Native AOT deployment

Article • 09/12/2023

Publishing your app as *Native AOT* produces an app that's self-contained and that has been ahead-of-time (AOT) compiled to native code. Native AOT apps have faster startup time and smaller memory footprints. These apps can run on machines that don't have the .NET runtime installed.

The benefit of Native AOT is most significant for workloads with a high number of deployed instances, such as cloud infrastructure and hyper-scale services. .NET 8 adds ASP.NET Core support for native AOT.

The Native AOT deployment model uses an ahead-of-time compiler to compile IL to native code at the time of publish. Native AOT apps don't use a just-in-time (JIT) compiler when the application runs. Native AOT apps can run in restricted environments where a JIT isn't allowed. Native AOT applications target a specific runtime environment, such as Linux x64 or Windows x64, just like publishing a self-contained app.

Limitations in the .NET Native AOT deployment model

.NET 8+

AOT support in .NET 8 is more comprehensive than in .NET 7. However, there are still some limitations. For more information, see Limitations of Native AOT deployment.

Prerequisites

Windows

Visual Studio 2022 , including the **Desktop development with C++** workload with all default components.

Publish Native AOT using the CLI

1. Add `<PublishAot>true</PublishAot>` to your project file.

This property enables Native AOT compilation during publish. It also enables dynamic code-usage analysis during build and editing. It's preferable to put this setting in the project file rather than passing it on the command line, since it controls behaviors outside publish.

XML

```xml
<PropertyGroup>
    <PublishAot>true</PublishAot>
</PropertyGroup>
```

2. Publish the app for a specific runtime identifier using `dotnet publish -r <RID>`.

   The following example publishes the app for Windows as a Native AOT application on a machine with the required prerequisites installed.

   ```
   dotnet publish -r win-x64 -c Release
   ```

   The following example publishes the app for Linux as a Native AOT application. A Native AOT binary produced on Linux machine is only going to work on same or newer Linux version. For example, Native AOT binary produced on Ubuntu 20.04 is going to run on Ubuntu 20.04 and later, but it isn't going to run on Ubuntu 18.04.

   ```
   dotnet publish -r linux-arm64 -c Release
   ```

The app is available in the publish directory and contains all the code needed to run in it, including a stripped-down version of the coreclr runtime.

Check out the Native AOT samples available in the dotnet/samples repository on GitHub. The samples include Linux and Windows Dockerfiles that demonstrate how to automate installation of prerequisites and publish .NET projects with Native AOT using containers.

AOT-compatibility analyzers

.NET 8+

The `IsAotCompatible` property is used to indicate whether a library is compatible with Native AOT. Consider when a library sets the `IsAotCompatible` property to `true`, for example:

XML

```
<PropertyGroup>
    <IsAotCompatible>true</IsAotCompatible>
</PropertyGroup>
```

The preceding configuration assigns a default of `true` to the following properties:

- `IsTrimmable`
- `EnableTrimAnalyzer`
- `EnableSingleFileAnalyzer`
- `EnableAotAnalyzer`

These analyzers help to ensure that a library is compatible with Native AOT.

Native debug information

.NET 8+

By default, Native AOT publishing produces debug information in a separate file:

- Linux: *.dbg*
- Windows: *.pdb*
- macOS: *.dSYM* folder

The debug file is necessary for running the app under the debugger or inspecting crash dumps. On Unix-like platforms, set the `StripSymbols` property to `false` to include the debug information in the native binary. Including debug information makes the native binary larger.

XML

```
<PropertyGroup>
    <StripSymbols>false</StripSymbols>
</PropertyGroup>
```

Limitations of Native AOT deployment

Native AOT apps have the following limitations:

- No dynamic loading, for example, `Assembly.LoadFile`.
- No run-time code generation, for example, `System.Reflection.Emit`.

- No C++/CLI.
- Windows: No built-in COM.
- Requires trimming, which has limitations.
- Implies compilation into a single file, which has known incompatibilities.
- Apps include required runtime libraries (just like self-contained apps, increasing their size as compared to framework-dependent apps).
- System.Linq.Expressions always use their interpreted form, which is slower than run-time generated compiled code.
- Not all the runtime libraries are fully annotated to be Native AOT compatible. That is, some warnings in the runtime libraries aren't actionable by end developers.

The publish process analyzes the entire project and its dependencies for possible limitations. Warnings are issued for each limitation the published app may encounter at run time.

Version specific limitations

.NET 8+

- Diagnostic support for debugging and profiling with some limitations.
- Support for some ASP.NET Core features. For more information, see ASP.NET Core support for Native AOT.

Build native libraries

Publishing .NET class libraries as Native AOT allows creating libraries that can be consumed from non-.NET programming languages. The produced native library is self-contained and doesn't require a .NET runtime to be installed.

Publishing a class library as Native AOT creates a native library that exposes methods of the class library annotated with UnmanagedCallersOnlyAttribute with a non-null `EntryPoint` field. For more information, see the native library sample available in the dotnet/samples repository on GitHub.

Platform/architecture restrictions

The following table shows supported compilation targets.

.NET 8+

Expand table

Platform	Supported architecture	Notes
Windows	x64, Arm64	
Linux	x64, Arm64	
macOS	x64, Arm64	
iOS	Arm64	Experimental support
iOSSimulator	x64, Arm64	Experimental support
tvOS	Arm64	Experimental support
tvOSSimulator	x64, Arm64	Experimental support
MacCatalyst	x64, Arm64	Experimental support
Android	x64, Arm64	Experimental, no built-in Java interop

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback