# Developing and Deploying .NET Applications on AWS

*October 2019*

# Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

# Contents

# Abstract

As the standard application development framework for Microsoft Windows, developing and deploying .NET applications on Amazon Web Services (AWS) is a key activity to help organizations achieve the scale and agility offered by cloud computing.

Whether it's migrating legacy .NET Framework applications or creating modern microservices using .NET Core, AWS offers a wide range of end-to-end services, tools and solutions for application development, deployment and maintenance, and continues to be a preferred platform to run traditional and modern .NET applications.

This paper focuses on introducing the AWS tools and services that are directly suited for .NET development and deployment. It serves as a starting point for .NET architects and developers who wish to develop, build, deploy and maintain their applications on AWS. It describes the approaches that can be used to deploy .NET applications on AWS, and details the options, choices, and services that can help readers get the most business value from their cloud-based .NET workloads.

# Introduction

Developing and deploying applications are critical aspects of providing modern organizations with new and innovative services while helping them maintain and operate their existing capabilities. Although there are an increasingly diverse set of application development technologies, .NET has been the *de facto* standard for Windows since it was first released by Microsoft, and with a growing ecosystem of alternative .NET implementations, it is increasingly being chosen for a variety of cross-platform workloads.

Nonetheless, no application is an island, and .NET applications not only depend on environments to execute in, but also require a plethora of additional services, including, but not limited to, relational databases, queuing middleware, authentication and authorization services, file storage, networking, caching, and a variety of operational monitoring and logging services.

AWS provides a reliable, scalable and global infrastructure platform with a broad set of global cloud-based services. With over 160 services that can be provisioned quickly without upfront capital expenses, AWS provides the ideal environment to not only deploy existing .NET applications, but also to create new, modern and innovative .NET applications.

This paper focuses on the key AWS services for developing and deploying .NET applications. For information on the full range of services, refer to the homepage on the AWS website.

## Working with Different Variants of .NET

Given the wide variety of .NET implementations, each of which provides slightly different capabilities and packaging tools, it's useful to understand the key differences between the implementations to understand the full set of options and how best to run .NET applications on AWS.

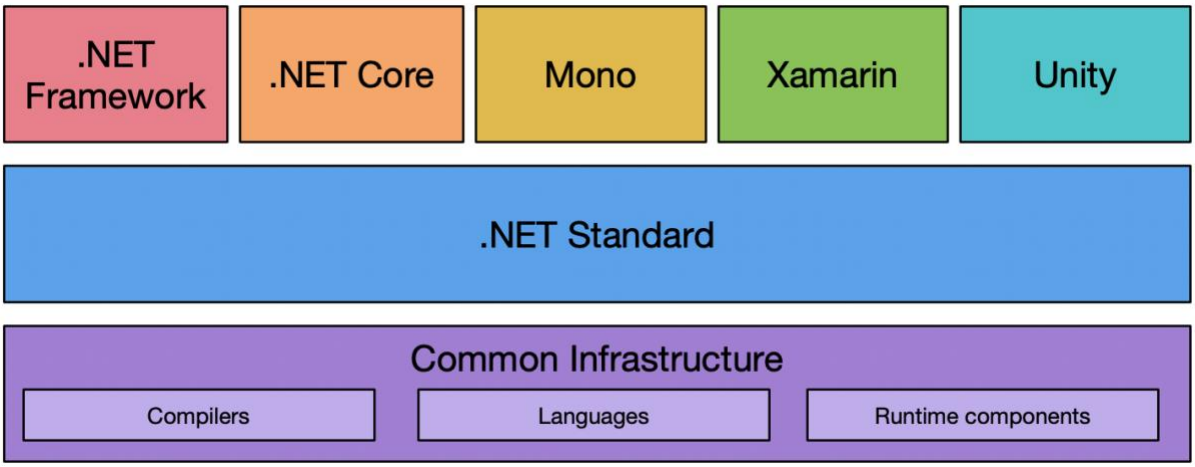


*Figure 1: Common variants of .NET*

## .NET Framework

The .NET Framework is the original implementation of .NET and exclusively runs on Windows. Although it is closed-source there are many open-source projects developed on top of it, including the ASP.NET MVC, Entity Framework, and Enterprise Library.

Since it has been around for a long time, most legacy and existing .NET applications are developed for the .NET Framework, and it also has the richest set of libraries, assemblies, and an ecosystem of packages.

One of the key challenges for .NET Framework applications is that backward-compatibility can be broken by changes in the framework and programming techniques across different versions. There are a number of different versions, some of which are in-place updates for previous versions and cannot be installed side-by-side. Therefore, a server hosting an application running on an earlier version of .NET Framework, may or may not be able to host applications that need newer versions at the same time.

.NET Framework is primarily used for monolithic applications, and with .NET Core set to officially replace the .NET Framework when .NET 5 is released[1], it is no longer recommended for new applications.

You can run .NET Framework server applications in two ways:

1. Directly deployed on a Windows server
2. Running inside a Windows container

Both methods are supported on AWS, and in later sections of this paper, we discuss the various deployment methods and tools suitable for running .NET Framework applications.

## Mono

The Mono project was launched shortly after the .NET Framework, and provides an open-source, cross-platform implementation of the .NET Framework. Supporting both 32- and 64-bit systems on various architectures, Mono runs on a wide variety of operating systems, including: Android, BSD, Linux, macOS, Solaris, Sony PlayStation 4, and Windows.

With Mono version 6.4 supporting C# 8 and many features from the .NET Framework 4.7, it is increasingly used in niche use cases when cross-platform applications require specific capabilities not available in .NET Core.

You can run Mono applications on AWS using Amazon Elastic Compute Cloud (Amazon EC2), and containers, and it can run on Windows and the Linux operating system.

## Xamarin

Spawned as an offshoot from Mono for mobile development, Xamarin provides a number of tools and libraries for developing GUI applications for a variety of operating systems, including Android, iOS, macOS, tvOS, watchOS, and Windows.

Given its use for developing mobile applications, one of the key AWS services that supports Xamarin development is AWS Device Farm, a service for testing Android and iOS applications on a variety of physical devices, including phones and tablets.

Xamarin also implements the .NET Standard. Although it predominately runs on end-user devices, you can run Xamarin applications on AWS using Windows running on Amazon EC2, or within Amazon Elastic Container Service (Amazon ECS) or Amazon Elastic Kubernetes Service (Amazon EKS) containers.

## Unity

Unity is a cross-platform game engine supporting a wide range of platforms, and can be used to create 2D and 3D games. It uses Mono as a scripting engine, and supports .NET Standard 2.0.

Although Unity client applications, by their very nature, run on end-user devices, you can run Unity applications on Windows and Linux hosted on EC2. Additionally, since many modern games provide multiplayer options and rely on various online services—for example, for storing scores—you can also use Unity with the Amazon GameLift game server hosting, or other tools described on the AWS Game Tech webpage.

Although it's predominately used for games, Unity's growing popularity for creating XR content—including Virtual Reality, Augmented Reality, and Mixed Reality—means its increasingly finding use for architecture, design, and engineering applications.

## .NET Core

.NET Core is a modern, open-source, cross-platform implementation of .NET, and runs on Windows, Linux and macOS. However, although .NET Core provides many of the same interfaces and method signatures as the .NET Framework, there are a variety of differences, making it potentially difficult to migrate applications from the .NET Framework to .NET Core.

Although there are some limitations for migrating existing .NET Framework applications to .NET Core, this can be simplified by checking compatibility using the .NET Portability Analyzer, and by using the Windows Compatibility Pack.

.NET Core is the recommended platform for modern scalable and high-performance applications, and, unlike .NET Framework, its design makes it ideal for targeting microservices architectures. You can run .NET Core applications on AWS as direct deployments on Windows or Linux EC2 instances, on Windows or Linux containers running on EC2 instances, serverless Linux containers running on AWS Fargate, or serverless AWS Lambda functions. These services are discussed in more detail in later sections of this paper.

## .NET Standard

Microsoft initially created Portable Class Libraries (PCL) to allow libraries to be shared across different implementations of .NET. However, since .NET Core introduced some additional cross-platform constraints, a new way was needed to share libraries across different .NET implementations so Microsoft created the .NET Standard which defines a common subset of libraries available in all compliant .NET implementations.

Although the .NET Standard is constantly evolving and now has multiple versions, by developing libraries against a specific version of .NET Standard you can ensure DLLs

can be reused in all flavors of .NET that support that version of .NET Standard, with no need to change the code or recompile.

# Running .NET Applications in the AWS Cloud

The AWS Cloud provides a number of benefits, including elasticity, scalability, and flexibility, but many legacy applications were designed with the server hardware and infrastructure being critical aspects of the application's design.

Architects and software engineers had little choice but to shape .NET applications into existing deployment environments, which generally involved a fixed set of resources that often needed to be shared across a number of applications or services.

