nOps

The Ultimate Guide To Karpenter

By nOps

aws

CONTENTS

By nOps

The Ultimate Guide To Karpenter

Introduction

Almost half of Kubernetes customers on AWS have reported that autoscaling clusters through the Kubernetes Cluster AutoScaler is both restrictive and challenging.

Now, imagine a world where Kubernetes is more aware of your application requirements.

In this world, Kubernetes would automatically select the most cost-effective instance types for your application, helping you to reduce your EKS costs.

This is where Karpenter comes in. Kaprenter is a high-performance, flexible, and open-source Kubernetes cluster autoscaler which is built for AWS. It can significantly enhance both cluster efficiency and application availability by deploying right sized resources when there is a sudden change in application load.

In this eBook, we take an in-depth look into the Karpenter– what it is and how it can help you manage Kubernetes clusters more efficiently. More importantly, we discuss how you can supercharge your Karpenter experience with nOps.

What is Karpenter?

Karpenter is designed to simplify and improve provisioning in Kubernetes clusters. It is an autoscaler that can provision new nodes automatically in response to unscheduled pods. It aggregates all the resource requirements of pending pods and identifies optimal instance types to run them.

It can run on any Kubernetes cluster as a controller and automatically scale custom resource definitions (CRDs) based on the current workload requirements.
Karpenter can help optimize resource utilization and reduce cloud costs in many ways, including:

- Automated scaling
- Rightsizing instances;
- Instance lifecycle management
- Spot instance optimization
- Custom pluggable scheduling strategy

What is the role of Karpenter in Kubernetes?

The role of Karpenter in Kubernetes is simple – Streamline the process of scaling resources present in a Kubernetes cluster by automating the management of different spot instances and resources.

Before the introduction of Karpenter, Kubernetes users had to adjust cluster computing capacity, either through Kubernetes Cluster AutoScaler or Amazon EC2 AutoScaling Groups to prevent wasted node capacity. But Cluster Autoscaler doesn't make scaling decisions, it simply spins up a new node if the CPU and memory of the pending pod will not fit on the current node. This means if the nodes are oversized, a lot of money can be wasted from underutilized infrastructure.

It wasn't just complex and time-consuming but also limited in features.

With the help of Karpenter, it becomes possible to combine all the resource requests for unscheduled pods and automate decision-making for launching new nodes or terminating the existing nodes. This, in turn, leads to decreased infrastructure costs and scheduling latencies.

Karpenter also has a high-level API that allows users to manage the low-level details of scaling resources according to the current demand. The most significant benefit of Karpenter is that it can ensure that applications always have the resources they might need to handle traffic peaks, all the while decreasing costs for idle resources.

A brief history of Karpenter

Karpenter was officially introduced by AWS in November 2022. As an open-source Kubernetes add-on licensed under Apache License 2.0, Karpenter is designed to work with any Kubernetes cluster in just about any environment–including on-premise and cloud.

Getting started with Karpenter

Karpenter can help scale Kubernetes clusters as a node lifecycle management solution by monitoring incoming pods and releasing the right instances at the right time.

Installation and Configuration guide for Karpenter

Let's get into details about how you can install and configure Karpenter. Currently, Karpenter supports EKS and Kops on AWS. But before you move forward with the steps, ensure you are running a Kubernetes cluster on a cloud provider supported by Karpenter.

1. Install The Utilities And Tools Required

To install Karpenter, you will need the following utilities:

- **AWS CLI:** AWS CLI or Command Line Interface allows users to access and manage different AWS services through simple commands. It provides a convenient way to build scripts and automate repetitive tasks. Once you install AWS CLI, configure it so that the user account you select has privileges to create an EKS cluster. You can also verify that the AWS CLI can authenticate by running aws sts get-caller-identity.
- **Kubernetes CLI:** It is a command line tool that can be used to interact with Kubernetes clusters and manage them effectively through a series of commands. Some of the common commands include deployments, services, namespaces, and managing pods. Kubernetes CLI can be installed using package managers or containerized versions like kubectl.
- **eksctl**: It is a simple command line utility that can be used to create and manage Kubernetes clusters on AWS EKS.
- **helm**: The package managers for Kubernetes, helm charts are essentially a set of YAML that help define, install, and upgrade different Kubernetes applications. It can also configure Kubernetes applications and deploy applications in the Kubernetes cluster with just one command.

2. Setup Environment Variables

After installing the tools, you should set the Karpenter version number and the environment variables.

