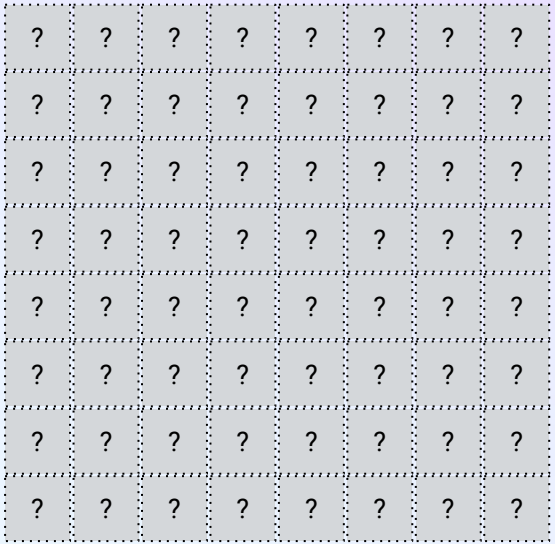
Our Goals

1. **Identify** the best instance types
2. **Scale** the best instance types

Instance Type Selection: Simplified Criteria

Cost: Back to the Bin packing Problem





How can we binpack 1000s of unique workloads?

DATADOG

Cost: Scheduling Simulations

Instance Type Advisor

Set of Pods

- Selected via a NodeSelector, e.g. “get all pods running on nodes that have label foo=bar”

Instance Catalog

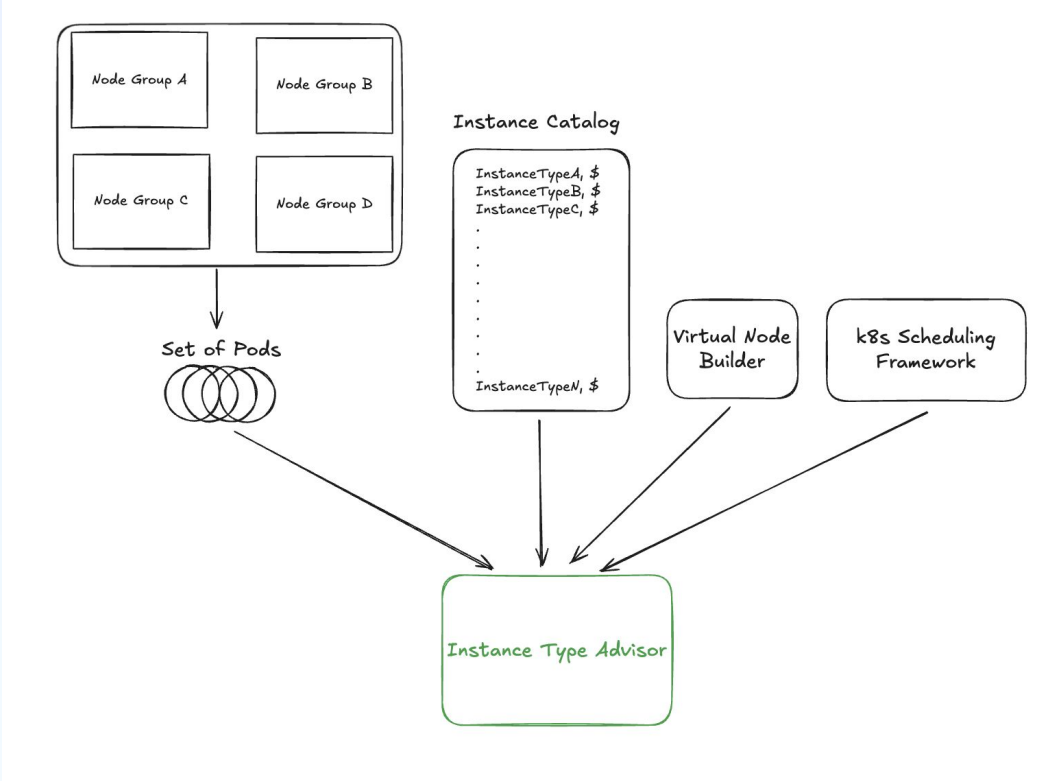
- Catalog of all possible instance types and their specs, including cost per hour, cpu/mem capacity, etc.

Virtual Node Builder

- Interface to build virtual nodes with desired configuration

Scheduling Framework

- Upstream k8s scheduling framework, or other scheduler plugins possible



DATADOG

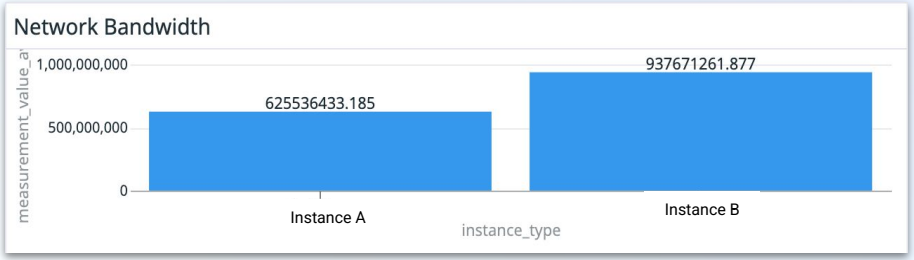
Cost: Scheduling Simulations

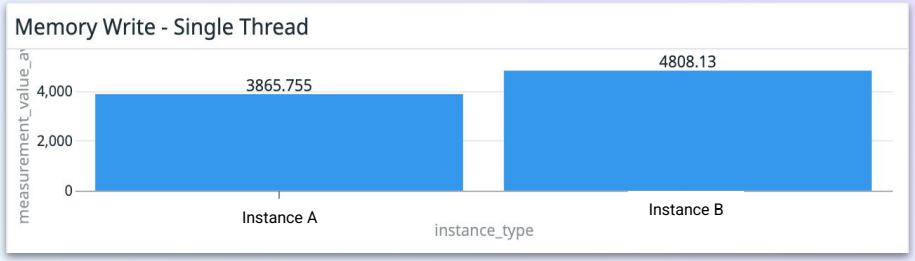
Instance Type	Hourly Cost	Number of Nodes	Requested CPU %	Requested Memory %
m6a.8xlarge (current)	5555.55	152	62	76
r6a.4xlarge (best)	4444.44	160	95	62
c6a.8xlarge	6655.55	180	54	85

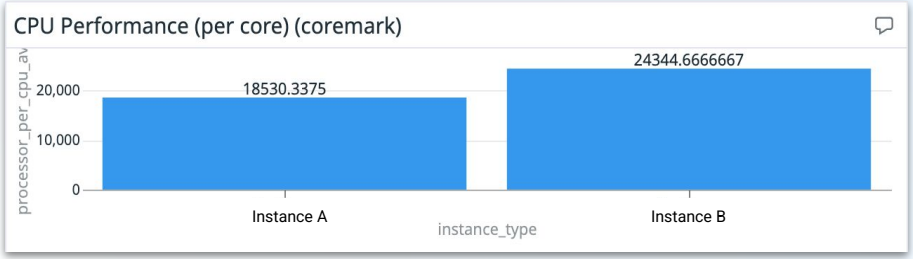
The Instance Type Advisor results allows us to compare the cost and bin packing efficiency for each instance type for our specific set of pods.

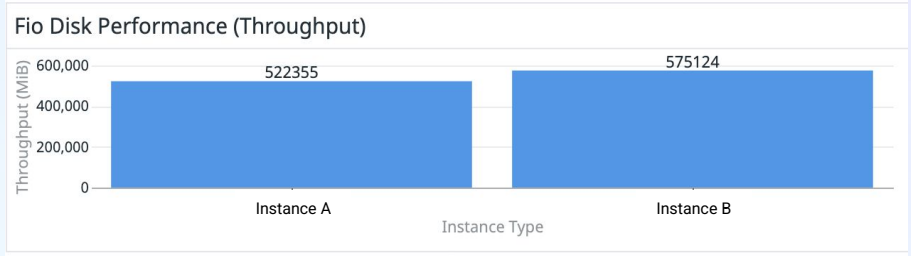
We run automated scheduling simulations in every cluster, and store the results in a CRD in the cluster itself.

