Web APIs mit ASP.NET Core Native AOT

DWX 2024

think tecture

Kenny Pflug
@feO2x
Software Architect Consultant

Today's menu

- What is .NET Native AOT?
- About performance: startup time, memory consumption, app size, throughput in ASP.NET Core
- Compatibilities and Incompatibilities
- Q&A

think tecture

Kenny Pflug

Consultant Software Architect @ Thinktecture AG

- Distributed Systems with ASP.NET Core
- .NET internals
- Cloud-native

kenny.pflug@thinktecture.com @feO2x https://www.thinktecture.com

Let‘s go native (1)

To enable native AOT, set **<PublishAot>** to true in your csproj file.

This will enable compilation to native code during **dotnet publish**.

If you do not require a specific culture in your backend, consider setting **<InvariantGlobalization>** to true, too. This will strip out any globalization features except the invariant culture.

```
<Project Sdk="Microsoft.NET.Sdk.Web">

    <PropertyGroup>
        <TargetFramework>net8.0</TargetFramework>
        <Nullable>enable</Nullable>
        <InvariantGlobalization>true</InvariantGlobalization>
        <PublishAot>true</PublishAot>
    </PropertyGroup>

</Project>
```

Let‘s go native (2)

In your Program.cs file, use **WebApplication.CreateSlimBuilder** to create an instance of **WebApplicationBuilder**.

This will strip the following features:

- Fewer logging providers: Windows Event Log, ETW and Linux LTTng, as well as debugger console are left out
- No support for static web assets in referenced projects
- No IIS integration
- No HTTPS
- No Quic (HTTP/3) support
- No Regex or alpha constraints in Minimal API routing

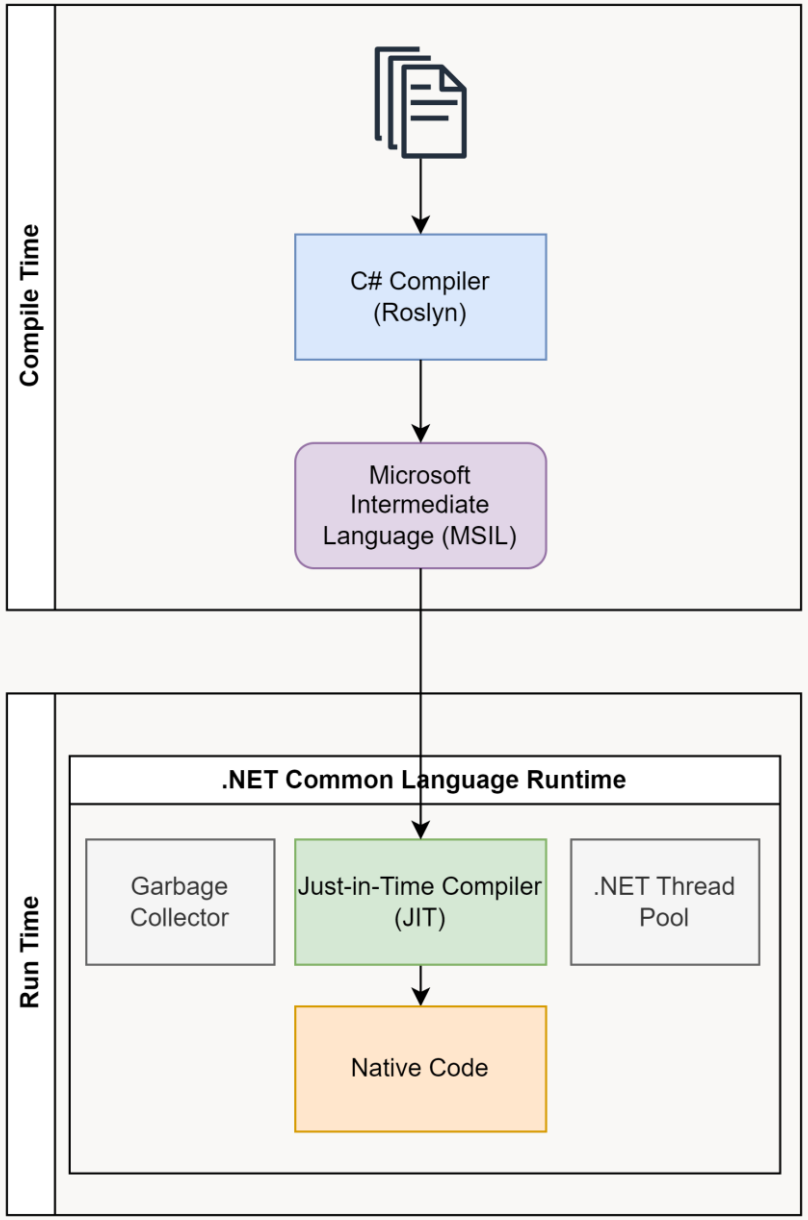
These features can be re-enabled.

```
public static async Task<int> Main(string[] args)
{
    Log.Logger = Logging.CreateBootstrapLogger();
    try
    {
        var app = WebApplication
            .CreateSlimBuilder(args)
            .ConfigureServices(Log.Logger)
            .Build()
            .ConfigureMiddleware();
        await app.RunAsync();
        return 0;
    }
    catch (Exception e)
    {
        Log.Fatal(e, "Could not run web app");
        return 1;
    }
    finally
    {
        await Log.CloseAndFlushAsync();
    }
}
```

Publishing the regular .NET way

When publishing a .NET app the regular way, your Code is compiled by the C# Compiler (csc, Roslyn) to Microsoft Intermediate Language (**MSIL**).

