AWS Prescriptive Guidance

Choosing an approach for modernizing .NET applications

AWS Prescriptive Guidance: Choosing an approach for modernizing .NET applications

Table of Contents

Choosing an approach for modernizing .NET applications

Mathew George and Fabian Jahnke, Cloud Application Architects, AWS Professional Services

November 2021 (last update (p. 24): December 2021)

Choosing the right modernization strategy for .NET legacy applications can be a complex decision. This guide provides best practices for technical decision makers who want to understand the approaches for migrating their .NET legacy applications to Amazon Web Services (AWS) and modernizing them in the AWS Cloud.

Modernization involves replatforming or refactoring legacy enterprise applications by combining modern infrastructure, architecture, and organizational patterns. Modernization helps maximize resiliency, engineering efficiency, business agility, and operational excellence.

The .NET developer platform has evolved from the .NET Framework to .NET Core and .NET 6. (.NET 6 offers additional performance improvements. For more information, see Announcing .NET 6 – The Fastest .NET Yet on the Microsoft .NET blog.) You can modernize your .NET legacy applications and take advantage of the performance, cost savings, and robust ecosystem of the Linux operating system, or by switching from .NET Framework to .NET Core or .NET 6.

The best practices provided in this document help guide your .NET application migration and modernization efforts. The guide discusses possible migration and modernization strategies, constraints, and AWS services you can use. Your options include rehosting (lift and shift) your .NET application in the cloud, as well as containerizing, decomposing to microservices, and adopting a serverless architecture.

Decision matrix

The following table summarizes the migration and modernization options for legacy .NET applications, based on your use case and resources.

Use case	Migration strategy and architecture				
	Rehost	**Replatform as a Windows container**	**Re-architect as a Linux container**	**Re-architect as microservices in Linux containers**	**Re-architect as microservices without containers**
You have resources for refactoring.	✗ No	✗ No	✓ Yes	✓ Yes	✓ Yes
Your .NET legacy application is in constant use.	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
You can resolve .NET Framework dependencies.	✗ No	✗ No	✓ Yes	✓ Yes	✓ Yes
You can remove Windows dependencies.	✗ No	✗ No	✓ Yes	✓ Yes	✓ Yes
You want to run your application as a native Windows application on an Amazon Elastic Compute Cloud (Amazon EC2) instance.	✓ Yes	✗ No	✗ No	✗ No	✗ No
Your code can be ported from .NET Framework to .NET Core or .NET 6.	✗ No	✗ No	✓ Yes	✓ Yes	✓ Yes
You want to split your monolithic application.	✗ No	✗ No	✗ No	✓ Yes	✓ Yes

The following sections describe these options in detail:

Rehosting

Rehosting (lift and shift) is the process of migrating your on-premises application to the cloud without modifying it. This strategy is used mostly to migrate large-scale applications to satisfy specific business goals, such as launching a product in an accelerated timeline or leaving an on-premises data center. The applications are rehosted on Amazon Elastic Compute Cloud (Amazon EC2) Windows instances that meet the requirements of the applications you migrate.

Use cases

This migration strategy is useful in any of the following scenarios:

- The legacy .NET application must run as a native Windows application.
- Time and resources to modernize the application are not available.
- The legacy .NET application is a commercial off-the-shelf (COTS) application.

Advantages

Rehosting provides the following benefits, when compared with on-premises .NET applications:

- Minimum effort, because it requires no code or architectural changes
- Reduced cost
- Better compliance and security, because it uses the AWS infrastructure and security best practices

Disadvantages

- Doesn't take full advantage of the performance, scalability, and resiliency options of the AWS Cloud
- Difficult to integrate with state-of-the-art cloud services