Performance: Benchmarks





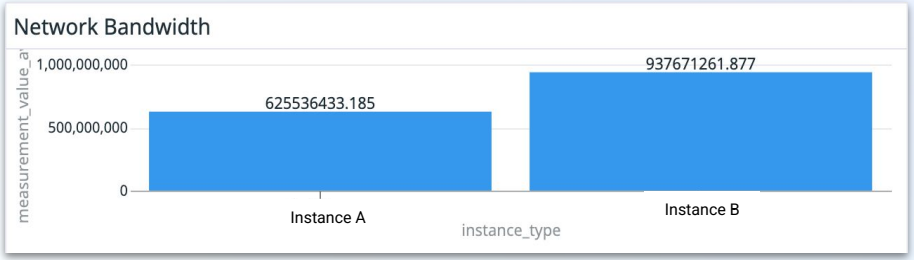


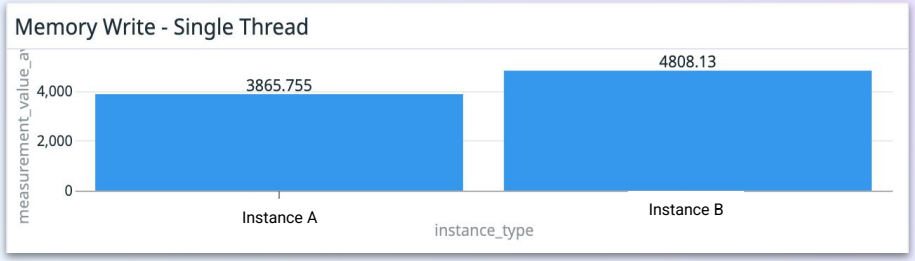


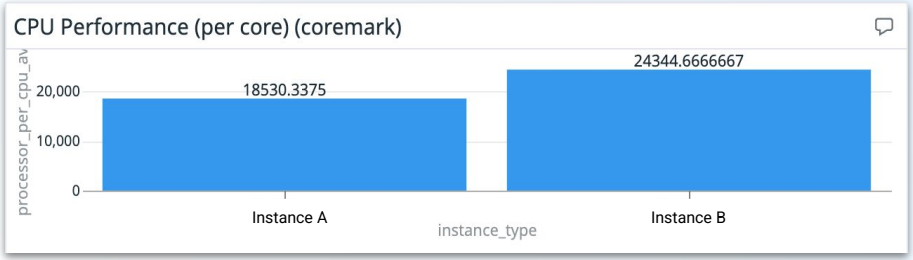
Performance Benchmarks for network, cpu, memory, and storage for us to weigh cost vs. performance for every instance type.

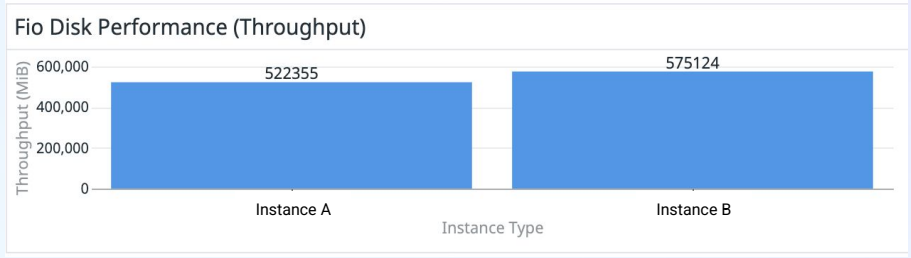
DATADOG

Performance: Benchmarks









Performance Benchmarks for network, cpu, memory, and storage for us to weigh cost vs. performance for every instance type.

“Instance B is 20% more expensive than Instance A, but 30% more performant”.

DATADOG

Reliability: Instance Attributes

We have the flexibility to express various instance type reliability preferences via **CEL Go selectors.**

This allows us to specify preferences such as:

- Capacity per environment
- Instance size ranges
- Performance preferences

```
message Environment {
  string cluster = 1;
  string provider = 2;
  string region = 3;
  string datacenter = 4;
  string environment = 5;
  string lineage = 6;
  string flavor = 7;
  int64 maxnodes = 8;
  bool edge = 9;
  bool experimental = 10;
}
```

```
message InstanceType{
  string name = 1;
  string provider = 2;
  string region = 3;
  string arch = 4;
  string family = 5;
  int64 cpus = 6;
  double memory = 7;
  int64 generation = 8;
  double cost = 9;
  int64 bandwidth = 10;
}
```

DATADOG

Reliability: Instance Attributes

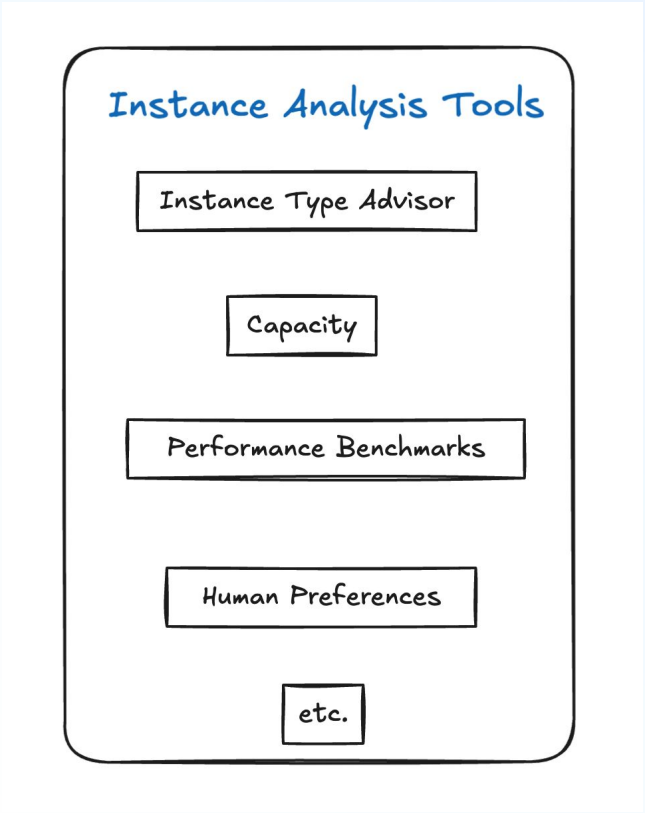
- instance.family in [“m6g”, “r6g”, “c6g”] && instance.cpus >= 8 && instance.cpus <= 32
- instance.family in [“m5n”, “r5n”, “c5n”] && env.edge
- Instance.generation==8 && env.cluster in [“cluster-foo”, “cluster-bar”]
- instance.family == “fancy-new-type” && env.experimental

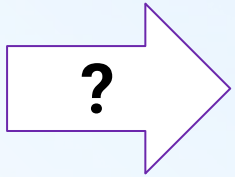
Instance Type Selection: Simplified Criteria

