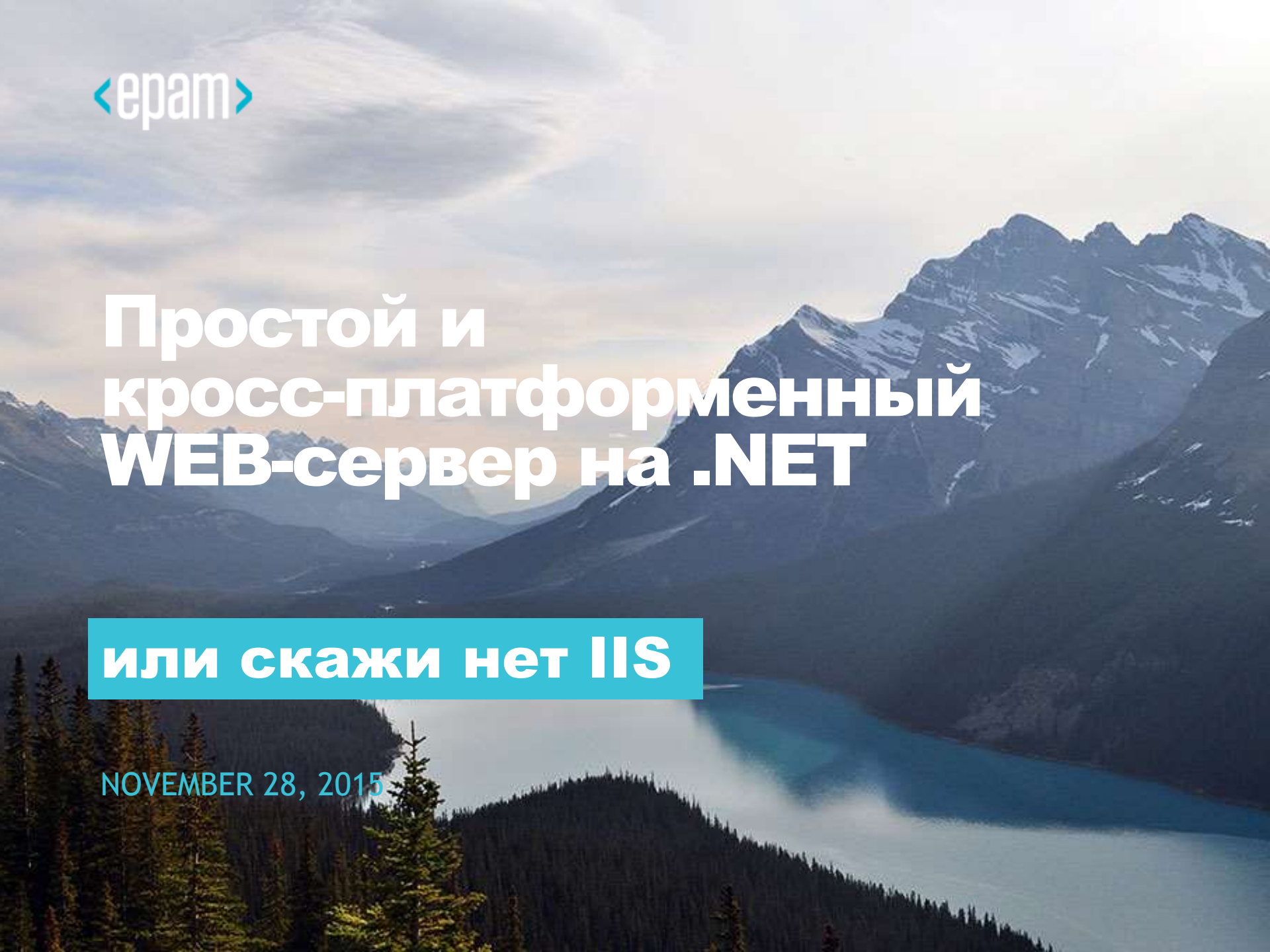


Простой и кросс-платформенный WEB-сервер на .NET

или скажи нет IIS

NOVEMBER 28, 2015

Кто мы

Александр Иванов

- С 2007 года занимаюсь Web разработкой на .NET

Aleksandr_Ivanov@epam.com

Роман Правук

- С 2004 года занимаюсь Web разработкой
- Ресурсный Менеджер группы Web .Net разраьотчиков

Roman_Pravuk@epam.com

Мы Делали Продукт

Муки выбора архитектуры серверной части

Нам был нужен REST сервис

- Легкость владения (админы сказали - нет тяжелым веб серверам)
- Слабая аппаратная часть (квоты на Cloud)
- Скорость разработки

Из чего выбирали

- Node.js
- WCF self hosted
- ASP.NET MVC WEB API
- Spring MVC + Apache Tomcat

Первый претендент – Node.js

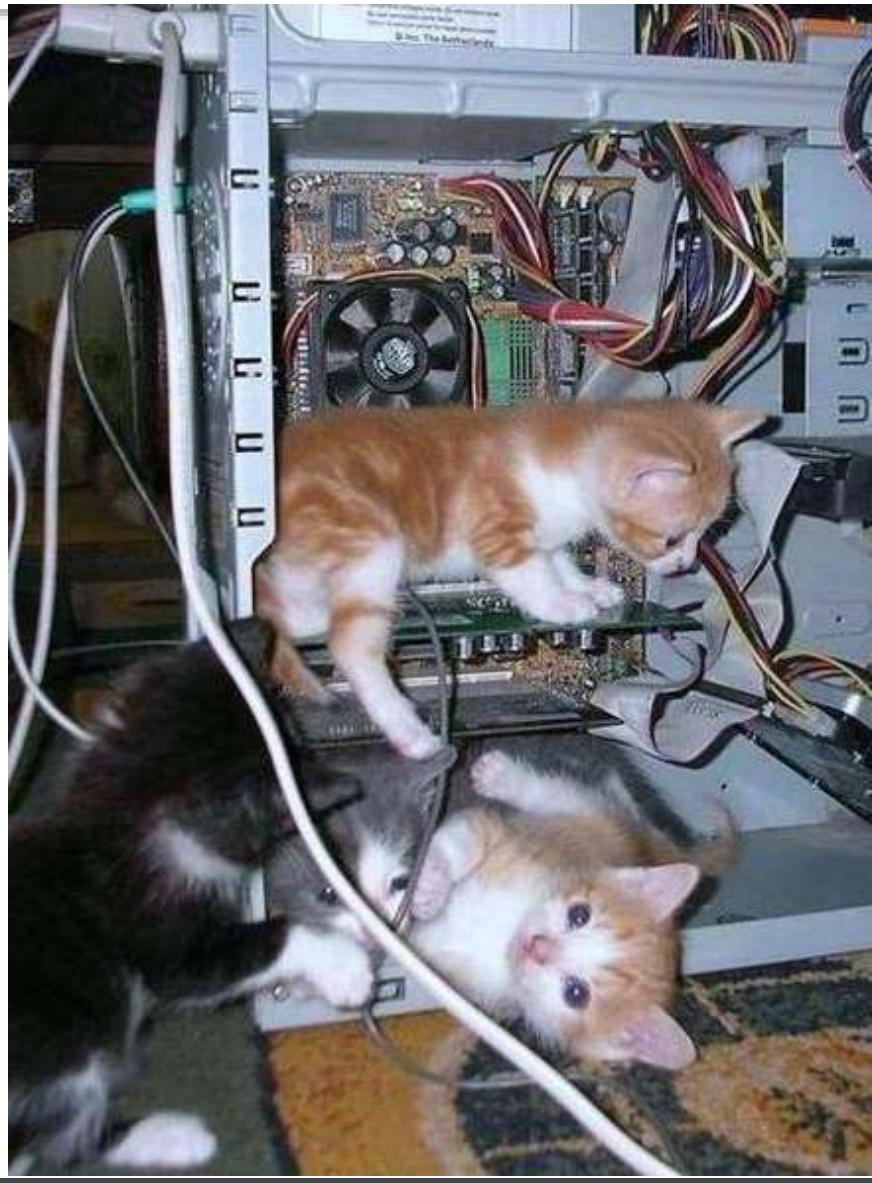
- Просто установить
- Легко обслуживать
- Можно быстро написать REST сервис

Почему не Node.js

- Один поток
- Любой из запросов забирающий управление на себя наглухо забирает всю очередь
- Нет адекватных средств работы с AD и Exchange Server
- Не типизированный
- Нет средств синхронизации потоков

Почему не Java WEB-стэк

- Не было экспертизы
- Сложно поддерживать Apache Tomcat. Админы отказались.



ASP.NET

- 2002 год

  .NET Framework 1.0 и ASP.NET Web Forms

- 2009 год

  ASP.NET MVC

- 2012 год

  ASP.NET WEB API и Self-Hosting

Проблемы ASP.NET

- WEB-приложения только под Windows
- IIS – единственная опция WEB-сервера в .NET (кроме WEB API)
- Зависимость от System.Web.dll – огромная монолитная сборка
- ASP.NET – закрытая платформа


