



AWS Journey

From Deploy to Done: AWS Owns Operations for the Life of Your Application



From Deploy to Done: AWS Owns Operations for the Life of Your Application: AWS Journey

Table of Contents

Overview	1
How This Journey Works	1
Quickstart	3
Go Deeper	4
The Operational Ownership Gap	5
Go Deeper	6
Managed Application Platform and Lifecycle Management	7
Go Deeper	9
Workload Fit and AWS Portfolio Comparison	10
Go Deeper	12
Migrating From Comparative Platforms	13
From Heroku, Render, or Railway	13
From On-Premises .NET/Java/PHP	13
From Azure App Service	14
From Self-Managed EC2 Instances	14
Go Deeper	15
Pricing and Cost Modeling	16
Go Deeper	18
Deployment Frameworks	19
Console Deployment	19
AWS CLI and SDKs	19
Infrastructure as Code (Terraform)	19
GitHub Actions	19
Deployment Strategies	20
Go Deeper	21
Additional Resources	22
Related AWS Services	22
Other Related Assets	22
Feedback	23

Overview

Who should read this journey: Full-Stack Leads, Technical Founders/CTOs, .NET/Java Application Owners, Backend Engineers in organizations where application delivery outpaces operational capacity.

Every application your team ships adds operational weight: deployments to orchestrate, patches to apply, scaling to tune, health to monitor. In organizations where application delivery outpaces operational capacity, that weight compounds until infrastructure management consumes more engineering time than product development. The applications keep growing. The operational capacity does not.

AWS Elastic Beanstalk is a managed application platform where you provide the application and AWS takes permanent operational responsibility for the environment that runs it. Deploying, scaling, patching, monitoring, upgrading: continuously, for the life of the application. This journey starts with a hands-on deployment, then explains why operational ownership transfer matters, helps you evaluate whether Elastic Beanstalk fits your workloads and where it sits in the broader AWS portfolio, and gives you practical paths for migration, cost modeling, and CI/CD integration.

Whether you are migrating from a platform that has shifted its development focus, moving established .NET and Java applications off on-premises servers, or replacing hand-managed deployments on self-managed EC2 instances, this journey gives you a structured path to managed application operations.

How This Journey Works

This journey is structured so you experience the platform first, then decide whether it fits, then expand usage across your organization. You start by deploying a sample application (Quickstart), then evaluate the operational model and workload fit, then migrate existing applications, understand cost dynamics, and integrate deployment into your engineering workflows. Each section is self-contained. Enter at the section that matches where you are today.

How This Journey Works

Section	What You Get
Quickstart	Deploy a sample app and see the full experience firsthand

Section	What You Get
The Operational Ownership Gap	Understand the problem this platform solves
Managed Application Platform and Lifecycle Management	What AWS takes operational responsibility for across the application lifecycle
Workload Fit and AWS Portfolio Comparison	Decide whether Elastic Beanstalk fits your workloads, or whether ECS/EKS is the better path
Migrating From Comparative Platforms	Bring existing applications from Heroku, on-premises, or other clouds
Pricing and Cost Modeling	Understand costs: no service fee, pay for resources, portfolio economics
Deployment Frameworks	Integrate with GitHub Actions, Elastic Beanstalk CLI, and CI/CD pipelines

Quickstart

Deploy a sample application to AWS Elastic Beanstalk. The goal is to see the full deployment experience before reading about architecture, positioning, or operational philosophy.

What you will do:

1. Open the [Elastic Beanstalk console](#). If prompted, choose **Switch to new version** to access the latest console experience.
2. Choose **Deploy application**. This process triggers creating the environment and application together.
3. On the **Deployment mode** step, choose **Standard** (a dedicated environment per application, including Windows and .NET Framework workloads) or **Cluster** (multiple applications on shared infrastructure). This choice comes before you configure the application.
4. Name your application. In Standard Mode, select a platform (Node.js, Python, Java, .NET, or another supported runtime). In Cluster Mode, there is no platform to choose; you provide source code, a Dockerfile, or a container image.
5. Use the sample application provided by the console, or upload your own source bundle.
6. Scroll to the bottom of the configuration page. Skip intermediary steps for networking and health monitoring; these can be configured later.
7. Choose **Create**. Elastic Beanstalk creates the application and provisions its first environment. Wait for provisioning to finish.
8. When the health indicator turns green, choose the environment URL. Your application is live.

No VPC design. No load balancer configuration. No IAM policy authoring. The platform provisions networking, compute, scaling, health monitoring, and a public endpoint as part of the creation workflow.