Our Goals

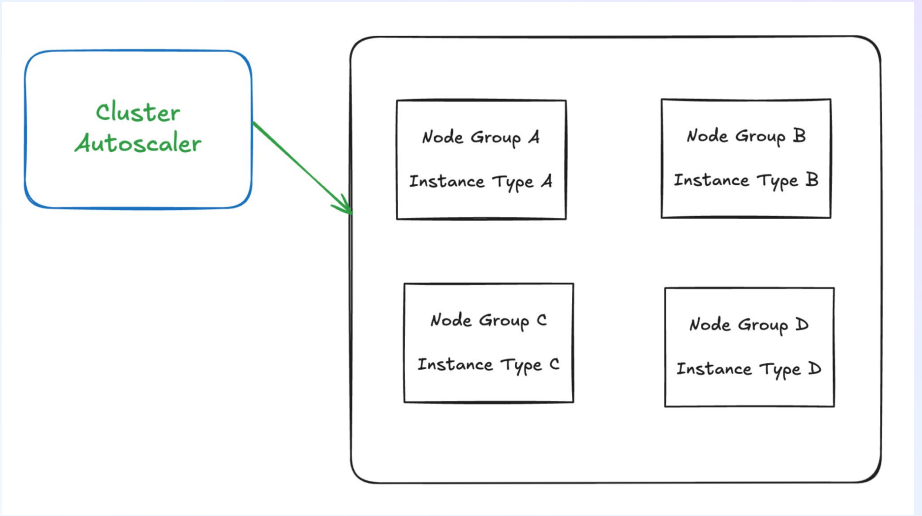
1. **Identify** the best instance types ✅
2. **Scale** the best instance types

Autoscaling Ecosystem



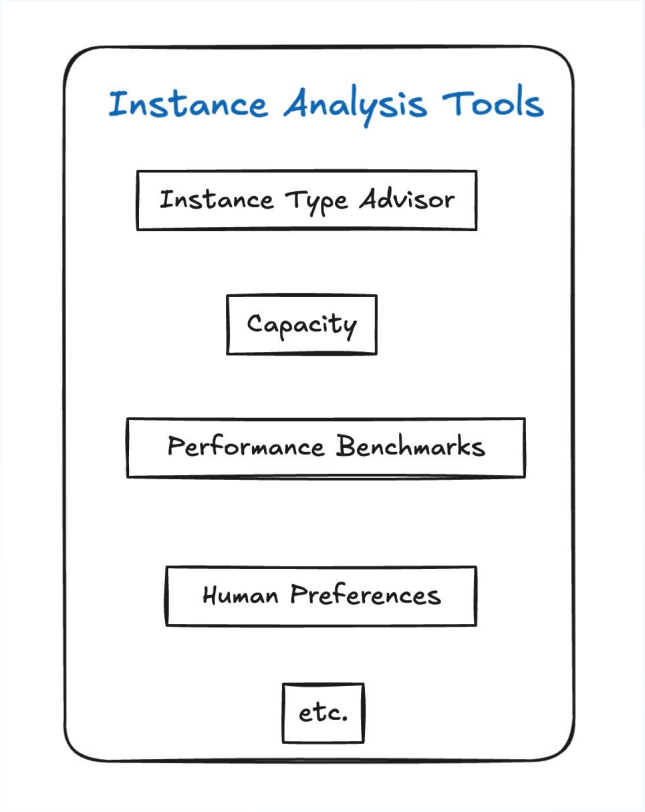


How does the Cluster Autoscaler know what the best instance type is?



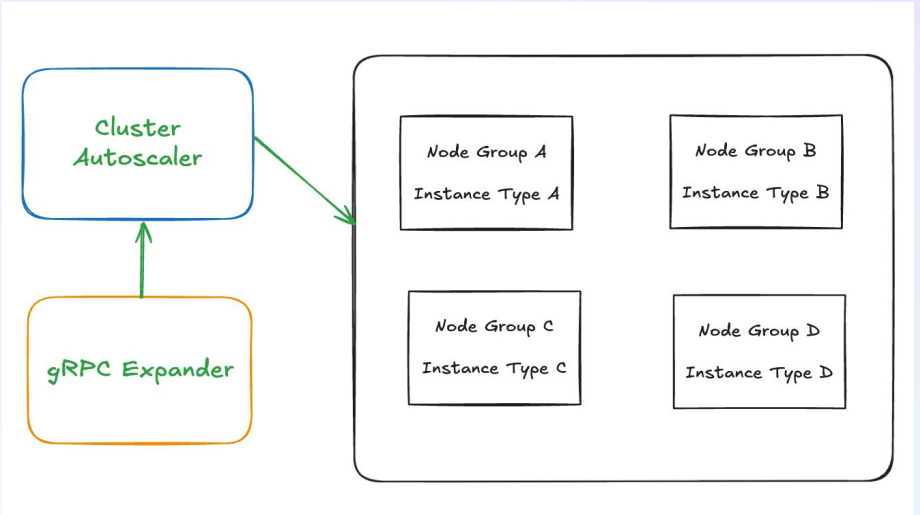
DATADOG

Autoscaling Ecosystem



?

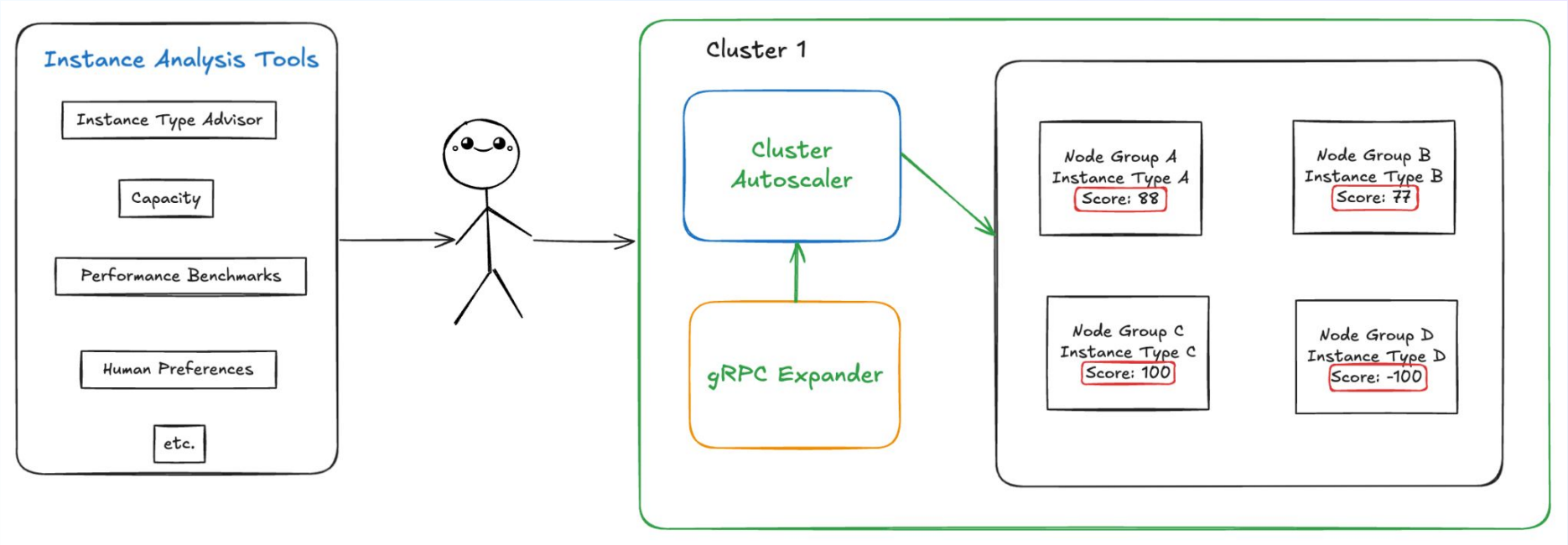
How does the gRPC Expander know what the best instance type is?



