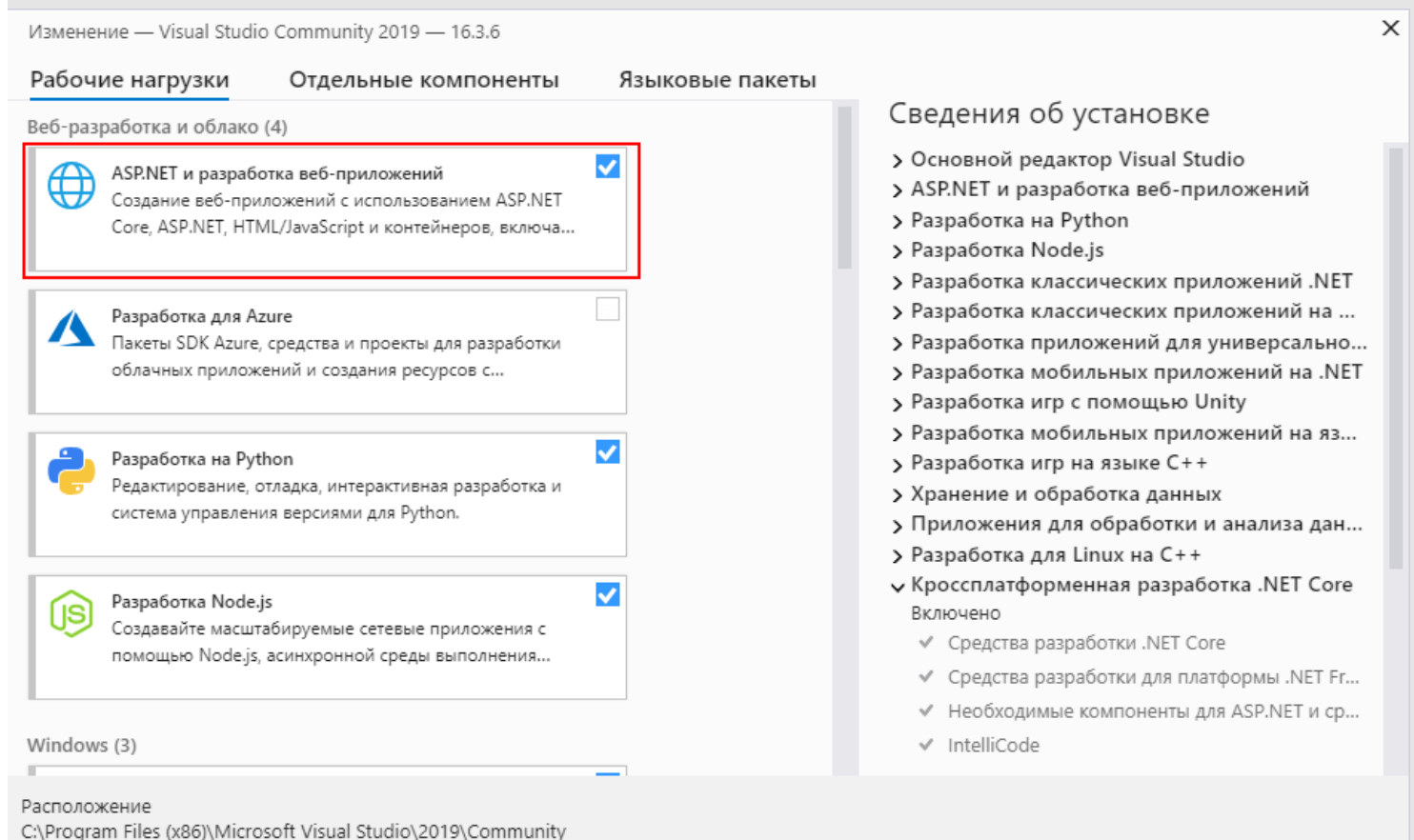


# Начало работы с ASP.NET Core

Для разработки под ASP.NET Core мы можем использовать различный инструментарий. Если нашей рабочей платформой является Windows, то мы можем использовать полнофункциональную среду разработки Visual Studio. Если мы разрабатываем на Mac OS или Linux, то можем использовать расширенный редактор кода **Visual Studio Code**. Данный редактор также может работать и под Windows. В рамках данного руководства преимущественно будет использоваться среда Visual Studio 2019.

