**Zero-Downtime\* Web Apps for ASP .NET Core 3.1**

By Shahed C on June 29, 2020

[2 Replies](https://wakeupandcode.com/zero-downtime-web-apps-for-asp-net-core-3-1/#comments)



This is the twenty-sixth of a new [series of posts](https://wakeupandcode.com/aspnetcore/#aspnetcore2020) on ASP .NET Core 3.1 for 2020. In this series, we’ve covered 26 topics over a span of 26 weeks from January through June 2020, titled **ASP .NET Core A-Z!** To differentiate from the [2019 series](https://wakeupandcode.com/aspnetcore/#aspnetcore2019), the 2020 series mostly focused on a growing single codebase ([NetLearner!](https://wakeupandcode.com/netlearner-on-asp-net-core-3-1/)) instead of new unrelated code snippets week.

Previous post:

* [YAML-defined CI/CD for ASP .NET Core 3.1](https://wakeupandcode.com/yaml-defined-ci-cd-for-asp-net-core-3-1/)

**NetLearner on GitHub**:

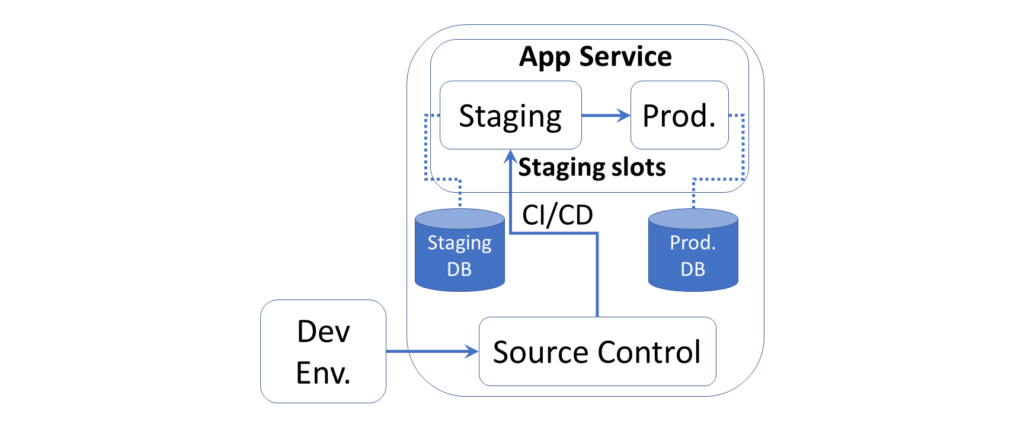
* Repository: <https://github.com/shahedc/NetLearnerApp>
* v0.26-alpha release: <https://github.com/shahedc/NetLearnerApp/releases/tag/v0.26-alpha>

**In this Article:**

* Z is for Zero-Downtime\* Web Apps
* Availability
* Backup & Restore
* Continuous Integration & Continuous Deployment
* Deployment Slots
* EF Core Migrations in Production
* Feature Flags
* References

**Z is for Zero-Downtime\* Web Apps for ASP .NET Core**

If you’ve made it this far in this ASP .NET Core A-Z series, hopefully you’ve learned about many important topics related to ASP .NET Core web application development. As we wrap up this series with a look at tips and tricks to *attempt* zero-downtime, this last post itself has its own lettered A-F mini-series: **A**vailability, **B**ackup & Restore, **C**I/CD, **D**eployment Slots, **E**F Core Migrations and **F**eature Flags.

Deploying to Staging Slots in AsuApp Service

**NOTE**: While it may not be possible to get 100% availability 24/7/365, you can ensure a user-friendly experience free from (or at least with minimal) interruptions, by following a combination of the tips and tricks outlined below. This write-up is not meant to be a comprehensive guide. Rather, it is more of an outline with references that you can follow up on, for next steps.

**Availability**

To improve the availability of your ASP .NET Core web app running on Azure, consider running your app in multiple regions for HA (High Availability). To control traffic to/from your website, you may use [Traffic Manager](https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-overview) to direct web traffic to a standby/secondary region, in case the primary region is unavailable.

Consider the following 3 options when running a web app in multiple Azure regions. In these scenarios, the primary region is always active and the secondary region may be passive (as a hot or cold standby) or active. When both are active, web requests are load-balanced between the two regions.

|  |  |  |
| --- | --- | --- |
| **Options** | **Primary Region** | **Secondary Region** |
| A | Active | Passive, Hot Standby |
| B | Active | Passive, Cold Standby |
| C | Active | Active |

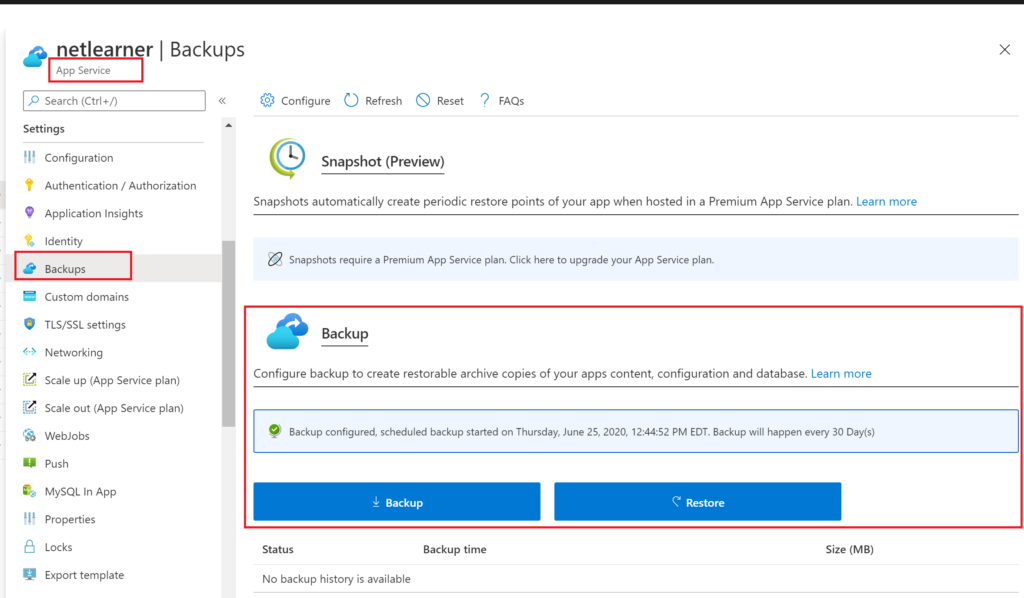
For more information on achieving High Availability (HA) in multiple regions, check out the official docs at:

* HA multi-region web apps: <https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/app-service-web-app/multi-region>

If you’re running your web app in a Virtual Machine (VM) instead of Azure App Service, you may also consider [Availability Sets](https://docs.microsoft.com/en-us/azure/virtual-machines/windows/manage-availability). This helps build redundancy in your Web App’s architecture, when you have 2 or more VMs in an Availability Set. For added resiliency, use [Azure Load Balancer](https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-overview) with your VMs to load-balance incoming traffic. As an alternative to Availability Sets, you may also use [Availability Zones](https://docs.microsoft.com/en-us/azure/availability-zones/az-overview) to counter any failures within a datacenter.

**Backup & Restore**

Azure’s App Service lets you [back up and restore](https://docs.microsoft.com/en-us/azure/app-service/manage-backup) your web application, using the Azure Portal or with [Azure CLI commands](https://docs.microsoft.com/en-us/azure/app-service/scripts/cli-backup-onetime). Note that this requires your App Service to be in at least the Standard or Premium tier, as it is not available in the Free/Shared tiers. You can create backups on demand when you wish, or schedule your backups as needed. If your site goes down, you can quickly restore your last good backup to minimize downtime.