DATADOG

Autoscaling Ecosystem

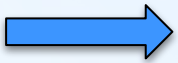
A human can translate the preferences into something the gRPC Expander can read



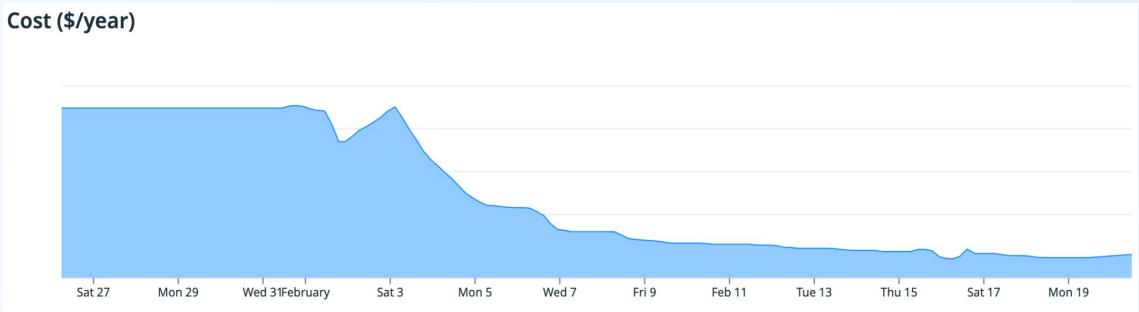
Results

Apply: transition to suggested type

m6g.8xlarge -> r6g.8xlarge

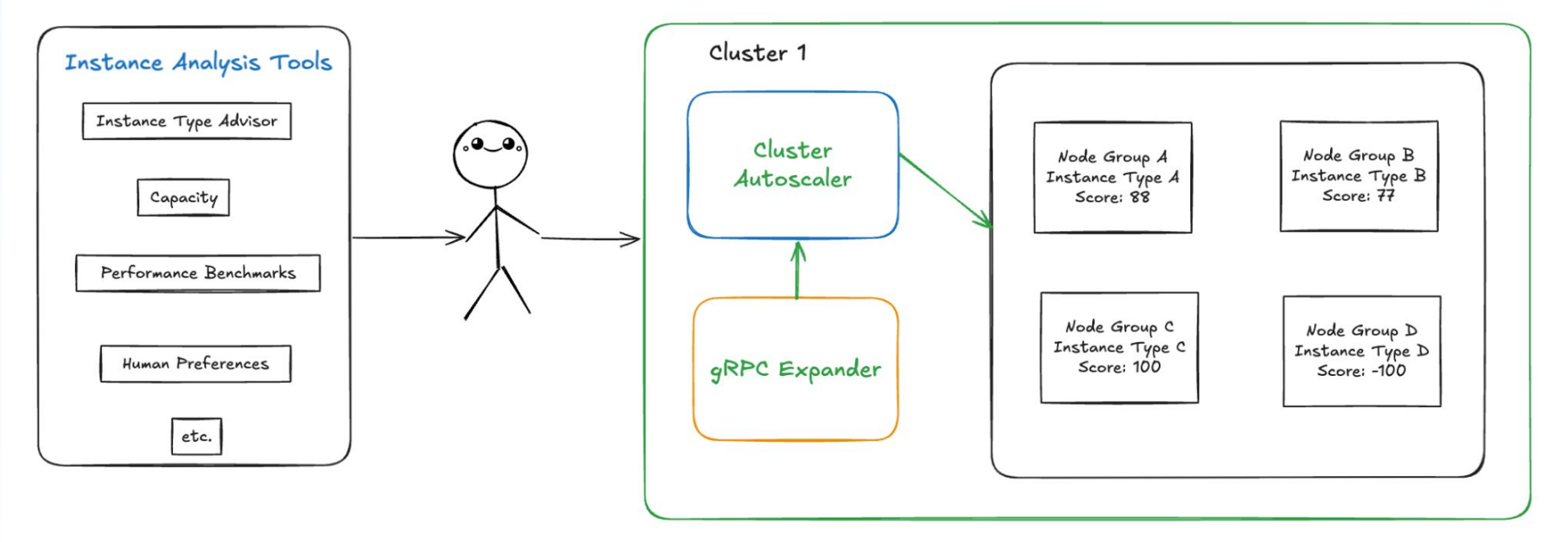


Save: we reduced costs by over 60% in a single cluster



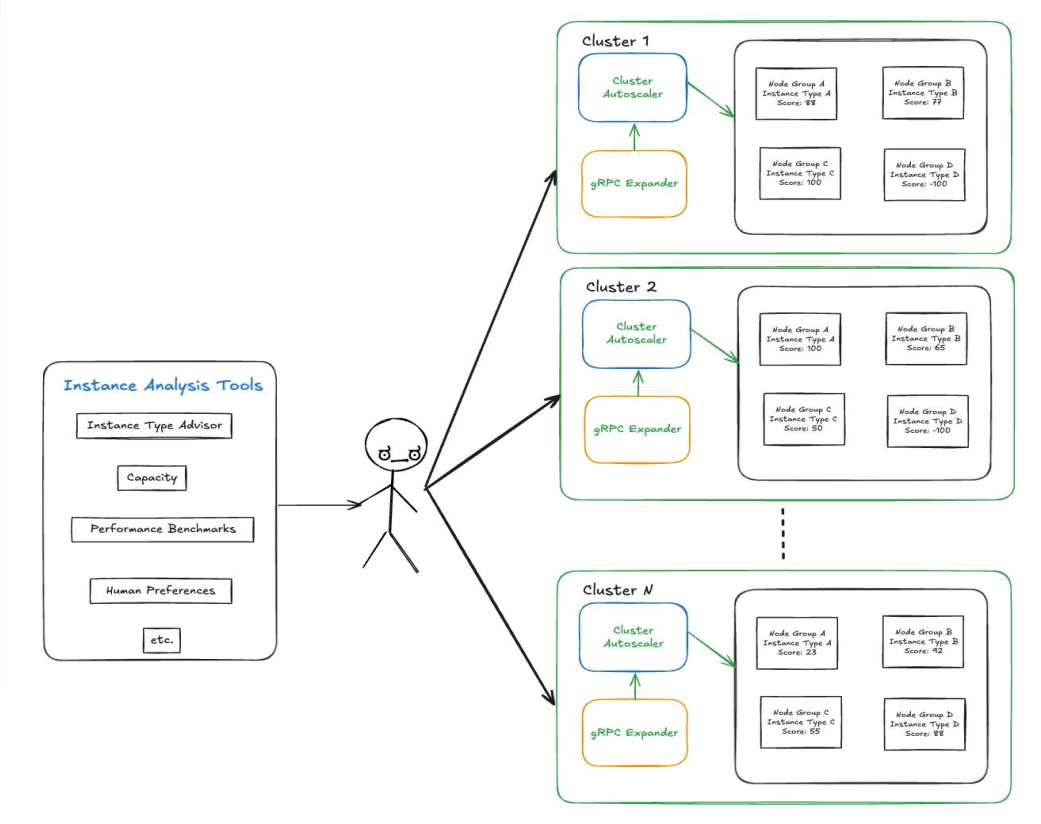
DATADOG

Autoscaling Ecosystem



DATADOG

Autoscaling Ecosystem



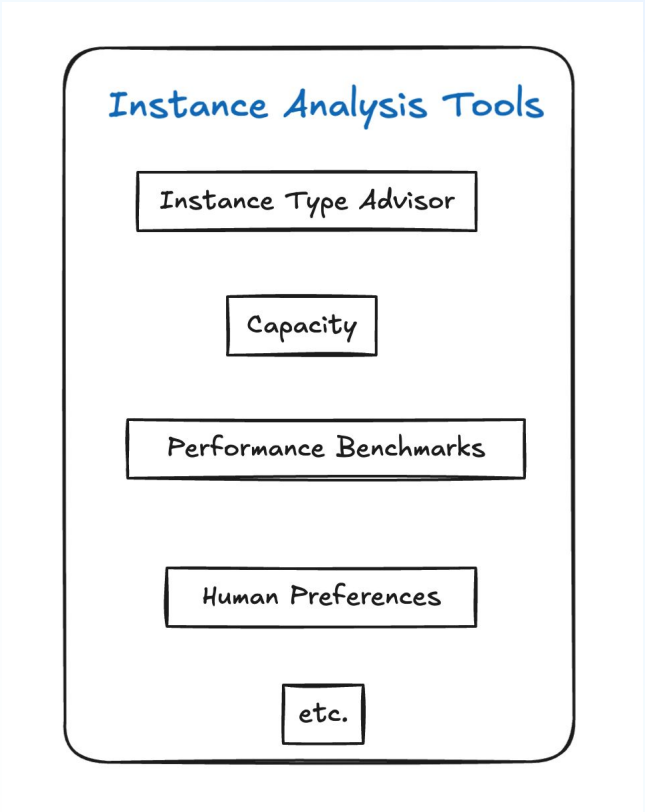