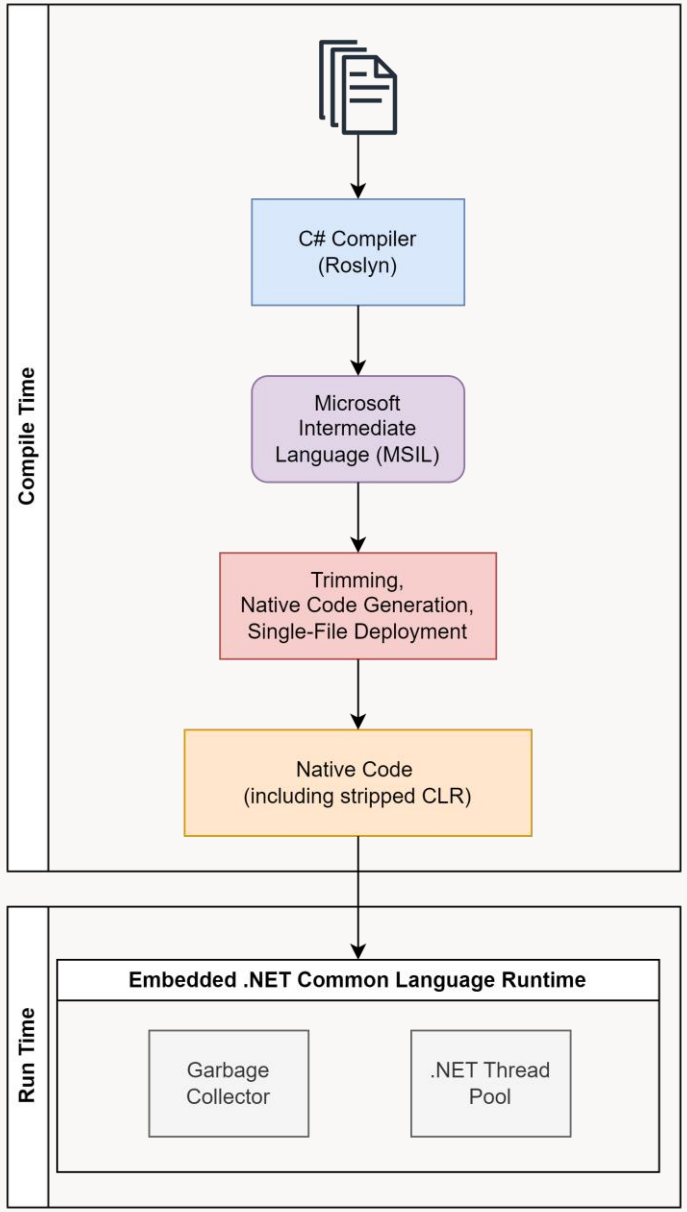
During execution, the Common Language Runtime (CLR) is started which has one important component: the Just-in-Time Compiler (**JIT**). It will take MSIL code and compile it to the target platform we are currently running on (compile once - run anywhere, compilation tiers).



Publishing with Native AOT

When publishing with **<PublishAot>** set to true, additional steps will be performed:

- Trimming MSIL code: via static code analysis, uncalled members in MSIL code will be removed to reduce the overall size of code
- Native Code generation: MSIL is now compiled to a target platform using a native compiler.
- The resulting native code is bundled into a single file. A stripped version of the CLR is included. The JIT no longer exists during run time, amongst other things.

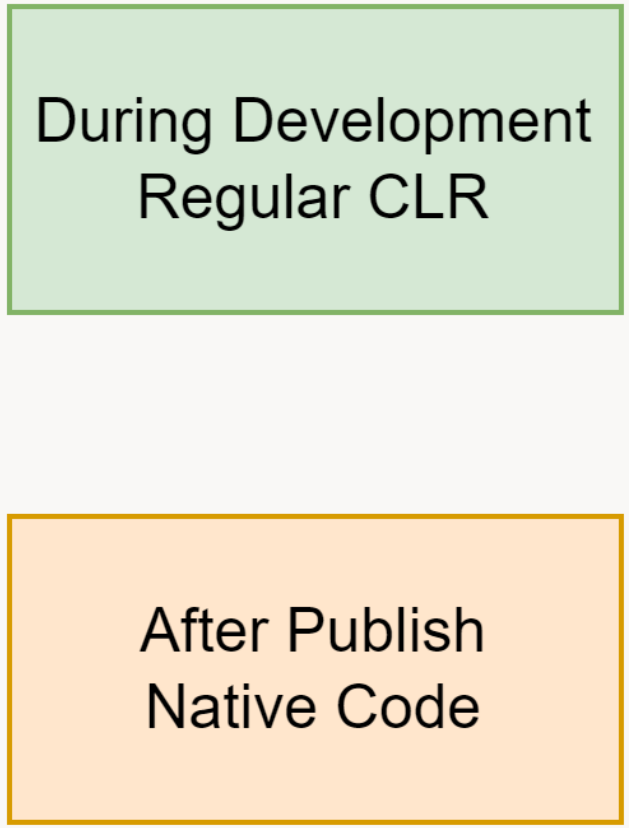



Demo Time
Publishing with Native AOT

During development, there is no native code

Native code will only be generated when you run **dotnet publish**.

- During development, you only use commands like **dotnet build, dotnet run** or **dotnet test**
- These do not produce native code: you will use the regular CLR in these scenarios
- Microsoft incorporates Roslyn analyzers to tell you whether some parts of your code are incompatible with Native AOT
- Currently, there is no test framework that supports native code out of the box

