#24 TECHARTIKEL

Performancevergleich: .NET 9 gRPC Service in Docker (Normal, Chiseled, AOT)

01.12.2025

Marius Roser

Inhalt

Performancevergleich: .NET 9 gRPC Service in Docker (Normal, Chiseled, AOT)

Bei der Containerisierung von .NET-Anwendungen gibt es verschiedene Möglichkeiten der Optimierung durch die Imageauswahl. In diesem Artikel untersuchen wir die Performance eines einfachen gRPC-Services, der mit verschiedenen Docker-Image-Konfigurationen bereitgestellt wird. Drei Ansätze werden verglichen: ein Standard-.NET-Image, ein Chiseled-Image und ein Image, welches nur die minimalen Abhängigkeiten und eine Ahead-of-Time (AOT)-kompilierte Anwendung enthält. Ziel ist es, die Auswirkungen dieser Konfigurationen auf die Imagegröße, die Startzeit und die Request-Performance zu analysieren.

1. Projekt-Setup

Der Benchmark basiert auf einem einfachen gRPC-Service, der in C# implementiert ist. Für den Vergleich wurden drei verschiedene Dockerfiles verwendet, um die jeweiligen Image-Konfigurationen zu erstellen:

Dies ist das Standard-Docker-Image für .NET-Anwendungen mit der vollständigen Laufzeitumgebung. Es bietet eine vollständige Umgebung, ist aber tendenziell größer.

```dockerfile
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["GrpcBenchmark.csproj", "."]
RUN dotnet restore "./GrpcBenchmark.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "./GrpcBenchmark.csproj" -c $BUILD_CONFIGURATION -o
/app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./GrpcBenchmark.csproj" -c $BUILD_CONFIGURATION -o
/app/publish /p:UseAppHost=false

FROM mcr.microsoft.com/dotnet/nightly/aspnet:9.0 AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "GrpcBenchmark.dll"]
```

Chiseled Images sind eine optimierte Variante, die nur die absolut notwendigen Komponenten der .NET Runtime enthalten. Dies führt zu deutlich kleineren Images und einer reduzierten Angriffsfläche.

```dockerfile
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["GrpcBenchmark.csproj", "."]
```

```
RUN dotnet restore "./GrpcBenchmark.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "./GrpcBenchmark.csproj" -c $BUILD_CONFIGURATION -o
/app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./GrpcBenchmark.csproj" -c $BUILD_CONFIGURATION -o
/app/publish /p:UseAppHost=false

FROM mcr.microsoft.com/dotnet/nightly/aspnet:9.0-noble-chiseled AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "GrpcBenchmark.dll"]
```

Ahead-of-Time (AOT)-Kompilierung übersetzt den .NET-Code direkt in nativen Maschinencode während des Builds. Dadurch ist eine Just-in-Time (JIT)-Kompilierung zur Laufzeit nicht notwendig und so kann ein Image mit minimalen Abhängigkeiten verwendet werden, was zu schnelleren Startzeiten und potenziell besserer Laufzeitleistung führt.

```dockerfile
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
WORKDIR /src

COPY *.csproj .
RUN dotnet restore -r linux-x64

COPY . .
RUN apt-get update && apt-get install -y build-essential
RUN dotnet publish -c Release -o /app/build -r linux-arm64 --self-contained
true /p:PublishAot=true

FROM mcr.microsoft.com/dotnet/nightly/runtime-deps:9.0-noble-chiseled-extra
WORKDIR /app
COPY --from=build /app/build .
ENTRYPOINT ["./GrpcBenchmark"]
```

Für die Performance-Messung wurden die Tools ghz (gRPC benchmarking tool) und grpcurl (für die Startzeitmessung) verwendet.

1.1. gRPC Service

Der gRPC-Service, der in diesem Benchmark verwendet wird, besteht aus einer einfachen Methode, die einen Namen entgegennimmt und eine Begrüßungsnachricht zurückgibt. Die Definition des Dienstes ist in der .proto Datei wie folgt festgelegt:

```
syntax = "proto3";

option csharp_namespace = "GrpcBenchmark";

package greet;

service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply);
}

message HelloRequest {
  string name = 1;
}

message HelloReply {
  string message = 1;
}
```

2. Benchmark-Ergebnisse

Benchmark Image	Anfragen/ Sekunde	Durchschnitt (ms)	Langsamste Anfrage (ms)	Schnellste Anfrage (ms)	Image-größe	Startzeit (s)
grpc-benchmark-normal	22777.89	8.01	115.42	0.63	256MB	1.155751
grpc-benchmark-chiseled	22503.30	8.06	113.57	0.75	125MB	1.095969
grpc-benchmark-aot	28891.98	6.17	43.67	0.85	107MB	0.338422

3. Analyse der Ergebnisse

3.1. Imagegröße

Das Standard-Image ist mit 256MB erwartungsgemäß das größte, da es die gesamte .NET ASP.NET Runtime und alle Abhängigkeiten enthält.

