Kestrel

Kestrel je multiplatformní webový server vyvinutý společností Microsoft pro ASP.NET Core aplikace. Jedná se o lehký, vysokovýkonný server napsaný v C#, který je výchozím webovým serverem pro ASP.NET Core.

Základní charakteristika

Kestrel je postaven na asynchronním I/O a poskytuje moderní přístup k hostování webových aplikací:

- **Cross-platform** – běží na Windows, Linux a macOS
- **Vysoký výkon** – jeden z nejrychlejších webových serverů
- **Lightweight** – minimální paměťová náročnost
- **Asynchronní** – založen na async/await vzoru
- **Open-source** – vyvíjen na GitHubu
- **Self-contained** – zabudován přímo v aplikaci

Historie

Kestrel byl vyvinut jako součást revoluce .NET Core:

- **2016** – první verze s ASP.NET Core 1.0
- **2017** – ASP.NET Core 2.0, výrazné zlepšení výkonu
- **2018** – ASP.NET Core 2.1, HTTP/2 podpora
- **2019** – ASP.NET Core 3.0, gRPC podpora
- **2020** – .NET 5, Kestrel jako součást unified .NET
- **2021+** – kontinuální vylepšování výkonu a funkcí

Architektura

Kestrel využívá moderní technologie:

Libuv vs Socket Transport

Kestrel podporuje dva transportní vrstvy:

- **Libuv** – asynchronní I/O knihovna (původně z Node.js)
- **Socket Transport** – nativní .NET Sockets API (výchozí od .NET Core 2.1)

Socket Transport je rychlejší a lépe integrovaný s .NET ekosystémem.

HTTP Pipeline

Zpracování požadavku v Kestrelu:

1. **Transport layer** přijme TCP/IP spojení
2. **Connection middleware** zpracuje spojení
3. **Protocol layer** parsuje HTTP protokol
4. **Application middleware** zpracuje požadavek
5. Odpověď je vrácena klientovi

Použití Kestrel

Standalone server

Kestrel může běžet samostatně jako webový server:

```
// Program.cs
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello from Kestrel!");

// Kestrel naslouchá na portu 5000 (HTTP) a 5001 (HTTPS)
app.Run();
```

Spuštění:

```
dotnet run
```

Aplikace je dostupná na:

- `http://localhost:5000`
- `https://localhost:5001`

S reverzní proxy

V produkčním prostředí se Kestrel obvykle používá za reverzní proxy:

```
Internet → Reverzní Proxy (Nginx/IIS/Apache) → Kestrel → ASP.NET Core App
```

Důvody pro reverzní proxy:

- **SSL/TLS terminace** – offloading šifrování
- **Load balancing** – distribuce zátěže mezi servery
- **Caching** – cache statického obsahu
- **Compression** – komprese odpovědí
- **Security** – firewall, rate limiting

- **Static files** – efektivnější obsluha statických souborů

Konfigurace Kestrel

Základní konfigurace

```
var builder = WebApplication.CreateBuilder(args);

// Konfigurace Kestrel
builder.WebHost.ConfigureKestrel(serverOptions =>
{
    // Nastavení limitů
    serverOptions.Limits.MaxConcurrentConnections = 100;
    serverOptions.Limits.MaxRequestBodySize = 10 * 1024 * 1024; // 10 MB
    serverOptions.Limits.KeepAliveTimeout = TimeSpan.FromMinutes(2);
    serverOptions.Limits.RequestHeadersTimeout = TimeSpan.FromSeconds(30);

    // HTTP/2 limity
    serverOptions.Limits.Http2.MaxStreamsPerConnection = 100;
});

var app = builder.Build();
app.Run();
```