This does not scale with many clusters

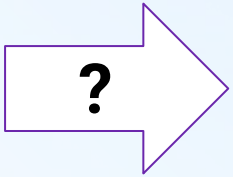
Scores can frequently change

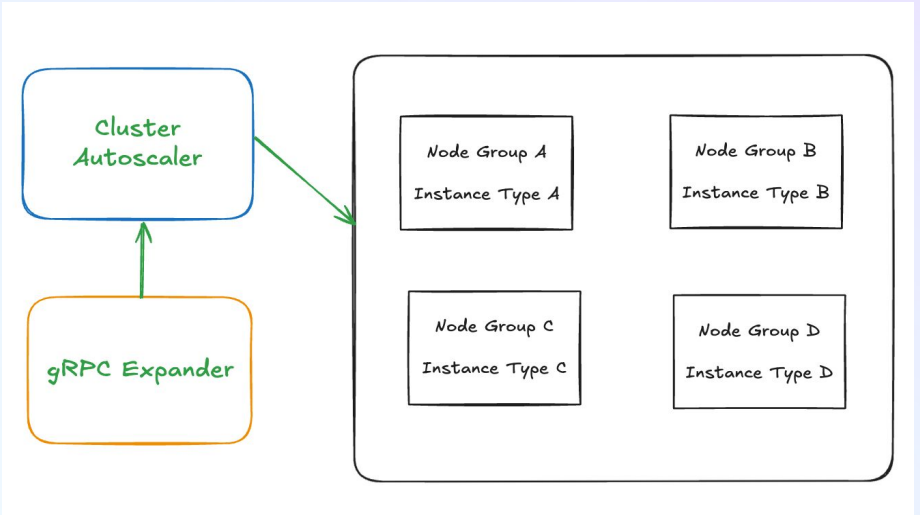
DATADOG

Autoscaling Ecosystem

We need a scalable and dynamic way to know the best instance type

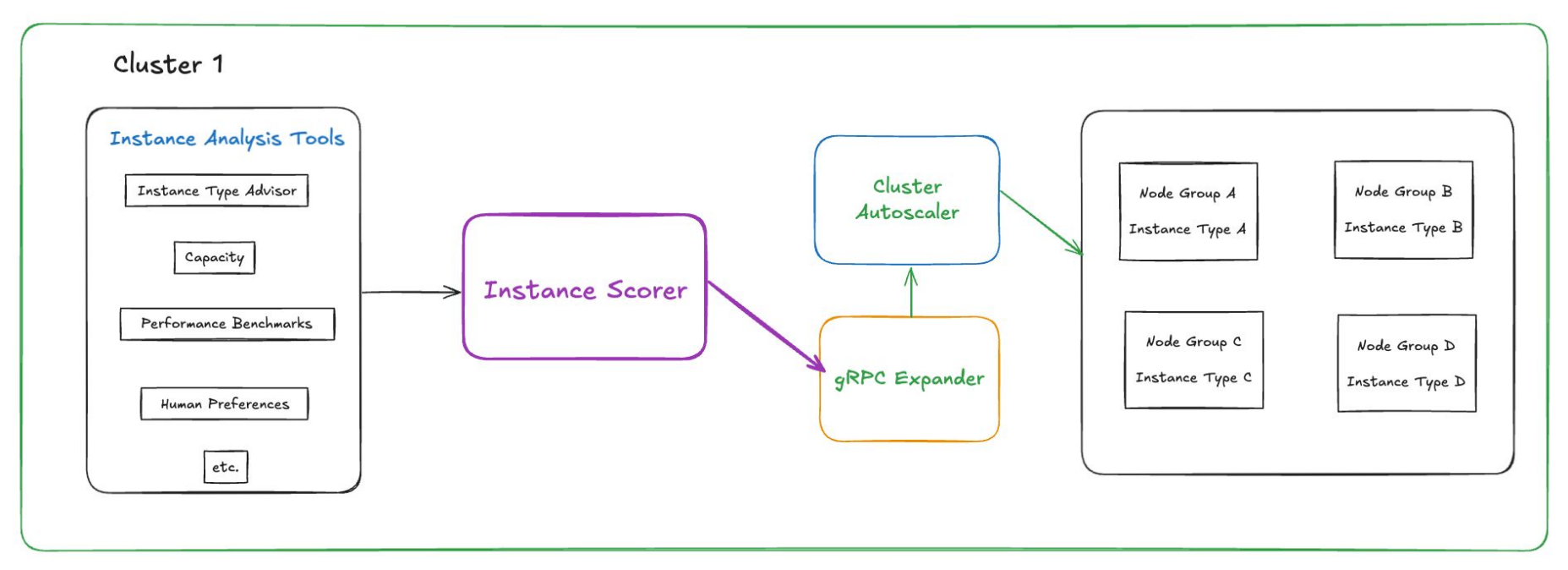






DATADOG

Dynamic Scaling Ecosystem



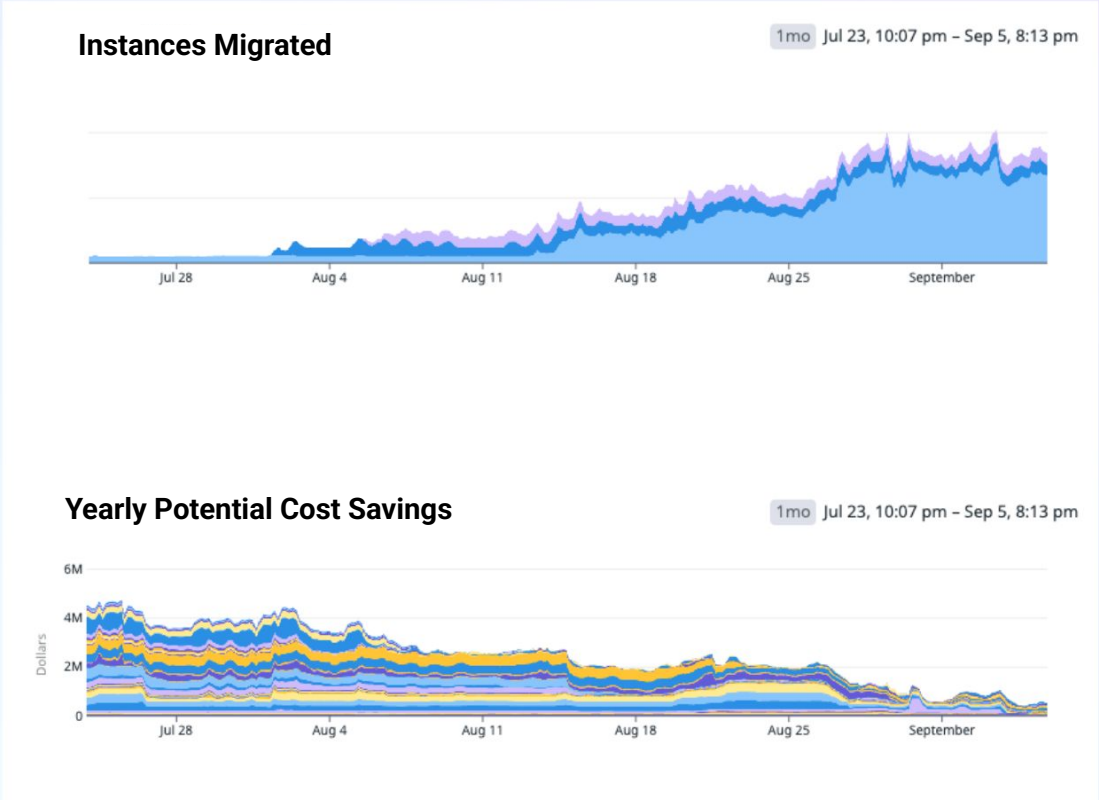
DATADOG

Results

- Estimated savings of $4M/year
- Migration done automatically across dozens of clusters