For the many legacy .NET applications, the most suitable compute choice for running applications in AWS is using virtual machines, using either AWS Elastic Beanstalk or Amazon EC2. In some cases, it's also possible to run .NET applications in Windows containers, and you can also run .NET applications on Amazon EC2 bare metal instances, either by running directly on the Windows OS of the host instance or alternatively by running Hyper-V on the instance.

In contrast, modern .NET applications can be designed to take advantage of all the cloud benefits by using Infrastructure as Code (IaC) and DevOps practices. Not only can modern applications use the traditional set of compute choices, but they can also target various types of serverless environment, including AWS Fargate or AWS Lambda.

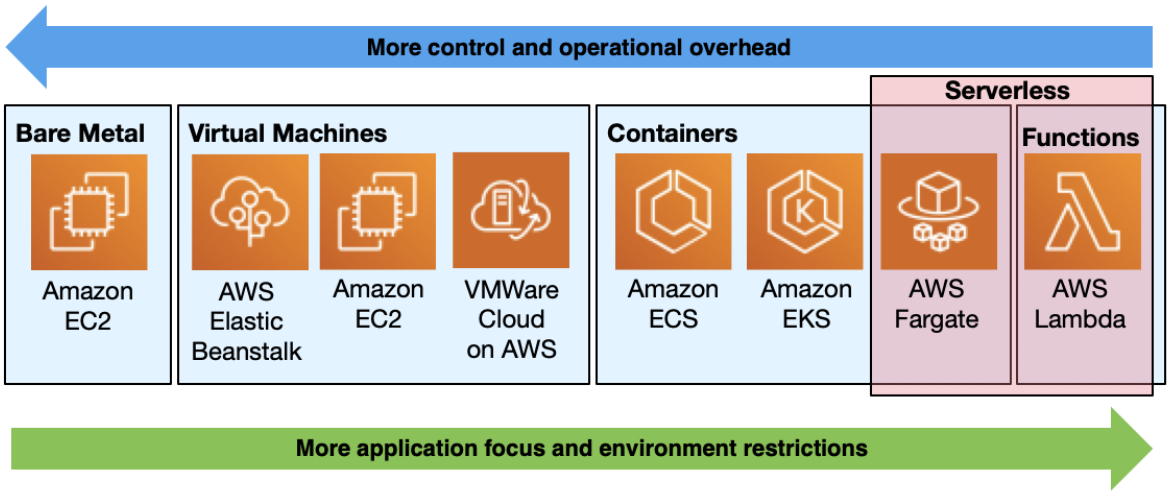


*Figure 2: AWS deployment targets for compute workloads*

Although the variety of compute choices can be daunting for new applications, a good rule of thumb is to consider serverless options for innovative and highly-elastic workloads, and then consider the various container options or virtual machines when there are specific dependencies on the environment or when more precise control is needed.

For legacy .NET applications, virtual machines are often considered the natural choice, although the integration of Docker with Windows means the use of containers is growing increasingly common, since they bring with them a number of benefits, including immutable deployment and improved resource utilization.

Subsequent sections cover each of these compute choices in more detail. However, prior to choosing a compute environment, you must choose an operating system for hosting an application and choose a suitable architectural style.

## Choosing a Host Operating System

Although Windows remains the natural choice for legacy applications using the .NET Framework, the rise of .NET Core means Linux is now an equally viable choice for new and future .NET applications. One of the challenges in choosing an OS is they have broadly reached a state of commoditization, with the current focus on OS evolution being largely about increased efficiency of resource use, as shown by the growing popularity of containers, and the future lure of library operating systems.

Another factor driving the choice of OS is the current architectural wisdom to explicitly declare and isolate dependencies, as promoted by the 12-factor app approach, which also aligns to the single process model of containers. Given the rich set of services built into Windows, it is common for legacy .NET Framework applications to implicitly depend on a variety of services—such as Active Directory for authentication and authorization, COM+ for distributed transaction processing, or DFS for file sharing—but with the move towards explicitly declaring and isolating such dependencies, relying on Windows' intrinsic features no longer holds the lure for .NET applications that it once did.

## Building Monoliths or Microservices

One of the most common ways to build enterprise applications is as a single, unified application, in which all components are tightly coupled, and working from a shared database. When the .NET Framework was released such monolithic applications were widespread, and even today, it's not uncommon to see ASP.NET applications with over a hundred thousand lines of code, that have to be deployed to a single IIS instance.

As enterprise applications grew bigger, new challenges began to emerge out of this approach. The first problem is managing the resources available to an application. As monolithic applications grow bigger, they invariably require more resources, from compute and memory requirements, through to storage and network bandwidth. Although these issues can be solved by scaling the application servers vertically up or horizontally out, this approach naturally scales the whole application, even if a single module needs the additional resources.

The second problem is complexity. Monolithic applications with tightly coupled modules grow increasingly complex over time, which can make maintenance so complicated that even the smallest changes require significant effort for development, testing, and deployment. Although there's an inherent simplicity in the design of monolithic applications, the increasing complexity adds friction to the business need for agility.

Because of the challenges inherent in monolithic applications, many modern applications have shifted to a new paradigm, commonly known as a microservices architecture. Microservices are small services providing a bounded context of functionality, each using their own data store, and predominantly integrating with other services by using event-driven communication.

Although microservices introduce their own complexities, such as how to separate data or how to distribute services, breaking monolithic applications into loosely coupled microservices can help overcome many of the problems with monolithic applications. Aside from the architectural benefits of microservices, the loose coupling in microservices means each service can be deployed and scaled independently. By ensuring each microservice has its own development lifecycle, DevOps teams are no longer tied to other team's release cycles, and can therefore increase their deployment frequency, improving their agility, and increasing the business's ability to respond to change.

Although .NET Core can be used for a variety of application architectures, its lightweight and cross-platform nature makes it ideal for microservices, and it's also highly suitable for deploying to modern execution environments, including containers and serverless functions.

The following sections of this paper include several ways you can deploy both monolithic and distributed applications in the AWS Cloud. Monolithic deployment patterns are mostly applicable for legacy enterprise applications, or for developing new applications with limited complexity or scaling requirements, whereas microservices are commonly chosen for building optimized modern applications.

For more information on how to design and develop microservices, see the Implementing Microservices on AWS whitepaper.

# Migrating and Rehosting .NET Applications

When migrating any type of application to AWS, including legacy .NET Framework applications, there are a number of different approaches. These approaches are known as the six Rs of migration.

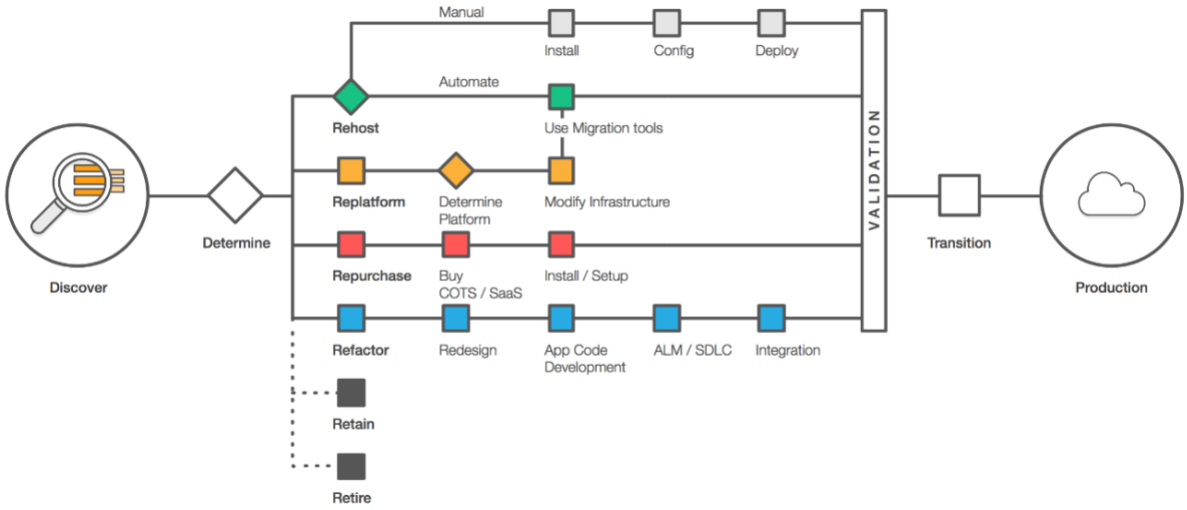


*Figure 3: Six Rs of migration*

Although there's a number of reasons to modernize applications by re-platforming or refactoring—including optimizing or adding new features—the simplest way to migrate .NET applications to AWS is to rehost the applications using either AWS Elastic Beanstalk or Amazon EC2.

## AWS Elastic Beanstalk

In many cases, you may not require full control over the underlying infrastructure used for hosting an application and may prefer a managed environment, allowing you to focus on the application and simply deploy application updates when needed.

AWS Elastic Beanstalk is the fastest and simplest way to deploy .NET applications on AWS. Developers find AWS Elastic Beanstalk convenient, because, for the most part, you do not have to change the way you have always worked. As a .NET developer, you can continue using your favorite programming languages with .NET Framework or .NET Core, and once you've completed developing your application on your computer, your

application will be ready within minutes to use without any infrastructure or resource configuration work.

