BASE2
SERVICES

eBOOK

Creating a Deployment First Strategy When Containerizing .Net on AWS

eBOOK | Creating a Deployment First Strategy When Containerizing .Net on AWS

Migrating workloads to the AWS cloud is a primary driver for modernizing applications. For companies producing SaaS solutions or other applications for the market, reducing costs and speeding development and deployment is key to maintaining revenues and increasing the ability to scale.

You may currently be on-premise or using another cloud provider and wish to migrate to the AWS cloud with applications based on the former .NET framework. If you move these applications to the latest .NET version (known previously as .NET 5/Core), you can modernize your application with containers, improve your deployment strategy and reduce costs by using non-Windows-based machines at the same time.

This can be accomplished by implementing a better CI/CD process, based on the open-source frameworks and tooling available through Linux containers and cloud-native architecture.

Migrating your application to the latest .NET version and containerizing can conceivably be done in-house. However, your developers may not have the time to devote to this process without negatively impacting workflows and production.

There are multiple ways to migrate your application, but in many cases, the full implications aren't recognized. If only the code gets ported, then modernization isn't fully leveraged and the benefits cannot be fully realized.

A successful approach to migration hinges on identifying and targeting how and where the greatest benefits can be obtained from a deployment-first strategy. The key focus should be on taking your company from a monolith environment to a lower-cost, Microservices based containerized environment as effortlessly as possible.

Is It Time to Migrate?

Challenges of remaining on older versions of .NET in large part relate to being limited to a monolith-type architecture. You have to use proprietary software, can't use a Linux environment and miss out on the multiple benefits provided by containerization.

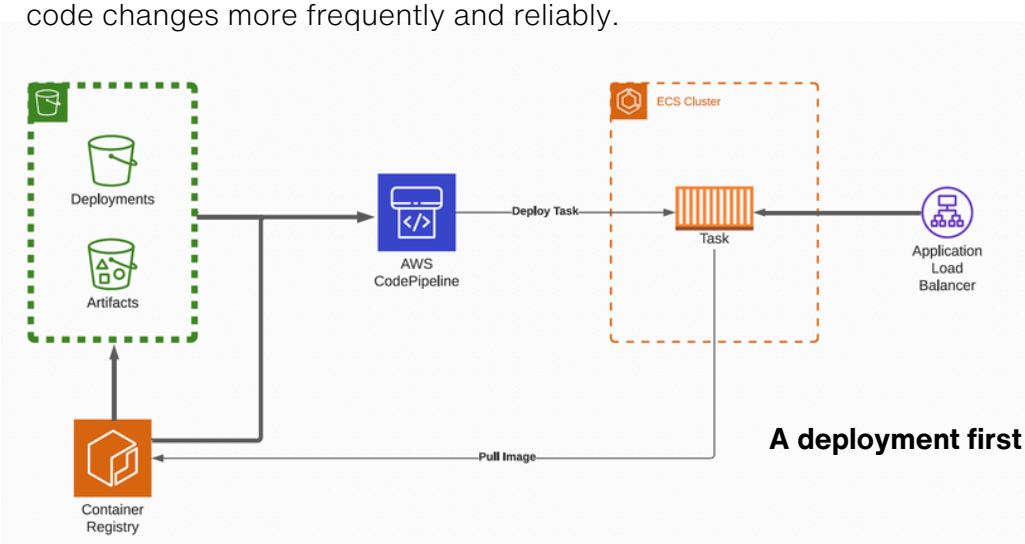
Many people think the challenge of migration will be bigger than it is, but it's not very hard to figure out where gaps are, and how to get there with proper analysis. Even when migration is undertaken, far too many companies then fail to follow through and leverage all of the newfound capabilities that come with a deployment first approach.

It's increasingly clear that migration to the latest version of .NET is something everyone needs to do eventually. What may not be clear are the benefits a complete modernization can offer. Migrating gives you the ability to run your application at a lower cost and delivers greater portability across platforms, better security, and more consistency overall.

You'll be able to run faster to achieve shorter deployment timeframes, scale as a whole or in segments as needed, and handle failure events with little to no involvement from staff. However, this can only be achieved by putting effort into the deployment strategy and ensuring your developers are governed by this strategy to achieve a modernized platform.

