EKS Autoscaling with Karpenter

Ananda Dwi Rahmawati

Hello!

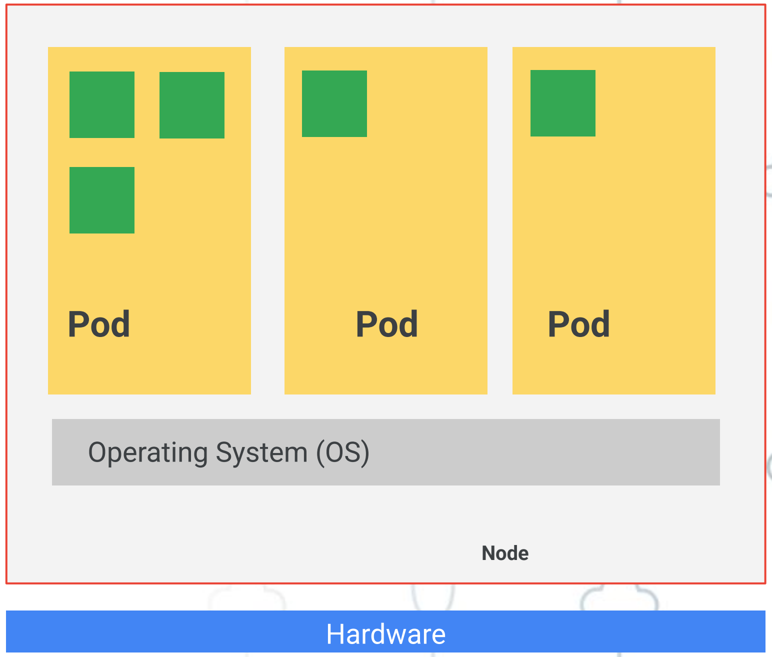
- Sr. Cloud Engineer @ Btech
- AWS Container Hero
- Open Source Enthusiast and Communities fellow
- Tech background: System, Networking, IaaS & PaaS Cloud, DevOps, a bit of Programming
  https://linktr.ee/misskecupbung

Agenda

- Kubernetes
- Autoscaler
- Karpenter
- Karpenter Demo
- QnA

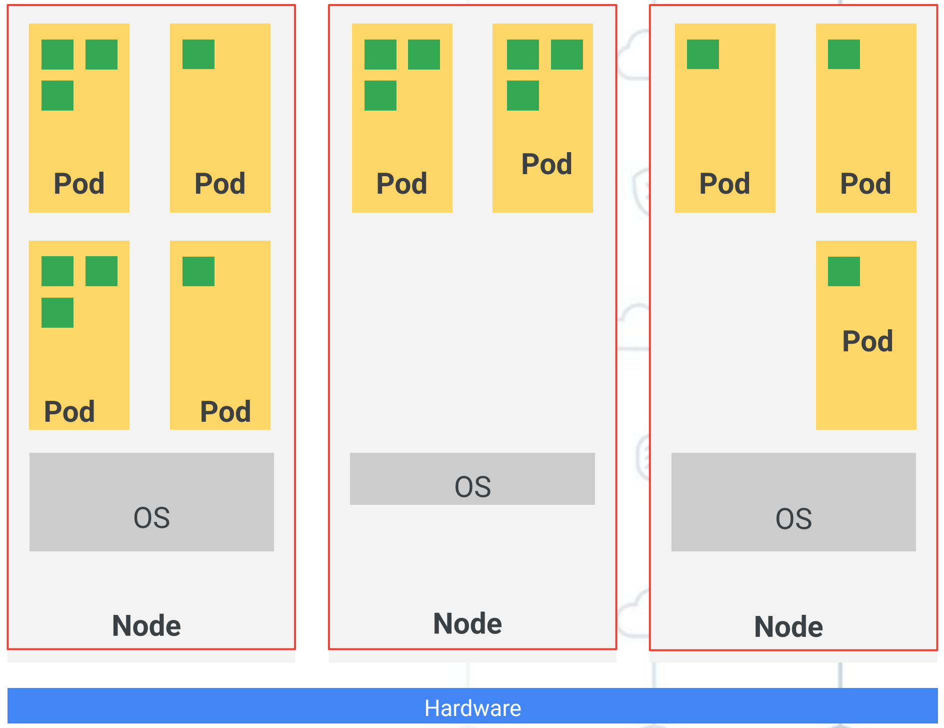
Pods and Workloads

- Containers execute within pods.
- A pod makes its environment available to containers; for example, its:
  - Network ports
  - IP address
  - Namespace
- The term workload is sometimes used for how to deploy a pod.