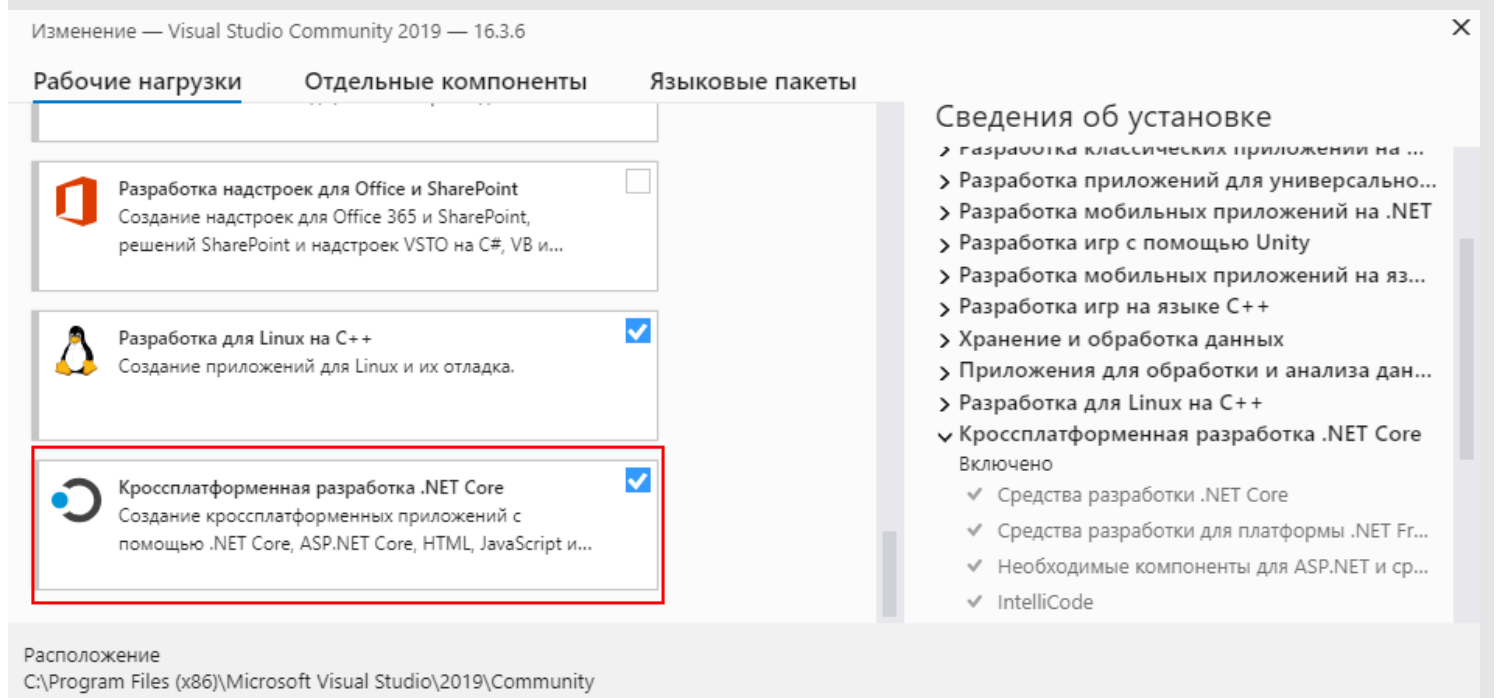
Программу для установки Visual Studio 2019 можно загрузить со страницы <https://www.visualstudio.com/downloads/>. В данном случае не важно, какой выпуск VS использовать - бесплатный Community или платные Professional или Enterprise. Все эти выпуски имеют встроенные средства для создания приложений на ASP.NET Core. В рамках этого руководства будет использоваться бесплатный выпуск VS 2019 Community.

Итак, загрузим установщик VS 2019 и запустим его. Вначале нам предлагается установить ряд опций. И так как мы будем работать с ASP.NET Core, то выбрать в программе для установке пункт **ASP.NET и разработка веб-приложений**:



Также при выборе этого пункта в поле справа можно выбрать также необязательные компоненты, которые будут устанавливаться вместе с ASP.NET. Можно выбрать все компоненты.

Кроме того, чуть ниже также в программе установщика нам надо выбрать другой пункт **Кроссплатформенная разработка .NET Core**:



Отметив все необходимые нам опции, выполним установку Visual Studio.

После установки откроем Microsoft Visual Studio 2019 и при создании проекта выберем пункт **ASP.NET Core Web Application** - тип проекта для создания веб-приложения ASP.NET Core:

# Create a new project

Search for templates (Alt+S)



[Clear all](#)

C#

All platforms

All project types



## Console App (.NET Core)

A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.

C# Linux macOS Windows Console



## ASP.NET Core Web Application

Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.

C# Linux macOS Windows Cloud Service Web



## Blazor App

Project templates for creating Blazor apps that run on the server in an ASP.NET Core app or in the browser on WebAssembly (wasm). These templates can be used to build web apps with rich dynamic user interfaces (UIs).