Endpoints konfigurace

```
builder.WebHost.ConfigureKestrel(options =>
{
    // HTTP na portu 5000
    options.ListenAnyIP(5000);

    // HTTPS na portu 5001
    options.ListenAnyIP(5001, listenOptions =>
    {
        listenOptions.UseHttps("certificate.pfx", "password");
    });

    // Localhost pouze
    options.ListenLocalhost(5002);

    // Konkrétní IP adresa
    options.Listen(IPAddress.Parse("192.168.1.100"), 5003);

    // Unix socket (Linux/macOS)
    options.ListenUnixSocket("/tmp/kestrel.sock");
});
```

appsettings.json konfigurace

```
{
  "Kestrel": {
    "Endpoints": {
      "Http": {
        "Url": "http://localhost:5000"
      },
      "Https": {
        "Url": "https://localhost:5001",
        "Certificate": {
          "Path": "certificate.pfx",
          "Password": "password"
        }
      }
    },
    "Limits": {
      "MaxConcurrentConnections": 100,
      "MaxRequestBodySize": 10485760,
      "KeepAliveTimeout": "00:02:00"
    }
  }
}
```

SSL/TLS certifikáty

Kestrel podporuje několik způsobů konfigurace SSL:

PFX soubor

```
options.ListenAnyIP(5001, listenOptions =>
{
    listenOptions.UseHttps("certificate.pfx", "password");
});
```

Development certifikát

```
# Vytvoření development certifikátu
dotnet dev-certs https --trust
```

```
// Automaticky použije development certifikát
options.ListenAnyIP(5001, listenOptions =>
{
    listenOptions.UseHttps();
});
```

Let's Encrypt

Pro produkční použití s automatickými certifikáty:

```
// Použití LettuceEncrypt package
services.AddLettuceEncrypt()
    .PersistDataToDirectory(new DirectoryInfo("LettuceEncrypt"),
"password");
```

SNI (Server Name Indication)

Více certifikátů na jednom portu:

```
options.ListenAnyIP(443, listenOptions =>
{
    listenOptions.UseHttps(httpsOptions =>
    {
        httpsOptions.ServerCertificateSelector = (context, domain) =>
        {
            if (domain == "example.com")
                return LoadCertificate("example.pfx");
            if (domain == "another.com")
                return LoadCertificate("another.pfx");
            return null;
        };
    });
});
```

HTTP/2 podpora

Kestrel plně podporuje HTTP/2:

```
builder.WebHost.ConfigureKestrel(options =>
{
    options.ConfigureEndpointDefaults(listenOptions =>
    {
        // HTTP/1.1 a HTTP/2
        listenOptions.Protocols = HttpProtocols.Http1AndHttp2;

        // Pouze HTTP/2
        // listenOptions.Protocols = HttpProtocols.Http2;
    });
});
```

HTTP/2 vlastnosti:

- **Multiplexing** – více požadavků přes jedno spojení

- **Server Push** – proaktivní posílání zdrojů
- **Header Compression** – HPACK algoritmus
- **Stream Prioritization** – prioritizace požadavků

HTTP/3 podpora

Kestrel podporuje HTTP/3 (QUIC protokol) od .NET 6:

```
builder.WebHost.ConfigureKestrel(options =>
{
    options.ListenAnyIP(5001, listenOptions =>
    {
        listenOptions.Protocols = HttpProtocols.Http1AndHttp2AndHttp3;
        listenOptions.UseHttps();
    });
});
```

HTTP/3 výhody:

- **QUIC transport** – postavený na UDP místo TCP
- **Nižší latence** – rychlejší navázání spojení
- **Connection migration** – odolnost proti změnám sítě
- **Built-in encryption** – vždy šifrované

Poznámka: HTTP/3 je experimentální a vyžaduje podporu na straně klienta.

gRPC podpora

Kestrel je výchozí server pro gRPC v .NET:

```
var builder = WebApplication.CreateBuilder(args);

// Přidání gRPC služeb
builder.Services.AddGrpc();

builder.WebHost.ConfigureKestrel(options =>
{
    // HTTP/2 je vyžadován pro gRPC
    options.ConfigureEndpointDefaults(listenOptions =>
    {
        listenOptions.Protocols = HttpProtocols.Http2;
    });
});

var app = builder.Build();

// Mapování gRPC služby
app.MapGrpcService<GreeterService>();
```

```
app.Run();
```

WebSockets

Plná podpora pro WebSocket protokol:

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.UseWebSockets();

app.Use(async (context, next) =>
{
    if (context.WebSockets.IsWebSocketRequest)
    {
        using var webSocket = await
context.WebSockets.AcceptWebSocketAsync();
        await HandleWebSocket(webSocket);
    }
    else
    {
        await next();
    }
});

app.Run();
```

Výkon a optimalizace

Kestrel je navržen pro vysoký výkon:

Benchmarky

V TechEmpower benchmarcích patří Kestrel mezi nejrychlejší:

- **Plaintext** – více než 7 milionů req/sec
- **JSON serialization** – přes 500 000 req/sec
- Top 10 ve většině kategorií

Optimalizace tipy

```
builder.WebHost.ConfigureKestrel(options =>
{
    // Response buffering
```

```
    options.AddServerHeader = false; // Odstranění Server header

    // Connection pooling
    options.Limits.MaxConcurrentConnections = 100;
    options.Limits.MaxConcurrentUpgradedConnections = 100;

    // Request size limits
    options.Limits.MaxRequestBodySize = 10 * 1024 * 1024;
    options.Limits.MaxRequestLineSize = 8 * 1024;
    options.Limits.MaxRequestHeadersTotalSize = 32 * 1024;

    // Timeouts
    options.Limits.KeepAliveTimeout = TimeSpan.FromMinutes(2);
    options.Limits.RequestHeadersTimeout = TimeSpan.FromSeconds(30);

    // HTTP/2
    options.Limits.Http2.MaxStreamsPerConnection = 100;
    options.Limits.Http2.HeaderTableSize = 4096;
    options.Limits.Http2.MaxFrameSize = 16384;
    options.Limits.Http2.InitialConnectionWindowSize = 128 * 1024;
});
```

Response Compression

```
builder.Services.AddResponseCompression(options =>
{
    options.EnableForHttps = true;
    options.Providers.Add<GzipCompressionProvider>();
    options.Providers.Add<BrotliCompressionProvider>();
});

var app = builder.Build();
app.UseResponseCompression();
```

Limity a kvóty

Konfigurace limitů pro zabezpečení a stabilitu:

```
builder.WebHost.ConfigureKestrel(options =>
{
    var limits = options.Limits;

    // Connection limits
    limits.MaxConcurrentConnections = 100;
    limits.MaxConcurrentUpgradedConnections = 100;

    // Request limits
    limits.MaxRequestBodySize = 10 * 1024 * 1024; // 10 MB
```

```
    limits.MaxRequestBufferSize = 1024 * 1024; // 1 MB
    limits.MaxRequestLineSize = 8 * 1024; // 8 KB
    limits.MaxRequestHeadersTotalSize = 32 * 1024; // 32 KB
    limits.MaxRequestHeaderCount = 100;

    // Timeouts
    limits.KeepAliveTimeout = TimeSpan.FromMinutes(2);
    limits.RequestHeadersTimeout = TimeSpan.FromSeconds(30);

    // Response limits
    limits.MaxResponseBufferSize = 64 * 1024; // 64 KB
    limits.MinRequestBodyDataRate = new MinDataRate(
        bytesPerSecond: 100,
        gracePeriod: TimeSpan.FromSeconds(10)
    );
    limits.MinResponseDataRate = new MinDataRate(
        bytesPerSecond: 100,
        gracePeriod: TimeSpan.FromSeconds(10)
    );
});
```

Integrace s reverzní proxy

Nginx

```
server {
    listen 80;
    server_name example.com;

    location / {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection keep-alive;
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Konfigurace Kestrel s Nginx:

```
builder.Services.Configure<ForwardedHeadersOptions>(options =>
{
    options.ForwardedHeaders = ForwardedHeaders.XForwardedFor
                             | ForwardedHeaders.XForwardedProto;
    options.KnownNetworks.Clear();
```

```
    options.KnownProxies.Clear();
});

var app = builder.Build();
app.UseForwardedHeaders();
```

IIS

Kestrel s IIS pomocí ASP.NET Core Module:

```
<!-- web.config -->
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <location path="." inheritInChildApplications="false">
    <system.webServer>
      <handlers>
        <add name="aspNetCore" path="*" verb="*"
             modules="AspNetCoreModuleV2" resourceType="Unspecified" />
      </handlers>
      <aspNetCore processPath="dotnet"
                  arguments=".\MyApp.dll"
                  stdoutLogEnabled="false"
                  stdoutLogFile=".\logs\stdout"
                  hostingModel="inprocess" />
    </system.webServer>
  </location>
</configuration>
```