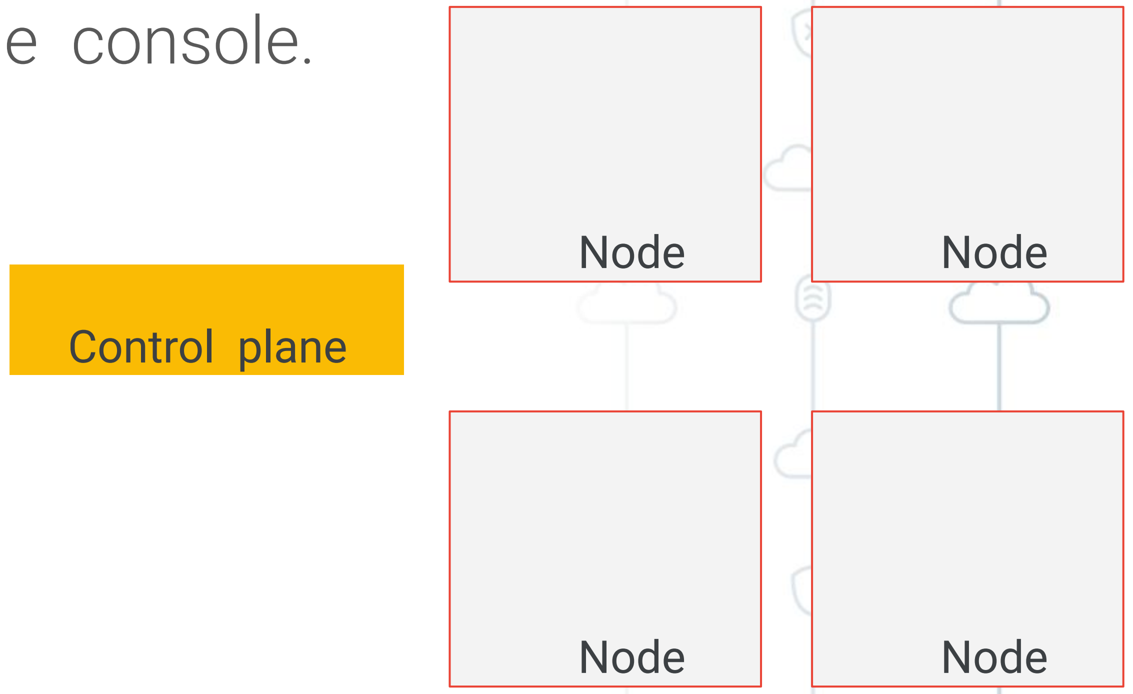
Nodes

- Each node:
  - Is a virtual machine.
  - Has its own instance of the OS.
- Nodes provide services to the pods.



Clusters

- Clusters are a set of one or more nodes.
- The control plane (primary node) controls the other nodes in the cluster.
- The control plane is not visible in the console.



Cluster Autoscaler

Cluster Autoscaler is a tool that automatically adjusts the size of the Kubernetes cluster when one of the following conditions is true:

- there are pods that failed to run in the cluster due to insufficient resources.
- there are nodes in the cluster that have been underutilized for an extended period of time and their pods can be placed on other existing nodes.

https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler

Open-source cluster autoscaler that automatically provisions new nodes in response to unschedulable pods.

Built with ❤️ at AWS

Karpenter

Karpenter improves the efficiency and cost of running workloads on Kubernetes clusters by:

- **Watching** for pods that the Kubernetes scheduler has marked as unschedulable
- **Evaluating** scheduling constraints (resource requests, nodeselectors, affinities, tolerations, and topology spread constraints) requested by the pods
- **Provisioning** nodes that meet the requirements of the pods
- **Removing** the nodes when the nodes are no longer needed

Benefit(s)