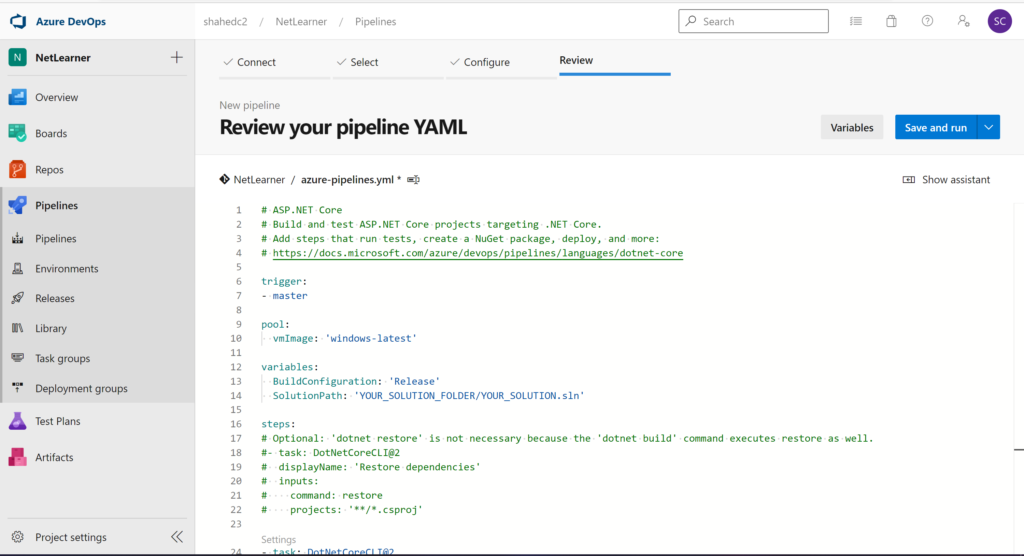
App Service: Backups

**NOTE**: You may use the new Snapshots feature to *“automatically create perioidic restore points of your app”*. However, this feature is only available in a Premium App Service Plan.

In addition to the app itself, the backup process also backs up the Web App’s configuration, file contents and the database connected to your app. Database types include SQL DB (aka SQL Server PaaS), MySQL and PostgreSQL. Note that these backups include a *complete* backup, and not incremental/delta backups.

**Continuous Integration & Continuous Deployment**

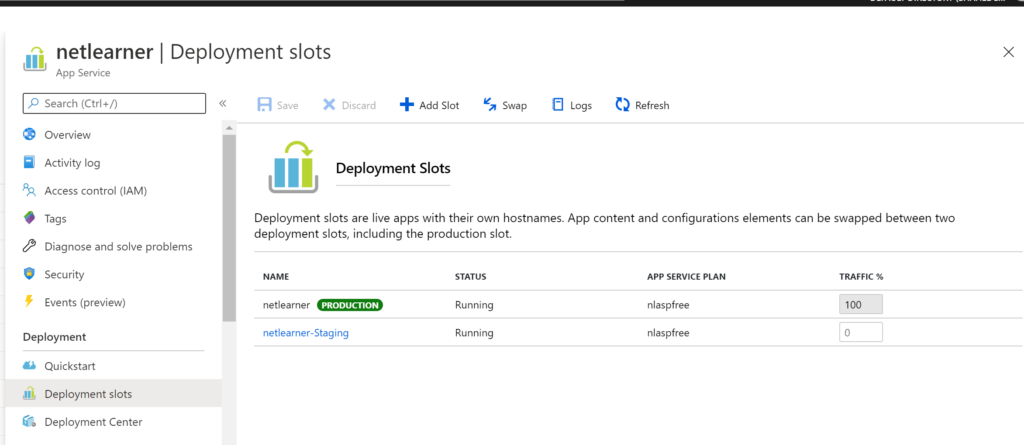
In the [previous post](https://wakeupandcode.com/yaml-defined-ci-cd-for-asp-net-core-3-1/), we covered CI/CD with YAML pipelines. Whether you have to fix an urgent bug quickly or just deploy a planned release, it’s important to have a proper CI/CD pipeline. This allows you to deploy new features and fixes quickly with minimal downtime.

YAML Sample in Azure DevOps

* YAML Sample: <https://github.com/shahedc/NetLearnerApp/blob/main/azure-pipelines.yml.txt>

**Deployment Slots**

Whether you’re deploying your Web App to App Service for the first time or the 100th time, it helps to test out your app before releasing to the public. [Deployment slots](https://docs.microsoft.com/en-us/azure/app-service/deploy-staging-slots) make it easy to set up a *Staging Slot*, warm it up and swap it immediately with a *Production Slot*. Swapping a slot that has already been warmed up ahead of time will allow you to deploy the latest version of your Web App almost immediately.