AWS services

- Amazon EC2
- AWS Elastic Beanstalk

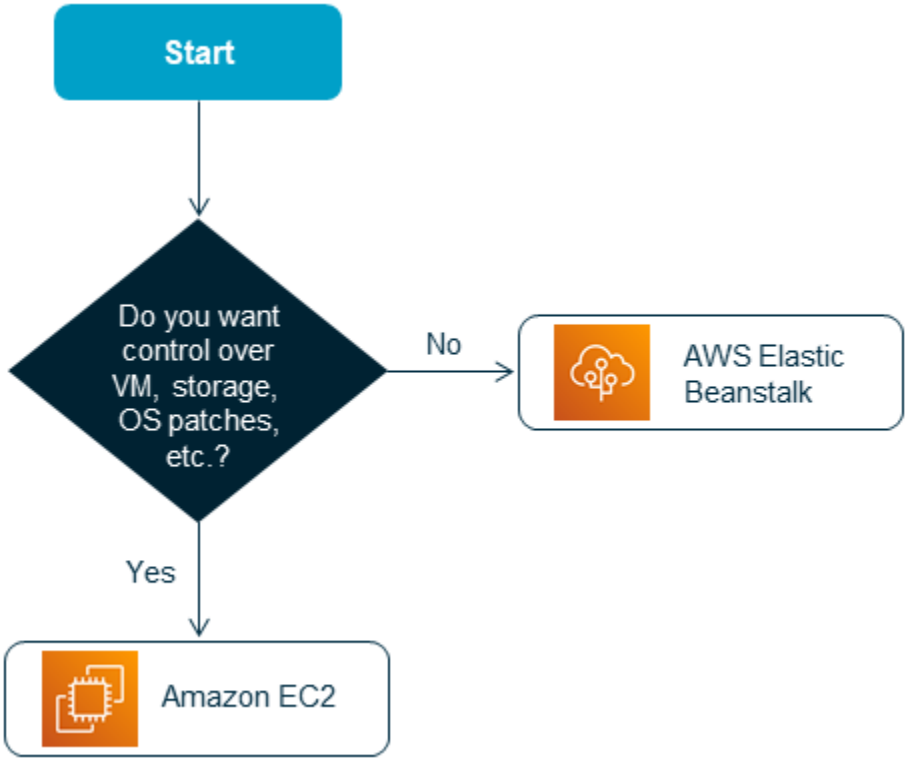
Tools

Tool	Purpose	Resource
Windows Web Application Migration Assistant	This tool is an interactive PowerShell script that migrates entire websites and their configurations to Elastic Beanstalk.	Migrating ASP.NET applications to Elastic Beanstalk (AWS blog post)

Deployment decisions

You can choose from two deployment options:

- If you want complete control over the configuration of your compute environment, including memory and storage settings, and control over operating system patches: migrate your .NET application to Amazon EC2.
- If you don't require full control over the infrastructure: use Elastic Beanstalk. Elastic Beanstalk automatically sets up a managed environment for your application.



Replatforming as a Windows container

Replatforming your .NET application as a Windows container helps you achieve your business objectives with less effort than refactoring. It lets you takes advantage of container technologies without changing the core architecture of your .NET application. Windows applications can be converted to containers without much effort.

.NET Framework-based containers support Windows Server 2016 or 2019 as the host operating system.

Use cases

This migration strategy is useful in any of the following scenarios:

- You're unable to resolve .NET Framework dependencies.
- You're unable to resolve Windows dependencies.
- You don't have the resources to refactor the application to .NET Core or .NET 6.

Advantages

This migration approach provides the following benefits, when compared with on-premises .NET applications:

- Minimal effort
- Improved resource utilization
- Improved security
- Better deployment options

Disadvantages

- License costs for the host Windows operating system

AWS services

For storing container images:

- Amazon Elastic Container Registry (Amazon ECR)

For orchestrating Windows containers:

- Amazon Elastic Container Service (Amazon ECS)
- Amazon Elastic Kubernetes Service (Amazon EKS)
- Amazon EC2 hosting Docker with Windows containers