The key advantage to this model is that it is not too different from the way most existing and legacy applications work. Therefore, it can be an excellent option to maximize the benefits of deploying legacy applications in the cloud, without a significant migration overhead.

For example, you can create an AWS Elastic Beanstalk deployment for an ASP.NET web application that uses Internet Information Services (IIS) and .NET Core to host your application. You build your application locally or through a pipeline and push the binaries into your AWS Elastic Beanstalk environment. AWS Elastic Beanstalk manages the remaining environment configuration and setup.

AWS Elastic Beanstalk provisions, operates, maintains, scales, monitors, heals, updates, and patches Windows and IIS for you, enabling you to focus on your application code rather than its operating environment. While providing all these benefits, AWS Elastic Beanstalk also allows you to retain complete control over your application resources, allowing you to seamlessly make changes to the way your applications run inside AWS Elastic Beanstalk environment.

To get started with AWS Elastic Beanstalk you create *Environments* for your application, such as Dev, Test, or Production. Every time you make an application change, you compile and package your new build and upload it as a new *Application Version* in your AWS Elastic Beanstalk *Application*. This allows you to deploy any application versions across your application environments with a few clicks.

You can also choose to deploy one or more applications in each of your AWS Elastic Beanstalk environments, using deployment manifests to configure multiple applications to run in each IIS web-server. It also allows you to control *Application Pools* associated with your web applications in IIS, therefore making it possible to host multiple applications with a shared application pool, or assigning dedicated application pools to each application.

You can further customize and configure your web-server in AWS Elastic Beanstalk using configuration files, which let you install additional software packages, files, windows services, or other dependencies your application needs to run.

AWS Elastic Beanstalk also supports several deployment options, including All at once, Rolling, Rolling with additional batch, and Immutable. Furthermore, through integration

with Amazon Route 53, AWS Elastic Beanstalk supports environment URL swapping, therefore making it easy to implement a blue-green deployment model.

There are no additional costs associated with using AWS Elastic Beanstalk, and you only pay for the underlying resources used to run your application, such as the EC2 instances, load balancers, and Amazon RDS database instances.

Although AWS Elastic Beanstalk can help you quickly move your .NET applications to AWS with minimal changes, if you need more control of the underlying infrastructure, then directly using Amazon EC2 virtual machines allows you to fine tune your infrastructure configuration.

## Amazon EC2 Instances

Amazon Elastic Compute Cloud provides a flexible compute service with a wide variety of virtual machines and bare metal instance types.

AWS is responsible for operating everything up to host OS and hypervisor level, giving you full access and control over the guest operating system. You are responsible for patching, updating, securing, and maintaining the Windows or Linux OS, application server, web server, and any application code. However, since you have full control of your environment, you also have complete flexibility to configure your environment as you need. For example, you may want to use Nginx or Apache HTTP Server instead of IIS as your web server.

Amazon EC2 instances provide the highest level of flexibility and control in the cloud. This flexibility often proves essential for legacy applications, but can also be useful for many modern applications. You have the choice of both x86 (32-bit), as well as x64 (64-bit) platforms in Amazon EC2. Furthermore, you have access to most popular Linux versions and all Windows Server versions from Windows Server 2008 to Windows Server 2019. For modern Windows applications, you can also use Semi-Annual Channel Windows releases, including 1709, 1803, 1809 and 1903.

For any of these, you can choose from Amazon Machine Images (AMIs) provided by AWS, numerous community and AWS Partner AMIs available in the AWS Marketplace, or you can create or import your own images.

In addition to these choices, you can also benefit from various features, such as automatic scaling, self-healing, and deep integration with other services, such as AWS Identity and Access Management (IAM), AWS Key Management Service (AWS KMS), or Amazon Elastic Block Storage (Amazon EBS).

There are a variety of ways you can deploy your .NET applications on Amazon EC2 instances, from manual deployments to using Infrastructure as Code with AWS CloudFormation templates and Continuous Integration/Continuous Deployment pipelines.

## AWS Systems Manager

AWS Systems Manager is a service for hybrid and cross-platform infrastructure management. Although it is designed to help system-administrators maintain their infrastructure resources, some of its capabilities are also extremely useful for developers and DevOps engineers.

One of the most basic application requirements is the need for configuration variables, for example, an external service's URL, or a database connection string. A common practice is to store these variables in an *app.config* or *web.config* configuration file, or to store them in environment variables. However, this requires updating the configuration on all the application servers, which requires a significant effort when working with a multi-server environment. The AWS Systems Manager Parameter Store provides an alternative of a centralized location for storing configuration variables, allowing configuration values to be updated in a single place and retrieved by all application instances.

Another common use case handled by Systems Manager is the ability to run a particular command on multiple servers. For example, you may have a PowerShell cmdlet to delete files from a local application cache. One way to run the command is to open a Remote Desktop Protocol (RDP) session to the target servers and manually run the command. However, if the command must run on dozens, hundreds, or even thousands of servers, this approach becomes increasingly impractical. Fortunately, you can use AWS Systems Manager Run Command to securely run the command at any scale.

You can also use AWS Systems Manager State Manager for handling drift-management and ensuring compliance of your target server configurations. State Manager supports PowerShell Desired State Configuration (DSC) and enables you to use DSC Managed Object Format files to define your desired state using declarative language. For example, you can specify the installed state of Windows Communication Foundation (WCF) as the desired state on a server, and DSC will ensure WCF is installed. AWS Systems Manager augments PowerShell DSC through integration with Parameter Store, Amazon Simple Storage Service (Amazon S3) and Amazon CloudWatch.

For more details, see Run compliance enforcement and view compliant and non-compliant instances using AWS Systems Manager and PowerShell DSC on the AWS Management Tools Blog.

Finally, you can use AWS Systems Manager Automation to simplify complex operations and define dynamic workflows that orchestrate invocation of AWS Systems Manager or any other AWS APIs in fully automated runbooks.

For example, you can define these steps in a document to update EC2 instances:

- Provision a new EC2 instance using an updated AMI
- Bootstrap the new instance and deploy the application in offline mode
- Shutdown the old instance
- Switch the new instance into online mode

By specifying these steps in a Systems Manager Automation document, the steps can be saved as a reusable runbook, ensuring updates can be carried out consistently and shared between members of the development and operations teams.

# Modernizing and Replatforming .NET Applications

With the push to unlock business agility by using modern development and operations practices known as DevOps, modern applications are increasingly designed for flexibility using the principles of evolutionary design and a variety of best practices.

From the use of immutable infrastructure to increase deployment consistency, to the use of automation, Continuous Integration (CI), and Continuous Deployment (CD) to speed up delivery, there are a growing number of practices that help deliver business value.

Although many of these approaches can be partially applied to traditional architectures, modern application architectures are evolving to best take advantage of these modern development practices. With the AWS Cloud increasing the speed of evolution, now is the perfect time to design or replatform .NET applications to align with modern practices.

## Running Applications in Containers

Containers allow applications to be bundled with their own libraries and configuration files, and then executed in isolation on a single OS kernel, bringing a number of benefits, including:

- **Isolation and high-density**: Containerization ensures application isolation both in terms of security and data access, as well as resource allocation. It is therefore a reliable solution to run multiple tasks or applications on the same host. This approach allows you to maximize overall resource utilization and minimize idle capacity, also known as a “high-density” deployment.
- **Runtime packaging and seamless deployment**: Containers include application code or binaries along with all the dependencies needed to keep the application running. This approach ensures the application behaves consistently in all environments, from a developer laptop to a production environment. It also greatly simplifies migrating applications from one host to another.
- **High availability (HA)**: Container orchestrators provide an abstraction layer on top of conventional hosting environments and keep track of running containers. You no longer have to run applications, but rather tell the orchestrator which applications are expected to run. The orchestration engine keeps track of the existing state and evaluates it against the expected state and corrects as needed. Consequently, if an application goes down the orchestrator immediately spins up another container to run your application in the next available host.
- **Resource management for distributed systems**: Containerization is an effective approach to run microservices and other types of distributed systems. The deployment abstraction provided by containers allows you to focus on your applications rather than their dependencies with underlying hosts and infrastructure.

The following sections discuss the four container services available in AWS.

### Amazon Elastic Container Service

Amazon ECS is a highly scalable and high-performance container orchestration service. It has been natively developed in AWS and offers deep integration with AWS services such as Elastic Load Balancing, Amazon Virtual Private Cloud (Amazon VPC), IAM, AWS Batch, and Amazon CloudWatch.

Amazon ECS is suitable for a broad range of containerized applications, from long-running applications and microservices, to batch jobs and High-Performance Computing

workloads, and supports both Linux and Windows containers. Linux containers are available in Amazon Linux and other Linux distributions, and Windows containers are available in Windows 2016 and later.

To use Amazon ECS, you can either use one of the prebuilt Amazon ECS optimized AMIs to spin up a cluster of host instances, or you can build your own AMIs by adding the Amazon ECS container agent to an existing or custom-built EC2 host.