Решает архитектурные проблемы ASP.NET
OWIN

OWIN

- Open
- Web
- Interface
- for .Net

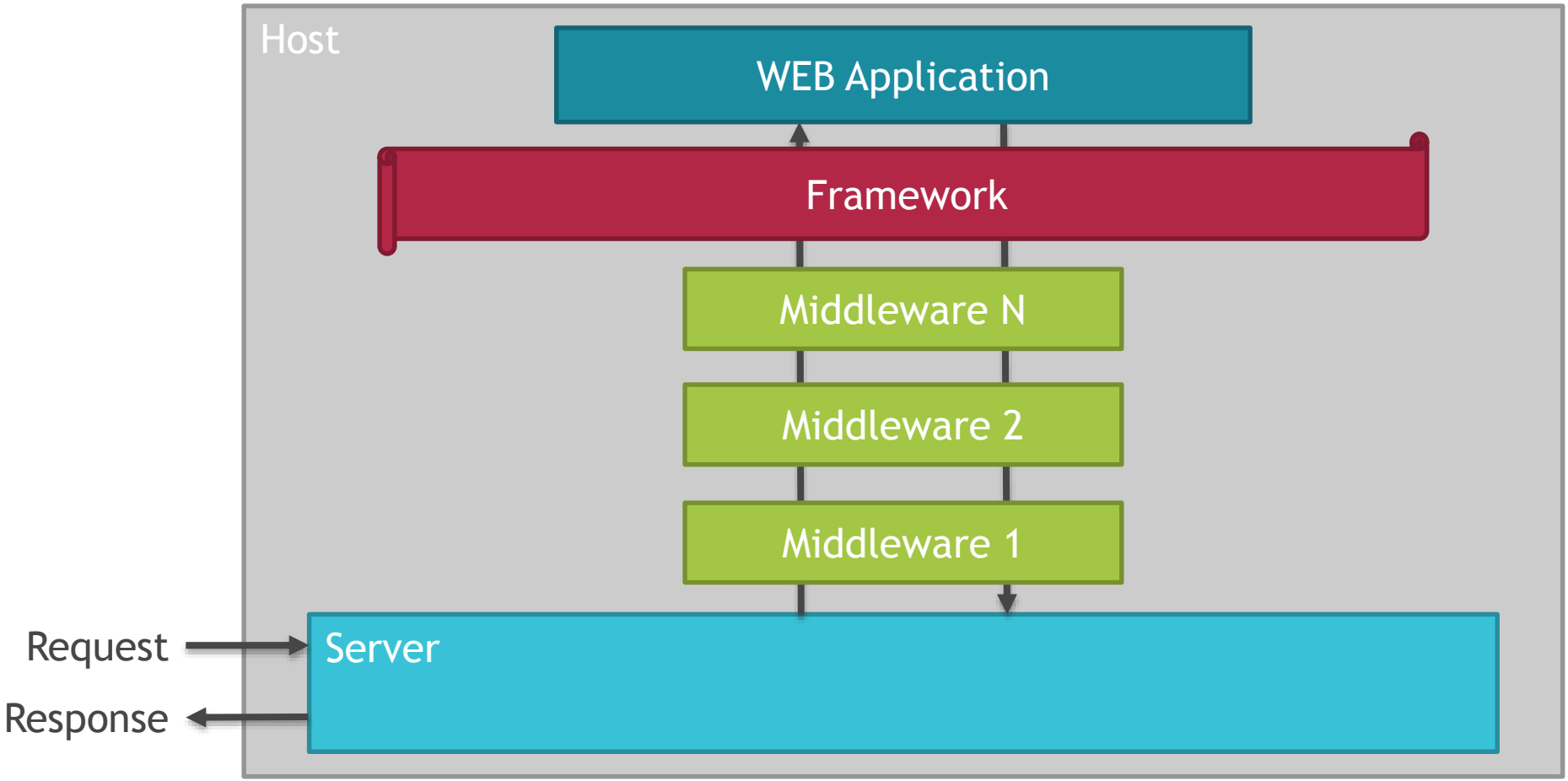
Открытый WEB-стандарт для .NET

- OWIN – это стандарт (спецификация), не Фреймворк
- ASP.NET 5 реализует стандарт OWIN
- OWIN регулирует порядок взаимодействия между WEB-сервером и WEB-приложением
- Стандарт направлен на создание небольших и простых модулей для разработки WEB-приложений
- OWIN – это открытый стандарт

  https://github.com/owin

  http://www.owin.org

Составные части OWIN



Интерфейсы OWIN

Environment Dictionary

```
IDictionary<string, object>
```

Application Delegate

```
Func<IDictionary<string, object>, Task>
```

Environment Dictionary – Request Data

Key Name	Description
"owin.RequestBody"	A Stream with the request body (Stream.Null if no request body)
"owin.RequestHeaders"	An IDictionary<string, string[]> of request headers.
"owin.RequestMethod"	"GET", "POST", etc.
"owin.RequestPath"	A string containing the request path relative to the "root" of app delegate.
"owin.RequestPathBase"	A string containing the portion of the request path corresponding to the "root"
"owin.RequestProtocol"	"HTTP/1.0", "HTTP/1.1"
"owin.RequestQueryString"	"foo=bar&baz=qx" (without leading "?"; empty string if no query params)
"owin.RequestScheme"	"http", "https"

Environment Dictionary – Response Data & other

Key Name	Description
"owin.ResponseBody"	A Stream used to write out the response body, if any.
"owin.ResponseHeaders"	An IDictionary<string, string[]> of response headers.
"owin.ResponseStatusCode"	An optional int containing the HTTP response status code. The default is 200.
"owin.ResponseReasonPhrase"	An optional string containing the reason phrase associated the given status code.
"owin.ResponseProtocol"	An optional string containing the protocol name and version (e.g. "HTTP/1.0")
"owin.CallCancelled"	A CancellationToken indicating if the request has been cancelled/aborted.
"owin.Version"	The string "1.0" indicating OWIN version.

Application Delegate + Environment Dictionary

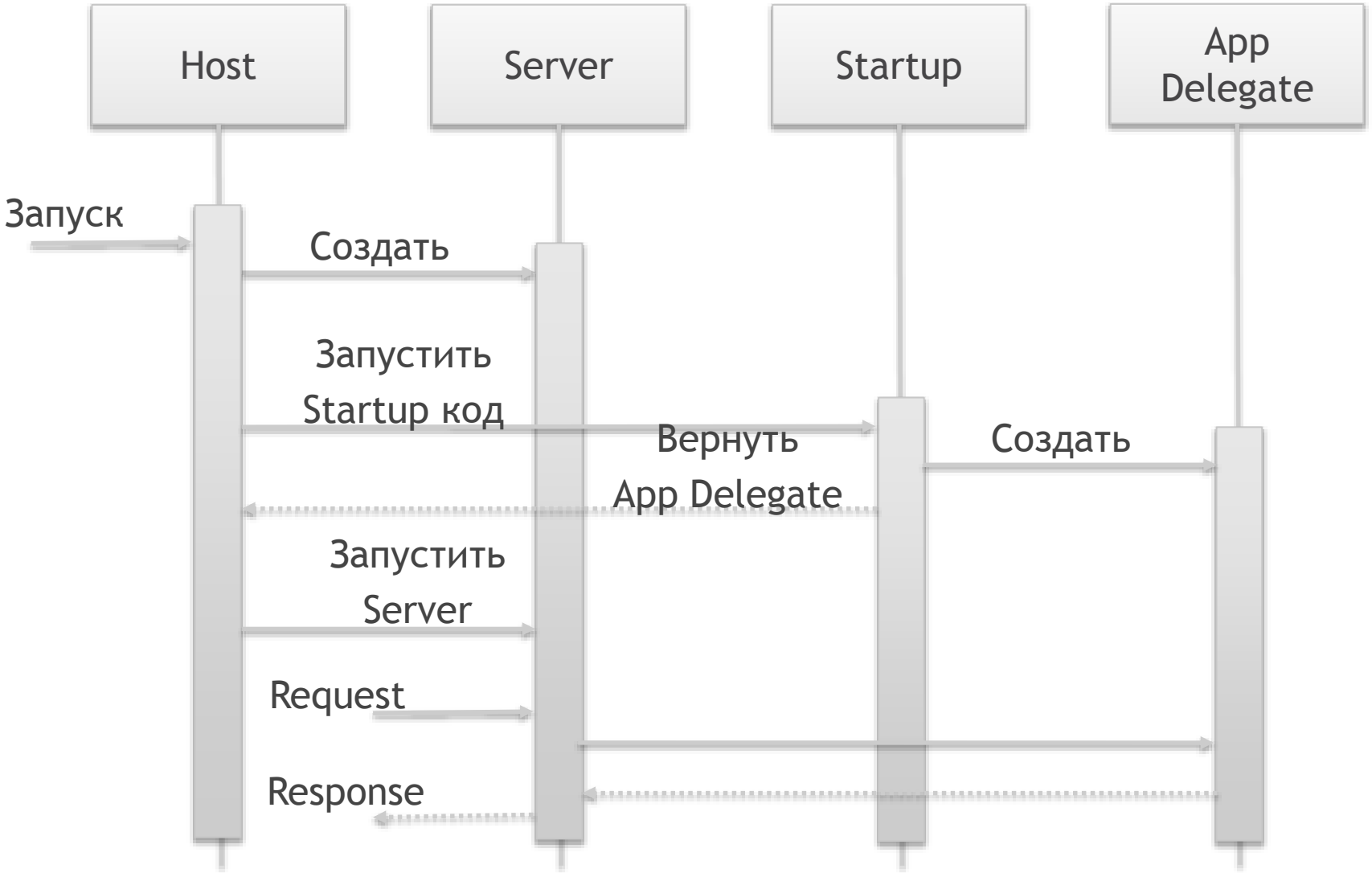
Это - HttpContext

```
public Task AppDelegate(IDictionary<string, object> environment) {
  string responseText = "Hello, World!";
  byte[] responseBytes = Encoding.UTF8.GetBytes(responseText);

  Stream responseStream = (Stream) environment["owin.ResponseBody"];
  IDictionary<string, string[]> responseHeaders =
    (IDictionary<string, string[]>) environment["owin.ResponseHeaders"];

  responseHeaders["Content-Type"] = new [] {"text/plain"};
  responseHeaders["Content-Length"] = new [] {responseBytes.Length.ToString()};

  return responseStream.WriteAsync(responseBytes, 0, responseBytes.Length);
}
```