Not sure if you are ready to make the move to the latest version of .NET and Linux containerization? Your organization might be closer to being able to shift than you realize. An analysis could reveal that the steps needed to prepare you for the move might be fewer and much less complex than you think.
What is a Deployment First Strategy?

Deployment first focuses on the potential for cost savings and scaling through the implementation of rapid continuous integration, continuous deployment (CI/CD) and automation. An approach that enables application development teams to deliver code changes more frequently and reliably.



A deployment first CD architecture

CI/CD basics
CI establishes an automated, consistent way to build, package and test applications, helping teams collaborate more effectively, commit code changes more frequently, and improve software quality. CD carries on this process, automating the delivery of applications to selected infrastructure, development and testing environments.

Continuous testing can be implemented as a set of automated regression, performance and other tests executed in the CI/CD pipeline. Unlike in a monolith environment, where deployments are expensive and therefore infrequent, a CI/CD DevOps practice allows passing builds to be deployed directly to production environments on a daily or even hourly schedule.

Benefits of CI/CD pipelines
CI/CD pipelines are ideal for businesses that need to both improve applications frequently and maintain a reliable delivery process. By using this pipeline to standardize builds, develop tests and automate deployments, you facilitate a stable manufacturing process for deploying code changes.

This DevOps best practice helps address imbalances between developers who seek to push frequent changes, and operations that desire stable applications. By integrating automation, developers can push changes more frequently, in stable environments that provide greater operational stability.

Since all environments have standard configurations, the delivery process can include rigorous and continuous testing. Environment variables can be effortlessly separated from the application, and rollback procedures automated. This keeps your developers focused on the task of enhancing and improving applications, rather than on the system details of delivery.

Impact of implementing CI/CD pipelines can be reported on through predetermined DevOps key performance indicators (KPIs) such as deployment frequency, change lead time and incident mean time to recovery (MTTR).

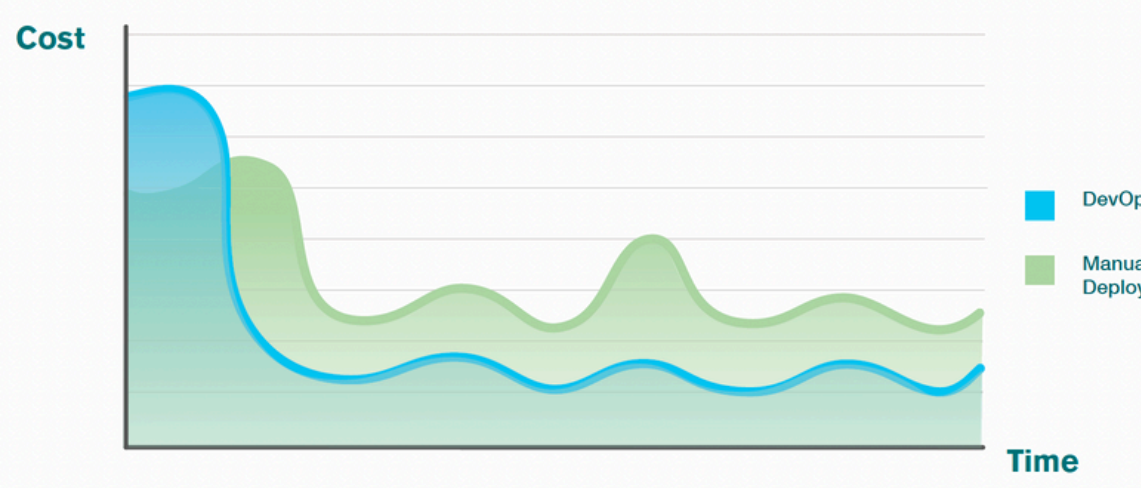
By treating the entire infrastructure and CI/CD pipeline as code, automation of many processes becomes possible, cutting down on demands on your developers and allowing you to speed up DevOps processes. It also enables you to implement version control branching.

Cost Benefits of a Deployment First approach
The initial costs of developing a CI/CD process are higher than manually building the infrastructure and releasing code. However, the ongoing cost benefits are far greater and can easily balance the investment.

An environment that is manually built requires manual patching on a regular basis to remain secure, up to date, and stable. Manual patching also requires downtime, and (in most cases) a manual restart if an environment ever falls over. Consistency between production, test and development environments can be difficult to maintain, potentially impacting production deployments with buggy software.