For more information on running Windows containers on ECS, see Migrating .NET Classic Applications to Amazon ECS on the AWS Compute Blog.

### Amazon Elastic Kubernetes Service

Kubernetes is one of the most popular open source orchestration engines for containerized workloads, and allows you to run containerized applications using the same toolset on-premises and in the cloud.

Amazon EKS makes it easy to deploy, manage and scale containerized applications using Kubernetes on AWS by managing clusters of Amazon EC2 instances and running containers on those instances. Amazon EKS provides a management plane for a highly available Multi-AZ Kubernetes cluster, and you can join to it your additional worker nodes as EC2 instances.

Since Linux containers are available in all versions of Kubernetes, you can run .NET Core applications on any version of it. In contrast, Windows containers are only available starting with Kubernetes 1.14, but they are only supported in Windows 2019 and later.

### Amazon Elastic Container Registry

Amazon Elastic Container Registry (ECR) is a fully-managed, highly-available and secure Docker container registry that helps developers store, manage, and deploy Docker container images.

Amazon ECR is integrated with other AWS services, such as AWS IAM, and provides a repository to store container images which you can then easily use from Amazon ECS, AWS Fargate, and Amazon EKS.

### AWS Fargate

AWS Fargate is a serverless compute engine for Amazon ECS, that abstracts away details of the underlying host infrastructure such as the instance types, instance sizes, and host OS version.

By letting you focus on designing and building your applications and removing the need to manage the underlying infrastructure, AWS Fargate can help and reduce the operational overheads of using containers.

AWS Fargate supports Linux containers and is a powerful option for running .NET Core applications. For more details, see Hosting ASP.NET Core applications using AWS Fargate on the AWS Compute Blog.

## Creating Serverless Applications with AWS Lambda

Containers provide a high level of flexibility; however, you still need to manage your container images, including the guest OS and any application dependencies.

For example, suppose you need to deploy an ASP.NET Core application. In addition to the application, the container image also must include a choice of guest OS, the .NET Core runtime library, the ASP.NET Kestrel engine, and a web server such as Nginx or Apache. Although this gives you more control over your runtime environment, it also means additional undifferentiated efforts, and in most cases this level of control is not required.

AWS Lambda solves this problem by providing a serverless Function-as-a-Service (FaaS) model, which automatically manages the underlying compute resources for you. C# code can be uploaded into a Lambda function, and everything else is handled by AWS Lambda.

AWS Lambda provides the highest level of abstraction, simplicity, efficiency and scalability for running .NET code in the cloud. It is simple because it allows developers to run their code without having to worry about the infrastructure that runs it; efficient because there is no charge when the code is not running; and scalable because it seamlessly handles load fluctuations. AWS Lambda supports many of popular programming languages, including .NET Core and PowerShell.

Lambda functions are often deployed behind API instances in Amazon API Gateway, which provide managed endpoints that act as front doors for consuming applications to access data or backend functionality. API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management. API Gateway can also be used with workloads running on EC2 instances or ECS tasks.

Although API Gateway helps manage well-defined APIs, when you don’t need the governance features it offers, you can deploy Lambda functions behind an Application

Load Balancer (ALB), which allows load to scale elastically without having to maintain a set of managed APIs.

For more information on how to create applications using AWS Lambda, see the Serverless Architectures with AWS Lambda whitepaper.

## Storage Solutions for .NET Applications on AWS

Most applications need various storage requirements, such as relational or NoSQL databases, flat files, object storage, block storage, or various types of in-memory cache tiers. .NET applications are no exception and depending on their functionalities, may require one or more of these solutions.

AWS includes a number of purpose-built relational database services, such as Amazon Relational Database Service (Amazon RDS), Amazon Aurora, and Amazon Redshift, the fastest growing cloud data warehouse service,

There are also a number of specialized databases, including Amazon DynamoDB for key-value and document storage, Amazon Neptune for graph data, Amazon Quantum Ledger Database, and Amazon Managed Blockchain.

.NET applications can use Amazon Simple Storage Service (Amazon S3) for object storage, and Amazon Elastic File Storage (Amazon EFS) for Linux-based shared file systems, or Amazon FSx for Windows file shares.

Finally, Amazon Elastic Block Storage is an easy to use, high performance block storage service designed for use with Amazon EC2 for both throughput and transaction intensive workloads at any scale.

You can build extremely high-performance .NET applications using elasticity and flexibility of managed AWS services for in-memory caching, such as Amazon ElastiCache for Redis and Memcached, or Amazon Elasticseach Service.

## Artificial Intelligence and Machine Learning with .NET

Artificial Intelligence (AI) and Machine Learning (ML) are increasingly being used to solve new types of problems, and are becoming fundamental parts of many modern applications.

AWS provides a rich set of services that aim to put AI and ML in the hands of every developer. You can either choose from a set of pre-trained AI services—including

computer vision, language, recommendations, and forecasting—or you can build, train and deploy custom models with support for all the popular open-source frameworks.

Amazon Rekognition allows you to add image and video analysis to your applications, and can identify objects, text, and activities, and can also be used for facial recognition.

There are a number of services for working with language, allowing you to easily add intelligence and process natural language in your .NET applications. Amazon Transcribe automatically converts speech to text, making it simple to enable speech in your applications. Amazon Textract extracts text from scanned documents, after which you can use Amazon Comprehend to find insights or relationships in text, helping you to extract key phrases, classify text, or analyze sentiment. Amazon Translate allows you to translate text between over 25 languages, and helps you build .NET applications that can localize content.

Amazon Polly turns text into lifelike speech, and Amazon Lex allows you to build conversational interfaces and chatbots into any application using voice and text. Amazon Lex can recognize the intent from a piece of text, enabling you to orchestrate a conversation and build applications with highly engaging user experiences.

Amazon Personalize allows you to enhance your applications with real-time personalization and recommendations, by working with an activity stream from your application, identify what is meaningful, and helping you serve personalized content to users.

Finally, Amazon Forecast is an accurate time-series forecasting service, allowing your application to process historical time-series data and to produce meaningful forecasts based on trends in the data.

Although the pre-trained AI services allow you to enhance your .NET applications with a rich set of capabilities, you can also build your own machine learning models. Amazon SageMaker is a managed service that enables developers and data scientists to quickly build, train, and deploy machine learning models. Using the AWS .NET SDK, you can set up, configure, and execute Amazon SageMaker jobs, allowing you to push new ML boundaries.

Access from .NET applications to all the pre-trained services and to Amazon SageMaker is available through the AWS SDK for .NET.

# Developing .NET Applications

One of the fundamental requirements when developing .NET applications to run or integrate with AWS services is having easy-to-use tools to help access the services, integrate with the development workflow and tooling, and enable higher levels of developer productivity.

Although choosing the right tools depends on numerous factors—including development approach, team composition, and organizational standards—AWS provides a rich set of tools that can be used together or alone to help .NET developers make the most of the services.

## AWS .NET SDKs

One of the great advantages AWS offers over traditional, on-premises resources is that its services can be accessed through REST APIs, enabling integration from any programming or scripting environment capable of stringing together HTTPS requests and sending them over the internet.

However, although REST APIs are flexible, it's easier for developers to work in their native language than it is to work with REST APIs, and the AWS SDK for .NET helps developers get started quickly by providing native .NET APIs to the AWS services.

The AWS SDK for .NET is distributed as multiple NuGet packages, or as a single MSI installer, and contains assemblies for .NET Standard 2.0, and also .NET Framework 3.5 and .NET Framework 4.5.

As the standard for .NET package management, NuGet is the preferred option for installing the SDK, and provides a number of service-specific packages, such as AWSSDK.EC2 or AWSSDK.S3, each of which depends on the AWSSDK.Core package, which is automatically installed when you reference a service package in the NuGet Package Manager.

Alternatively, you can install all of the packages using a single MSI installer, which also includes the AWS Toolkit for Visual Studio, and the AWS Tools for Windows PowerShell.

If you're working with older versions of .NET that don't support .NET Standard 2.0, then the versions of the SDK prior to v3.5 also contain Portable Class Library assemblies, and the AWS Mobile SDK for Unity and AWS Mobile SDK for .NET and Xamarin are available for older versions of Unity and Xamarin.

## AWS Toolkit for Visual Studio

The AWS Toolkit for Visual Studio is a plugin for Visual Studio that makes it easier to develop, debug, and deploy .NET applications that use Amazon Web Services.

The toolkit supports Visual Studio versions 2008 and later for Windows. The latest versions are distributed in the Visual Studio Marketplace, and but all versions can be downloaded from the AWS Toolkit for Visual Studio web page.

The toolkit contains a rich set of features that help configure and deploy new solutions, and can be broken down into a number of core features, most of which are available from the *AWS Explorer* in Visual Studio's *View* menu.

## AWS Toolkit for Visual Studio Code

Software development teams working on codebases written in multiple programming languages and for multiple platforms increasingly choose Visual Studio Code as their Integrated Development Environment (IDE), due to its flexibility and low resource requirements.