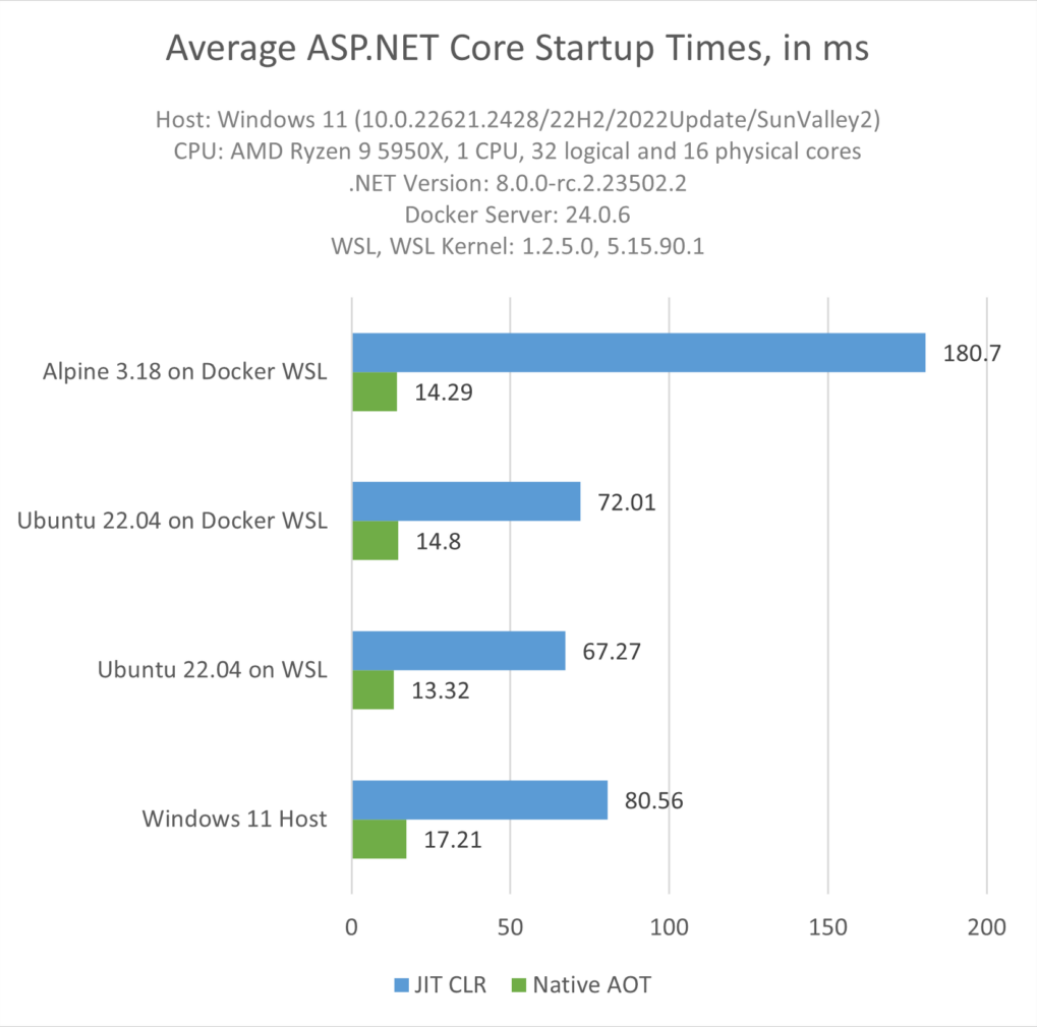


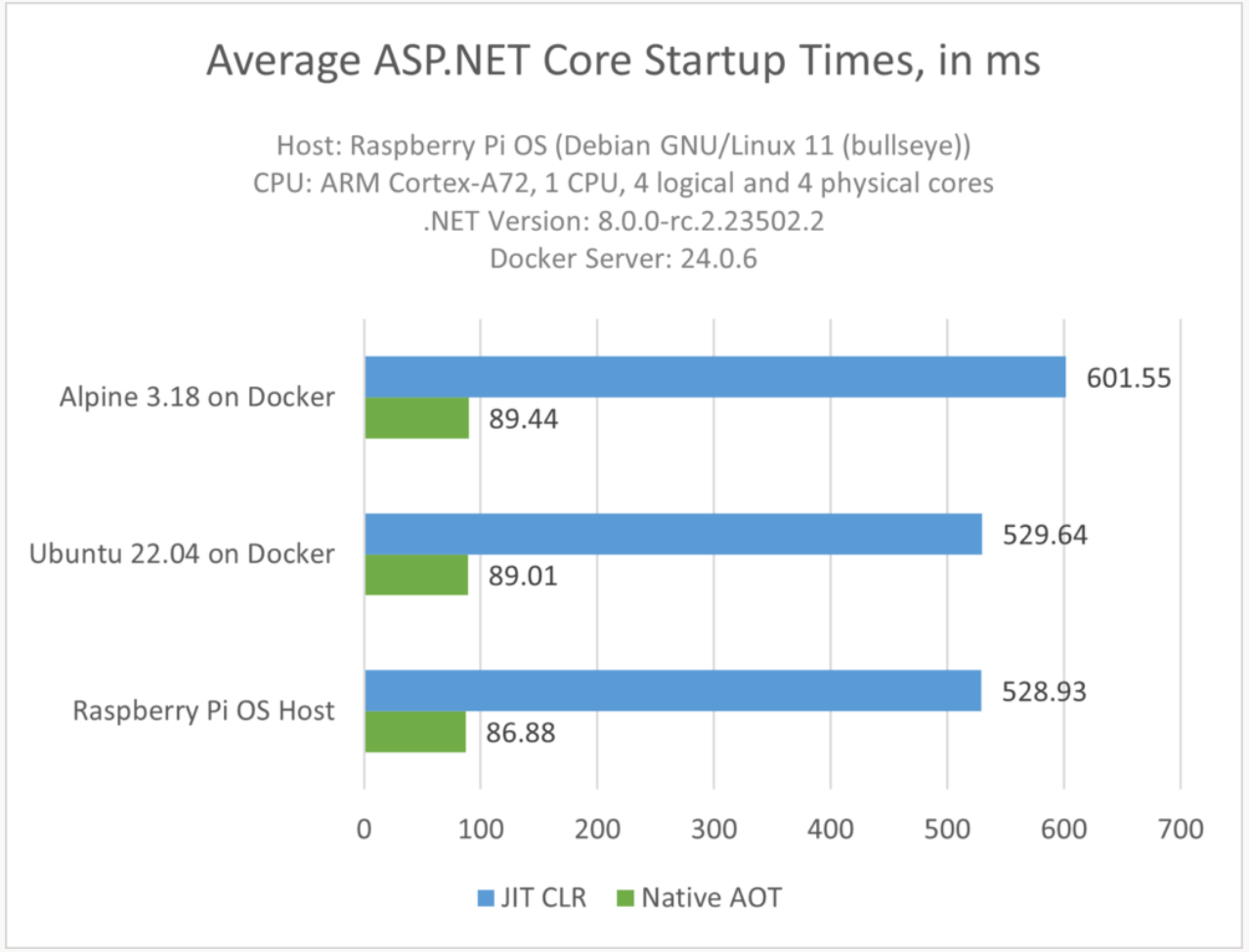
think tecture

ASP.NET Core Native AOT Performance Benchmarks

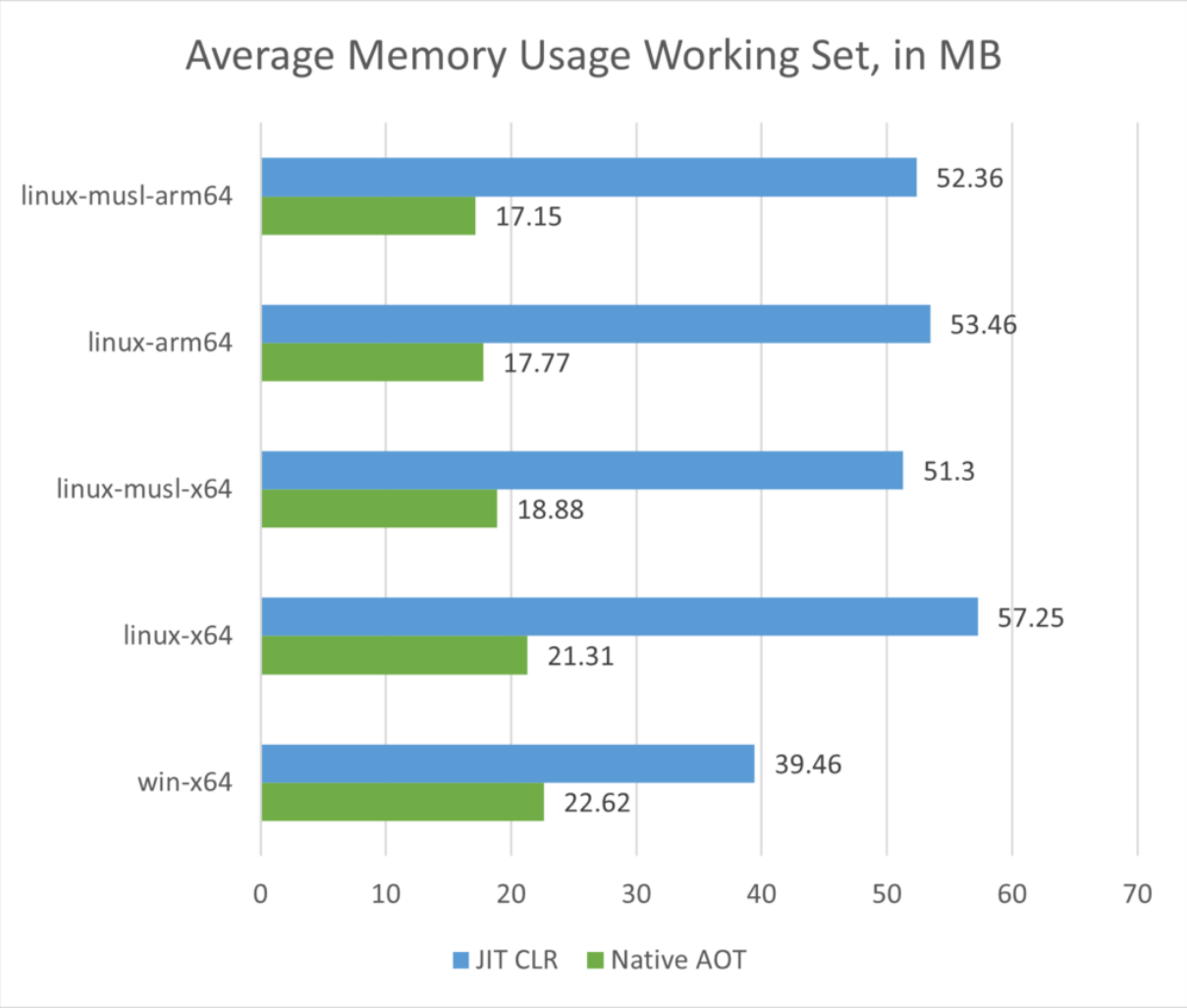

Demo Time
A look at the benchmark code

Average Startup Time

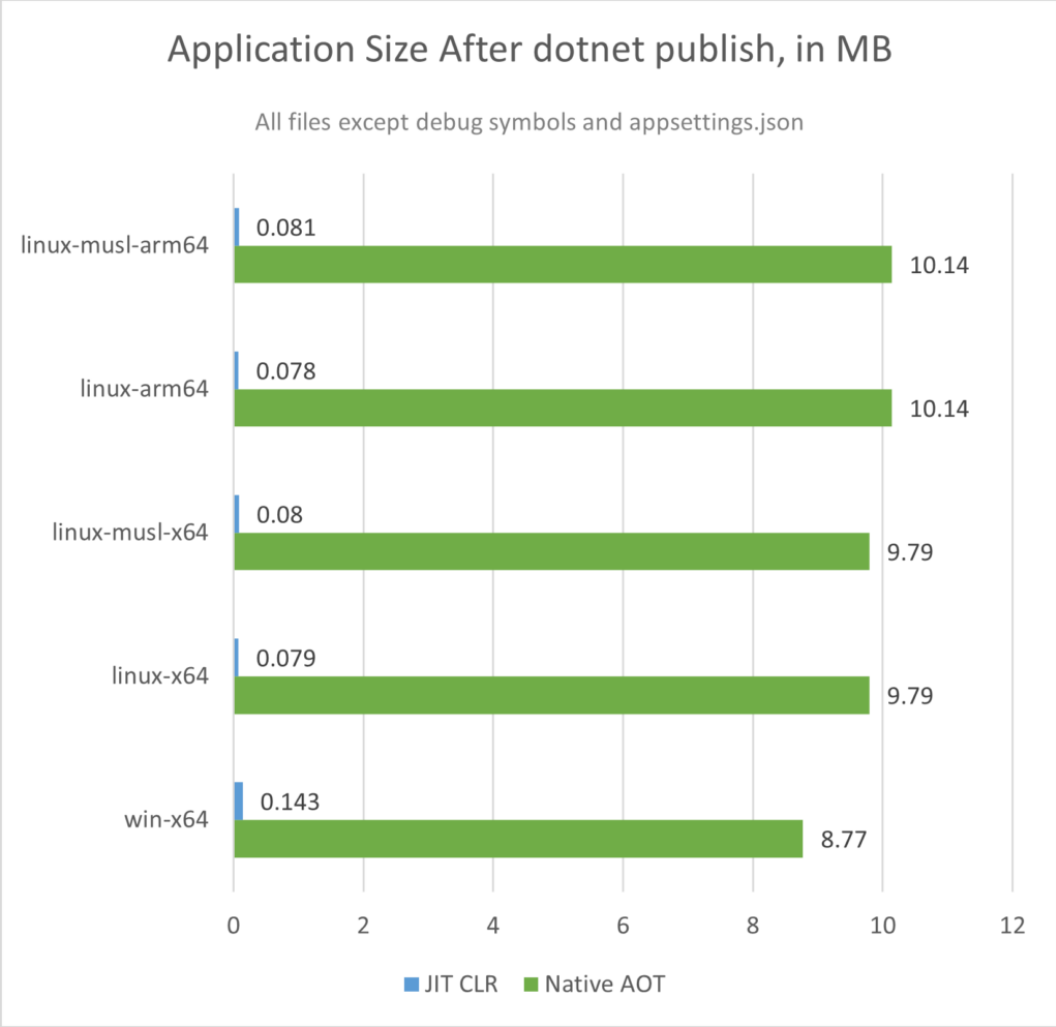


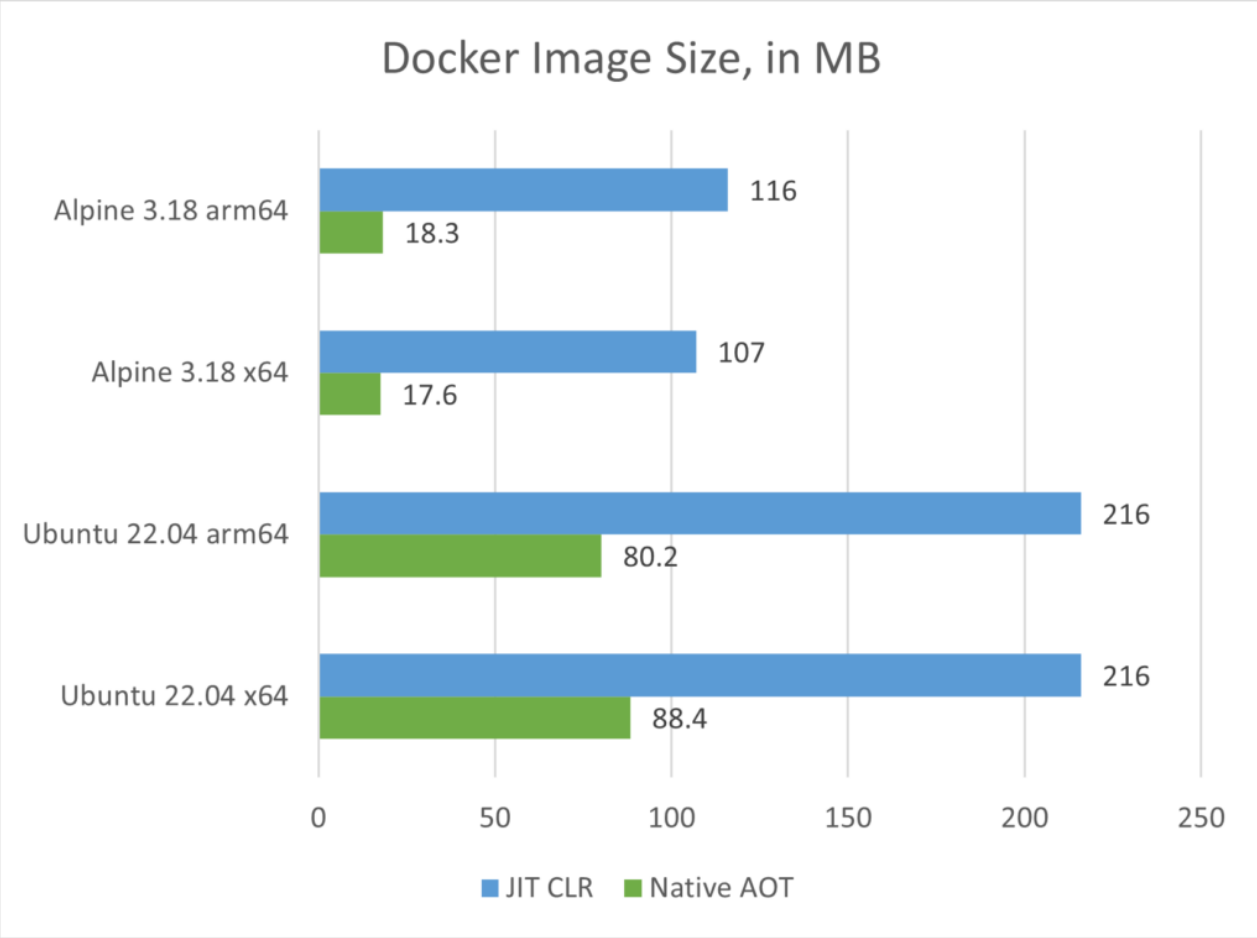


Average Memory Usage after Startup



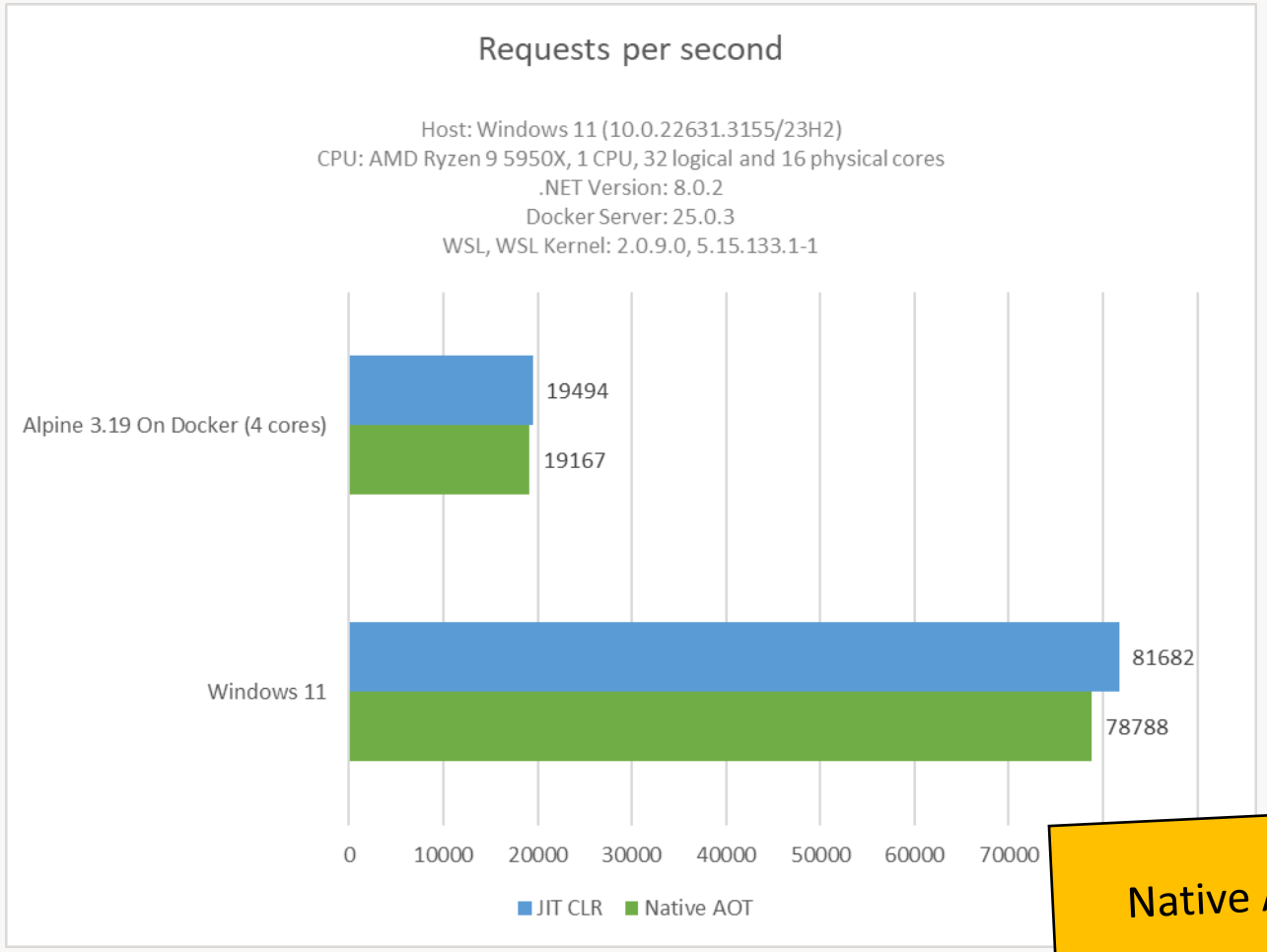
Application Size after dotnet publish






Demo Time
Another look at other benchmark code

Requests per second under spike load



Native AOT is 3.5% to 1% slower

Why slower?

- Dynamic Profile-Guided Optimization (PGO) is activated by default in .NET 8 but can only be used with the JIT at run time.
- JIT can also detect hardware capabilities at runtime and optimize for that – precompiled Native AOT code targets highly compatible target instructions
- The new Dynamic Adaptation To Application Sizes (DATAS) mode for the .NET Garbage Collector is enabled by default, but has a small performance regression when collecting Generation 0 – the .NET team tries to fix this in an upcoming release

In the end

- Startup times, general memory footprint and app size is reduced significantly
- Throughput is slightly slower, but this is negligible