[Next](#)

На следующем шаге дадим какое-нибудь имя проекту, например, HelloApp, и определим для него местоположение на жестком диске:

# Configure your new project

ASP.NET Core Web Application

C#

Linux

macOS

Windows

Cloud

Service

Web


Project name

HelloApp

Location

C:\Users\Eugene\Source\Repos\ASPNET\

...

Solution name 

HelloApp

☐

Place solution and project in the same directory

Back


Create


После этого отобразится окно выбора шаблона нового приложения:


# Create a new ASP.NET Core web application


.NET Core


ASP.NET Core 5.0


**ASP.NET Core Empty**  
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

**ASP.NET Core Web API**  
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

**ASP.NET Core Web App**  
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

**ASP.NET Core Web App (Model-View-Controller)**  
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

**ASP.NET Core with Angular**  
A project template for creating an ASP.NET Core application with Angular

**ASP.NET Core with React.js**

Authentication

No Authentication

Change

Advanced

☒ Configure for HTTPS

☐ Enable Docker Support  
(Requires [Docker Desktop](#))

Linux

Author: Microsoft  
Source: Templates 5.0.1

Get additional project templates

Back

Create

Здесь нам доступно несколько типов проектов:

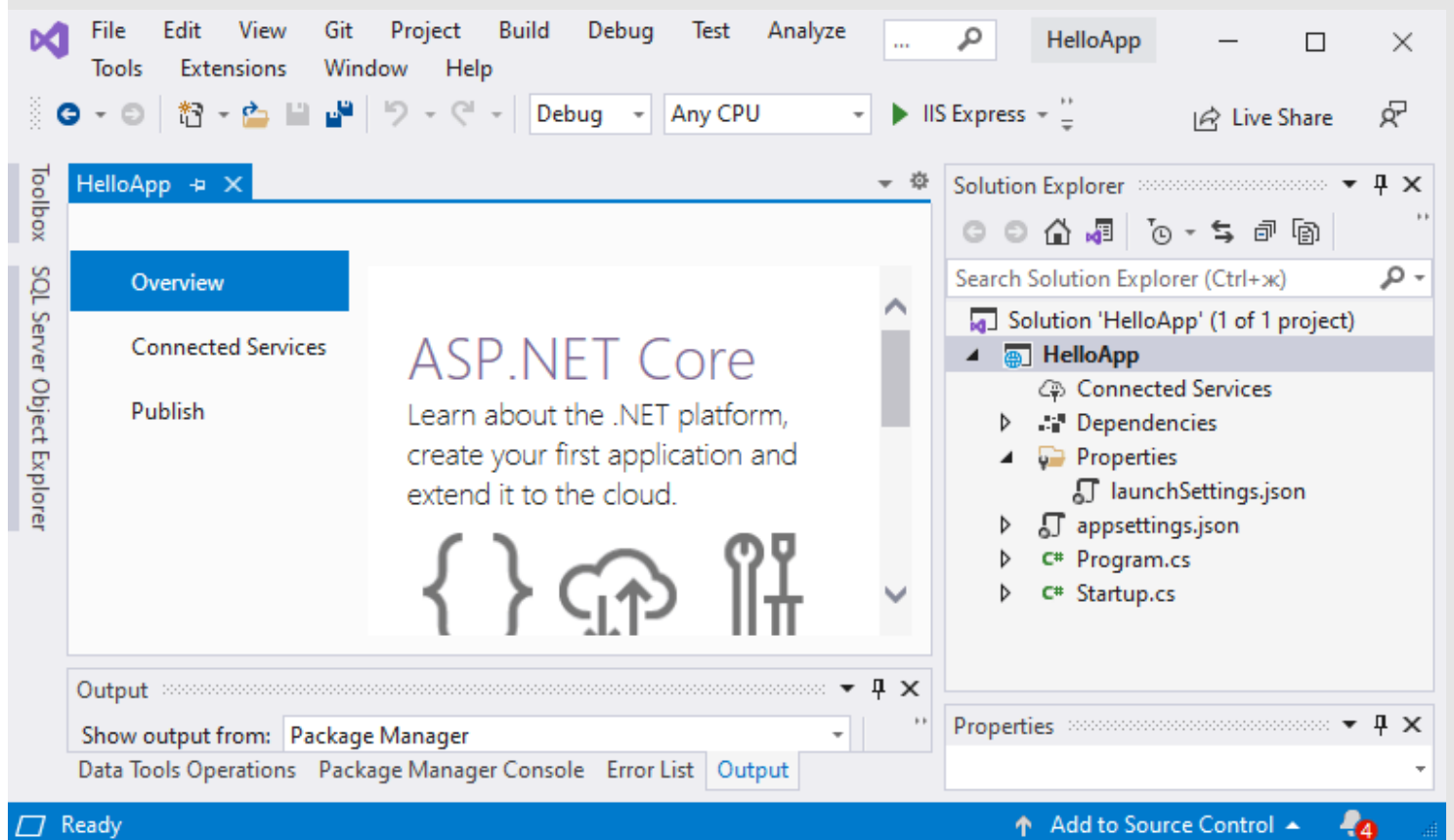
- **ASP.NET Core Empty:** пустой шаблон с самой минимальной функциональностью для создания приложений с нуля
- **ASP.NET Core Web API:** проект веб-приложения, который использует архитектуру REST для создания веб-сервиса
- **ASP.NET Core Web App:** проект, который для обработки запросов по умолчанию использует Razor Pages
- **ASP.NET Core Web App(Model-View-Controller):** проект, который использует архитектуру MVC
- **ASP.NET Core with Angular:** проект, предназначенный специально для работы с Angular 2+.
- **ASP.NET Core with React.js:** проект, который использует React.JS
- **ASP.NET Core with React.js and Redux:** проект, который использует React.JS и Redux