Возвращаем Task

Task пишет в Response

Запуск OWIN приложения



Пример WEB-сервера

Startup.cs

```
using System.Threading.Tasks;
using Microsoft.AspNet.Builder;
using Microsoft.AspNet.Http;

namespace My.OwinServer {
  public class Startup {
    public void Configure(IApplicationBuilder app) {
      app.Use(next => AppDelegate);
    }

    private Task AppDelegate(HttpContext context) {
      string responseText = "Hello, World!";
      return context.Response.WriteAsync(responseText);
    }
  }
}
```

Создаём Middleware pipeline

Обработчик запросов

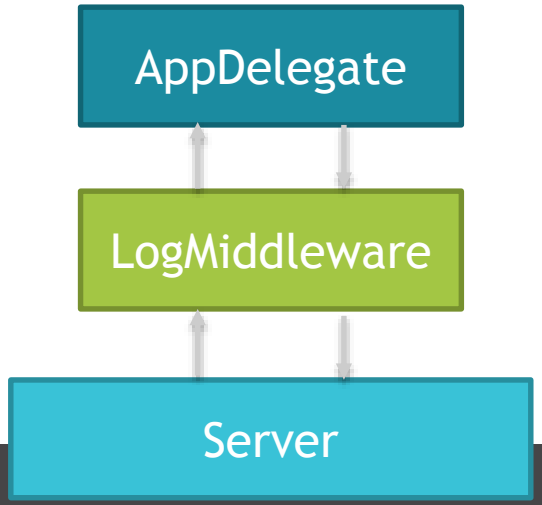
Цепочка Middleware

```
public class LogMiddleware {
  private readonly RequestDelegate _next;

  public LogMiddleware(RequestDelegate next)
  {
    _next = next;
  }

  public async Task Invoke(HttpContext context)
  {
    string path = context.Request.Path;
    Console.WriteLine("Begin requst " + path);

    await _next(context);

    Console.WriteLine("End requst " + path);
  }
}
```

Следующий Middleware

```
public class Startup {
  public void Configure(IApplicationBuilder app)
  {
    app.UseMiddleware<LogMiddleware>();
    app.Run(AppDelegate);
  }
  private Task AppDelegate(HttpContext context)
  {
    string responseText = "Hello World!";
    return context.Response
        .WriteAsync(responseText);
  }
}
```

Цепочка Middleware



Доступные Middleware

- Identity
- Routing
- Security (Cookies, Facebook, Google, Microsoft, Twitter, etc.)
- Localization
- Session
- Diagnostics
- WebSockets
- StaticFiles
- ResponseCaching
- etc.

Собираем WEB-сервер

DNX-проект

```
{
  "dependencies": {
    "Microsoft.AspNet.Owin": "1.0.0-rc1-final",
    "Microsoft.AspNet.Server.Kestrel": "1.0.0-rc1-final",
    "Microsoft.AspNet.Hosting": "1.0.0-rc1-final"
  },

  "frameworks": {
    "dnxcore50": {}
  },

  "commands": {
    "web": "Microsoft.AspNet.Server.Kestrel --server.urls http://localhost:5000"
  }
}
```