Potential Cost Savings

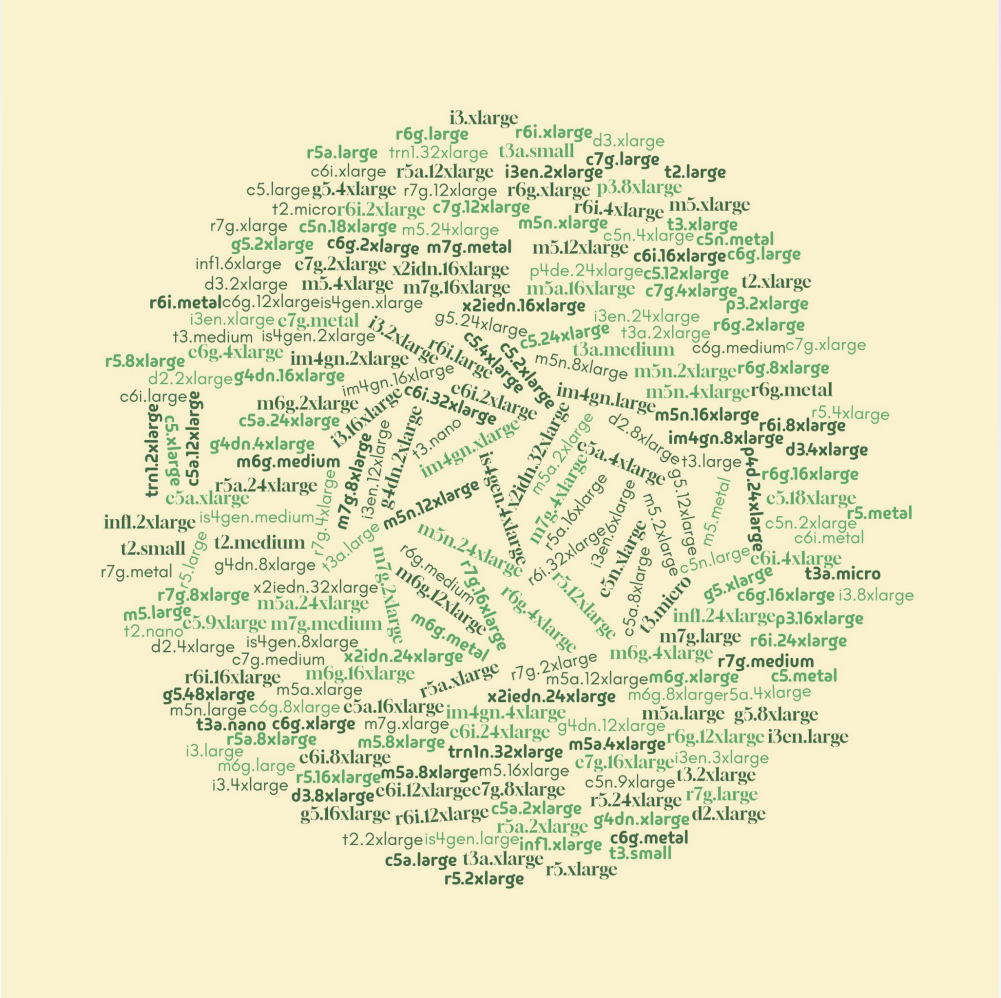
= Current Cost - Optimal Cost



DATADOG

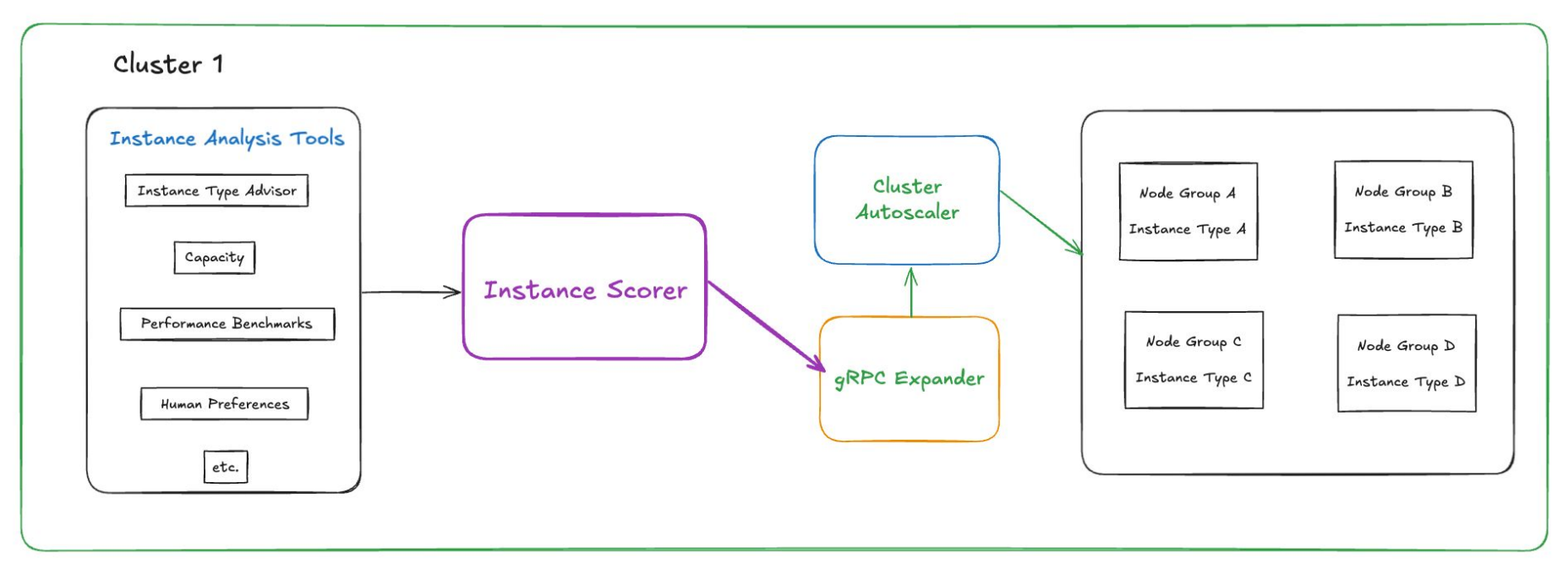
Instance Types

- Cost
- CPU
- Memory
- Storage
- Performance
- Capacity
- etc.