What just happened underneath: Elastic Beanstalk created an Auto Scaling group, configured a load balancer, set up health checks, attached security groups, provisioned compute, deployed your application, and started monitoring. You provided the application. AWS built and now operates the environment.

Important: Application and Environment. An *application* is the logical container for your work; it groups your code versions and the environments that run them. An *environment* is the collection

of running AWS resources that serves one version of your code at a live URL. An application holds one or more environments (for example, one application with separate development, staging, and production environments). You always begin by creating the application; Elastic Beanstalk launches your first environment inside it as part of the same flow, so there is no separate "create an environment" step to do first. In Standard Mode you choose a platform for the environment; in Cluster Mode there is no platform to choose because you provide your application as a container.

Go Deeper

Go Deeper

- [Learn How to Get Started with Elastic Beanstalk \(Console Tutorial\)](#) - Full walkthrough with screenshots
- [QuickStart: Deploy a Node.js Application](#) - Node.js deployment walkthrough
- [QuickStart: Deploy a Docker Application](#) - Container-based quickstart
- [QuickStart: Deploy .NET Core on Linux](#) - .NET deployment path

The Operational Ownership Gap

Now that you have seen the deployment experience, consider why it matters at organizational scale.

A backend engineer spends two days configuring auto-scaling, load balancing, and SSL certificates for a Spring Boot service. The deploy works. Nobody touches it for three months because touching it might break it again. A platform update rolls out and disassociates a configuration. The team discovers it when customers report the application is unreachable: four hours to diagnose what changed. The team sets CPU-based scaling thresholds at deploy time. Six months later, traffic patterns shift. The application drops requests during a spike because nobody recalibrated.

These are not edge cases. They are the first incidents every engineering organization encounters when application delivery outpaces operational capacity.

Elastic Beanstalk prevents these scenarios by managing configuration associations through platform updates (no disassociation on upgrade), and by automatically transitioning scaling signals from static thresholds to traffic-aware responses as usage patterns change.

This is the operational ownership gap. It appears in every growing engineering organization where application delivery outpaces operational capacity, and it widens with every workload added to production.

The cost is not measured in infrastructure spend. It is measured in what teams do not ship. Every sprint that includes "upgrade the deployment pipeline" is a sprint that does not include the feature a customer asked for. Every incident answered by an engineer who has a product standup the next morning is a day of diminished output that never shows up in any dashboard.

What a managed application platform changes:

The platform makes operational decisions on your behalf, before incidents occur. It decides what healthy looks like. It decides what to do when healthy stops being true. It decides how to communicate what happened and what it did about it. All at deploy time, not at incident time. The team that chose that platform does not get paged at 3 AM because the platform already knew what to do.

In the next section, we explore how Elastic Beanstalk's managed platform model addresses these operational gaps for the life of the application.

Go Deeper

Go Deeper

- [What is AWS Elastic Beanstalk?](#) - Service overview and operational model
- [AWS Elastic Beanstalk Use Cases](#) - Traditional migration, container hosting, web app development
- [AWS Elastic Beanstalk FAQs](#) - Pricing, supported platforms, operational scope

Managed Application Platform and Lifecycle Management

Elastic Beanstalk accepts your application as source code, a Dockerfile, or a pre-built container image, and takes ongoing operational responsibility for the running environment. You define the application. AWS handles deploying, scaling, patching, monitoring, and upgrading the platform underneath it. That responsibility does not expire, does not require renewal, and does not shift back to you when the platform evolves.

What AWS takes responsibility for:

- **Deployment orchestration:** Rolling updates with health-aware progression and automatic rollback on failure. A release either succeeds completely or gets reversed automatically.
- **Scaling:** Auto-scaling responds to application-level signals (request rate, queue depth, resource utilization), not static thresholds configured once and forgotten.
- **Patching:** OS patches, runtime updates, and security fixes applied on a managed schedule during configurable maintenance windows. Immutable deployment mechanism ensures zero-downtime patching.
- **Health monitoring:** Continuous monitoring with automated response to degradation. Health events are detected, responded to, and documented automatically.
- **Platform lifecycle:** Infrastructure upgrades happen without migration plans or maintenance sprints. Your application continues running while the platform evolves underneath it.
- **AI-powered troubleshooting:** Environment analysis identifies root causes and suggests remediation, reducing mean time to resolution for application issues.