The AWS Toolkit for Visual Studio Code is an open source plug-in for Visual Studio Code that helps developers get started faster and provides an integrated experience for developing, deploying, and testing serverless applications.

## AWS Tools for PowerShell

PowerShell is a scripting environment built on .NET, and is widely used as the standard scripting tool on Windows, but is also available for MacOS, and Linux. Although primarily used for executing OS-level management scripts, it's frequently used by .NET developers as part of their build and deployment pipelines.

AWS Tools for PowerShell lets developers directly access AWS services from within PowerShell scripts, allowing them to manage and interact with AWS services with their standard toolset, and removing the need to call the AWS SDK for .NET directly from within scripts.

## Test Tools

Test automation plays a critical part in DevOps, and is the fundamental development practice that enables continuous integration and continuous delivery.

Many developers opt to run their integration tests in environments hosted in AWS. However, if you prefer to execute some tests locally, AWS SAM Local and the AWS .NET Mock Lambda Test Tool can help when developing AWS Lambda functions. In addition, the Localstack project is an open source tool that runs AWS APIs locally and can be called directly from your test suites.

Finally, when creating mobile applications using Xamarin, the AWS Device Farm gives access to a wide variety of physical phones and devices, providing an effective environment for testing Android and iOS applications.

# Continuous Integration and Continuous Delivery

Software development has always included a number of delivery activities, such as building and packaging new releases, testing release integration with other systems, and finally deploying new application releases in a production environment.

In the traditional delivery approach, when projects took months to deliver a functioning application, these activities were mostly manual. However, as the frequency of software delivery grows to multiple versions per week or per day, these undifferentiated activities become delivery bottlenecks.

Continuous Integration and Continuous Delivery are the combination of tools and techniques to help overcome these bottlenecks by automating the integration and delivery of applications.

## Infrastructure as Code

Modern deployment patterns require that applications—and the services and infrastructure and those applications depend on—can be provisioned and deployed reliably and consistently.

Given the complexity of deploying modern applications and infrastructure, doing so in a repeatable manner requires the deployment to be automated, and the practice and processes of automating infrastructure deployment are commonly known as Infrastructure as Code.

### AWS CloudFormation

AWS CloudFormation provides a declarative language that lets you describe and provision all the infrastructure resources in your AWS cloud environment.

Using a simple text file called an AWS CloudFormation template, you can model resources across all regions and accounts, with the file serving as the single source of truth for your cloud environment. By keeping CloudFormation templates along with your application code in the same code repository, you can ensure code changes are bundled together with infrastructure changes, ensuring integrity and enabling reliable deployment.

Templates can be written using JavaScript Object Notation (JSON) or YAML. There are many predeveloped templates you can use as starting point, or you can create your templates from scratch.

You can use a variety of methods to deploy templates and provision resources, including the AWS Management Console, AWS Command Line Interface (AWS CLI), PowerShell or the AWS SDK for .NET.

A deployed version of a CloudFormation template is called a CloudFormation Stack. You can instantiate one or multiple stacks based on each CloudFormation template, as well as delete already deployed stacks and all resources associated with them. Therefore, AWS CloudFormation is also a powerful way to quickly deploy, duplicate, provision or deprovision resources of your applications.

CloudFormation Stacks are always deployed in a single AWS account and Region, but you can use CloudFormation StackSets to deploy your templates across multiple AWS accounts and regions.

## AWS Cloud Development Kit

Although CloudFormation provides a flexible mechanism to define cloud infrastructure as code, its use of declarative syntax is not well suited in all situations.

For infrastructure requiring a high number of inter-related services, or that is best defined using iteration, the resulting CloudFormation template can easily grow to hundreds or thousands of lines, which raises its own complications.

The AWS Cloud Development Kit (AWS CDK) allows you to define cloud resources in various programming languages, including TypeScript, JavaScript, Python, C#, and Java. Developers use one of the supported languages to write code that defines reusable cloud components known as Constructs, which can then be composed into Stacks and Apps.

Once you've defined an AWS CDK App, you can use the AWS CDK toolkit to synthesize a CloudFormation template, and then to deploy the defined resources to AWS.

Although using the AWS CDK adds an additional level of complexity to your Infrastructure as Code, by using an imperative language it allows you to work with high-level abstractions, rich logic, and enables the sharing of infrastructure definitions as reusable libraries of components.

# Using AWS Developer Tools

There are a number of services collectively known as the AWS Developer Tools, which are designed to solve common DevOps requirements and provide development agility and continuous innovation.

You may already be using a CI/CD pipeline. Although you can continue using your favorite tools and easily integrate them with AWS services, you can also use AWS developer tools to create a complete pipeline, or complement, or extend the other tools you are using. Using AWS Developer Tools relieves you from managing infrastructure of your CI/CD pipeline tools and helps you further increase efficiency and productivity of your infrastructure and developers.

## Version Control

Most .NET developers are familiar with Git repositories and are well versed in using them to collaborate on software development projects.

Maintaining a Git repository for a single project may not be a big challenge, but as the number of projects grows, managing a Git repository can become burdensome. You have to make sure your source control server is available all the time, its performance does not degrade and its storage is scaled to cater for increasing demand.

AWS CodeCommit is a fully managed source control service that makes it easy to securely host private Git repositories in a highly scalable way. You can use AWS CodeCommit to store anything from your source code to other binaries and dependencies that go with your code.

For example, you might be developing an ASP.NET web application that also includes external DLL dependency files and several gigabytes of graphics and other multimedia files. Separating source code from binary files increases the risk of inconsistencies and

bugs that have nothing to do with the code. Using AWS CodeCommit, you can store all of these in one repository and avoid such problems.

As an alternative to AWS CodeCommit, both AWS CodeBuild and AWS CodePipeline integrate with GitHub, and AWS CodeBuild also integrates with Bitbucket.

## Build and Package Applications

Building the source code is one of the key steps in any CI/CD pipeline. Once a new version is committed in your source control system, you need a build service to pull the latest version, build and package it so the new version can be deployed in a target environment.

One way to do this is using build servers that you dedicate for this purpose. However, as the number of concurrent projects and number of builds in each project grows, these build servers have to be scaled out to provide more build capacity. Otherwise, your builds wait in a queue, which can result in decreased productivity of your developers.

AWS CodeBuild is a fully managed build service that compiles source code, runs tests, and produces software packages that are ready to deploy. It seamlessly scales and concurrently processes multiple builds, therefore eliminating waiting time, increasing developer productivity, and you only pay for time the build container is running.

AWS CodeBuild includes a pre-packaged build environment for .NET Core on Linux and Windows, or you can use pre-built Docker images, such as official Microsoft images for the .NET Framework, or create your own custom build environment by creating a Docker image.

For more details on creating a custom build environment for .NET Framework, see Extending AWS CodeBuild with Custom Build Environments for the .NET Framework on the AWS DevOps Blog.

## Application Deployment

Once a new application version is built and packaged, it must be deployed in a target environment for end users to access it. There are a couple of deployment strategies commonly employed, including mutable in-place deployment, or immutable deployments when the entire infrastructure stack is replaced.

There are also a variety of ways to deploy .NET applications, and AWS CodeDeploy is a deployment service that integrates with AWS CodePipeline and helps automate application deployments to Amazon EC2 instances, Amazon ECS services, on-

premises instances, and serverless Lambda functions. It supports both in-place mutable deployment, as well as immutable deployment using the blue-green deployment model.

The AWS CodePipeline service can also be used for deployment and integrates with a number of deployment providers, including AWS CloudFormation, AWS Elastic Beanstalk, Amazon ECS, AWS Service Catalog, and also AWS CodeDeploy.

### Building a CI/CD Pipeline

Each of the previously discussed developer tools can be used individually or in combination with your existing tools, but you can also integrate them together to form a complete end-to-end CI/CD pipeline.

AWS CodePipeline is an orchestration service that allows you to model the different stages of your software release process. It can be integrated with other AWS developer tools for building, testing and deploying your software versions. It can also easily be extended to adapt to your specific needs. You can use its pre-built plugins or your own custom plugins in any step of your release process.

For example, you can pull your source code from GitHub, use your on-premises Jenkins build server, run load tests using a third-party service, or pass on deployment information to your custom operations dashboard.

## Seamless Integration with Azure DevOps

The main integration point for Azure DevOps with AWS is through Azure DevOps pipelines. You can configure Azure DevOps pipeline to build, test, package and release software to different AWS environments. You can use the following methods for this integration.

### AWS Tools for Visual Studio Team Services

AWS Tools for Visual Studio Team Services is available on Azure DevOps extension market place. These extensions can be installed by navigating to the *Extensions Marketplace* through Azure DevOps.

Once installed, you can choose from a set of pipeline tasks that can be included in your pipeline to integrate with AWS.