Should we all switch to Native AOT now?

think tecture

ASP.NET Core Native AOT
What works, what doesn’t?

The most important thing

Native AOT does not support unbound reflection. Unfortunately, this feature is used by many frameworks, such as:

- Serializers (Newtonsoft.JSON, XmlSerializer)
- Dependency Injection Containers
- Object/Relational Mappers
- Some ASP.NET Core specific frameworks (SignalR, MVC)

Especially dynamic loading of assemblies (Assembly.Load) and generating code at runtime is not possible. The workaround is to generate the corresponding code at compile time, usually via Source Generators or similar tools.

See ASP.NET Core support for Native AOT

Logging

In Cloud Native scenarios, logs are usually written to the console in JSON format, so that collectors can pick them up

- Microsoft.Extensions.Logging works well for console and file
- Serilog works without issues for console and file, but creates trim warnings

If you require other logging providers, for example to log to a database, you should evaluate if they work properly with Native AOT.

In this talk, we focus on Serilog.

Serilog.Settings.Configuration doesn't work

- This popular package cannot be used with Native AOT
- Internally, it uses unbound reflection to instantiate Serilog-related objects and registers it with the loggerConfiguration
- The workaround here is to create your own custom log settings which can be properly deserialized

```
{
    "Serilog": {
        "Using": [ "Serilog.Sinks.Console" ],
        "MinimumLevel": {
            "Default": "Information",
            "Override": {
                "Microsoft.AspNetCore": "Warning"
            }
        },
        "WriteTo": [
            {
                "Name": "Console",
                "Args": {
                    "formatter": "Serilog.Formatting.Compact.CompactJsonFormatter, Serilog.Formatting.Compact"
                }
            }
        ]
    }
}
```

```
builder.Host.UseSerilog((context, configuration) =>
{
    configuration.ReadFrom.Configuration(context.Configuration);
});
```

Reflection works somewhat, but…

- In general, listing type members and calling them via reflection works
- Reflection in .NET can hide references between two pieces of code from static code analysis
- This is especially true when calculating type names or searching for all types that implement a certain interface: the target types might simply be trimmed during publishing
- Code cannot be emitted at run time

```
18      private static IResult GetUnboundReflectionData()
19      {
