e-Book

Managing EKS at scale

Dealing with the complexities of deploying, managing, and scaling EKS across multiple AWS accounts

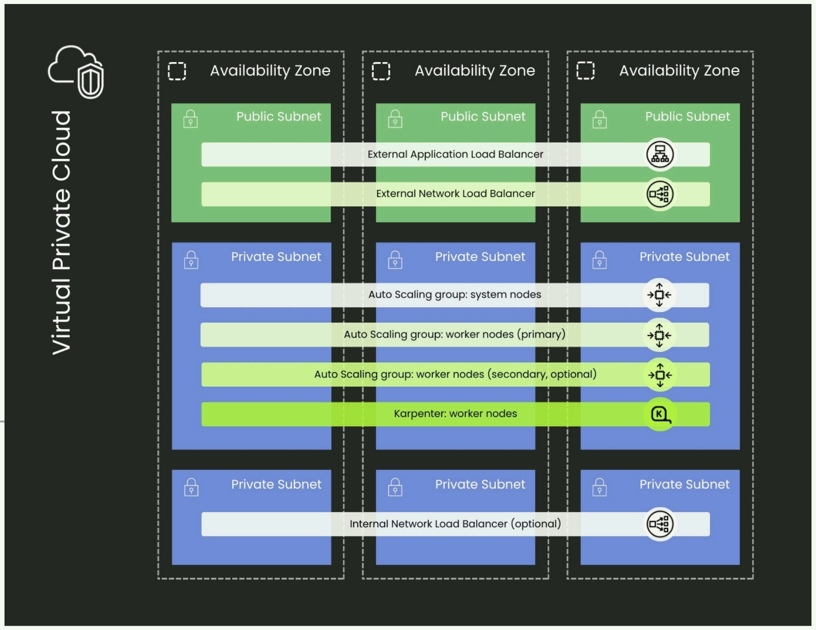
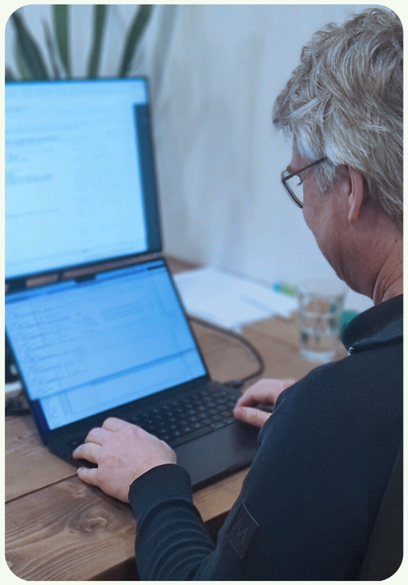

Virtual Private Cloud
Availability Zone
Availability Zone
Availability Zone
Public Subnet
Public Subnet
Public Subnet
External Application Load Balancer
External Network Load Balancer
Private Subnet
Private Subnet
Private Subnet
Auto Scaling group: system nodes
Auto Scaling group: worker nodes (primary)
Auto Scaling group: worker nodes (secondary, optional)
Karpenter: worker nodes
Private Subnet
Private Subnet
Private Subnet
Internal Network Load Balancer (optional)

Table of contents

Introduction

If you're reading this, you must be a well seasoned cloud engineer looking for ways to master the management of AWS EKS at scale, right? Or you might just be interested in how to approach these kinds of challenges. Either way, great to have you here!



This guide's aim is to ensure that every development team can focus on their applications without being bogged down by the intricacies of the infrastructure that powers them. This guide shares insights and methodologies that have proven effective in real-world scenarios. Because efficiency and automation are not just goals; they're necessities.

Drawing from firsthand experience, this ebook addresses the complexities of deploying, managing, and scaling EKS across multiple AWS accounts.It provides a clear account of what works, the challenges faced, and strategies for overcoming them.

WHEN TO USE EKS

EKS may be a good choice if you have large or complex applications, as it can scale more easily and handle more traffic. If you are new to Kubernetes, ECS may be a better choice, as it is more beginner-friendly.

We'll walk you through the entire journey, from the initial setup of EKS clusters to the automation strategies that keep them running smoothly. This includes an exploration of the tools and practices that have streamlined operations, reduced overhead, and maintained high agility and reliability across a cloud infrastructure.

Whether you're a cloud engineer refining your EKS approach or part of a team scaling your Kubernetes operations.

Let's dive into the specifics of managing EKS at scale, focusing on automation, best practices, and lessons learned.
Enjoy!

On behalf of Aknostic's engineering team,

Jurg van Vliet - CEO

1

Setting the foundations

Anchor your strategy on core requirements and guiding principles

Setting the foundations

Before we go into depth on the matter, to effectively manage EKS at scale, you should anchor your strategy on core requirements and guiding principles. These foundational elements shape your approach and ensure that your cloud infrastructure can support the rapid pace of innovation and the diverse needs of your development teams.

Core requirements and guiding principles

- central management of AWS infrastructure & cluster services
- local configuration of parameters
- automated deployments & upgrades
- knowledge sharing culture
- self-sufficient development teams
- lower costs

Let's start with the obvious thing here: centralization is key. By managing your AWS infrastructure and Kubernetes services from a single point of control, you can maintain consistency, enforce security standards, and streamline the deployment process across all your environments. This approach allows you to automate repetitive tasks, reducing the potential for human error and freeing up your teams to focus on more strategic initiatives.

While central management provides the oversight you need, flexibility at the local level is crucial. Development teams need the ability to configure parameters specific to projects, such as Node Pools and Application Load Balancers (ALBs), without compromising the overall integrity of your infrastructure. This balance between central governance and local autonomy is essential for catering to each project's unique requirements.

Automation stands at the heart of your operations. Automating deployments and upgrades ensures that your clusters are always running the latest software versions, packed with the newest features and security patches. This minimizes downtime and significantly reduces the workload on your technical operations team, allowing them to concentrate on more complex challenges.

By fostering a knowledge-sharing culture, teams can learn from each other's experiences and document best practices. This creates a valuable knowledge base for the entire organization. A disciplined approach is essential to ensure this culture is maintained.

Making development teams self-sufficient is another key objective. By minimizing dependencies and enabling independent deployment, teams can accelerate development cycles and reduce bottlenecks. Providing them with the necessary tools and permissions allows for innovation, experimentation, and faster adaptation to changing requirements.

This approach reduces IT infrastructure management overhead, allowing developers to focus on their core competencies. By abstracting away complexities, developers can concentrate on delivering value rather than being bogged down in operational details. This approach is designed with cost efficiency in mind, through automation, efficient resource utilization, and elimination of redundant efforts.

2 →

Initial setup and transition

Supporting a growing number of applications and development teams

Initial setup and transition

We're making an estimated guess here, but your (company's) wish to evolve the EKS setup is probably driven by the need to support a growing number of applications and development teams. Your setup may already include multiple EKS clusters of varying sizes to meet the diverse requirements of different projects. Whatever scenario applies to you, each team or project should typically operate within separate AWS environments for development, staging, and production, ensuring a clear separation and facilitating smoother deployment processes.

It's recommended to set up a comprehensive architecture for managing EKS clusters within a VPC spanning three Availability Zones, ensuring high availability and fault tolerance. It includes public subnets for external Application Load Balancers (ALBs) and optional Network Load Balancers (NLBs), non-routable private subnets for system and worker nodes (both primary and secondary), and routable private subnets for internal NLBs. This setup strategically distributes components to maximize performance, security, and scalability, providing a robust foundation for running EKS at scale.

Your Initial Setup
When you begin your transition, your clusters might operate on earlier versions of Kubernetes. Over time, as Kubernetes evolves, you can meticulously upgrade them to stay abreast of the latest versions, ensuring you can utilize the newest features and security enhancements Kubernetes offers.

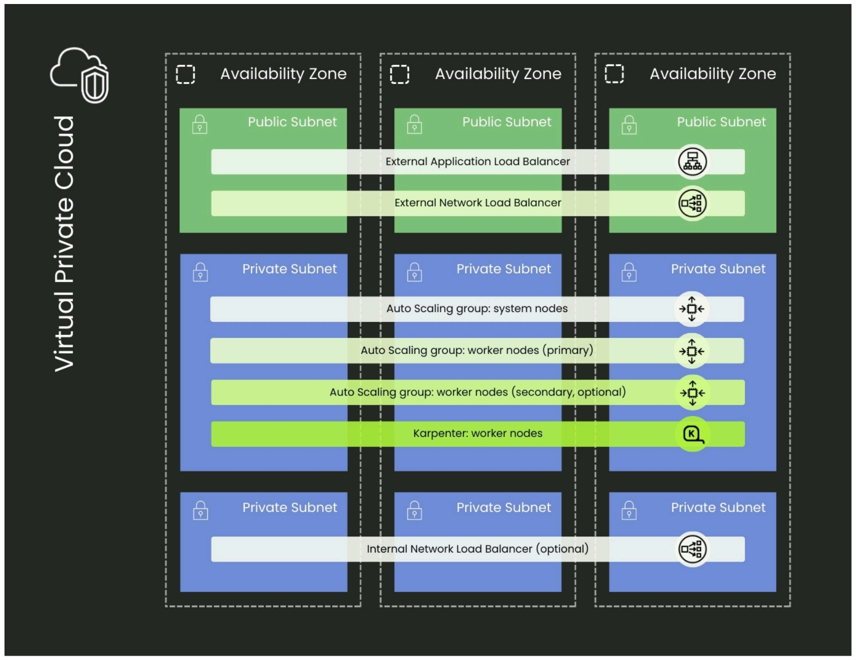
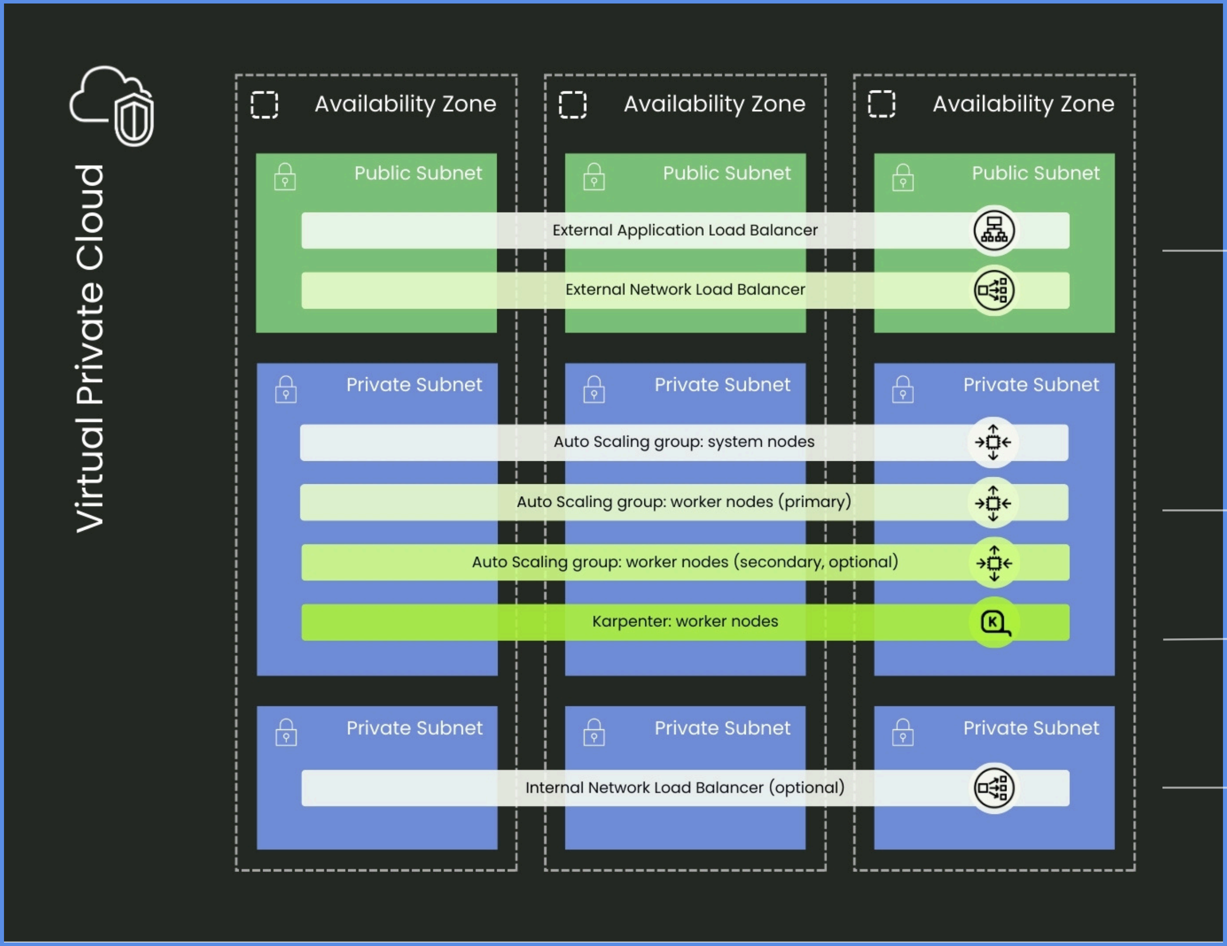

Virtual Private Cloud
Availability Zone
Availability Zone
Availability Zone
Public Subnet
Public Subnet
Public Subnet
External Application Load Balancer
External Network Load Balancer
Private Subnet
Private Subnet
Private Subnet
Auto Scaling group: system nodes
Auto Scaling group: worker nodes (primary)
Auto Scaling group: worker nodes (secondary, optional)
Karpenter: worker nodes
Private Subnet
Private Subnet
Private Subnet
Internal Network Load Balancer (optional)

Diagram 1: architecture for managing EKS clusters

Architecture for managing EKS clusters


Virtual Private Cloud
Availability Zone
Availability Zone
Availability Zone
Public Subnet
Public Subnet
Public Subnet
External Application Load Balancer
External Network Load Balancer
Private Subnet
Private Subnet
Private Subnet
Auto Scaling group: system nodes
Auto Scaling group: worker nodes (primary)
Auto Scaling group: worker nodes (secondary, optional)
Karpenter: worker nodes
Private Subnet
Private Subnet
Private Subnet
Internal Network Load Balancer (optional)

External Application Load Balancer
Routes traffic from the internet to services in the public subnet, typically configured for HTTPS with SSL termination for secure external access.

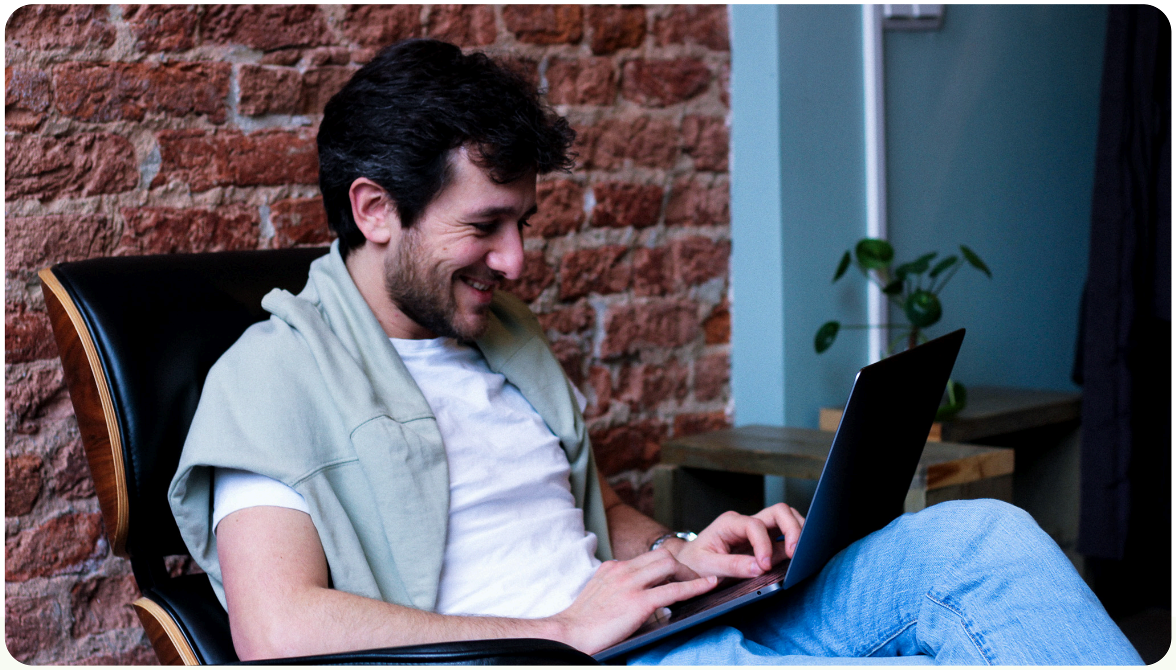
Auto Scaling Group
Hosts critical system components like the Kubernetes control plane, DNS, and network management. Scaling ensures the availability of core services.

Private Subnets
Isolated from direct internet access for enhanced security. Used to host internal components such as system nodes and worker nodes.

EKS setup
t's recommended to set up a comprehensive architecture for managing EKS clusters within a VPC spanning three Availability Zones

The clusters' sizes may vary, reflecting the specific needs of the applications they host. Depending on the application's user base, complexity, and performance requirements, a cluster can have multiple worker nodes. This flexibility allows you to tailor your resources to match demand, optimizing performance and cost.

Skip the line with Karpenter
As your understanding of EKS deepens and your needs evolve, you will need a more dynamic, scalable infrastructure. There are numerous tools available for Kubernetes, each fulfilling a specific need. If you don't have a good overview of what's on the market and not enough time to proficiently benchmark them, we advise you to make use of Karpenter, an open-source tool developed by AWS. Karpenter marked a significant shift in managing your clusters, offering unparalleled flexibility. With Karpenter, you can specify instance types, availability zones, and purchasing options (on-demand vs. spot instances) with granular control, directly addressing the unique requirements of each deployment.



This move is not just a technical upgrade; It also allows you to scale your clusters more efficiently, responding to traffic spikes without manual intervention and significantly reducing the time required to scale up resources. Moreover, Karpenter's ability to interact directly with the EC2 Fleet API streamlines operations, cutting down the time to scale from a few minutes to tens of seconds in many cases.

The Impact of the New Setup
The transition to a Karpenter-based infrastructure should profoundly impact operations. It enhances the clusters' responsiveness, enabling you to handle sudden increases in load easily. This agility ensures that applications remain performant and reliable, even under the most demanding conditions.

Furthermore, moving to Karpenter and adopting more advanced AWS services like the AWS Load Balancer Controller for Kubernetes allows you to automate and fine-tune your load-balancing strategies. This improves applications' availability and resilience and gives your clients' development teams more autonomy in managing their services.

3 Infrastructure pipelines

The backbone of EKS deployment

Infrastructure pipelines: The backbone of EKS deployment

Having established the foundational setup of your EKS clusters and implemented automation strategies for smooth operations, the next step is to scale your infrastructure efficiently across multiple environments. This involves adopting a robust, centralized management approach to ensure consistency, control, and scalability.

This chapter will delve into a multi-account AWS architecture designed to streamline EKS cluster management and deployments. This architecture makes use of AWS CodeCommit, AWS CloudFormation, and AWS CodePipeline to manage resources centrally from a core management account. In our experience, this ensures seamless deployment and operational consistency across development, acceptance, and production environments. Let's explore how this setup can transform your EKS management practices.

Infrastructure backbone

Without a structured approach to managing EKS clusters at scale, you risk encountering significant inefficiencies and inconsistencies. Integrating CloudFormation StackSets with AWS CodePipeline is crucial. This combination forms the backbone of your infrastructure pipelines, enabling you to deploy and manage EKS clusters across multiple AWS accounts with precision and control. Without this structured integration, you could face fragmented deployments, increased manual intervention, and potential configuration drifts, ultimately leading to operational headaches and reduced scalability.

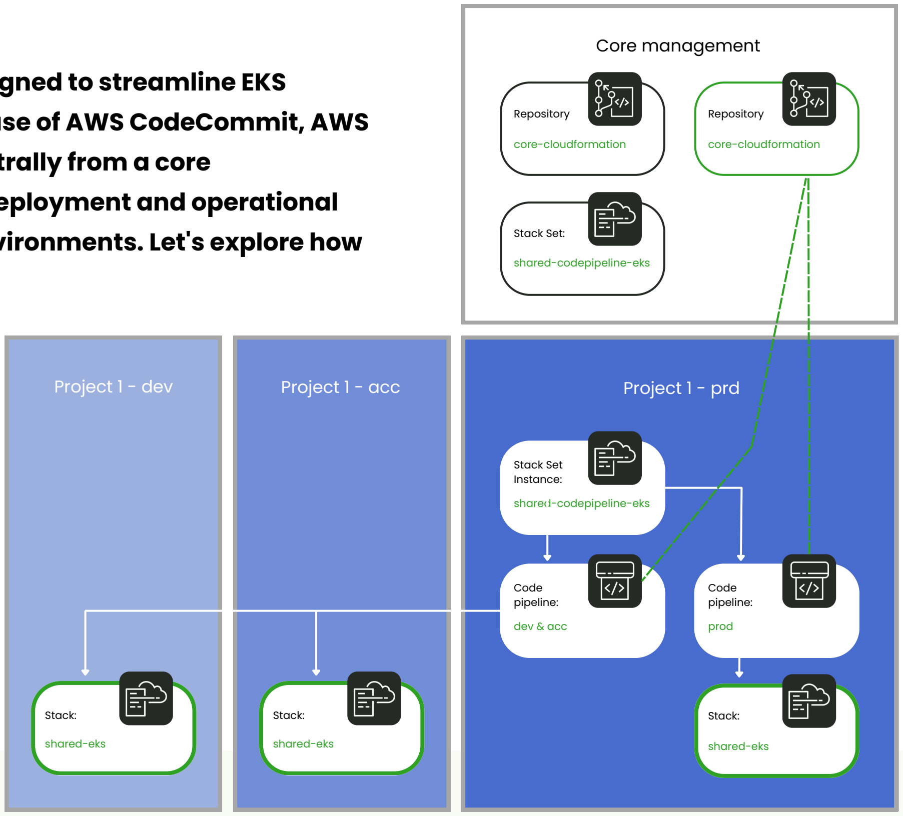


Diagram 2: infrastructure pipelines

The diagram on page 15 shows a clear and structured way you can set up a multi-account AWS architecture for managing and deploying EKS clusters and related resources. It features a core management account (core-mgmt) that centralizes the configuration and deployment processes using AWS CodeCommit, AWS CloudFormation, and AWS CodePipeline. The core-mgmt account contains repositories for CloudFormation templates, which are used to create and manage stacks, and a centralized CodePipeline that orchestrates deployments.

Project-specific accounts (project1-dev, project1-acc, and project1-prd) utilize these centrally managed resources through CloudFormation StackSets, ensuring consistent infrastructure deployment across environments.

The development and acceptance environments (project1-dev and project1-acc) have their own CloudFormation stacks deployed from the shared pipeline, while the production environment (project1-prd) has a dedicated pipeline for more controlled deployments. This architecture enhances efficiency, scalability, and consistency in managing EKS clusters across multiple AWS accounts.

All ArgoCD instances can run on a dedicated management cluster. This separation of concerns enhances security by isolating the deployment mechanism from the application.

Argo Project, 2024

To get a deeper understanding how these elements work together, let's break down this process by discussing them individually:

CloudFormation StackSets: Centralized Management with a Twist
CloudFormation StackSets allow you to manage AWS resources across multiple accounts and regions through a single operation. By defining your infrastructure as code, you can ensure consistency, repeatability, and scalability in your deployments. With StackSets you can deploy resources in multistage deployment pipelines across your accounts.

Stack Set Instances Deploy Code Pipelines
You can utilize StackSet instances for each project to deploy AWS CodePipeline in the project's production accounts. These pipelines are the arteries through which your code and infrastructure updates flow from source control to each environment. By managing these pipelines through StackSets, you can maintain high control and visibility over the deployment process, ensuring that every account is aligned with your central governance model.

CodePipeline: The Deployment Maestro
AWS CodePipeline orchestrates the deployment of your EKS stacks across development, staging, and production accounts.

Each pipeline is configured to trigger automatically upon code commits, pulling in the latest changes and initiating the deployment process. This automation ensures that your environments are always up-to-date with the latest codebase, minimizing manual intervention and accelerating the delivery of new features and updates.

A Three-Stage Deployment Process
The deployment process is meticulously structured into three stages, reflecting your commitment to quality and reliability:

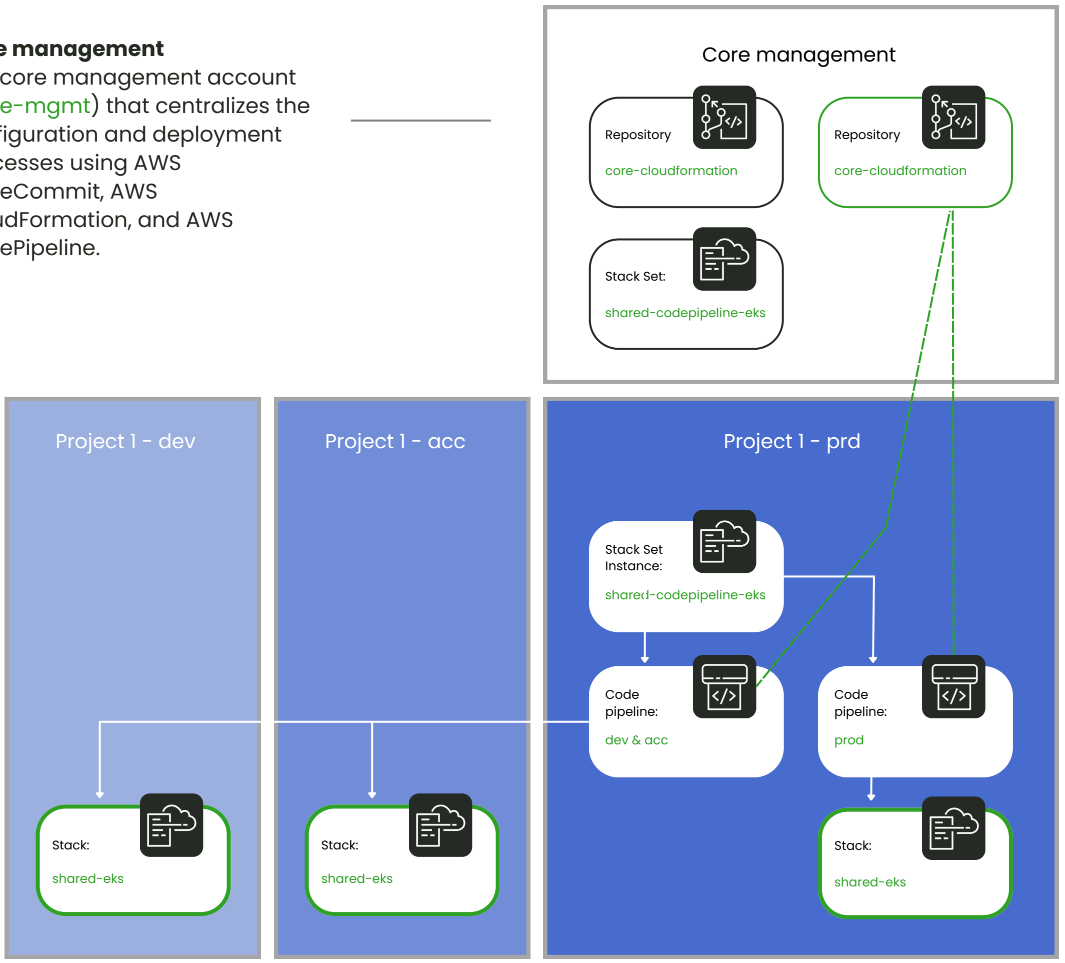
Development: The first stage targets your development accounts, where new features and updates are deployed for initial testing and validation. Most heavy lifting occurs in this environment, with developers pushing changes frequently to iterate rapidly on new ideas.

Staging: Once changes have passed initial tests in the development stage, they move to the staging environment. This stage serves as a pre-production checkpoint, where deployments are scrutinized under conditions that closely mimic the production environment. It's a critical step for catching any issues before they impact users, e.g., load testing.