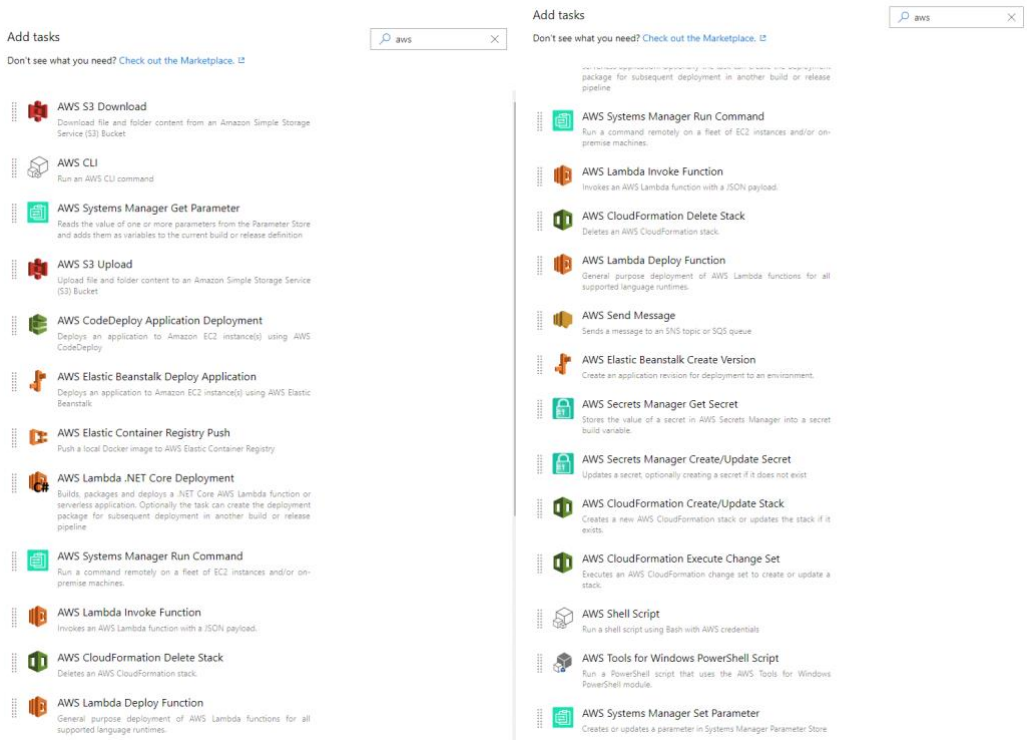


*Figure 4: Various tasks for AWS Tools for Visual Studio Team Services*

These building blocks can then be used to construct complex deployment pipelines. *Figure 5* shows an example pipeline designed to build, test and publish an ASP.NET core web application to an AWS Elastic Beanstalk environment. See *Table 1* for a description of steps.

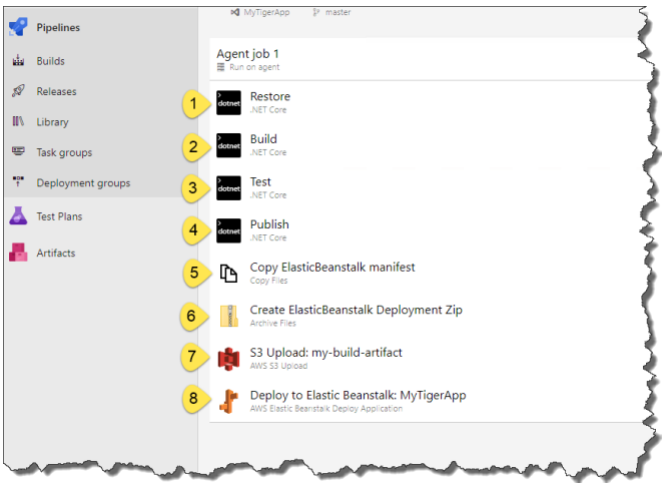


*Figure 5: Pipeline for building, testing, and deploying an ASP.NET Core application to AWS Elastic Beanstalk*

*Table 1: Pipeline step descriptions*

| Step | Description |
|---|---|
| **1** | Executes .NET Core build task, such as Git pull. |
| **2** | Executes .NET Core build task. |
| **3** | Executes .NET Core test task. |
| **4** | Executes .NET Core publish task. |
| **5** | Copies an AWS Elastic Beanstalk manifest file into the bundle |
| **6** | Creates a zip archive from newly published website content. |
| **7** | Uploads the zip archive to an S3 bucket. |
| **8** | Deploys the application in an AWS Elastic Beanstalk environment. |

### Custom Scripts

If you need functionalities beyond those provided through extensions published by AWS, or if you need more fine-grained control over your pipeline, you can use AWS CLI or AWS Tools for PowerShell to create a custom task or step in Azure DevOps pipeline.

# Security and Operations

## Application Security

Application security posture and requirements vary based on the type of application, scale of deployment, and choice of environment, but there are common principles and practices that serve as solid foundations across all types of applications, and .NET applications running on AWS are no exception.

At the most basic level, the key principle for developing secure .NET applications is ensuring requests from upstream users and systems are trusted, and ensuring requests sent to downstream systems are also trusted. Applications therefore need to safeguard any credentials or sensitive information they require to operate, control the exposure of the data they process, and integrate with security providers in a protected and maintainable way.

Running applications in a secure manner starts with developing secure applications, but also requires operating them in a secure environment. Security is the highest priority at AWS, and there are various AWS services that can help meet the requirements of the most security-sensitive organizations.

The following sections focus on developing secure applications. For more information on security, including auditing, data governance, network security, encryption in transit and at rest, see AWS Cloud Security or read the AWS Security Best Practices whitepaper.

## Programmatic Authentication and Authorization

AWS Identity and Access Management service provides a comprehensive set of authentication and authorization features. IAM allows granular permissions to be granted to AWS services. Permissions can then be assigned to users, groups of users, and users or services running with a particular role.

### Developer Access Control

Users can be assigned access keys consisting of an access key ID and a secret access key, which can be used to cryptographically sign requests to the AWS API or the AWS CLI. Since access keys grant the same access rights as the associated IAM user, it is important they're stored securely and not stored in code, configuration files, or plain text files.

When developing applications with Microsoft Visual Studio, we recommend that you use the AWS Toolkit for Microsoft Visual Studio, which contains an Account Profile feature for storing access keys. The Account Profile stores keys in an encrypted form on the developer's machine, and the developer can refer to keys in code by using the Account Profile name. This stops the keys from being exposed in code or a configuration file.

For more details on working with Account Profiles in Microsoft Visual Studio, see Providing AWS Credentials.

There are many other best practices for using the access keys—such as key rotation, and using least privilege. To learn more, see Best Practices for Managing AWS Access Keys.

### Runtime Access Control

Although access keys are well-suited for developers, when deploying .NET applications to an AWS environment, we recommend that you use IAM role-based access in place of access keys.

You can create an IAM role with permissions based on the principle of least privilege. The role is then assigned to the compute environment hosting the .NET application, whether it requires EC2 virtual machines, ECS containers, or Lambda serverless functions. This way, you no longer need to manually create and rotate programming keys to authenticate your applications. Instead, IAM generates temporary keys and automatically rotates them on your behalf, allowing your applications running on AWS to get authenticated and authorizes to use AWS resources securely and seamlessly.

To learn more about role-based access, see Granting Access Using an IAM Role.

### Runtime Access Control with Temporary Credentials

Although IAM role-based access works well for .NET applications running in AWS, for applications hosted outside of AWS—such as desktop or mobile applications—or for applications that execute with multiple roles, or across multiple AWS accounts, it's better to provide an additional level of access control by using temporary credentials.

The AWS Security Token Service (AWS STS) can be used to generate temporary credentials, and can either be accessed through a single, global endpoint or from a series of regional endpoints.

Temporary credentials are generated dynamically when requested, and can last from a few minutes to several hours. Once credentials expire, they can no longer be used to make API requests. However, the user can request new credentials, as long as they still have permissions to do so.

To learn about how to program with AWS STS using .NET, see: Making Requests Using IAM User Temporary Credentials - AWS SDK for .NET.

## Active Directory

Active Directory is Microsoft's directory service and provides a wide set of capabilities to authenticate and authorize users, services and computers in Windows domains.

Although there are a number of reasons why .NET applications may need to interact with Active Directory, the most common use case is that .NET applications running on Windows are more likely to be running under a process whose permissions have been authorized by Active Directory.

Since Linux doesn't natively use Active Directory, .NET applications are more likely to need Active Directory on AWS when they're being run on Windows hosts. However, Active Directory integration is also possible for Linux-based applications. In any of these

cases, you can either use AWS Managed Microsoft AD, which runs an actual Active Directory instance on AWS managed infrastructure, or you can use Active Directory Connector (AD Connector), a directory gateway which redirects requests to your on-premises Active Directory servers. Many businesses also choose to self-manage their Active Directory infrastructure to maintain full control over it, while running it on AWS and leveraging the flexibility, scalability and efficiency of the AWS Cloud.

More information on Microsoft Active Directory on AWS is covered in the Active Directory Domain Services on AWS whitepaper, and additional details can be found in the Securing the Microsoft Platform on Amazon Web Services whitepaper.