Das Chiseled-Image ist mit 125MB deutlich kleiner (fast die Hälfte des normalen Images). Dies liegt daran, dass es nur die minimalen Komponenten enthält, die für die Ausführung der Anwendung erforderlich sind.

Das Image mit der AOT-kompilierten Anwendung ist das kleinste. Da der Code direkt in nativen Maschinencode kompiliert wird, sind viele der .NET Runtime-Abhängigkeiten, die zur JIT-Kompilierung benötigt werden, nicht mehr erforderlich. Dies macht es ideal für Umgebungen, in denen die Imagegröße kritisch ist.

3.2. Startzeit

Die Startzeit einer Anwendung bezieht sich auf die Dauer von dem Moment, in dem die Anwendung gestartet wird, bis sie vollständig betriebsbereit ist und Anfragen

verarbeiten kann. Dies ist ein kritischer Performanceindikator, insbesondere in Cloud- und Microservice-Architekturen, aus mehreren Gründen:

In dynamischen Umgebungen, in denen Anwendungen je nach Last schnell hoch- oder herunterskaliert werden müssen (z.B. bei Serverless-Funktionen), ist eine schnelle Startzeit entscheidend, um auf Lastspitzen reagieren zu können und "Cold Starts" zu minimieren. In Cloud-Umgebungen, insbesondere bei Pay-per-Use-Modellen, können lange Startzeiten zu höheren Kosten führen, da Ressourcen länger vorgehalten werden müssen, bevor sie produktiv genutzt werden können.

Zu den gemessenen Startzeiten des Benchmarks lässt sich folgendes sagen:

- Die Startzeit des normalen Images ist hier am höchsten, da die JIT-Kompilierung zur Laufzeit erfolgen muss.

- Das Chiseled-Image zeigt eine schnellere Startzeit. Dies ist auf die reduzierte Größe und die optimierte Struktur zurückzuführen, die weniger Overhead beim Starten verursacht.

- Das Image mit der AOT-kompilierten Anwendung zeigt eine sehr schnelle Startzeit. Da der Code bereits nativ kompiliert ist, entfällt der JIT-Kompilierungsschritt vollständig, was den Startvorgang erheblich beschleunigt.

3.3. Anfragen pro Sekunde und Latenz

Ein höherer Wert an Anfragen pro Sekunde deutet hier auf einen höheren Durchsatz und eine bessere Fähigkeit zur Bewältigung einer hohen Anzahl von Anfragen hin.

Latenz (Durchschnitt, langsamste, schnellste) misst die Zeit, die eine einzelne Anfrage benötigt, um vom Client zum Server zu gelangen, dort verarbeitet zu werden und eine Antwort zurückzusenden. Eine niedrigere Latenz bedeutet eine schnellere Reaktion des Services. Die Werte für 'Slowest' und 'Fastest' geben die Extremwerte der Latenzverteilung an, während 'Average' den Durchschnitt darstellt. Eine geringe Latenz ist entscheidend für Anwendungen, die eine schnelle Interaktion erfordern.

- Das normale Image bietet mit einem Durchschnitt von 8ms eine solide Performance, aber die JIT-Kompilierung kann zu Beginn der Ausführung zu einer höheren Latenz führen.

- Das Chiseled-Image zeigt mit einem Durschnitt von 8ms keinen großen Unterschied zum normalen Image. In diesem Benchmark liegt der Vorteil hier hauptsächlich in der kleineren Imagegröße.

- Das AOT-kompilierte Image zeigt in diesem Benchmark mit einem Durchschnitt von 6ms die beste Performance, was hauptsächlich an der fehlenden JIT-Kompilierung liegt.

4. Einschränkungen

Trotz der beeindruckenden Vorteile von AOT, insbesondere bei Startzeit und Imagegröße, gibt es ein paar Einschränkungen. Native AOT-Kompilierung in .NET schränkt die Verwendung bestimmter dynamischer Features zur Laufzeit ein, wie z.B. Reflection oder dynamische Code-Generierung. Während gRPC vollständig unterstützt wird, gibt es derzeit nur partielle Unterstützung für Minimal APIs. MVC wird nicht unterstützt. Gerade hier zeigt sich jedoch die Stärke von gRPC im Kontext von AOT: Da gRPC-Services auf statisch definierten Schnittstellen (Protocol Buffers) basieren und typischerweise weniger dynamische Laufzeitfeatures benötigen, sind sie von diesen AOT-Einschränkungen kaum betroffen.

5. Fazit

Das normale Image ist gut für die Entwicklung und wenn die Imagegröße und Startzeit keine kritischen Faktoren sind. Das Chiseled Image ist eine ausgezeichnete Wahl, wenn die Imagegröße und die Startzeit optimiert werden sollen, ohne auf die Vorteile der JIT-Kompilierung zur Laufzeit zu verzichten. Ein Image mit AOT-kompilierter Anwendung ist ideal für Szenarien, in denen extrem schnelle Startzeiten und minimale Imagegrößen entscheidend sind, wie z.B. bei Serverless-Funktionen oder Coldstarts in Cloudumgebungen. Beachtet werden sollten hier allerdings die genannten Einschränkungen.