20          var typeName = $"{Namespace}.Calculator";
21          var type = Type.GetType(typeName);
22          if (type is null)
23          {
24              return Results.NotFound();
25          }
26
27          var addMethod = type.GetMethod("Add", BindingFlags.Public | BindingFlags.Static);
28          if (addMethod is null)
29          {
```

WebApp › Reflection › UnboundReflectionEndpoint › GetUnboundReflectionData

Terminal | Local

```
PS D:\Programming\dotnet-native-aot-webinar\WebApp> dotnet publish -o bin/native-aot
MSBuild version 17.9.4+90725d08d for .NET
  Determining projects to restore...
  Restored D:\Programming\dotnet-native-aot-webinar\WebApp\WebApp.csproj (in 211 ms).
  WebApp -> D:\Programming\dotnet-native-aot-webinar\WebApp\bin\Release\net8.0\win-x64\WebApp.dll
  Generating native code
D:\Programming\dotnet-native-aot-webinar\WebApp\Reflection\UnboundReflectionEndpoint.cs(21): Trim analysis warning IL2057: WebApp.Reflection.UnboundReflectionEndpoint.GetUnboundReflectionData(): Unrecognized value passed to the parameter 'typeName' of method 'System.Type.GetType(String)'. It's not possible to guarantee the availability of the target type. [D:\Programming\dotnet-native-aot-webinar\WebApp\WebApp.csproj]
```

Avoid Reflection in Native AOT apps!

