JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 8 (2018) 258-262
Wysłane: 2018-05-29
Przyjęte: 2018-06-19

Porównanie modeli hostowania aplikacji na platformie ASP.NET Core

Kamil Zdanikowski*, Beata Pańczyk
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawione zostało porównanie modeli hostowania aplikacji na platformie ASP.NET Core. Dostępne modele hostowania zostały opisane i porównane, a następnie przeprowadzone zostały ich testy wydajnościowe. Dla każdego modelu zastosowano scenariusze testowe realizujące te same funkcje, a ich wydajność została określona za pomocą liczby przetwarzanych żądań na sekundę. Uzyskane rezultaty wskazują, że standardowa konfiguracja jest najmniej wydajna, a zastosowanie innej, np. IIS z serwerem Kestrel, czy same serwery Kestrel lub HTTP.sys mogą zapewnić kilkukrotny wzrost wydajności w porównaniu z modelem standardowym.

Słowa kluczowe: asp.net core; .net; model hostowania; iis; kestrel

*Autor do korespondencji.

Adres e-mail: kamil.zdanikowski@pollub.edu.pl

Hosting models comparison of ASP.NET Core application

Kamil Zdanikowski*, Beata Pańczyk
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents hosting models comparison of ASP.NET Core application. Available hosting models were described and compared and then performance comparison was carried out. For each model the same test scenarios were executed and their performance was determined by number of requests per second which host was able to process. The results obtained show that standard model is the least efficient one and using one of the other configurations, for example, IIS with Kestrel (in-process), Kestrel or HTTP.sys might provide even several times better performance compared to standard model.

Keywords: asp.net core; .net; hosting model; iis; kestrel

*Corresponding author.

E-mail address: kamil.zdanikowski@pollub.edu.pl

1. Wstęp

ASP.NET Core to nowa platforma stworzona przez Microsoft, służąca do tworzenia aplikacji internetowych działających po stronie serwera. Pomimo tego, że nazwiązuje ona nazwą do poprzednika (ASP.NET 4.6 MVC) jest to jednak platforma napisana w całości od nowa.

Jej powstanie związane było bezpośrednio z powstaniem platformy .NET Core, czyli odpowiednika .NET Framework, który może działać nie tylko na systemie Windows, ale również na systemach Linux i macOS. Z tego powodu postanowiono napisać nową platformę ASP.NET Core – która również może działać na wielu systemach operacyjnych – i porzucić stare nazewnictwo (zgodnie z którym nowa platforma nazywałaby się ASP.NET 5) [1].

Z takimi zmianami musiały wiązać się również zmiany po stronie hostowania aplikacji. Aplikacje napisane przy użyciu poprzedniej wersji platformy hostowane mogły być tylko na serwerze IIS (Internet Information Services), który stworzony został przez Microsoft i uruchomiony mógł być jedynie na systemie operacyjnym Windows. W przypadku platformy ASP.NET Core postanowiono stworzyć nowy sposób hostowania, w którym aplikacja uruchamiana jest ze swoją wewnętrzną implementacją serwera HTTP (np. *Kestrel*). Dzięki temu możliwe jest uzyskanie jednakowego zachowania na każdej platformie (Windows, Linux, macOS) [2].

Z uwagi na prostotę oraz niedojrzałość niektórych serwerów wewnętrznych dostępnych w ASP.NET Core nie zawsze zalecane jest wystawianie ich bezpośrednio do sieci publicznej. W takich przypadkach stosuje się model, gdzie przed aplikacją z wewnętrznym serwerem znajduje się tzw. serwer reverse proxy (np. IIS), którego celem jest wstępne przetworzenie zapytania oraz przekazanie go dalej do serwera wewnętrznego.

W niniejszym artykule przedstawione zostaną najpopularniejsze modele hostowania aplikacji na platformie ASP.NET Core oraz przeprowadzone zostaną ich testy wydajnościowe. Dla każdego modelu uruchomiony zostanie ten sam zestaw testów, a wynikiem będzie liczba zapytań na sekundę, które możliwe były do przetworzenia przez aplikację w danym modelu hostowania.

W sieci można znaleźć testy wydajnościowe platformy ASP.NET Core, ale nie pokrywają one wszystkich dostępnych modeli hostingowych. Wyniki przedstawione w jednym z artykułów [3] dostępnych w internecie pokazują, że wykorzystanie serwera Kestrel zamiast serwera IIS może zapewnić nawet kilkukrotny wzrost wydajności.

2. Platforma oraz scenariusze testowe

Do przeprowadzenia testów wydajności wykorzystana została przygotowana do tego celu autorska aplikacja, która uruchamiana była w różnych modelach hostowania. Za każdym razem wykorzystywane były te same scenariusze testowe.

2.1. Konfiguracja środowiska i wykorzystane narzędzia

Konfiguracja sprzętowa środowiska testowego oraz wykorzystane narzędzia i biblioteki wraz z ich wersjami znajdują się w tabelach 1 i 2.

Tabela 1. Konfiguracja sprzętowa środowiska testowego

System operacyjny	Windows 10 x64
CPU	Intel Core i3-3110M
Pamięć RAM	12 GB DDR3

Tabela 2. Wykorzystane biblioteki i środowiska uruchomieniowe

Narzędzie	**Wersja**
.NET Core Runtime	2.1.300
ASP.NET Core	2.1.0-preview1
IIS	10.0.17134.1
Kestrel	2.1.0-preview1
k6	0.20.0

Do przetestowania wydajności użyte zostało narzędzie k6 wyprodukowane przez firmę Load Impact. Jest to darmowe oraz proste w obsłudze narzędzie umożliwiające generowanie ruchu http [4]. Umożliwia ono konfigurację kilku parametrów – wykorzystywane w testach parametry to:

- *vus* – wirtualni użytkownicy (w praktyce ilość pętli wysyłających zapytania)
- *duration* – czas trwania testu

Konfiguracja narzędzia oraz listing wykorzystywanego skryptu, który wysyła zapytania pod określony adres i sprawdza czy kod odpowiedzi jest równy 200 znajdują się odpowiednio w tabeli 3 i na listingu 1.

Tabela 3. Konfiguracja narzędzia k6

Parametr	**Wartość**
vus	8
duration	15s

Wartość parametru *vus* dobrano w taki sposób, aby wysycenie procesora podczas testów utrzymywało się na poziomie 100%. Czas testu ustalono na 15s, aby uniknąć wpływu krótkotrwałych niestabilności na wpływ całego testu. Przy czasie ustawionym na 15s testy pokazały, że uzyskiwane wartości dotyczące ilości zapytań na sekundę są stabilne i nie zmieniają się znacząco pomiędzy kolejnymi wywołaniami testu.

Przykład 1. Skrypt narzędzia k6 wykorzystywany podczas testów

```
import { check } from "k6";
import http from 'k6/http';
export default function() {
  const response = http.get(ADRES_APLIKACJI);
  check(response, {
    "is status 200": (r) => r.status === 200
  });                 };
```

2.2. Scenariusze testowe

W celu przetestowania wydajności przygotowane zostało kilka scenariuszy testowych. Obejmują one standardowe czynności wykonywane przez serwery aplikacji internetowych, czyli:

- generowanie dynamicznego widoku HTML,
- zwracanie danych w postaci czystego tekstu,
- zwracanie danych w formacie JSON,
- zwracanie danych w formacie XML.

Generowany widok HTML składa się ze 100 przycisków oraz 10 list wyboru (każda z 10 opcjami). Scenariusz testujący wydajność zwracania danych w postaci czystego tekstu zwraca wszystkie znaki alfabetu łacińskiego i cyfry. Przykładowe modele zwracane jako JSON i XML przedstawione zostały na listingach 2 i 3.

Przykład 2. Przykładowa struktura danych zwracanych jako JSON

```
{   "id": "5822a71f-d28e-449c-a220-d282239f9d09",
    "stringValue": "5822a71f",
    "intValue": 1580445059,
    "listOfInts": [
          1790594724,
          428614593,
          978875039,
          437814156,
          556857108,
          1579485844,
          904515399,
          1189173955,
          1344600976,
          159188863
    ]
}
```

Przykład 3. Przykładowa struktura danych zwracana jako XML

```
<SimpleClass
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <Id>7a9dcdfe-7c88-4a79-a8a9-53ea7e1437d6</Id>
    <StringValue>7a9dcdfe</StringValue>
    <IntValue>1933853648</IntValue>
    <ListOfInts>
          <int>546007864</int>
          <int>19000976</int>
          <int>1405144961</int>
          <int>921898622</int>
          <int>1267375503</int>
          <int>1710179990</int>
          <int>1652055825</int>
          <int>824934734</int>
          <int>1114826130</int>
          <int>136438785</int>
    </ListOfInts>
</SimpleClass>
```

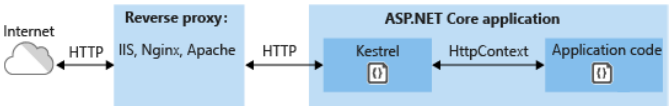
3. Modele hostowania aplikacji ASP.NET Core

Istnieje kilka możliwości hostowania aplikacji napisanych przy użyciu platformy ASP.NET Core. W kolejnych podrozdziałach przedstawione zostaną wykorzystane modele.

3.1. Kestrel + IIS (reverse proxy, out-of-process)

Ten model to standardowa i najpopularniejsza forma hostowania aplikacji ASP.NET Core. Kestrel wykorzystywany

jest jako wewnętrzny serwer HTTP, a dodatkowo przed nim wystawiony jest serwer IIS, który wstępnie przetwarza zapytanie i przekazuje je do Kestrela. Schemat konfiguracji przedstawiony został na rysunku 1 [5, 6].



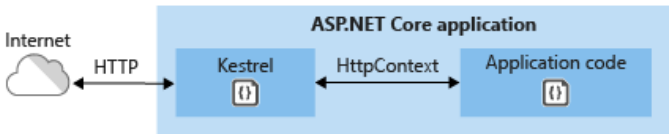
Rys. 1. Konfiguracja Kestrel + IIS (out-of-process) [6]

3.2. Kestrel + IIS (reverse proxy, in-process)

Ten model hostowania dostępny jest od wersji ASP.NET Core 2.1.0-preview1. W odróżnieniu od standardowej metody aplikacja z Kestrelem nie jest uruchamiana jako oddzielny proces, ale wewnątrz procesu IIS. Dzięki temu pominięty został narzut związany z przesłaniem zapytania HTTP z IIS do Kestrela i na odwrót pomiędzy procesami, co powinno pozytywnie wpłynąć na wydajność [7].

3.3. Kestrel

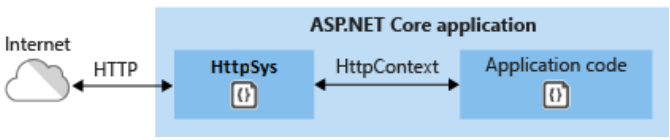
Model dostępny od początku istnienia platformy ASP.NET Core jednak przez długi czas niepolecany do stosowania w sieci zewnętrznej z uwagi na niedojrzałość serwera Kestrel (rozwijany jest dopiero od około 2 lat). Polega na hostowaniu aplikacji bez użycia pośredniczącego serwera reverse proxy. Umożliwia hostowanie aplikacji na dowolnym systemie operacyjnym. Schemat konfiguracji przedstawiony został na rysunku 2 [2].



Rys. 2. Konfiguracja Kestrel [2]

3.4. HTTP.sys

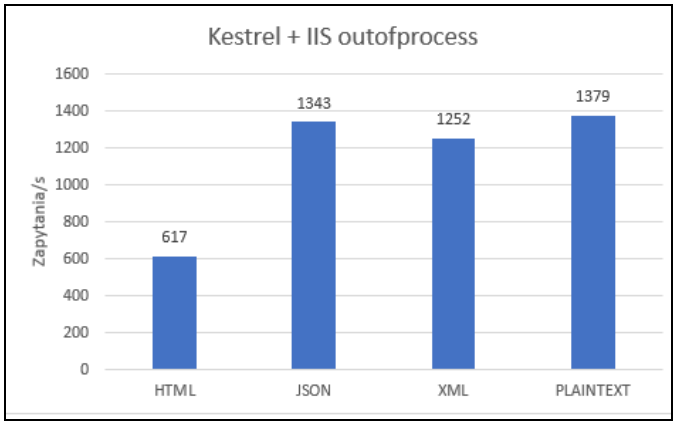
Podobnie jak Kestrel dostępny od początku istnienia ASP.NET Core (wcześniej używał nazwy WebListener). Możliwy do użycia tylko na systemie Windows z uwagi na wykorzystywanie jego wewnętrznych modułów obsługi HTTP. Użycie serwera HTTP.sys wyklucza użycie IIS jako reverse proxy, ponieważ nie są one ze sobą kompatybilne. Podobnie jak w przypadku IIS nie ma przeciwwskazań, aby wystawiać aplikację z tym serwerem do sieci zewnętrznej (HTTP.sys używany jest wewnętrznie przez IIS i rozwijany przez ponad 15 lat). Schemat konfiguracji przedstawiony został na rysunku 3 [2].