Production: The final stage is the deployment to production accounts, where the changes become available to end-users. Deployments to production are handled with extra care, often requiring manual approval to proceed. This stage embodies your commitment to delivering stable, reliable software to your users, ensuring that new features and updates enhance their experience without disruption.

Architecture for infrastructure pipelines

Core management
The core management account (core-mgmt) that centralizes the configuration and deployment processes using AWS CodeCommit, AWS CloudFormation, and AWS CodePipeline.



AWS CodeCommit
Stores infrastructure-as-code for CloudFormation templates.

Developer platform
The development and acceptance environments (project1-dev and project1-acc) have their own CloudFormation stacks deployed from the shared pipeline, while the production environment (project1-prd) has a dedicated pipeline for more controlled deployments.

AWS CodePipeline
Automates deployment workflows between environments.

AWS CloudFormation
Deploys and manages shared EKS infrastructure across accounts.

Scaling your infrastructure
This next step involves adopting a robust, centralized management approach to ensure consistency, control, and scalability.

The Outcome: Efficiency, Consistency, and Speed
By leveraging CloudFormation StackSets and AWS CodePipeline, you can achieve a level of efficiency and consistency previously unattainable. The infrastructure pipelines automate the heavy lifting of deployment and management, freeing your client's teams to focus on innovation and development. Moreover, this approach has significantly accelerated your ability to deliver new features and updates, keeping you agile and responsive in a competitive landscape.

Structured Branching Strategy for Environment Separation
Managing and automating the deployment of Amazon Elastic Kubernetes Service (EKS) environments necessitates a structured and strategic approach designed to balance agility with stability across various stages of development. At the heart of this strategy lies a branching methodology that segments the infrastructure codebase into distinct branches, each tailored to specific environments.

This includes a dedicated branch for the *Development* and *Staging* environments, which facilitates rapid development and testing, alongside a separate branch for the *Production* environment, which ensures that changes deployed here are rigorously tested and approved.

This clear separation allows for precise governance and control over deployments. It aligns with the best practices for infrastructure as code (IaC), ensuring what gets deployed and when it is managed.

The deployment process is further refined by an automated system with built-in manual checkpoints. For Development and Staging environments, deployments are triggered automatically by changes to their respective branches, supporting a fast-paced development cycle that enables teams to iterate on new features and fixes swiftly. Conversely, the Production environment introduces a manual approval step, a critical checkpoint. This step is crucial for ensuring that all changes destined for production are thoroughly reviewed and vetted, thereby minimizing the risk of disruptions to end users and maintaining the integrity and reliability of the production environment.

Local Configuration Management
The AWS Systems Manager Parameter Store manages local configuration details. This approach centralizes configuration management, allowing for secure storage and easy retrieval of configuration parameters. It enables development teams to adjust their environment configurations without altering the core deployment scripts, adding a layer of flexibility and control.

Deployment Automation via CodeBuild
The deployment process leverages AWS CodeBuild, a fully managed continuous integration service, to execute the deployment jobs. These jobs are responsible for configuring and bootstrapping the EKS clusters and preparing the stage for ArgoCD.

The Outcome: A Streamlined Deployment Pipeline
By employing this structured approach to EKS deployment, the team achieves several key outcomes:

- Efficiency: Automated deployments to development and staging environments accelerate the development cycle, allowing teams to focus on building and testing rather than manual deployment processes.
- Stability: The manual approval process for production deployments ensures that only thoroughly tested and reviewed changes make it to the live environment, maintaining the stability and reliability of the production services.
- Flexibility: Managing local configurations through the Parameter Store and automating cluster configuration with *eksctl* and *kubectl* allows teams to tailor their environments without compromising security or compliance.

4 →

EKS cluster management

Advanced technologies to use through ArgoCD

Networking with VPC CNI

First up is VPC CNI. Networking is the lifeblood of any Kubernetes cluster, and the AWS VPC CNI plugin is a cornerstone. It can integrate your clusters seamlessly with AWS's scalable networking infrastructure, ensuring your pods are first-class citizens within the VPC and benefit from native AWS features like security groups and VPC flow logs.

Storage Solutions: EFS CSI and EBS CSI

Storage can be another critical component of your infrastructure. You can utilize both the EFS CSI and EBS CSI drivers to provide your applications with the flexibility and performance they need. EFS for shared, scalable file storage and EBS for block storage, ensuring your applications have access to the persistent storage solutions they require, optimized for their specific needs.

Autoscaling: From Cluster Autoscaler to Karpenter

Autoscaling is where you can genuinely embrace the dynamic nature of cloud-native applications. Starting with the Cluster Autoscaler, you can easily manage the scale of your nodes based on demand, ensuring you are as cost-effective as you are efficient. Karpenter takes this further by offering more responsive and faster scaling alongside the Vertical Pod Autoscaler (VPA) and Horizontal Pod Autoscaler (HPA), which adjust your pods' resources and replicas to meet the current needs, ensuring your applications are always running optimally.

Logging and Metrics with Datadog, Fluentd, and Cloudwatch

Visibility into your operations is a must. Datadog, Fluentd, and Cloudwatch provide a comprehensive view of your environment. Datadog for its powerful monitoring capabilities, Fluentd for its log aggregation, and Cloudwatch for its seamless integration with AWS services, ensuring you have all the metrics and logs at your fingertips to make informed decisions.

Advanced tools for streamlined EKS cluster management

In this chapter, we highlight the range of technologies to use through ArgoCD (see next section) to streamline operations. These tools facilitate continuous deployment, configuration management, and scalability of your EKS clusters. By automating these processes, you reduce manual intervention, minimize errors, and maintain consistency across your environments.

Networking with VPC CNI

First up is VPC CNI. Networking is the lifeblood of any Kubernetes cluster, and the AWS VPC CNI plugin is a cornerstone. It can integrate your clusters seamlessly with AWS's scalable networking infrastructure, ensuring your pods are first-class citizens within the VPC and benefit from native AWS features like security groups and VPC flow logs.

Storage Solutions: EFS CSI and EBS CSI

Storage can be another critical component of your infrastructure. You can utilize both the EFS CSI and EBS CSI drivers to provide your applications with the flexibility and performance they need. EFS for shared, scalable file storage and EBS for block storage, ensuring your applications have access to the persistent storage solutions they require, optimized for their specific needs.

Autoscaling: From Cluster Autoscaler to Karpenter

Autoscaling is where you can genuinely embrace the dynamic nature of cloud-native applications. Starting with the Cluster Autoscaler, you can easily manage the scale of your nodes based on demand, ensuring you are as cost-effective as you are efficient. Karpenter takes this further by offering more responsive and faster scaling alongside the Vertical Pod Autoscaler (VPA) and Horizontal Pod Autoscaler (HPA), which adjust your pods' resources and replicas to meet the current needs, ensuring your applications are always running optimally.

Challenges of Managing Kubernetes Clusters
Managing multiple Kubernetes clusters presents a significant challenge, primarily due to operational overheads, complexity, and the steep learning curve associated with Kubernetes' complex ecosystem. As organizations scale and deploy across various environments; production, staging, and development, the need for a robust tool to streamline management becomes crucial. Learn more about this via our blog:

Managing Ingress with NGINX and AWS LB Controller

Ingress management is handled using NGINX and the AWS Load Balancer Controller. NGINX offers flexibility and powerful routing capabilities, while the AWS LB Controller allows you to leverage AWS's native load balancing solutions, ensuring your services are always accessible and performant.

The AWS Load Balancer Controller and NGINX manage traffic routing and contribute to your security strategy. The AWS Load Balancer Controller can also be integrated with security-focused features like rate limiting, IP whitelisting, and WAF capabilities to protect against common web vulnerabilities.

Streamlined Deployments with Helm and ArgoCD

Deployment strategies are an area you can mainly focus on optimizing. Helm charts allow you to package and deploy applications consistently. At the same time, ArgoCD enables a GitOps approach to continuous delivery, ensuring your deployments are automated, auditable, and aligned with the infrastructure as code (IaC) philosophy.

5

Deployment with ArgoCD

Enhance automation and control

Deployment with ArgoCD

After establishing a structured approach to managing EKS clusters using integrated CloudFormation StackSets and AWS CodePipeline, the next step involves implementing advanced deployment strategies to enhance automation and control. By adopting GitOps workflows with ArgoCD, you can further streamline the management of Kubernetes clusters across multiple environments.

Diagram 3 illustrates a GitOps workflow using ArgoCD to manage Kubernetes clusters across DEV, ACC, and PRD environments. Centralized Git repositories store Kubernetes manifests and configurations, serving as the source of truth. The management cluster runs ArgoCD instances for each environment, continuously synchronizing cluster states with the repository, ensuring consistency.

Project-specific clusters (DEV, ACC, PRD) run various drivers and agents, with ArgoCD automating the deployment of updates from Git. This setup exemplifies a structured approach, ensuring reliable and consistent management of Kubernetes clusters across different environments through automated synchronization and version control.

How to integrate ArgoC

With the theoretical model behind managing Kubernetes explained, you might wonder how to integrate ArgoCD into your ecosystem and the benefits it can bring to your operations? In the following subsections, we carefully explain to you what practicalities are involved in doing so.