AutoMapper doesn't work

- AutoMapper uses LINQ Expression Trees internally which results in exceptions at runtime
- Recommendation: use Mapperly instead
- Mapperly creates exactly the mapping code I would write using Source Generators
- Avoid placing business logic into object-to-object mappers

```
1 usage  Kenny Pflug *
19      public static async Task<IResult> GetToDos(
20          IGetToDosSession session,
21          IMapper mapper,
22          CancellationToken cancellationToken = default
23      )
        {
25          var toDoList = await session.GetToDoListAsync(cancellationToken);
26          var dtoList = mapper.Map<List<ToDoItem>, List<ToDoListDto>>(toDoList);
27          return Results.Ok(dtoList);
28      }
29  }
```

```
WebApp › GetToDos › GetToDosEndpoint › MapGetToDos

Terminal  Local

[20:00:25 INF] Content root path: D:\Programming\dotnet-native-aot-webinar\WebApp\bin\native-aot
[20:00:57 ERR] HTTP GET /todos responded 500 in 3.9148 ms
System.TypeInitializationException: A type initializer threw an exception. To determine which type, inspect the InnerException's StackTrace property.
 ---> System.ArgumentNullException: Value cannot be null. (Parameter 'method')
   at System.ArgumentNullException.Throw(String) + 0x2b
   at System.Linq.Expressions.Expression.Call(MethodInfo, Expression) + 0xbd
   at AutoMapper.Execution.ExpressionBuilder..cctor() + 0x2a7
   at System.Runtime.CompilerServices.ClassConstructorRunner.EnsureClassConstructorRun(StaticClassConstructionContext*) + 0xb9
   --- End of inner exception stack trace ---
   at System.Runtime.CompilerServices.ClassConstructorRunner.EnsureClassConstructorRun(StaticClassConstructionContext*) + 0x14a
   at System.Runtime.CompilerServices.ClassConstructorRunner.CheckStaticClassConstructionReturnGCStaticBase(StaticClassConstructionContext*, Object) + 0xd
   at AutoMapper.MapperConfiguration..ctor(MapperConfigurationExpression) + 0x568
   at Microsoft.Extensions.DependencyInjection.ServiceCollectionExtensions.<>c.<AddAutoMapperClasses>b__12_2(IServiceProvider sp) + 0x3f
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteRuntimeResolver.VisitRootCache(ServiceCallSite, RuntimeResolverContext) + 0x85
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteVisitor`2.VisitCallSite(ServiceCallSite callSite, TArgument argument) + 0x93
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteRuntimeResolver.Resolve(ServiceCallSite, ServiceProviderEngineScope) + 0x3d
```

MediatR doesn’t work

- MediatR won’t work because the constructor of the Mediator type seems to be trimmed out
- Use Mediator as an alternative, which almost has the same API and features
- Mediator uses Source Generators to create the code for the mediator object at compile time
- It supports pipes and filters to handle cross-cutting concerns

```
25  public sealed class ToDoRequest : IRequest<List<ToDoListDto>>
26  {
        1 usage
27      public static ToDoRequest Instance { get; } = new();
28  }
29
    1 usage  Kenny Pflug
30  public sealed class ToDoRequestHandler : IRequestHandler<ToDoRequest, List<ToDoListDto>>
31  {
32      private readonly IGetToDosSession _session;
33
        Kenny Pflug
34      public ToDoRequestHandler(IGetToDosSession session)
35      {
36          _session = session;
37      }
38
        Kenny Pflug
39      public async Task<List<ToDoListDto>> Handle(ToDoRequest request, CancellationToken cancellationToken)
40      {
41          var toDoList = await _session.GetToDoListAsync(cancellationToken);
42          var dtoList = toDoList.MapToDtoList();
43          return dtoList;
44      }
45  }
```

```
WebApp  GetToDos  GetToDosEndpoint
Terminal  Local
[07:44:25 ERR] HTTP GET /todos responded 500 in 3.2819 ms
System.InvalidOperationException: A suitable constructor for type 'MediatR.Mediator' could not be located. Ensure the type is concrete and services are registered for all parameters of a public constructor.
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFactory.CreateConstructorCallSite(ResultCache, ServiceIdentifier, Type, CallSiteChain) + 0x30d
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFactory.TryCreateExact(ServiceDescriptor, ServiceIdentifier, CallSiteChain, Int32) + 0x287
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFactory.TryCreateExact(ServiceIdentifier, CallSiteChain) + 0x94
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFactory.CreateCallSite(ServiceIdentifier serviceIdentifier, CallSiteChain callSiteChain) + 0x12d
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFactory.GetCallSite(ServiceIdentifier, CallSiteChain) + 0x54
   at Microsoft.Extensions.DependencyInjection.ServiceProvider.CreateServiceAccessor(ServiceIdentifier serviceIdentifier) + 0x7e
   at System.Collections.Concurrent.ConcurrentDictionary`2.GetOrAdd(TKey, Func`2) + 0xf6
   at Microsoft.Extensions.DependencyInjection.ServiceProvider.GetService(ServiceIdentifier, ServiceProviderEngineScope) + 0x44
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.ServiceProviderEngineScope.GetService(Type serviceType) + 0x42
   at Microsoft.Extensions.DependencyInjection.ServiceProviderServiceExtensions.GetRequiredService(IServiceProvider, Type) + 0x50
   at Microsoft.Extensions.DependencyInjection.ServiceProviderServiceExtensions.GetRequiredService[T](IServiceProvider provider) + 0x29
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteRuntimeResolver.VisitDisposeCache(ServiceCallSite, RuntimeResolverContext) + 0xe
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteVisitor`2.VisitCallSite(ServiceCallSite callSite, TArgument argument) + 0xb5
   at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteRuntimeResolver.Resolve(ServiceCallSite, ServiceProviderEngineScope) + 0x3d
```

MassTransit doesn’t work

- MassTransit uses unbound reflection to setup its pipes and filters
- Use native access libraries like RabbitMQ.Client to access message brokers
- Polly.Core works well for resilience
- In Cloud Native environments, use sidecars to unify logic (Dapr, Istio)

```
7   public static class EventingModule
8   {
        1 usage
9       public static IServiceCollection AddEventing(this IServiceCollection services) =>
10          services.AddMassTransit(
11              x =>
12              {
13                  x.AddConsumer<MyMessageConsumer>();
14                  x.UsingRabbitMq(
15                      (context, configurator) =>
16                      {
17                          configurator.ConfigureJsonSerializerOptions(
18                              options =>
19                              {
20                                  options.TypeInfoResolverChain.Insert(0, AppJsonSerializationContext.Default);
21                                  return options;
22                              });
23                          configurator.Host(
24                              "localhost",
25                              5672,
26                              "/",
27                              h =>
28                              {
29                                  h.Username("guest");
30                                  h.Password("guest");
31                              }
32                          );
33                          configurator.ConfigureEndpoints(context);
34                      }
35                  );
36              }
37          );
```