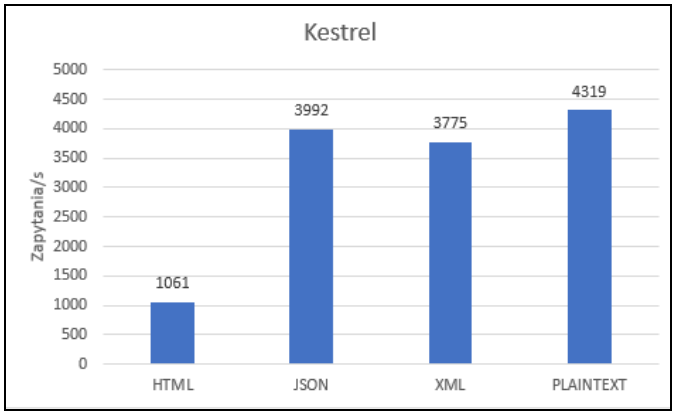
Rys. 3. Konfiguracja HTTP.sys [2]

4. Zestawienie wyników

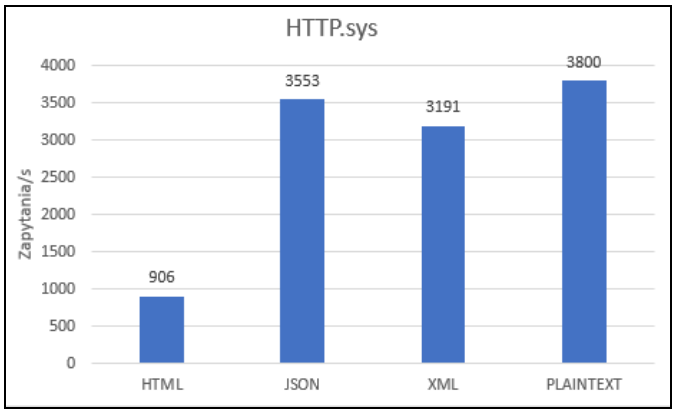
Wizualizacje wyników dla poszczególnych modeli hostowania i scenariuszy testowych przedstawione zostały w postaci wykresów na rysunkach 4, 5, 6 i 7.



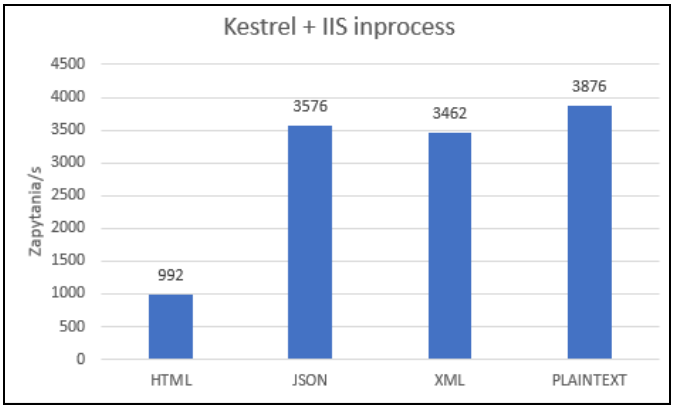
Rys. 4. Wykres dla modelu Kestrel + IIS (outofprocess)



