Azure App Configuration can be used with ASP.NET Core applications to centrally manage and externalize configuration settings, including feature flags. It offers a .NET configuration provider library, enabling you to add App Configuration as a configuration source with minimal code changes.

Here's how to integrate Azure App Configuration with your ASP.NET Core app:

1. **Create an App Configuration store:** Start by creating an App Configuration store in the Azure portal.
2. **Install the NuGet package:** Add the Microsoft.Extensions.Configuration.AzureAppConfiguration NuGet package to your project.
3. **Configure your ASP.NET Core app:** In your app's Program.cs or Startup.cs file, add the App Configuration provider to the configuration builder:

Code

using Microsoft.Extensions.Configuration;  
  
 var builder = WebApplication.CreateBuilder(args);  
 builder.Configuration.AddAzureAppConfiguration(options => {  
 options.Connect(builder.Configuration["AppConfiguration:ConnectionString"]);  
 options.Configure(builder.Configuration["AppConfiguration:SentinelKey"]); *//Optional*  
 options.UseFeatureFlags(); *//Optional*  
 *//You can also use an Identity,*   
 *//instead of the connection string*  
 *//options.UseAzureIdentity(builder.Configuration["AppConfiguration:IdentityUrl"]);*  
 *//You can use options to configure the connection string*  
 *//options.ConfigureKeyValue(kv => {*  
 *// return true; //This will be evaluated to True or False, can be a function*  
 *//});*  
  
 *//If you want to specify a label.*  
 *//options.Select(x => x.Label == "Production");*  
  
 *//Options to configure the behavior of the sentinel key*  
 *//options.Retry(options => {*  
 *//});*  
  
 *//It supports different types of authentication, with different methods*  
 });  
  
 var app = builder.Build();

1. **Access the configuration:** You can then access your configuration settings in your controllers, services, or other components using the IConfiguration interface.

Code

*// In a controller*  
 public class MyController : Controller  
 {  
 private readonly IConfiguration \_configuration;  
  
 public MyController(IConfiguration configuration)  
 {  
 \_configuration = configuration;  
 }  
  
 public IActionResult Index()  
 {  
 string appSetting = \_configuration["MySetting"];  
 *// Access feature flag (if enabled)*  
 bool isFeatureEnabled = \_configuration.GetSection("FeatureFlags").Get<Dictionary<string, bool>>()["MyFeature"];  
 return View(new { AppSetting = appSetting, IsFeatureEnabled = isFeatureEnabled });  
 }  
 }

1. **Enable Dynamic Configuration (Optional):** For dynamic configuration, Microsoft recommends using SentinelKey. You can use the Configure method in Program.cs to specify the sentinel key.

Benefits of using Azure App Configuration:

* **Centralized configuration:** Store and manage your application settings in a single place.
* **Secure storage:** App Configuration provides robust security features.
* **Dynamic configuration:** Modify settings without redeploying your application.
* **Feature flags:** Enable or disable features dynamically.
* **Consistent configuration across environments:** Share settings across multiple environments.

<https://kumarashwinhubert.com/secure-your-azure-functions-use-managed-identity-for-azurewebjobsstorage>