Source: Karpenter

Kubernetes environment variables help define a pod's configuration. These configurations determine how pods or containers in Kubernetes that host applications will run in the given environment.

Here is the list of the main environment variables with their corresponding CLI flags and descriptions.

Environment Variable	CLI Flag	Description
DISABLE_WEBHOOK	--disable-webhook	Disable the admission and validation webhooks (default = false)
ENABLE_PROFILING	--enable-profiling	Enable the profiling on the metric endpoint (default = false)
HEALTH_PROBE_PORT	--health-probe-port	The port the health probe endpoint binds to for reporting controller health (default = 8081)
KARPENTER_SERVICE	--karpenter-service	The Karpenter Service name for the dynamic webhook certificate
KUBE_CLIENT_BURST	--kube-client-burst	The maximum allowed burst of queries to the kube-apiserver (default = 300)
KUBE_CLIENT_QPS	--kube-client-qps	The smoothed rate of qps to kube-apiserver (default = 200)
LEADER_ELECT	--leader-elect	Start leader election client and gain leadership before executing the main loop. Enable this when running replicated components for high availability. (default = true)
MEMORY_LIMIT	--memory-limit	Memory limit on the container running the controller. The GC soft memory limit is set to 90% of this value. (default = -1)
METRICS_PORT	--metrics-port	The port the metric endpoint binds to for operating metrics about the controller itself (default = 8080)
WEBHOOK_PORT	--webhook-port	The port the webhook endpoint binds to for validation and mutation of resources (default = 8443)

Source: ***Karpenter***

Remember that if you open a new shell, you will have to set up some or all of the environment variables again.

3. Setup A Cluster

With the help of eksctl, create a basic EKS cluster. The configuration of the cluster should include the following:

- Leverage **CloudFormation** to set up the infrastructure for the EKS cluster. AWS CloudFormation is a service that essentially helps you set up AWS resources so that you can spend less time managing those resources and more time focusing on your core applications. CloudFormation takes care of configuring and provisioning resources for you.
- Create an **AWS IAM Role and a Kubernetes service account**. Then sync them both using IRSA (IAM roles for Service Accounts), so Karpenter has the required permissions to launch instances. Karpenter needs permission through IRSA in order to make privileged requests to AWS through a service account.
- Add the Karpenter node role to **aws-auth configmap**, which will allow nodes to connect. The aws-auth ConfigMap gets automatically created and applied to clusters when a managed node group is created or when a new node group is created through eksctl.
- For the **kube-system and Karpenter namespaces**, use AWS EKS-managed node groups. (Note that kube-system is the default space for system objects in Kubernetes, while namespaces help organize clusters into virtual sub-clusters in Kubernetes. There can be any number of namespaces within a cluster. Each namespace might be logically different, but they still have the ability to communicate with each other.)

When Karpenter is installed, it creates its own namespace where it can deploy its custom resources, controllers, and other components. If you want to use Fargate for namespaces, you will have to uncomment fargateProfile settings and comment out the managed Node groups settings.

- Set up variables for KARPENTER_IAM_ROLE_ARN.
- Create a separate role to allow spot instances.
- Finally, to install Karpenter, run helm chart.

4. Setup A Default Provisioner

You need to set up a default provisioner when you install Karpenter for the first time. The provisioner is responsible for setting up constraints for nodes created by Karpenter as well as the pods that run on those very nodes.

A provisioner can set default nodes, limit pods that can run on each node, and limit node creation in certain instance types, zones, and architectures.

With the help of provisioners, Karpenter can schedule and provision decisions based on different pod attributes and, in turn, eliminate the need to manage each node group separately.
In order to create a default provisioner, you can use the command below.

```
cat <<EOF | kubectl apply -f -
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
  requirements:
    - key: karpenter.sh/capacity-type
      operator: In
      values: ["spot"]
  limits:
    resources:
      cpu: 1000
  providerRef:
    name: default
  ttlSecondsAfterEmpty: 30
---
apiVersion: karpenter.k8s.aws/v1alpha1
kind: AWSNodeTemplate
metadata:
  name: default
spec:
  subnetSelector:
    karpenter.sh/discovery: ${CLUSTER_NAME}
  securityGroupSelector:
    karpenter.sh/discovery: ${CLUSTER_NAME}
EOF
```

Source: ***Karpenter***

Though it's important to note that the provisioner can only create capacity if the sum of all the required capacity is less than the limit specified.

Once the above step is done, Karpenter will get activated and ready to provision nodes.