Rys. 5. Wykres dla modelu Kestrel

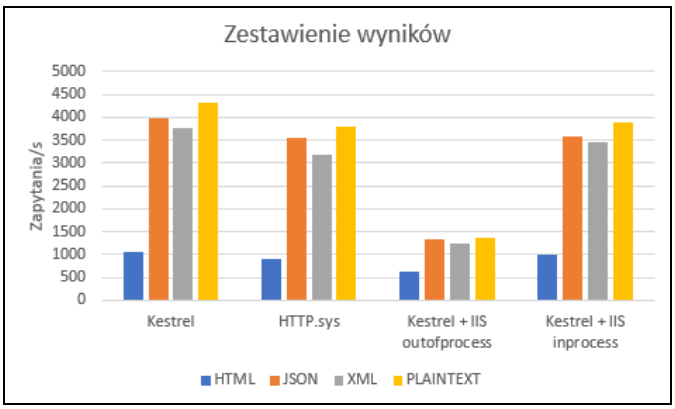


Rys. 6. Wykres dla modelu HTTP.sys



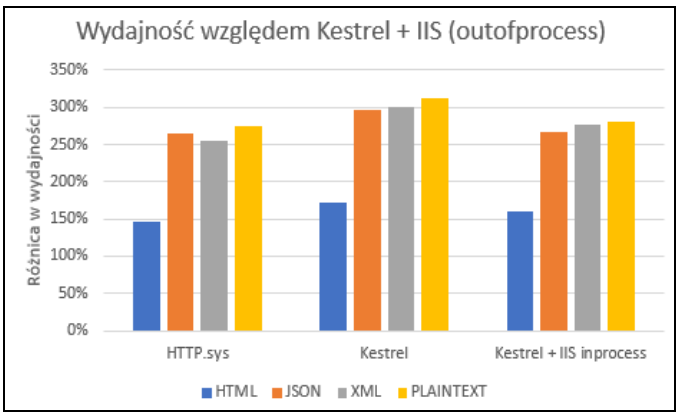
Rys. 7. Wykres dla modelu Kestrel + IIS (inprocess)

Na rysunku 8 znajduje się wykres ze zbiorczym zestawieniem wyników.



Rys. 8. Zbiorcze zestawienie wyników

Wydajność w porównaniu do standardowego modelu Kestrel + IIS (outofprocess) przedstawiona została na rysunku 9, gdzie 100% to wydajność modelu Kestrel + IIS (outofprocess).



Rys. 9. Wydajność względem modelu Kestrel + IIS (outofprocess)

5. Omówienie wyników

Standardowy oraz polecany model hostowania aplikacji to połączenie serwera wewnętrznego Kestrel z serwerem pośredniczącym (reverse proxy) IIS. Z testów wynika, że jest to najmniej wydajny model. Spowodowane jest to faktem, że zapytanie HTTP musi zostać przesłane dwukrotnie pomiędzy oddzielnymi procesami IIS oraz wewnętrznego serwera Kestrel. Pierwszy raz w momencie, gdy klient wysyła zapytanie do serwera, wtedy proces IIS wstępnie przetwarza zapytanie zgodnie z konfiguracją serwera IIS, a następnie przekazuje je do Kestrela. Drugi raz w momencie, gdy zapytanie zostanie przetworzone przez aplikację i odpowiedź jest gotowa do wysłania do klienta, wtedy to Kestrel przekazuje odpowiedź do IIS, a ten z kolei do klienta.

Kolejną mozliwością jest hostowanie aplikacji wyłącznie za pomocą serwera wewnętrznego Kestrel. Przez długi czas model ten polecany był do użycia tylko w przypadku aplikacji znajdujących się w sieci wewnętrznej ze względu na niedojrzałość serwera i możliwe braki w bezpieczeństwie oraz możliwościach konfiguracyjnych. Od wersji 2.0.0 serwera (czyli również od wersji 2.0.0 platformy ASP.NET Core) nadaje się on do wystawienia w sieci publicznej, jednak wciąż w porównaniu z IIS ma znikome możliwości konfiguracyjne. Dużą zaletą tego modelu jest możliwość hostowania aplikacji na dowolnym systemie operacyjnym. Z przeprowadzonych testów wynika, że jest to najszybsza ze wszystkich użytych konfiguracji. Na podstawie rysunku 9 można stwierdzić, że w porównaniu do standardowego modelu jest on wydajniejszy o około 70% w przypadku zapytań zwracających widok HTML i nawet 200% dla danych w postaci JSON, XML i zwykłego tekstu.

Następna konfiguracja to użycie wyłącznie serwera wewnętrznego HTTP.sys. Wadą takiego rozwiązania jest możliwość hostowania aplikacji wyłącznie na systemie operacyjnym Windows. Nie ma natomiast przeciwwskazań do wystawienia serwera HTTP.sys w sieci publicznej – serwer IIS wewnętrznie również korzysta z modułu HTTP.sys. Posiada on duże możliwości konfiguracyjne i jest rozwijany od ponad 15 lat. Po przeprowadzeniu testów można stwierdzić, że model ten umożliwia uzyskanie zdecydowanie wyższej wydajności od konfiguracji standardowej (50% w przypadku HTML i około 180% w przypadku JSON, XML i czystego tekstu). Jest on jednak mniej wydajny od serwera Kestrel o około 10-20%.