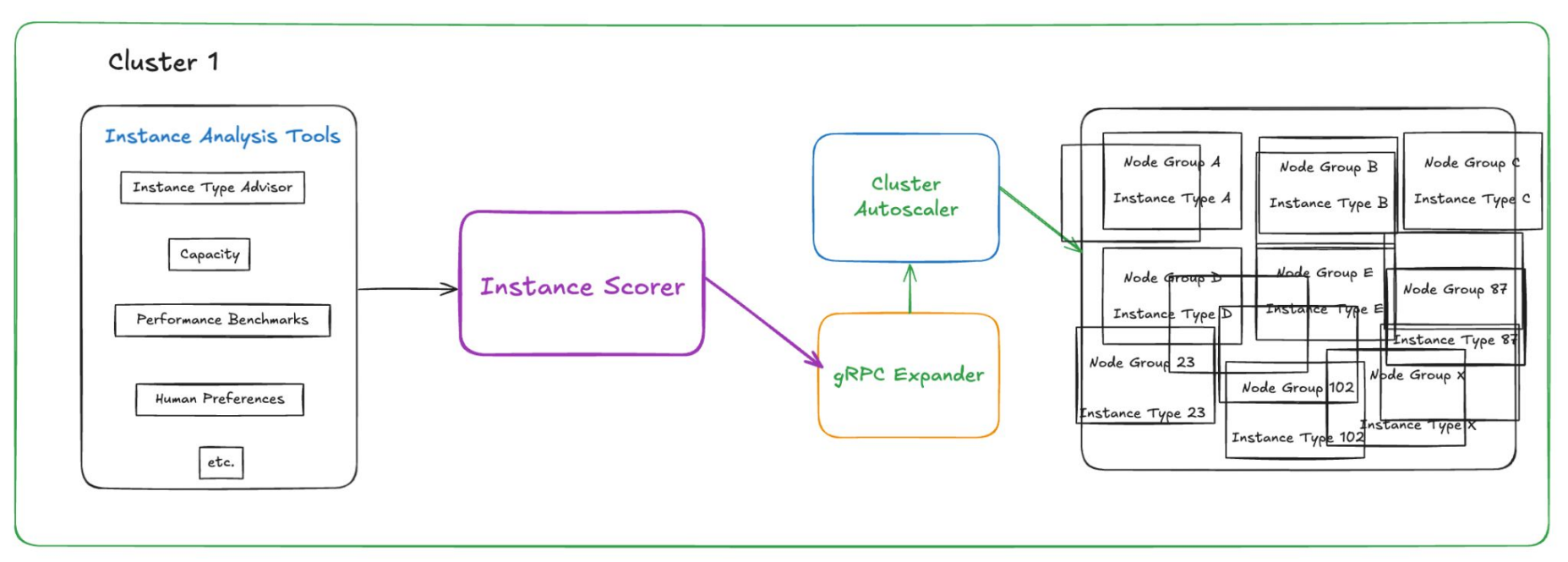
DATADOG

Instance Types



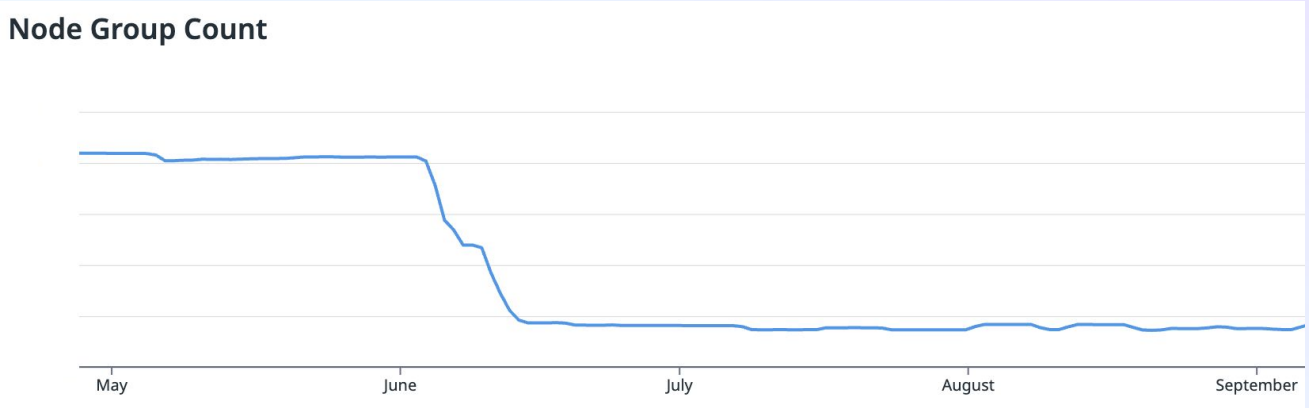
How do make sure the best node group exists?

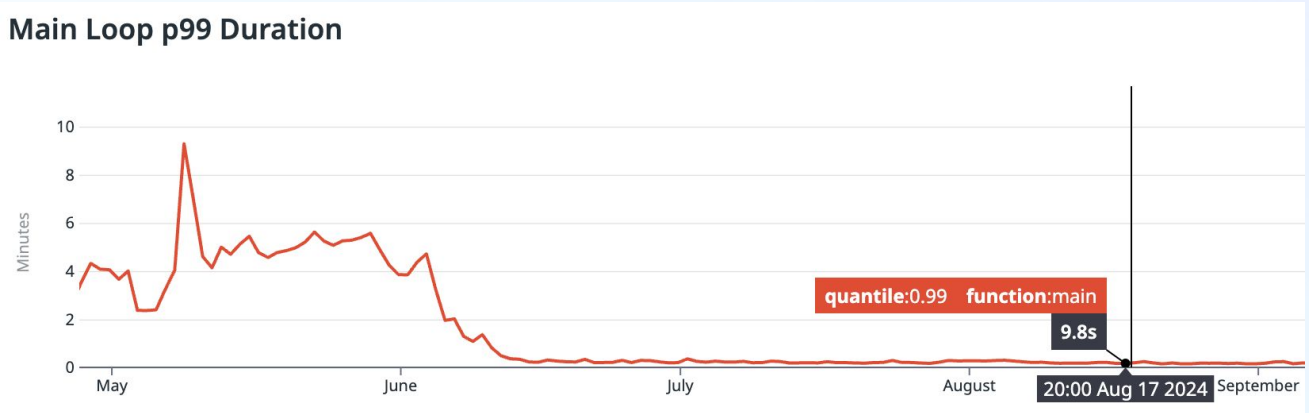
Instance Types x 100+



DATADOG

Cluster Autoscaler Performance vs # Node Groups



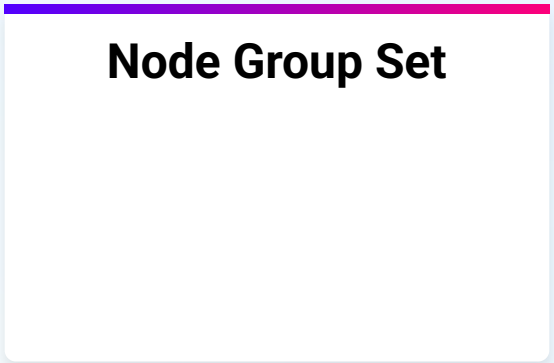


DATADOG

Platform Requirements

Our platform must be able to:

- **Scaleup the most cost-efficient instance type**



Platform Requirements

Our platform must be able to:

- **Scaleup the most cost-efficient instance type**

Node Group Set

- **Best instance type**

Platform Requirements

Our platform must be able to:

- Scaleup the most cost-efficient instance type
- **Schedule pods requesting up to 64 CPU and 256GB Memory**

Node Group Set

- Best instance type

Platform Requirements

Our platform must be able to:

- Scaleup the most cost-efficient instance type
- **Schedule pods requesting up to 64 CPU and 256GB Memory**

Node Group Set

- Best instance type
- **Best big instance type**

Platform Requirements

Our platform must be able to:

- Scaleup the most cost-efficient instance type
- Schedule pods requesting up to 64 CPU and 256GB Memory
- **Provide fallback instances in case the main instance types run out of capacity**

Node Group Set

- Best instance type
- Best big instance type

Platform Requirements

Our platform must be able to:

- Scaleup the most cost-efficient instance type
- Schedule pods requesting up to 64 CPU and 256GB Memory
- **Provide fallback instances in case the main instance types run out of capacity**