Best Practices For Setting Up Karpenter

- ## Use Spot Instances With Interruption Handling

Spot instances provide significant cost savings compared to on-demand instances, but they can get interrupted if the demand increases beyond the available capacity.

Enabling interruption handling in Karpenter can help manage involuntary interruptions, like with spot instances that can subsequently cause workload disruptions. It can also handle other events like maintenance, instance terminating, and instance stopping events.

To enable interruption handling, you just need to enable aws.interruptionQueueName in the Karpenter Settings

- ## Avoid Custom Launch Templates

Karpenter guidelines recommend avoiding custom launch templates since they don't support the automatic upgrade of nodes, multi-architecture support, or securityGroup discovery.
Instead of launch templates, you can use custom user data or directly add custom AMIs in AWS node templates.

- ## Configure Node Expiration On Your Provisioner

With Karpenter, it's possible to expire nodes automatically after a specific time period without causing any downtime. This helps ensure that all the nodes are always running the latest security patches.

- ## Set Up Provisioners According To Your Workload Types

Stateful workloads are less tolerant to node churn which is why it's advised to set up a provisioner that only uses on-demand instances for these types of workloads. For stateless fault-tolerant workloads, you can set up a provisioner that only uses spot instances.

- ## Setup A Large Pool Of Instance Types

One of the main benefits of Karpenter is its ‘just in time capacity’ which basically means that Karpenter chooses an instance type that fits your workload as best as possible. But to leverage the power of this feature, you need to set up a large pool of instances.

If you limit the instance types, you won’t be able to maximize the benefits of using Karpenter.

- ## Always specify the architecture in your provisioner

If the architecture is not specified, Karpenter might launch instance types with ARM CPU, for which your applications may not be ready.

- ## Specify resources for your deployments/pods

Karpenter will use the pods resource request and limits for its calculations which is why it is necessary to specify resources for deployments/ nods. Not specifying resources can cause unexpected behavior in the cluster scaling.

- ## Configure the right parameters for Karpenter

With so many configuration options available on Karpenter, it's best to make the most of it by optimizing parameters in the right way and ensuring your workloads are always streamlined and scheduled. For instance, you can define custom node selectors, set custom resource limits, and specify minimum or maximum node counts.

What Are The Real-World Use Cases Of Karpenter?

Karpenter offers a range of real-world use cases demonstrating its value and versatility in effectively managing Kubernetes clusters. Some prominent use cases of Karpenter include:

- Rapidly changing workloads: Karpenter takes a proactive approach to node provisioning, enabling it to swiftly adapt to fluctuating workload demands. If your Kubernetes cluster frequently experiences shifts in resource requirements or has workloads characterized by short-lived, high-intensity bursts, Karpenter ensures more efficient scaling to meet those dynamic demands promptly.

- Granular control over node lifecycle: Karpenter provides fine-grained control over node termination through its Time-To-Live (TTL) settings. This can be useful for scenarios where you need to manage node lifecycles based on factors such as cost considerations, usage patterns, or scheduled maintenance, allowing for fine-grained control over resource utilization.

- Optimizing resource utilization: Karpenter's customizable scaling policies and support for diverse instance types can help optimize resource utilization in your cluster. If you need to manage various workloads with different resource requirements, Karpenter can help ensure that your cluster provisions nodes tailored to your workloads' needs.

- Advanced scheduling and affinity rules: Karpenter supports advanced scheduling and affinity rules to help you better manage workload placement and resource allocation in your cluster. If you have specific requirements for workload distribution or need to enforce strict resource constraints, Karpenter provides the flexibility to handle these scenarios.

- Better handling of spot instances: Karpenter's approach to spot instances offers better cost optimization and flexibility compared to the Cluster Autoscaler. Karpenter can automatically provision a mix of on-demand and spot instances, dynamically choosing the most cost-effective options that meet your workloads' resource demands.

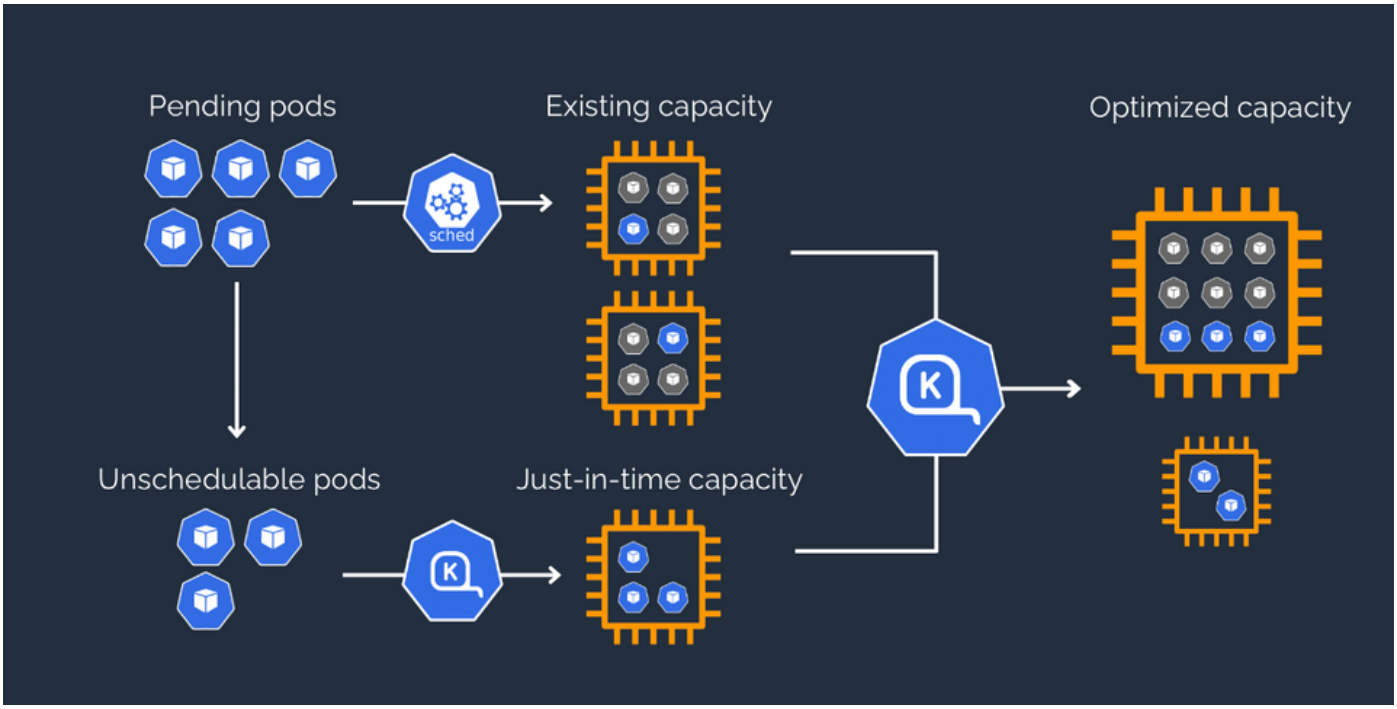
These capabilities collectively contribute to a more streamlined and efficient management of your Kubernetes clusters.

Understanding Karpenter's Architecture

Karpenter's architecture is based on a Kubernetes controller, which interacts with the API server of Kubernetes. Karpenter works directly with the Kubernetes scheduler and observes incoming nodes in the cluster. It automatically starts or stops nodes to optimize cluster utilization and application availability.

When there is sufficient capacity present in a Kubernetes cluster, the Kubernetes scheduler follows the same process of placing and scheduling incoming pods. But when there are pods that cannot be scheduled due to the existing cluster capacity, Karpenter takes over and interacts directly with the cloud provider to kickstart the minimum computing resources required for the incoming pods and the nodes associated with them.

Similarly, Karpenter constantly monitors clusters, and when a pod gets removed, it also terminates any underutilized nodes to optimize resource usage.



Core Components of Karpenter

The main components of Karpenter, include:

Karpenter Controller:

The controller manages the node lifecycle as well as resources. It decides when to scale up or down depending on the current availability of resources and demand. The Karpenter Controller also provisions and deprovisions EC2 instances to maintain the required state of the cluster.

Karpenter API server:

It provides an API that helps define the current state of the cluster by using CRDs (custom resource definitions).

Karpenter Scheduler:

The scheduler is responsible for scheduling pods based on the current resource requirements and node availability. It works alongside the Kubernetes scheduler to make all scheduling decisions. It can override the Kubernetes scheduler when there isn't sufficient capacity in the cluster of incoming pods. The scheduler allows users to define their own custom scheduling strategies according to their specific requirements.

Karpenter Scheduler:

The scheduler is responsible for scheduling pods based on the current resource requirements and node availability. It works alongside the Kubernetes scheduler to make all scheduling decisions. It can override the Kubernetes scheduler when there isn't sufficient capacity in the cluster of incoming pods. The scheduler allows users to define their own custom scheduling strategies according to their specific requirements.