When you automate the build of your environments and software, you eliminate the need for manual patching and manual restarts. Development, test and production environments remain consistent, meaning less buggy code in production. In most cases, only minor ongoing changes will ever be required to the automation code.

Our experience has shown that the build cost of automation code is approximately 40% higher than manually built environments. However, completing manual changes to environments causes an ongoing cost exceeding 60% that of automated environments, and adds the risk of downtime or missed patches.



Version control
Traditionally, you'll select the desired strategy for defining protocols over how new code is merged into standard branches for development, testing and production, and create additional feature branches for those that require longer development cycles.

As each feature is completed, developers can merge the changes from feature branches into the primary development branch, but if multiple features are being developed concurrently, this process can get bogged down. If you don't have the ability to deploy regularly and developers are left to work on projects for weeks at a time, it increases the risk of bugs and longer cycle times to production.

With version control, your developers regularly commit all new work to the trunk, and release branches can be worked on and tested concurrently as main work continues on the trunk.

When the branch is frozen for final testing before release, it can be tagged as a reference snapshot, then packaged and released to customers. Release branches remain in maintenance mode, while tags represent final shipped versions.

Microservices
Migration opens up a lot of other options and how you deploy them. By containerizing, you can enable a deployment first culture, taking advantage of the offerings of AWS and enabling a modernization of workloads and workflows.

Microservices mean you can break up apps into lots of little pieces, and run them as isolated services. This has multiple benefits, chiefly the ability to scale independently, and avoid issues related to one piece breaking and causing the entire workflow to grind to a halt.

Instead, development teams can be broken up to work on smaller bits, allowing you to prioritize a specific customer or feature. Since each piece has its own deployment pipeline, you can contain problems and speed deployments, increasing velocity through continuous delivery to get features out the door faster.

Separated workflows
If the majority of the team is working steadily to develop and release a major feature as requested, but a new customer signs up with a specific condition or demand, you can advance bits of the new project without slowing down progress.

To quickly fast-track the necessary item, your CI/CD can spin up one or more new environments along the pipeline, and one or more developers, often called a feature team, can be assigned to go off onto that path. The production environment can continue to run, while the development environment can run more tests for performance and security easily without impacting overall productivity.

At the same time, you avoid having the larger team stepping on the feature teams' toes with the bigger application. This allows you to bring a level of control to your workflows and priorities that you can't achieve in a monolith environment. It is ideally suited for rapid feature releases.

Why Use Linux Containers?
Migrating to AWS and Linux containers delivers a broad range of operational benefits.

High-density, isolated application hosting
Linux containerization allows your applications to be bundled with their own libraries and configuration files. They can then be executed within the container, in isolation on a single OS kernel.

This high-density approach can significantly reduce costs associated with hosting, allowing you to run multiple applications or tasks on a single host while maintaining security and access to data. By using containers, you can minimize idle capacity and maximize overall resource utilization improving overall cost efficiency.

Cross-platform functionality
Containers also include application code or binaries, as well as any and all required dependencies. This capability for runtime packaging and seamless deployment makes it simple to migrate applications from one host to another.
Your applications can be readily tested to ensure that each one behaves consistently in all environments, whether it is being utilized on a developer laptop or a production environment. This allows for rapid feedback and correction for a faster release.

Container Orchestration
Container orchestration automates the deployment, management, scaling, and networking of containers. Container orchestration can be used in any environment where you use containers. It can help you to deploy the same application across different environments without needing to redesign it.

Orchestrators give you an abstraction layer on top of conventional hosting environments, so you can stop “running applications'' and control everything from the orchestrator. It will keep track of running containers, continually monitor the existing state, evaluate it against the expected state, and provide corrections as needed.

Should an application go down, the orchestrator automatically spins up another container to run your application on the next available host. This deployment abstraction that containers provide will enable you to focus on your applications, without splitting time and resources to handle dependencies with underlying hosts and infrastructure.

AWS provides many services that remove the heavy lifting from managing infrastructure and configuration, and the need to maintain internal expertise. ECS and EKS are container orchestrators that are managed by AWS requiring little to no time from your developers.

