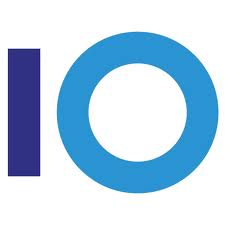
**Универзитет Св. ʻʻКирил и Методијʼʼ – Скопје**

**Факултет за информатички науки и компјутерскo**

**инженерство**

**Дипломска работа**

Анализа на **.NET Core**

Ментор: Кандидат:

Проф. Д-р. Иван Чорбев Војдан Гичаровски

Индекс број: 111040

Скопје, 2017

Содржина

[Апстракт 3](#_Toc481962292)

[Клучни зборови 3](#_Toc481962293)

[Вовед 4](#_Toc481962294)

[Отворен изворен код 5](#_Toc481962295)

[Наменета за повеќе оперативни системи и платформи 6](#_Toc481962296)

[Mono 7](#_Toc481962297)

[Xamarin 7](#_Toc481962298)

[Разлики на .NET Core и .NET Framework повици 8](#_Toc481962299)

[Компоненти 9](#_Toc481962300)

[C# Компајлер – Roslyn 9](#_Toc481962301)

[Алатка за градење софтвер – MSBuild 10](#_Toc481962302)

[Алатки за командна линија – dotnet 12](#_Toc481962303)

[Библиотеки што се користат при извршување на софтвер – CoreCLR 13](#_Toc481962304)

[Библиотеки – CoreFX 14](#_Toc481962305)

[.NET Стандард 15](#_Toc481962306)

[Менаџер на пакети – NuGet 17](#_Toc481962307)

[Рамка за развивање веб апликации - ASP.NET Core 19](#_Toc481962308)

[OWIN 20](#_Toc481962309)

[Kestrel 21](#_Toc481962310)

[Docker 22](#_Toc481962311)

[Интегрирана околина за развој - Visual Studio 2017 23](#_Toc481962312)

[Пример апликација и разлика при развој со .NET Framework 4.6.2 и .NET Core 1.0 23](#_Toc481962313)

# Апстракт

Во оваа дипломска работа се прави анализа на .NET Core рамката изработена од компанијата Microsoft. .NET Core е следната верзија на рамката .NET Framework којашто се користи при изработка на сите типови на апликации како веб, мобилни и клиентски апликации со помош на јазикот C#.

Во почетокот на оваа дипломска работа ќе се зборува за историското значење на .NET Core и како тоа влијае на програмерската заедница во целост, но и како влијае конкретно на компанијата Microsoft.

Главна цел е да се опфатат новостите и разликите на .NET Core во однос на неговиот претходник .NET Framework, но исто така и да се објаснат подетално сите негови компоненти и промените кои ги претрпеле со новата верзија на .NET.

## Клучни зборови

.NET Core, .NET Framework, ASP.NET Core, NuGet, Roslyn, .NET Standard, Open source, Cross platform, Docker, OWIN, Kestrel

# Вовед

Денес живееме во дигитално доба каде што компјутерите и мобилните уреди ни помагаат при извршување на секојдневните обврски, ни ги олеснуваат и забрзуваат здодевните задачи како плаќање сметки, пазарење, но и ни овозможуваат да се забавуваме и комуницираме со пријателите, да можеме да играме игри, да слушаме музика или да гледаме некој интересен филм.

Развитокот на компјутерите не е краток и едноставен, од првиот електричен компјутер со сијалици оспособен за програмирање, до денешните компјутери со микро-транзистори и по неколку гигабајти меморија и огромна процесорска моќ поминати се повеќе од 70 години. Развојот на хардверот и материјалите што се користат за негова изработка, но и самиот процес на изработка напреднале значително за во денешно време да можеме во нашиот џеб да носиме компјутер со поголема процесорска моќ од компјутерот што се користел за лансирање и водење на леталото Аполо 11.

Паралелно како што се развивал хардверот со него се развивал и софтверот којшто го користи. Од првата верзија на Windows развиена во 1985 до денес можеме да забележиме огромен напредок во функционалностите што ги нуди, интеракцијата и начинот на изработка на апликации.

За да може да се изработи една апликација потребни се компајлер за соодветниот јазик кој што се користи за програмирање, кој во тоа време бил јазикот Ц и библиотеки за интеракција со периферните уреди. Во денешно време постојат голем број на јазици за програмирање меѓу кои и јазикот C# којшто е осмислен, изработен и одржуван од страна компанијата Microsoft. Овој јазик сам по себе не може да се користи за да се направи продуктивна апликација, туку мора да се употребува во комбинација со рамката .NET за изработка на апликација способна за извршување и комуникација со периферните уреди.

.NET претставува колекција на библиотеки за менаџмент на меморија, редослед на извршување и преведување на програмскиот код во инструкции разбирливи за компјутерот. Во сржта .NET содржи библиотека наречена CLR (common language runtime) што всушност го преведува напишаниот код во соодветни машински инструкции спремни за извршување. Останатите библиотеки помагаат за интеракција со периферните уреди и ги олеснуваат операциите со различните типови на информации (броеви, букви, реченици, низи, итн.).

# Отворен изворен код

Во изминатите години Microsoft беше компанија којашто своите технологии ги изработуваше во тајност, но со новата верзија на .NET се реши да го објави изворниот код на технологијата .NET Core, компајлерот и компонентите. Со овој чекор Microsoft придонесе за развој на технологија којашто е најблизу до потребите на развивачите на софтвер коишто ќе ја употребуваат за развој на најразлични апликации.

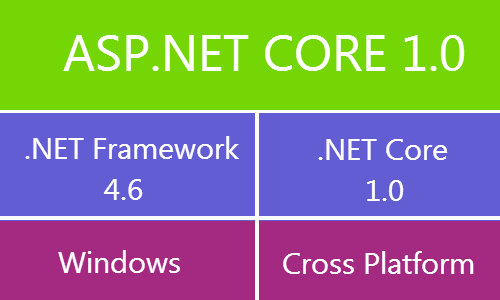
Кодот е отворен за разгледување и надополнување за сите заинтересирани развивачи, но тие мораат да ги почитуваат правилата поставени од страна тимот на Microsoft што е одговорен за соодветната технологија. Овие правила обично се однесуваат на стилот на пишување код, шаблоните што се употребуваат, стилот на пишување GIT пораки, авторски права на код, процес за поднесување за барање за додавање или модифицирање на код, начин и место за дискусија за проблемите што се треба да се решат, стандард на јавни повици и структури којшто библиотеките мора да го запазат и други правила кои што се специфични за секоја компонента поодделно.

Иако кодот е отворен за модификации стандардот за функционалностите што оваа рамка мора да ги нуди е однапред одреден и напишан како засебна библиотека .NET Standard којашто мора да биде интегрирана, што во суштина значи дека сите рамки што ја интегрираат оваа базична библиотека ќе го имаат истиот јавен интерфејс за своите базични функционалности. Сите останати дополнителни функционалности развиени во рамките на .NET Core ќе бидат специфични само за оваа рамка.

Овој стандард го решава проблемот кој се јавува при споделување на библиотеки во различни платформи и рамки. На пример ако еден развивач на софтвер напише библиотека за манипулација на дати и временски зони и таа библиотека е зависна од .NET Standard 1.6 тогаш таа библиотека ќе може да биде искористена во било која апликација што користи рамка (.NET Framework vNext или .NET Core 1.0) што го интегрира соодветниот стандард.

# Наменета за повеќе оперативни системи и платформи

Најзначајната промена на .NET рамката со изработката на верзијата .NET Core е што ќе биде достапна за развој и користење на повеќе платформи и оперативни системи. Во минатото Microsoft се фокусирале да изградат рамка за развој којашто работи ексклузивно за Windows, но сега тие изработуваат рамка којашто ќе може да се извршува на повеќето познати дистрибуции на Linux и на macOS.



**Слика бр. 1 Достапни платформи за ASP.NET Core апликација**

Оваа промена има посебно значење за развивачите на софтвер затоа што сега нема да мора да се потпираат на други алатки за програмирање за повеќе платформи одеднаш како Mono, Cordova или Qt, туку ќе можат да ја користат рамката .NET Core.

Исто така оваа промена ги засегнува и клиентите затоа што сега веб апликациите не мора да бидат хостирани ексклузивно на Windows сервер, туку може и на сервер со Linux оперативен систем доколку има поддршка за .NET Core рамката.

## Mono

Mono е првиот обид да се произведе платформа со ист јавен стандард како .NET Framework којашто би работела на повеќе различни оперативни системи и уреди како Linux, Solaris, BSD, macOS, Android, iOS, Wii, Xbox 360, PlayStation 3. Овој проект е изработен со цел да се олесни развивањето на .NET апликации на Linux оперативниот систем, но и да се даде можност да се развиваат квалитетни апликации со користење на C# јазикот за повеќе платформи одеднаш.

Структурата на Mono платформата е направена според стандардите и шаблонот на .NET Framework коишто ги има објавено Microsoft под стандардите ECMA-334 и ECMA-335, односно развиени се C# компајлер за преведување на C# код во среден јазик, CLR (common language runtime) компајлер за статичко, навремено и предвремено компајлирање на среден јазик во машински код за соодветниот процесор, библиотеки според .NET Standard спецификацијата.

## Xamarin

Xamarin е компанија којашто е основана во 2011 од развивачите на проектот Mono со главна цел да развие платформа и интегрирана работна околина за развивање на софтвер за мобилните оперативни системи Android, iOS и Windows Phone во јазикот C# на било кој оперативен систем.

Овој потфат има повеќе големи значења за софтверската индустрија и за развивачите на софтвер за мобилни уреди. Пред да се развие овој проект развивачите на софтвер беа принудени да развиваат апликации поодделно за секој мобилен оперативен уред што најчесто значи развивање на истата апликација три пати во различни технологии и јазици соодветно за Android, iOS и Windows Phone. Потоа се развила платформата Cordova којшто во суштина апликациите ги прикажува како веб страни, иако овој начин на развивање апликации за повеќе оперативни системи одеднаш заштедува време, најчесто не се практикува затоа што апликациите развиени на овој начин се многу бавни.

Xamarin работи како обвивка на базичните библиотеки што ги нудат алатките за развивање на трите мобилни платформи што значи дека при компајлирање на напишаниот код во C# тој се преведува во базичен код за соодветната платформа. Со цел за користењето на овој софтвер да биде лесно и за развивачите коишто ги користат оперативните системи Linux и macOS, компанијата Xamarin има развиено интегрирана околина Xamarin Studio за работа со цел да ја замени околината за развој Visual Studio произведена од Microsoft којашто работи ексклузивно на Windows.

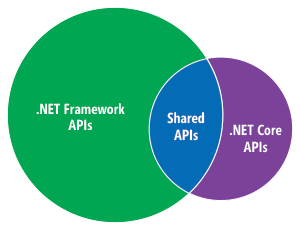


**Слика бр. 2 Xamarin архитектура на мобилни апликации**

Во февруари 2016 година Microsoft и Xamarin потпишале договор за откуп со што сега оваа компанија припаѓа на Microsoft. Ова има големо значење за развивачите на софтвер што користат Linux и macOS оперативни системи затоа што Microsoft започна со официјална поддршка на интегрираната околина Xamarin Studio. Со развитокот на .NET Core и неговата одлика дека може да се извршува на повеќе оперативни системи, развивачите на софтвер за прв пат добиваат шанса да можат да развијат веб, клиентски и мобилен софтвер во Xamarin Studio, што наликува со своите функционалности на Visual Studio(но ги нема сите), во јазикот C# на било кој оперативен систем што е поддржан од Xamarin Studio и .NET Core.

# Разлики на .NET Core и .NET Framework повици

Како што забележавме за разлика од .NET Framework, новата рамка .NET Core може да се извршува на повеќе различни оперативни системи, но за да може да се изведе тоа потребно е да се исфрлат сите функционалности што се специфични само за оперативниот систем Windows. Најголем дел од исфрлените функционалности се технологиите за изградба на клиентски апликации Windows Forms и WPF, па поради тоа може да се заклучи дека .NET Core претставува подмножество на повици и технологии коишто се способни да се извршуваат на повеќе оперативни системи.



**Слика бр. 3 Разлика помеѓу .NET Core и .NET Framework**

# Компоненти

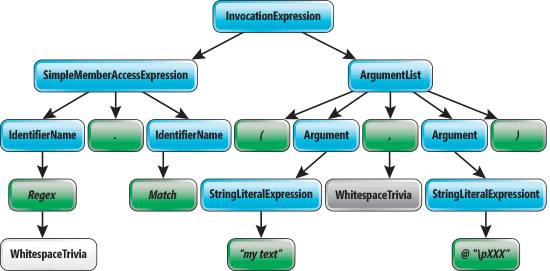
Развивањето на софтвер со помош на рамката .NET Core можеби изгледа како едноставна и лесна работа, но доколку навлеземе подлабоко во компајлирањето и извршувањето на напишаниот код, тестирање и пакување со цел оптимално прикачување на апликацијата на соодветната машина што ќе ја извршува брзо ќе сфатиме дека целата рамка е многу сложена и со цел да се олесни нејзиното одржување и ажурирање таа е поделена во повеќе компоненти.

