PrepAway

Developing Solutions for Microsoft Azure

Study Guide

Exam AZ-204

Azure for .NET Core Developers

Preface

Every developer is striving hard to have a skill upgrade from mere a developer to a Cloud developer. And with the growing pace of cloud programming, this upgradation is not simple. This book will help a developer, especially the one working with Microsoft technologies, to be specific, a .NET Core developer, to seamlessly cover this said journey.

Newly release .Net Core 3.0 / 3.1 including, Azure Function V3, which got available for Production use in January 2020, are among the technology stacks covered in this book. The book not only focuses on one way of working with Azure Cloud services but includes other popular and trending way of managing Azure resources with the software application. With focusing on ease of understanding the subject, many super cool features of Azure products and services are also amended to the learning course.

From exploring the most used Azure services to touching the newest version of offerings, this book is aimed to cover everything from a developer perspective. Code exercise, Code blocks, azure service implementation, application secrets keys management, free superfast hosting options along with live debugging of code hosted on Cloud, are some of the key takeaway from the book. Over the 7 chapters in this book, you will learn the following,

Chapter 1: This Chapter will present with hug level overview of Microsoft Azure, its components, its features, and its offerings. It will also present services every developer should know about. These services will include, further chapters topics as well, so as to reader should be well aware of what is coming in in further course content.

Chapter 2: This chapter will present with what is Azure App services all about. Again, a basic handshake with the Azure Web App service. Going further, it will cover the ways we can host our .NET ore MVC application to Azure Cloud – Azure Web App. Also, it will setup a basic CICD for the application using GitHub.

Chapter 3: This chapter will present with Azure CosmosDB introduction as a Database as a service model for .NET core application as backend. This will further present the implementation of Azure CosmosDB in the .NET core application.

Chapter 4: This chapter will present with designing and implementing Azure storage services in our .NET Core applications, how seamlessly it can be configured and managed.

Chapter 5: This chapter will present an amazing feature of Azure storage offered by Microsoft Azure for hosting the static contents. We will be working with the feature using Azure CLI. Also, we will learn about hosting the latest .NET Core application using Angular, ReactJS with Visual Studio 2019 as IDE.

Chapter 6: This chapter will present with amazing service of Azure on how to seamlessly secure your .NET core application secrets keys using Azure KeyVault and App service configuration.

Chapter 7: This chapter will present you with Azure serverless offerings capabilities. Azure Function will be introduced and detailed. The scenario of creating Thumbnails out of uploaded user pics will be presented here using Azure functions.

Table of Contents

CHAPTER 1

Azure Ecosystem

Firstly, big congratulations to you for selecting Azure in your career path and having your professional skills upgrading with the super cool, intelligent cloud, Microsoft Azure. From here to the end of this book, I will be your partner in the journey of enhancing your skillset from .NET technologies to NET Core with Microsoft Azure. In this chapter, I will give you an eagle view about Azure Ecosystem. What a developer should know about Azure along with the subject's exercise, which you will be learning in the coming chapters.

Structure

- Azure and its components
- Azure services
- Working with Azure
- ARM templates
- Azure CLI
- Azure PowerShell

Objective

The objective of this chapter is to understand the following:

- Services developer should know
- What we will cover at the end of the book
- Prerequisite and setup to start with

Azure and its components

Let us study Azure and its components in detail in the following sections.

What is Azure?

As Microsoft says, *Microsoft Azure is an ever-expanding set of cloud services to help your organization meet your business challenges. It is the freedom to build, manage, and deploy applications on a massive, global network using your favorite tools and frameworks.*

As mentioned, Azure does come with flexibility and ease to work with different operating systems such as Windows, Linux, with multiple language support such as C#, JAVA, Python, and so on. Along with different developers' tools such as Visual Studio, Visual Code, and so on. When it comes to the backend, it does support not only different databases such as SQL, MongoDB, Cassandra, PostgreSQL, but also different types of databases such as Relational, NoSQL, Graph, and so on. It comes with a wide range of scalable infrastructure offerings as well as serverless offerings. And same goes when it comes for pricing, with fixed monthly plans to per execution plan.

The reason behind this amazing Ecosystem is to empower you to have the liberty to go for the desired technology stack in terms of the operating system, compute database, language, tools, and pricing. You should focus more on working on solutions to meet the business goals, and the underline technology should never be constrained. Working with Microsoft Azure brings you in pleasure to work with world-class technology stack. In short, Azure is very much compatible with your designed stack.

As you may have knowledge of their different key areas in the software development process. Now, here I am more specifically, talking with the developer's perspective.

To list it:

1. Networking
2. Storage
3. Servers
4. Virtualization
5. Operating System
6. Middleware

7. Runtime
8. Data
9. Application

Considering the preceding nine points, let the responsibility of each area be shared as follows:

- **Infrastructure as a Service (IaaS):** Azure gains the responsibility or control for points 1 to 4 in the preceding list, and the rest are managed by you. To name, Azure `Virtual Machine`.

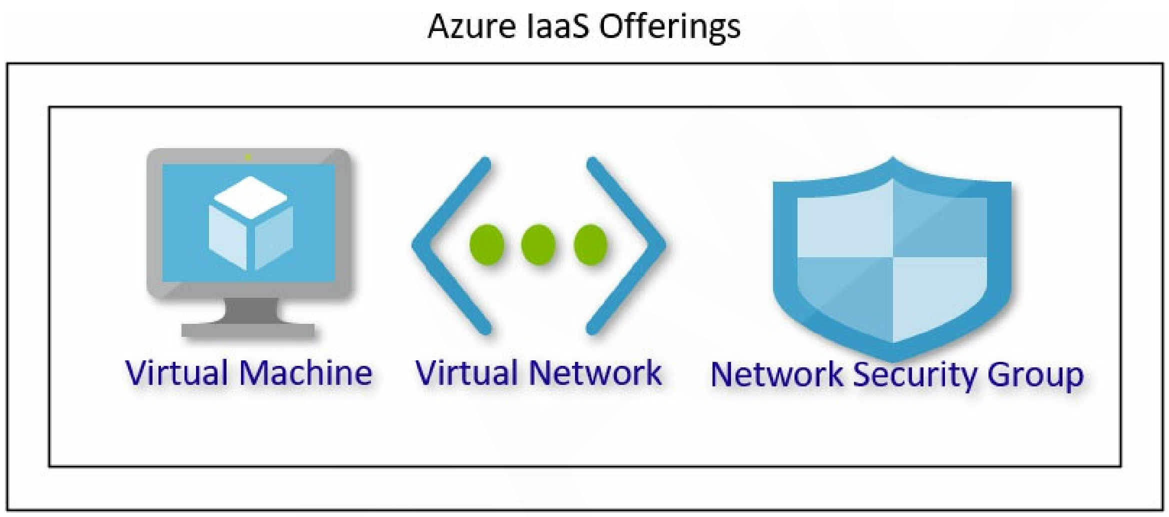


Figure 1.1

- **Platform as a Service (PaaS):** To name, Azure gains the responsibility or control for points 1 to 7 in the preceding list, and the rest are managed by you. To name, Azure Web App Service:

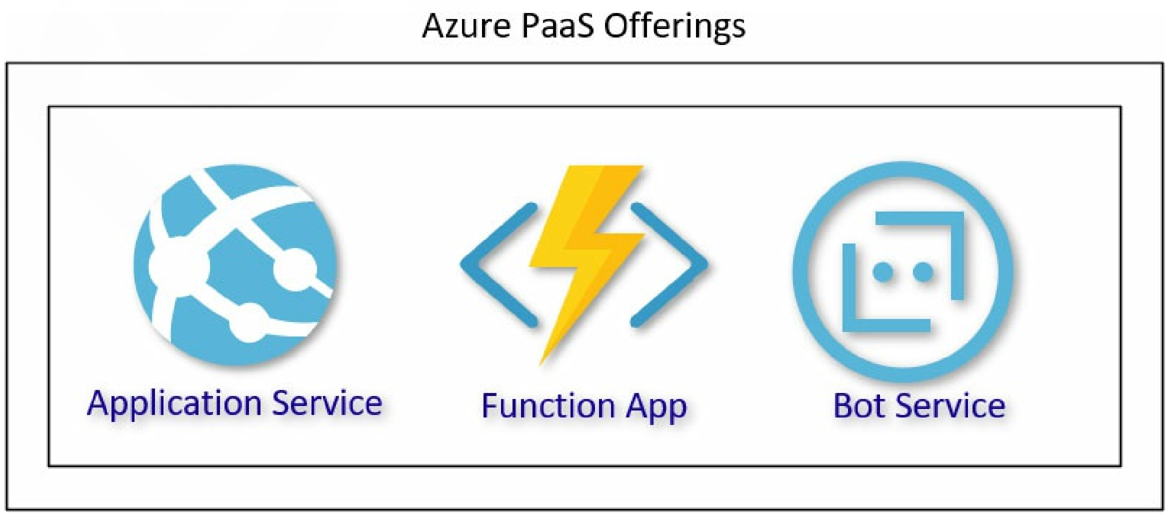


Figure 1.2

- **Software as Service (SaaS):** To name, Azure gains the responsibility or control for points 1 to 9 in the preceding list, and the rest are managed by you. To name, cloud-based programs available in Office 365, such as Microsoft Office Tools, Email.

Azure further categorizes into:

- **Integration Platform as a Service (IPaaS):** Services involved in integration architecture, to name, Azure **`Event Grid`**:

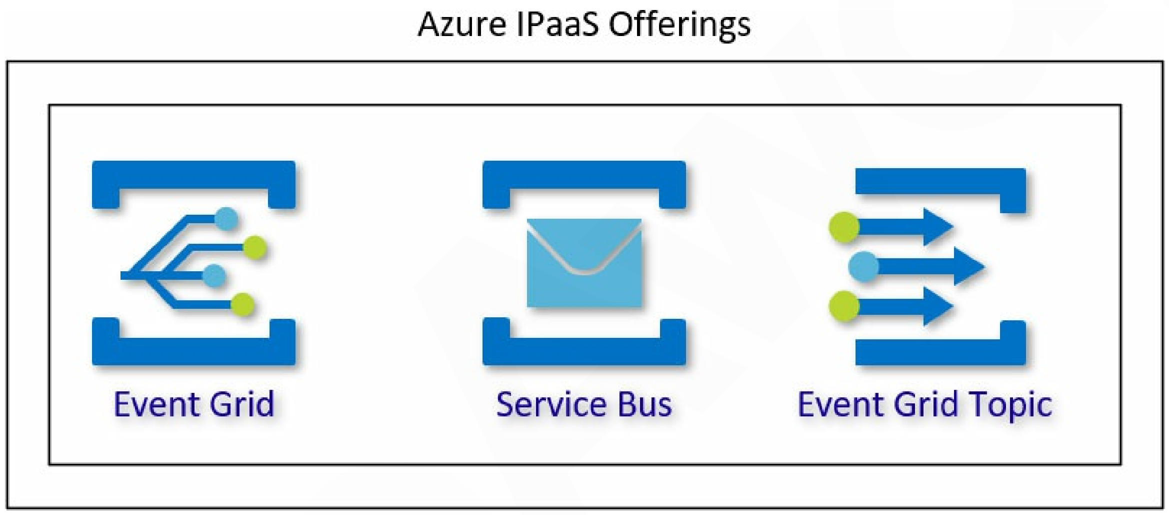


Figure 1.3

- **Desktop as a Service (DaaS):** To name, **`Windows Virtual Desktop`**:

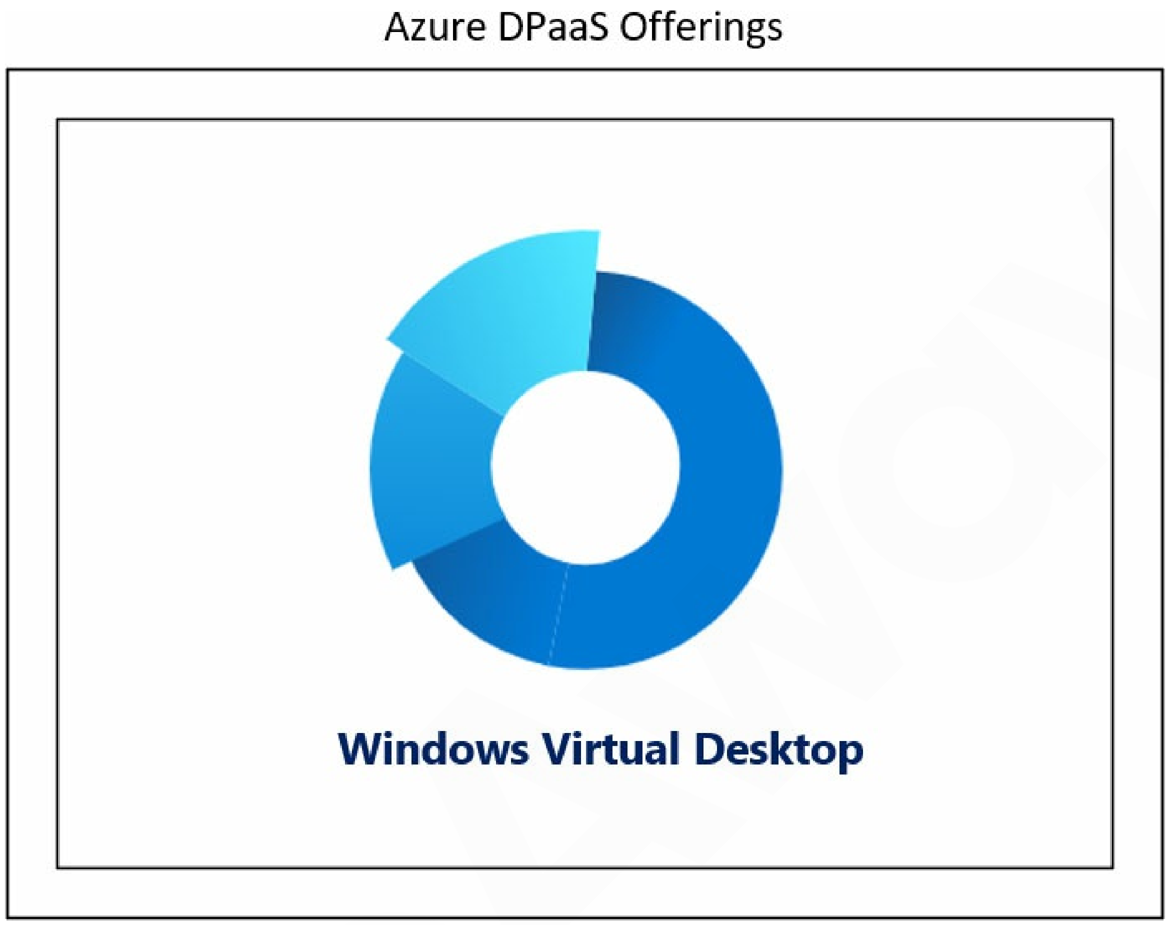


Figure 1.4

- **Database as a Service (DBaaS):** As the name suggests, offerings about serve backend. To name, Azure **`CosmosDB`**:

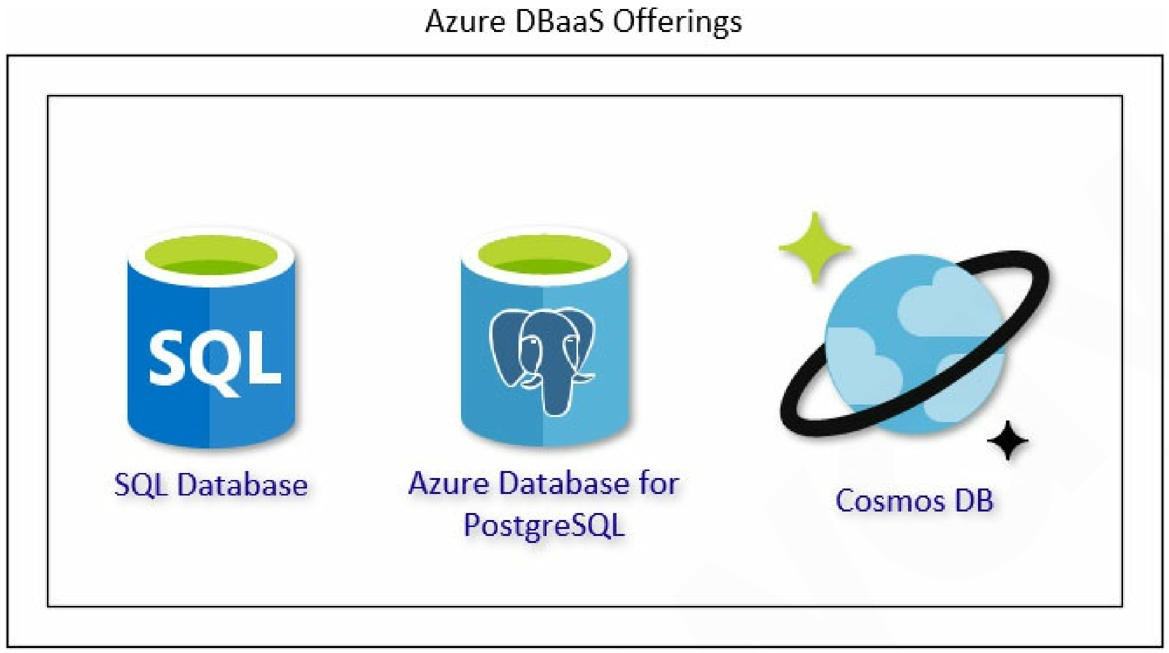


Figure 1.5

- **Blockchain as a Service (BaaS):** To name, **`Azure Blockchain Service`**, few to be listed among. This later got globally adopted by all other Cloud vendors in the race. And interestingly for all the categories of Cloud Computing, Microsoft Azure offers different services and products:

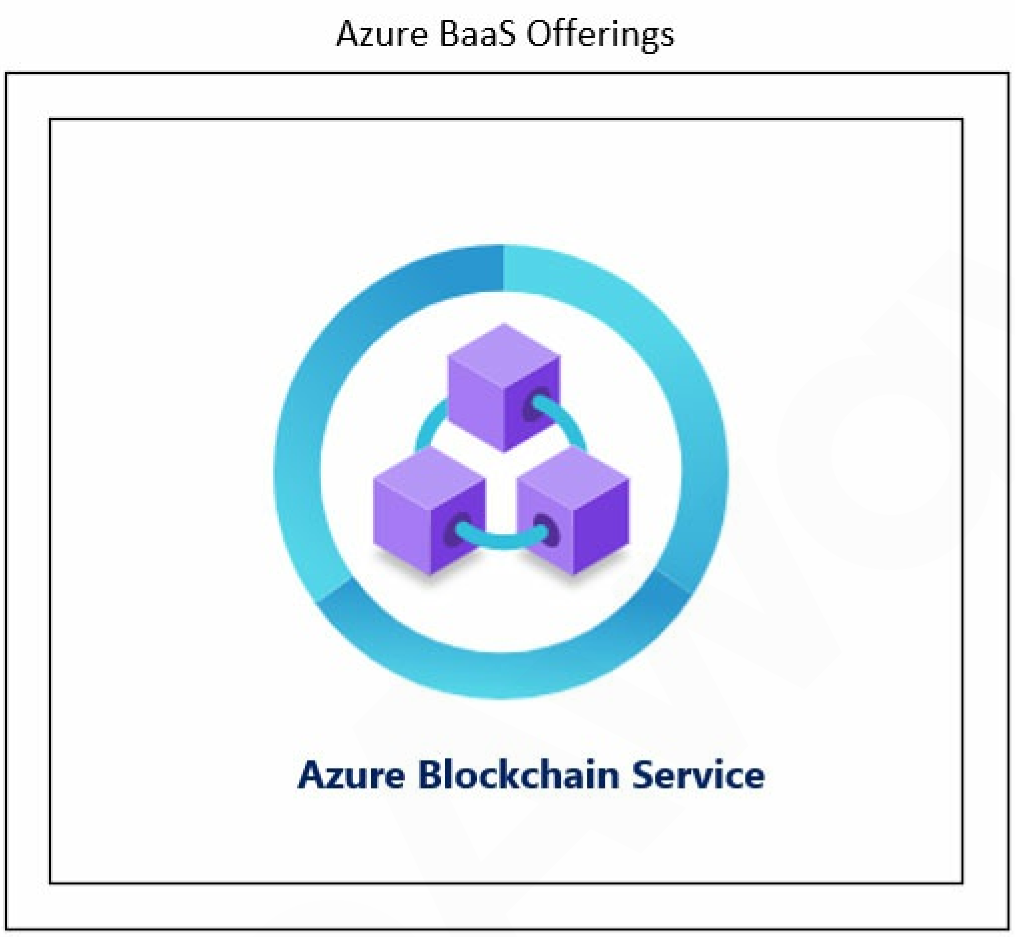
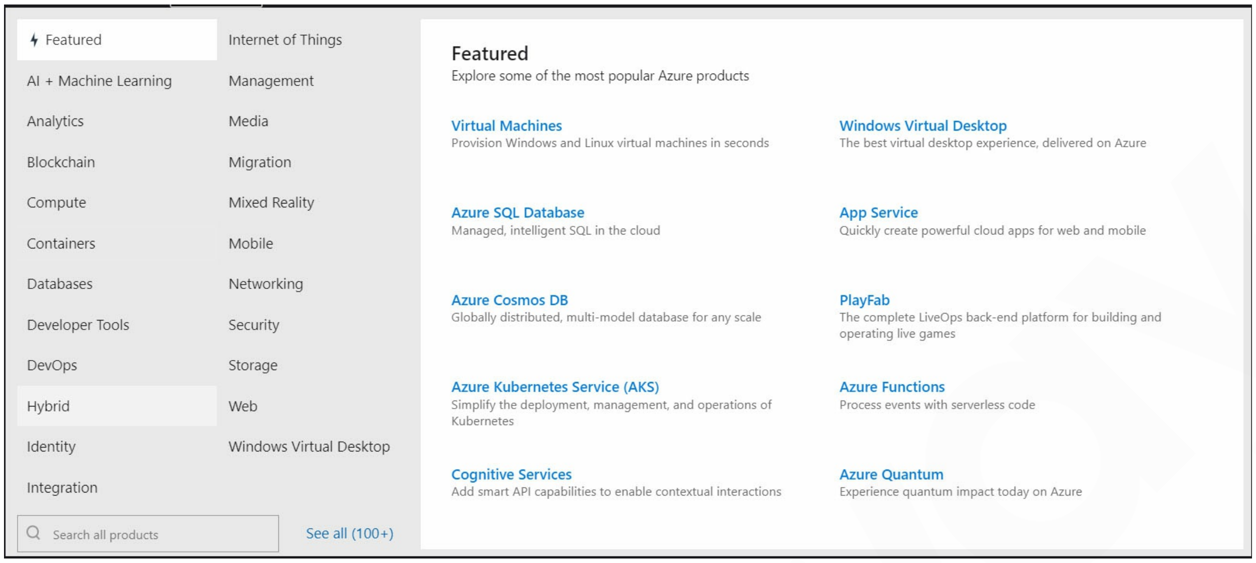


Figure 1.6

At the time of writing this book, Azure offers 100+ services ready to work with, as can be seen in the following screenshot:



Featured
Explore some of the most popular Azure products

Virtual Machines
Provision Windows and Linux virtual machines in seconds

Windows Virtual Desktop
The best virtual desktop experience, delivered on Azure

Azure SQL Database
Managed, intelligent SQL in the cloud

App Service
Quickly create powerful cloud apps for web and mobile

Azure Cosmos DB
Globally distributed, multi-model database for any scale

PlayFab
The complete LiveOps back-end platform for building and operating live games

Azure Kubernetes Service (AKS)
Simplify the deployment, management, and operations of Kubernetes

Azure Functions
Process events with serverless code

Cognitive Services
Add smart API capabilities to enable contextual interactions

Azure Quantum
Experience quantum impact today on Azure

Figure 1.7

Azure has data centers across the globe. Azure combines these data centers into regions. Now, each region has multiple data centers to ensure that recovery from disasters is quick and efficient. Again, when I talk about regions, Azure has more global regions than any other cloud provider—offering the scale needed to bring applications closer to users around the world, preserving data residency, and offering comprehensive compliance and resiliency options for customers.

At the time of writing this book, Azure is spread across 55 regions worldwide and is growing at a faster rate. The recent I read the news was in the country of Israel.

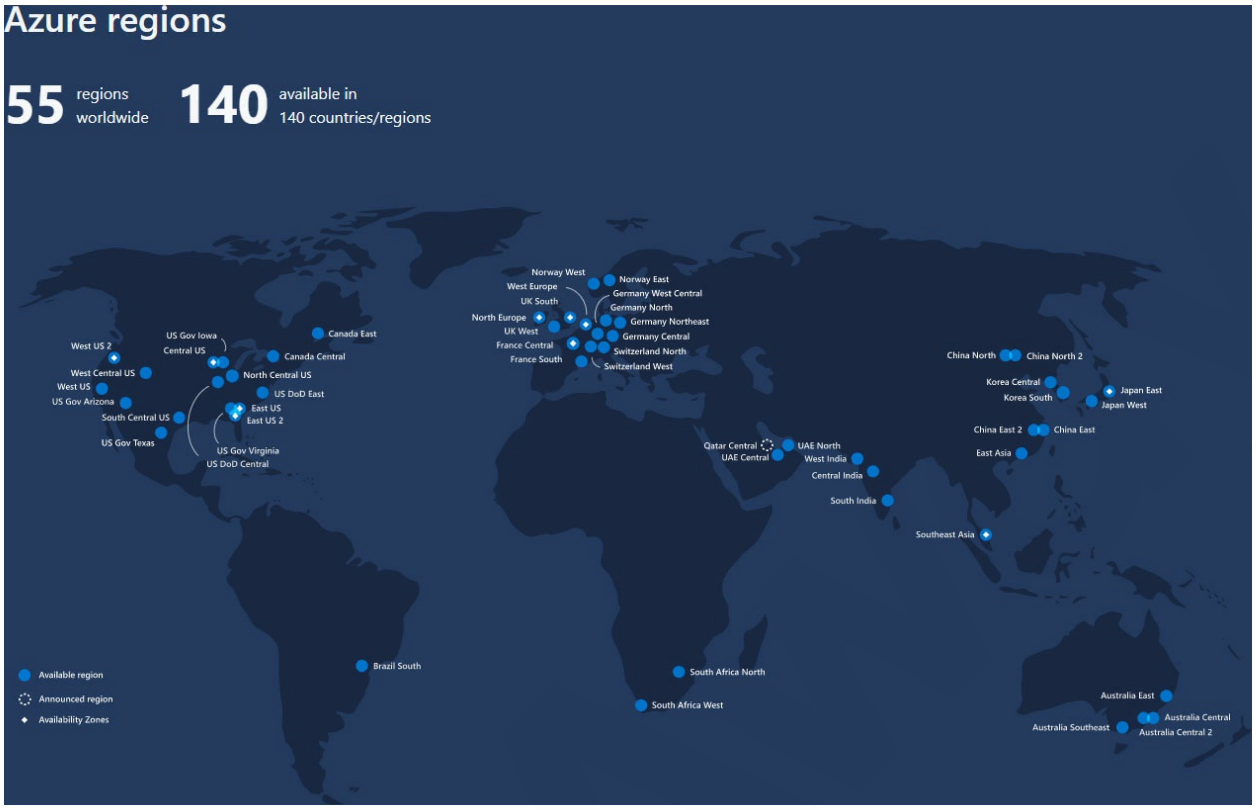


Figure 1.8

There are many topics that could get covered as part of fundamentals, but my objective in this book is to target the development aspect.

I would strongly recommend you to learn and target Azure Fundamentals certification. I always advised any aspiring Azure developer, with novice Cloud skills, to start with the Fundamentals learning path.

At the time of writing this book, the exam number for Azure Fundamentals is AZ-900. Go to the following link to learn how to prepare and pass Az-900 certifications. This exam course will introduce you to the wide world of Azure Ecosystem in a broader way.

Kudos! If you are already done with this certification!

Azure services – every developer must know

Widely discussed topic it is. Azure has multiple offerings in different areas such as compute, serverless, AI, and so on. As a developer, it’s not necessary to know all the services, but important is how to talk with these services. And to talk, you must learn them by diving more into it. Satya N. said, *Don’t be know-it-all, be a learn-it-all,* albeit it does refer to the company, but it can be implied to any course path.

But in my opinion, to list, following services such as Azure Apps, Azure Storage, Azure CosmosDB, Azure KeyVault are among the few that every developer must be aware of.

Working with Azure

Among many, one of the cool things about Azure is that there are multiple choices about how you go about dealing with your Azure resources. Whether you want to create a new web app, add Secrets in KeyVaults, configure the identity of app services, or stop any service on demand, there have many choices for how to achieve the same.