Resource management
Containerization delivers immutable resource management as an effective approach to run microservices and other types of distributed systems designed to continually improve infrastructure and application functionality.

When moving to Linux containers, you can offload host resource management onto AWS. This means your infrastructure is less of a burden operationally, as the platform can manage all hosts your applications are running on.

You may not even need to manage the OS, but if you do, Linux is more lightweight than Windows. With Linux containers, each service runs individually. If one service needs to scale up, you don’t have to scale up each service but can scale independently using small containers.

Immutability and repeatability
Linux containers deliver isolated, immutable infrastructures that are consistent, reliable, and predictable. This approach supports repeatability and reduces configuration drifts. Deployments are simplified since they don't have to support upgrade scenarios; every upgrade is simply a new deployment.

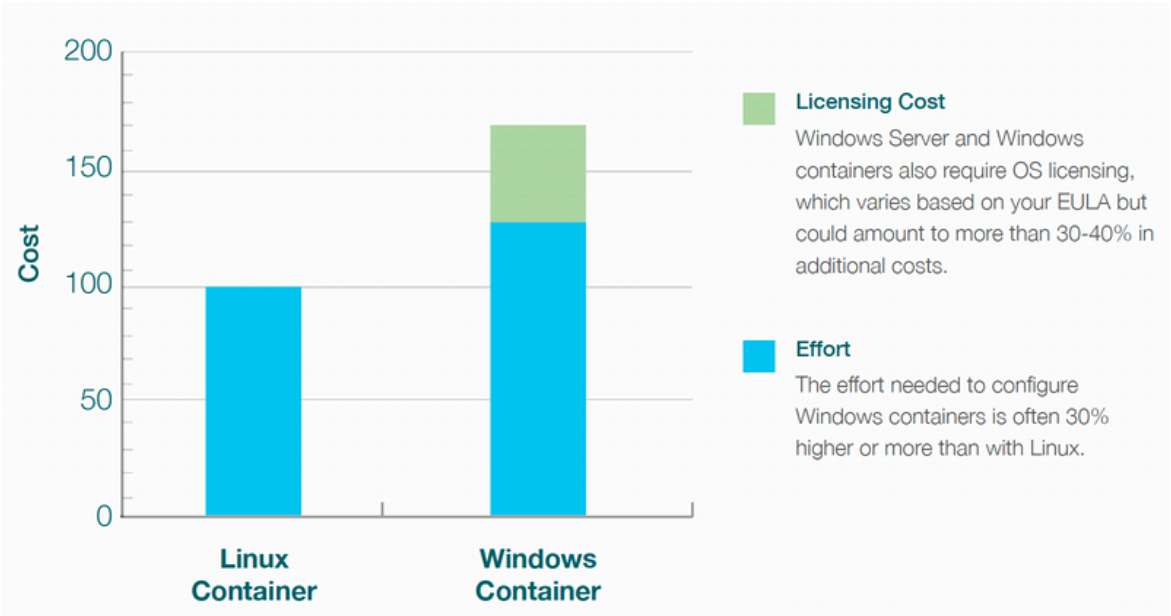
Since there are no differences between environments, testing and debugging are also simplified. The same process to deploy the new version is used to roll back to older versions, making deployments safer, and improving security.
The base images are consistent and repeatable, making it easy to scale on demand and implement auto-scaling for even faster growth and feature deployment. Taking advantage of a cloud-native architecture enhances all of these benefits.

Cost benefits of Linux containers
Linux containers provide significant cost benefits compared to Windows containers and Windows Server.

Linux containers are smaller in size than Windows containers and support most task definitions reusing the amount of additional scripting typically done in PowerShell. In addition, Linux containers have no licensing costs.

The effort needed to configure Windows containers is often 30% higher or more than with Linux, and requires skills not usually available in people with container experience.