## C# Компајлер – Roslyn

Како што работата на развивач на софтвер е да ја преведе бизнис логиката побарана од клиентот во соодветен код, така работата на компајлерот е да го преведе тој код во соодветни машински инструкции што може да ги разбере компјутерот, но во овој случај не се случува тоа. Поради тоа што првично програмерите имаат избор да развиваат софтвер со јазиците C# и Visual Basic, решението на Microsoft е овие јазици да се преведат во некој среден јазик (CIL – Common intermediate language) на пониско ниво близу машински код а тој потоа да се изврши со помош на CLR.

.NET Platform Compiler или попознат како Roslyn компајлер е алатката којашто го преведува C# кодот во соодветен CIL код. Со новата верзија на .NET Core доаѓаат и нови функционалности на компајлерот во форма на анализатори на синтакса, семантички анализатори и декомпајлери. Овие новости не се зависни од .NET Core рамката туку од верзијата и синтаксата којашто се поддржува на C# јазикот, но тоа не значи дека веќе постоечките функционалности не може да се подобруваат и да се поправаат доколку има некоја грешка или нелогичност при нивно извршување.

Библиотеките коишто го сочинуваат компајлерот сепак се дел од .NET Core рамката иако не зависат од нејзините функционалности, а со тоа се и дел од проектот што го градиме. Во претходните .NET Framework верзии овие библиотеки не беа дел од рамката, туку се додаваа во проектот како NuGet пакети со што се олеснува нивното ажурирање, но во .NET Core рамката тие се составен дел и секогаш се користат истите верзии од овие библиотеки во секој проект што ја користи оваа рамка, па со тоа се одржува конзистентност на нивното верзионирање. Најновата верзија на Roslyn компајлерот ја поддржува јазикот C# верзија 7.0.



**Слика бр. 4 Пример за визуелизација на синтаксички анализиран код**

При компајлирање на .NET проектите се повикува MSBuild алатката којашто го гради решението според стратегијата најдена во конфигурациска датотека, но секоја C# датотека се компајлира посебно со помош на csc.exe како повикувачки команден интерфејс, односно C# компајлерот Roslyn кој доаѓа заедно со .NET рамката со соодветна верзија вклучена во оперативниот систем. При извршување на апликацијата понекогаш се јавува потреба да се интерпретираат датотеки во реално време (пример razor датотеки), при што тогаш се користи истиот Roslyn компајлер, но се повикува по потреба од страна на самата апликација при нејзино извршување.

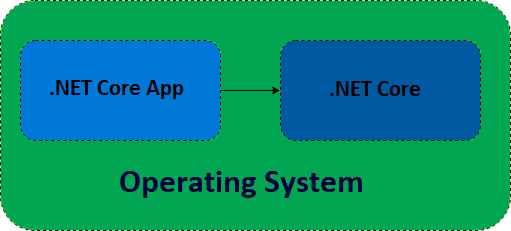
## Алатка за градење софтвер – MSBuild

MSBuild (Microsoft Build Tools) е алатка за создавање, компајлирање, тестирање, пакување и прикачување на апликации и генерирање на документација. MSBuild е независен од Visual Studio, но Visual Studio го користи MSBuild во позадина за извршување на своите задачи. Тоа значи дека доколку сакаме да изградиме некој продукт можеме да го направиме тоа без воопшто да користиме Visual Studio, туку само алатката MSBuild.

Во првичната алфа верзија на рамката .NET Core се отфрли користењето на конфигурацискаta датотека .csproj со .xproj и project.json, со што се раздели стратегијата за компајлирање на проектот во датотеката .xproj и листата на NuGet пакетите и верзиите на .NET рамката којашто се користи во датотеката project.json. Оваа промена не траела долго затоа што тимот од Microsoft што ја изработува рамката .NET Core одлучил во верзијата 1.1.1 да се врати на старата структура на конфигурирање со .csproj датотека.

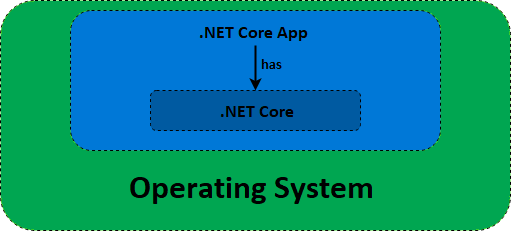
.NET Core апликациите може да се спакуваат на два различни начина, едниот начин е да се искористи инсталираната .NET рамка и да се извршува со нејзина помош, а вториот начин е да се вклучи .NET рамката како дел од проектот при што иако корисниот ја нема инсталирано на својот компјутер сепак ќе може да ја изврши апликацијата без никаков проблем.

Првиот начин на користење на .NET Core се нарекува изградба зависна од рамка (анг. Framework-dependent deployments - FDD) при што единствен параметар е името на рамката од којашто зависи проектот, пример aspcore1.1. Овој начин на градење на апликацијата има предност затоа што апликацијата може да се користи на било кој оперативен систем којшто ја има инсталирано .NET Core рамката без повторно да ја градиме апликацијата.



**Слика бр. 5 Апликација зависна од рамка**

Вториот начин се нарекува самостојно градење (анг. Self-contained deployments - SCD) и притоа се предава параметар што означува која верзија на извршување на рамката се користи, пример доколку сакаме да ја изградиме апликацијата за Windows 10 x64 мораме да предадеме параметар win10-64. Овој начин нуди можност да може да се извршува оваа апликација на било кој компјутер со соодветниот оперативен систем без разлика дали ја има инсталирано .NET Core рамката затоа што целата библиотека е вградена во самата апликација.



**Слика бр. 6 Апликација со вградена рамка**

## Алатки за командна линија – dotnet

Една од поголемите новости во .NET Core е алатката за командна линија (.NET Core CLI) којашто служи за управување со животниот циклус на една апликација во развој. Во претходните верзии на .NET рамката оваа задача ја имаше околината DNX (Dot Net eXecution).

DNX се состои од повеќе компоненти коишто сега се заменети или надградени за полесна употреба. Иако примарната употреба на DNX била за изградба на ASP.NET MVC 5 апликации, послужи како база за изградба на .NET Core CLI. Во стржта DNX ги содржи целата .NET Framework рамка, команден интерфејс за интеракција со DNX и апликацијата што се гради со негова помош, менаџмент на NuGet библиотеки и околина за извршување.

Со .NET Core CLI веќе нема потреба да се користи .NET Core библиотеката локално затоа што таа сега е NuGet библиотека којашто се симнува по потреба. Околината за извршување на .NET Core апликации самостојно е способна да се користи на повеќе платформи и е дел од .NET Core рамката, па поради тоа .NET Core CLI нема потреба да ја содржи оваа околина.