DNX

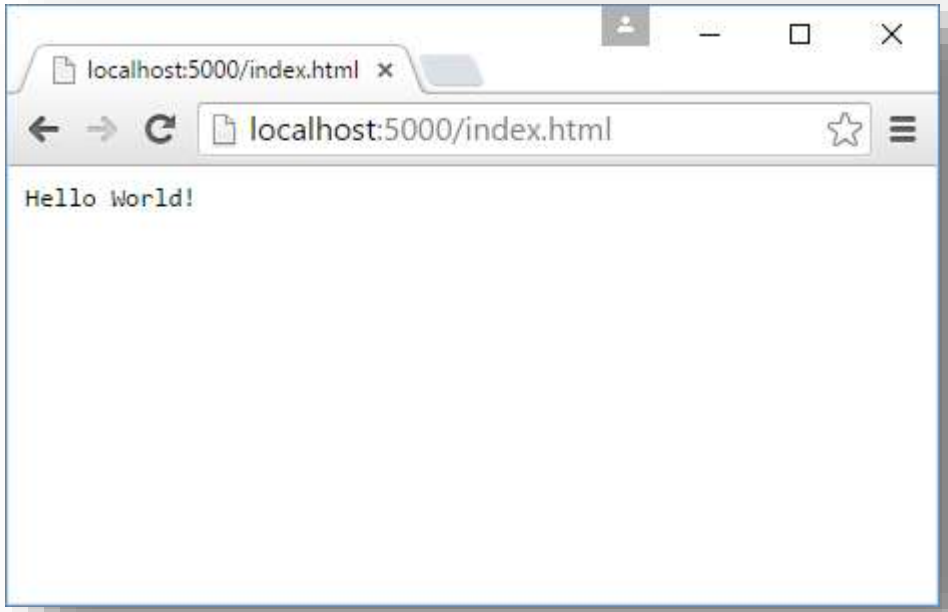
DNX - это SDK и среда выполнения

- **dnvm** - .NET Version Manager
- **dnu** - .NET Development Utility
- **dnx** - The .NET Execution Environment.

Команды определены в файле **project.json**:

```
"commands": {
    “kestrel": "Microsoft.AspNet.Server.Kestrel --server.urls http://localhost:5000",
    "web": "Microsoft.AspNet.Server.WebListener --server.urls http://localhost:5000"
}
```

Запускаем WEB-сервер



Сборка:

```
$ dnu restore
$ dnu build
```

Запуск:

```
$ dnx web
Hosting environment: Production
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
Begin requst /index.html
End requst /index.html
```

Добавляем ASP.NET WEB API

project.json

```
{
  "webroot": "wwwroot",
  "dependencies": {
    "Microsoft.AspNet.Mvc": "6.0.0-rc1-final",
    "Microsoft.AspNet.Server.Kestrel": "1.0.0-rc1-final",
    "Microsoft.AspNet.StaticFiles": "1.0.0-rc1-final"
  },
  "commands": {
    "web": "Microsoft.AspNet.Server.Kestrel --server.urls http://localhost:5000/"
  },
  "frameworks": {
    "dnxcore50": { }
  },
  "exclude": [
    "wwwroot",
    "node_modules",
    "bower_components"
  ]
}
```

WEB API приложение

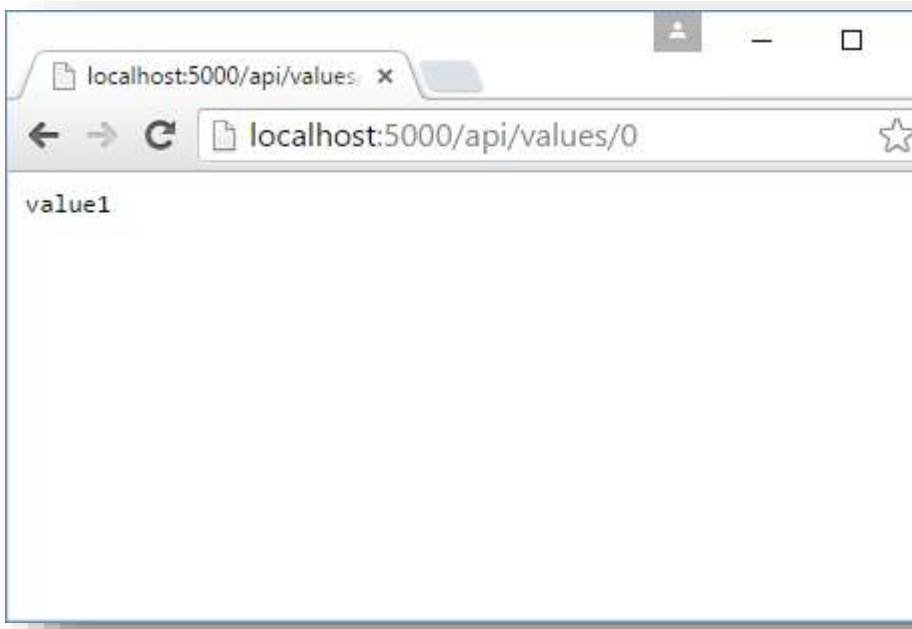
Startup.cs

```
using Microsoft.AspNet.Builder;
using Microsoft.Framework.DependencyInjection;

public class Startup {
  public void ConfigureServices(IServiceCollection services) {
    services.AddMvc();
  }

  public void Configure(IApplicationBuilder app) {
    app.UseStaticFiles();
    app.UseMvc();
  }
}
```