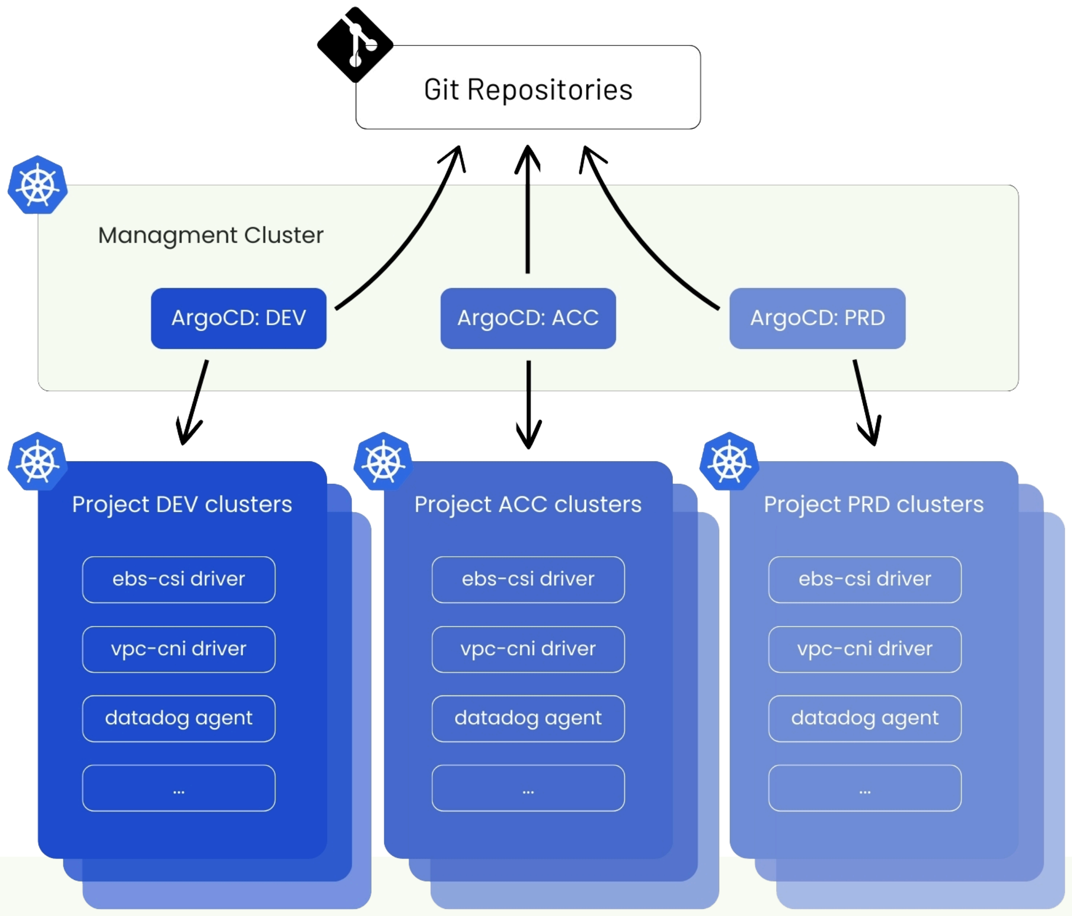


Diagram 3: deployment with ArgoCD

Tailored ArgoCD installations for each environment

Understanding each environment's unique demands and requirements — development, staging, and production — you can opt for separate ArgoCD installations for each. This approach allows you to tailor the deployment strategies and permissions according to these environments' specific needs and security protocols. By segregating the installations, you can ensure that the configurations and resources are optimized for the tasks at hand, whether rapid iteration in development or stability and security in production.

Centralized anagement through a separate cluster

All ArgoCD instances can run on a dedicated management cluster. This separation of concerns enhances security by isolating the deployment mechanism from the application workloads and provides a centralized point of control for managing deployments across all environments. The management cluster acts as the command center, where you can orchestrate your deployments, monitor their status, and enforce compliance and governance policies.

Automatic synchronization with Git

One core principle of GitOps is managing infrastructure and application configurations as code stored in Git repositories. ArgoCD excels in this area by automatically synchronizing the declared state in your Git repositories to all configured clusters. Whenever a change is committed to a repository, ArgoCD detects the update and applies it across the relevant environments, ensuring that your live configurations always match the intended state declared in Git. This automation reduces the potential for human error and significantly accelerates your deployment processes.

Environment-specific configurations

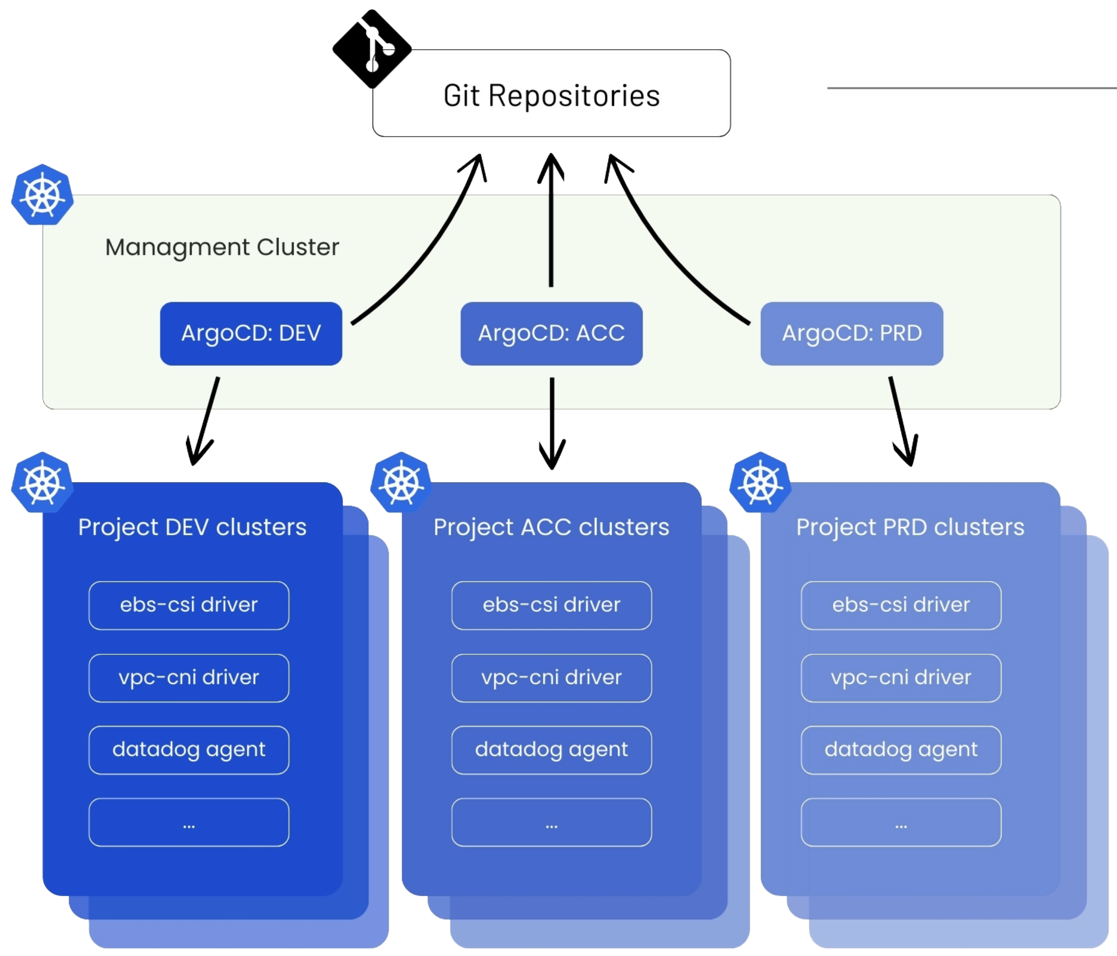
ArgoCD's flexibility lets you specify versions, parameters, and configurations unique to each environment directly within your Git repositories. This capability is crucial for managing the nuances between environments, such as scaling parameters, resource allocations, and feature toggles. By defining these configurations in Git, you can maintain a single source of truth for your infrastructure and application states. This simplifies audits and rollbacks and enables precise control over what gets deployed, where, and when.

The Impact of ArgoCD on your operations

Adopting ArgoCD can mark a significant milestone in your journey toward a more automated, secure, and efficient deployment pipeline. It can streamline your operations and boost your confidence in the reliability and consistency of your deployments. With ArgoCD, you can achieve operational maturity that allows you to focus more on innovation and less on the mechanics of deployment, knowing that your infrastructure and applications are always in sync with your intentions.

As you continue to evolve and scale your operations, tools like ArgoCD will remain central to your strategy. They enable you to navigate the complexities of modern cloud-native development with agility and assurance.

Kubernetes clusters across environments



Git Repositories
Stores the infrastructure and application configuration files that ArgoCD uses to deploy resources across clusters.

Management Cluster
ArgoCD is a continuous delivery tool that automates the deployment of applications and infrastructure to Kubernetes clusters based on GitOps principles.

3. Project DEV, ACC, and PRD Clusters
Each environment (DEV, ACC, PRD) hosts its own set of Kubernetes clusters, ensuring isolation and proper environment-specific configurations

4. ebs-csi driver, vpc-cni driver, datadog agent
These are key components for the clusters: EBS CSI driver for persistent storage, VPC CNI driver for networking, and Datadog agent for monitoring and logging.

Adopting GitOps workflows with ArgoCD
Deployment in ArgoCD syncs changes from Git repositories to Kubernetes clusters. It ensures the cluster state matches the Git configuration.

6

Mastering autoscaling

Nodes and pods in harmony

Mastering autoscaling

Now you've successfully deployed you set of operators through ArgoCD, Autoscaling emerges as a key instrument for seeking operational excellence within your Kubernetes clusters, ensuring your infrastructure is responsive and efficient. This capability is twofold, encompassing the scaling of nodes — the physical or virtual servers that host your pods — and the pods themselves.