Базичните команди коишто можат да се искористат се:

* new
* restore
* build
* publish
* run

Командата “new” служи за креирање нов проект од типот наведен како следен аргумент. Синтаксата за искористување на оваа команда е dotnet new <Template> каде што <Template> може да се замени со console, classlib, mstest, xunit, web, mvc, webapi, nugetconfig, webconfig, sln. Оваа команда креира нов проект од соодветниот тип и ги иницијализира соодветните NuGet библиотеки.

Командата “restore” ги симнува од NuGet репозиториумот соодветните NuGet библиотеки наведени како зависности во проектот. Оваа команда мора да се изврши во главниот директориум на проектот за којшто сакаме да ги симнеме соодветните библиотеки.

Командата “build” служи за градење и компајлирање на проектот и библиотеките од коишто зависи при што се повикува алатката MSBuild. Излезот од оваа команда содржи .dll датотеки за самата апликација и библиотеките и .pdb датотеки за дебагирање на апликацијата. Оваа команда има додатни параметри за специфицирање на профилот за градење и целната платформа.

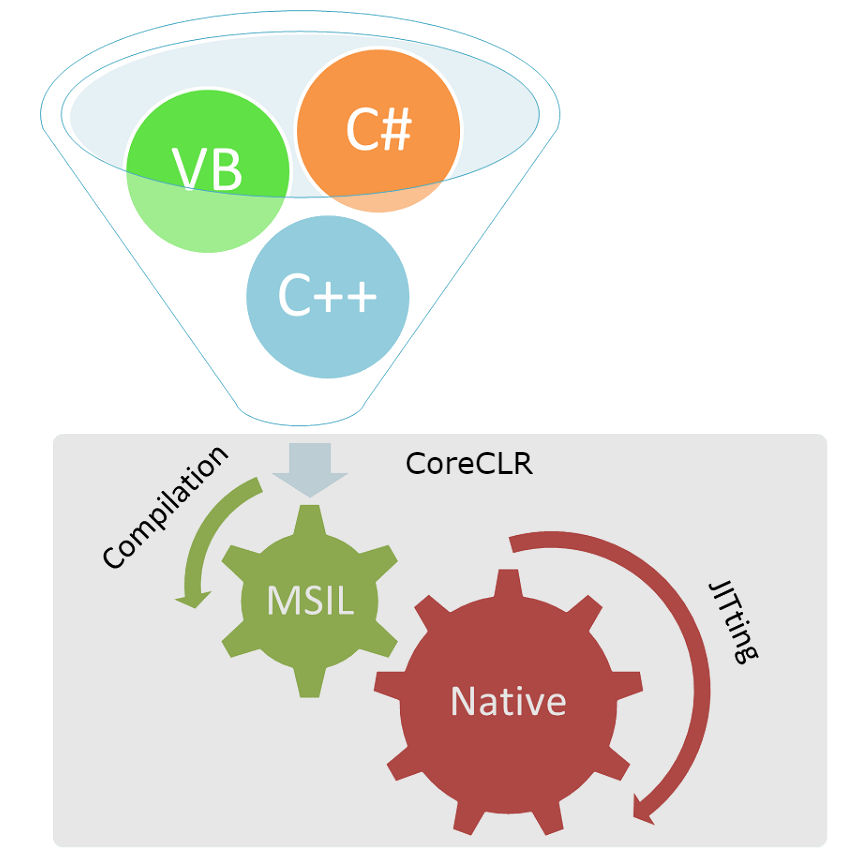
Командата “publish” е слична со командата “build” со таа разлика што резултатот не содржи датотеки за дебагирање.

Командата “run” служи за извршување на апликациите od типот апликации зависни од рамката(FDD) преку командна линија. За оваа команда може да се специфицираат верзија на рамка и датотека на конфигурација.

Во склоп на оваа алатка може да се користат и команди за менаџирање на миграции на бази на податоци, извршување на тестови и менаџирање нa NuGet пакети.

## Библиотеки што се користат при извршување на софтвер – CoreCLR

Како еден од најзначајните новости на .NET рамката е CoreCLR којашто претставува околина за извршување на средниот јазик CIL. Откако компајлерот Roslyn ќе го искомпајлира C# кодот во CIL и ќе создаде .exe и .dll датотеки, при извршување на апликацијата задачата на CoreCLR е да го преведе овој CIL код во машински код.



**Слика бр. 7 Пат на преведување и извршување на C#, VB и VS C++ код**

Во претходните верзии на .NET оваа околина за извршување апликации беше достапна само на Windows оперативниот систем или како дел од проектот Mono или DNX за други оперативни системи, но сега со .NET Core CLR може да се извршуваат апликации на Windows, Linux, OSx и FreeBSD оперативните системи и на x86, x64 и ARM архитектури на процесори.

CoreCLR во себе содржи компоненти за алокација на меморија, вчитување на класи, систем на типови, систем за менаџмент на процеси, имплементација на примитивни типови како броеви, букви и реченици. Освен овие базични компоненти, репозиториумот на CoreCLR содржи и компоненти коишто се во блиско сродство, како:

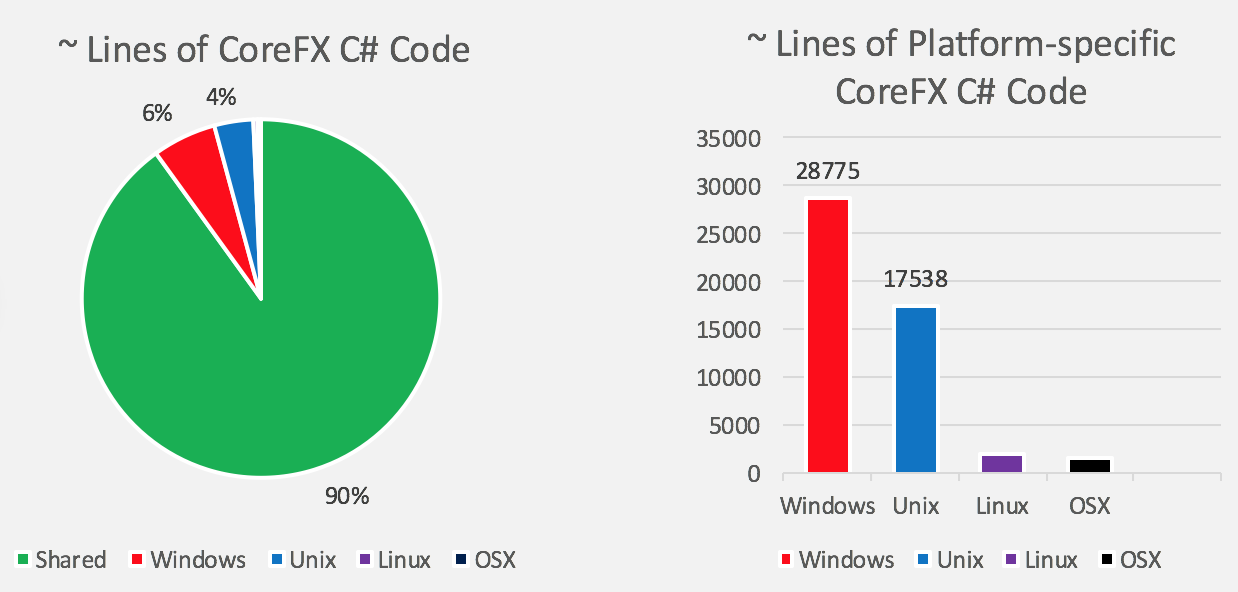
* компајлер во реално време (JIT) за преведување на код во среден јазик во машински код
* асемблер и дисасемблер за средниот јазик
* обвивка за процесирање на динамички поврзани библиотеки
* јавна библиотека за развој врз CoreCLR