Kolejną ciekawą możliwością jest konfiguracja dostępna dopiero od wersji 2.1.0-preview1 platformy ASP.NET Core, która również polega na wykorzystaniu serwera wewnętrznego Kestrel i serwera pośredniczącego IIS. Jednak w odróżnieniu od modelu standardowego aplikacja z serwerem Kestrel nie jest uruchamiana wewnątrz własnego procesu, a wewnątrz procesu IIS. Dzięki takiemu rozwiązaniu zaobserwować można znaczny wzrost wydajności. Liczba przetwarzanych zapytań na sekundę jest mniejsza od konfiguracji z samym Kestrelem o około 10-20% jednak uzyskujemy dużo większe możliwości konfiguracyjne za sprawą wykorzystania IIS.

6. Wnioski

W artykule porównane zostały różne modele hostowania aplikacji napisanej przy użyciu platformy ASP.NET Core. Uniezależnienie się od serwera IIS oraz systemu operacyjnego

Windows to jeden z głównych elementów, który odróżnia platformę ASP.NET Core od jej poprzednika ASP.NET 4.6.

Najwydajniejszym z testowanych modeli okazał się model używający serwera Kestrel bez użycia serwera pośredniczącego reverse-proxy, jednak równocześnie jest to model posiadający dość małe możliwości konfiguracyjne. Najmniej wydajnym, ale z kolei zapewniającym największe bezpieczeństwo i możliwości konfiguracyjne jest model składający się z serwera wewnętrznego Kestrel oraz serwera reverse-proxy IIS.

Otrzymane wyniki zgadzają się z tymi, które przedstawione zostały w artykule znajdującym się w sieci [3]. W obu przypadkach zastosowanie samego serwera Kestrel, zamiast konfiguracji Kestrel + IIS (outofprocess) zapewniło kilkukrotny wzrost wydajności.

Należy zwrócić jednak uwagę na fakt, że testy przeprowadzone były na zwykłym laptopie z procesorem Intel Core i3-3110M, a liczba zapytań na sekundę możliwych do przetworzenia sięgała kilku tysięcy. Na wyspecjalizowanym sprzęcie prawdopodobnie bez problemu można uzyskać wartości rzędu kilkudziesięciu lub nawet kilkuset tysięcy zapytań na sekundę. Wartości takie prawdopodobnie nigdy nie będą konieczne do uzyskania w aplikacjach biznesowych, a nawet jeśli to zdecydowanie wcześniej natkniemy się inne ograniczenia wydajnościowe – np. związane z zapytaniami do bazy danych.

Wybór modelu hostingowego w większości przypadków należy zatem oprzeć o wymagane możliwości konfiguracyjne oraz łatwość zarządzania aplikacją. W przypadku aplikacji w sieci wewnętrznej najlepszym rozwiązaniem będzie użycie samego serwera wewnętrznego Kestrel z uwagi na jego dużą wydajność oraz łatwość uruchomienia. Dla aplikacji w sieci publicznej, które mogą wymagać bardziej skomplikowanej konfiguracji należy użyć konfiguracji Kestrel + IIS. Na ten moment najbezpieczniejszy jest model standardowy, czyli oddzielny proces IIS i oddzielny proces aplikacji z Kestrelem, jednak prace nad nowym modelem (Kestrel z aplikacją działający wewnątrz procesu IIS) ciągle trwają i niedługo powinien być on gotowy do użycia produkcyjnego [8]. Wtedy model ten będzie najodpowiedniejszym z uwagi na połączenie zalet wynikających z użycia samego Kestrela (duża wydajność) i serwera pośredniczącego IIS (duże możliwości konfiguracyjne i bezpieczeństwo).

Literatura

[1] A. Freeman, Pro ASP.NET Core MVC, Apress, 2016

[2] https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/?view=aspnetcore-2.0&tabs=aspnetcore2x [20.05.2018]

[3] https://odetocode.com/blogs/scott/archive/2016/10/25/asp-net-core-and-the-enterprise-part-2-hosting.aspx [12.06.2018]

[4] https://docs.k6.io/docs [20.05.2018]

[5] A. Lock, ASP.NET Core in Action, Manning, 2017

[6] https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/aspnet-core-module?view=aspnetcore-2.0 [20.05.2018]

[7] https://blogs.msdn.microsoft.com/webdev/2018/02/28/asp-net-core-2-1-0-preview1-improvements-to-iis-hosting/ [20.05.2018]

[8] https://github.com/aspnet/IISIntegration/issues/878 [12.06.2018]