## User Identity Management

Successful web or mobile applications can reach millions of users, and it's critical such applications have a robust and scalable approach for user and identity management.

Applications can have specific requirements, such as using their own identity store, or leveraging an existing identity provider such as Facebook, Google, or Amazon, and some need to use a combination of their own identity store with existing identity providers. This is often coupled with requirements for a user interface that handles user registration, login, user verification, and forgotten passwords.

Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, including Facebook, Google, and Amazon, and enterprise identity providers via SAML 2.0.

Amazon Cognito can also be used to control access to REST APIS through integration with the Amazon API Gateway service, and can reduce the work to authenticate web application users by integrating with the Application Load Balancer.

Security features include multi-factor authentication, checks for compromised credentials, account takeover protection, and phone and email verification. Amazon Cognito also supports application specific identity stores, user profiles, and customized workflows and user migration through AWS Lambda triggers.

For more information, see Getting Started with Amazon Cognito. To learn how you can authenticate .NET application using Amazon Cognito, see Authenticating Users with Amazon Cognito.

## Storing and Retrieving Secrets

Your .NET application likely connects to one or more external systems, from database servers, through to cache servers, message queues, or even other applications.

Connections to external systems are secured using some form of secret information, including connection strings and a variety of system credentials. Storing and retrieving secret information in a secure manner is vital to the security of the overall application. Although .NET applications frequently use configuration files to store secrets, they come with an inherent risk of being stolen, as frequently seen when a developer mistakenly checks a configuration file into a public repository along with the source code.

A better approach is to store secrets in a secure repository, and AWS Secrets Manager can help by storing and retrieving secret information in an encrypted format. Secrets can be stored as a JSON string, allowing the application to store secrets in a variety of formats. When a .NET application needs the secret, it makes an API call to AWS Secrets Manager to fetch the secret using the secret name.

Access to AWS Secrets Manager APIs is granted through IAM policies. Applications need explicit permissions to access these secrets. The best practice is to have these IAM policies attached to an IAM role, which is then assigned to the runtime environment of your application (that is, the EC2 instance, ECS task, or Lambda function that hosts your application). This approach ensures, not only that those secrets do not leave the boundaries of AWS services, but also credentials that authorize access to those secrets are also confined within your AWS account and never leave it.

For ASP.NET Core applications, there is also a NuGet package available called the AWS .NET Configuration Extension for Systems Manager, which automatically loads and refreshes secrets from Parameter Store and AWS Secrets Manager into the configuration object for easy consumption by application code.

Another best practice is to periodically rotate secrets to palliate risk of their potential compromise. AWS Secrets Manager provides features that help automatically rotate secrets.

You can also use AWS Secrets Manager local cache library for .NET to improve availability, reduce latency and lower costs. For more information, see How to use AWS Secrets Manager client-side caching in .NET on the AWS Security Blog.

AWS Secrets Manager provides additional benefits, such as centralized secret management, allowing secrets to be shared by multiple users or applications.

For more information on managing secrets with AWS Secret Manager, see Tutorial: Storing and Retrieving a Secret.

# Monitoring

Successful DevOps requires excellent communication between the development of new application capabilities and their subsequent operation. One of the key benefits over other approaches is that it shortens the feedback loop between development and operations.

Although structuring teams to include development and operations improves human communication, running modern and distributed systems is complicated, and it is critical that the team has a good understanding of the current state and the performance of the system components, the interactivity between components, as well as the historic view of the system's behavior.

Designing a suitable approach to monitor .NET application behavior is relatively simple but requires a combination of approaches—from logging events and errors from your application and AWS resources, through recording metrics, showing current status dashboards, sending and automating responses to alerts, and providing tracing to help isolate problems.

While the traditional .NET monitoring approaches and third-party libraries still work in AWS, implementing a modern, approach generally requires introducing one or more additional AWS services or third-party tools. Although it's not necessary to use all these tools, mature DevOps teams invariably use a multi-layered approach to carefully monitor the system, track performance, and provide alerts when notable or exceptional events occur.

## Amazon CloudWatch

The cornerstone for monitoring applications running on AWS is Amazon CloudWatch, a group of services that can store log files, track metrics, send alarms, and execute automated actions when specific events are triggered.

Sending data to CloudWatch from Windows applications can be handled automatically using the Amazon CloudWatch agent, which runs as a Windows service, making it easy to integrate with CloudWatch from .NET applications hosted on Amazon EC2, Amazon ECS, or Amazon EKS.

Amazon CloudWatch provides a number of key features. CloudWatch dashboards are customizable home pages in the CloudWatch console that can be used to monitor resources and view the metrics and alarms for your AWS resources. CloudWatch Metrics stores data about the performance of your systems, and allows publishing your own application metrics. CloudWatch Alarms can monitor one or more metrics, and can trigger a variety of actions, including automatic scaling, or sending a notification to an Amazon Simple Notification Service (Amazon SNS) topic.

Amazon CloudWatch Logs stores and monitors log files, and can be used for centralized access to log files from a variety of applications, systems, and AWS services. Although Logs can be sent from Windows using the CloudWatch agent, you can configure many .NET logging libraries—including log4net, NLog, and Serilog—to send log entries to CloudWatch, and call CloudWatch directly using the AWS SDK for .NET. For .NET serverless functions running in AWS Lambda, you can send messages to CloudWatch Logs by either writing output to `stdout` or `stderr` using the `Console` class, or by using the `ILambdaContext` object.

Once logs are stored, you can view the logs from multiple sources as a time-ordered flow of events, search the logs, and display them in custom dashboards. Although CloudWatch Logs provides a number of common logging features, sometimes there are use cases that fit more closely with other logging tools. Common tools used alongside or instead of CloudWatch Logs include Amazon Elasticsearch Service, Splunk, or Loggly.

Amazon CloudWatch Events receives system events from AWS resources, and can be used to send notifications or run automated scripts when specific conditions are met. Rules are defined to match particular sets of events and conditions, and, once triggered, events can be routed to target actions, allowing notifications to be sent, or custom actions to execute. CloudWatch Events can also be run on a schedule, and provides a flexible tool to trigger various types of system automation.

## Amazon CloudWatch Application Insights for .NET and SQL Server

Whereas CloudWatch gives you a rich set of tools to customize your approach to monitoring, Amazon CloudWatch Application Insights for .NET and SQL Server enables application owners to easily monitor their application stack. It automatically sets up and analyses important metrics and logs from across their application resources in real time, and uses machine learning techniques to discover anomalies and errors. CloudWatch Application Insights for .NET and SQL Server creates automated dashboards for detected problems, helping application owners troubleshoot faster and reduce the mean

time to resolution (MTTR) for their application issues and improve Service Level Agreements (SLAs).

- **Automatically recognized application metrics and logs**: CloudWatch Application Insights for .NET and SQL Server scans your application resources and provides a list of recommended metrics and logs to monitor, and sets them up automatically, reducing your effort spent in setting up monitoring for your applications.
- **Intelligent problem detection**: CloudWatch Application Insights for .NET and SQL Server uses prebuilt rules and machine learning algorithms to dynamically monitor and analyze symptoms of a problem across your application stack and detect application problems. It helps you reduce the overhead of dealing with individual metric spikes, or events, or log exceptions, and instead get notified on real problems, along with contextual information about these problems.
- **Faster troubleshooting**: CloudWatch Application Insights for .NET and SQL Server assesses the detected problems to give you insights on them, such as the possible root cause of the detected problem and list of metrics and logs impacted because of the problem. You can provide feedback on generated insights to make the problem detection engine specific to your use case.

For example, consider you have an ASP .NET application backed by a SQL Server database, and your database starts malfunctioning due to high memory pressure, leading to HTTP 500 errors in your application server. Previously, to identify the problem and triage, you would have to go through your metrics dashboards, sift through server, application error and database logs, and possibly use third-party tools.

With CloudWatch Application Insights for .NET and SQL Server and its intelligent analytics, you can find the layer (SQL database, in this case) in your application stack causing the problem just by looking at the dynamically created dashboard of the related metrics anomalies, and log file snippets. This significantly reduces alert fatigue and the time and effort required to troubleshoot and return your application to a healthy state.

## Auditability and Change Tracking

Effective DevOps requires that teams have a transparent view of changes made to the services and infrastructure running their applications. AWS CloudTrail helps provide this transparency by monitoring and logging AWS API calls, effectively recording actions taken by users, roles, or AWS services as CloudTrail events. These events include actions in the AWS Management Console, AWS Command Line Interface, and AWS SDKs and APIs, allowing changes to be audited.

You can view and monitor CloudTrail events in the CloudTrail console, and you can store log files in Amazon S3 or send them to CloudWatch Logs. You can use CloudTrail events sent to CloudWatch to trigger alarms based on metrics, and to trigger CloudWatch events, allowing automated actions to be executed when specific API calls are logged. This combination of using CloudTrail and CloudWatch can be a highly effective approach for creating auto-healing scripts for your environment, or can form part of an advanced infrastructure automation strategy.