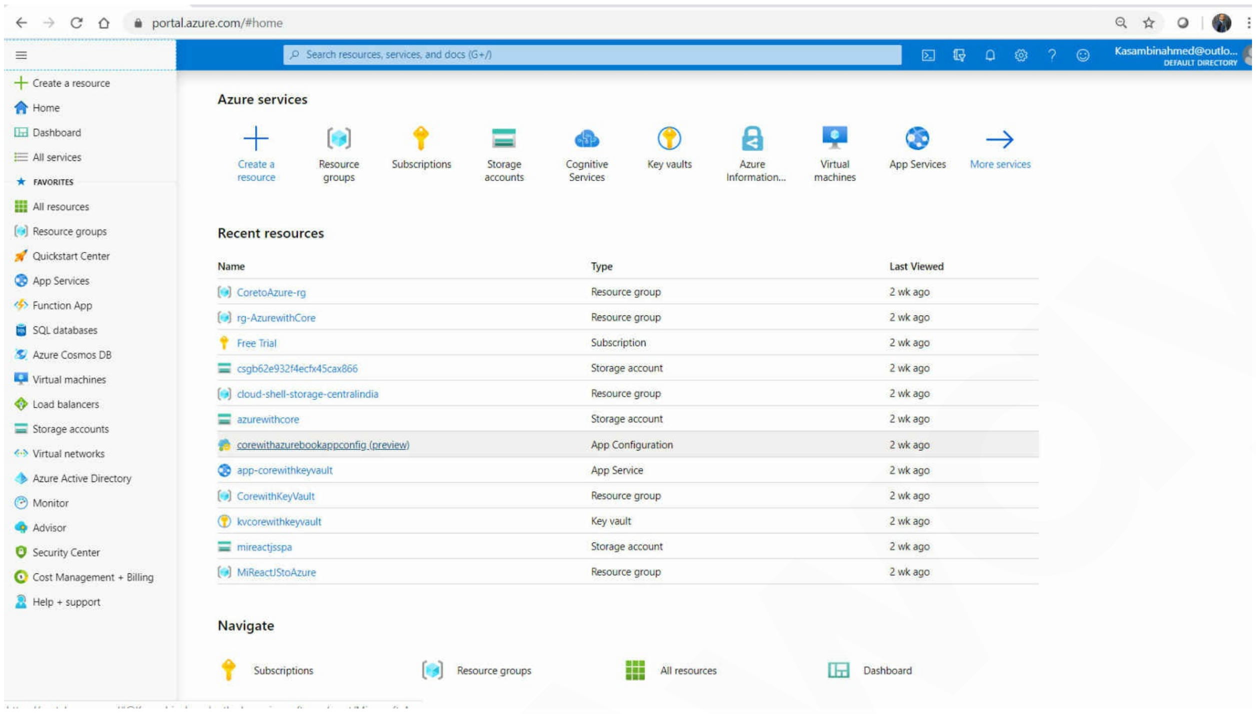
Let’s discuss what the ways are. To start with:

Azure portal

It is an amazing user interface to view, control, and manage your Azure resources. It comes with easy to use, multiple shortcut keys for better productivity, different themes to suit your mood on work 😊, with constant updates. Almost every task or activity can be accomplished using the Microsoft Azure portal. You can get into the portal even with no subscription, though with very limited privileges.

To start with, you need to enter following URL, **https://portal.azure.com/**

And you will be presented by following `Home` screen:



Recent resources

Name	Type	Last Viewed
CoretoAzure-rg	Resource group	2 wk ago
rg-AzurewithCore	Resource group	2 wk ago
Free Trial	Subscription	2 wk ago
csgb62e932f4ecfx45cax866	Storage account	2 wk ago
cloud-shell-storage-centralindia	Resource group	2 wk ago
azurewithcore	Storage account	2 wk ago
corewithazurebookappconfig (preview)	App Configuration	2 wk ago
app-corewithkeyvault	App Service	2 wk ago
CorewithKeyVault	Resource group	2 wk ago
kvcorewithkeyvault	Key vault	2 wk ago
mireactjsspa	Storage account	2 wk ago
MiReactJStoAzure	Resource group	2 wk ago

Figure 1.9

Many times it does happen; when you create any resource and explore the blades for that particular resource, you got to know many features associated with the azure resource you initially were unaware of. Like for instance, the following is the first level blade for Azure Storage:

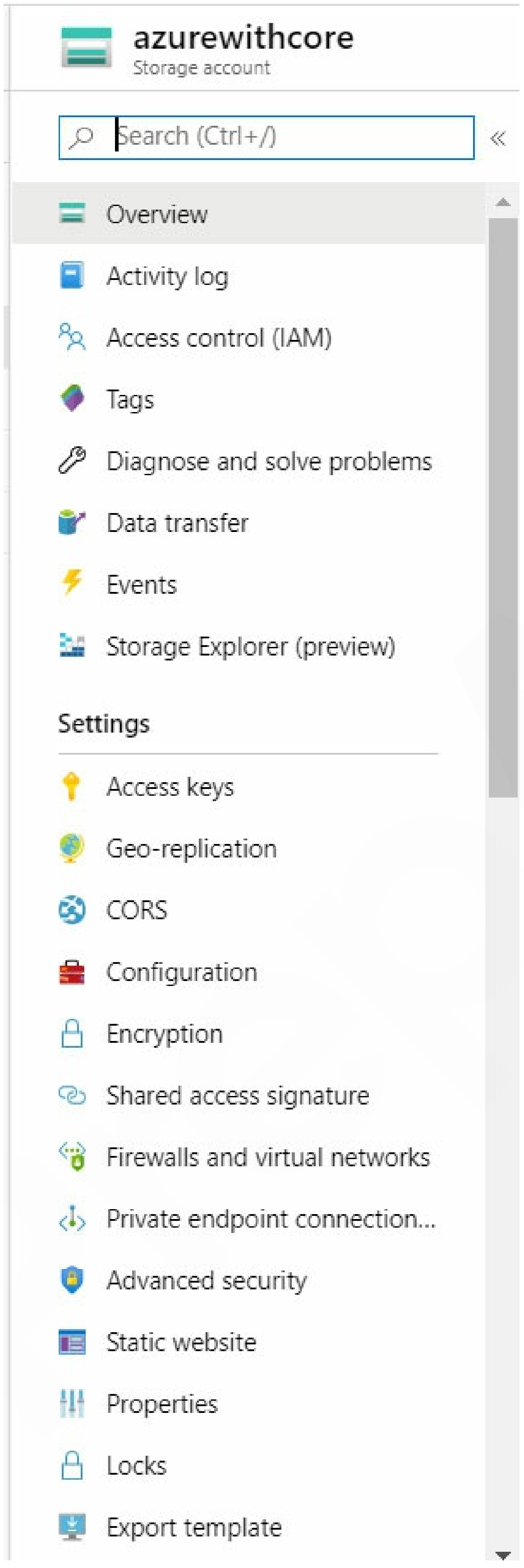

azurewithcore
Storage account
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Data transfer
Events
Storage Explorer (preview)
Settings
Access keys
Geo-replication
CORS
Configuration
Encryption
Shared access signature
Firewalls and virtual networks
Private endpoint connection...
Advanced security
Static website
Properties
Locks
Export template

Figure 1.10

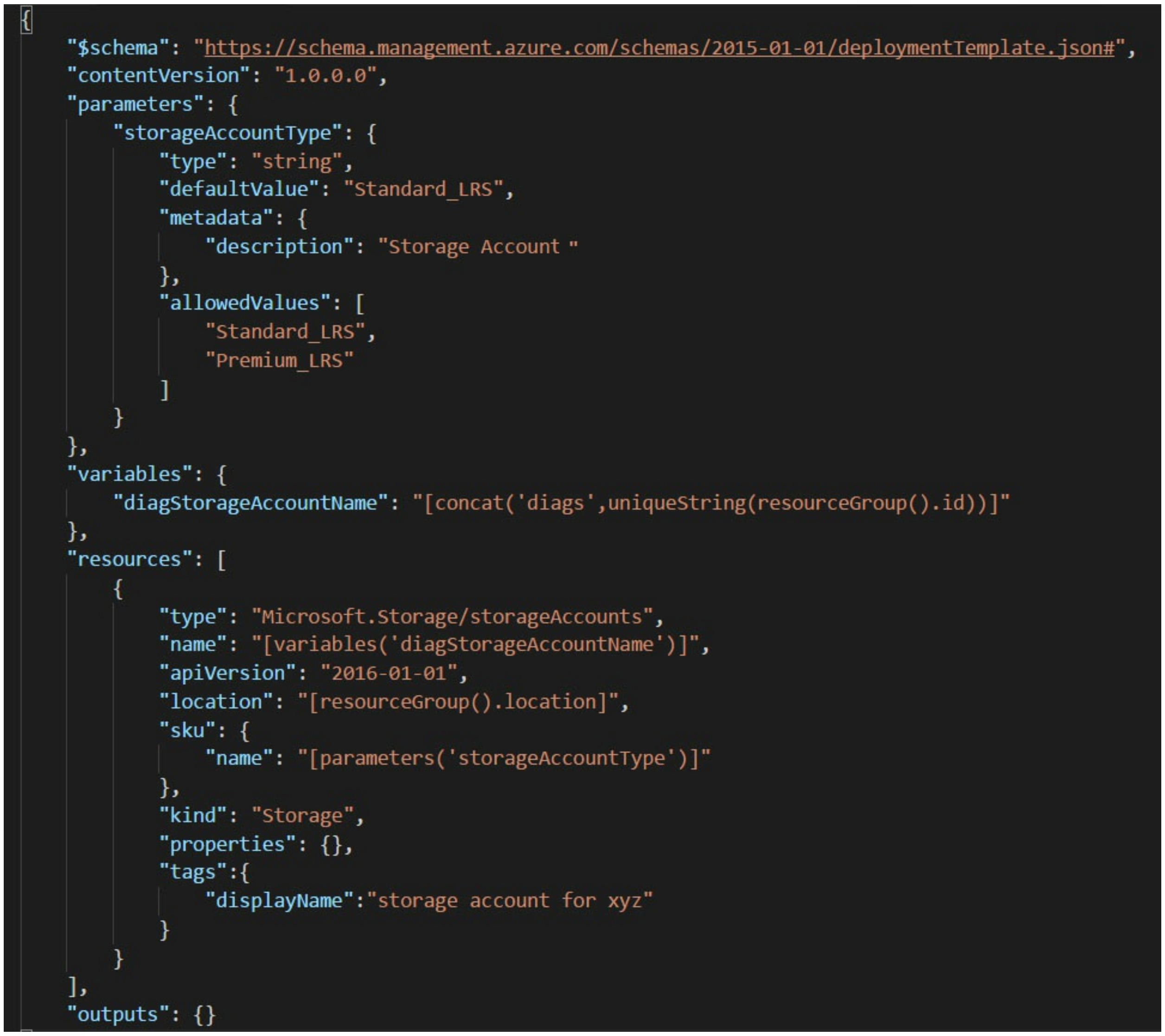
In the coming chapters, I will be using the Azure portal for resource provisioning.

Note: While the portal is great for experimenting and exploring, it is not a good choice for performing a repetitive task, like bulk deployments.

ARM templates

Azure Resource Manager (ARM) Templates are a way to declare the properties of any azure resource you want, the types, names in a JSON file that can reside in your source control, and managed like any other code file. ARM Templates gives you the capability to spin up Azure *Infrastructure as code*.

The following screenshot is the sample structure of the ARM template:



```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageAccountType": {
            "type": "string",
            "defaultValue": "Standard_LRS",
            "metadata": {
                "description": "Storage Account "
            },
            "allowedValues": [
                "Standard_LRS",
                "Premium_LRS"
            ]
        }
    },
    "variables": {
        "diagStorageAccountName": "[concat('diags',uniqueString(resourceGroup().id))]"
    },
    "resources": [
        {
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[variables('diagStorageAccountName')]",
            "apiVersion": "2016-01-01",
            "location": "[resourceGroup().location]",
            "sku": {
                "name": "[parameters('storageAccountType')]"
            },
            "kind": "Storage",
            "properties": {},
            "tags":{
                "displayName":"storage account for xyz"
            }
        }
    ],
    "outputs": {}
```

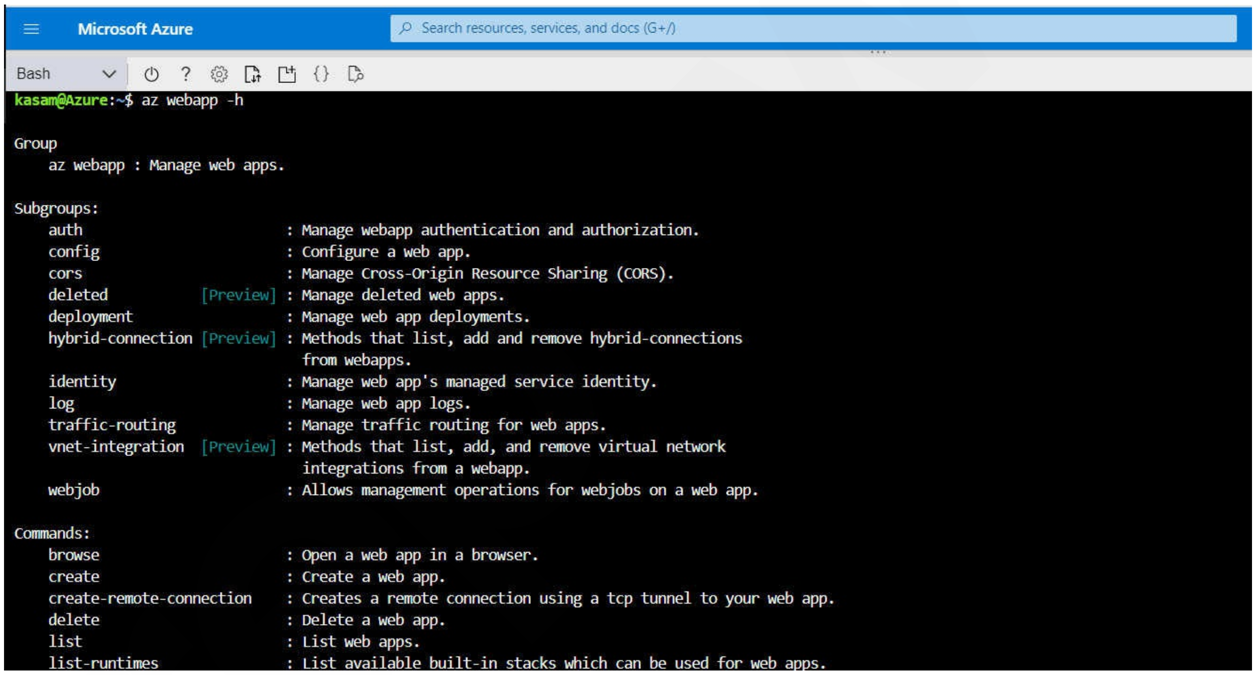
Figure 1.11

Azure CLI

Azure Command Line Interface is a fully cross-platform command-line experience, with capabilities on a par with Azure PowerShell. The Azure portal provided an icon to open the CLI mode right in the browser with the Bash option. You can also select the **`PowerShell`** option right in there.

It syntaxes with az command, following by the service name. It's much easier to use once you are used to working with it. With help command, you can dig down into more options available and gets the activities done seamlessly.

I will be using Azure CLI incoming chapter's exercises for few services. The following screenshot is how it looks in the Azure portal:



```
kasam@Azure:~$ az webapp -h

Group
    az webapp : Manage web apps.

Subgroups:
    auth                          : Manage webapp authentication and authorization.
    config                        : Configure a web app.
    cors                          : Manage Cross-Origin Resource Sharing (CORS).
    deleted             [Preview] : Manage deleted web apps.
    deployment                    : Manage web app deployments.
    hybrid-connection   [Preview] : Methods that list, add and remove hybrid-connections
                                    from webapps.
    identity                      : Manage web app's managed service identity.
    log                           : Manage web app logs.
    traffic-routing               : Manage traffic routing for web apps.
    vnet-integration    [Preview] : Methods that list, add, and remove virtual network
                                    integrations from a webapp.
    webjob                        : Allows management operations for webjobs on a web app.

Commands:
    browse                        : Open a web app in a browser.
    create                        : Create a web app.
    create-remote-connection      : Creates a remote connection using a tcp tunnel to your web app.
    delete                        : Delete a web app.
    list                          : List web apps.
    list-runtimes                 : List available built-in stacks which can be used for web apps.
```

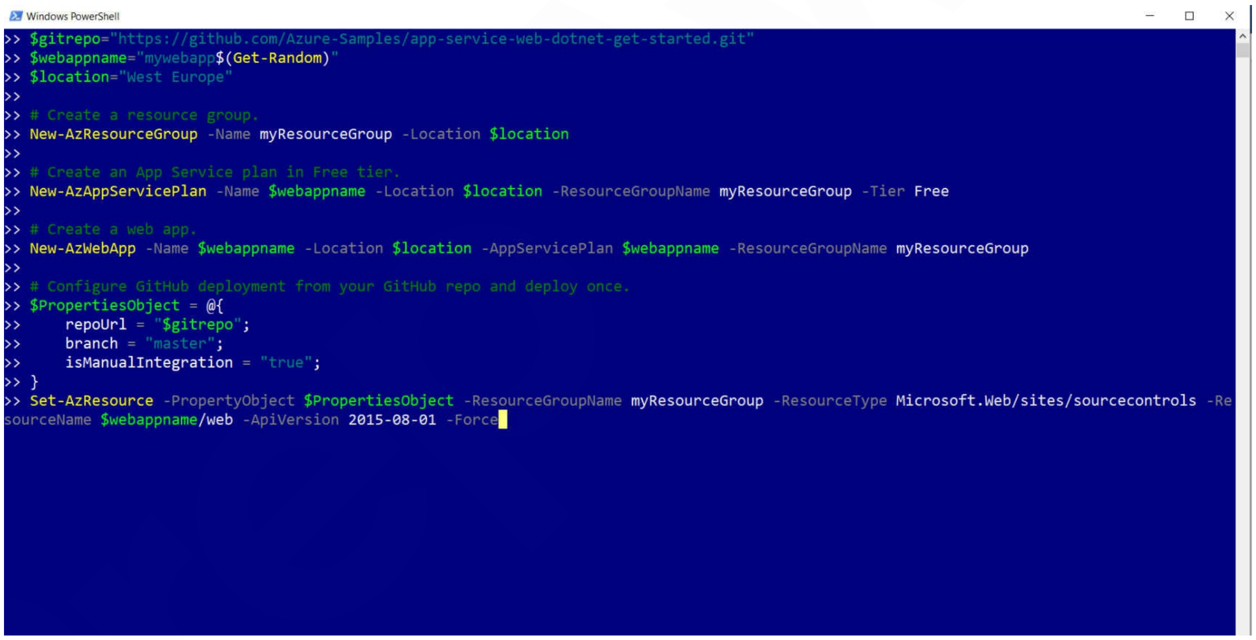
Figure 1.12

Azure PowerShell

Azure PowerShell is a wide-ranging collection of PowerShell cmdlets that allow you to do pretty much any managing operations with Azure. The commands use the PowerShell prefixes of `New-`, `Get-`, `Set-`, and `Remove-`, to create, read, update and delete operations.

Almost all the Azure services have PowerShell cmdlets available, so they can automate any activities taken care of by the portal. There are many examples present in GitHub to start with. Many developers are not familiar with working with PowerShell, but if you aim to automate the azure resource management, PowerShell is one of the best ways to achieve it.

Along with Windows, Azure PowerShell is also available on macOSand Linux, with the current az module:



```
>> $gitrepo="https://github.com/Azure-Samples/app-service-web-dotnet-get-started.git"
>> $webappname="mywebapp$(Get-Random)"
>> $location="West Europe"
>>
>> # Create a resource group.
>> New-AzResourceGroup -Name myResourceGroup -Location $location
>>
>> # Create an App Service plan in Free tier.
>> New-AzAppServicePlan -Name $webappname -Location $location -ResourceGroupName myResourceGroup -Tier Free
>>
>> # Create a web app.
>> New-AzWebApp -Name $webappname -Location $location -AppServicePlan $webappname -ResourceGroupName myResourceGroup
>>
>> # Configure GitHub deployment from your GitHub repo and deploy once.
>> $PropertiesObject = @{
>>     repoUrl = "$gitrepo";
>>     branch = "master";
>>     isManualIntegration = "true";
>> }
>> Set-AzResource -PropertyObject $PropertiesObject -ResourceGroupName myResourceGroup -ResourceType Microsoft.Web/sites/sourcecontrols -Re
sourceName $webappname/web -ApiVersion 2015-08-01 -Force
```

Figure 1.13

The preceding screenshot is the sample screen for PowerShell window running Azure cmdlets.

Next comes

REST APIs, which can be used for initial testing using any tool such as Postman, and management libraries for different programming languages. I will be using the same in the coming chapters for .NET core applications.

Azure provides multiple ways to connect, automate, and interact with the intellectual cloud. All methods do require users and codes to get authenticated with valid credentials before they can be leveraged into any software system, as mentioned in the earlier part of the chapter.

What we will cover in the book

This book will definitely help you to make your application talk to Azure services. I will make you go through Azure App services and the tips and tricks to connect with it and leverage its intelligent features. I will be starting the talk with:

- **Azure Web App services:** The way you can host and run the application.
- **Azure CosmosDB:** The seamless way of designing your backend.
- **Azure Storage:** Managing storage, along with the supercool features of hosting the client-side applications.
- **Security Management:** Best way and approach to deal with secret keys in your application.
- **Azure Functions:** Developing, debugging, deploying Serverless offering using favorite IDE Visual Studio.

Also, I will be explaining to you how to manage Azure resources, sometimes using Microsoft Azure Portal and Azure Command Line Interface, CLI. And along with learning, I will let you know a few questions, frequently asked by interviewers, isn’t it interesting?

I believe this book is the very first book to have its exercise using .NET Core 3.1 and Visual Studio 2019. Our focus will be on making your hand dirty with the must know azure services and making you comfortable enough to dive in more in it.

Prerequisite and setup

The best part here is *to make a Start*, and that is what you are already doing, learning, and reading this book. Another most important prerequisite is your zeal to learn. Concentrate on what I am trying to convey you in upcoming chapters, repeat the exercise on your own, you will find the difference in your confidence level with respect to Microsoft Azure as a developer, before and after reading this book.

Along with the above, you must have a:

- **Valid Azure Subscription:** Now, this subscription can be a free trial account offered by Microsoft Azure. It is required to work with Azure services & products. You can explore more about Azure Free trial account by watching Video at following given shorten link **http://bit.ly/az-free**
- **Visual Studio 2019:** Here, you can use any IDE of your choice to work with .NET core applications. But in this book, I have detailed all code exercise using the most favorite IDE of not only mine but most of the developers, Visual Studio 2019. I am using community edition along with installing Azure Workloads. You can download it from following given shorten link **http://bit.ly/vsstudio2019**
- **.NET Core SDKs:** Do remember to install the latest SDKs available for .NET Core from following given shorter link **http://bit.ly/aspdotnetcore3**. I have used .NET Core 3.1 for most of the exercise.

And yes, not required to mention but the Internet with good bandwidth to connect with.

Conclusion

So, you must now be having a gist of what you are dealing with and going to dive in ahead in upcoming chapters. Also, along with Azure Services, you as a .NET Core developer must be aware of the required prerequisites and setup to perform the exercise and homework added as a part of each chapter.

In the next chapter, I will be starting with the Azure Web App service. Let’s start!

Questions

1. List any three Azure services under IaaS models offerings?
2. List any three Azure services under PaaS models offerings?
3. List any three Azure services under IPaaS models offerings?
4. What are the different ways of managing Azure resources?
5. What are the benefits of the Azure Resource Manager?
6. Explain working with Azure CLI?

CHAPTER 2

My App on Cloud - Microsoft Azure

Here we go, in the earlier *Chapter 1, Azure Ecosystem*, I made you familiar with Azure and its components, now, let's jump into the first step to kick off our application to move it over to Azure. As I detailed you in an earlier chapter, about important services every developer must know in Azure, let's have a deep dive into the very first service, Azure Web App Service. At the end of this chapter, you will be ready to deploy your Web, Mobile, API apps seamlessly in no time.

Nowadays, software systems consist of web applications, REST APIs, and mobile back ends. Azure App Service provides a service where you can host your software components. App Service provides windows as well as Linux environments. It supports services developed in .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. These App Services can be configured and scaled very easily.

As App Service is hosted in the Azure environment, Azure takes care of security, load balancing, autoscaling, and automated management. Continuous Integration Continuous Deployment or Delivery will be hereafter mentioned as CI CD capabilities are also available from Azure DevOps, GitHub, Docker Hub. You will choose the App Service plan to run your application. Azure App Service (**https://docs.microsoft.com/en-us/azure/app-service/overview**) is a fully managed compute platform. You will be charged on the basis of the quantity of azure compute resources that you use in your App Service–plan.

To host web applications, Azure has services such as **Service Fabric, Azure Virtual Machines**. Additional services can be used if you want more control over the environment and architecture.

Structure

- The popularity of App Service
- Create an ASP.NET Core web app
- Deployment
- CI CD

Objectives

- All about Microsoft Azure Web App Service
- Hosting our application to Azure Web App using Azure Portal
- Hosting our application to Azure Web App using Visual Studio 2019
- Different ways of deploying the application to Azure web app
- Implementing continuous deployment with GitHub as a repository

Why is App Service so much popular?

When I mentioned, App Service is one of the key services, every developer should know, and took it as the very first chapter to get you introduced with, followingare some key features justifying my claims, rather boost up your eagerness to learn this service:

Developer’s view

1. Azure marketplace provides the number of application templates that can speed up your development. For example, WordPress, Joomla, and Drupal.
2. Various Visual Studio tools and extensions are available, which will simplify development, debugging, and deployment.
3. App Service has first-class support for .Net frameworks, including Java,PHP, Python. You can also run **PowerShell and other scripts or executables** as background services.
4. App Service enables authentication, offline data sync, push notifications, for mobile app development and also supports CORS, which will streamline the mobile app development process.
5. App Service can provide an environment to execute script on-demand without having to provision or manage infrastructure explicitly, and pay if your code was executed and at the time of execution.

IT Pros view

1. CI CD can be setup using Azure DevOps with VSTS, GitHub, BitBucket, and Docker Hub. You can plan the succession of your web app deployment and execute it from Dev to QA to production.
2. You can scale your application manually as well as set to auto-scaling as required. Azure promises high availability irrespective of the data center where the application is hosted.
3. We can connect the web app to on-premise infra using hybrid connections or Azure virtual networks. It can be connected to 50 connectors for enterprise systems (such as SAP), SaaS services (such as Salesforce), and internet services (such as Facebook).
4. App Service is **International Organization for Standardization (ISO), Service Organization Control (SOC),** and **Payment Card Industry (PCI)** compliant. Authenticate users with Azure Active Directory or with social login (Google, Facebook, Twitter, and Microsoft). Create IP address restrictions and manage service identities.

Using Azure App Service, we can host:

- Windows web apps
- Linux web apps
- Mobile apps
- Azure functions
- Docker containers

Azure App Service is a dedicated environment for App Service apps where the application will run securely and on a large scale. You can choose App Service when you need:

- Secure network access
- Very large scale for your application
- A significant amount of memory utilization

Workers are roles that host customer apps. At the time of writing this book, Workers are available in three fixed sizes:

- One vCPU/3.5 GB RAM
- Two vCPU/7 GB RAM
- Four vCPU/14 GB RAM

As per the hosting plan selected by User, associate Infra will be removed or added to App Service. It also comes with a fixed monthly rate for this Infra and not depends on the size change of the ASE. ASEs are deployed into a virtual network and isolated to run only a single customer's applications. Customers have control over application network traffic. An ASE can be either internet-facing with a public IP address or internal-facing with only an Azure **internal load balancer (ILB)** address.

Apps also frequently need to access corporate resources, such as internal databases and web services. If you deploy the ASE in a virtual network that has a VPN connection to the on-premises network, the apps in the ASE can access the on-premise resources.

Azure App Service now on Linux

Now you can host your web application to App Service using the Linux App Service environment. It supports various languages to develop web apps such as NodeJS, PHP Python java, and dot net core as well. App Service on Linux supports a number of Built-in images in order to increase developer productivity.

App Service on Linux service does not have a **Free or Shared** tier. You cannot create Web App for Containers in an App Service plan already hosting non-Linux Web Apps. Microsoft does not recommend adding Windows and Linux apps in the same resource group.

You can monitor your application through logs. You can check log files in Docker logs using the SCM site or FTP location. If you enable diagnostic logging and Docker container logging, stout, and stderr will be logged in log files. App Service detects the settings change and restarts the container for you automatically.

Dot Net Core App in Azure

Dot net core app can be deployed to Azure App Service. Azure App Service has 64 as well as 32-bit runtimes. 32-bit applications can be built and deployed using .NET core SDK, and for deploying 64-bit applications, we can use the Kudu console.

Now we are going to see how to deploy your first ASP.NET Core web app to Azure App Service. We are going to create a web application using Dot Net core 2.2 and deploy it to Azure App Service through Visual Studio 2019.

Prerequisites are as follows

- **Visual Studio:**
    - Install the latest version of Visual Studio; in our case, it is Visual Studio 2019. Also, install and enable ASP.NET and web development workload. After installation check for updates, if there are any, please install them as well.
- **Azure Subscription:**
    - You need Azure subscription to create and access Azure services such asApp Service, VMs, and so on. if you don’t have a subscription, you can create a free account using a link **https://azure.microsoft.com/en-gb/free/dotnet/**. You will get 12 months’ usage for popular services with 13,300 initial credit.