Karpenter Webhook:

It is the responsibility of the Karpenter Webhook to validate the admission of newly created or updated Kubernetes objects. The Webhook makes sure that the objects are in the correct format and comply with policies that have been defined by Karpenter.

Metrics and logging:

There are built-in metrics that come with Karpenter, which allow users to monitor the overall performance and health of the cluster.

Apart from these components, Karpenter also has two in-built services for node allocation:

- Allocator: The allocator is responsible for allocating any unscheduled pods available to the nodes which have the minimum amount of latency.
- Reallocator: The reallocator, on the other hand, is responsible for reallocating pods back to nodes and when the nodes are not running at their maximum capacity. While the allocator gets to work immediately, the reallocator does not. The reallocator only comes once to reallocate pods across the different nodes. Then it effectively terminates any pods not being used to avoid any extra computing cloud costs.

While the goal of the allocator is to decrease latency, the goal of the reallocator is to decrease costs.

Karpenter vs other Autoscaling solutions

Amazon Elastic Kubernetes Service (EKS) provides two autoscaling solutions: The open-source Karpenter and Cluster Autoscaler. While Karpenter works directly with Amazon EC2 instances, Cluster Autoscaler leverages Auto Scaling Groups (ASG).

What is Cluster Autoscaler?

The Cluster Autoscaler adjusts the number of nodes in a cluster automatically when pods get rescheduled to other nodes or when they fail entirely.

Before going into how Cluster Autoscaler compares to Karpenter, let's discuss some of the main terms associated with it.

→ Auto Scaling Group (ASG):

An Auto Scaling group is a collection of EC2 instances that get treated as one logical group for automatic scaling. They are configured to launch instances that automatically join their cluster. Maintaining automated scaling and the number of instances present in an ASG are two of the core functionalities. ASG can also launch spot instances and on-demand instances.

→ EKS Node Groups:

This is a Kubernetes abstraction that comprises a group of nodes within a cluster. When part of a single node group, nodes often share similar properties like taints and labels. Based on these properties, different nodes in a cluster get grouped together.

The Cluster Autoscaler uses custom controllers in order to automatically adjust cluster sizes. It monitors the resource utilization of nodes and resource requests of pods. When it detects a node that is running out of resources, it automatically creates a new node to handle the demand.

Similarly, when it detects an underutilized node, Cluster Autoscaler removes it to save on costs. It can also effectively work across different availability zones simultaneously, which can lead to resilience and high availability.

Cluster Autoscaler works for most environments, but it has some limitations.

- Underutilization of resources: In case there is not enough load generated in the Kubernetes workload, the Cluster Autoscaler may not even get triggered to add new nodes to the cluster – which can in turn lead to underutilization of cluster node resources. In most situations, all the memory of the nodes and CPU cores are also not used to their full capacity, which also causes underutilization.

- Manual creation of node groups required: When deployments need specific CPU architecture or different CPU-based instance types, new node groups have to be manually created when deployed.

- Timeout-based error handling: In case an error occurs when creating a node in a node group, Cluster Autoscaler would not find out about it until the timeout. This can cause an invariable delay in scaling out.

How does Karpenter address the present limitations of Cluster Autoscaler?

Karpenter gets around the limitations of Cluster Autoscaler by eliminating one of the main core concepts of the former: Creating node groups. In fact, it works on the very concept of no node group.

Here are some of the many benefits of Karpenter, which also helps it address the main limitations of Cluster Autoscaler:

- **Provision capacity directly:**

Based on the resource requirements of Kubernetes workloads, Karpenter can provision capacity directly. In other words, Karpenter can automatically provision instances that match the present resource requirements of the workload. This also ensures that there is no overprovisioning or under provisioning of resources.

It also allows Karpenter to reduce costs by boosting any present low-cost instances that match the workload's resource requirements. For example, if the workload requires high memory and CPU, but there is no high storage requirement, then Karpenter can boost an instance from an instance family that has the required high memory and CPU but at the same time has lower costs and storage capacity.

- **Selects instance types from pod resource requests:**

When you specify resource requirements for pods, Karpenter uses that information to select the right instance types for the nodes. Karpenter reviews the resource requirements of all the unscheduled pods and then selects the instance type which fulfills the resource requirements.

- **Dynamic provisioning of nodes:**

Karpenter automatically provisions new nodes according to the changes in workload demand. For instance, when the demand for resources increases in the Kubernetes cluster, Karpenter can provision new nodes to make sure there is enough capacity available to meet the demand. By constantly monitoring the resource utilization of new nodes, Karpenter can also automatically terminate nodes that are no longer needed, which in turn saves cost.