## Библиотеки – CoreFX

CoreFX претставува колекција на библиотеки коишто припаѓаат на .NET Core рамката. Секој директориум којшто почнува со Microsoft.\* или System.\* во “src” директориумот претставува имплементација на библиотека. Секоја библиотека има “src” директориум каде што се наоѓа имплементацијата на соодветната библиотека и “ref” директориум каде што се наоѓа дефиницијата за таа библиотека.

Некои од библиотеките имплементираат базични функционалности па поради тоа нивниот изворен код е дел од CoreCLR репозиториумот, а во CoreFX се наоѓа само нивната дефиниција. При пристап на овие базични библиотеки се повикува имплементацијата дефинирана во System.Private.Corelib асемблито во CoreCLR библиотеката.

Како дел од CoreFX иако работата на најголемиот број на библиотеки не зависи од оперативниот систем, може да забележиме дека има библиотеки коишто мора да зависат од оперативниот систем на којшто би се извршувале. Ваков пример се библиотеките за криптографија и библиотеките за работа со периферни уреди, па поради оваа причина тие се имплементирани засебно за секој оперативен систем. Исклучок на ова правило е библиотеката за работа со регистарот на податоци којшто постои исклучиво само на Windows оперативниот систем, па поради тоа постои имплементација само за овој оперативен систем.

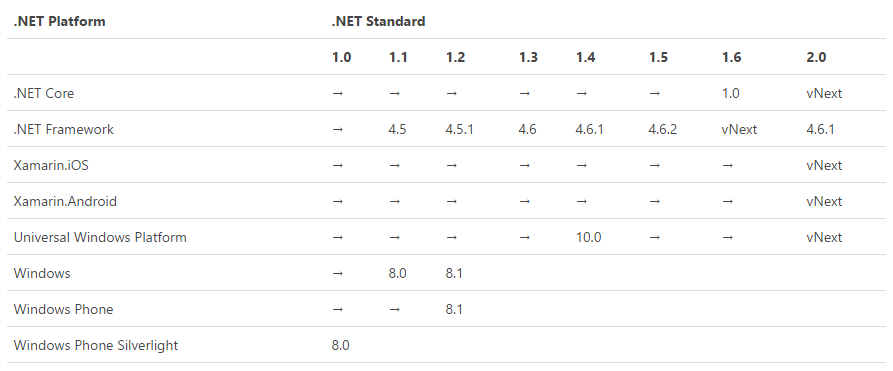


**Слика бр. 8 Број на линии код во CoreFX и нивна распределба по оперативни системи**

## .NET Стандард

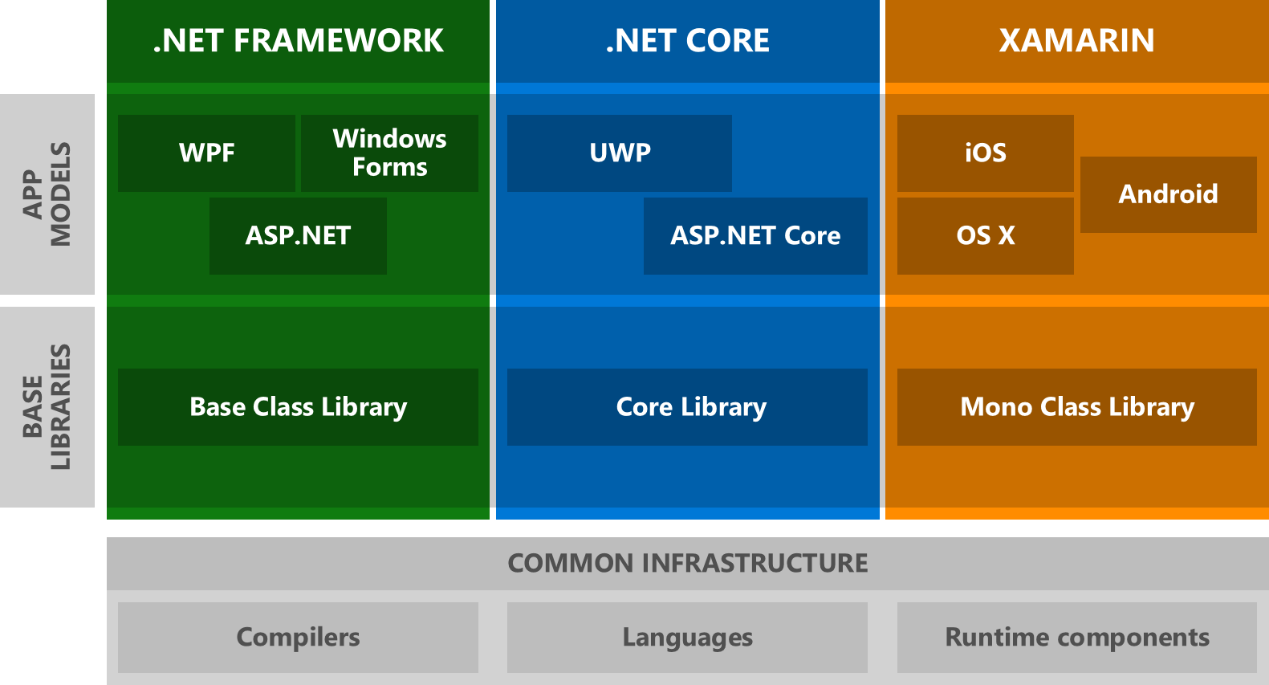
Со цел да може да се извршува .NET Core апликација на било кој оперативен систем, корисникот мора да се осигура дека .NET Core рамката е присутна на оперативниот систем. Оваа библиотека ги имплементира сите базични функционалности за секој оперативен систем, но постојат и функционалности коишто се специфични за оперативниот систем на којшто се извршува, што значи дека е возможно корисникот да пристапува функционалност достапна само на Windows или на Linux. За да се минимизираат проблемите коишто може да се јават од оваа природа, Microsoft вовеле стандард наречен .NET Standard којшто всушност претставува колекција на библиотеки коишто гарантирано се достапни на секој оперативен систем што има инсталирано соодветна верзија на .NET рамката.