Create an ASP.NET Core web app

To develop web applications, Visual Studio provides a project template. It gives a boilerplate implementation of the MVC application. Perform the following steps:

1. Open Visual Studio 2019 and create a project by selecting **`File | New | Project`**.

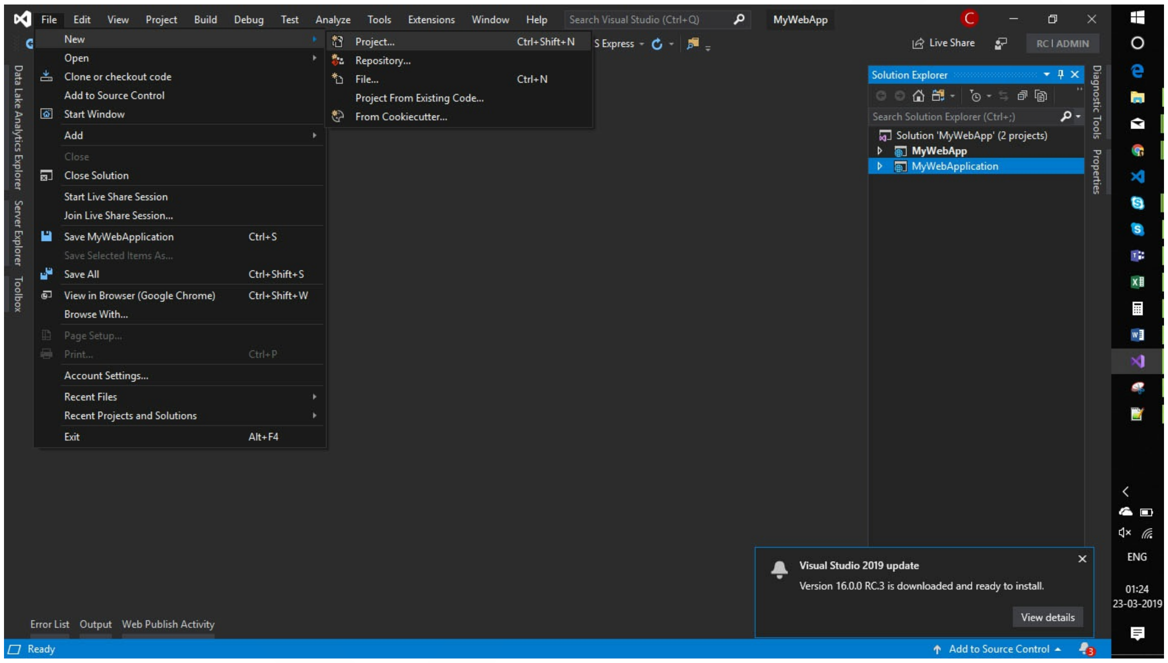
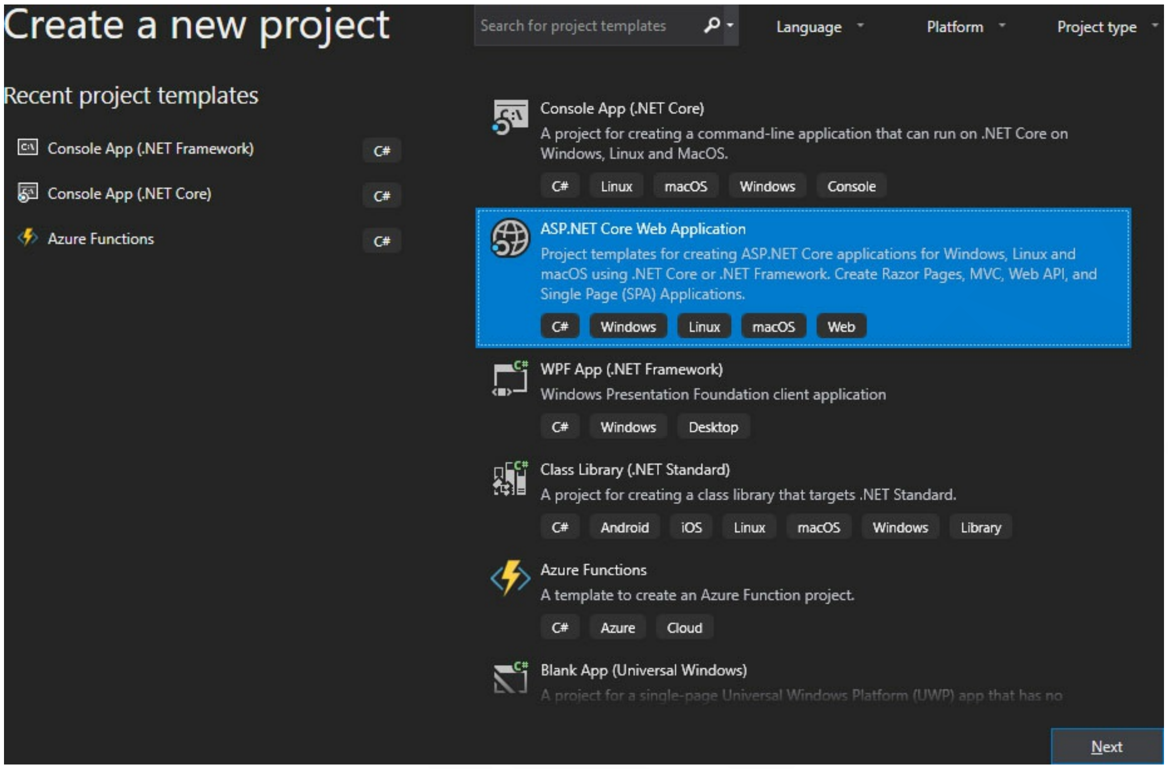


Figure 2.1

2. In the **`Create a new project`** dialog, select the **`ASP.NET Core Web Application`** template. And click on **`Next`**:



Create a new project

Search for project templates | Language | Platform | Project type

Recent project templates

- Console App (.NET Framework) C#
- Console App (.NET Core) C#
- Azure Functions C#

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 applications for Windows, Linux and macOS using .NET Core or .NET Framework. Create Razor Pages, MVC, Web API, and Single Page (SPA) Applications.
C# Windows Linux macOS Web

WPF App (.NET Framework)
Windows Presentation Foundation client application
C# Windows Desktop

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
C# Android iOS Linux macOS Windows Library

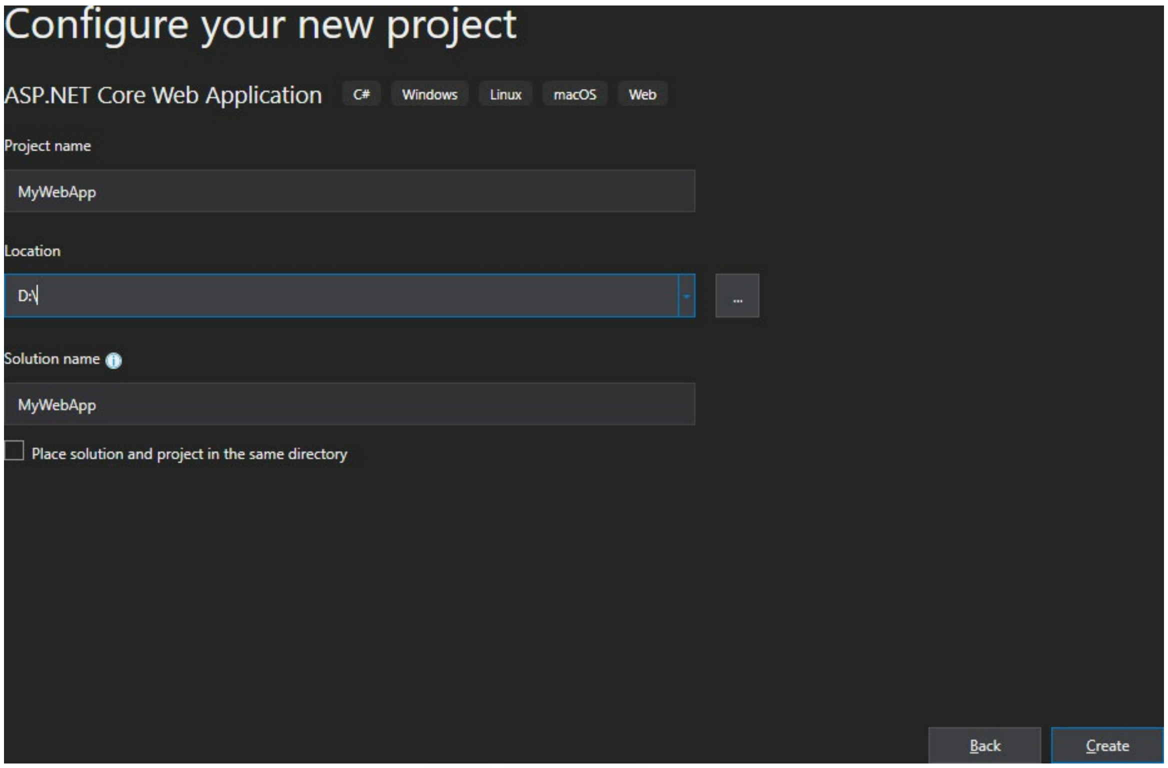
Azure Functions
A template to create an Azure Function project.
C# Azure Cloud

Blank App (Universal Windows)
A project for a single-page Universal Windows Platform (UWP) app that has no

Next

Figure 2.2

3. You can specify the name for the project and browse, select the **`Location`** where you want to save your project files. As it is our first project, it will ask for the **`Solution name`**. Specify the name for the solution and click on **`Create`**.


Configure your new project

ASP.NET Core Web Application

Project name: MyWebApp

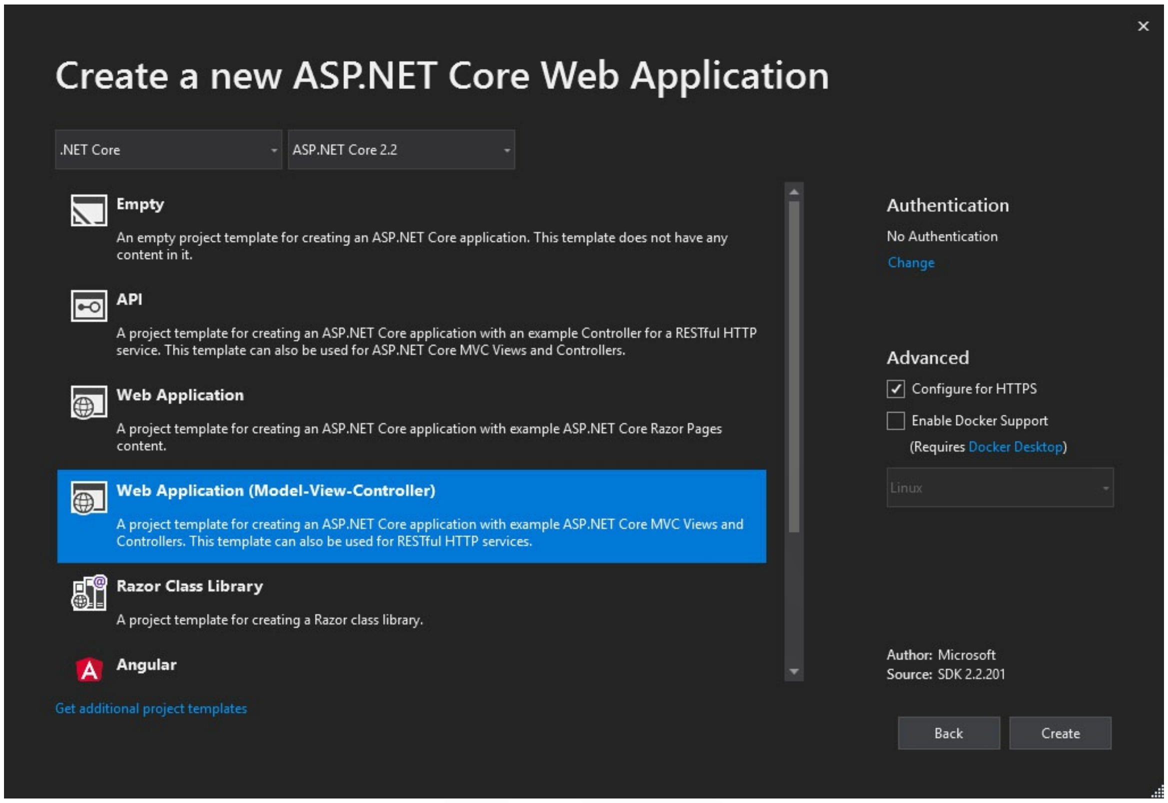
Location: D:\

Solution name: MyWebApp

☐ Place solution and project in the same directory

Figure 2.3

4. In the next wizard screen, you can select the boilerplate development template. As we are going to create an MVC web app, select **`Web Application (Model–View–Controller)`** framework.



Create a new ASP.NET Core Web Application

.NET Core | ASP.NET Core 2.2

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

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.

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

Web Application (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.

Razor Class Library
A project template for creating a Razor class library.

Angular

Get additional project templates

Authentication
No Authentication
Change

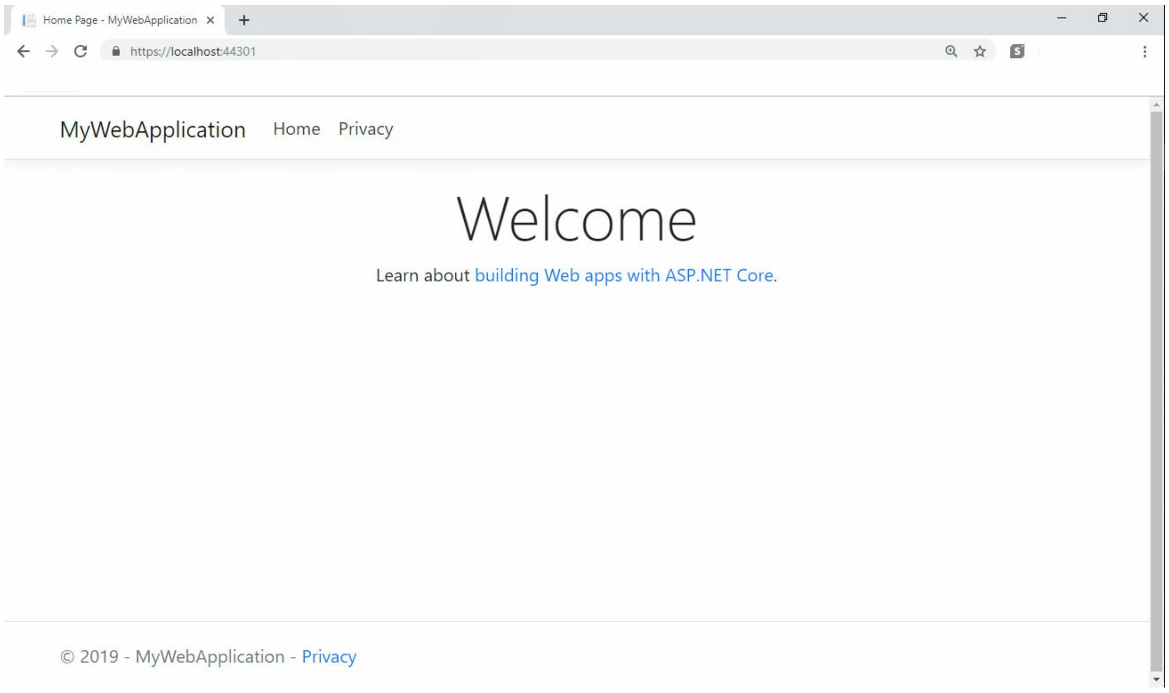
Advanced
[✓] Configure for HTTPS
[ ] Enable Docker Support
(Requires Docker Desktop)
Linux

Author: Microsoft
Source: SDK 2.2.201

Back | Create

Figure 2.4

5. From the main menu in Visual Studio, select **`Debug | Start Debugging`** or press *F5* to run the web app locally. It will host the application to IIS express and open the local site in the browser.



Home Page - MyWebApplication
https://localhost:44301

MyWebApplication Home Privacy

Welcome

Learn about building Web apps with ASP.NET Core.

© 2019 - MyWebApplication - Privacy

Figure 2.5

Deployment

We have created an MVC web app and run it locally to check implementation. We can deploy the web app to Azure App Service using the following ways:

- Publish wizard in Visual Studio
- Publish profile of existing App Service
- FTP

Using Publish wizard in Visual Studio

Now we will deploy the web application to Azure App Service.

1. To deploy a solution, right-click on the project and click **`Publish`**.

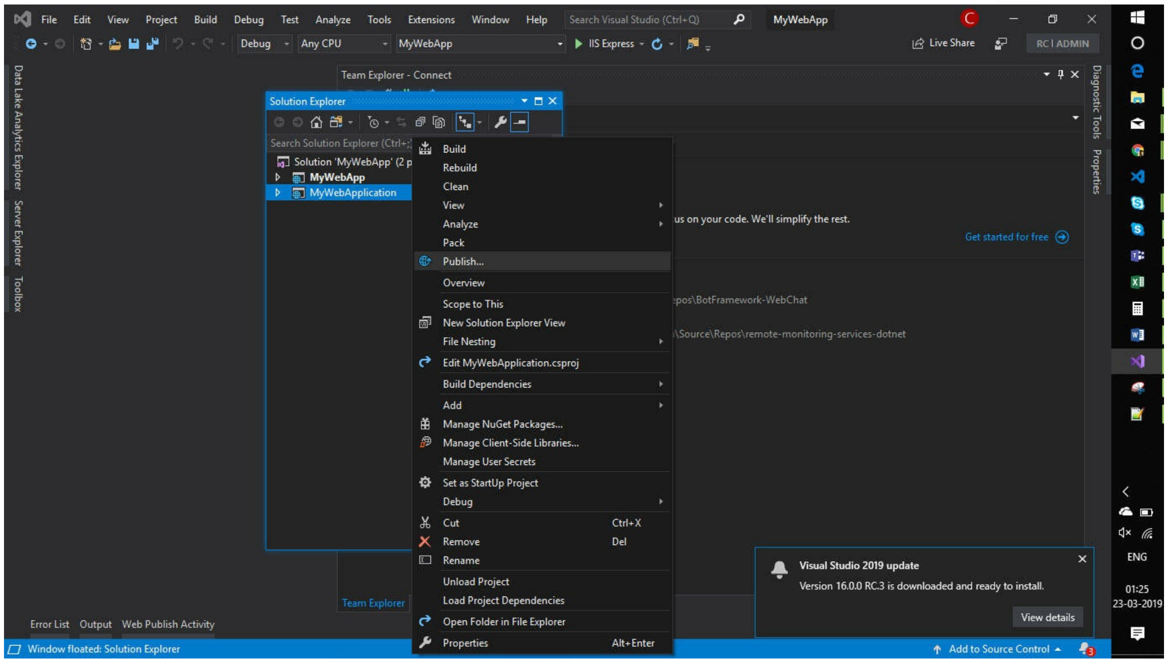
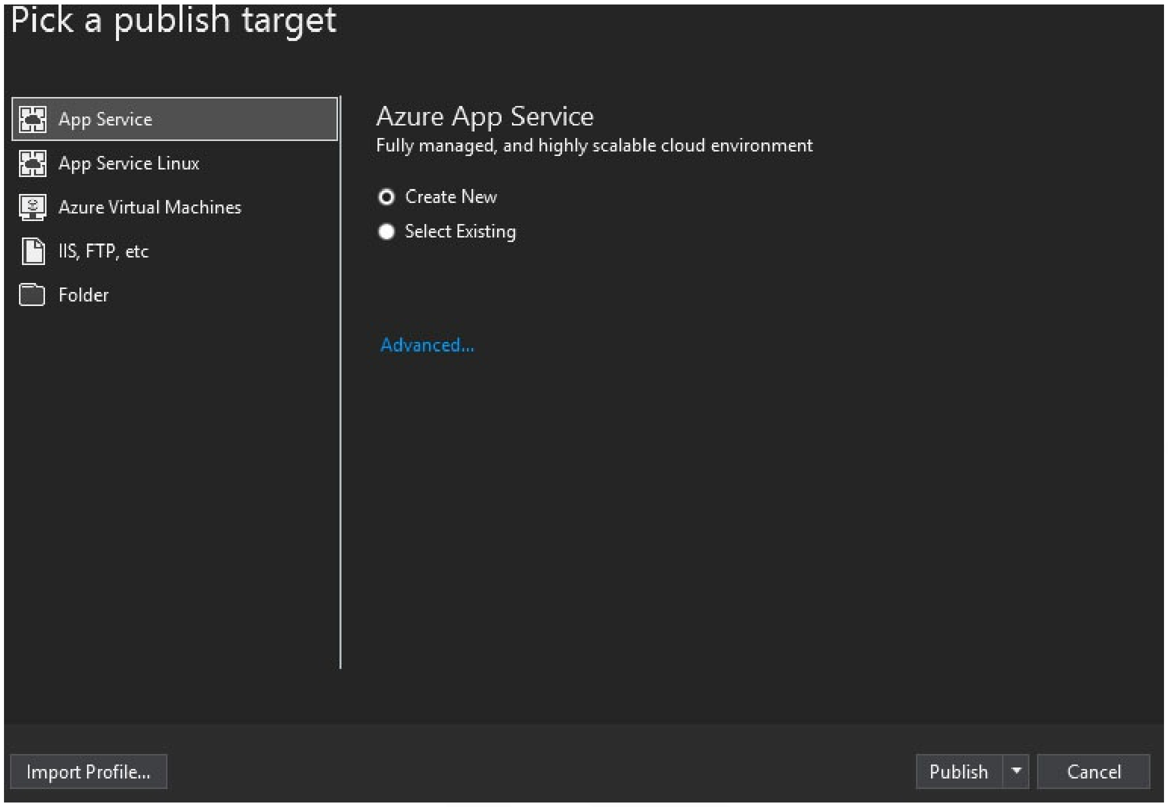


Figure 2.6

2. It will ask you to select the target to publish your code. Select **`App Service`** and check to create a new radio button click on **`Publish`**.


Pick a publish target

Azure App Service

Fully managed, and highly scalable cloud environment

Figure 2.7

Sign in to Azure

In the `Create App Service` dialog, click **`Add`** an account, and sign in to your Azure subscription. If you’re already signed in, then select the account you want from the dropdown.

Create a resource group

A **resource group** is a logical container into which Azure resources such as web apps, databases, and storage accounts are deployed and managed. Next to **`Resource Group`**, select **`New`**. Name the resource group and select **`OK`**.

We can group resources under the resource group. So, if, in the future, we want to delete resources, then we can directly delete a resource group instead of an individual resource. From the left menu in the Azure portal, select **`Resource groups`** and then select respective resource groups.

On the resource group page, make sure that the listed resources are the ones you want to delete. Select **`Delete`**, type name in the text box, and then select **`Delete`**.

Create an App Service plan

An **App Service plan** specifies the location, size, and features of the web server farm that hosts your app. Multiple web apps can share a single App Service plan.

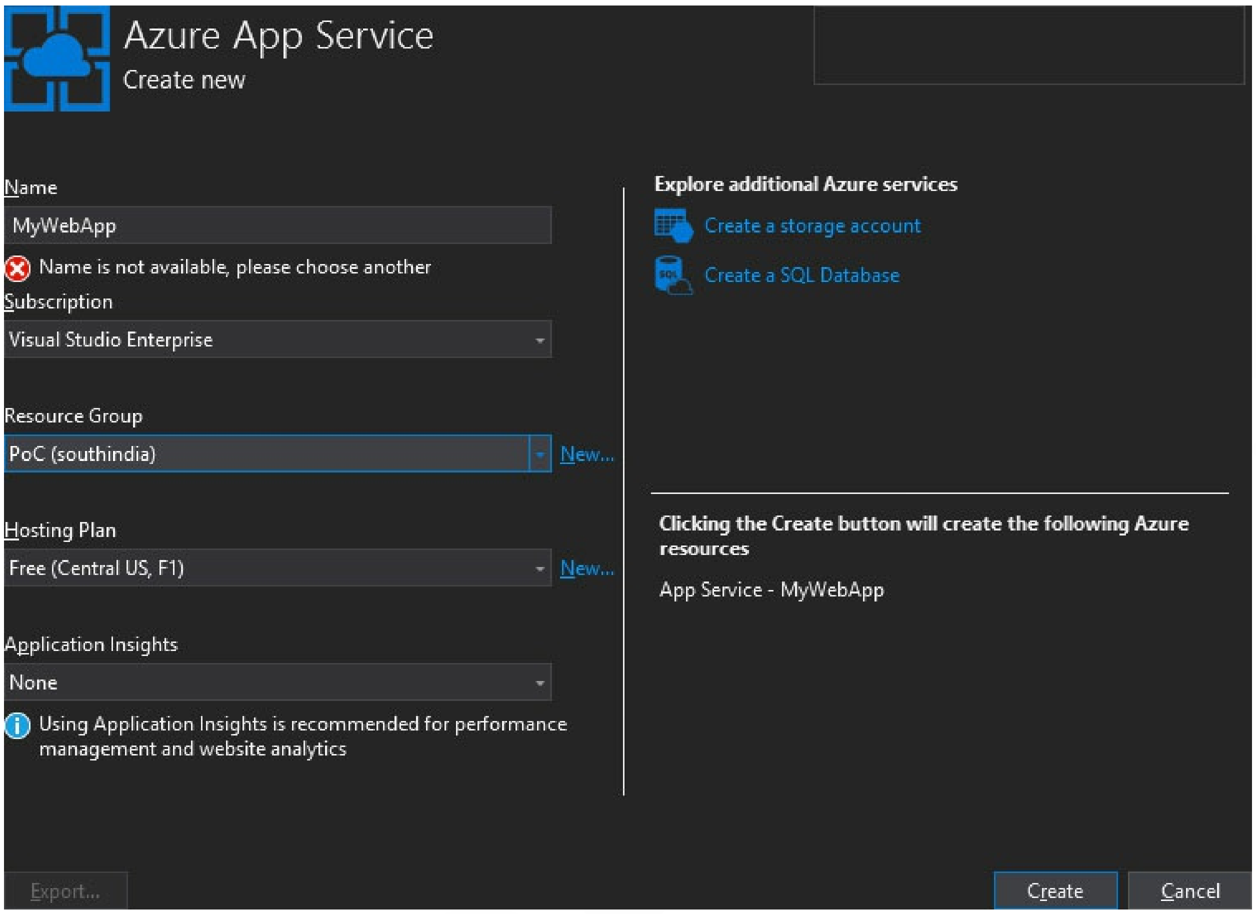
App Service plan defines:

- Region (for example West Europe, Central US, or Southeast Asia)
- Instance size (small, medium, or large)
- Scale count (1 to 15 instances)
- SKU (Free, Shared, Basic, Standard, or Premium)

Next to **`Hosting Plan`**, select **`New`**. In the **`Configure Hosting Plan`** dialog, use the settings in the table following the screenshot:

Setting	Description
App Service Plan	Name of the App Service plan.
Location	The datacenter where the web app is hosted.
Size	The pricing tier determines to host features.

Select **`OK`**.


Azure App Service

Create new

Name

MyWebApp

Name is not available, please choose another

Subscription

Visual Studio Enterprise

Resource Group

PoC (southindia)

Hosting Plan

Free (Central US, F1)

Application Insights

None

Using Application Insights is recommended for performance management and website analytics

Explore additional Azure services

Create a storage account

Create a SQL Database

Clicking the Create button will create the following Azure resources

App Service - MyWebApp

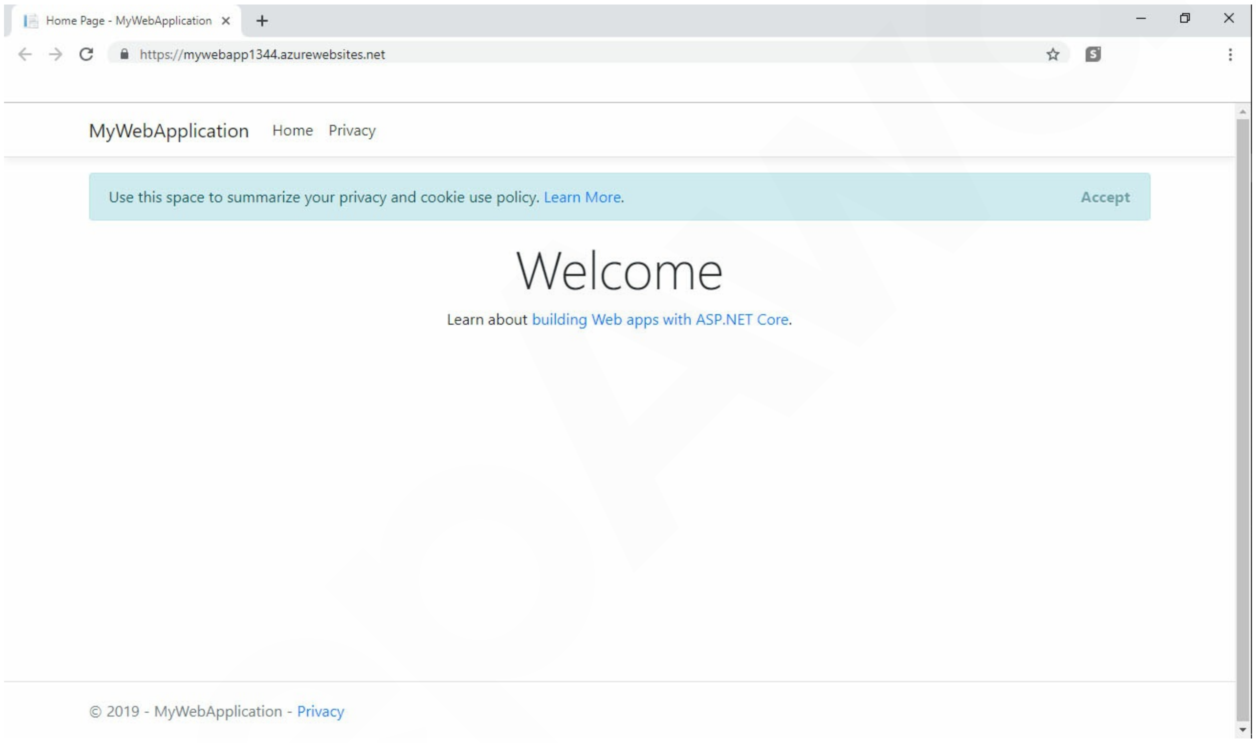
Figure 2.8

Create and publish the web app

Provide an app name. It should be unique as it will be used in the URL of the application. The URL of the web app is **http://<app_name>.azurewebsites.net**.

Select **`Create`** to start creating Azure resources.

Once the wizard completes, it publishes the ASP.NET Core web app to Azure and then launches the app in the default browser.


Home Page - MyWebApplication
https://mywebapp1344.azurewebsites.net
MyWebApplication
Home
Privacy
Accept

Use this space to summarize your privacy and cookie use policy. Learn More.

Welcome

Learn about building Web apps with ASP.NET Core.