- **Provision capacity directly:**

Based on the resource requirements of Kubernetes workloads, Karpenter can provision capacity directly. In other words, Karpenter can automatically provision instances that match the present resource requirements of the workload. This also ensures that there is no overprovisioning or under provisioning of resources.

It also allows Karpenter to reduce costs by boosting any present low-cost instances that match the workload's resource requirements. For example, if the workload requires high memory and CPU, but there is no high storage requirement, then Karpenter can boost an instance from an instance family that has the required high memory and CPU but at the same time has lower costs and storage capacity.

- **Closer integration with Kubernetes:**

Since Karpenter is majorly designed to work with Kubernetes, its integration with Kubernetes is better and more seamless as compared to ASGs.

- **More effective scaling:**

Karpenter can scale up or down more quickly and effectively as compared to ASGs. This is because Karpenter can make granular scaling decisions and also allocate resources on the pod level.

Let's take a look at Karpenter vs Cluster Autoscaler in more detail.

Feature	Karpenter	Cluster Autoscaler
Resource Management	Based on the current resource requirements of unscheduled pods, Karpenter takes a proactive approach for provisioning nodes.	Based on the resource utilization of existing nodes, Cluster Autoscaler takes a reactive approach to scale nodes.
Node management	Karpenter scales, provisions, and manages nodes based on the configuration of provisioners.	Cluster Autoscaler manages nodes based on the resource demands of the present workload.
Cluster management	Karpenter supports multi-cluster management of nodes and their corresponding node groups.	Cluster Autoscaler supports multiple node groups across different clusters.
Scaling	Karpenter offers more effective and granular scaling functionalities based on specific workload requirements. In other words, it scales according to the actual usage. It also allows users to specify particular scaling policies or rules to match their requirements.	Cluster Autoscaler is more focused on node-level scaling, which means it can effectively add more nodes to meet any increase in demand. But this also means that it may not be as effective in downscaling resources.
Scheduling	Karpenter can effectively schedule workloads based on different factors like availability zones, resource requirements, and costs. It can also optimize the scheduling of workloads to maximize efficiency.	With Cluster Autoscaler, scheduling is more simple as it is designed to scale up or down based on the present requirements of the workload.
Nature of availability	Since Karpenter is present on top of Kubernetes and it leverages the infrastructure of Kubernetes, it is inherently designed to be fault tolerant and available.	While the Cluster Autoscaler tool is a standalone tool, it can be installed over any Kubernetes cluster. But since it is not integrated closely with Kuberntes like Karpenter is, it may require some kind of manual intervention to maintain availability.

Advanced Karpenter Techniques

1. Using instances that fit your workload requirements

There are a large variety of instance sizes and families available on AWS, and each of them has to be optimized for different types of workloads.

When you are defining Provisioners with Karpenter, focus on choosing instance sizes and families which perfectly fit the resource requirements of the workload, which will directly lead to improved performance and efficiency.

2. Use multiple provisioners for different types of workloads

When you have a complex Kubernetes environment with different workloads, it's crucial to use different provisioners for each workload in order to optimize the resources available for each workload.

With Karpenter, you can use multiple provisioners so that you can manage scheduling and scaling more efficiently. You can allocate resources for different workloads and scale each of the provisioners separately on its own.

Supercharge Karpenter with the help of nOps!

Karpenter has a lot of potential, but its capabilities are limited because it's a new open-source project. nOps, with its advanced ML and AI-driven approach, goes beyond what Karpenter can do,
builds on its amazing potential, and adds on to its capabilities.

nOps can select the most cost-optimal nodes to schedule them based on a single cluster view while also keeping an eye on your RIs and Savings Plan commitments. It continuously rebalances cluster nodes based on workload changes across the entire AWS ecosystem.

More importantly, nOps can help you configure and manage Karpenter through a simple user interface instead of a complicated command line interface.

Here are some of the many ways nOps can supercharge Karpenter:

- Leverage spot instances to optimize savings by up to 90%
- A 60-minute advance termination prediction where nOps constantly watches the spot market to predict a shift in demand and avoid any spot interruptions
- Price reconsideration, which constantly reevaluates the instance mix to ensure the best pricing
- A user-friendly interface to manage Karpenter resources and configuration
- In-depth reports to track the current and pasts spending on instances

Ready to make the most of Karpenter while also saving up on your cloud budget?

Sign up with nOps today!