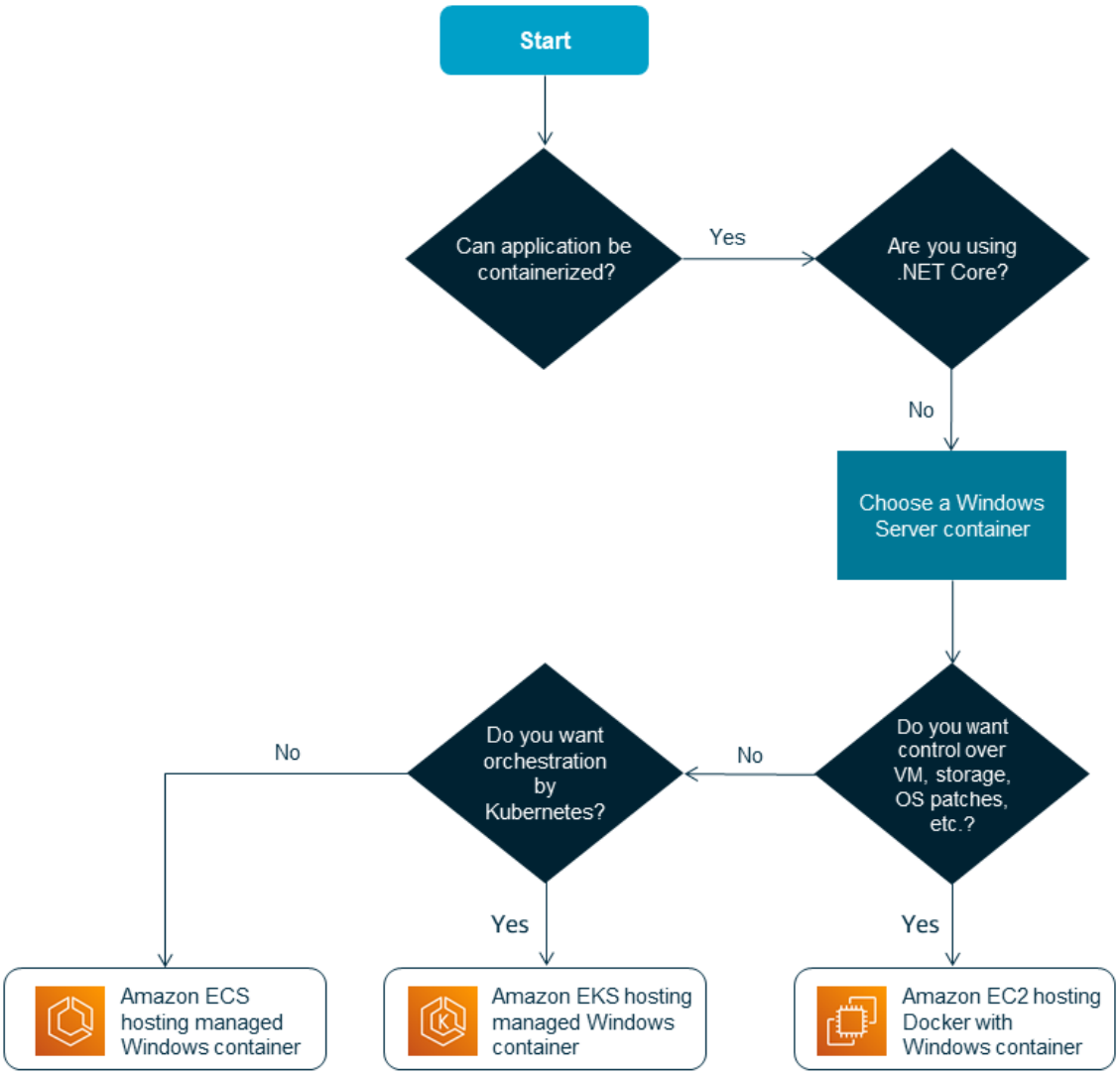
Tools

Tool	Purpose	Resource
AWS App2Container (A2C)	A2C is a command line tool for modernizing .NET and Java applications by converting them into containerized applications with minimal effort.	• Details • Documentation

Deployment decisions

You can choose from three deployment options:

- If you want complete control over the configuration of your compute environment, including memory and storage settings, and control over operating system patches: deploy your application as a Windows container on an EC2 instance.
- If you want the container to be managed by Kubernetes: deploy your application as a Windows container on Amazon EKS.
- If you want the container to be managed by Amazon ECS: deploy your application as a Windows container on Amazon ECS.