© 2019 - MyWebApplication - Privacy

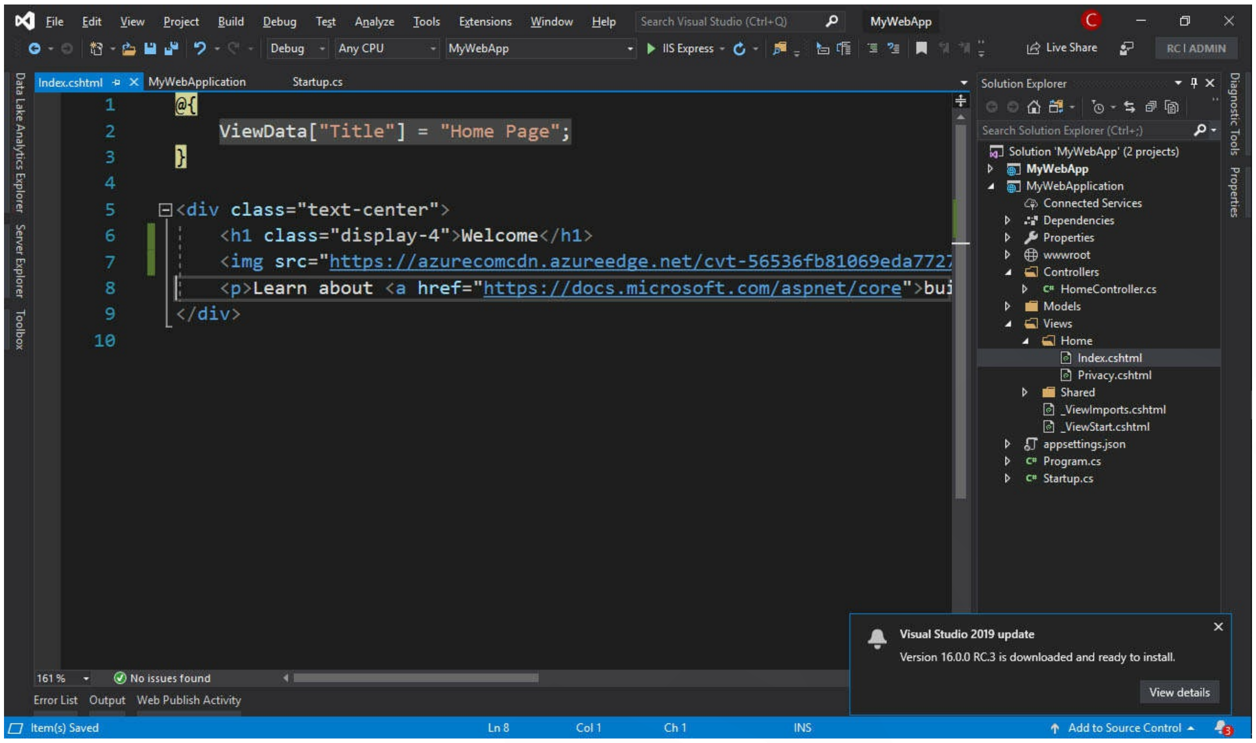
Figure 2.9

ASP.NET Core web app is running live in **`Azure App Service`**.

Let's update a solution and redeploy it. From the **`Solution Explorer,`** open **StartUp.cs**

Add

```
<img src="https://azurecomcdn.azureedge.net/cvt-
56536fb81069eda7727cd793e5f8e35194a748477a1b819a39e0a37e708ee516/
map-large.svg" />
```

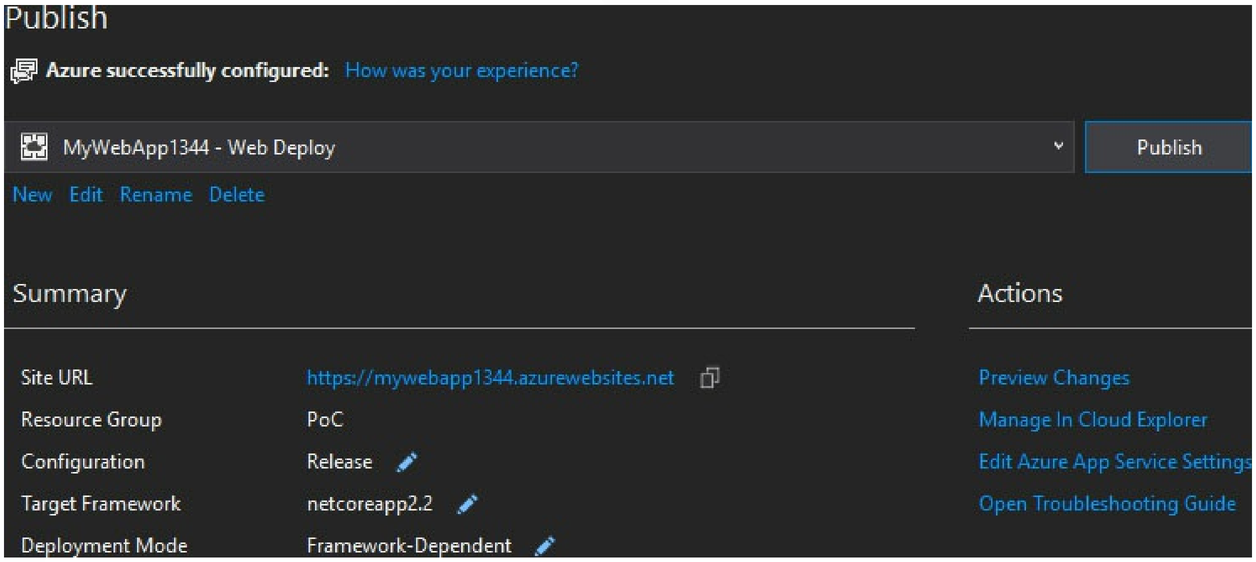


```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <img src="https://azurecomcdn.azureedge.net/cvt-56536fb81069eda7727
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">bui
</div>
```

Visual Studio 2019 update
Version 16.0.0 RC.3 is downloaded and ready to install.

Figure 2.10

To redeploy to Azure, right-click the project and select **`Publish`**.
In the publish, the summary page, select **`Publish`**.



Publish

Azure successfully configured: How was your experience?

MyWebApp1344 - Web Deploy

Summary

Site URL	https://mywebapp1344.azurewebsites.net
Resource Group	PoC
Configuration	Release
Target Framework	netcoreapp2.2
Deployment Mode	Framework-Dependent

Figure 2.11

When publishing completes, Visual launches a browser to the URL of the web app.

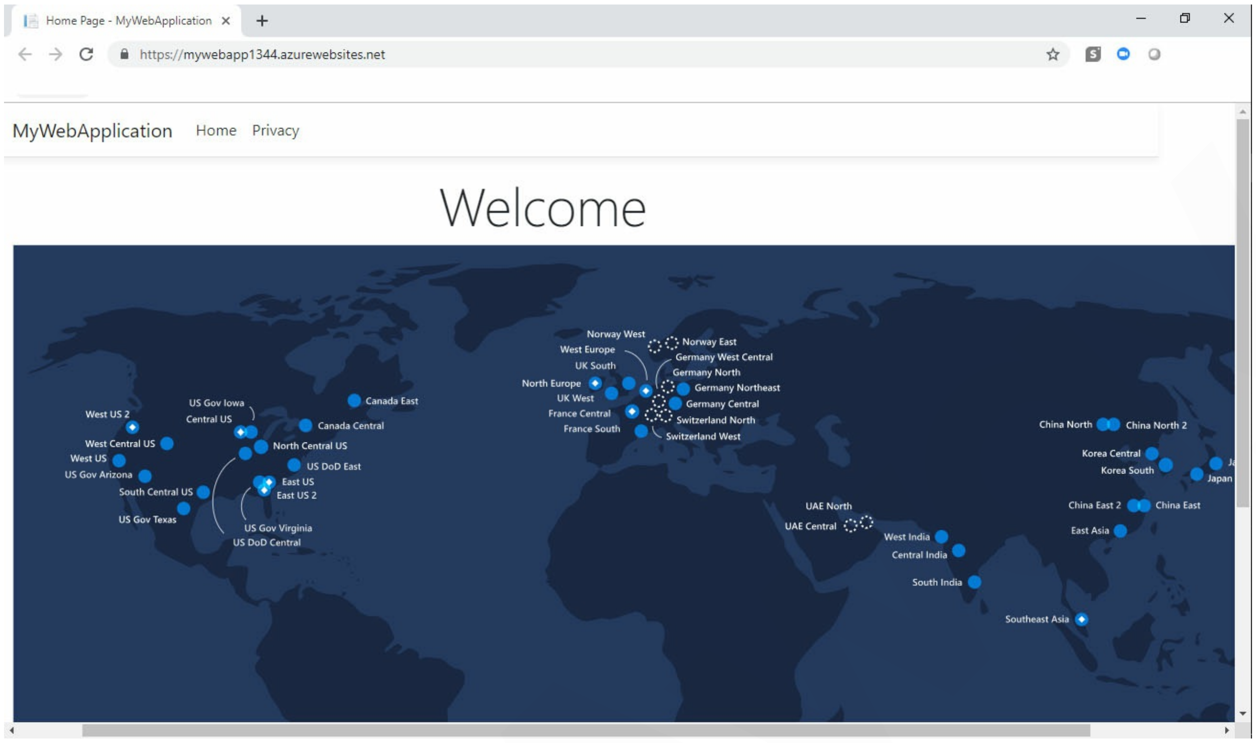
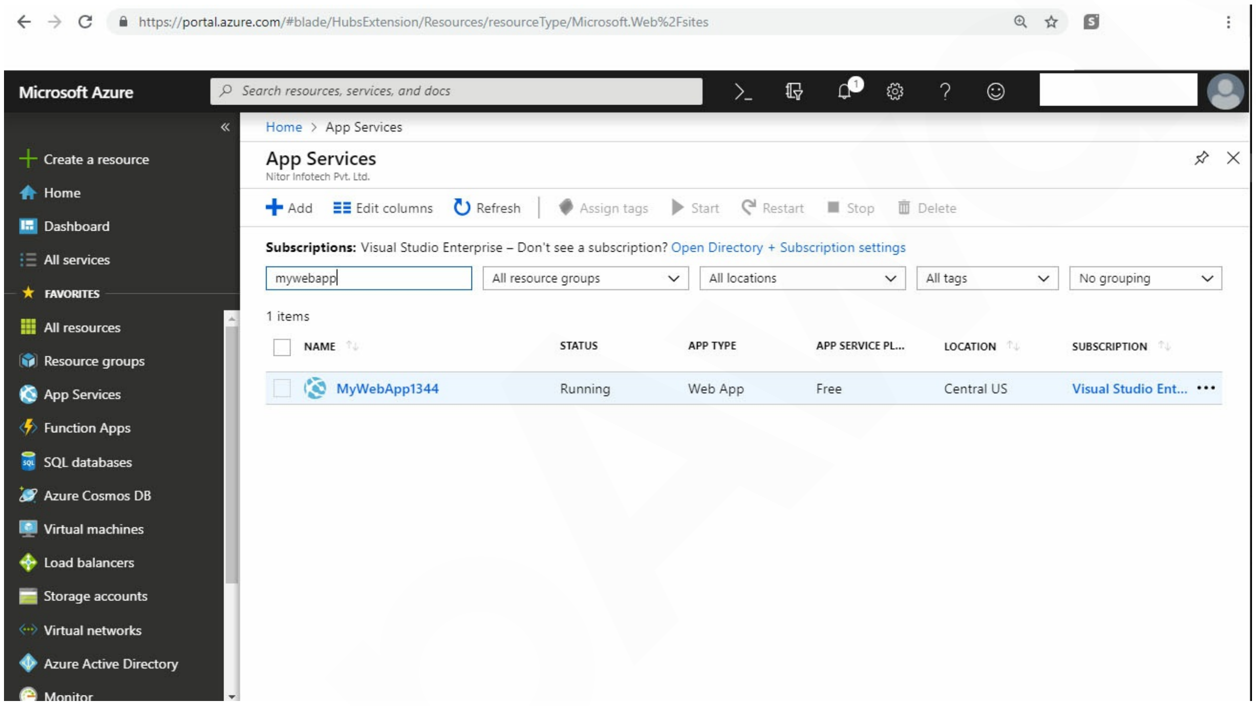

Home Page - MyWebApplication
https://mywebapp1344.azurewebsites.net
MyWebApplication
Home
Privacy
Welcome
Norway West
Norway East
West Europe
Germany West Central
UK South
Germany North
North Europe
Germany Northeast
UK West
Germany Central
France Central
Switzerland North
France South
Switzerland West
Canada East
US Gov Iowa
West US 2
Central US
Canada Central
West Central US
North Central US
West US
US DoD East
US Gov Arizona
East US
South Central US
East US 2
US Gov Texas
US Gov Virginia
US DoD Central
China North
China North 2
Korea Central
Korea South
Japan
China East 2
China East
East Asia
UAE North
UAE Central
West India
Central India
South India
Southeast Asia

Figure 2.12

Using existing App Service's publish profile

You can find Azure App Service resources at the **Azure portal**, and you can control it from there.

From the left-hand side menu, select **`App Services`**| select the name of your Azure app.



Subscriptions: Visual Studio Enterprise – Don't see a subscription? Open Directory + Subscription settings

mywebapp | All resource groups | All locations | All tags | No grouping

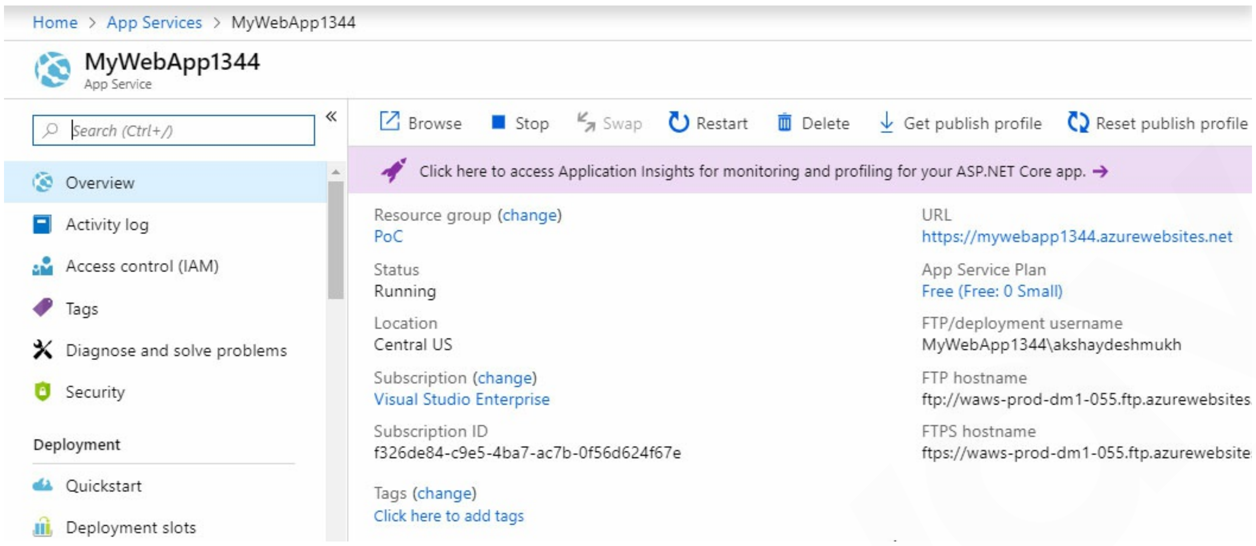
1 items

NAME	STATUS	APP TYPE	APP SERVICE PL...	LOCATION	SUBSCRIPTION
MyWebApp1344	Running	Web App	Free	Central US	Visual Studio Ent...

Figure 2.13

You see your web app's **`Overview`** page.

Here, you can perform basic resource management tasks such as **`Browse`**, **`Stop`**, **`Start`**, **`Restart`**, and **`Delete`**.



Click here to access Application Insights for monitoring and profiling for your ASP.NET Core app. →

Resource group (change)
PoC

Status
Running

Location
Central US

Subscription (change)
Visual Studio Enterprise

Subscription ID
f326de84-c9e5-4ba7-ac7b-0f56d624f67e

Tags (change)
Click here to add tags

URL
https://mywebapp1344.azurewebsites.net

App Service Plan
Free (Free: 0 Small)

FTP/deployment username
MyWebApp1344\akshaydeshmukh

FTP hostname
ftp://waws-prod-dm1-055.ftp.azurewebsites

FTPS hostname
ftps://waws-prod-dm1-055.ftp.azurewebsite

Figure 2.14

In the previous deployment, we have used **`Publish`** wizard in Visual Studio 2019, where we configured all settings and created publish profile.

When we want to deploy updates to existing App Service, we can import publish profiles for respective App Service and use it in publish wizard of Visual Studio 2019. It will be faster than selecting all settings manually.

Now let's create a new **`App Service`** in azure. Click on **`Create a resource`**.

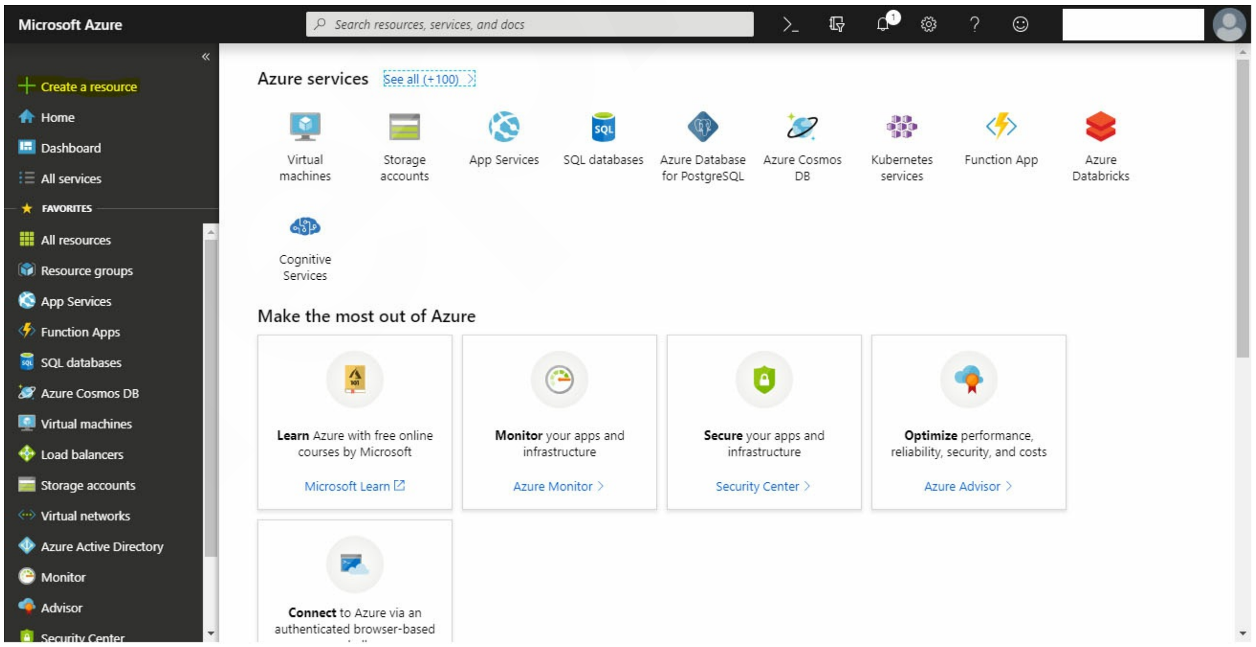
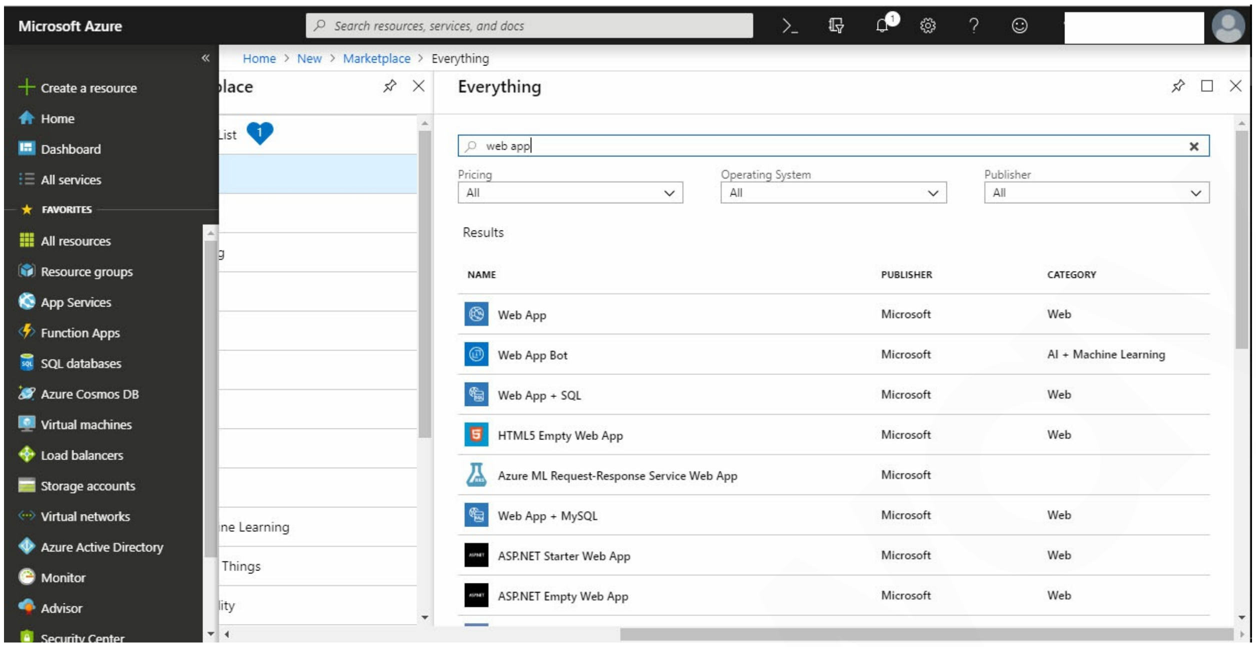


Figure 2.15

Search for the web app. It will list down all matching services.



web app

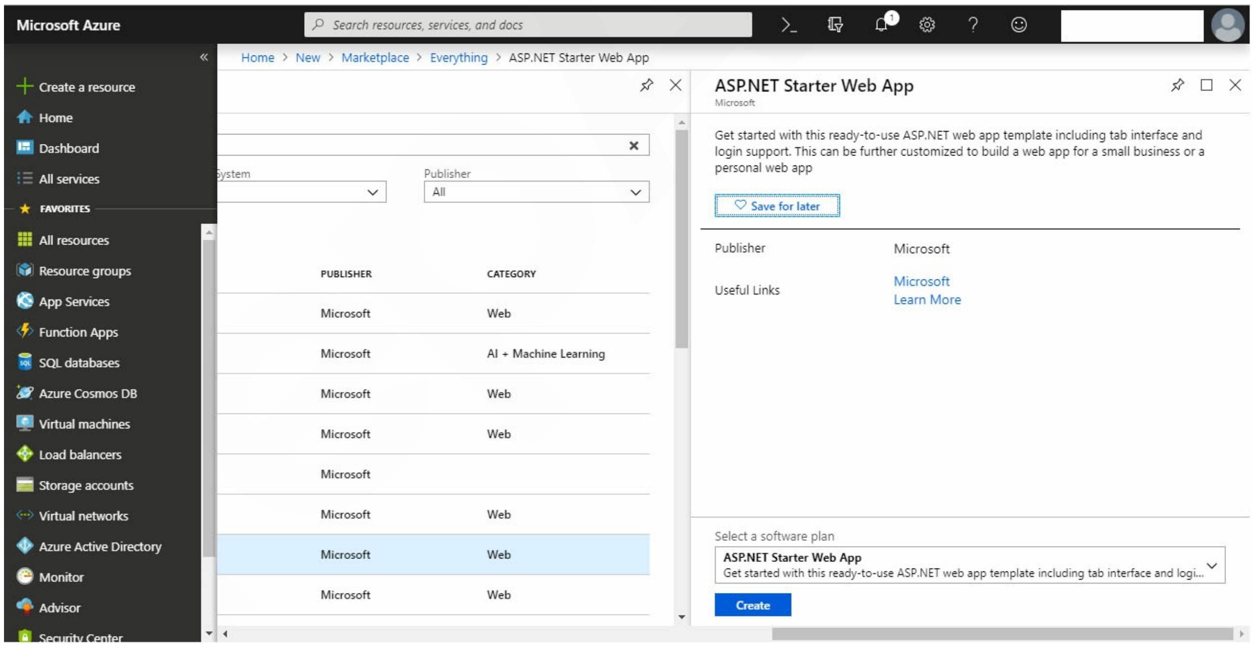
Pricing	Operating System	Publisher
All	All	All

Results

NAME	PUBLISHER	CATEGORY
Web App	Microsoft	Web
Web App Bot	Microsoft	AI + Machine Learning
Web App + SQL	Microsoft	Web
HTML5 Empty Web App	Microsoft	Web
Azure ML Request-Response Service Web App	Microsoft	
Web App + MySQL	Microsoft	Web
ASP.NET Starter Web App	Microsoft	Web
ASP.NET Empty Web App	Microsoft	Web

Figure 2.16

Select **`ASP.NET Starter Web App`**. And click on **`Create`**.



ASP.NET Starter Web App
Microsoft

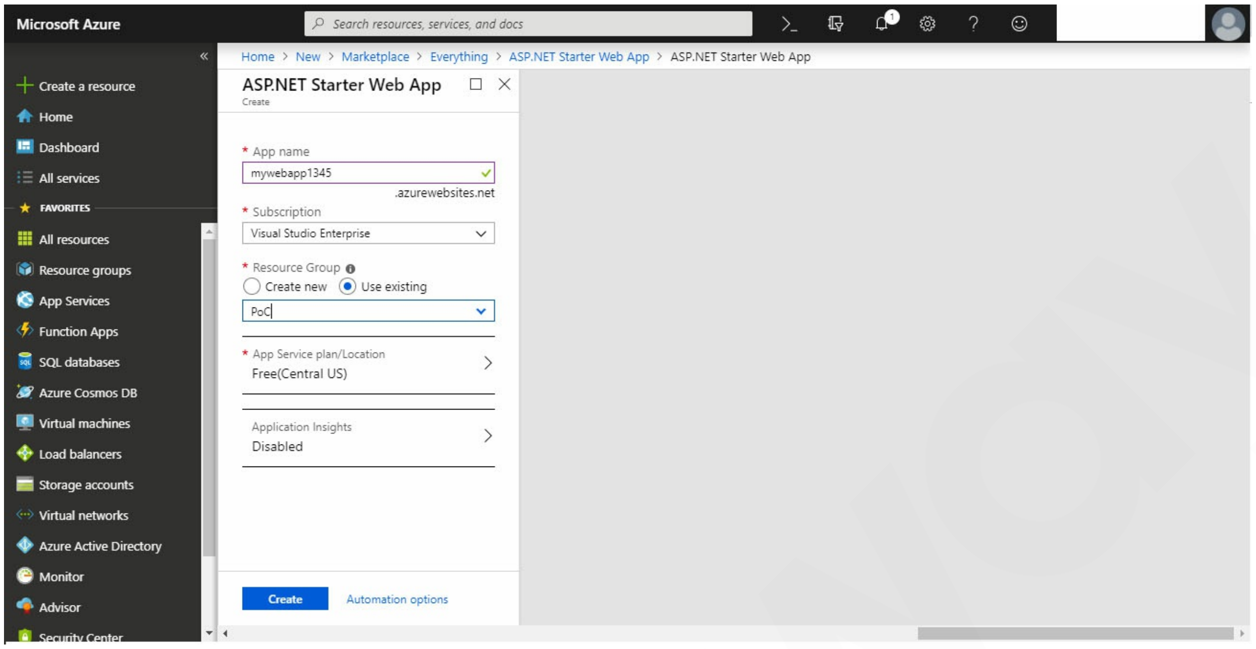
Get started with this ready-to-use ASP.NET web app template including tab interface and login support. This can be further customized to build a web app for a small business or a personal web app

Publisher	Microsoft
Useful Links	Microsoft Learn More

Select a software plan
ASP.NET Starter Web App
Get started with this ready-to-use ASP.NET web app template including tab interface and logi...

Figure 2.17

Our previous web app name was `mywebapp1344`, so I will give its name as `mywebapp1345`. I will select the same resource group as **`PoC`** as the previous one, the free tier of App Service plan. Click on **`Create`**.

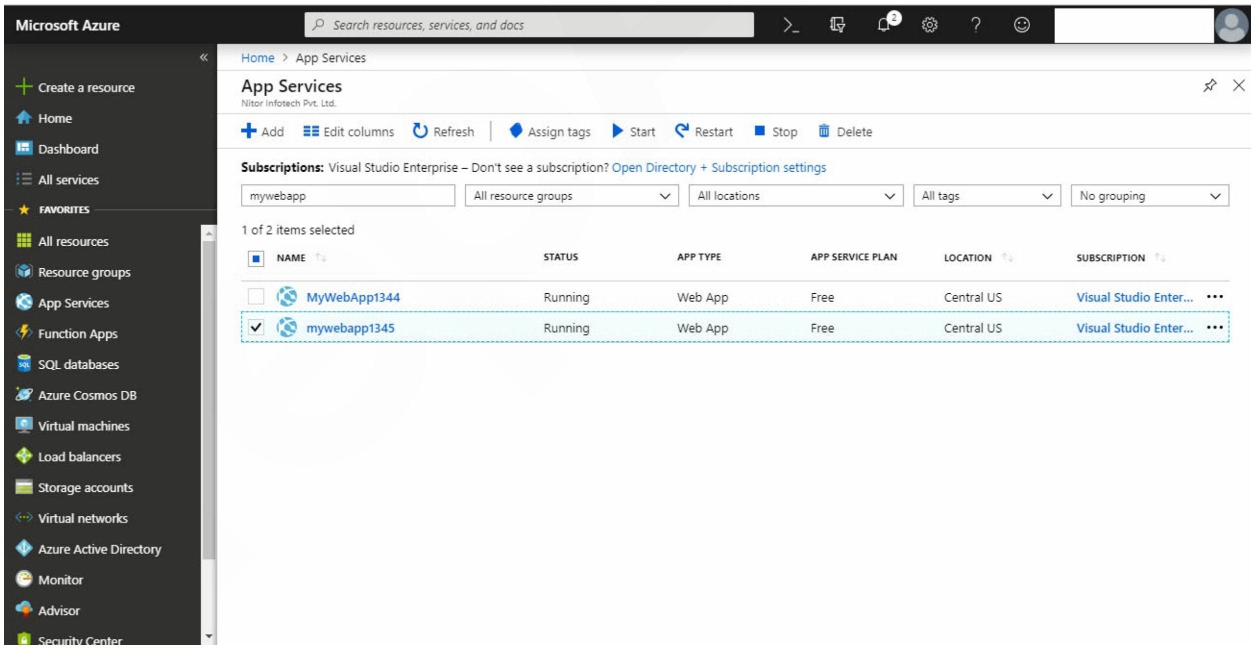


ASP.NET Starter Web App

* App name	mywebapp1345 .azurewebsites.net
* Subscription	Visual Studio Enterprise
* Resource Group	Create new [✓ Use existing]
	PoC
* App Service plan/Location	Free(Central US)
Application Insights	Disabled

Figure 2.18

When the deployment is succeeded, you can check web applications under the **`App Services`** tab.