Hosting modely v IIS:

- **InProcess** – aplikace běží v IIS worker procesu (w3wp.exe)
- **OutOfProcess** – IIS forwarduje na Kestrel

Apache

```
<VirtualHost *:80>
    ServerName example.com

    ProxyPreserveHost On
    ProxyPass / http://127.0.0.1:5000/
    ProxyPassReverse / http://127.0.0.1:5000/

    RequestHeader set X-Forwarded-Proto "http"
</VirtualHost>
```

Unix sockets

Na Linuxu a macOS lze použít Unix sockets místo TCP:

```
builder.WebHost.ConfigureKestrel(options =>
{
    options.ListenUnixSocket("/tmp/kestrel.sock");
});
```

Nginx konfigurace:

```
server {
    listen 80;
    location / {
        proxy_pass http://unix:/tmp/kestrel.sock;
    }
}
```

Výhody Unix sockets:

- **Vyšší výkon** – nižší overhead než TCP
- **Bezpečnost** – file system permissions
- **Jednoduchost** – žádné port konflikty

Logging a diagnostika

Základní logging

```
var builder = WebApplication.CreateBuilder(args);

// Konfigurace loggingu
builder.Logging.AddConsole();
builder.Logging.AddDebug();
builder.Logging.SetMinimumLevel(LogLevel.Information);

// Kestrel logging
builder.WebHost.ConfigureKestrel((context, options) =>
{
    options.ConfigureEndpointDefaults(listenOptions =>
    {
        // Logování spojení
        listenOptions.UseConnectionLogging();
    });
});
```

Request logging

```
var app = builder.Build();

// HTTP logging middleware
app.UseHttpLogging();

// Vlastní logging middleware
app.Use(async (context, next) =>
{
    var logger = context.RequestServices
        .GetRequiredService<ILogger<Program>>();

    logger.LogInformation(
        "Request: {Method} {Path} from {IP}",
        context.Request.Method,
        context.Request.Path,
        context.Connection.RemoteIpAddress
    );

    await next();

    logger.LogInformation(
        "Response: {StatusCode}",
        context.Response.StatusCode
    );
});
```

Diagnostika výkonu

```
// EventSource monitoring
dotnet-trace collect --process-id <PID>

// Metrics
builder.Services.AddSingleton<IHostedService, MetricsService>();
```

Deployment

Standalone executable

```
# Publikace jako self-contained
dotnet publish -c Release -r linux-x64 --self-contained

# Spuštění
./MyApp
```

Systemd služba (Linux)

```
# /etc/systemd/system/myapp.service
[Unit]
Description=My ASP.NET Core App
After=network.target

[Service]
Type=notify
WorkingDirectory=/var/www/myapp
ExecStart=/var/www/myapp/MyApp
Restart=always
RestartSec=10
KillSignal=SIGINT
SyslogIdentifier=myapp
User=www-data
Environment=ASPNETCORE_ENVIRONMENT=Production
Environment=DOTNET_PRINT_TELEMETRY_MESSAGE=false

[Install]
WantedBy=multi-user.target
```

Spuštění služby:

```
sudo systemctl enable myapp
sudo systemctl start myapp
sudo systemctl status myapp
```

Docker

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ["MyApp.csproj", "./"]
RUN dotnet restore "MyApp.csproj"
COPY . .
RUN dotnet build "MyApp.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "MyApp.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
```

```
ENTRYPOINT ["dotnet", "MyApp.dll"]
```

Spuštění kontejneru:

```
docker build -t myapp .
docker run -d -p 8080:80 --name myapp myapp
```

Bezpečnost

Bezpečnostní doporučení pro Kestrel:

- **HTTPS vždy** – použít SSL/TLS v produkci
- **Za proxy** – nikdy nevystavovat Kestrel přímo na internet
- **Request limits** – nastavit rozumné limity
- **CORS** – správně nakonfigurovat Cross-Origin požadavky
- **Rate limiting** – omezit počet požadavků
- **Security headers** – přidat bezpečnostní HTTP hlavičky
- **Input validation** – validovat všechny vstupy
- **Secrets** – neukládat hesla v kódu