WEB API контроллер

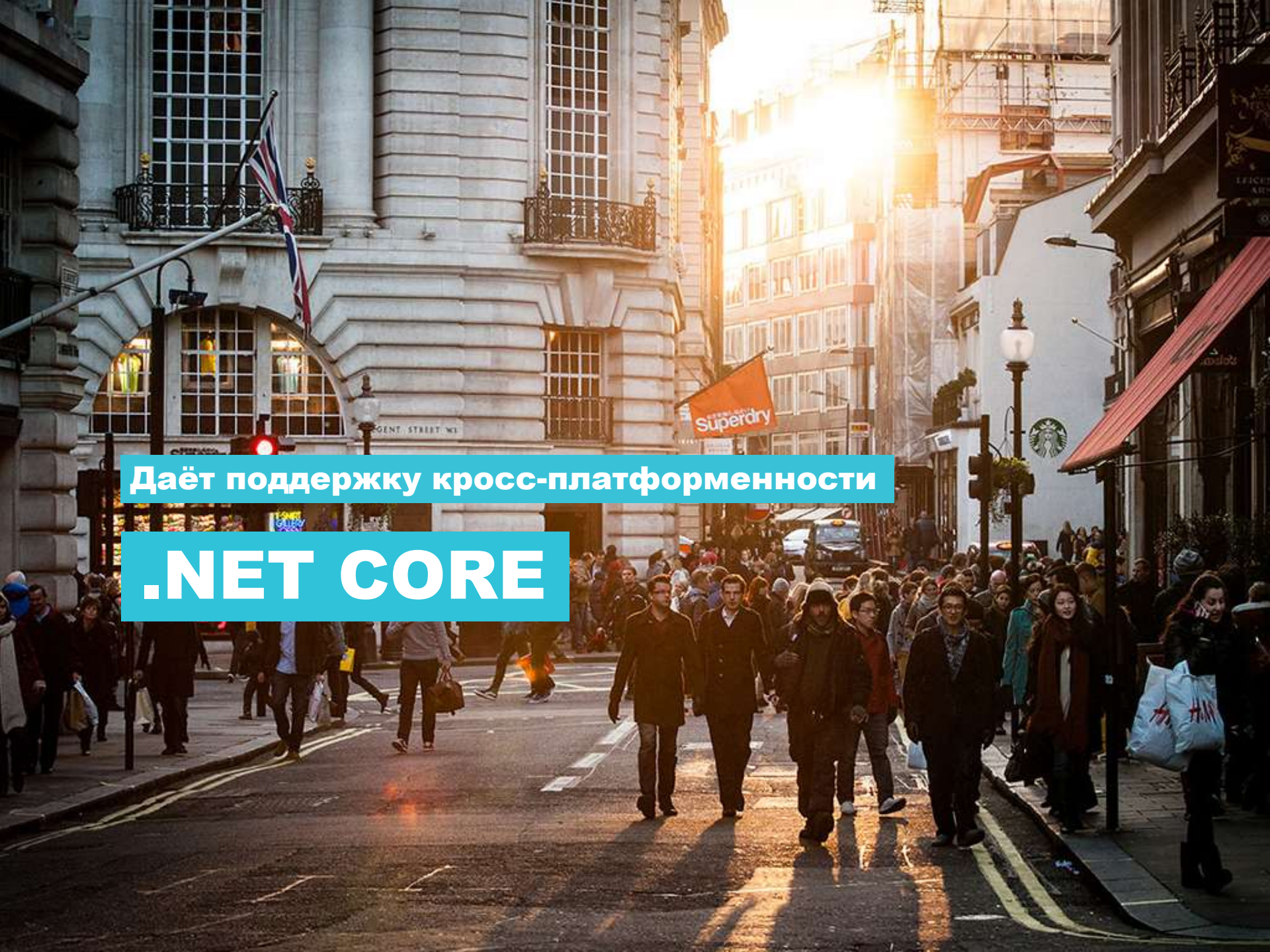
ValuesController.cs

```
using Microsoft.AspNet.Mvc;

[Route("api/[controller]")]
public class ValuesController : Controller {
  private readonly List<string> _values = new List<string> { "value1", "value2" };

  [HttpGet]
  public IEnumerable<string> Get() {
    return _values;
  }
  [HttpGet("{id}")]
  public string Get(int id) {
    return 0 <= id && id < _values.Count
      ? _values[id] : "Value not found";
  }
  [HttpPost]
  public void Post([FromBody]string value) {
    _values.Add(value);
  }
  [HttpPut("{id}")] ...
}
```



Куда мы двигаемся дальше

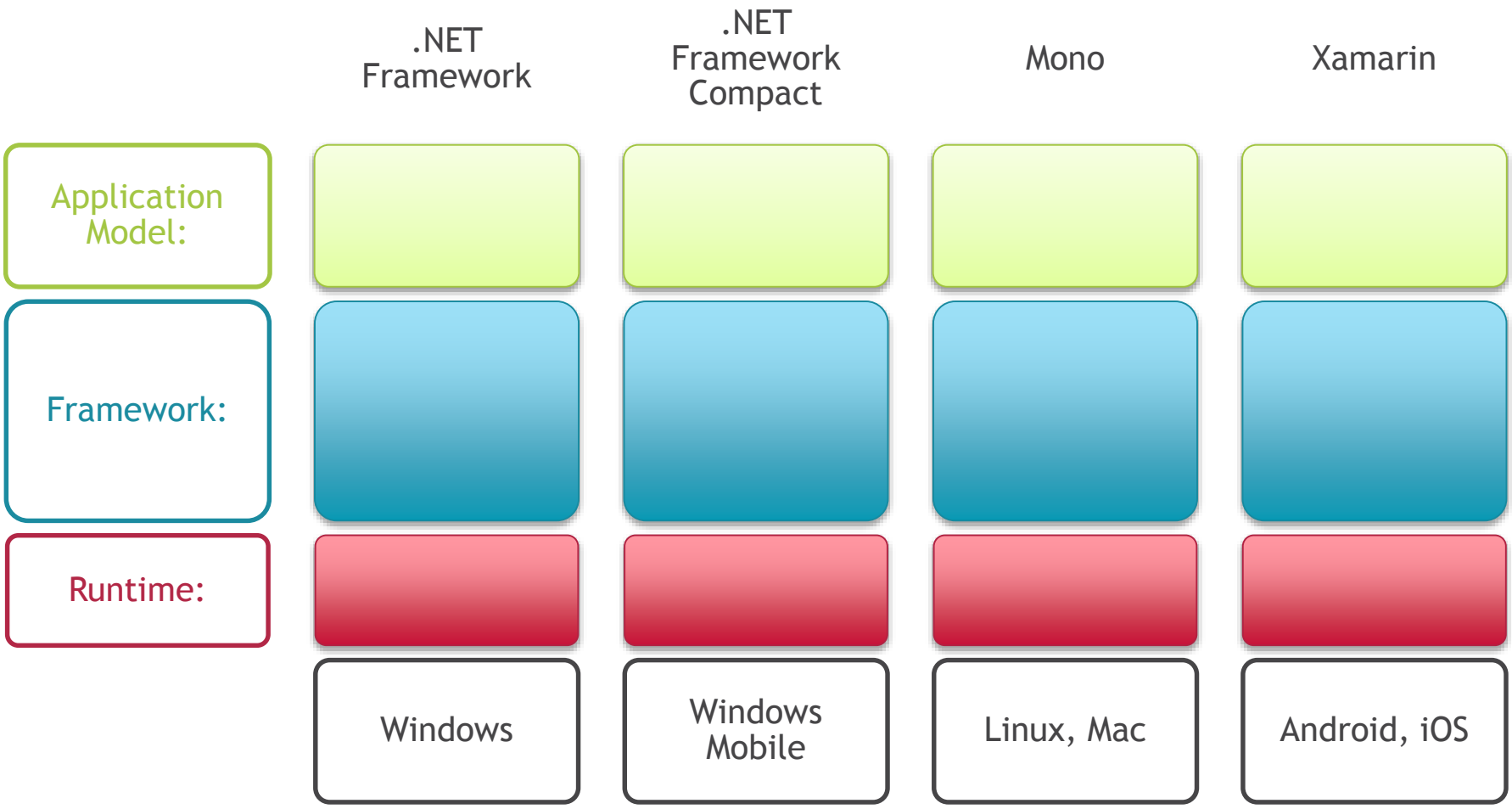
1. Запустить всё под Linux на Mono
   - Не получится: EWS использует SecureString
2. Всё на .NET Core
   - EWS нет для .NET Core