App Services
Nitor Infotech Pvt. Ltd.

Subscriptions: Visual Studio Enterprise – Don't see a subscription? Open Directory + Subscription settings

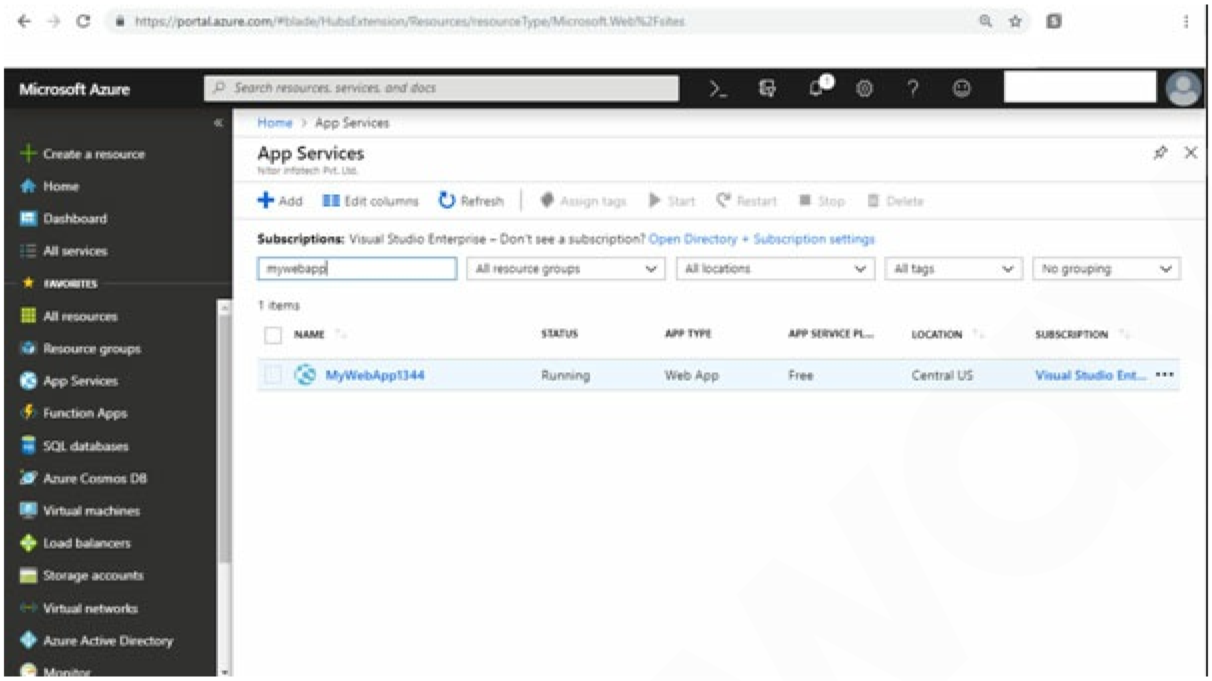
mywebapp | All resource groups | All locations | All tags | No grouping

1 of 2 items selected

NAME	STATUS	APP TYPE	APP SERVICE PLAN	LOCATION	SUBSCRIPTION
MyWebApp1344	Running	Web App	Free	Central US	Visual Studio Enter...
✓ mywebapp1345	Running	Web App	Free	Central US	Visual Studio Enter...

Figure 2.19

Select web application, and you will be redirected to the **`App Service`** overview page. Here you will find URL of web application:



Subscriptions: Visual Studio Enterprise – Don't see a subscription? Open Directory + Subscription settings

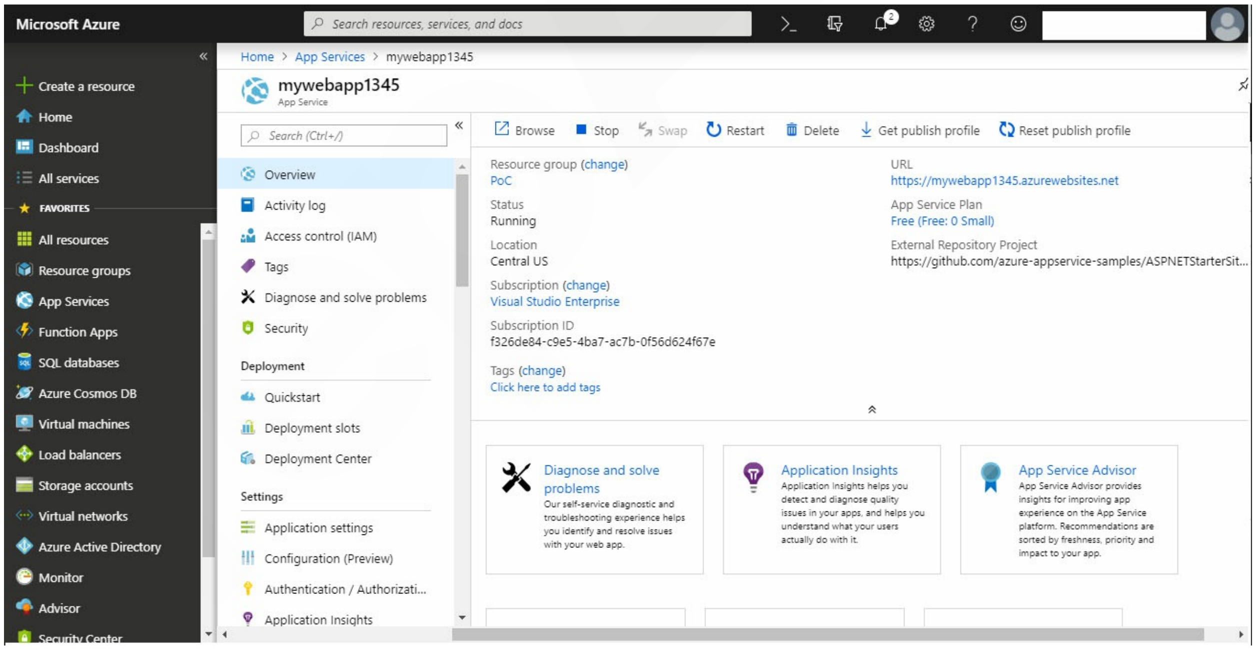
mywebapp | All resource groups | All locations | All tags | No grouping

1 items

NAME	STATUS	APP TYPE	APP SERVICE PL...	LOCATION	SUBSCRIPTION
MyWebApp1344	Running	Web App	Free	Central US	Visual Studio Ent...

Figure 2.20

Navigate to web app URL.



Resource group (change)
PoC

Status
Running

Location
Central US

Subscription (change)
Visual Studio Enterprise

Subscription ID
f326de84-c9e5-4ba7-ac7b-0f56d624f67e

Tags (change)
Click here to add tags

URL
https://mywebapp1345.azurewebsites.net

App Service Plan
Free (Free: 0 Small)

External Repository Project
https://github.com/azure-appservice-samples/ASPNETStarterSit...

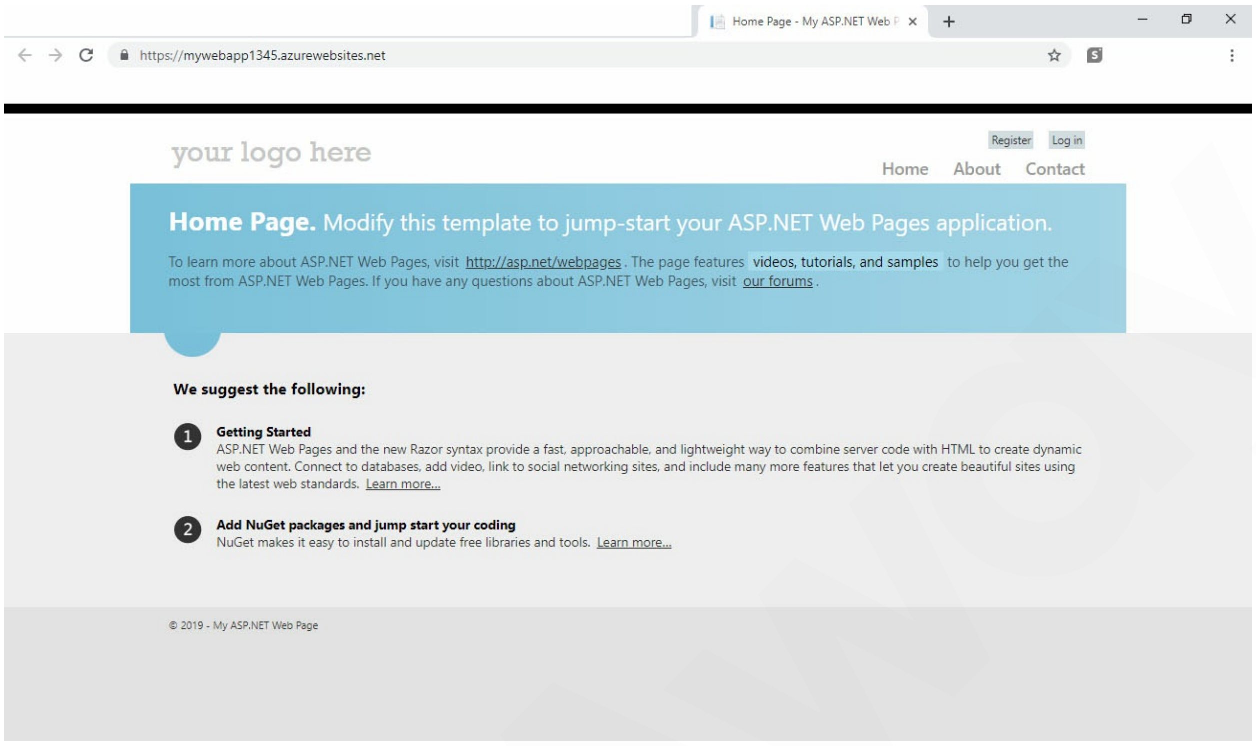
Diagnose and solve problems
Our self-service diagnostic and troubleshooting experience helps you identify and resolve issues with your web app.

Application Insights
Application Insights helps you detect and diagnose quality issues in your apps, and helps you understand what your users actually do with it.

App Service Advisor
App Service Advisor provides insights for improving app experience on the App Service platform. Recommendations are sorted by freshness, priority and impact to your app.

Figure 2.21

As we have selected the ASP.Net starter web application, it has created an application with pre-added views and controllers.



Home Page - My ASP.NET Web P
https://mywebapp1345.azurewebsites.net

your logo here

Register Log in

Home About Contact

Home Page. Modify this template to jump-start your ASP.NET Web Pages application.

To learn more about ASP.NET Web Pages, visit http://asp.net/webpages . The page features videos, tutorials, and samples to help you get the most from ASP.NET Web Pages. If you have any questions about ASP.NET Web Pages, visit our forums .

We suggest the following:

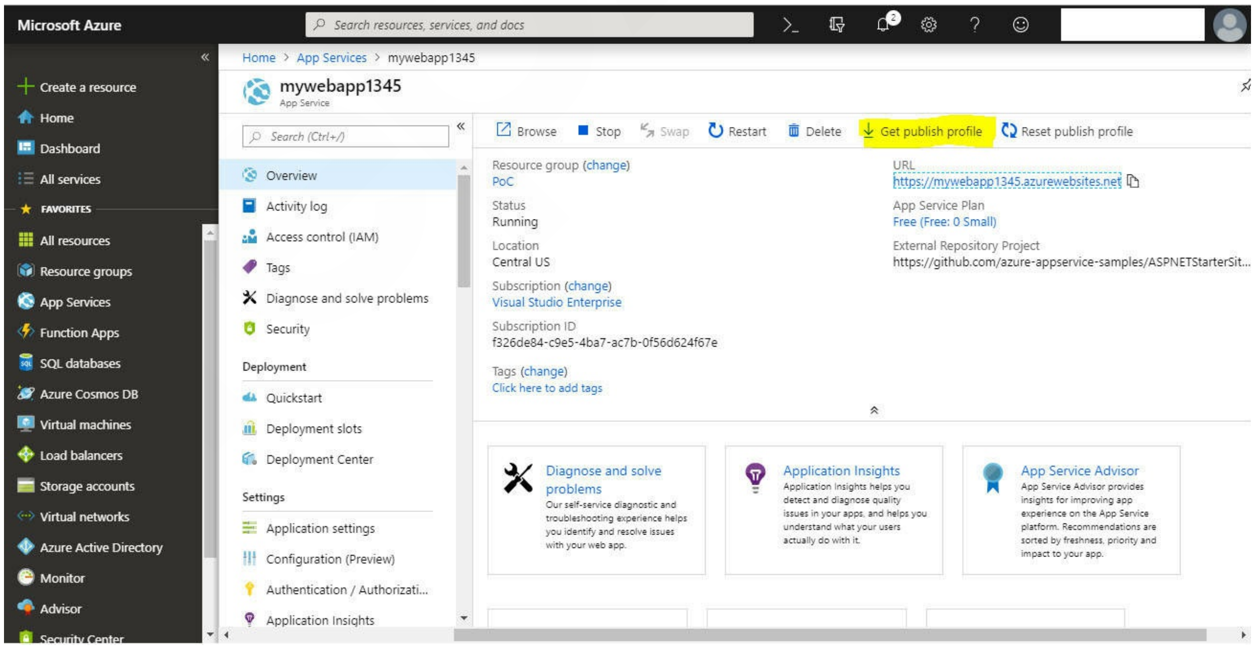
1 **Getting Started**
ASP.NET Web Pages and the new Razor syntax provide a fast, approachable, and lightweight way to combine server code with HTML to create dynamic web content. Connect to databases, add video, link to social networking sites, and include many more features that let you create beautiful sites using the latest web standards. Learn more...

2 **Add NuGet packages and jump start your coding**
NuGet makes it easy to install and update free libraries and tools. Learn more...

© 2019 - My ASP.NET Web Page

Figure 2.22

Now let’s deploy our Visual Studio web application on it. For that, go back to the **`App Service`** overview page. Here you will find **`Get publish profile`** option:



Resource group (change)
PoC

Status
Running

Location
Central US

Subscription (change)
Visual Studio Enterprise

Subscription ID
f326de84-c9e5-4ba7-ac7b-0f56d624f67e

Tags (change)
Click here to add tags

URL
https://mywebapp1345.azurewebsites.net

App Service Plan
Free (Free: 0 Small)

External Repository Project
https://github.com/azure-appservice-samples/ASPNETStarterSit...

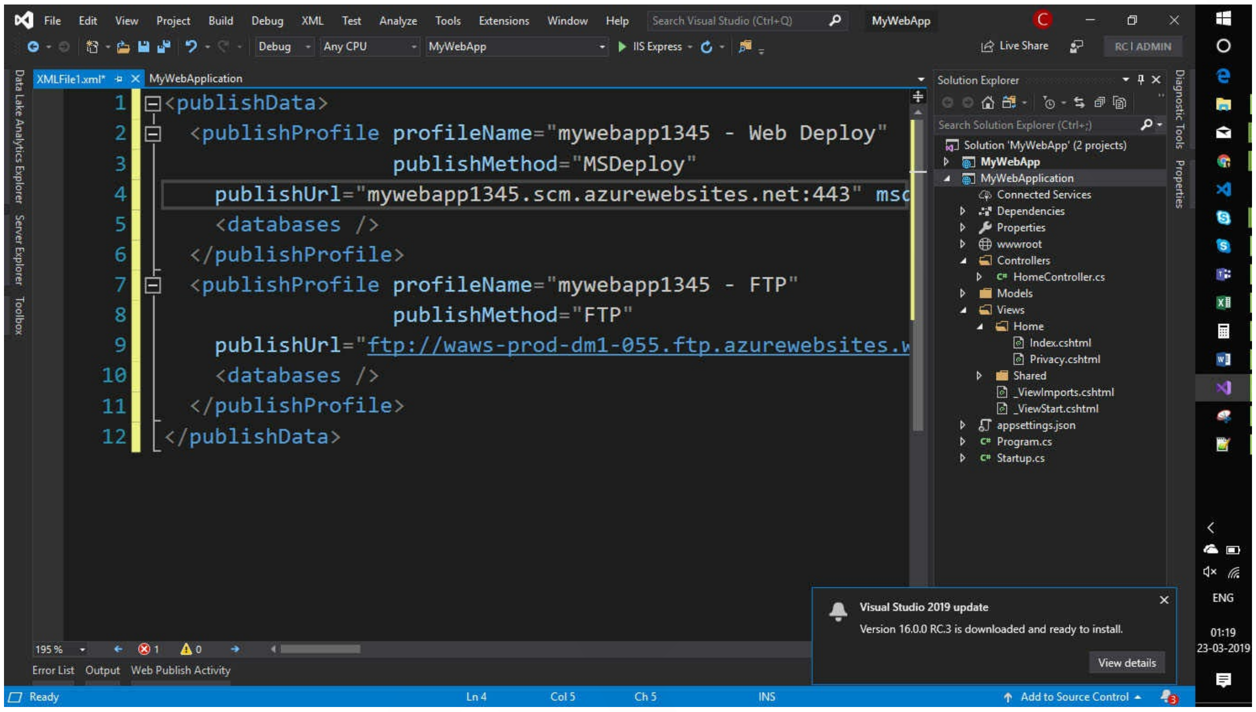
Diagnose and solve problems
Our self-service diagnostic and troubleshooting experience helps you identify and resolve issues with your web app.

Application Insights
Application Insights helps you detect and diagnose quality issues in your apps, and helps you understand what your users actually do with it.

App Service Advisor
App Service Advisor provides insights for improving app experience on the App Service platform. Recommendations are sorted by freshness, priority and impact to your app.

Figure 2.23

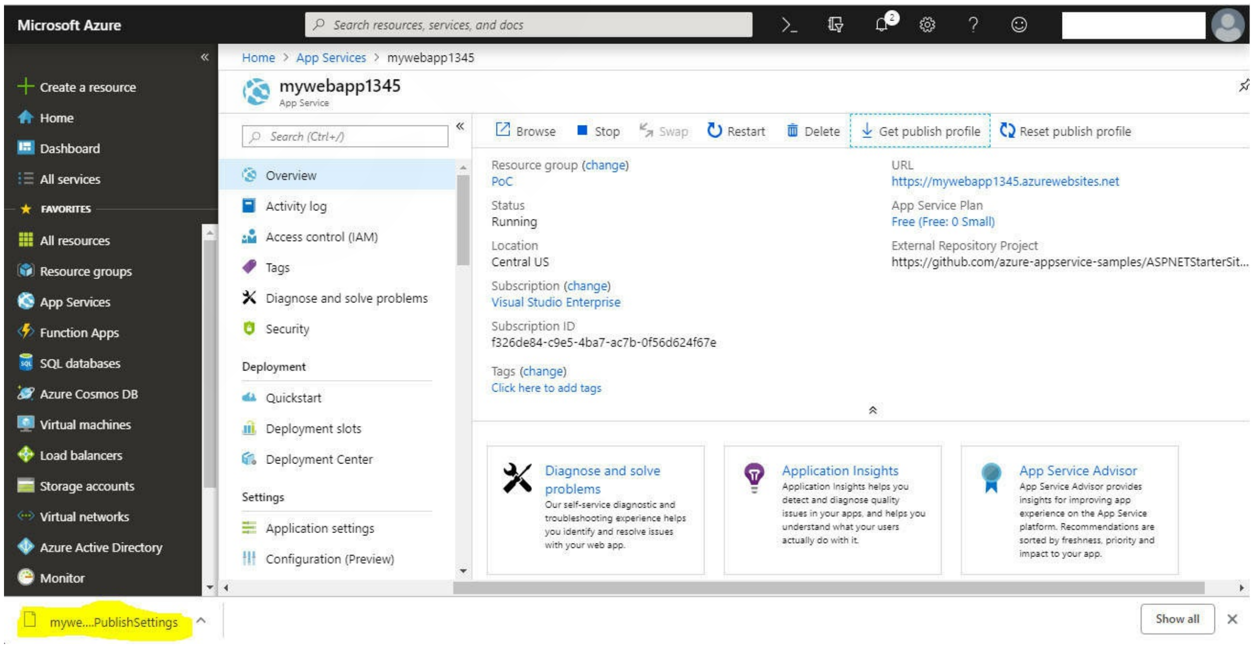
Click on **`Get publish profile`**. It will download publish profile settings of **`App Service`**, a file with a `.publishsettings` file extension, to your local machine. The following code shows a partial example of the file. It contains two publishing profiles that you can use in Visual Studio, one to deploy using Web Deploy, and one to deploy using `FTP`:



```xml
<publishData>
  <publishProfile profileName="mywebapp1345 - Web Deploy"
                  publishMethod="MSDeploy"
    publishUrl="mywebapp1345.scm.azurewebsites.net:443" msc
    <databases />
  </publishProfile>
  <publishProfile profileName="mywebapp1345 - FTP"
                  publishMethod="FTP"
    publishUrl="ftp://waws-prod-dm1-055.ftp.azurewebsites.w
    <databases />
  </publishProfile>
</publishData>
```

Visual Studio 2019 update
Version 16.0.0 RC.3 is downloaded and ready to install.
View details
Solution Explorer
Solution 'MyWebApp' (2 projects)
MyWebApp
MyWebApplication
Ready

Figure 2.24

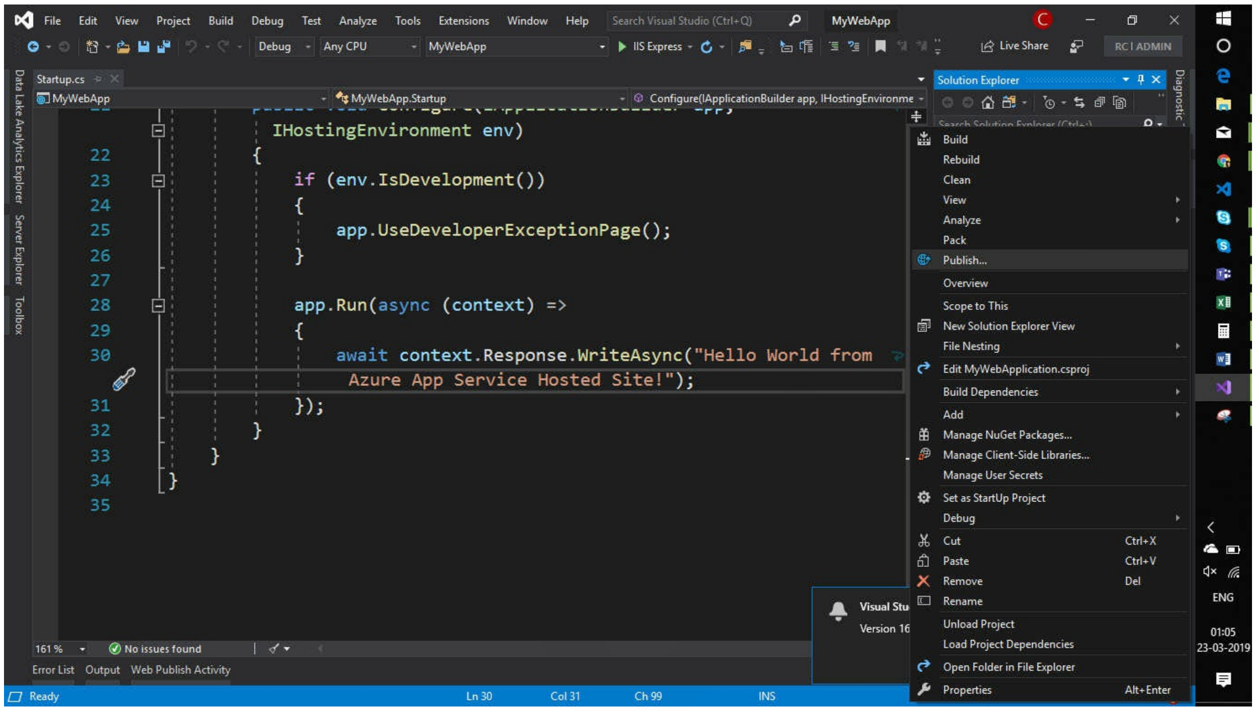


Resource group (change)
PoC
Status
Running
Location
Central US
Subscription (change)
Visual Studio Enterprise
Subscription ID
f326de84-c9e5-4ba7-ac7b-0f56d624f67e
Tags (change)
Click here to add tags

URL
https://mywebapp1345.azurewebsites.net
App Service Plan
Free (Free: 0 Small)
External Repository Project
https://github.com/azure-appservice-samples/ASPNETStarterSit...

Figure 2.25

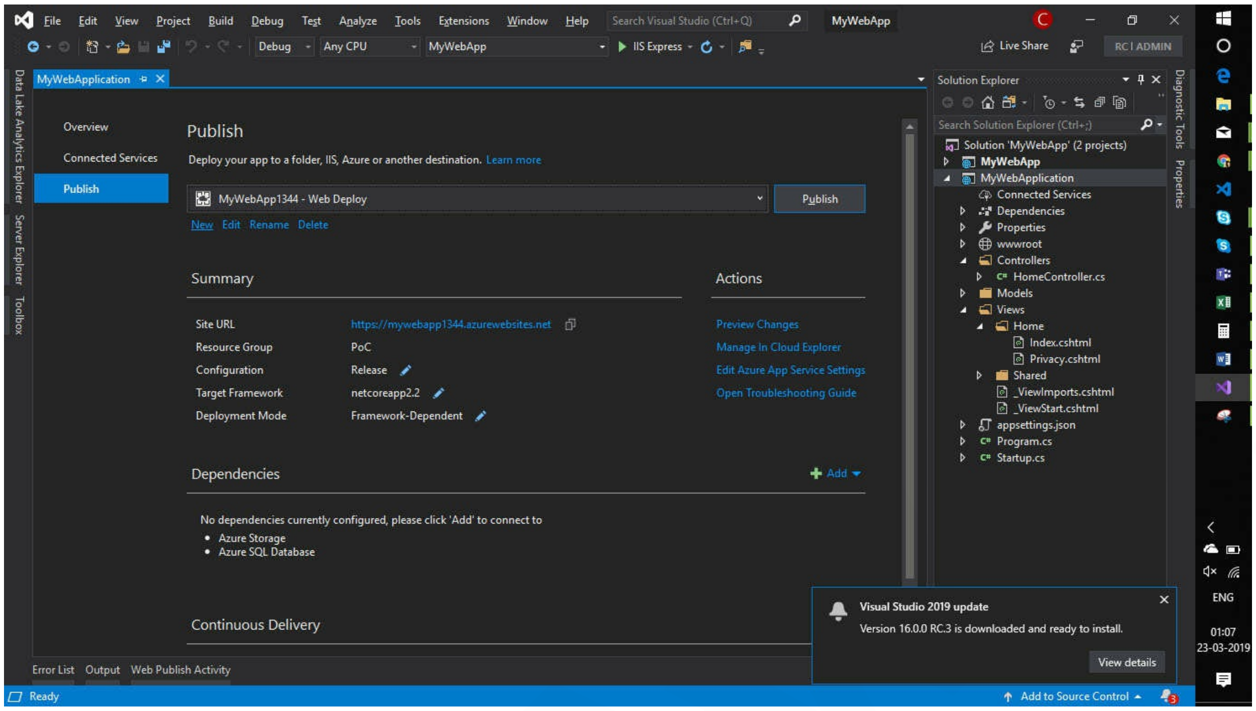
Now let’s go back to Visual Studio. Right-click on the project and select **`Publish`** option.



```
                IHostingEnvironment env)
            {
                if (env.IsDevelopment())
                {
                    app.UseDeveloperExceptionPage();
                }

                app.Run(async (context) =>
                {
                    await context.Response.WriteAsync("Hello World from
                     Azure App Service Hosted Site!");
                });
            }
        }
    }
```

Figure 2.26

It will open a **`Publish`** wizard screen. It will show previously created publish settings of `mywebapp1344`. As we want to deploy code to `mywebapp1345`, click on **`New`**:


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search Visual Studio (Ctrl+Q)
MyWebApp
Debug
Any CPU
IIS Express
Live Share
RC I ADMIN
MyWebApplication
Overview
Connected Services
Publish
Solution Explorer
Error List Output Web Publish Activity
Ready
Add to Source Control

Publish

Deploy your app to a folder, IIS, Azure or another destination. Learn more

MyWebApp1344 - Web Deploy | Publish

New Edit Rename Delete

Summary

Site URL	https://mywebapp1344.azurewebsites.net
Resource Group	PoC
Configuration	Release
Target Framework	netcoreapp2.2
Deployment Mode	Framework-Dependent

Actions

Preview Changes
Manage In Cloud Explorer
Edit Azure App Service Settings
Open Troubleshooting Guide

Dependencies

No dependencies currently configured, please click 'Add' to connect to
- Azure Storage
- Azure SQL Database

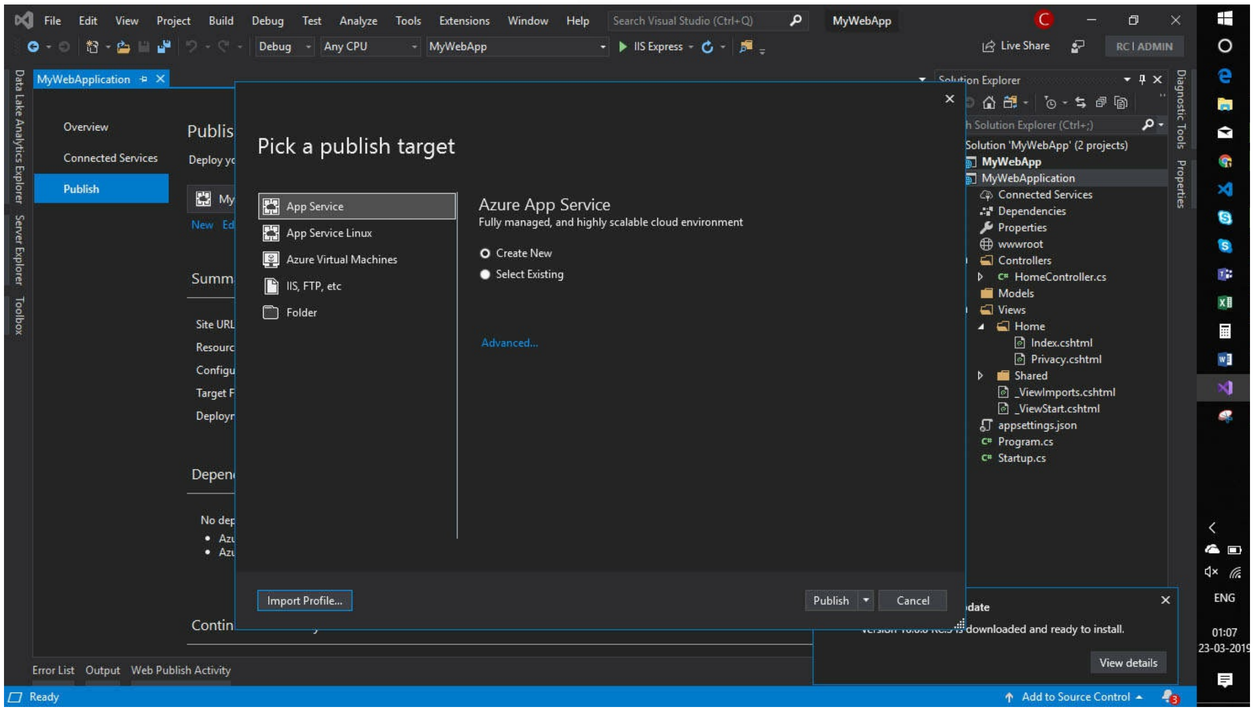
Continuous Delivery

Visual Studio 2019 update
Version 16.0.0 RC.3 is downloaded and ready to install.
View details

Solution 'MyWebApp' (2 projects)
MyWebApp
MyWebApplication
Connected Services
Dependencies
Properties
wwwroot
Controllers
HomeController.cs
Models
Views
Home
Index.cshtml
Privacy.cshtml
Shared
_ViewImports.cshtml
_ViewStart.cshtml
appsettings.json
Program.cs
Startup.cs

Figure 2.27

It will show a popup screen that asks for the publishing target. Select **`Import Profile`**.