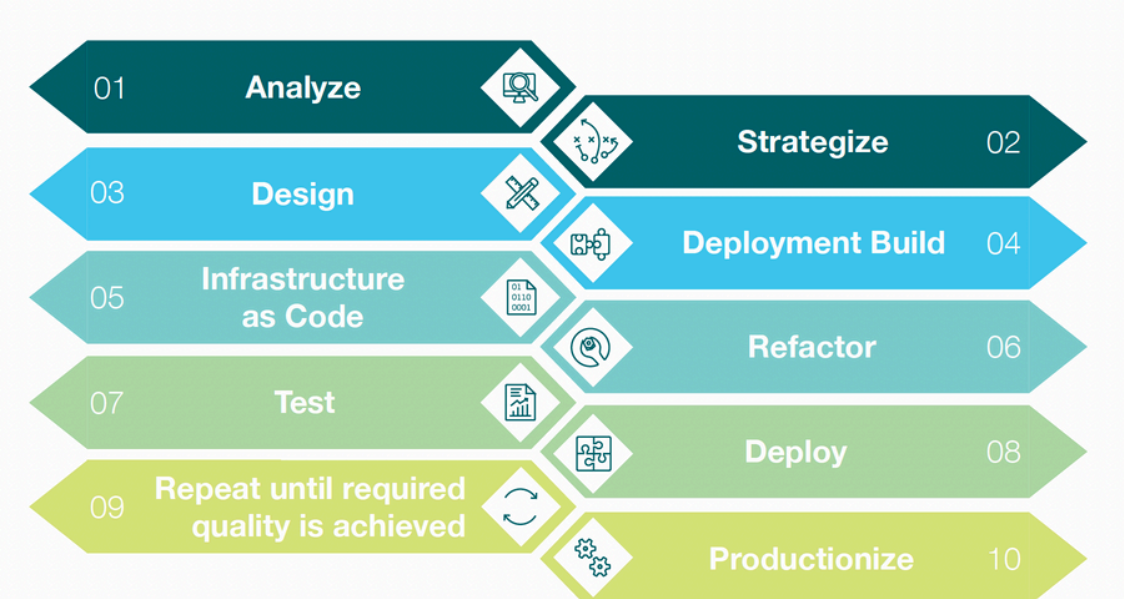
Windows Server and Windows containers also require OS licensing, which varies based on your End User License Agreement (EULA) but could amount to more than 30-40% in additional costs.



Why is a Cloud-Native Architecture Important?
Companies benefit from implementing a cloud-native architecture in multiple ways and across many dimensions. Primarily, a cloud-native architecture can minimize downtime, reduce infrastructure provisioning times, enhance scalability and performance, and cut down on operational tasks. All of your O/S and its support is handled by your cloud vendor, including scalability, uptime and patching. This alone is a strong argument for migrating to the cloud. Your team can focus on developing the business logic rather than having to configure the infrastructure.

While you can implement a cloud-native architecture yourself, the costs can be prohibitive. Overhead and hiring the right talent for implementation and ongoing maintenance can require continual justification for CAPEX. By outsourcing the implementation, and ongoing engagement and management, you can shift these costs to your OPEX budget, with an ongoing monthly fee as compared to a large annual expenditure.

Steps of Migrating to .Net
Completing the following steps can help ensure a flawless migration from older .NET versions (such as .NET Framework) to the latest version of .NET (previously known as .NET 5/Core).



1. Analyze
Scan your .NET Framework applications using the AWS Porting Assistant for .NET and generate a code compatibility assessment report, identifying dependencies and code that are incompatible with the latest .NET version, and estimating the level of effort involved for your migration.

2. Strategize
Formulate a roadmap to achieve a deployment first .NET migration. Ensure you give yourself enough time to code your infrastructure. You will need to refactor. Keep this in mind when setting yourself timelines.

3. Design
Designing your infrastructure is the next step. With containerization, you will be able to separate services easily. Remember this in the design phase so you can take full advantage of this option once you launch on the AWS cloud. Determining and agreeing on your branching and tagging strategies is the next step. It is critical that every developer is using the same conventions.

4. Deployment Build
The baseline CI/CD pipelines need to be put in place next, to guide workflows and allow automation where applicable.

5. Infrastructure as Code
The infrastructure should be designed and developed using code only. Manually configured infrastructure cannot be automated or repeated increasing the risk of failure. This standardizes your workflows for easy migration across environments, and enables fast, easy reproduction with the click of a button. Using code provides for reproducibility, recoverability, auditing, automation and change management.

6. Refactor
Make all necessary code changes to your application according to the assessment report. The assessment report provides your development team insight into exactly where to focus their efforts and outlines the expected effort in application code changes, such as updating packages and replacing libraries.