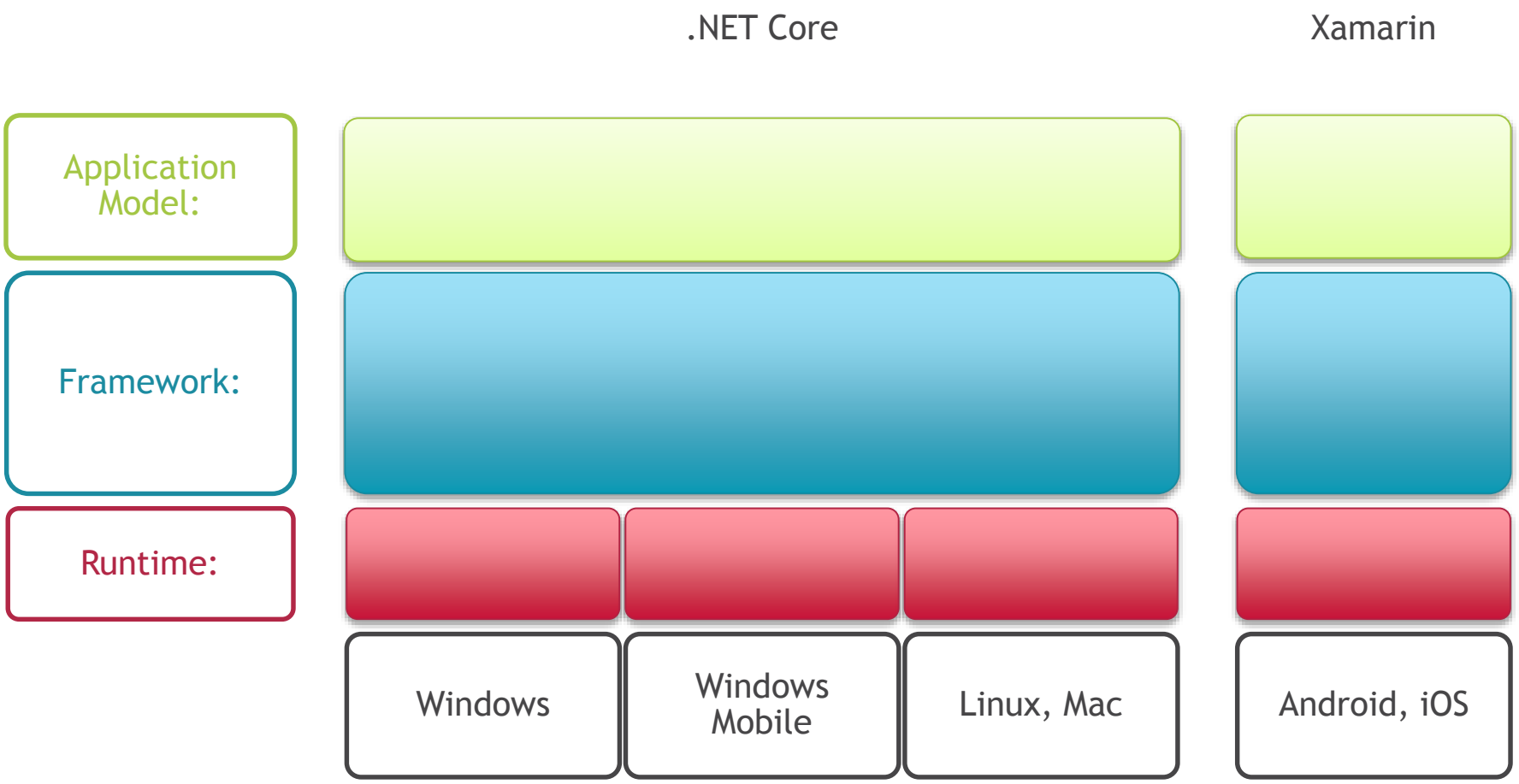
Даёт поддержку кросс-платформенности

.NET CORE

Существующие реализации .NET



.NET Core


.NET Core
Xamarin
Application Model:
Framework:
Runtime:
Windows
Windows Mobile
Linux, Mac
Android, iOS

.NET Core

.NET Core

- CoreCLR – runtime
- CoreFX – набор библиотек – Подмножество библиотек .NET Framework.