What you retain control over:

- Application code and dependencies
- Environment configuration (instance type, environment variables, secrets)
- Deployment timing (you trigger deploys; the platform executes them safely)
- Scaling boundaries (you set min/max; the platform manages within those bounds)
- Maintenance window scheduling

Two Deployment Models

Elastic Beanstalk offers two deployment models. Both share the same application-centric interface and the same operational promise. The difference is how the underlying infrastructure is organized:

Two Deployment Models

Dimension	Standard Mode	Cluster Mode
Best for	Windows/.NET Framework workloads on IIS, applications that cannot be containerized, teams that want one dedicated environment per application	Multiple applications that benefit from shared infrastructure, teams consolidating a growing application portfolio
Infrastructure model	Each application gets its own dedicated compute resources	Multiple applications share infrastructure automatically for better efficiency
Deployment speed	Standard provisioning per deployment cycle	Fast one-time initial setup, then significantly faster subsequent deployments
Scaling signals	Infrastructure-level metrics (CPU, network)	Application-level signals (request rate, queue depth)
Cost model	Pay for dedicated resources per environment	Shared infrastructure: cost per application decreases as portfolio grows
Supported inputs	Source code in supported runtimes (Java, Python, Node.js, Ruby, PHP, Go, .NET)	Source code, Dockerfiles, or pre-built container images: any Linux runtime
Observability	CloudWatch integration for metrics, logs, health	Built-in OpenTelemetry with native CloudWatch and third-party compatibility

Scaling signals are specific to each deployment mode and cannot be mixed across modes. Standard Mode scales on infrastructure metrics; Cluster Mode scales on application-level signals.

Both models are production-grade. Standard Mode remains the right choice for Windows/.NET Framework workloads on IIS and for teams that want one dedicated environment per application, where dedicated instances also satisfy per-instance Windows licensing requirements.

Go Deeper

Go Deeper

- [Managed Platform Updates](#) - How automatic patching works
- [Troubleshooting with AI Analysis in AWS Elastic Beanstalk](#) - AI-powered environment diagnostics
- [Deploying with Docker Containers to Elastic Beanstalk](#) - Container platform branches
- [AWS Elastic Beanstalk Introduces Cluster Mode \(AWS News Blog\)](#) - Cluster Mode overview and first look
- [Elastic Beanstalk Compliance Validation](#) - HIPAA, PCI, SOC, FedRAMP eligibility

Workload Fit and AWS Portfolio Comparison

Elastic Beanstalk is one of several AWS compute services. Choosing the right one depends on your workload characteristics, team structure, and where you are in your portfolio growth. This section provides honest positioning across the AWS portfolio.

When Elastic Beanstalk is the right fit:

- Your team ships web applications, APIs, or backend services and wants AWS to own deployment, scaling, patching, and health response for the life of the application
- You provide source code, Dockerfiles, or container images and expect a production environment without configuring the infrastructure layer
- You run fewer than 10 applications and want a single operational model across all of them
- You are migrating from a developer-focused PaaS (Heroku, Render, Railway) that you have outgrown
- You are migrating established applications from on-premises application servers (.NET/IIS, Java/Tomcat, PHP/Apache)
- Your workloads run on standard or accelerated compute and you want AWS to operate the environment rather than manage a custom orchestration layer yourself

When another AWS service is the better fit:

When another AWS service is the better fit

Signal	Better Fit	Why
You need low-level control over custom networking overlays or a service mesh	Amazon ECS or Amazon EKS	These workloads require infrastructure-level control that a managed application platform intentionally abstracts away
Multiple teams need independent deployment pipelines with shared infrastructure governance	Amazon ECS or Amazon EKS with platform engineering investment	Multi-team isolation, RBAC, namespace-level policies require orchestration primitives

Signal	Better Fit	Why
Your workload is event-driven with pay-per-invocation pricing and no long-running processes	AWS Lambda	Event-driven execution model, automatic scaling per request, no server management
You are deploying a JavaScript meta-framework (Next.js, Nuxt, Astro) as a frontend application	AWS Amplify Hosting	Purpose-built for frontend deployment with framework-specific optimizations
You have a dedicated platform engineering team building an internal developer platform	Amazon EKS	Your team has the capacity and intent to own operational tooling
You need dedicated Windows instances for per-instance .NET Framework licensing	Elastic Beanstalk Standard Mode	Dedicated instances satisfy per-instance licensing; full Windows Server/IIS support

Portfolio Maturity: The Graduation Path

Elastic Beanstalk handles production operations for applications at portfolio scale. However, as organizational complexity grows, some workloads will naturally require capabilities beyond what a managed application platform provides:

Portfolio Maturity: The Graduation Path

Portfolio Stage	Characteristics	Right Model
Fewer than 5 workloads	Single team, standard compute, web apps and APIs	Elastic Beanstalk handles everything
5 to 20 workloads	Growing team, containerized services, shared infrastructure benefits	Elastic Beanstalk Cluster Mode, shared infrastructure economics

Portfolio Stage	Characteristics	Right Model
20+ workloads with multi-team isolation, compliance isolation, custom networking	Multiple teams, infrastructure governance	ECS or EKS with dedicated platform engineering investment

This is a natural graduation point. When your organization's operational needs grow to require multi-team isolation or custom networking governance, investing in platform engineering capacity with ECS or EKS is the right decision. Many organizations run Elastic Beanstalk alongside ECS or EKS, using each where it fits best.

Request architecture guidance from an AWS specialist to evaluate your workload portfolio and identify which applications belong on which compute model. [Request a specialist session](#)

Go Deeper

Go Deeper

- [AWS Elastic Beanstalk Use Cases](#) - Where Elastic Beanstalk fits in the compute landscape
- [Migrating .NET Applications: Elastic Beanstalk Hosting Option \(Prescriptive Guidance\)](#) - EB vs ECS vs EKS decision framework for .NET
- [AWS Elastic Beanstalk FAQs](#) - Service boundaries and capabilities

Migrating From Comparative Platforms

If you are moving applications from another platform, this section provides entry paths based on your current environment.

From Heroku, Render, or Railway

Teams commonly migrate from these platforms when they encounter cost scaling pressure at growth, compliance requirements (HIPAA, PCI, FedRAMP) that the platform cannot satisfy, need for VPC network isolation, or portfolio-scale operations (running many applications under one consistent operational model instead of configuring each one separately) that single-app platforms cannot address.

These platforms accept application source code and manage the runtime. Elastic Beanstalk accepts the same inputs: source code with a supported runtime, a Dockerfile, or a pre-built container image. Key differences after migration:

- Elastic Beanstalk environments run in your AWS account (full VPC, IAM, and networking control available if needed)
- No per-dyno or per-service platform fee. You pay for the underlying AWS resources.
- Applications that outgrew single-app platforms gain portfolio-scale operations under one interface
- Compliance eligibility (HIPAA, PCI, SOC, FedRAMP) that developer-focused platforms typically cannot provide

Migration approach: Start with a parallel environment. Deploy your application to Elastic Beanstalk alongside your existing platform. Validate health checks, scaling behavior, deployment rollback, secrets management, and observability. Cut traffic over once validated.

From On-Premises .NET/Java/PHP

Applications running on Windows Server/IIS, Tomcat, or Apache can land on Elastic Beanstalk without re-architecture:

- **Windows/.NET Framework:** Use the `eb migrate` command to discover IIS configurations and deploy directly to Elastic Beanstalk Windows environments

- **.NET Core/6+:** Deploy to Linux-based environments for cost efficiency
- **Java (Spring Boot, Tomcat):** Provide a WAR/JAR file or source. Elastic Beanstalk provisions Tomcat and manages the runtime
- **PHP (Laravel, WordPress):** Provide source. Elastic Beanstalk provisions Apache/nginx and manages the stack

From Azure App Service

Azure App Service and Elastic Beanstalk serve similar operational roles: accept application code, manage the runtime environment. Key architectural differences:

- **Resource visibility:** Elastic Beanstalk environments run in your AWS account with full visibility into underlying resources (EC2, ALB, Auto Scaling groups). App Service abstracts infrastructure entirely.
- **Networking:** Azure VNet integration maps to AWS VPC configuration. Review subnet layouts, private endpoints, and DNS resolution patterns before migrating.
- **Identity:** Azure AD/Entra ID integration maps to AWS IAM and Amazon Cognito. Service-to-service authentication patterns will need re-mapping.

Service mapping: Key Vault to AWS Secrets Manager, Azure Monitor to Amazon CloudWatch, Azure SQL to Amazon RDS, Azure Blob Storage to Amazon S3, Azure Service Bus to Amazon SQS/SNS.

Review your networking, identity, and data dependencies before migrating. The [Prescriptive Guidance pattern for Azure-to-EB migration](#) provides a step-by-step architectural walkthrough.

From Self-Managed EC2 Instances

If your team currently deploys to EC2 instances with hand-configured Auto Scaling groups, load balancers, and patching scripts, Elastic Beanstalk manages all of those components as part of the application environment. You retain the same underlying infrastructure (EC2, ALB, Auto Scaling) but transfer the operational responsibility for orchestrating them.

Get architecture guidance for your specific migration scenario. [Request a specialist session](#)

Go Deeper

Go Deeper

- [Migrate an ASP.NET Web Application to Elastic Beanstalk \(Hands-on Tutorial\)](#) - Step-by-step IIS migration
- [Performing Basic IIS Migrations with eb migrate](#) - CLI-based Windows migration
- [Migrate a .NET Application from Azure App Service to AWS Elastic Beanstalk](#) - Prescriptive Guidance pattern
- [Application Migration Workshop: Replatform with Elastic Beanstalk](#) - Hands-on lab
- [Simplify Application Migrations to AWS with Elastic Beanstalk](#) - Migration walkthrough for PaaS and on-premises workloads

Pricing and Cost Modeling

Elastic Beanstalk has no service fee in either deployment mode. You pay only for the underlying AWS resources your applications consume. Both modes run your application on Amazon EC2 compute, billed at standard EC2 rates, and you can use On-Demand Instances, Reserved Instances, Savings Plans, or Spot Instances.

What you pay for:

What you pay for

Resource	Standard Mode	Cluster Mode
Compute	EC2 instances you select, billed at standard EC2 rates	EC2 compute matched to the CPU and memory you allocate, billed at standard EC2 rates
Load balancing	Application, Network, or Classic Load Balancer for load-balanced environments (a single-instance environment has none)	One Application Load Balancer per environment
Storage	EBS volumes per instance	Container image storage (Amazon ECR)
Managed foundation	None	Two additional charges: a flat Amazon EKS control-plane fee per cluster, and an Amazon EKS Auto Mode management fee on the compute it provisions
Monitoring	CloudWatch (basic instance metrics included; custom metrics and logs at standard rates)	CloudWatch, metered per pod and container; Amazon Managed Service for

Resource	Standard Mode	Cluster Mode
		Prometheus available as a lower-cost alternative at scale

Understanding the Cluster Mode managed foundation charges:

Cluster Mode runs your applications on a managed Amazon EKS foundation, which adds two charges that Standard Mode does not have:

- **Amazon EKS control-plane fee:** a flat hourly fee per cluster for the managed control plane. It does not scale with instance size, node count, or the number of applications. Applications deployed into the same subnet share a cluster, so you generally pay one cluster fee rather than one per application.
- **Amazon EKS Auto Mode management fee:** a fee on top of the EC2 instance cost for provisioning, scaling, and patching the underlying nodes. This management fee is charged independently of your EC2 purchase option, so Reserved Instances, Savings Plans, and Spot discounts reduce the instance cost but not the Auto Mode fee.

When each mode is more cost-efficient:

For a single application at low traffic, Standard Mode on dedicated compute is typically more cost-efficient, because the EKS control-plane fee and Auto Mode management fee add a baseline that a single application cannot offset. AWS guidance places this crossover around workloads spending under \$500 per month. The cost advantage of Cluster Mode emerges at portfolio scale, where multiple applications share managed infrastructure and can use less total compute than running each on its own dedicated instance.

Free tier: The AWS Free Tier offer applies to Standard Mode only, for new AWS accounts during the 6-month Free Tier period. Cluster Mode is not Free Tier eligible.

Cost optimization levers:

- **Reserved Instances, Savings Plans, or Spot Instances** reduce EC2 compute costs for predictable or fault-tolerant workloads (note these discounts apply to the instance cost, not the Cluster Mode Auto Mode management fee)
- **Shared infrastructure in Cluster Mode** lets multiple applications share nodes, so a portfolio can use less total compute than one dedicated instance per application

- **Right-sizing** through managed scaling ensures you are not over-provisioning during low-traffic periods
- **Amazon Managed Service for Prometheus** is available as a lower-cost alternative to CloudWatch for metrics at scale in Cluster Mode
- **No platform fee** means your only costs are the AWS resources your applications actually use

Go Deeper

Go Deeper

- [AWS Elastic Beanstalk Pricing](#) - Official pricing page, including the managed foundation charges
- [Amazon EKS Pricing](#) - EKS control-plane and Auto Mode fee details
- [AWS Elastic Beanstalk FAQs: Pricing Section](#) - Common cost questions

Deployment Frameworks

Once your application runs on Elastic Beanstalk, integrate deployment into your existing engineering workflows.