Кроме того, здесь мы можем указать версию ASP.NET Core в выпадающем списке, но в данном случае оставим значение по умолчанию - ASP.NET Core 5.0.

Также здесь можно указать тип аутентификации, который по умолчанию используется в проекте, и подключить контейнер Docker.

Также здесь есть флажок "Configure for HTTPS". При установке этого флажка проект при отладке и тестировании по умолчанию будет запускаться по протоколу HTTPS. В данном случае установка и неустановка этого флажка не имеет значения. Кроме того, даже если мы установили эту отметку, то впоследствии через свойства проекта можно отменить запуск через HTTPS или, наоборот, заново установить.

Среди этих шаблонов выберем **ASP.NET Core Empty**, все остальные значения оставим по умолчанию и нажмем на кнопку OK. И Visual Studio создает новый проект:



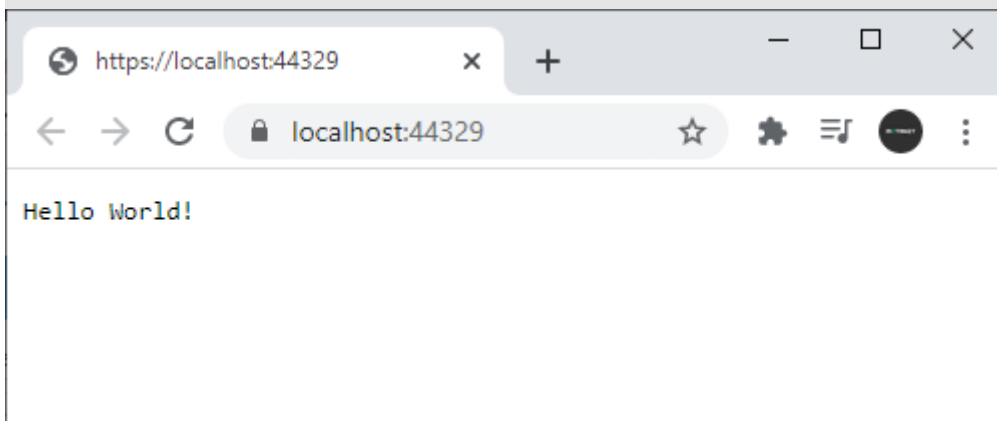
## Структура проекта ASP.NET Core

Рассмотрим базовую структуру стандартного проекта ASP.NET Core. Проект **ASP.NET Core Empty** содержит очень простую структуру - необходимый минимум для запуска приложения:

- **Connected Services:** подключенные сервисы из Azure
- **Dependencies:** все добавленные в проект пакеты и библиотеки, иначе говоря зависимости

- **Properties:** узел, который содержит некоторые настройки проекта. В частности, в файле `launchSettings.json` описаны настройки запуска проекта, например, адреса, по которым будет запускаться приложение.
- **appsettings.json:** файл конфигурации проекта в формате json
- **Program.cs:** главный файл приложения, с которого и начинается его выполнение. Код этого файла настраивает и запускает веб-хост, в рамках которого разворачивается приложение
- **Startup.cs:** файл, который определяет класс `Startup` и который содержит логику обработки входящих запросов

Данная структура, конечно, не представляет проект полнофункционального приложения. И если мы запустим проект, то в браузере увидим только строку "Hello World!", которая отправляется в ответ клиенту с помощью класса `Startup`:



При создании других типов проектов ASP.NET структура будет отличаться, соответственно начальный проект будет иметь больше функционала, однако это тот каркас, от которого мы можем отталкиваться, добавляя в него какие-то свои файлы и папки.