.NET Core поддерживает платформы:

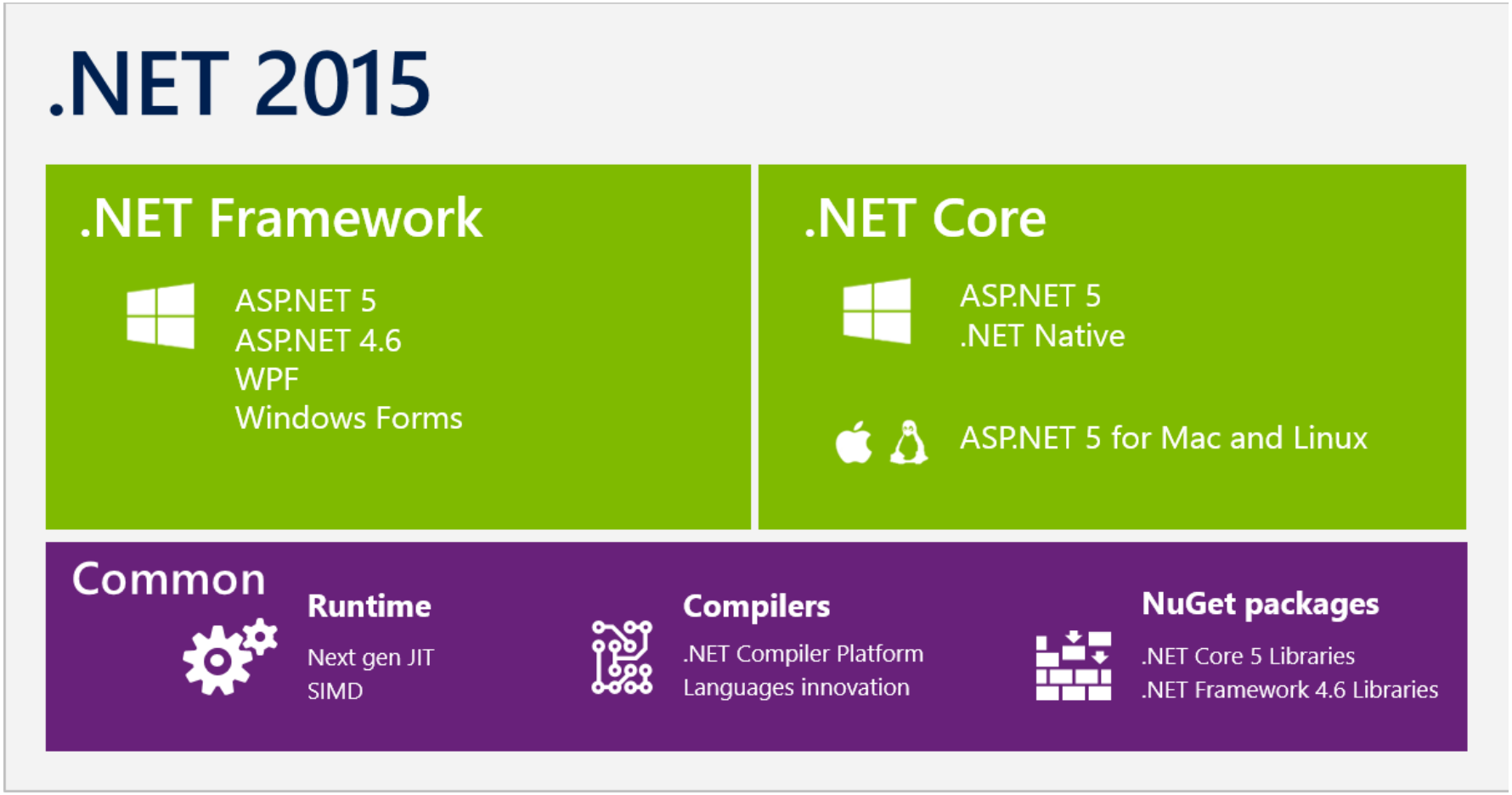
- Windows
- Linux
- Mac

Что даёт .NET Core

- Портируемость
- Кроссплатформенность
- Модульность – CoreFX разбит на небольшие NuGet пакеты. Позволяет включить только те зависимости, которые нужны.
- NET Core – это Open-Source проект. https://github.com/dotnet/core


.NET Core
ASP.NET 5

ASP.NET 5


.NET 2015
.NET Framework
ASP.NET 5
ASP.NET 4.6
WPF
Windows Forms
.NET Core
ASP.NET 5
.NET Native
ASP.NET 5 for Mac and Linux
Common
Runtime
Next gen JIT
SIMD
Compilers
.NET Compiler Platform
Languages innovation
NuGet packages
.NET Core 5 Libraries
.NET Framework 4.6 Libraries

ASP.NET 5 на .NET Core

- Оптимизирован под Cloud и Серверную архитектуру
  - Использует мало памяти - проект включает только те зависимости, которые реально использует
  - Высокая пропускная способность
- Кросс-платформенный: Windows, Linux, Mac.
- Microsoft будет развивать .NET Core

Проблемы ASP.NET 5 и .NET Core

- CoreFX – содержит лишь подмножество библиотек .NET Framework
- Пока ещё мало пакетов в NuGet поддерживают .NET Core

ASP.NET 5 - Summary

- Кросс-платформенный
- Независимый от WEB-сервера
- Модульный подход
- Выше производительность
- Требует меньше памяти
- Портируемое приложение

Полезные ссылки

- http://www.asp.net
- https://github.com/aspnet
- https://github.com/dotnet
- https://github.com/owin
- https://docs.asp.net
- http://blogs.msdn.com/b/dotnet/archive/2014/12/04/introducing-net-core.aspx