7. Test
Unit testing is simplified, since a side benefit of the latest .NET version is that the same container used in production can be run on a developer's laptop environment. This allows them to run the local version with all dependencies including database, to improve local development from their laptop.

8. Deploy
Create the test infrastructure from the infrastructure code templates. This can be achieved by using a Docker image and a CI/CD pipeline to complete the container image, then publishing that to the repository on AWS.

9. Repeat until required quality is achieved
Once your infrastructure is stabilized, you can complete the final testing, including penetration and vulnerability scans. Vulnerabilities will be identified in the pipeline, causing the deployment to fail and give feedback, and if anything breaks, it can be fixed before the next deployment attempt.

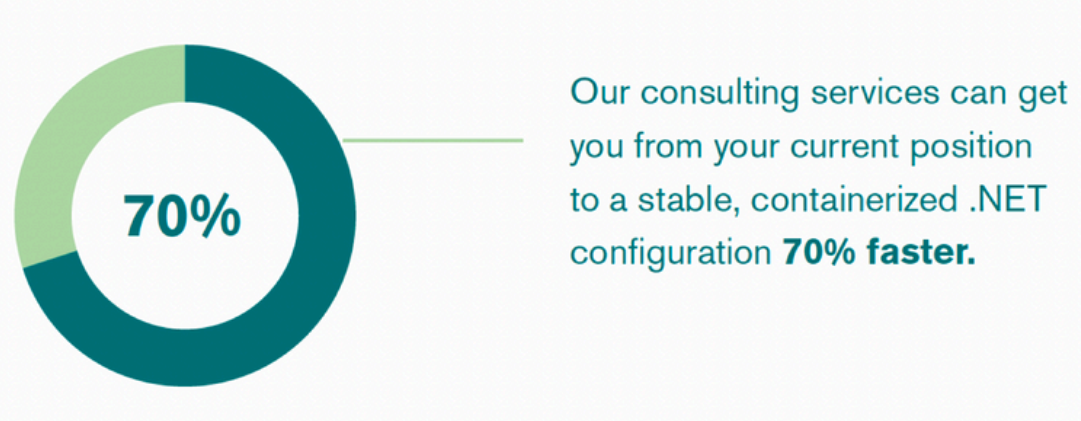
10. Productionize
The final step is building the production infrastructure. By deploying workloads on AWS using CI/CD you can be assured consistency and reliability of your deployment is achieved.

Before going live, be sure that you have your monitoring and security in place. Containerizing with Linux and adoption of a deployment first strategy can deliver multiple benefits and provide a foundation for long-term stability.

Benefits of Linux and Containerization
This approach is all about automation. The more you can automate, the better your result will be. Removing the need to manually deploy, patch or maintain uptime is a significant saving in time and cost.

Additionally, with the reduced size of Linux containers, you will find that scaling in this environment is much quicker, reducing the time your customers need to wait for a performing system.

Deployment first strategies introduce repeatability, scalability and automation. These features assist in the ongoing performance of the system, but also the ongoing ability for your developers to continually develop, without impact. Developers are not delayed by long periods of waiting for changes with the introduction of feature teams and isolated environments.



Many organisations have already started to move to deployment first Linux container architectures for .NET. Not only have they achieved scalability and improved developer throughput, they have also discovered considerable cost savings.

If you think it's time for migration, but don't feel prepared or equipped to handle it yourself, we can help. Our consulting services can get you from your current position to a stable, containerized .NET configuration 70% faster. Then, you can start to enjoy the benefits of deployment first, Linux-based architecture.

Get in touch

BASE2 Services is a global leader in Cloud Delivery, Operations and Management. Specialising in DevOps, AWS and cloud-native computing.

We enable organisations to accelerate innovation through automated, highly secure, repeatable and scalable cloud-based solutions. Contact us today to find out how we can help you.

USA

11801 Domain Blvd
3rd Floor
Austin TX 78758
+1 646 586 9485

AUSTRALIA

Level 3
44 Clifton St
Prahran VIC 3181
1300 713 559

GERMANY

4. OG
Potsdamer Str. 182
10783 Berlin
+49 30 2000 5370

info@base2services.com

This document and all its components are protected by copyright. No part may be reproduced, copied or transmitted in any form or by any means without the prior written permission of base2Services Pty Ltd.

base2services.com.au