.NET Core 1.0 верзијата се води според .NET Standard 1.6, што значи дека секој оперативен систем којшто ја има инсталирано .NET Core 1.0 рамката ќе може да извршува апликации, односно библиотеки што ги користат апликациите изградени според .NET Standard 1.6.



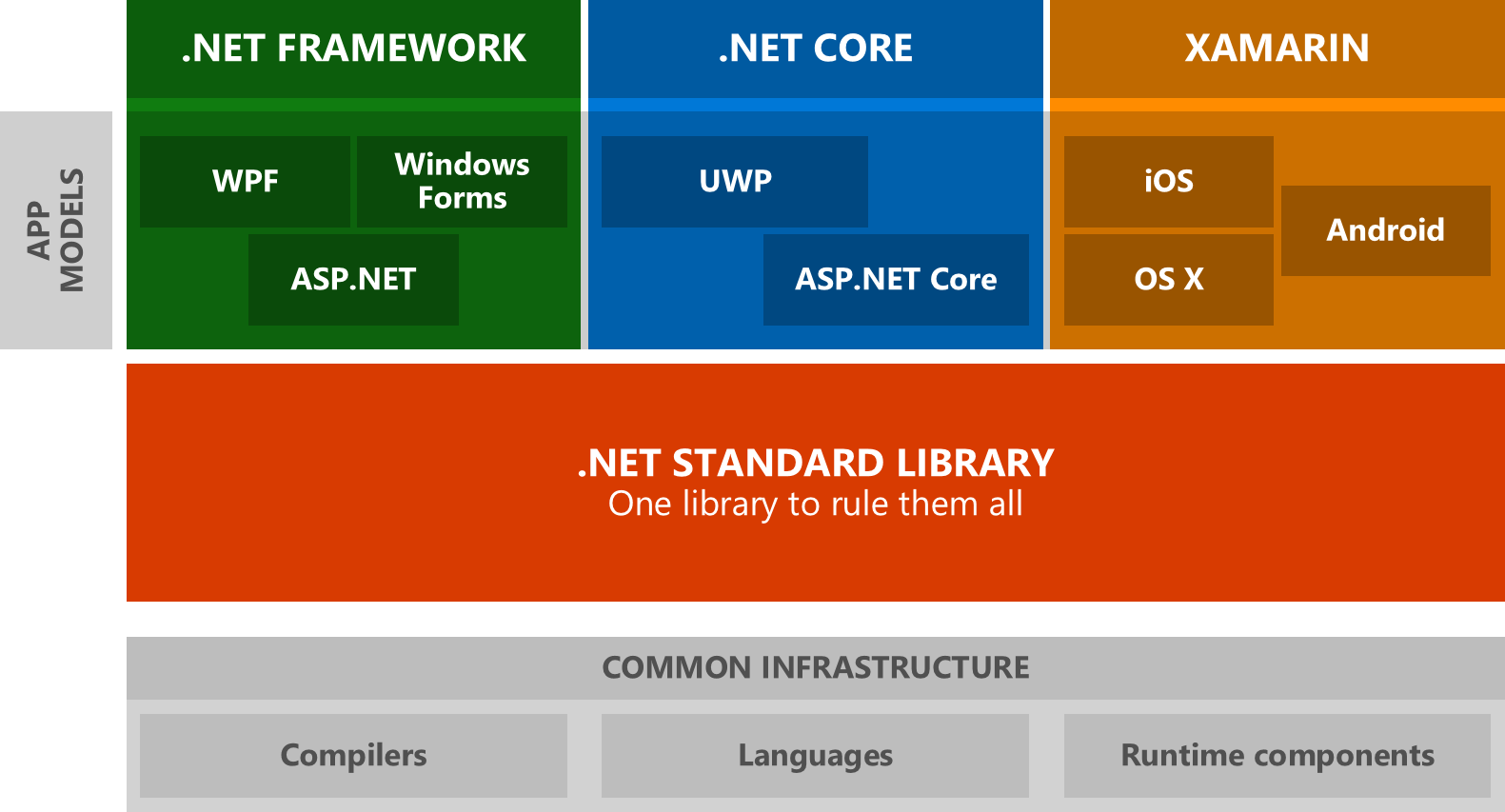
**Слика бр. 9 Поддржани верзии на .NET Standard**

Постојат три официјално поддржани рамки за развој на апликации од различна природа: .NET Framework, .NET Core и Xamarin. Доколку .NET Standard не постои тогаш секоја од овие рамки би имала различна колекција на базични библиотеки при што би се јавил проблем за интеграција на сопствена библиотека во сите три рамки одеднаш. Овој проблем бил решен со изградба на портабилни класни библиотеки коишто можат да се извршуваат на сите рамки без проблеми, но тогаш само пресекот на функционалностите што ги нудат рамките би биле достапни за развивачите на сопствената библиотека. За да се унифицираат функционалностите што ги нудат сите рамки, тие мора да се водат според стандард на заеднички јавни повици .NET Standard.



**Слика бр. 10 Проблематичен пристап за работа доколку не постои .NET Standard**

Овој стандард е компатибилен со своите постари верзии, што значи дека доколку корисникот има инсталирано .NET Core 1.0 верзија на својата машина, тој може да интегрира библиотеки изградени со било која верзија на .NET Standard помала или еднаква на верзијата 1.6. Овој начин на имплементација на оваа колекција на библиотеки го елиминира проблемот кој се јавува при надградба на верзијата на .NET рамката на соодветната машина.