Start
Can application be containerized?
Yes
Are you using .NET Core?
No
Choose a Windows Server container
Do you want control over VM, storage, OS patches, etc.?
No
Do you want orchestration by Kubernetes?
No
Yes
Yes
Amazon ECS hosting managed Windows container
Amazon EKS hosting managed Windows container
Amazon EC2 hosting Docker with Windows container

Re-architecting as a Linux container

By porting your .NET Framework applications to .NET Core or .NET 6, you can run your applications on multiple platforms, reduce your license costs, increase performance, and improve scalability.

Use cases

This migration strategy is useful in any of the following scenarios:

- You have the resources and time available to refactor your application.
- You're able to resolve all .NET Framework dependencies.
- You have a long-running application.

Advantages

This migration approach provides the following benefits, when compared with on-premises .NET applications:

- Lower total cost of ownership (TCO)
- Improved security and performance
- Accelerated innovation
- Benefits of converting to cloud-native applications
- Open-source

Disadvantages

- Effort and cost of refactoring

AWS services

For storing container images:

- Amazon ECR

For orchestrating containers:

- Amazon ECS, or Amazon ECS with AWS Fargate
- Amazon EKS, or Amazon EKS with Fargate

AWS Fargate is a serverless, pay-as-you-go compute engine that lets you focus on building applications without managing servers. Fargate is compatible with both Amazon ECS and Amazon EKS.

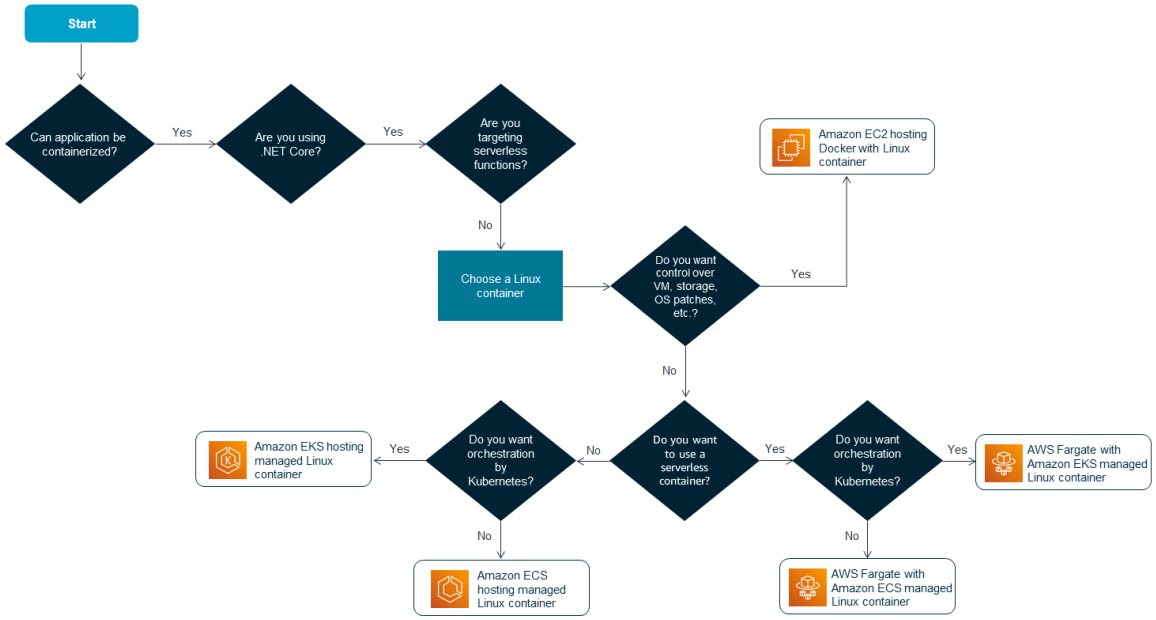
Tools

Tool	Purpose	Resource
Porting Assistant for .NET	This analysis tool scans .NET Framework applications and generates a .NET Core compatibility assessment. The assessment helps you port your applications to Linux faster.	• Details • Documentation
AWS App2Container (A2C)	A2C is a command line tool for modernizing .NET and Java applications by converting them into containerized applications with minimal effort.	• Details • Documentation

Deployment decisions

You can choose from five deployment options:

- If you want complete control over the configuration of your compute environment, including memory and storage settings, and control over operating system patches: deploy your application as a Linux container on an EC2 instance.
- If you want the container to be managed by Kubernetes and run as a serverless container: deploy your application as a Linux container on Amazon EKS with Fargate.
- If you want the container to be managed by Amazon ECS and run as a serverless container: deploy your application as a Linux container on Amazon ECS with Fargate.
- If you want the container to be managed by Kubernetes, but you want to manage the compute resources of the container yourself: deploy your application as a Linux container on Amazon EKS.
- If you want the container to be managed by Amazon ECS, but you want to manage the compute resources of the container yourself: deploy your application as a Linux container on Amazon ECS.