WebApp Eventing

Terminal Local

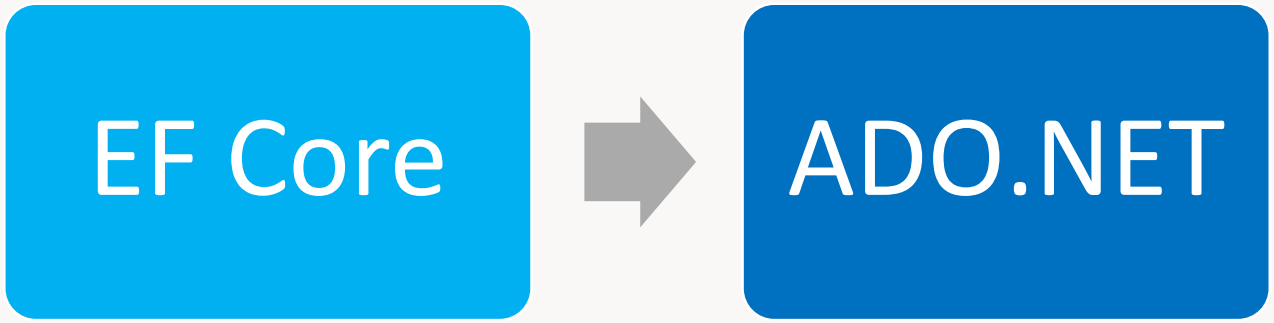
```
[22:04:40 FTL] Could not run web app
MassTransit.ConfigurationException: An exception occurred during bus creation
 ---> System.NotSupportedException: 'MassTransit.Middleware.DynamicFilter`2+KeyOutputFilter`1[MassTransit.ConsumeContext,System.Guid,MassTransit.ConsumeContext`1[WebApp.Eventing.MyMessage]]' is missing native code or metadata. This can happen for code that is not compatible with trimming or AOT. Inspect and fix trimming and AOT related warnings that were generated when the app was published. For more information see https://aka.ms/nativeaot-compatibility
   at System.Reflection.Runtime.General.TypeUnifier.WithVerifiedTypeHandle(RuntimeConstructedGenericTypeInfo, RuntimeTypeInfo[]) + 0x75
   at MassTransit.Middleware.DynamicFilter`2.CreateOutputPipe[T]() + 0x89
   at MassTransit.Middleware.DynamicFilter`1.GetPipe[T]() + 0x7a
   at MassTransit.Middleware.DynamicFilter`1.GetPipe[T,TResult]() + 0x17
   at MassTransit.Middleware.DynamicFilter`1.ConnectPipe[T](IPipe`1) + 0x1f
   at MassTransit.Middleware.ConsumePipe.ConnectConsumePipe[T](IPipe`1) + 0x5f
   at MassTransit.Configuration.ConsumerConnector`1.MassTransit.Configuration.IConsumerConnector.ConnectConsumer[TConsumer](IConsumePipeConnector, IConsumerFactory`1, IConsumerSpecification`1) + 0x1c1
   at MassTransit.Configuration.ConsumerConfigurator`1.Configure(IReceiveEndpointBuilder) + 0x65
   at MassTransit.Configuration.ReceiveEndpointConfiguration.ApplySpecifications(IReceiveEndpointBuilder) + 0x41
   at MassTransit.RabbitMqTransport.Configuration.RabbitMqReceiveEndpointConfiguration.CreateRabbitMqReceiveEndpointContext() + 0x4f
   at MassTransit.RabbitMqTransport.Configuration.RabbitMqReceiveEndpointConfiguration.Build(IHost) + 0x3c
   at MassTransit.RabbitMqTransport.Configuration.RabbitMqHostConfiguration.Build() + 0xc0
```

Testing works, but…

- No major test runner (xunit, nunit, vstest) has support for Native AOT out of the box
- When running them, you run in regular CLR mode – you can't be sure that everything works equally in Native AOT
- Solution: write integration tests that use docker with Testcontainers
- Alternative: publish the app, call into it with HttpClient (but setting up the environment will probably be harder)
- Integration tests take long, so my advice is to stick to the testing pyramid: many unit tests, some integration tests, even less E2E tests
- If you go for many integration tests: ensure that your CI/CD pipeline agents have the required resources!

EF Core doesn’t work

- EF Core does not work with Native AOT, even with pre-compiled models
- A preview version will be available for .NET 9, RTM for .NET 10
- You should resort to plain ADO.NET - I advise you to use the Humble Object Pattern
- Dapper.Aot can help you, but currently has no support for cancellation tokens



.NET Object/Relational Mappers and Native AOT

Tested:

- Dapper AOT ✔️ ⚠️
- LinqToDB ✔️ ⚠️
- Entity Framework Core ❌
- NHibernate ❌
- ServiceStack.OrmLite ❌
- PetaPoco ❌
- RepoDB ❌

Not tested:

- DevExpress XPO ❔
- LLBLGenPro ❔
- Massive ❔
- SqlMarshal ❔

think tecture

ASP.NET Core Native AOT
Conclusion

Pros and cons

- Startup time
- Memory Footprint
- App Size

- Framework/library support
  - No MVC, no Blazor, no SignalR, no EF Core
  - Many popular third-party frameworks/libraries don’t work
- Tooling
  - Unnecessary, inexplicable warnings during publish
  - No managed debugging

In which scenarios does it shine? Where not?

- Microservices: hosted in Kubernetes, run in conjunction with service mesh or sidecar
- Serverless
- Resource-constraint environment: embedded software, for example running on raspberry Pi

- Migration of existing projects
- Building anything else than Web APIs and console apps
- Early adoption

The future of Native AOT

- Microsoft will invest heavily in Native AOT in the upcoming releases of .NET - but some parts of the ecosystem (like MVC) will probably never be adapted
- Entity Framework Core will get support for Native AOT – but when?
- Make or break: will popular frameworks/libraries replace unbound reflection with generated code at compile time?

Sources

- ASP.NET Core support for Native AOT – Microsoft Learn
- Native AOT deployment – Microsoft Learn
- Comparing WebApplication.CreateBuilder to the new CreateSlimBuilder method – Andrew Lock | .NET Ecapades
- Native AOT with ASP.NET Core – Thinktecture Blog
- NativeAOT support – GitHub dotnet efcore issue
- Testcontainers – Website
- Grafana k6 documentation - Website
- Dynamically Adapting To Application Sizes – Maoni Stephens on Medium.com

think
tecture

Thanks!