Console Deployment

The Elastic Beanstalk console provides a step-by-step visual interface for deployment. Upload a source bundle or container image, then monitor deployment progress showing environment health transitions, instance-level status, and real-time event logging via the Deployments tab. View full deployment history with one-click rollback options for any previous version.

AWS CLI and SDKs

The AWS CLI and AWS SDKs let you script environment creation and deployment from your own tooling. Use `aws elasticbeanstalk create-application`, `create-application-version`, and `create-environment` to provision and deploy applications, including registering pre-built container images from Amazon ECR for multi-service architectures.

Infrastructure as Code (Terraform)

For teams managing infrastructure declaratively, the AWS provider for Terraform includes Elastic Beanstalk resources for applications, application versions, and environments, so you can version-control your environment definitions alongside the rest of your infrastructure.

GitHub Actions

The [official GitHub Action for Elastic Beanstalk](#) enables push-to-deploy workflows:

1. Application code merges to your main branch
2. GitHub Actions builds and packages the deployment artifact
3. The action uploads to S3 and triggers an Elastic Beanstalk deployment
4. Elastic Beanstalk manages rolling updates automatically, health checks, and automatic rollback

This eliminates custom deployment scripts while keeping your CI/CD pipeline in GitHub.

Deployment Strategies

Elastic Beanstalk supports multiple deployment strategies, selectable per environment. The strategies available differ by deployment mode: Standard Mode offers all-at-once, rolling, rolling with additional batch, and immutable; Cluster Mode offers all-at-once, rolling, immutable, and traffic-splitting deployments. Confirm the strategies available for your mode in the deployment strategy documentation linked below.

Deployment Strategies

Strategy	Behavior	Available In	Best For
All at once	Deploys to all instances simultaneously	Standard, Cluster	Development environments, fast iteration
Rolling	Deploys in batches, maintaining capacity	Standard, Cluster	Production with tolerance for brief reduced capacity
Rolling with additional batch	Launches new instances before retiring old	Standard	Production where full capacity must be maintained
Immutable	Launches entirely new instances, swaps on health	Standard, Cluster	Production where deployment failure cannot impact running instances
Traffic splitting	Routes a percentage of traffic to new version	Cluster	When you need to validate a new version with a subset of real production traffic before full rollout

All strategies include automatic rollback on health check failure. If a deployment causes health degradation, Elastic Beanstalk reverts to the previous version without manual intervention.

Go Deeper

Go Deeper

- [Deploy to Elastic Beanstalk with GitHub Actions](#) - CI/CD pipeline setup
- [AWS Elastic Beanstalk Now Supports GitHub Actions \(What's New\)](#) - Feature announcement
- [Elastic Beanstalk Deployment Strategies](#) - All-at-once, rolling, immutable, and traffic-splitting deployments
- [Elastic Beanstalk Deployments Tab \(What's New\)](#) - Real-time deployment monitoring
- [Deploy Containers by Using Elastic Beanstalk \(Prescriptive Guidance\)](#) - Docker deployment pattern

Additional Resources

Related AWS Services

[Amazon RDS](#) - Managed relational databases for application backends

[Amazon ElastiCache](#) - In-memory caching for session management and performance

[Amazon SQS](#) - Message queuing for background job processing

[AWS Secrets Manager](#) - Secrets and configuration management integrated with Elastic Beanstalk

[Amazon CloudWatch](#) - Monitoring, logging, and alerting for Elastic Beanstalk environments

[AWS CodePipeline](#) - Continuous delivery beyond GitHub Actions

Other Related Assets

[Migrate, Modernize and Manage .NET Web Applications Workshop \(4 labs\)](#) - Hands-on migration and management workshop

[Monitoring Elastic Beanstalk .NET Applications with CloudWatch and Grafana](#) - Advanced observability setup

[Deploy ASP.NET Core on AWS Elastic Beanstalk \(Video\)](#) - Step-by-step video walkthrough

[Easily Migrate .NET Web Apps to AWS with Elastic Beanstalk, Part 2](#) - Advanced migration patterns

[AWS Elastic Beanstalk AI-Powered Environment Analysis \(What's New\)](#) - AI troubleshooting announcement

[AWS Elastic Beanstalk Introduces Cluster Mode \(What's New\)](#) - Cluster Mode GA announcement

[AWS Services in Scope of Compliance Programs](#) - Full compliance eligibility listing

[QuickStart: Deploy a Docker Compose Application](#) - Multi-container quickstart

Feedback

We value your feedback on this journey. Take a moment to share your experience by completing our [survey](#).