Node Group Set

- Best instance type
  - **2 fallbacks**
- Best big instance type
  - **2 big fallbacks**

Platform Requirements

Our platform must be able to:

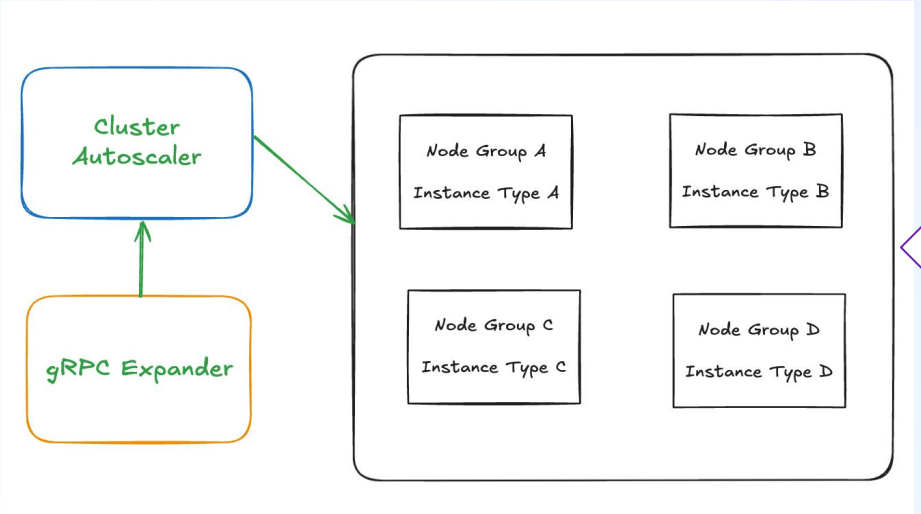
- Scaleup the most cost-efficient instance type
- Schedule pods requesting up to 64 CPU and 256GB Memory
- Provide fallback instances in case the main instance types run out of capacity

Node Group Set

- Best instance type
  - 2 fallbacks
- Best big instance type
  - 2 big fallbacks

Node Group Set Controller

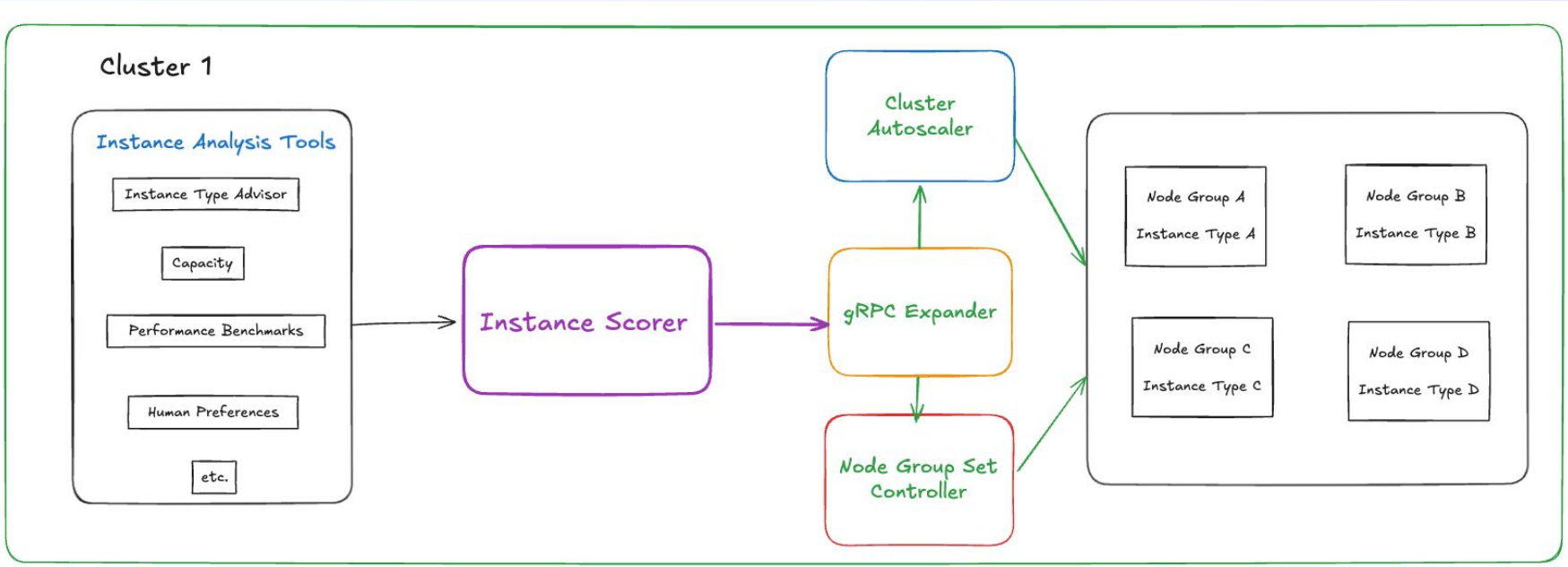
How does the Node Group Set Controller know what the best instance type is?



Node Group Set

- Best instance type
  - 2 fallbacks
- Best big instance type
  - 2 big fallbacks

gRPC Expander *Again*

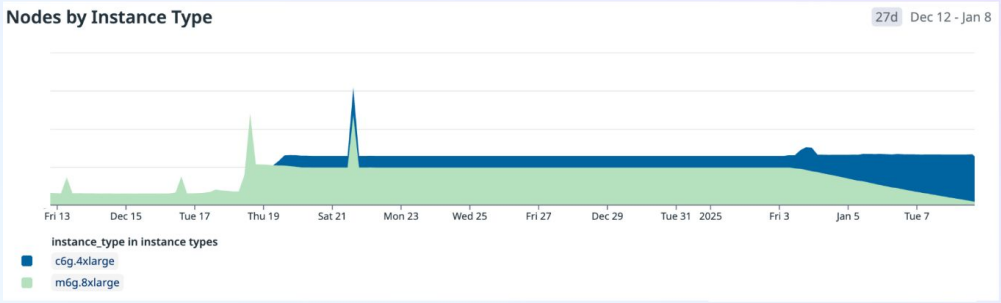


DATADOG

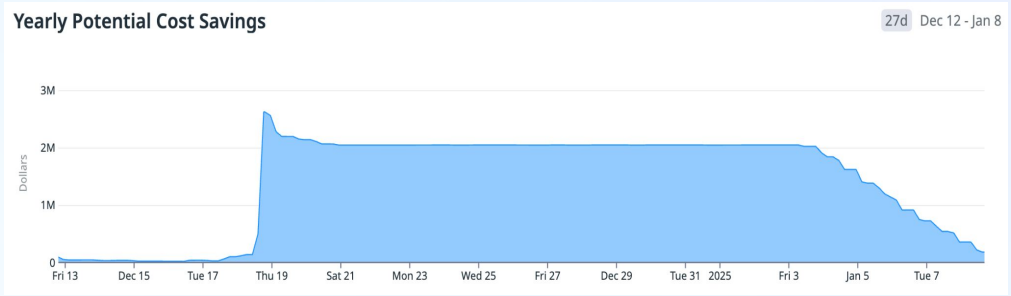
Results

New application deployed, caused shift in optimal instance type

m6g.8xlarge -> c6g.4xlarge



Save: observe ~$2M/year savings



DATADOG

Our Goals

1. **Identify** the best instance types ✅
2. **Scale** the best instance types ✅

Next Steps

Continue abstracting infrastructure details

Providing varieties of instance types
(network enhanced, local storage, etc)

Optimizing placement of applications

Thank you

Rahul Rangith

Software Engineer, Datadog

https://www.linkedin.com/in/rrangith

Ben Hinthorne

Software Engineer, Datadog

https://www.linkedin.com/in/ben-hinthorne

Feedback:

DATADOG