Start
Can application be containerized?
Yes
Are you using .NET Core?
Yes
Are you targeting serverless functions?
No
Amazon EC2 hosting Docker with Linux container
Choose a Linux container
Do you want control over VM, storage, OS patches, etc.?
Yes
No
Amazon EKS hosting managed Linux container
Yes
Do you want orchestration by Kubernetes?
No
Do you want to use a serverless container?
Yes
Do you want orchestration by Kubernetes?
Yes
AWS Fargate with Amazon EKS managed Linux container
No
Amazon ECS hosting managed Linux container
No
AWS Fargate with Amazon ECS managed Linux container

Re-architecting as microservices in Linux containers

A microservices architecture is an approach to developing a single application as a suite of small services. Each service runs in its own process and communicates with other services through lightweight mechanisms. This approach breaks down a monolithic application into smaller services, where each service serves a single purpose and is deployed as a container.

Use cases

This migration strategy is useful if:

- You want to break your monolithic system into microservices.
- You have the resources and time available for refactoring.
- You can resolve all .NET Framework dependencies.
- You have a long-running application.

Advantages

This migration approach provides the following benefits, when compared with on-premises .NET applications:

- Faster innovation because it's easier to add new features in a microservices architecture
- High availability and reliability
- Increased agility and on-demand scalability
- Independent deployment and modern continuous integration and continuous deployment (CI/CD) pipelines
- Strong module boundaries and technical diversity

Disadvantages

- Effort and cost of refactoring
- Operational complexity

AWS services

You can use the following AWS services to develop a microservices-based system:

- Amazon API Gateway
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- Amazon ECS

- Amazon EKS
- AWS Lambda
- AWS Fargate
- AWS CloudFormation or AWS Cloud Development Kit (CDK)
- AWS Identity and Access Management (IAM)
- Amazon Simple Storage Service (Amazon S3)
- Amazon ECR