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
MyWebApp
Solution Explorer
Ready
Add to Source Control

Pick a publish target

App Service
App Service Linux
Azure Virtual Machines
IIS, FTP, etc
Folder

Azure App Service
Fully managed, and highly scalable cloud environment

Create New
Select Existing

Advanced...

Import Profile... | Publish | Cancel

Figure 2.28

Select and browse publish profiles exported from the Azure site. And click on the **`Open`** button.

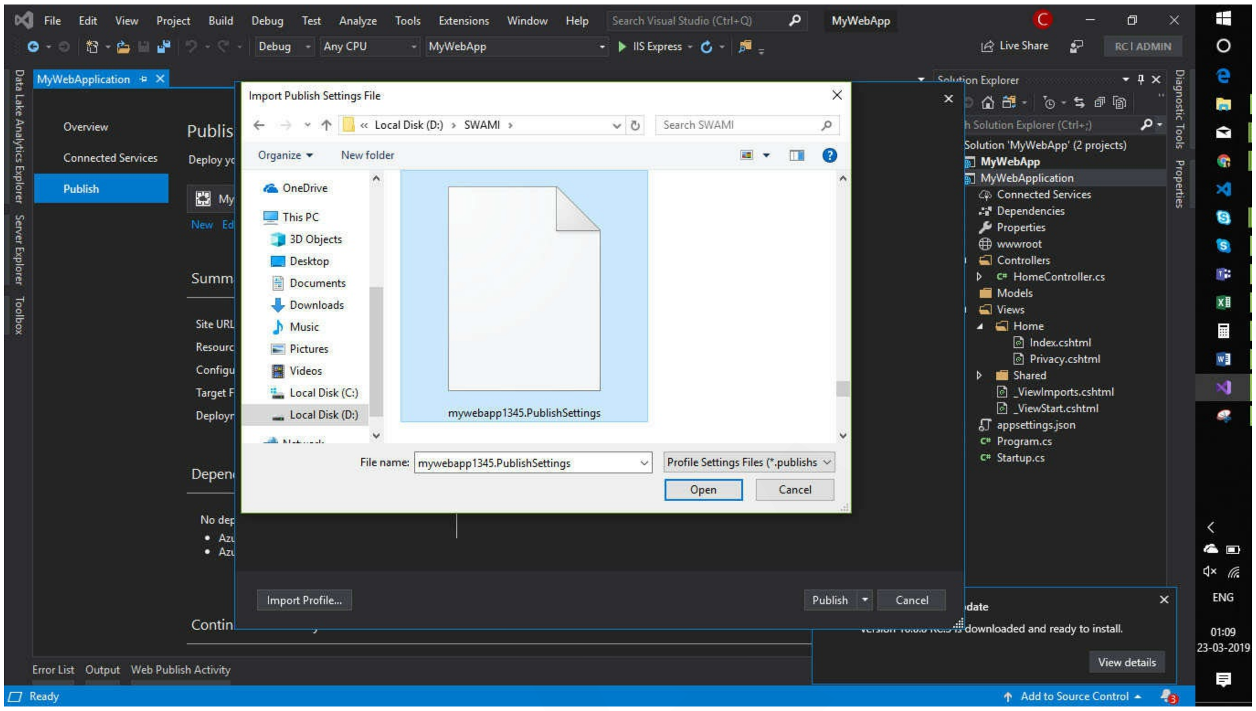
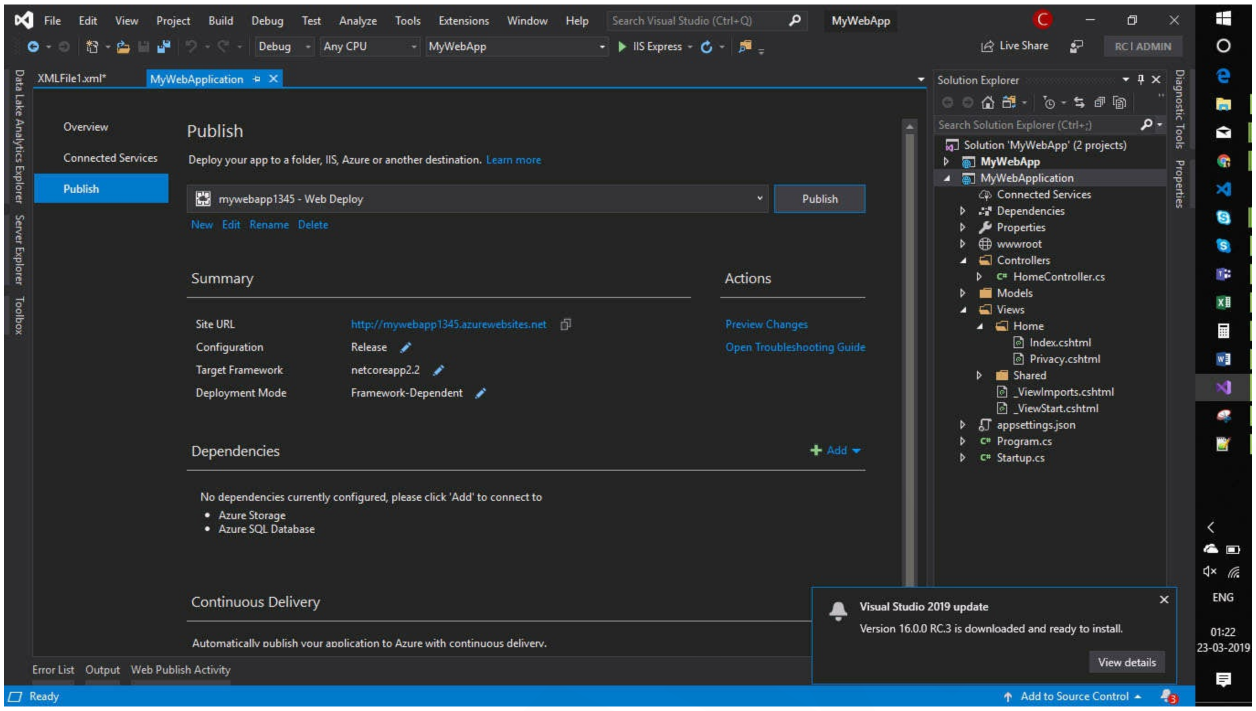


Figure 2.29

It will create a new publish setting in Visual Studio and create a new local publish profile of App `Servicemywebapp1345`. You can check to publish the profile title and site URL for the profile.



Publish

Deploy your app to a folder, IIS, Azure or another destination. Learn more

mywebapp1345 - Web Deploy

Summary

Site URL	http://mywebapp1345.azurewebsites.net
Configuration	Release
Target Framework	netcoreapp2.2
Deployment Mode	Framework-Dependent

Dependencies

No dependencies currently configured, please click 'Add' to connect to
- Azure Storage
- Azure SQL Database

Continuous Delivery

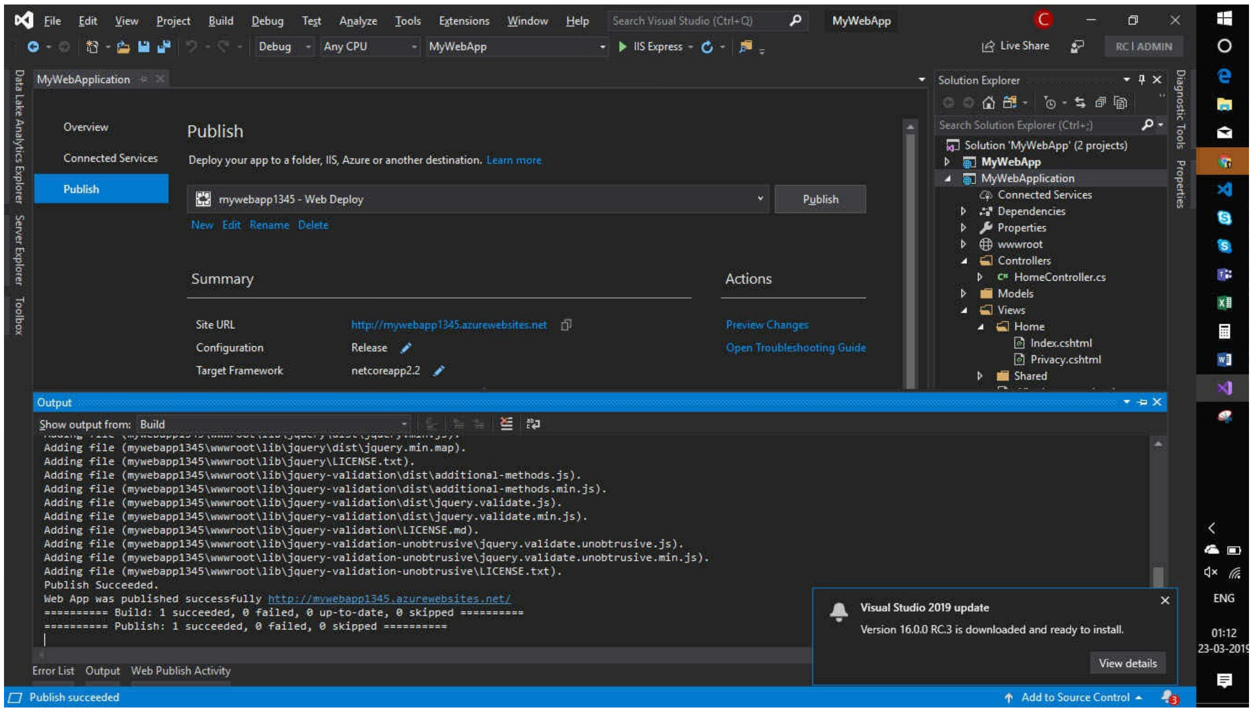
Automatically publish your application to Azure with continuous delivery.

Visual Studio 2019 update

Version 16.0.0 RC.3 is downloaded and ready to install.

Figure 2.30

And start deploying a web application to respective App Service.



```
Adding file (mywebapp1345\wwwroot\lib\jquery\dist\jquery.min.map).
Adding file (mywebapp1345\wwwroot\lib\jquery\LICENSE.txt).
Adding file (mywebapp1345\wwwroot\lib\jquery-validation\dist\additional-methods.js).
Adding file (mywebapp1345\wwwroot\lib\jquery-validation\dist\additional-methods.min.js).
Adding file (mywebapp1345\wwwroot\lib\jquery-validation\dist\jquery.validate.js).
Adding file (mywebapp1345\wwwroot\lib\jquery-validation\dist\jquery.validate.min.js).
Adding file (mywebapp1345\wwwroot\lib\jquery-validation\LICENSE.md).
Adding file (mywebapp1345\wwwroot\lib\jquery-validation-unobtrusive\jquery.validate.unobtrusive.js).
Adding file (mywebapp1345\wwwroot\lib\jquery-validation-unobtrusive\jquery.validate.unobtrusive.min.js).
Adding file (mywebapp1345\wwwroot\lib\jquery-validation-unobtrusive\LICENSE.txt).
Publish Succeeded.
Web App was published successfully http://mywebapp1345.azurewebsites.net/
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

Publish succeeded

Figure 2.31

After successful deployment, let’s go back to our **`App Service`** page anc navigate to **`App Service URL`**. It will open AppService with updated code from our Visual Studio web application same as `mywebapp1344`:

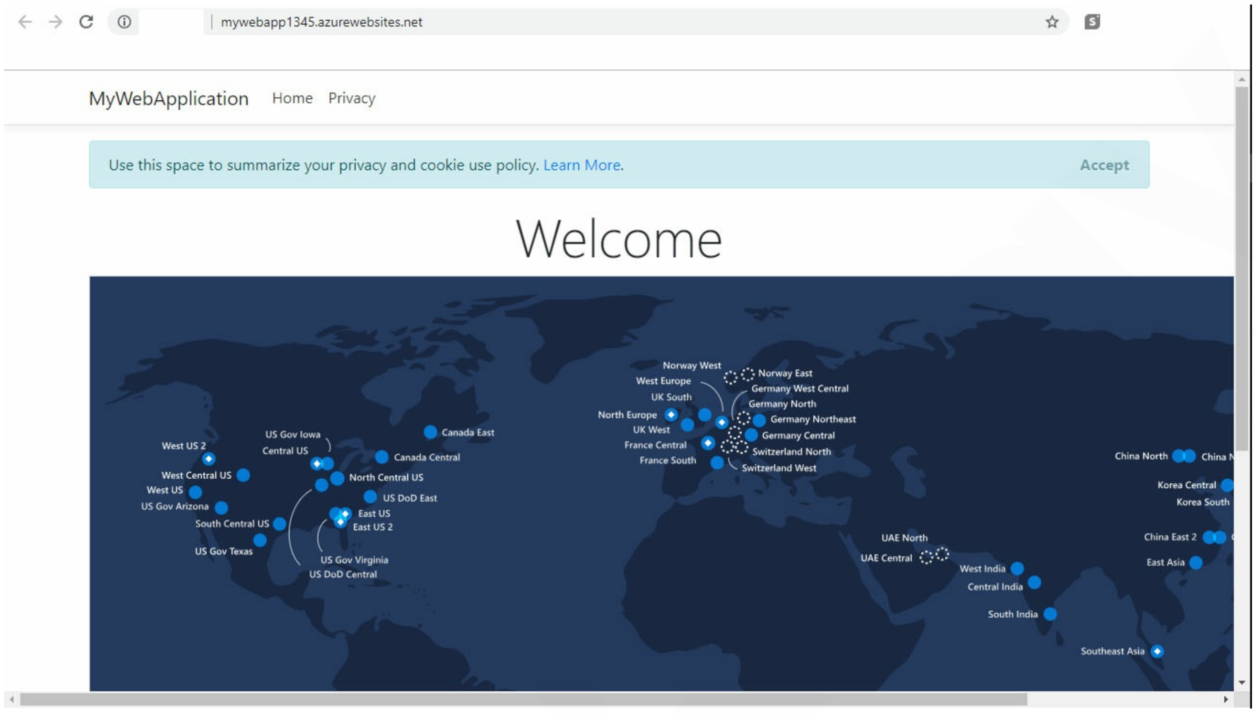


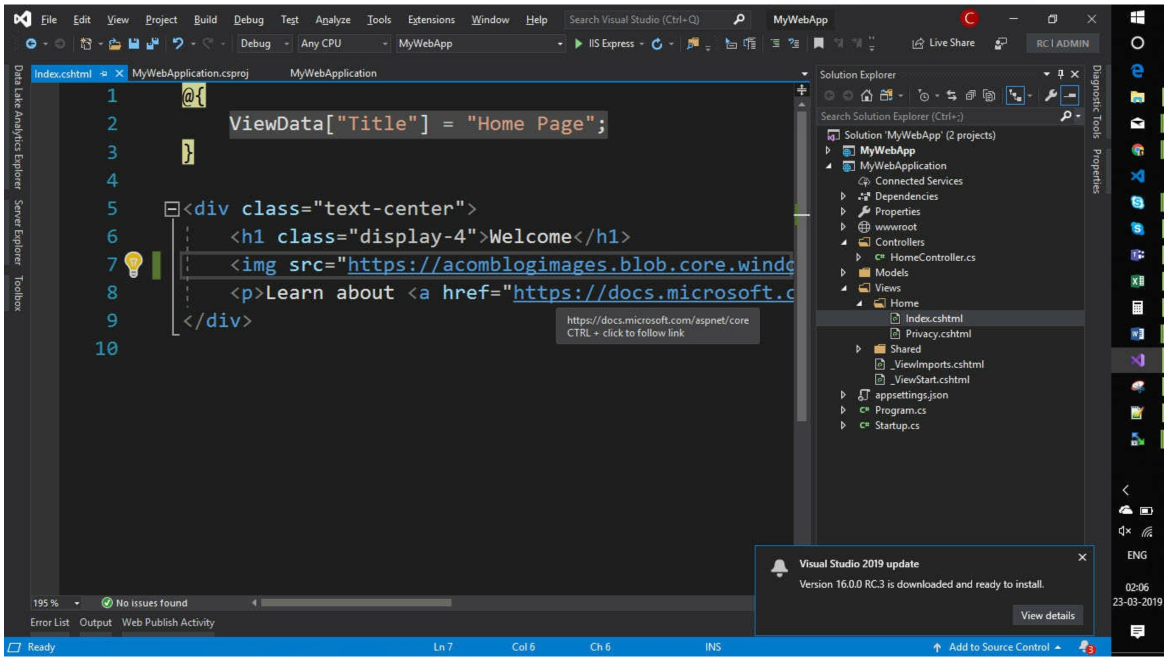
Figure 2.32

Deployment using FTP/s

Whenever you create App Service, FTP/s endpoint for your App Service files is provisioned by default, no additional settings are required.

1. Go to Visual Studio and open `Index.cshtml` file. And update image source as follows:

```
Welcome.
<img
src=”https://acomblogimages.blob.core.windows.net/media/Defau
/>
```



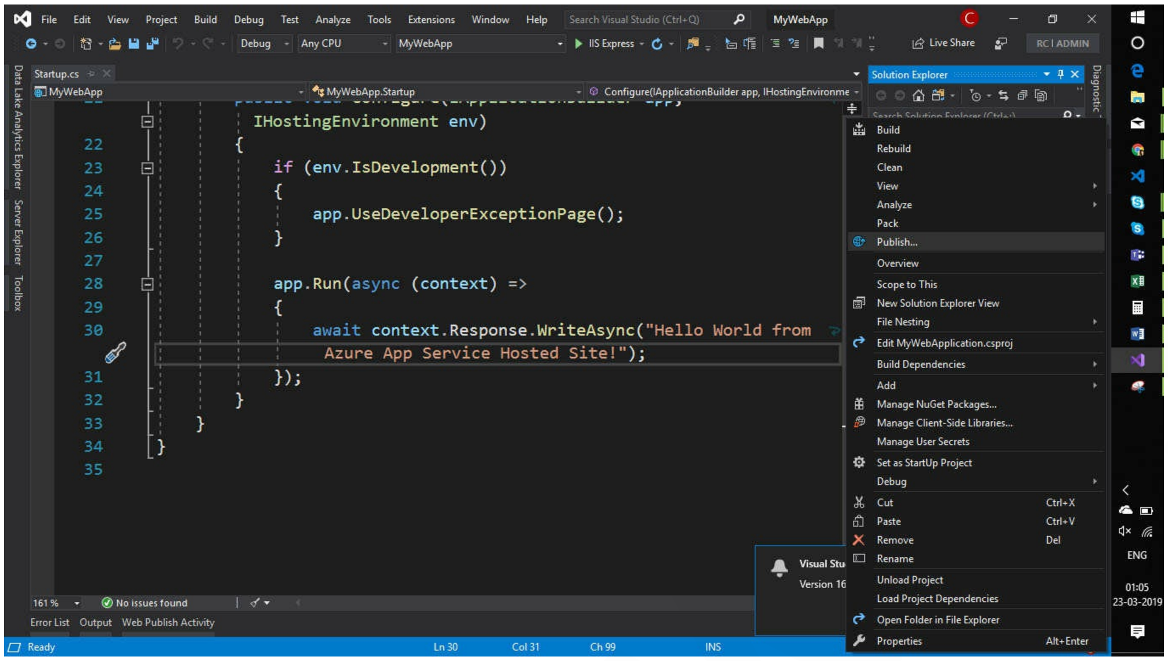
```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <img src="https://acomblogimages.blob.core.windo
    <p>Learn about <a href="https://docs.microsoft.c
</div>
```

https://docs.microsoft.com/aspnet/core
CTRL + click to follow link

Visual Studio 2019 update
Version 16.0.0 RC.3 is downloaded and ready to install.
View details

Figure 2.33

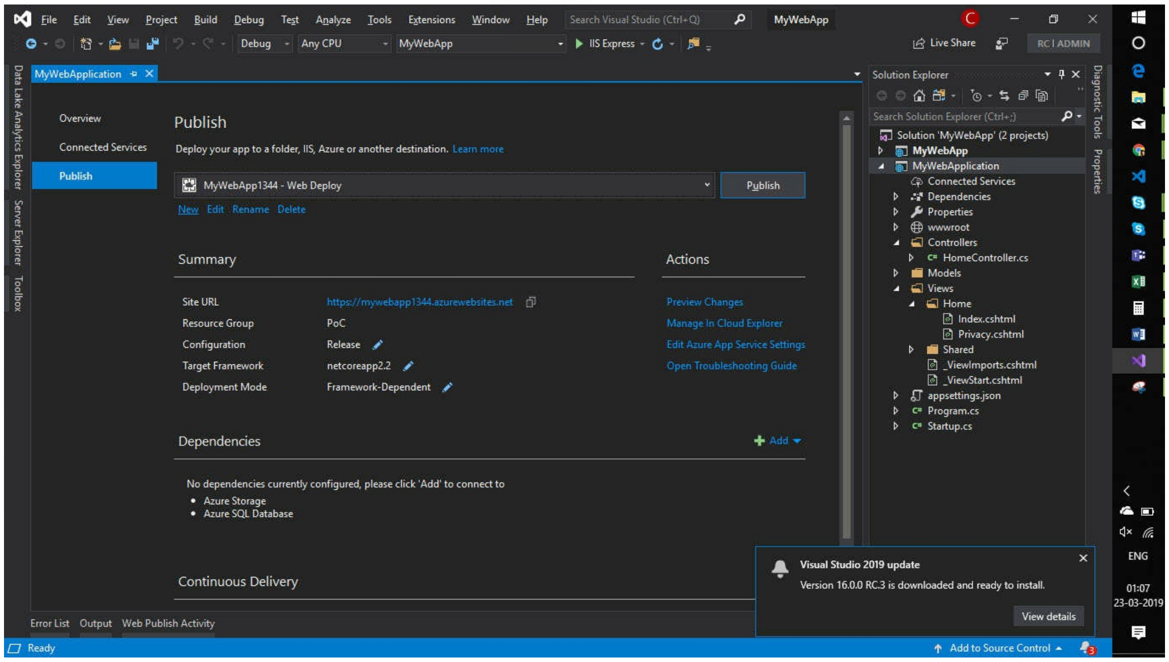
2. Now let’s go back to Visual Studio. Then right-click on the project and select **`Publish`** option.