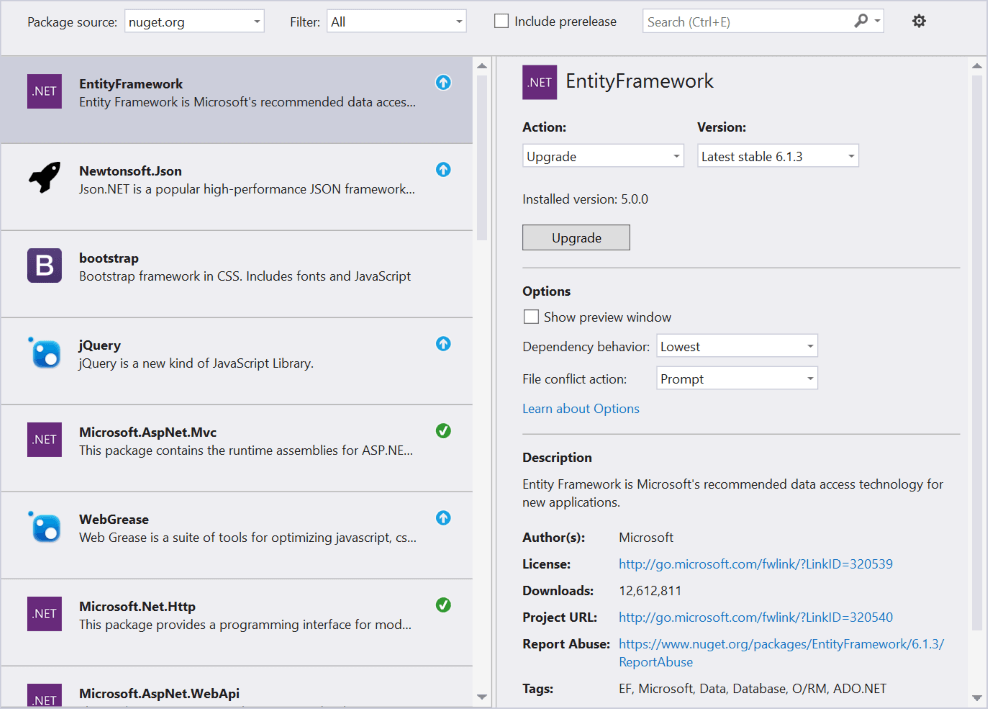
**Слика бр. 11 Архитектура на .NET рамките**

## Менаџер на пакети – NuGet

При развој на една .NET апликација потребно е да се користат најразлични библиотеки кои решаваат чести проблеми, за полесен пристап до овие библиотеки развивачите на .NET платформата нудат менаџер за пакети наречен NuGet.

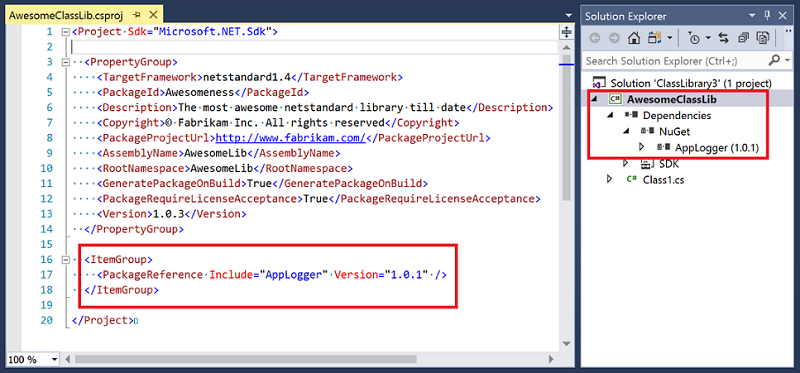
Најосновната функција на NuGet е пребарување и консумирање на библиотеки наречени NuGet пакети, но исто така и менаџмент на верзиите на веќе спуштените пакети, менаџмент на целната верзија на .NET рамката, нивно консолидирање низ повеќе проекти и градење на наши сопствени пакети за глобална консумација.

Пред официјалното пуштање во употреба на Visual Studio 2017, NuGet менаџерот бил посебен продукт интегриран во околината за развој. При спуштање на NuGet пакети, информациите за верзијата на целната .NET рамка, името и верзијата на NuGet пакетот се запишувал во packages.config датотека.



**Слика бр. 12 Кориснички интерфејс на NuGet менаџерот**

Новата верзија на NuGet 4.0 менаџерот веќе не е посебен продукт, туку е интегриран како дел на MSBuild со што ја снемува потребата за packages.config датотеката, затоа што сега пакетите се дефинираат во .csproj датотеката како PackageReference запис. Покрај оваа промена подобрени се времето на симнување на пакети и приказот на пакетите од кои зависи нашиот проект со тоа што не се симнуваат повеќе пати при промена на верзијата на пакетот и не се прикажуваат пакетите коишто не се од директна корист за проект што се гради.



**Слика бр. 13 Нов начин на запишување на референци кон пакети**

## Рамка за развивање веб апликации - ASP.NET Core

ASP.NET е платформа со отворен изворен код наменет за изработка на модерни веб апликации и сервиси со помош на C#, HTML5, CSS и JavaScript коишто можат да опслужат милиони корисници.

ASP.NET Core е новата генерација на ASP.NET платформата којашто има многубројни новини во својата структура. За разлика од својот претходник ASP.NET MVC 5 којшто се извршува само со .NET Framework рамката, оваа платформа може да се извршува и со .NET Core рамката, со што претставува прва веб платформа официјално поддржана од Microsoft којашто може да се хостира на Linux и macOS сервер.

Доколку направиме споредба со ASP.NET MVC 5 платформата можеме да забележиме дека новата платформа има вклучен сервер Kestrel во самиот процес на апликацијата и лесно може да се интегрира со IIS сервер, додека старата платформа има тесна интеграција со IIS сервер. Ова претставува проблем затоа што со MVC 5 развивачите се принудени да ја хостираат апликацијата на IIS сервер и Windows оперативен систем.

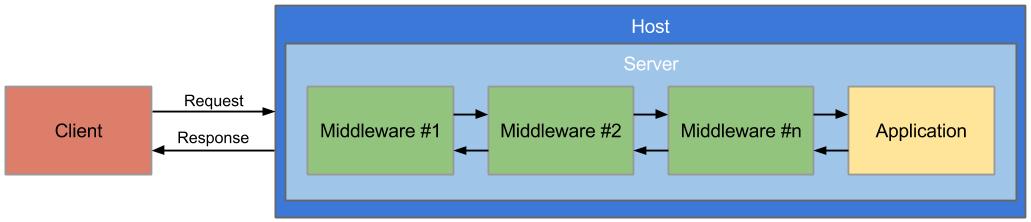
За разлика од ASP.NET MVC 5, ASP.NET Core го имплементира OWIN стандардот во своја имплементација на слоевита архитектура, но нуди и лесна интеграција со базичната OWIN библиотека.

Доколку сакаме да имплементираме веб апи со ASP.NET Core не мораме да имплементираме ново решение и да го хостираме како засебна веб страна туку ASP.NET Core претставува унификација на двата концепти веб апликација и веб апи.

### OWIN

OWIN преставува отворен веб стандард за .NET. Целта на OWIN е да ги раздвои целините сервер и апликација со тоа што ќе овозможи комуникација меѓу двете преку стандарден интерфејс. Во пракса за развивачите на софтвер OWIN игра голема улога при изградба на библиотеки коишто имаат интеракција со корисничките барања до веб апликација.

OWIN претставува платформа за изградба на апликација со слоевита архитектура така што секој слој има контрола на целиот контекст на корисничкото барање и може да одлучи дали има потреба да се повика наредниот слој.