```
// Security headers
app.Use(async (context, next) =>
{
    context.Response.Headers.Add("X-Content-Type-Options", "nosniff");
    context.Response.Headers.Add("X-Frame-Options", "DENY");
    context.Response.Headers.Add("X-XSS-Protection", "1; mode=block");
    context.Response.Headers.Add(
        "Content-Security-Policy",
        "default-src 'self'"
    );
    context.Response.Headers.Add(
        "Strict-Transport-Security",
        "max-age=31536000; includeSubDomains"
    );

    await next();
});

// Rate limiting
builder.Services.AddRateLimiter(options =>
{
    options.GlobalLimiter = PartitionedRateLimiter.Create<HttpContext,
string>(
        context => RateLimitPartition.GetFixedWindowLimiter(
            partitionKey: context.Connection.RemoteIpAddress?.ToString() ??
"unknown",
            factory: _ => new FixedWindowRateLimiterOptions
            {
                PermitLimit = 100,
```

```
                Window = TimeSpan.FromMinutes(1)
            }
        )
    );
});

var app = builder.Build();
app.UseRateLimiter();
```

Monitoring a metriky

```
// Prometheus metrics
builder.Services.AddSingleton<IHostedService, MetricsService>();

// Health checks
builder.Services.AddHealthChecks()
    .AddCheck("Kestrel", () => HealthCheckResult.Healthy());

var app = builder.Build();

app.MapHealthChecks("/health");
app.MapHealthChecks("/health/ready");
app.MapHealthChecks("/health/live");
```

Porovnání s jinými webovými servery

Vlastnost	Kestrel	IIS	Nginx	Node.js
Platforma	Multi-platform	Windows	Multi-platform	Multi-platform
Jazyk	C#	C/C++	C	JavaScript
Async I/O	Ano	Ano	Ano	Ano
HTTP/2	Ano	Ano	Ano	Ano
HTTP/3	Ano (exp.)	Ano	Ano (exp.)	Ne
Výkon	Vynikající	Velmi dobrý	Vynikající	Dobrý
Standalone	Ano	Ne	Ano	Ano
Load balancing	Ne (proxy)	Ano (ARR)	Ano	Ne (cluster)
Použití	ASP.NET Core	Windows apps	Reverse proxy	Node.js apps

Výhody a nevýhody

Výhody

- **Multiplatformní** – Windows, Linux, macOS
- **Vysoký výkon** – top tier v benchmarcích
- **Moderní protokoly** – HTTP/2, HTTP/3, WebSockets, gRPC
- **Lehký** – minimální paměťová náročnost

- **Jednoduchost** – snadná konfigurace a použití
- **Async** – plně asynchronní zpracování
- **Open-source** – transparentní vývoj
- **Integrace** – perfektní integrace s ASP.NET Core

Nevýhody

- **Vyžaduje proxy** – pro produkci doporučena reverzní proxy
- **Omezené funkce** – méně funkcí než plnohodnotné servery (IIS, Apache)
- **Závislost na .NET** – vyžaduje .NET runtime
- **Mladší** – méně historie než Apache nebo IIS

Kdy použít Kestrel

Ideální pro:

- ASP.NET Core aplikace (jakékoliv)
- Mikroservisy a API
- Cloud-native aplikace
- Kontejnerizované aplikace
- gRPC služby
- Real-time aplikace (SignalR, WebSockets)

Méně vhodné pro:

- Statické webové stránky (lepší Nginx)
- Pokud již máte komplexní IIS infrastrukturu
- Non-.NET aplikace

Související pojmy

- ASP.NET – webový framework
- .NET Core – runtime platforma
- IIS – Windows webový server
- Nginx – reverzní proxy
- Docker – kontejnerizace
- gRPC – RPC framework
- SignalR – real-time komunikace
- HTTP/2 – moderní HTTP protokol

Externí odkazy

- Kestrel na GitHubu
- Microsoft dokumentace
- TechEmpower Benchmarks
- .NET Blog