```
    IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World from
         Azure App Service Hosted Site!");
    });
}
}
}
```

Figure 2.34

3. It will open a **Publish** wizard screen. It will show previously created publish settings of `mywebapp1344`. As we want to deploy code to `mywebapp1345`, click on **New**.



Publish

Deploy your app to a folder, IIS, Azure or another destination. Learn more

MyWebApp1344 - Web Deploy   Publish

New Edit Rename Delete

Summary

Site URL	https://mywebapp1344.azurewebsites.net
Resource Group	PoC
Configuration	Release
Target Framework	netcoreapp2.2
Deployment Mode	Framework-Dependent

Actions

Preview Changes
Manage In Cloud Explorer
Edit Azure App Service Settings
Open Troubleshooting Guide

Dependencies

No dependencies currently configured, please click 'Add' to connect to
- Azure Storage
- Azure SQL Database

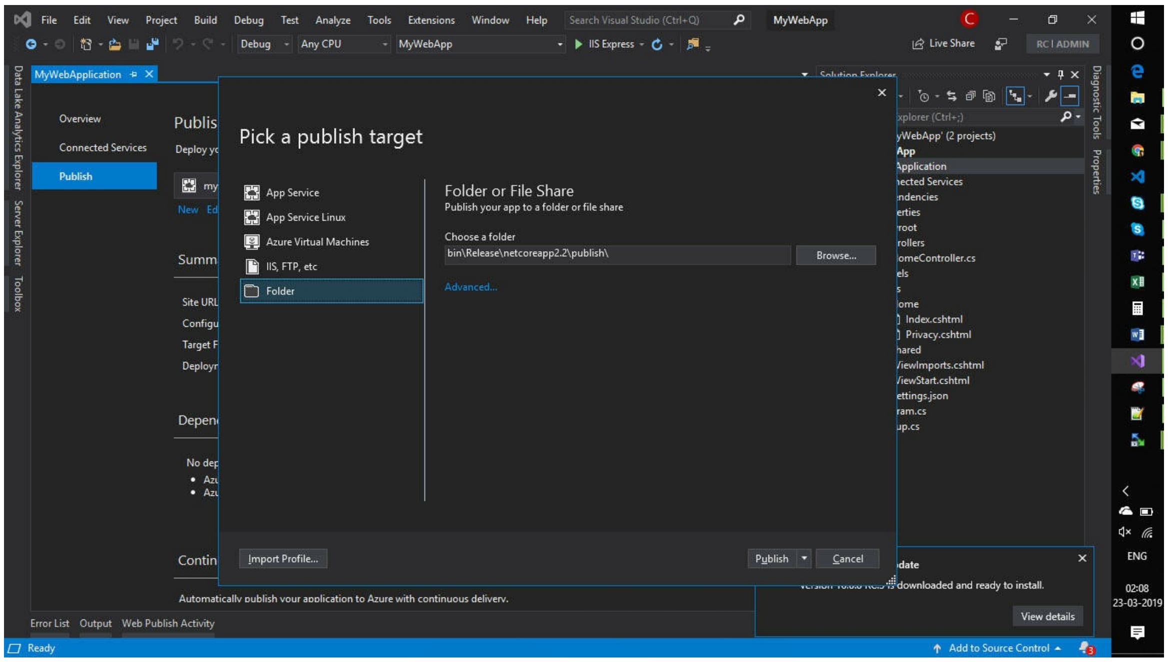
Continuous Delivery

Visual Studio 2019 update
Version 16.0.0 RC.3 is downloaded and ready to install.

Figure 2.35

4. It will show the popup screen, which asks for the publishing target. Select the **Folder** option and choose folder location where build files to

be placed. And click on **`Publish`**.



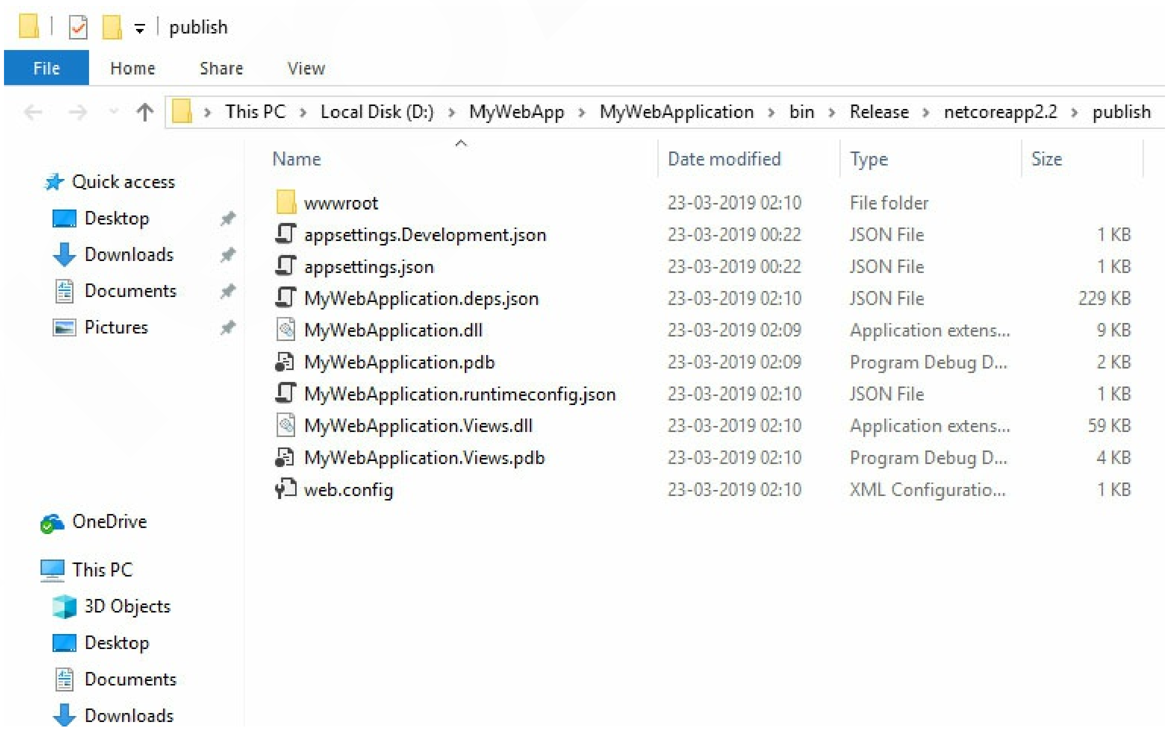
Folder or File Share
Publish your app to a folder or file share

Choose a folder
bin\Release\netcoreapp2.2\publish\
Browse...
Advanced...

Figure 2.36

5. After a successful build, files will be published at the selected location. In our case, it is:

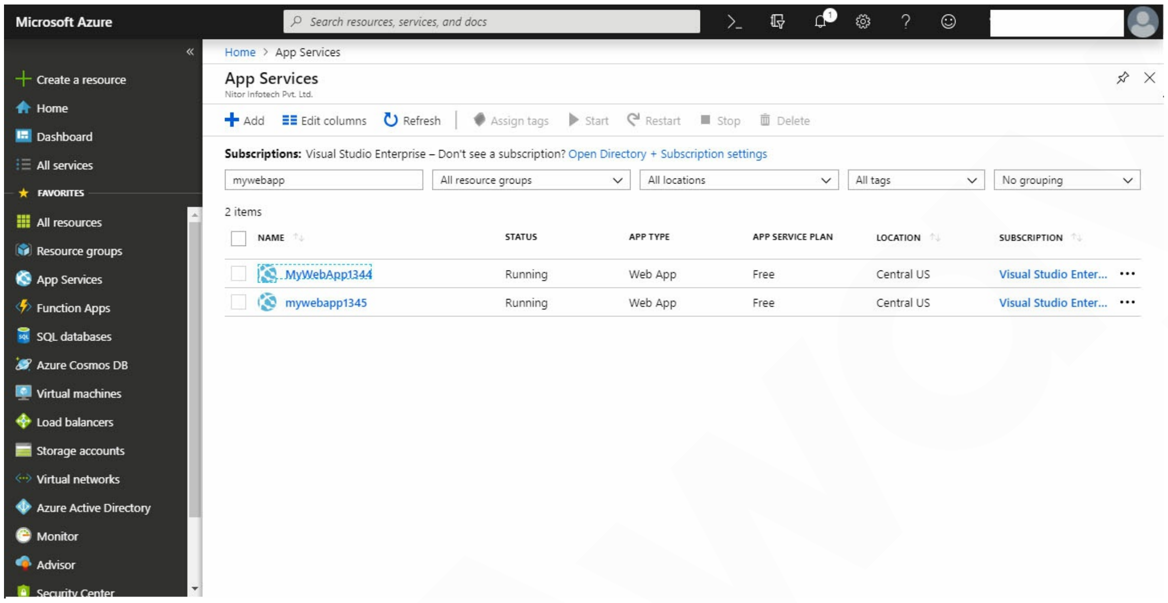
```
D:\MyWebApp\MyWebApplication\bin\Release\netcoreapp2.2\publis
```



Name	Date modified	Type	Size
wwwroot	23-03-2019 02:10	File folder	
appsettings.Development.json	23-03-2019 00:22	JSON File	1 KB
appsettings.json	23-03-2019 00:22	JSON File	1 KB
MyWebApplication.deps.json	23-03-2019 02:10	JSON File	229 KB
MyWebApplication.dll	23-03-2019 02:09	Application extens...	9 KB
MyWebApplication.pdb	23-03-2019 02:09	Program Debug D...	2 KB
MyWebApplication.runtimeconfig.json	23-03-2019 02:10	JSON File	1 KB
MyWebApplication.Views.dll	23-03-2019 02:10	Application extens...	59 KB
MyWebApplication.Views.pdb	23-03-2019 02:10	Program Debug D...	4 KB
web.config	23-03-2019 02:10	XML Configuratio...	1 KB

Figure 2.37

6. Let's go to `mywebapp1344` **`App Service`**:



Subscriptions: Visual Studio Enterprise – Don't see a subscription? Open Directory + Subscription settings

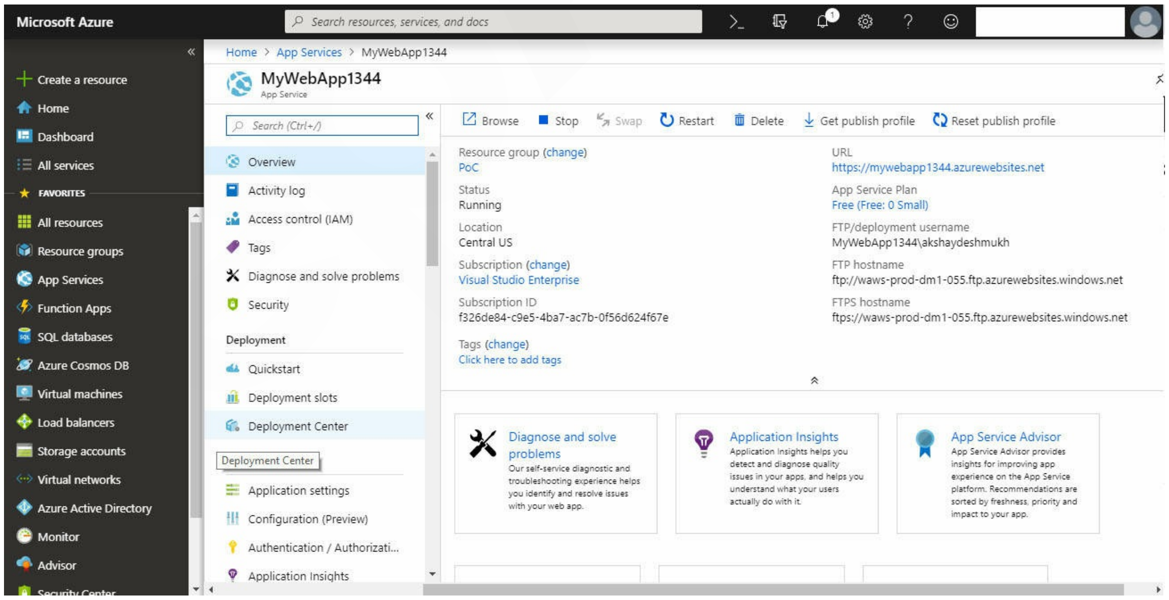
mywebapp | All resource groups | All locations | All tags | No grouping

2 items

NAME	STATUS	APP TYPE	APP SERVICE PLAN	LOCATION	SUBSCRIPTION
MyWebApp1344	Running	Web App	Free	Central US	Visual Studio Enter...
mywebapp1345	Running	Web App	Free	Central US	Visual Studio Enter...

Figure 2.38

7. Select the deployment center from the left-hand side menu in the context of **`App Service`**.



Resource group (change)
PoC

Status
Running

Location
Central US

Subscription (change)
Visual Studio Enterprise

Subscription ID
f326de84-c9e5-4ba7-ac7b-0f56d624f67e

Tags (change)
Click here to add tags

URL
https://mywebapp1344.azurewebsites.net

App Service Plan
Free (Free: 0 Small)

FTP/deployment username
MyWebApp1344\akshaydeshmukh

FTP hostname
ftp://waws-prod-dm1-055.ftp.azurewebsites.windows.net

FTPS hostname
ftps://waws-prod-dm1-055.ftp.azurewebsites.windows.net

Diagnose and solve problems
Our self-service diagnostic and troubleshooting experience helps you identify and resolve issues with your web app.

Application Insights
Application Insights helps you detect and diagnose quality issues in your apps, and helps you understand what your users actually do with it.

App Service Advisor
App Service Advisor provides insights for improving app experience on the App Service platform. Recommendations are sorted by freshness, priority and impact to your app.

Figure 2.39

8. **`Deployment Centre`** provides options to choose code location and build deployment strategy. As we are going to deploy files to FTP/s location of App Service, scroll down and select the **`FTP`** option. And click the

`Dashboard` button.

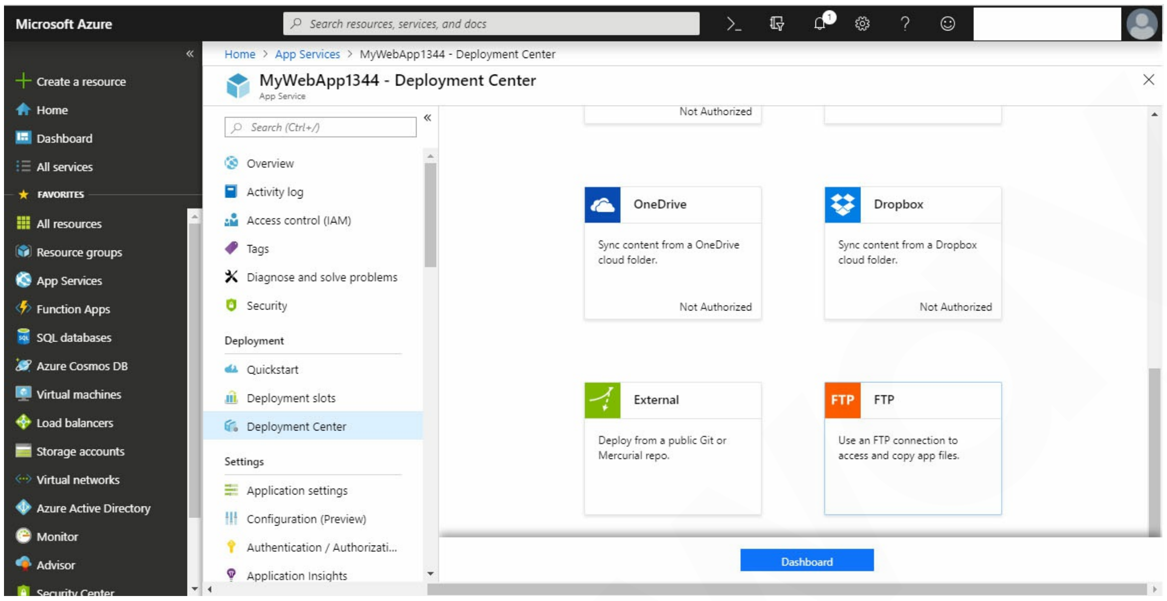
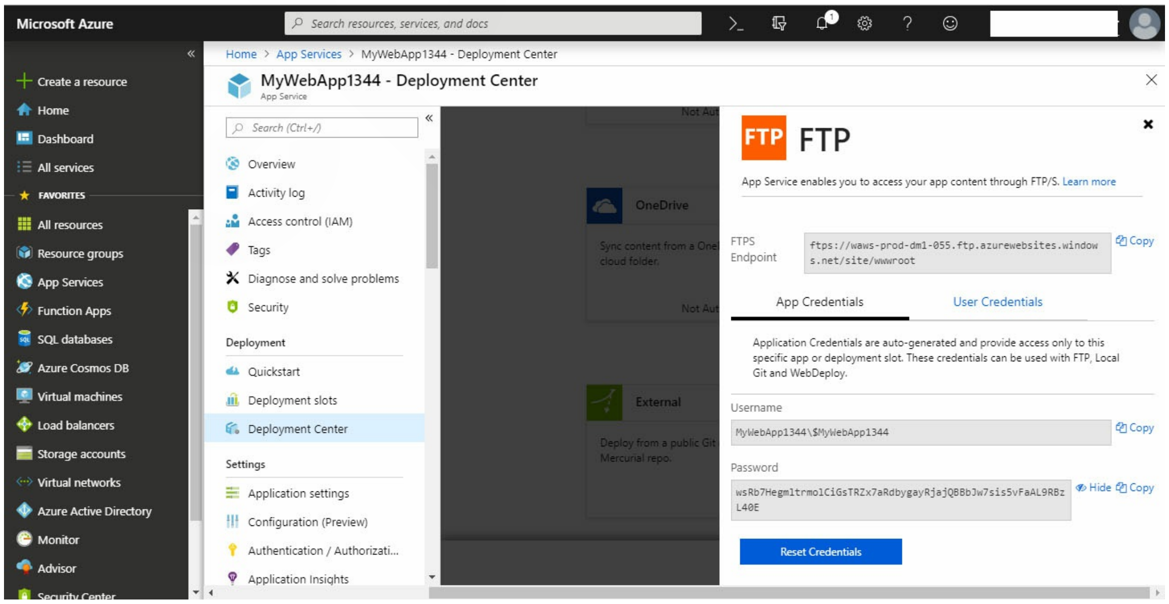

Microsoft Azure
Home > App Services > MyWebApp1344 - Deployment Center
MyWebApp1344 - Deployment Center
OneDrive
Sync content from a OneDrive cloud folder.
Not Authorized
Dropbox
Sync content from a Dropbox cloud folder.
Not Authorized
External
Deploy from a public Git or Mercurial repo.
FTP
Use an FTP connection to access and copy app files.
Dashboard

Figure 2.40

9. It will popup FTP details. It displays an FTP endpoint and username. To view **`Password`**, click on the **`Show`** button beside **`Password`** textbox. We are going to use details from the **`App Credentials`** tab. Application credentials are auto-generated and provide access only to this specific app or deployment slot. These credentials can be used with **`FTP, Local Git`**, and **`Web Deploy`**.



FTP

App Service enables you to access your app content through FTP/S. Learn more

FTPS Endpoint: ftps://waws-prod-dm1-055.ftp.azurewebsites.windows.net/site/wwwroot

App Credentials | User Credentials

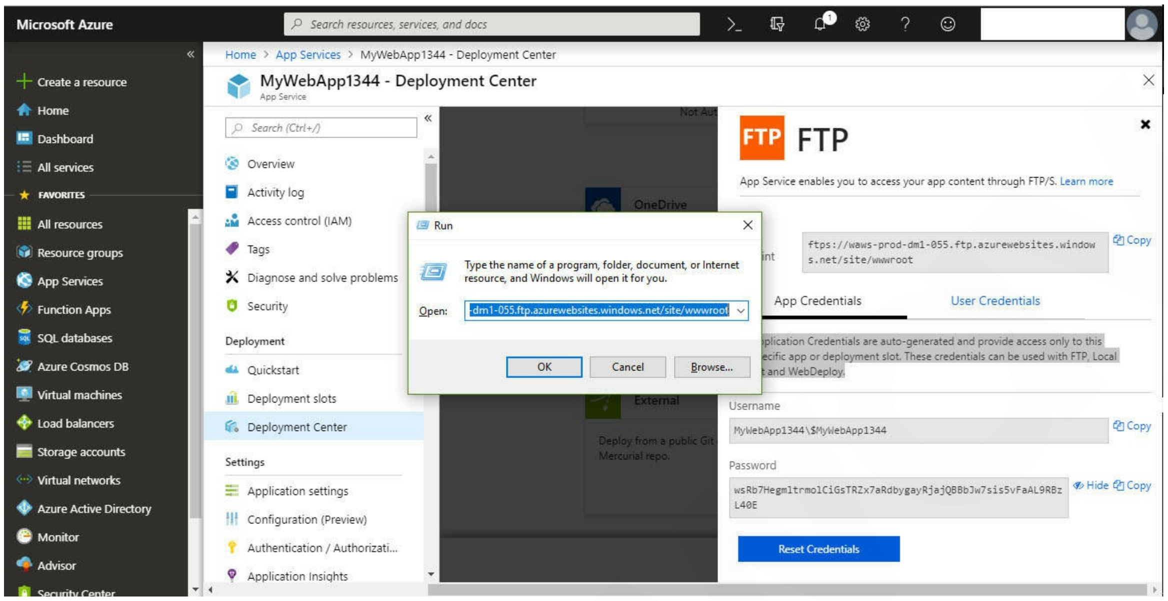
Application Credentials are auto-generated and provide access only to this specific app or deployment slot. These credentials can be used with FTP, Local Git and WebDeploy.

Username: MyWebApp1344\$MyWebApp1344

Password: wsRb7Hegm1trmolCiGsTRZx7aRdbygayRjajQBBbJw7sis5vFaAL9RBzL40E

Figure 2.41

10. Open **`Run`** shell and **`Open`** FTP location.



App Service enables you to access your app content through FTP/S. Learn more

ftps://waws-prod-dm1-055.ftp.azurewebsites.windows.net/site/wwwroot

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: -dm1-055.ftp.azurewebsites.windows.net/site/wwwroot

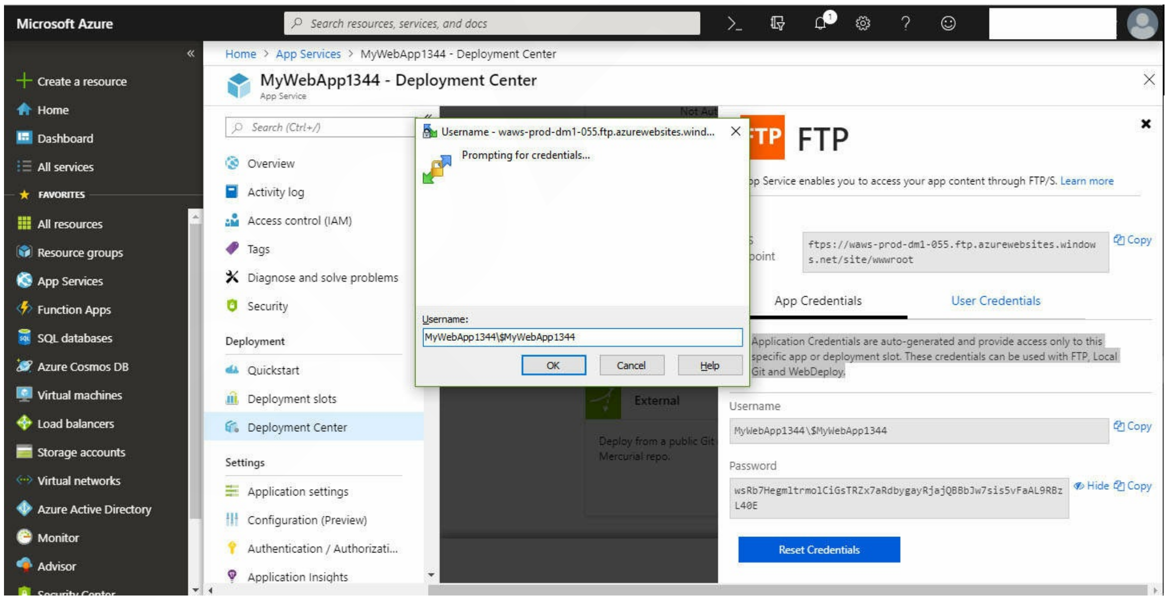
Application Credentials are auto-generated and provide access only to this specific app or deployment slot. These credentials can be used with FTP, Local Git and WebDeploy.

Username: MyWebApp1344\$MyWebApp1344

Password: wsRb7Hegm1trmolCiGsTRZx7aRdbygayRjajQBBbJw7sis5vFaAL9RBzL40E

Figure 2.42

11. It will prompt for credentials.



Username - waws-prod-dm1-055.ftp.azurewebsites.wind...

Prompting for credentials...

Username: MyWebApp1344\$MyWebApp1344

ftps://waws-prod-dm1-055.ftp.azurewebsites.windows.net/site/wwwroot

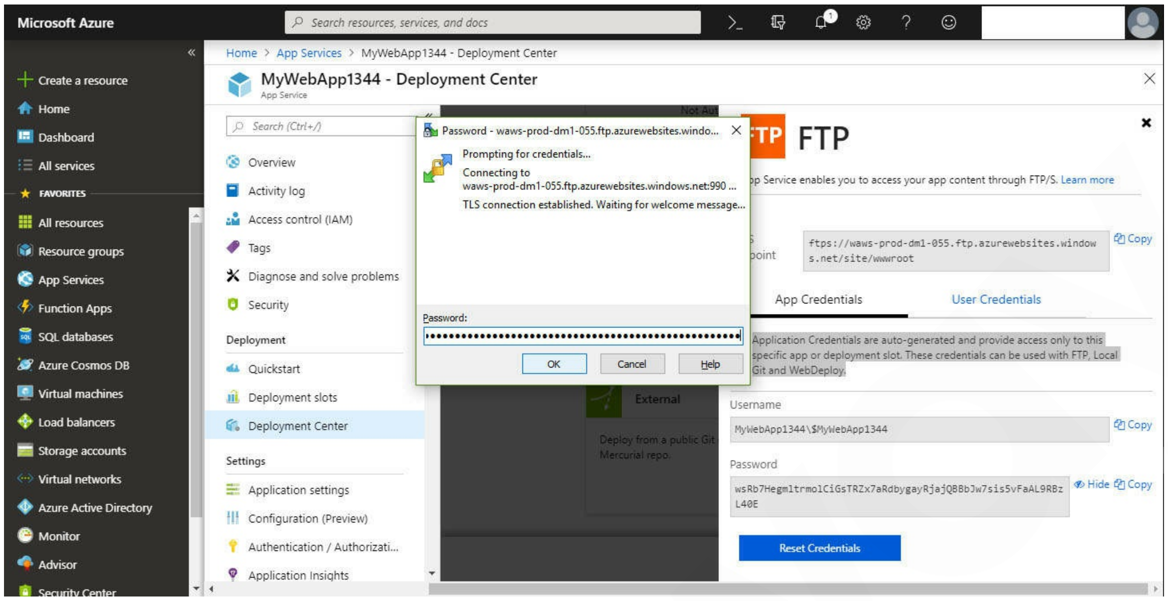
Application Credentials are auto-generated and provide access only to this specific app or deployment slot. These credentials can be used with FTP, Local Git and WebDeploy.

Username: MyWebApp1344\$MyWebApp1344

Password: wsRb7Hegm1trmolCiGsTRZx7aRdbygayRjajQBBbJw7sis5vFaAL9RBzL40E

Figure 2.43

12. Take a look at the following screenshot:



Password - waws-prod-dm1-055.ftp.azurewebsites.windo...

Prompting for credentials...
Connecting to waws-prod-dm1-055.ftp.azurewebsites.windows.net:990 ...
TLS connection established. Waiting for welcome message...

Password: ••

App Service enables you to access your app content through FTP/S. Learn more

FTPS endpoint: ftps://waws-prod-dm1-055.ftp.azurewebsites.windows.net/site/wwwroot

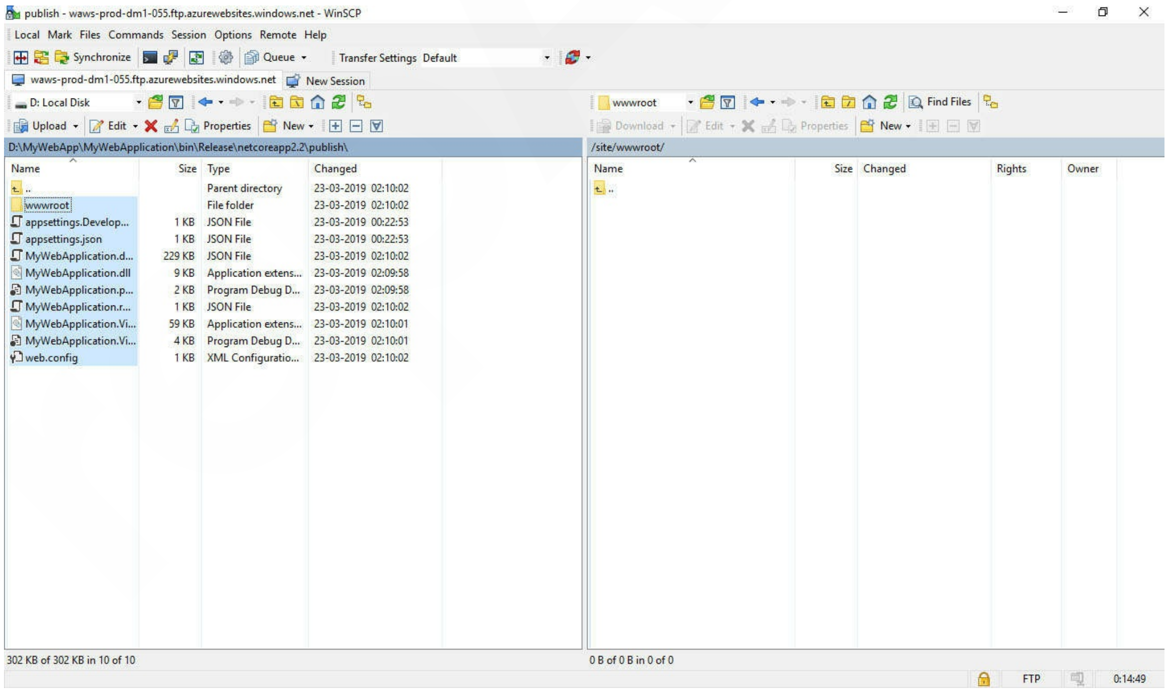
Application Credentials are auto-generated and provide access only to this specific app or deployment slot. These credentials can be used with FTP, Local Git and WebDeploy.

Username: MyWebApp1344\$MyWebApp1344

Password: wsRb7HegmltrmolCiGsTRZx7aRdbygayRjajQBBbJw7sis5vFaAL9RBzL40E

Figure 2.44

13. SCP will open your local machine folders and FTP location folders. Select publish files folder at the left-hand side, which is a local machine and `wwwroot` on the right-hand side.



publish - waws-prod-dm1-055.ftp.azurewebsites.windows.net - WinSCP

waws-prod-dm1-055.ftp.azurewebsites.windows.net

D:\MyWebApp\MyWebApplication\bin\Release\netcoreapp2.2\publish\

Name	Size	Type	Changed
..		Parent directory	23-03-2019 02:10:02
wwwroot		File folder	23-03-2019 02:10:02
appsettings.Develop...	1 KB	JSON File	23-03-2019 00:22:53
appsettings.json	1 KB	JSON File	23-03-2019 00:22:53
MyWebApplication.d...	229 KB	JSON File	23-03-2019 02:10:02
MyWebApplication.dll	9 KB	Application extens...	23-03-2019 02:09:58
MyWebApplication.p...	2 KB	Program Debug D...	23-03-2019 02:09:58
MyWebApplication.r...	1 KB	JSON File	23-03-2019 02:10:02
MyWebApplication.Vi...	59 KB	Application extens...	23-03-2019 02:10:01
MyWebApplication.Vi...	4 KB	Program Debug D...	23-03-2019 02:10:01
web.config	1 KB	XML Configuratio...	23-03-2019 02:10:02

302 KB of 302 KB in 10 of 10

/site/wwwroot/

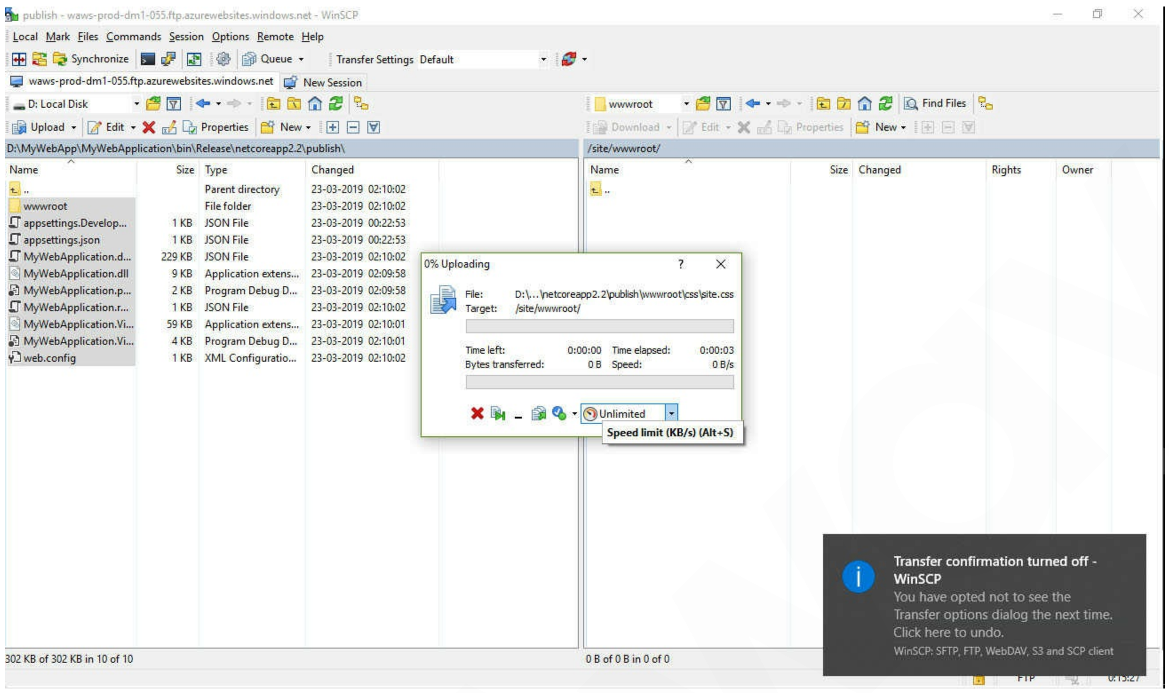
Name	Size	Changed	Rights	Owner
..				

0 B of 0 B in 0 of 0

0:14:49

Figure 2.45