In addition to using CloudTrail to monitor changes, AWS Config is a service that evaluates the configuration of your AWS resources, monitors configuration changes and compares them against desired configurations. Config can send notifications of changes using Amazon SNS, or you can create automated responses using CloudWatch Events, and automated remediation using AWS Systems Manager Automation.

## AWS X-Ray

One of the challenges of modern applications is they are built from a number of distributed components and services, making it difficult to determine the cause of issues and isolate the underlying responsible service or component. For example, a ASP.NET application may be running on a number of load-balanced EC2 instances, with each instance depending on a SQL Server database hosted on Amazon RDS, and additional functionality being provided by a number of microservices hosted across EC2 instances, containers, and AWS Lambda functions. Although CloudWatch provides rich capabilities to monitor each individual component, trying to isolate which distributed components is causing a problem is a challenge in its own right.

AWS X-Ray provides an SDK that allows you to trace incoming requests to your application, and trace requests from your application to AWS services, HTTP services, and databases.

For each request into your application, data is recorded as a series of segments that are grouped into a trace. Once a trace is recorded, you can view it from the trace history, allowing you to inspect your application's performance and behavior to help focus on potential problems.

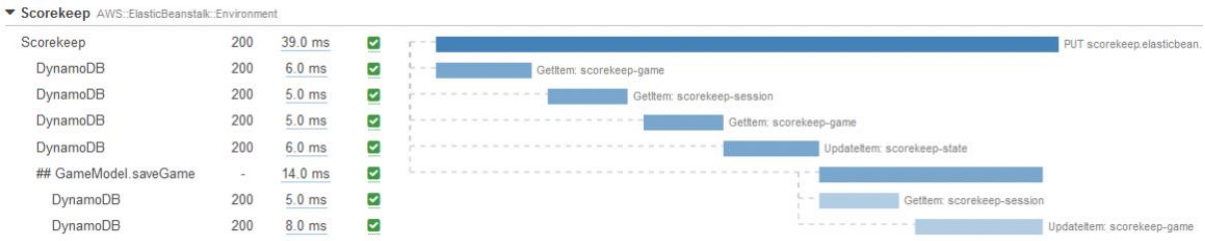


*Figure 6: AWS X-Ray trace example*

AWS X-Ray can also display **service graphs**, which show graphical views of the services used by your application, helping to isolate various issues, including faults, latency spikes, and possible bottlenecks.

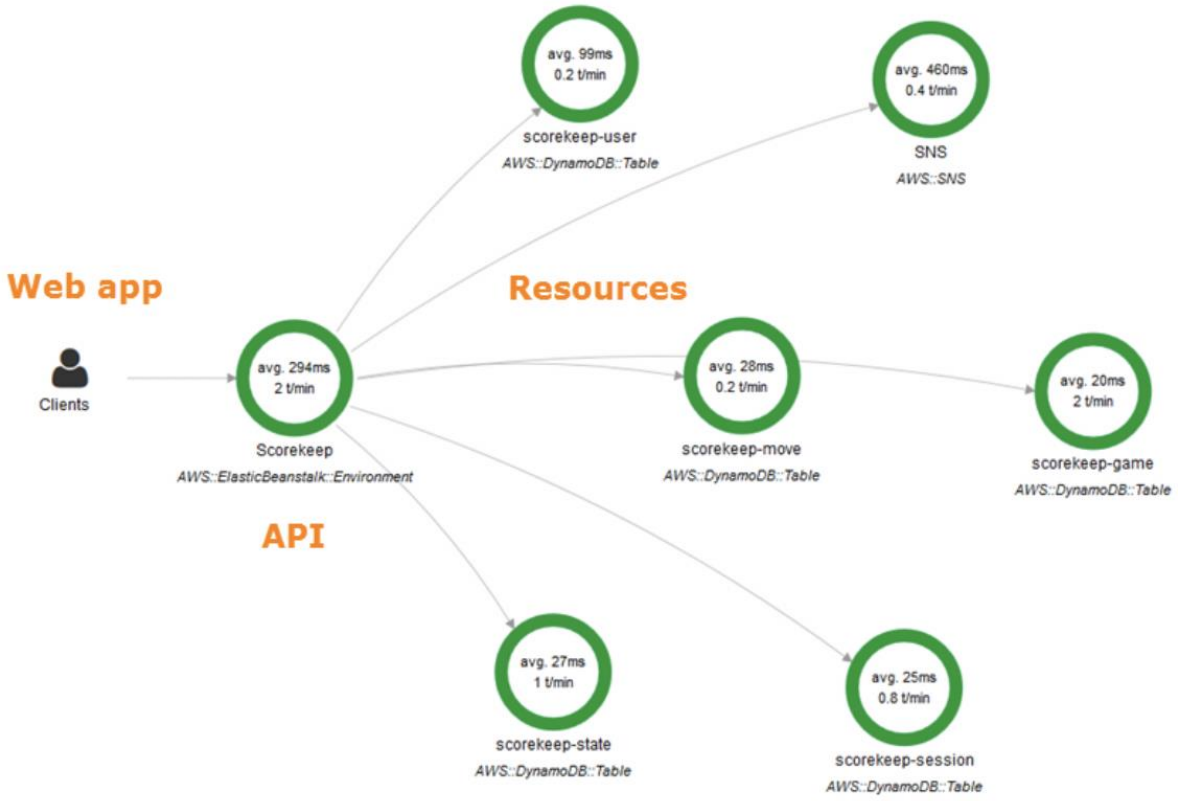


*Figure 7: AWS X-Ray service graph example showing integration across various components*

By providing a cohesive view of end-to-end application behavior, AWS X-Ray makes it possible to accurately and quickly identify sources of problems in distributed systems. At a glance, it helps locate an API, microservice, or component with problems, allowing you to drill down further with more detailed tools, such as Amazon CloudWatch.

### Additional AWS Service Logs

Although application and operating system monitoring can give a focused view on how an application is behaving, sometimes it's necessary to monitor the wider environment of AWS services.

Amazon VPC Flow Logs allow you to monitor IP traffic for your AWS resources. Flow Logs creates logs for network traffic going in and out of your VPC networks. For each flow log, you can choose to filter the traffic it records and also choose to store the logs in Amazon CloudWatch or Amazon S3. Flow logs are useful for troubleshooting a number of networking issues—such as overly-restrictive security group rules—and can also be used to audit the traffic passing through the network.

Closely related to Flow Logs, Amazon Route 53 Query Logging can track queries for DNS public hosted zones and can send logs to CloudWatch Logs.

When running .NET applications on EC2 instances or in containers, Elastic Load Balancing (ELB) allows you to spread load across multiple instances, letting your application scale and letting you take advantage of elasticity. You can use ELB Access Logs to monitor HTTP/HTTPS traffic to Application Load Balancers and TCP traffic to Network Load Balancers. Logs are captured as compressed files and stored in an Amazon S3 bucket, and can be used to analyze traffic patterns and troubleshoot load-balancing issues.

For high volume ASP.NET websites with a global presence, it's a common requirement to reduce load on the web servers. The Amazon CloudFront Content Delivery Network, helps by moving static content closer to users, tracking detailed information about every request, and storing the resulting logs in an Amazon S3 bucket.

Finally, for applications that need to store or share files, Amazon S3 provides a simple service to store and serve objects at scale. You can use S3 Server Access Logging to track access requests to your S3 buckets for troubleshooting and security audit purposes.

## Conclusion

Delivering business value by fully taking advantage of the AWS Cloud requires agile ways of working, flexible application architectures, and modern development practices.

Although .NET was considered as an exclusive Windows technology, the use of Xamarin for cross-platform mobile development, and the rise of .NET Core has helped turn .NET into a truly diverse cross-platform application framework.

This paper serves as a starting point for developing and deploying .NET applications on AWS; the real value of running .NET applications on AWS is in integrating them with the growing platform of innovative AWS services.

# Contributors

Contributors to this document include:

- Sepehr Samiei, Senior Solutions Architect, Amazon Web Services
- Mark Easton, Senior Solutions Architect, Amazon Web Services
- Aaron Schwam, Senior Manager, Amazon Web Services
- Kirk Davis, Senior Specialist Solution Architect, Amazon Web Services
- Sai Prashant Vajja, Specialist Solutions Architect, Amazon Web Services
- Ryan Pothecary, Partner Trainer, Amazon Web Services
- Brajendra Singh, Partner Solutions Architect, Amazon Web Services
- Immaya Kumar Jaganathan, Senior Solutions Architect, Amazon Web Services
- Christian Siegers, Senior Solutions Architect, Amazon Web Services
- Sriwantha Attanayake, Solutions Architect, Amazon Web Services
- Purvi Goyal, Senior Product Manager, Amazon Web Services
- Fatai Amoranbini, Cloud Application Architect, Amazon Web Services
- Tayo Olabumuyi, Principal Sales Leader, Amazon Web Services

# Document Revisions

| Date | Description |
|---|---|
| **October 2019** | First publication. |

# Notes

1 https://visualstudiomagazine.com/articles/2019/05/06/net-5.aspx