Node Autoscaling: ensuring Infrastructure Elasticity
A node autoscaling strategy is underpinned by two key components: you can start with the Cluster Autoscaler and later migrate to Karpenter.

The Cluster Autoscaler dynamically adjusts the number of nodes in a cluster based on the demand, effectively scaling the cluster up when the workload increases and down when it decreases.

This ensures that your cluster is never over-provisioned or under-resourced, aligning our infrastructure costs closely with actual usage.

Karpenter represents an evolution in node autoscaling, offering more rapid and efficient scaling decisions than the Cluster Autoscaler. It proactively launches and terminates nodes to match application demands, and its ability to make fine-grained decisions based on pod requirements allows for more efficient workload packing. This reduces costs and decreases the time it takes for pods to start running, enhancing the overall responsiveness of your applications.

Pod Autoscaling: adapting to workload variability
At the pod level, you can employ the Horizontal Pod Autoscaler (HPA) and the Vertical Pod Autoscaler (VPA) to ensure your applications can handle varying loads without manual intervention.

Karpenter represents an evolution in node autoscaling, offering more rapid and efficient scaling decisions than the Cluster Autoscaler. This reduces costs and decreases upstart time for pods.

Managing EKS at Scale, 2024

The HPA adjusts the number of pod replicas in a deployment or replica set based on observed CPU utilization or other selected metrics. This horizontal scaling approach is crucial for stateless applications that can be easily replicated to meet demand.

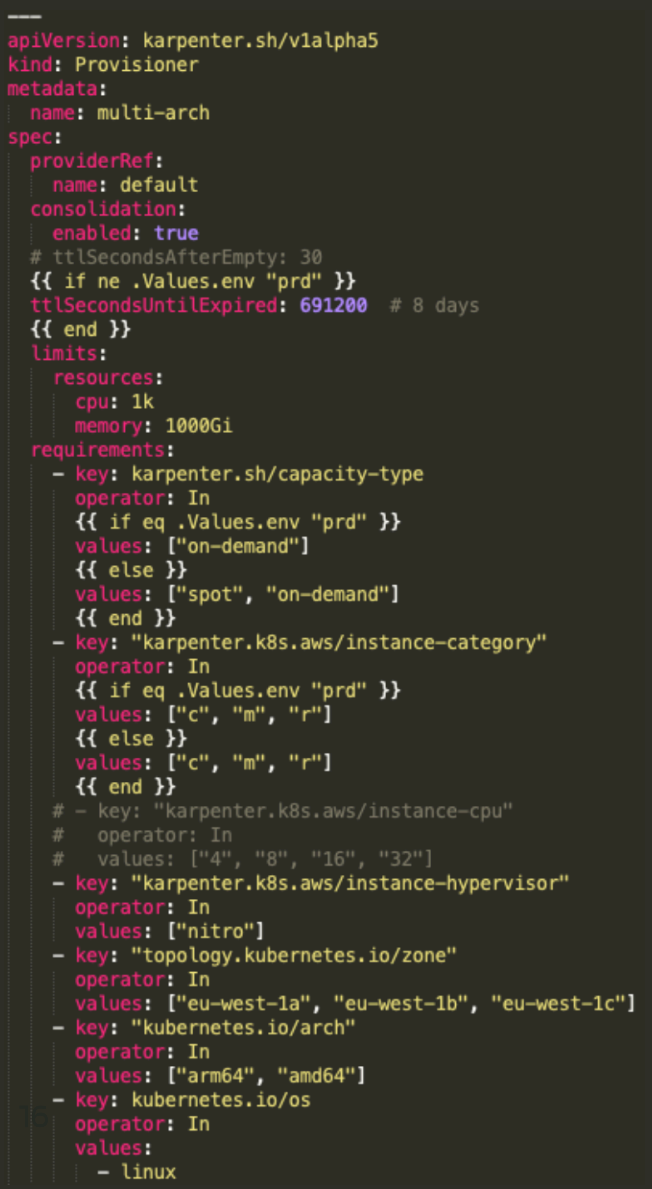
On the other hand, the VPA adjusts pods' CPU and memory reservations in a deployment, ensuring that each pod has the resources it needs without wasting infrastructure capacity. This vertical scaling approach is particularly beneficial for stateful applications or those with unpredictable resource requirements, as it allows each pod to be right-sized based on its workload.

The synergy of node and pod autoscaling

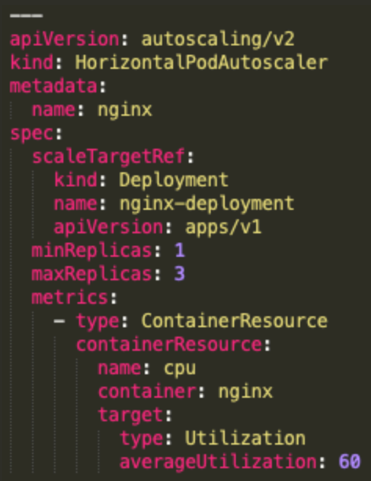
The combination of node and pod autoscaling ensures that your Kubernetes infrastructure is agile and cost-effective. By dynamically adjusting the number of nodes and the size or number of pods, you can ensure that your applications always perform optimally, regardless of the workload.

This autoscaling synergy enhances your operational efficiency and supports your sustainability goals by minimizing resource wastage. It exemplifies your commitment to leveraging cloud-native technologies to build a resilient, scalable, and efficient digital infrastructure.

Autoscaling example



```
---
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: multi-arch
spec:
  providerRef:
    name: default
  consolidation:
    enabled: true
  # ttlSecondsAfterEmpty: 30
  {{ if ne .Values.env "prd" }}
  ttlSecondsUntilExpired: 691200  # 8 days
  {{ end }}
  limits:
    resources:
      cpu: 1k
      memory: 1000Gi
  requirements:
    - key: karpenter.sh/capacity-type
      operator: In
      {{ if eq .Values.env "prd" }}
      values: ["on-demand"]
      {{ else }}
      values: ["spot", "on-demand"]
      {{ end }}
    - key: "karpenter.k8s.aws/instance-category"
      operator: In
      {{ if eq .Values.env "prd" }}
      values: ["c", "m", "r"]
      {{ else }}
      values: ["c", "m", "r"]
      {{ end }}
    # - key: "karpenter.k8s.aws/instance-cpu"
    #   operator: In
    #   values: ["4", "8", "16", "32"]
    - key: "karpenter.k8s.aws/instance-hypervisor"
      operator: In
      values: ["nitro"]
    - key: "topology.kubernetes.io/zone"
      operator: In
      values: ["eu-west-1a", "eu-west-1b", "eu-west-1c"]
    - key: "kubernetes.io/arch"
      operator: In
      values: ["arm64", "amd64"]
    - key: kubernetes.io/os
      operator: In
      values:
        - linux
```



```
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx
spec:
  scaleTargetRef:
    kind: Deployment
    name: nginx-deployment
    apiVersion: apps/v1
  minReplicas: 1
  maxReplicas: 3
  metrics:
    - type: ContainerResource
      containerResource:
        name: cpu
        container: nginx
        target:
          type: Utilization
          averageUtilization: 60
```

Putting the theory to work

The YAML configuration file above is for a Karpenter Provisioner, which manages the dynamic provisioning of nodes in a Kubernetes cluster. It specifies the API version (karpenter.sh/v1alpha5), resource type (Provisioner), and includes metadata such as the provisioner name (multi-arch). The spec section details the provider reference, consolidation settings, and node expiration time, conditionally set to 8 days for non-production environments.

The configuration also sets resource limits (1 CPU and 1000Gi memory) and various requirements for the nodes, including capacity type (differentiating between production and other environments), instance categories, hypervisor type (nitro), availability zones, CPU architectures (arm64 and amd64), and operating system (linux). This ensures nodes meet specific criteria for efficient and tailored resource management.

7

Enhancements & optimizations

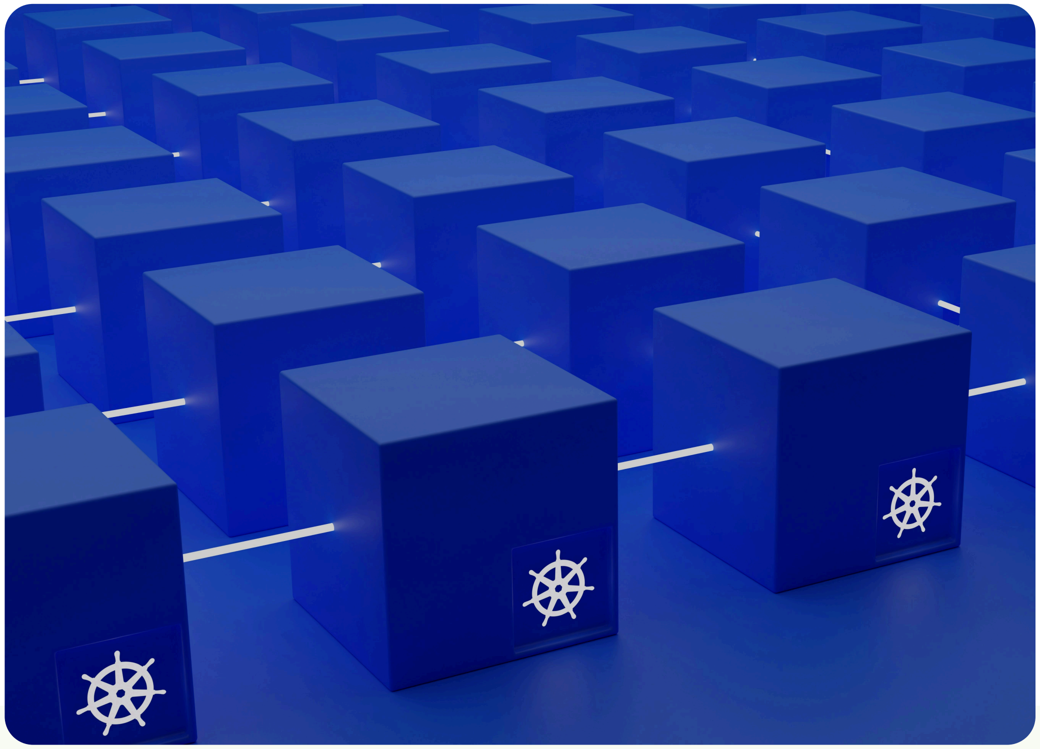
Defining an Ingress resource with appropriate annotations