Deployment Slots in Azure App Service

Note that Deployment Slots are only available in Standard, Premium or Isolated App Service tiers, not in the Free/Shared tiers. You can combine Deployment Slots with your CI/CD pipelines to ensure that your automated deployments end up in the intended slots.

**EF Core Migrations in Production**

We covered [EF Core Migrations](https://wakeupandcode.com/ef-core-migrations-in-asp-net-core/) in a previous post, which is one way of upgrading your database in various environments (including production). *But wait, is it safe to run EF Core Migrations in a production environment?*Even though you can use auto-generated EF Core migrations (written in C# or outputted as SQL Scripts), you may also modify your migrations for your needs.

I would highly recommend reading [Jon P Smith](https://twitter.com/thereformedprog)‘s two-part series on “Handling Entity Framework Core database migrations in production”:

* Part 1 of 2: <https://www.thereformedprogrammer.net/handling-entity-framework-core-database-migrations-in-production-part-1/>
* Part 2 of 2: <https://www.thereformedprogrammer.net/handling-entity-framework-core-database-migrations-in-production-part-2/>

What you decide to do is up to you (and your team). I would suggest exploring the different options available to you, to ensure that you minimize any downtime for your users. For any *non-breaking* DB changes, you should be able to migrate your DB easily. However, your site may be down for maintenance for any *breaking* DB changes.

**Feature Flags**

Introduced by the Azure team, the [Microsoft.FeatureManage](https://www.nuget.org/packages/Microsoft.FeatureManagement/)[ment](http://microsoft.featuremanagement/) package allows you to add [Feature Flags](https://docs.microsoft.com/en-us/azure/azure-app-configuration/quickstart-feature-flag-aspnet-core) to your .NET application. This enables your web app to include new features that can easily be toggled for various audiences. This means that you could potentially test out new features by deploying them during off-peak times, but toggling them to become available via app configuration.

To install the package, you may use the following **dotnet** command:

>dotnet add package Microsoft.FeatureManagement --version 2.0.0

If you prefer the Package Manager Console in Visual Studio, you may also use the following PowerShell command:

>Install-Package Microsoft.FeatureManagement -Version 2.0.0

By combining many/all of the above features, tips and tricks for your Web App deployments, you can release new features while minimizing/eliminating downtime. If you have any new suggestions, feel free to leave your comments.

**References**

* Highly available multi-region web application: <https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/app-service-web-app/multi-region>
* Design reliable Azure applications: <https://docs.microsoft.com/en-us/azure/architecture/reliability/>
* Manage the availability of Windows VMs in Azure: <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/manage-availability>
* What is Azure Load Balancer? <https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-overview>
* SLA for VMs: <https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/>
* Back up app – Azure App Service: <https://docs.microsoft.com/en-us/azure/app-service/manage-backup>
* Azure CLI Script Sample – Back up an app: <https://docs.microsoft.com/en-us/azure/app-service/scripts/cli-backup-onetime>
* CI/CD with Release pipelines: <https://docs.microsoft.com/en-us/azure/devops/pipelines/release>
* Continuous deployment – Azure App Service: <https://docs.microsoft.com/en-us/azure/app-service/deploy-continuous-deployment>
* Set up staging environments for web apps in Azure App Service: <https://docs.microsoft.com/en-us/azure/app-service/deploy-staging-slots>
* Handling Entity Framework Core database migrations in production: <https://www.thereformedprogrammer.net/handling-entity-framework-core-database-migrations-in-production-part-1/>
* Handling Entity Framework Core database migrations in production – Part 2: <https://www.thereformedprogrammer.net/handling-entity-framework-core-database-migrations-in-production-part-2/>
* Tutorial for using feature flags in a .NET Core app: <https://docs.microsoft.com/en-us/azure/azure-app-configuration/use-feature-flags-dotnet-core>
* Quickstart for adding feature flags to ASP.NET Core: <https://docs.microsoft.com/en-us/azure/azure-app-configuration/quickstart-feature-flag-aspnet-core>