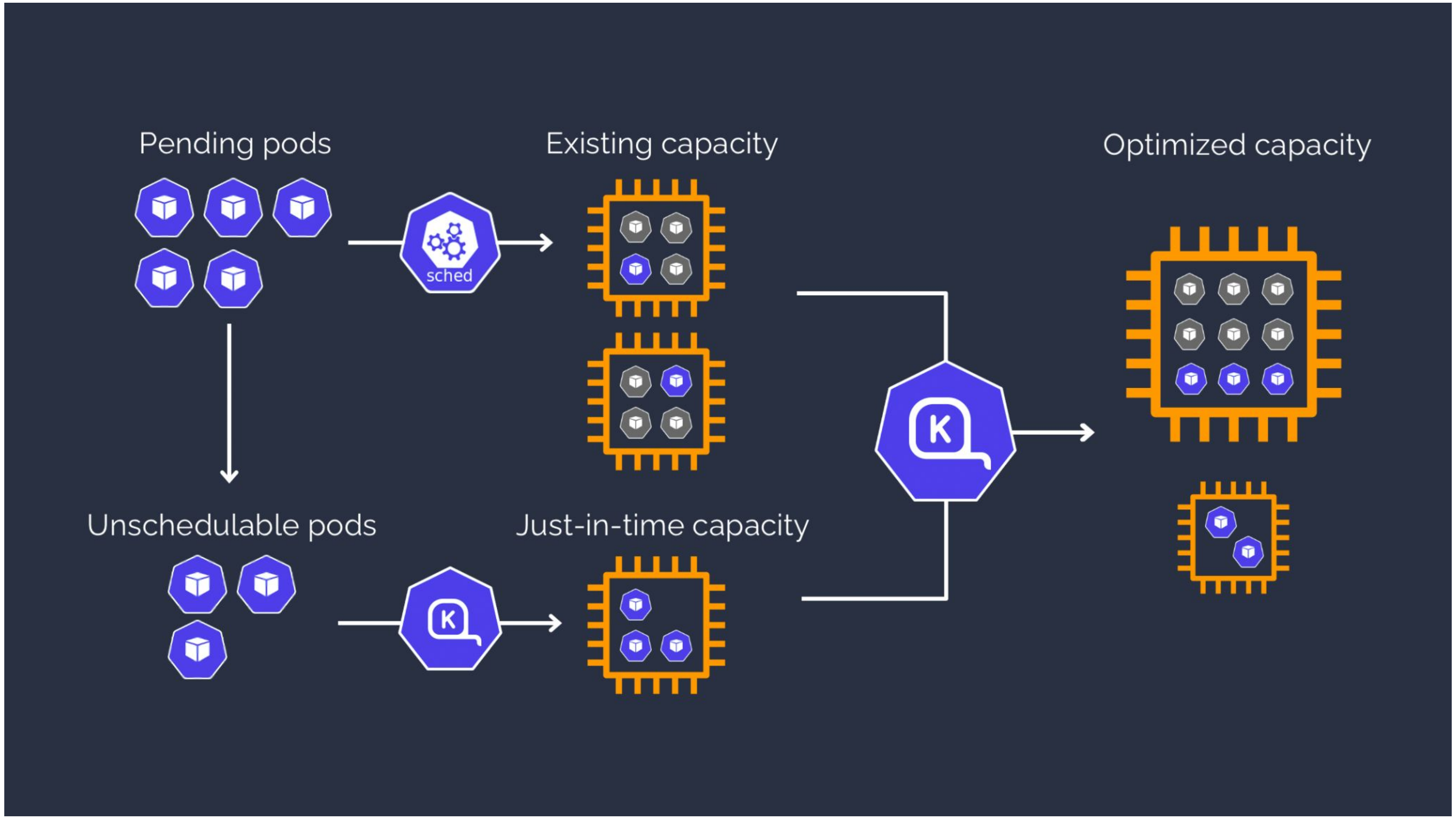
- Improve application availability
- Lower compute costs
- Minimize operational overhead

Reason to use Karpenter

- Provision nodes based on **workload** requirements
- Create diverse node configurations by **instance type**, using flexible workload provisioner options.
- Instead of managing many specific custom node groups, Karpenter could let you manage diverse workload capacity with a single, flexible provisioner.
- Achieve improved pod scheduling at scale by quickly **launching** nodes and **scheduling** pods.

Reason to use Karpenter (vs Cluster Autoscaler)

- When there are no node in the Node group that matches the requirements of the pod and the pod remains unscheduled, which could cause an outage.
- While debugging these types of issues a common error message in autoscaler logs is pod didn't trigger scale-up (it wouldn't fit if a new node is added)
- Using too big instances in node groups, which leads to low resource utilization and increased cost.
- Using too low instances in Node groups, which leads to node groups maxing out and resulting in unscheduled pods.
- No way to identify the optimal choice of instance types based on workloads.


Pending pods
Existing capacity
Optimized capacity
sched
Unschedulable pods
Just-in-time capacity

How it works

Demo

https://s.id/1CTug

References

- https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler
- https://karpenter.sh/
- https://aws.github.io/aws-eks-best-practices/karpenter/
- https://github.com/aws/karpenter
- https://www.youtube.com/watch?v=3QsVRHVdOnM

Thank You