**Слика бр. 14 Слоевита архитектура на апликација со OWIN**

Пример доколку сакаме да го запишуваме секое корисничко барање со IP адресата и времето на барање, можеме да изградиме OWIN слој којшто ќе ја извршува оваа функција. Добрата страна на овој пристап е што оваа функционалност може да ја искористиме на секоја платформа што го имплементира OWIN стандардот, како што се ASP.NET MVC 5, ASP.NET Core или базичната OWIN библиотеката со сопствен веб сервер.

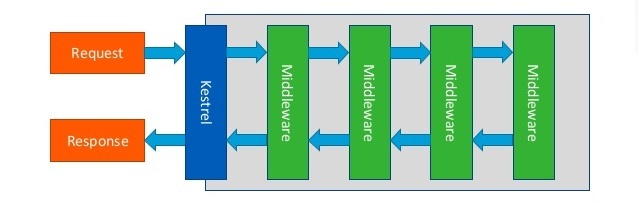
Доколку направиме споредба помеѓу ASP.NET MVC 5, ASP.NET WebApi 2 и ASP.NET Core можеме да забележиме дека иако сите го имплементираат OWIN стандардот, разликата во неговата интеграција е тоа што:

* ASP.NET MVC 5 има интеграција за прифаќање на кориснички барања со IIS серверот, а OWIN се повикува при прифаќање на секое корисничко барање
* ASP.NET Core го имплементира OWIN стандардот и користи слоевита архитектура, но не ја користи базичната OWIN библиотека туку има можност за адаптирање кон истата
* ASP.NET WebApi2 ја користи базичната OWIN библиотека и самата платформа претставува слој кон таа библиотека со што се нуди можност ASP.NET WebApi 2 да се хостира како конзолна апликација со OWIN веб сервер

Иако имплементацијата на OWIN библиотеката е различна во сите три платформи, користење на сопствена библиотека што претставува OWIN слој е возможна во секоја од овие платформи.

### Kestrel

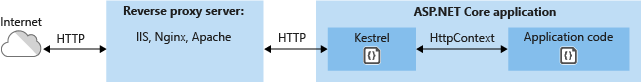
При развој на веб апликација потребен е сервер за хостирање за да се овозможи јавен пристап на веб апликацијата до потенцијалните клиенти. При хостинг на стандардна ASP.NET апликација мора да се користи IIS (анг. Internet Information Services) сервер стриктно на Windows оперативен систем. Овој проблем се јавува затоа што самата ASP.NET платформа не знае како да се справи со било кој формат на клиентско барање туку само со барање опслужено од IIS сервер. За да се реши овој проблем би можела да се адаптира платформата на повеќе веб сервери, но сепак тоа би претставувал макотрпен процес поради големиот број на различни сервери. При хостирање на ASP.NET Core апликација ова ограничување не постои токму заради Kestrel.



**Слика бр. 15 Kestrel и OWIN во ASP.NET Core апликација**

Kestrel претставува веб сервер извршуван во процесот на самата ASP.NET Core веб апликација, базиран на асинхроната библиотека libuv за процесирање на I/O барања компатибилна со повеќе оперативни системи. ASP.NET Core платформата знае како да се справи со клиентски барања опслужени од Kestrel со што може веб апликацијата да се постави на било кој оперативен систем што поддржува .NET Core апликации. Kestrel знае како да се справи со HTTP/S и WebSocket барања и е оптимизиран за работа со Unix сокети и Nginx серверот.

Тимот на Microsoft препорачува да не се користи Kestrel серверот за јавен пристап до веб апликацијата туку да се користи некој друг обратен прокси сервер којшто ќе ги пренасочува сите клиентски HTTP/S барања до Kestrel. За разлика од класичните веб сервери Kestrel не нуди заштита од напади од типот на премногу барања од еден клиент, преголеми барања или преголеми датотеки, менаџмент и лимитирање на достапни процесорски нишки. Токму поради овие проблеми се препорачува некој друг јавен сервер кој може да се справи со овие проблеми и ги пренасочува само валидните барања до Kestrel серверот.



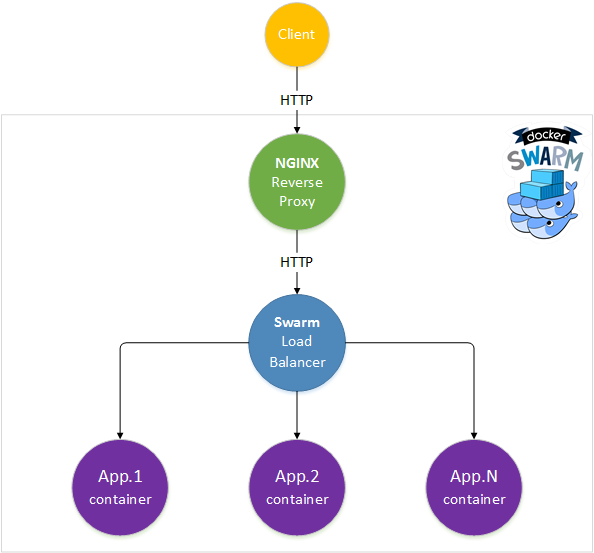
**Слика бр.. 16 Архитектура на хостинг со обратен прокси сервер**

### Docker

Многу често при развој на апликации се случува одредена функционалност којашто зависи од некој сервис да работи на еден компјутер, но да не работи на друг. Овие проблеми најчесто се јавуваат поради различна верзија или конфигурација на сервисот од кој што зависи одредената функционалност.

Едно решение за да се елиминира овој проблем е да се доставуваат сервисите и нивната конфигурација заедно со апликацијата. Docker претставува контејнер кој ја овозможува оваа практика за се извршува без помош на виртуелна машина, освен на Windows оперативен систем каде што е потреба Hyper-V виртуелна машина. За разлика од виртуелните машини, контејнерите не доаѓаат заедно со цел оперативен систем, туку само со библиотеки коишто ја изолираат апликацијата од оперативниот систем и ги менаџираат ресурсите потребни за да работи правилно.

Docker е поддржан на повеќе оперативни системи од кои Windows, повеќето дистрибуции на Linux и macOS се најпопуларни, но исто така имаат посебни интеграции со AWS и Azure сервисите.



**Слика бр. 17 Апликации хостирани во Docker контејнери**

ASP.NET Core има можност да се извршува преку Docker контејнер со што може да се намалат проблемите при развој на веб апликација на различни оперативни системи. Освен самата апликација може да се дефинираат и база на податоци, сервис за кеширање, обратен прокси сервер и други потребни сервиси за успешна изградба на апликацијата, притоа развивачите не мора да се грижат за оперативниот систем на којшто се извршуваат сите овие сервиси и самата веб апликација.

# Интегрирана околина за развој - Visual Studio 2017

# Пример апликација и разлика при развој со .NET Framework 4.6.2 и .NET Core 1.0

# Заклучок

# Референци