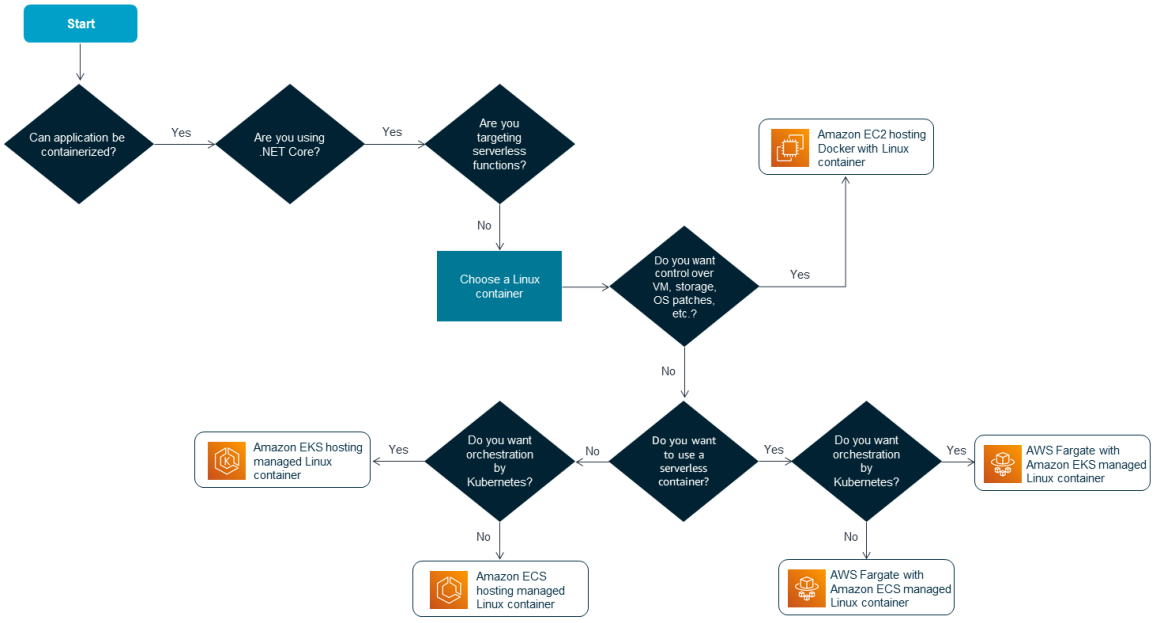
Tools

AWS Professional Services offers custom tools and services to help you refactor your monolithic applications into microservices.

Deployment decisions

You can choose from five deployment options:

- If you want complete control over the configuration of your compute environment, including memory and storage settings, and control over operating system patches: deploy your application as a Linux container on an EC2 instance.
- If you want the container to be managed by Kubernetes and run as a serverless container: deploy your application as a Linux container on Amazon EKS with Fargate.
- If you want the container to be managed by Amazon ECS and run as a serverless container: deploy your application as a Linux container on Amazon ECS with Fargate.
- If you want the container to be managed by Kubernetes, but you want to manage the compute resources of the container yourself: deploy your application as a Linux container on Amazon EKS.
- If you want the container to be managed by Amazon ECS, but you want to manage the compute resources of the container yourself: deploy your application as a Linux container on Amazon ECS.



Re-architecting as microservices without containers

AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes. Lambda runs your function only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time that you consume—there is no charge when your code isn't running. In this approach, a monolithic application is broken down into smaller services, where each service serves a single purpose. If the service isn't constantly running, it can be implemented as a Lambda function; otherwise, the service should run in a container.

Use cases

You can use this migration strategy in the following scenarios:

- You want to break your monolithic system into microservices.
- You have the resources and time available for refactoring.
- You can resolve all .NET Framework dependencies.
- Your applications do not run constantly; they run for a very short period of time.

Advantages

This migration approach provides the following benefits, when compared with on-premises .NET applications:

- Faster innovation because it's easier to add new features in a microservices architecture
- High availability and reliability
- Increased agility and on-demand scalability
- Independent deployment and modern CI/CD pipelines
- Strong module boundaries and technical diversity
- Cost savings
- Reduced infrastructure provisioning efforts

Disadvantages

- Effort and cost of refactoring
- Operational complexity
- No support for long-running applications

AWS services

These are some of the important AWS services that you can use to develop a microservices architecture with AWS Lambda:

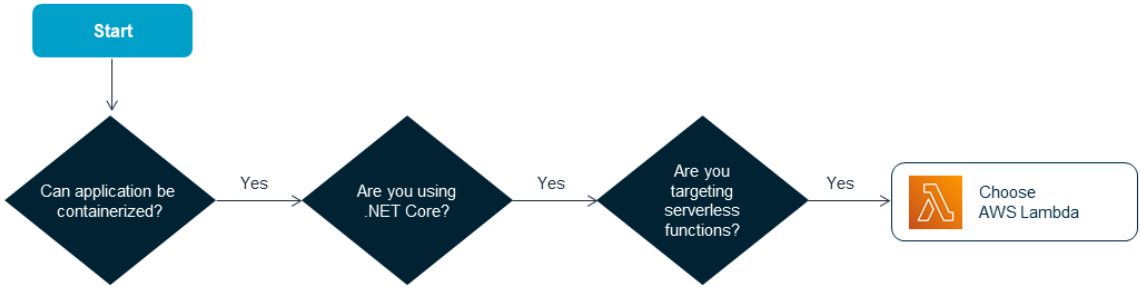
- Amazon API Gateway
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- AWS Lambda
- AWS CloudFormation or AWS CDK
- IAM
- Amazon S3

Tools

AWS Professional Services offers custom tools and services to help you refactor your monolithic applications into microservices.