Enhancements and optimizations

To effectively manage and route traffic in a Kubernetes cluster, leveraging an AWS Application Load Balancer (ALB) Ingress Controller is highly recommended. This setup involves defining an Ingress resource with appropriate annotations to configure the ALB's behavior, such as specifying it as internet-facing, setting target types to IP, and defining SSL policies and health check protocols. Additionally, the Ingress rules direct traffic to backend services based on specified hosts and paths, ensuring secure and efficient traffic management.

Pairing the Ingress resource with a corresponding Service resource that selects backend pods and exposes necessary ports completes the configuration. This approach allows seamless handling of incoming traffic, SSL termination, and health checks, providing a robust and scalable solution for managing Kubernetes services. By implementing these configurations, you ensure reliable and secure traffic routing, optimizing your cluster's performance and maintainability.

Let's discuss in-depth how you can optimize your setup.



The Transition to AWS Load Balancer Controller

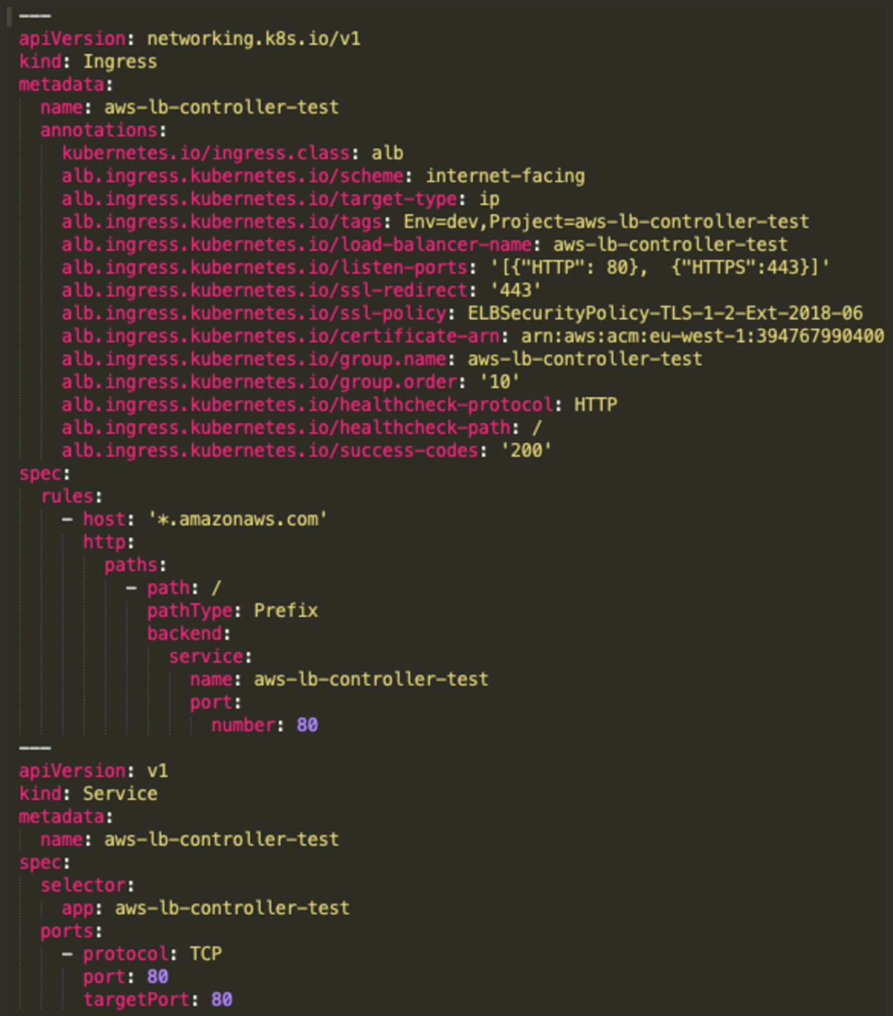
Use AWS Load Balancer Controller to streamline the ingress management. This tool dynamically provisions and manages AWS Elastic Load Balancers (ELB) directly from within the Kubernetes clusters. This has the following benefits:

Fine-grained control
With the AWS Load Balancer Controller, you can specify detailed configurations for each load balancer, tailoring them to your applications' specific needs. Depending on the use case, this includes selecting between Application Load Balancers (ALB) and Network Load Balancers (NLB).

Enhanced security
Integrating seamlessly with AWS WAF, the controller enables robust security rules at the load balancer level. This capability is crucial for protecting your applications from web-based threats without the complexity of managing security at the ingress controller level.

Simplified certificate management
The controller automates the provisioning and renewal of SSL certificates through AWS Certificate Manager (ACM), significantly reducing the operational hassle associated with SSL management. This automation ensures that applications are always served over secure connections with minimal manual intervention.

Scalability and reliability
Leveraging AWS's global infrastructure, the AWS Load Balancer Controller ensures your applications are highly available and scalable. It automatically adjusts the load balancing capacity based on incoming traffic, ensuring our applications remain responsive under varying load conditions.



```yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: aws-lb-controller-test
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/tags: Env=dev,Project=aws-lb-controller-test
    alb.ingress.kubernetes.io/load-balancer-name: aws-lb-controller-test
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80},  {"HTTPS":443}]'
    alb.ingress.kubernetes.io/ssl-redirect: '443'
    alb.ingress.kubernetes.io/ssl-policy: ELBSecurityPolicy-TLS-1-2-Ext-2018-06
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:eu-west-1:394767990400
    alb.ingress.kubernetes.io/group.name: aws-lb-controller-test
    alb.ingress.kubernetes.io/group.order: '10'
    alb.ingress.kubernetes.io/healthcheck-protocol: HTTP
    alb.ingress.kubernetes.io/healthcheck-path: /
    alb.ingress.kubernetes.io/success-codes: '200'
spec:
  rules:
    - host: '*.amazonaws.com'
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: aws-lb-controller-test
                port:
                  number: 80
---
apiVersion: v1
kind: Service
metadata:
  name: aws-lb-controller-test
spec:
  selector:
    app: aws-lb-controller-test
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Optimizing application resilience with (anti-) affinity rules

Ensuring that application pods are evenly distributed across available resources is paramount in Kubernetes environments. This distribution impacts the performance and reliability of your applications and plays a critical role in disaster recovery and high-availability strategies. To achieve this, leveraging Kubernetes' (anti-)affinity rules is a powerful feature that allows you to fine-tune how pods are scheduled and placed within your clusters.

(Anti-)affinity rules in Kubernetes allow you to specify how pods should be co-located or spread out across the cluster. By defining affinity rules, you can instruct the scheduler to place pods based on various criteria, such as proximity to other pods, specific node characteristics, or even across different availability zones for higher resilience.

Anti-affinity rules allow you to ensure that pods are not placed on the same node or within the same zone, which is crucial for avoiding single points of failure. This is particularly important for stateful applications or those with high availability requirements, as it minimizes the risk of simultaneous downtime.

Practical Benefits
Strategically spreading pods can optimize resource utilization across the cluster, preventing scenarios where specific nodes are overburdened while others are underutilized. This enhances the overall performance of your applications and contributes to cost efficiency by ensuring that we're making the most out of your allocated resources on AWS.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: nodegroup
              operator: In
              values:
                - default
            - key: topology.kubernetes.io/zone
              operator: In
              values:
                - eu-west-1a
```

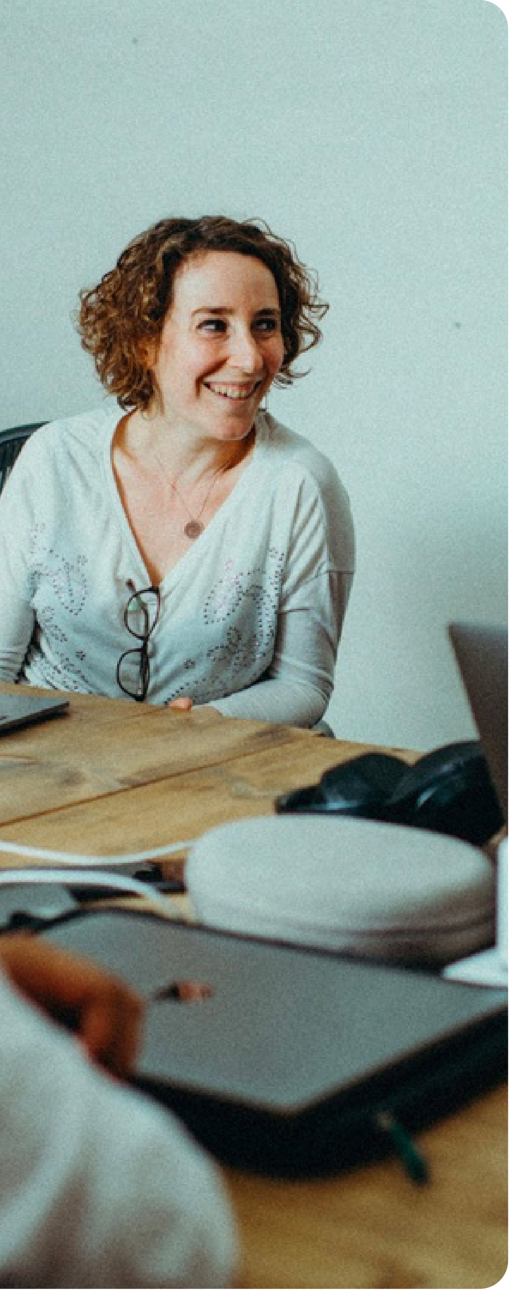
Implementing (Anti-)Affinity Rules
Implementing these rules involves defining specific labels and selectors within your pod specifications. For instance, you might use anti-affinity rules to ensure that pods belonging to the same application tier are not scheduled on the same physical host, thereby reducing the risk of correlated failures.
Additionally, the flexibility of these rules allows you to make dynamic adjustments based on your evolving needs. Whether scaling up during peak demand or deploying new services, you can rely on (anti-)affinity rules to maintain your desired state of distribution and resilience.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 6
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: nodegroup
                    operator: In
                    values:
                      - default
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchLabels:
                  app: nginx
              topologyKey: kubernetes.io/hostname
```

8 →

Monitoring with Datadog

Providing a robust and scalable solution for managing EKS



Monitoring with Datadog

As your EKS clusters grow and become more complex, and are hosting numerous of applications, ensuring their smooth operation becomes increasingly critical. In this section we would like to let you understand the importance of integrating a robust monitoring solution like DataDog with your Karpenter-managed Kubernetes environment.

Effective monitoring is not just about tracking resource usage; it's about gaining comprehensive visibility into your infrastructure, detecting anomalies, optimizing performance, managing costs, and ensuring compliance. By the end of this chapter, you'll understand why monitoring is essential for maintaining a high-performing, efficient, and scalable EKS infrastructure, and how to leverage DataDog to achieve these goals.

One of Datadog's key strengths is its comprehensive monitoring capabilities. By collecting metrics, logs, and traces, Datadog enables you to monitor your clusters' performance, identify bottlenecks, and detect anomalies in real-time. This level of insight is crucial for proactive issue resolution and optimizing your applications' performance.

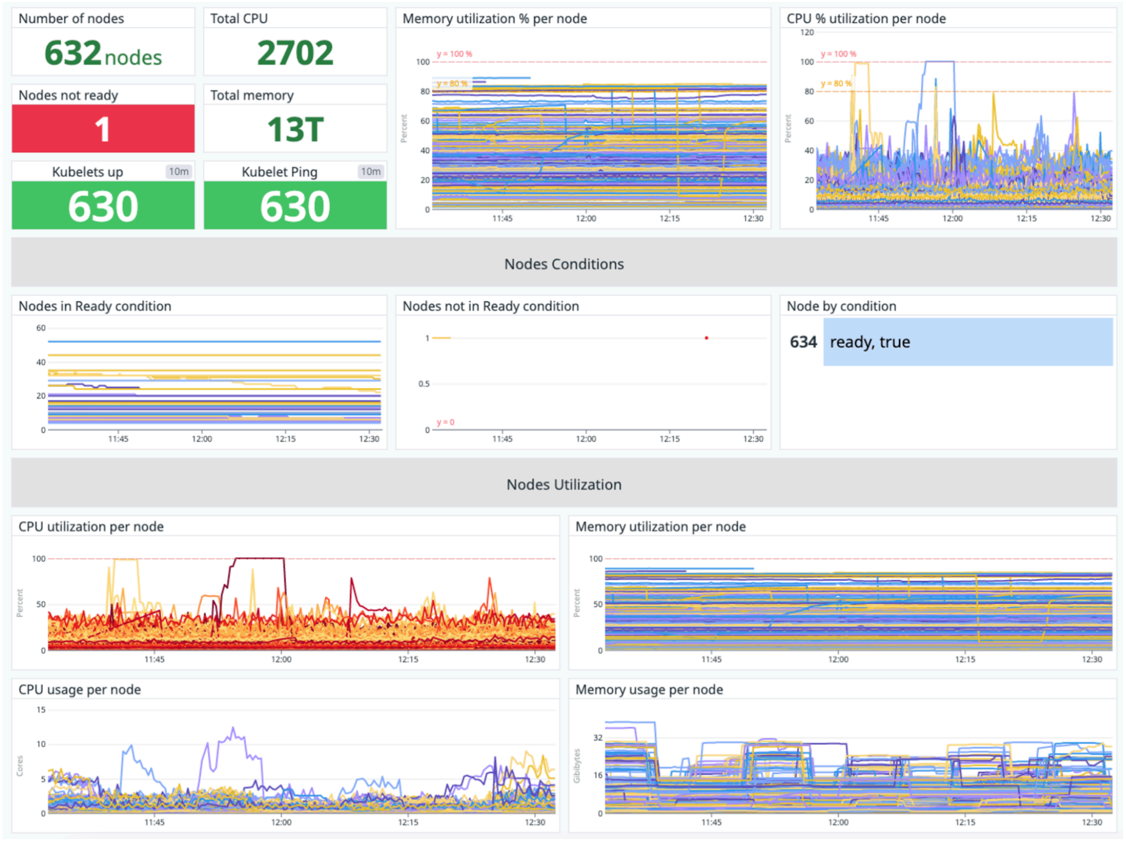
Why Datadog?

Datadog is a monitoring and analytics platform providing real-time insights into your entire stack. Its ability to aggregate metrics and logs from various sources, including Kubernetes clusters, nodes, and pods, gives you a holistic view of your infrastructure and applications.

DataDog can enhance your ability to monitor resource utilization, detect and respond to anomalies, plan capacity, manage costs, and maintain compliance. Comprehensive Monitoring

Enhanced visibility
Datadog's dashboards and alerting mechanisms enhance our visibility into your Kubernetes environments. Customizable dashboards allow you to tailor your monitoring views to your needs, ensuring critical metrics are always front and center. Meanwhile, sophisticated alerting capabilities ensure you are promptly notified of potential issues, allowing swift action to mitigate the impact.

Collaboration and troubleshooting
Datadog also facilitates team collaboration by providing a shared platform for monitoring and troubleshooting. This shared context is invaluable when diagnosing complex issues, enabling faster resolution and minimizing downtime. Moreover, Datadog's historical data and analytics features support your continuous improvement efforts by allowing you to analyze trends and make data-driven decisions.



Integration with Kubernetes
Datadog's seamless integration with Kubernetes is a critical factor in its effectiveness. By automatically discovering and monitoring containerized applications, Datadog simplifies the setup and maintenance of your monitoring infrastructure. This integration extends to various Kubernetes distributions and services, ensuring consistent monitoring across hybrid and multi-cloud environments.

9

The path ahead

Why managing EKS at scale is not just about mastering the tool

The path ahead

The journey to managing Amazon EKS at scale is a multifaceted one, filled with both challenges and opportunities. Throughout this eBook, we've explored the strategies, tools, and best practices that enable organizations to confidently deploy and operate EKS clusters across complex environments.

Starting with the foundations, we demonstrated how a centralized approach to managing infrastructure—while maintaining flexibility for individual teams—lays the groundwork for success. Automation, highlighted as the backbone of scalable operations, ensures consistency and reduces the burden of manual processes. Tools like AWS CloudFormation, CodePipeline, and StackSets were shown to bring structure and efficiency to even the most intricate setups.

We then delved into the powerful role of deployment tools like ArgoCD, which bring the benefits of GitOps to Kubernetes environments. By synchronizing configurations and enabling continuous deployment, ArgoCD transforms the way teams manage their clusters, providing both agility and reliability. Coupled with strategies like environment-specific configurations and a tailored branching model, these tools help bridge the gap between innovation and operational stability.

Autoscaling emerged as a critical capability for ensuring both cost efficiency and performance. The integration of Karpenter, along with tools like Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA), showcased how dynamic scaling at both the node and pod levels can ensure infrastructure elasticity while optimizing resource usage.

The book also addressed the importance of monitoring and visibility. As Kubernetes environments grow, solutions like Datadog offer real-time insights, anomaly detection, and collaborative troubleshooting, ensuring operational excellence while supporting a proactive approach to issue resolution.

Finally, we discussed enhancements and optimizations—covering everything from traffic routing with AWS Load Balancer Controller to leveraging anti-affinity rules for high availability. These strategies not only improve the performance and security of your applications but also ensure your architecture is prepared to scale seamlessly as demands evolve.

As organizations continue to adopt cloud-native technologies, the need for robust, scalable, and efficient infrastructure becomes even more critical. Managing EKS at scale is not just about mastering the tools—it's about creating a culture of innovation and collaboration. Teams empowered with the right frameworks and practices can shift their focus from managing infrastructure to delivering business value.

“This guide’s aim is to ensure that every development team can focus on their applications without being bogged down by the intricacies of the infrastructure that powers them. We shares insights and methodologies that have proven effective in real-world scenarios. Because efficiency and automation are not just goals; they're necessities.”

Jurg van Vliet

CEO of Aknostic