Deployment decisions

This migration and modernization approach is supported by AWS Lambda



Next steps and resources

After you choose a migration and modernization strategy for your .NET application, use the following resources to evaluate your enterprise and applications for readiness, and to begin the migration process:

- Strategy for modernizing applications in the AWS Cloud
- Evaluating migration readiness
- Evaluating modernization readiness for applications in the AWS Cloud
- AWS Migration Acceleration Program (MAP)

Migration tools

- Windows Web Application Migration Assistant
- AWS App2Container (A2C)
- Porting Assistant for .NET

AWS Prescriptive Guidance glossary

AI and ML terms (p. 17) | Migration terms (p. 18) | Modernization terms (p. 21)

AI and ML terms

The following are commonly used terms in artificial intelligence (AI) and machine learning (ML)-related strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

binary classification	A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as “Is this email spam or not spam?" or "Is this product a book or a car?"
classification	A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.
data preprocessing	To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.
deep ensemble	To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.
deep learning	An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.
exploratory data analysis (EDA)	The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.
features	The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.
feature transformation	To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the “2021-05-27 00:15:37” date into “2021”, “May”, “Thu”, and “15”, you can help the learning algorithm learn nuanced patterns associated with different data components.

multiclass classification	A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"
regression	An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).
training	To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.
target variable	The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.
tuning	To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.
uncertainty	A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the Quantifying uncertainty in deep learning systems guide.

Migration terms

The following are commonly used terms in migration-related strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

7 Rs	Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following: • Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition. • Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud. • Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com. • Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.

	• Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. This migration scenario is specific to VMware Cloud on AWS, which supports virtual machine (VM) compatibility and workload portability between your on-premises environment and AWS. You can use the VMware Cloud Foundation technologies from your on-premises data centers when you migrate your infrastructure to VMware Cloud on AWS. Example: Relocate the hypervisor hosting your Oracle database to VMware Cloud on AWS. • Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there's no business justification for migrating them. • Retire – Decommission or remove applications that are no longer needed in your source environment.
application portfolio	A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.
artificial intelligence operations (AIOps)	The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.
AWS Cloud Adoption Framework (AWS CAF)	A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.
AWS landing zone	A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.
AWS Workload Qualification Framework (AWS WQF)	A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.
business continuity planning (BCP)	A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.
Cloud Center of Excellence (CCoE)	A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-

	scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.
cloud stages of adoption	The four phases that organizations typically go through when they migrate to the AWS Cloud: • Project – Running a few cloud-related projects for proof of concept and learning purposes • Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model) • Migration – Migrating individual applications • Re-invention – Optimizing products and services, and innovating in the cloud These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.
configuration management database (CMDB)	A database that contains information about a company's hardware and software products, configurations, and inter-dependencies. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.
epic	In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program implementation guide.
heterogeneous database migration	Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.
homogeneous database migration	Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.
IT information library (ITIL)	A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.
IT service management (ITSM)	Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.
Migration Acceleration Program (MAP)	An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.
Migration Portfolio Assessment (MPA)	An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires

	login) is available free of charge to all AWS consultants and APN Partner consultants.
Migration Readiness Assessment (MRA)	The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.
migration at scale	The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.
migration factory	Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the CloudEndure Migration Factory guide in this content set.
operational-level agreement (OLA)	An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).
operations integration (OI)	The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.
organizational change management (OCM)	A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.
playbook	A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.
responsible, accountable, consulted, informed (RACI) matrix	A matrix that defines and assigns roles and responsibilities in a project. For example, you can create a RACI to define security control ownership or to identify roles and responsibilities for specific tasks in a migration project.
runbook	A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.
service-level agreement (SLA)	An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

Modernization terms

The following are commonly used terms in modernization-related strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

business capability	What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.
microservice	A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.
microservices architecture	An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.
modernization	Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.
modernization readiness assessment	An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.
monolithic applications (monoliths)	Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.
polyglot persistence	Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.
split-and-seed model	A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization's capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.
two-pizza team	A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

For more information, see the Two-pizza team section of the Introduction to DevOps on AWS whitepaper.

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

update-history-change	update-history-description	update-history-date
.NET 6 (p. 24)	Updated .NET references to reflect the latest version.	December 9, 2021
Initial publication (p. 24)	—	November 3, 2021