14. Select all files and copy them to the FTP location.



publish - waws-prod-dm1-055.ftp.azurewebsites.windows.net - WinSCP

D:\MyWebApp\MyWebApplication\bin\Release\netcoreapp2.2\publish\

Name	Size	Type	Changed
..		Parent directory	23-03-2019 02:10:02
wwwroot		File folder	23-03-2019 02:10:02
appsettings.Develop...	1 KB	JSON File	23-03-2019 00:22:53
appsettings.json	1 KB	JSON File	23-03-2019 00:22:53
MyWebApplication.d...	229 KB	JSON File	23-03-2019 02:10:02
MyWebApplication.dll	9 KB	Application extens...	23-03-2019 02:09:58
MyWebApplication.p...	2 KB	Program Debug D...	23-03-2019 02:09:58
MyWebApplication.r...	1 KB	JSON File	23-03-2019 02:10:02
MyWebApplication.Vi...	59 KB	Application extens...	23-03-2019 02:10:01
MyWebApplication.Vi...	4 KB	Program Debug D...	23-03-2019 02:10:01
web.config	1 KB	XML Configuratio...	23-03-2019 02:10:02

302 KB of 302 KB in 10 of 10

/site/wwwroot/

0% Uploading

File: D:\...\netcoreapp2.2\publish\wwwroot\css\site.css
Target: /site/wwwroot/

Time left: 0:00:00 Time elapsed: 0:00:03
Bytes transferred: 0 B Speed: 0 B/s

Unlimited
Speed limit (KB/s) (Alt+S)

0 B of 0 B in 0 of 0

Transfer confirmation turned off - WinSCP
You have opted not to see the Transfer options dialog the next time. Click here to undo.
WinSCP: SFTP, FTP, WebDAV, S3 and SCP client

Figure 2.46

15. Go back to web application URL and check if updates are uploaded correctly to the web application:

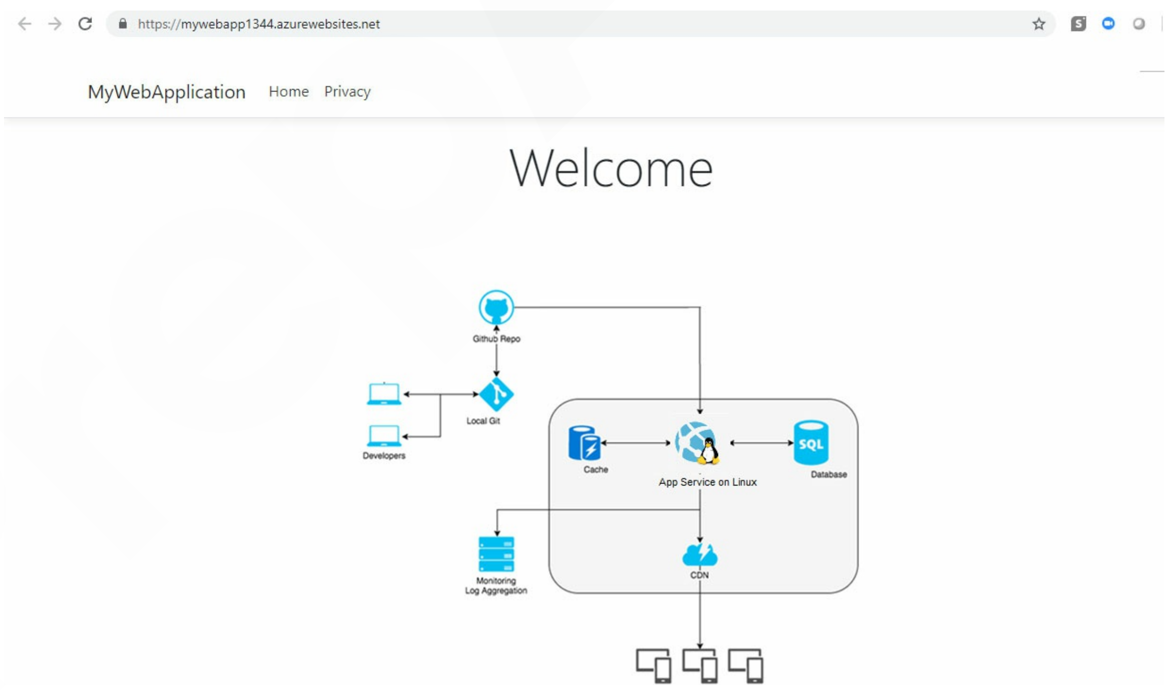


Figure 2.47

And you can see updates are made to the web application. The image is updated.

Why and what is CI CD?

Any software system consists of a number of applications. It’s easy to manage a small project or system. But when application complexity and team working on the system increases in size, it’s become challenging to manage the development and deployment process.

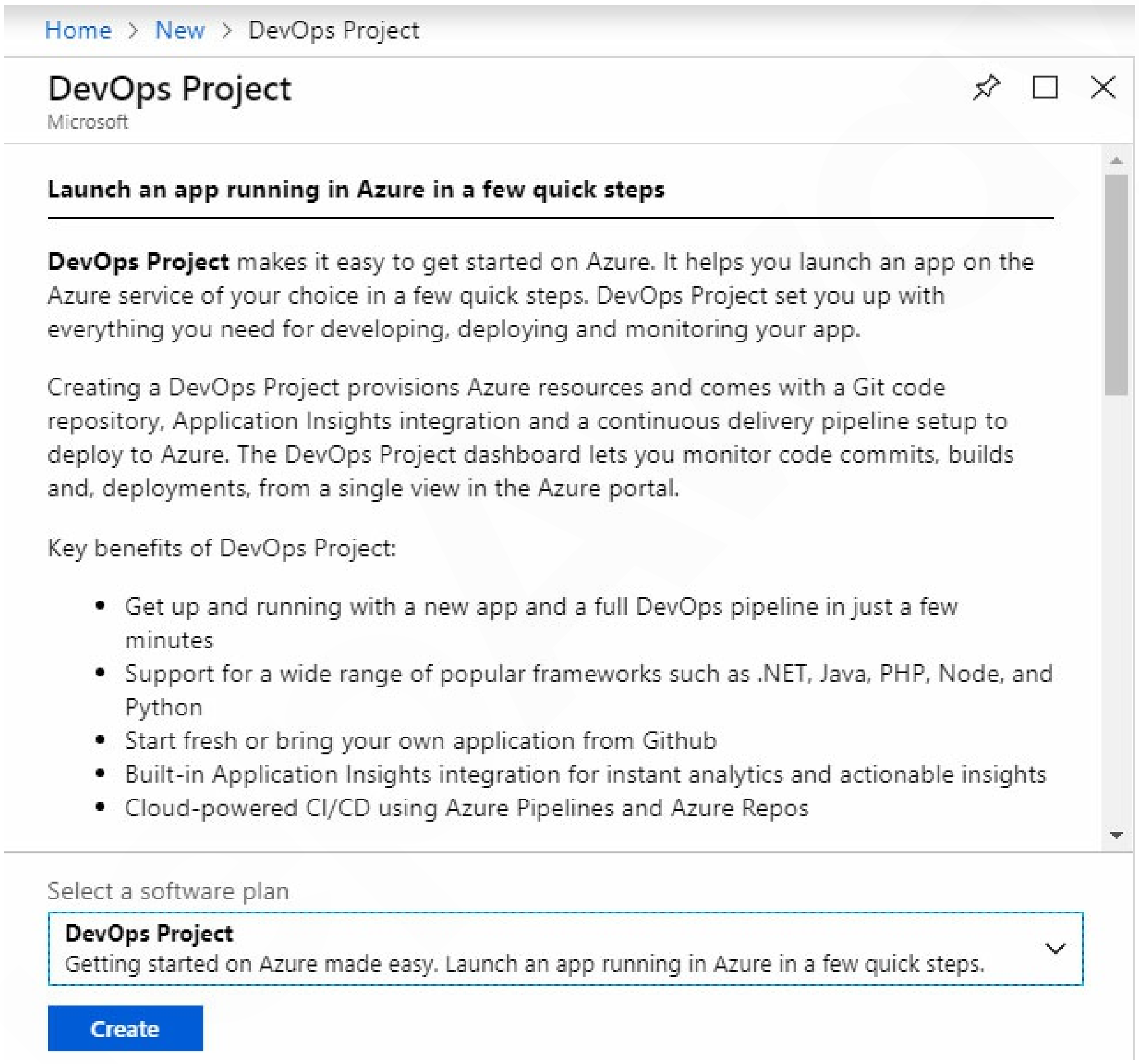
CI CD is the strategy to automate and streamline development, testing and release processes of the application. **Continuous integration (CI);** it focuses on integrating code or check-ins of the developer with the main repository. It will help to find integration issues in the early stages. It eases collaborative development from multiple teams. **Continuous deployment (CD);** it deploys the latest code from the main repository whenever updates are verified and merged with the respective branch.

We can create a CI CD pipeline for App Service in Azure, which will get updates from the GitHub repository. As a part of prerequisite we need to have:

- Azure Subscription
- GitHub repository with application code

Create a DevOps resource

1. Go to the Azure portal. And click on create a resource. Search for **`DevOps Project`**:



DevOps Project
Microsoft

Launch an app running in Azure in a few quick steps

DevOps Project makes it easy to get started on Azure. It helps you launch an app on the Azure service of your choice in a few quick steps. DevOps Project set you up with everything you need for developing, deploying and monitoring your app.

Creating a DevOps Project provisions Azure resources and comes with a Git code repository, Application Insights integration and a continuous delivery pipeline setup to deploy to Azure. The DevOps Project dashboard lets you monitor code commits, builds and, deployments, from a single view in the Azure portal.

Key benefits of DevOps Project:

- Get up and running with a new app and a full DevOps pipeline in just a few minutes
- Support for a wide range of popular frameworks such as .NET, Java, PHP, Node, and Python
- Start fresh or bring your own application from Github
- Built-in Application Insights integration for instant analytics and actionable insights
- Cloud-powered CI/CD using Azure Pipelines and Azure Repos

Select a software plan

DevOps Project
Getting started on Azure made easy. Launch an app running in Azure in a few quick steps.

Figure 2.48

2. Click on **`Create`**. It will redirect to **`DevOps Project`** creation wizard. Here select **`Bring Your Own Code`**. And click on **`Next`**.

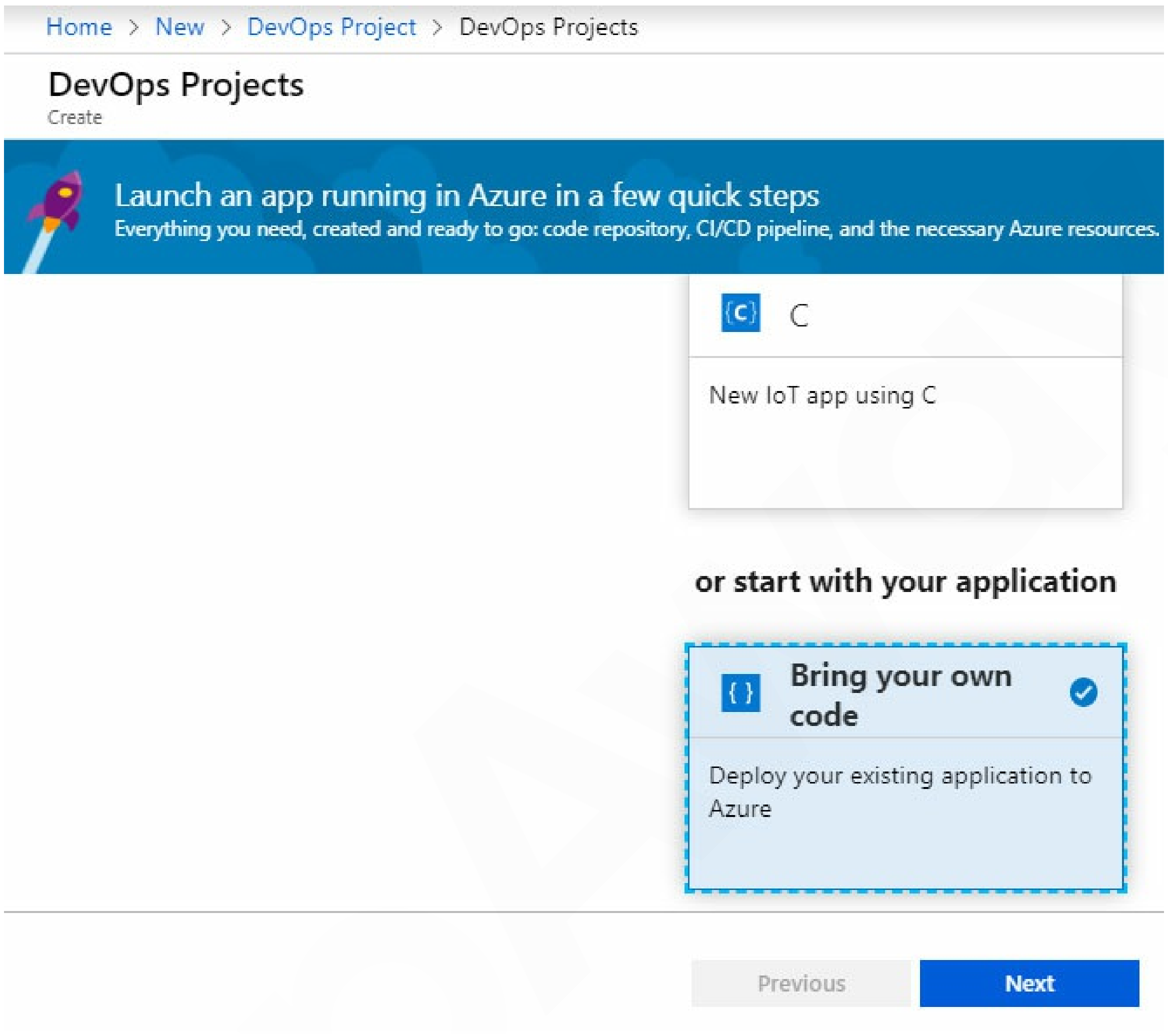
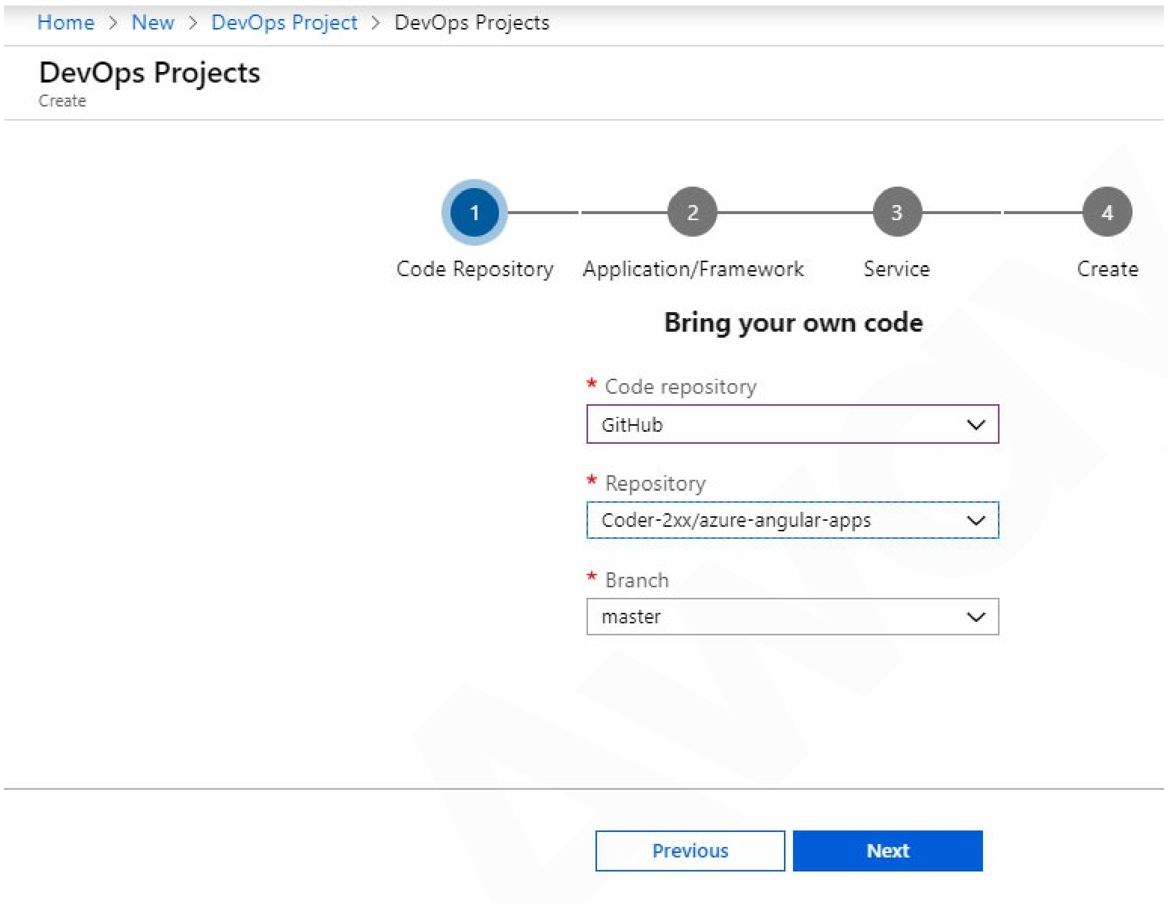

Home > New > DevOps Project > DevOps Projects
DevOps Projects
Create
Launch an app running in Azure in a few quick steps
Everything you need, created and ready to go: code repository, CI/CD pipeline, and the necessary Azure resources.
C
New IoT app using C
or start with your application
Bring your own code
Deploy your existing application to Azure
Previous
Next

Figure 2.49

3. Here select **`GitHub`** as code repository. If you are logged in to GitHub, it will populate all your repositories under the repository dropdown. Then select repository you want to configure CICD process. Select appropriate Brach. Click on **`Next`**.



Home > New > DevOps Project > DevOps Projects

DevOps Projects
Create

Bring your own code

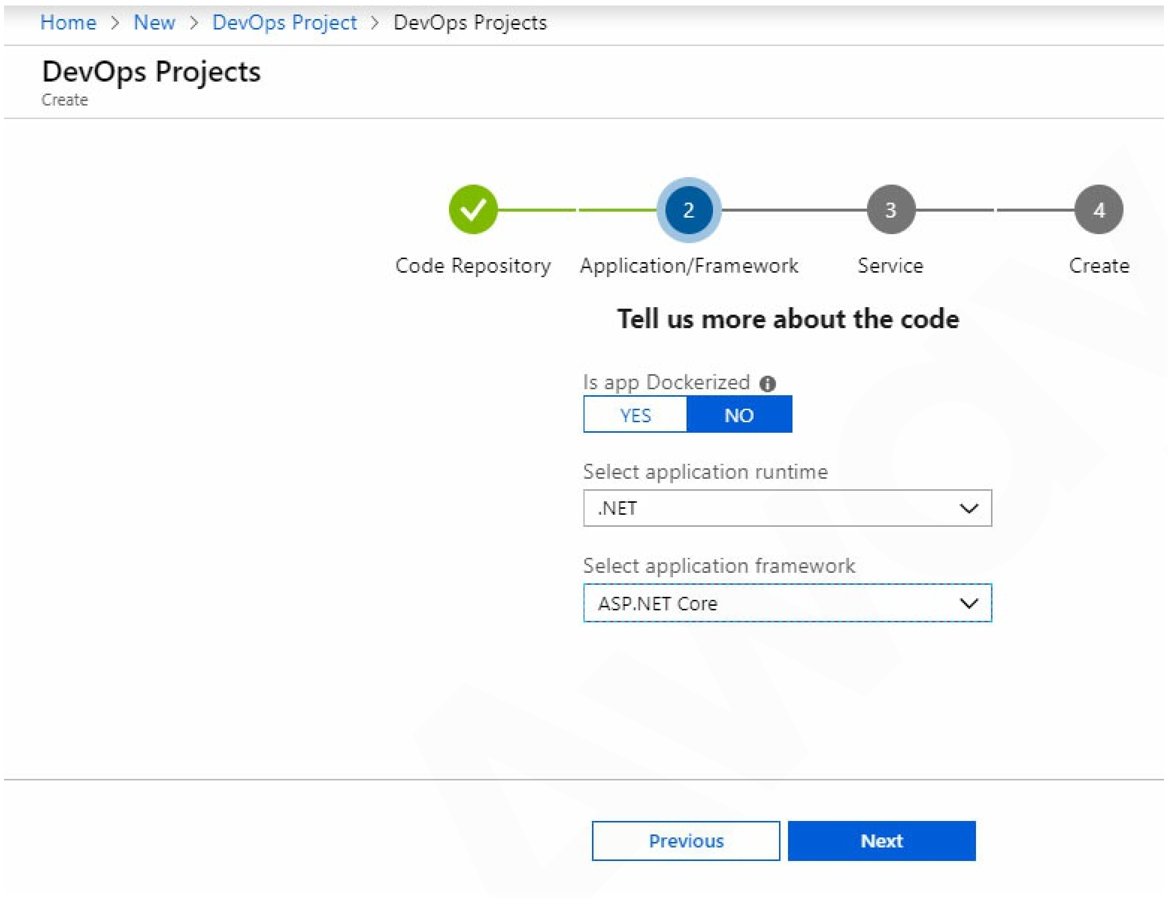
* Code repository: GitHub

* Repository: Coder-2xx/azure-angular-apps

* Branch: master

Figure 2.50

4. We are creating a new application, and our app is not deployed to docker yet. Hence select **`NO`**. Select application runtime as **`.NET`**. And application framework as **`ASP.NETCore`**.



Tell us more about the code

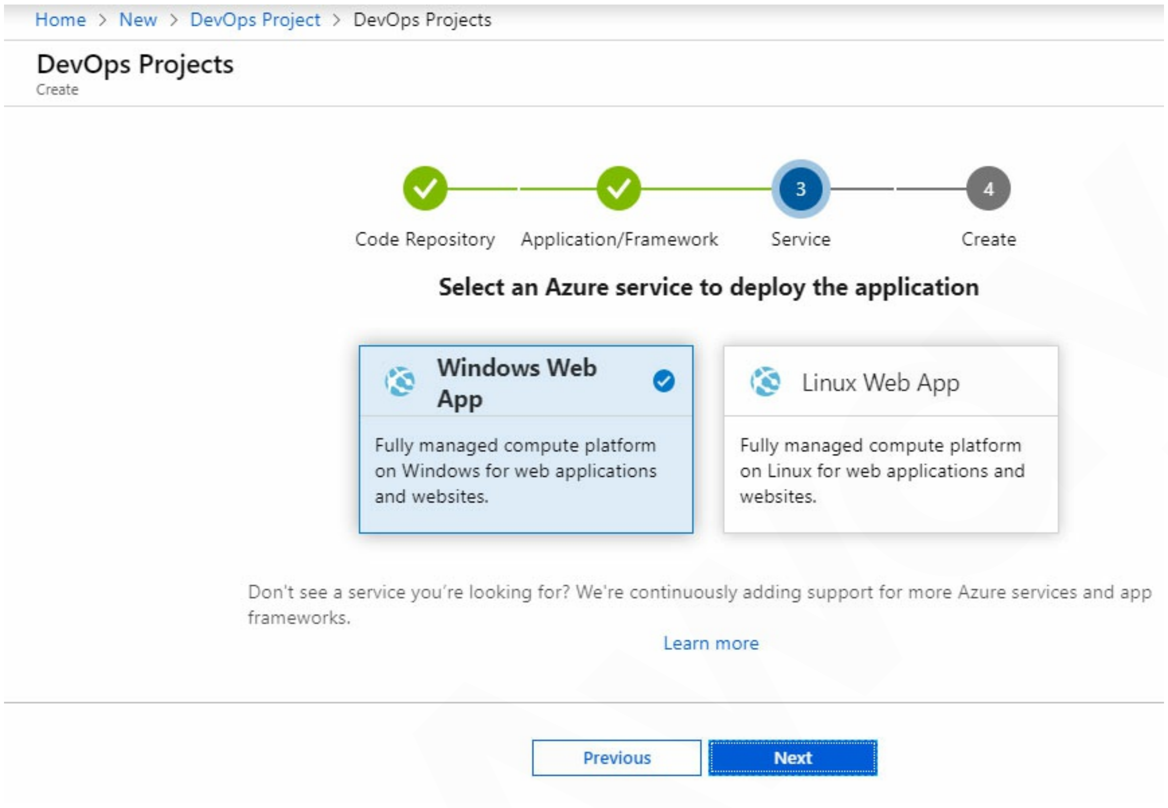
Is app Dockerized: YES [✓ NO]

Select application runtime: .NET

Select application framework: ASP.NET Core

Figure 2.51

5. We are going to deploy our app on windows compute service. Hence select **`Windows Web App`**:



Select an Azure service to deploy the application

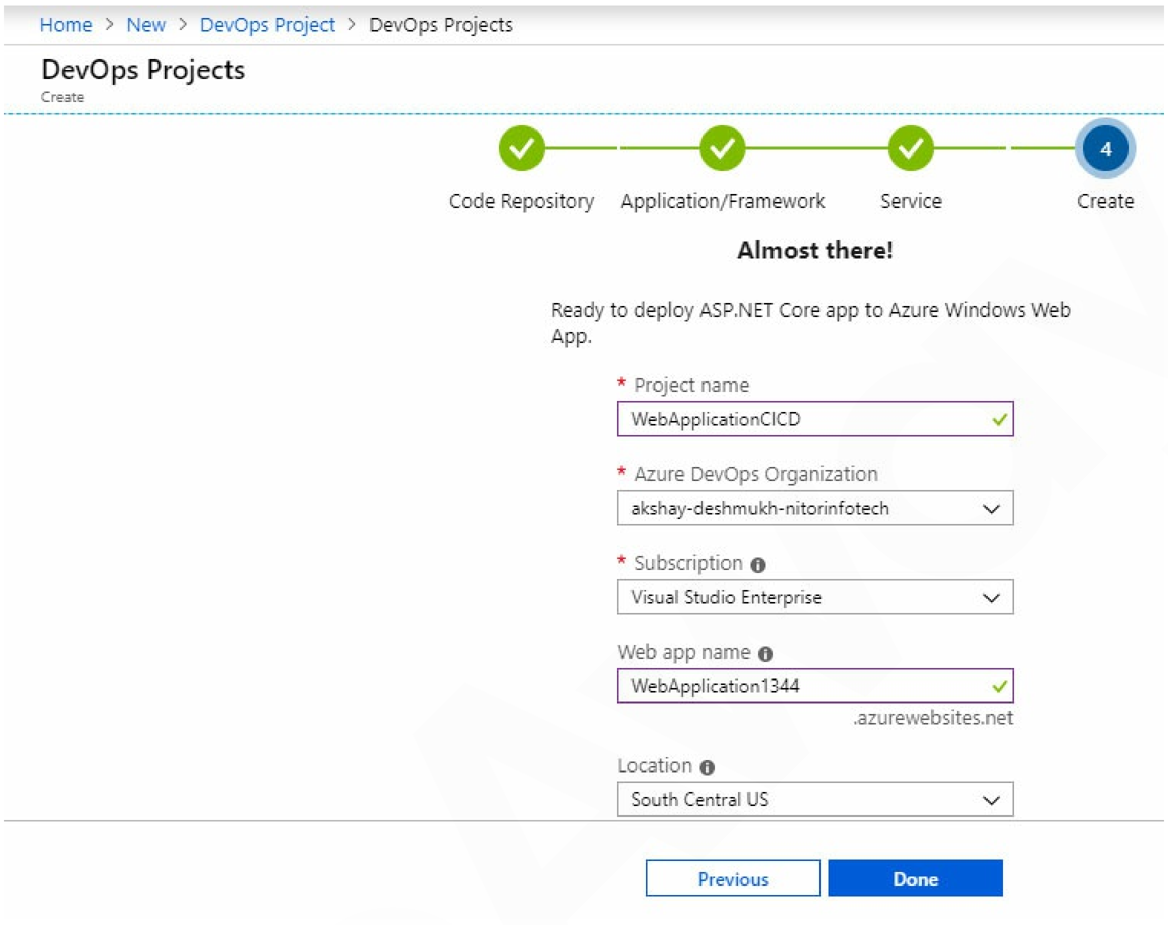
Windows Web App: Fully managed compute platform on Windows for web applications and websites.

Linux Web App: Fully managed compute platform on Linux for web applications and websites.

Don't see a service you're looking for? We're continuously adding support for more Azure services and app frameworks.

Figure 2.52

6. Here provide **`Project name`** for DevOps project. Select **`Azure DevOps Organization`** and **`Subscription`**. Provide a **`web app name`** where the application will be deployed and will be accessible by its URL. Provide **`Location`** for web application deployment. Click on **`Done`**:



DevOps Projects

Create

Almost there!

Ready to deploy ASP.NET Core app to Azure Windows Web App.

* Project name: WebApplicationCICD

* Azure DevOps Organization: akshay-deshmukh-nitorinfotech

* Subscription: Visual Studio Enterprise

Web app name: WebApplication1344 .azurewebsites.net

Location: South Central US

Figure 2.53

7. It will deploy the Azure DevOps project and build the CI CD pipeline.
8. Your application is enabled for CI CD. It will automatically deploy updates made in code and merged with the master branch. When developers push updates to the repository first, it will build the entire application to check build issues. If the build is successful, it will deploy the respective build to Azure App Service.
9. When build and deployment are done, you can refresh the application to see updates on the page.

Services comparison

To host web sites following services are available in Azure:

- **Azure App Service**
- **Virtual Machines**
- **Service Fabric**
- **Cloud Services**

Azure App Service

You can deploy or migrate an existing site to Azure App Service using an online migration tool or creating a new site using development tools. It provides you auto traffic management and scaling. You can schedule long-running jobs using Web jobs feature.

Service fabric

If you want to develop a web application from scratch and you are going to use microservices architecture for development, then Service fabric is the best choice. Service Fabric automatically manages service partitioning, scaling, and availability. You can hot services with Dot net core as well as **OWIN**. It provides access to infrastructure on which the application is hosted. Service Fabric is recommended for new development.

Virtual machine

If you have some legacy application or application which needs much effort to make compatible to run on App Service, then you can go for a virtual machine. You have to create VM and configure it precisely for security and high availability. You also need to manage VM for updates and patching. Azure Virtual Machines is **Infrastructure-as-a-Service (IaaS),** while App Service and Service Fabric are **Platform-as-a-Service (PaaS).**

Conclusion

Now, having your applications hosted in Azure is no more an alien topic for you. I would recommend you to perform all the steps as I mentioned earlier. Again, all preceding steps stand very much valid at the time of writing this chapter. Along with preceding listed points, for mastering the subject, I would request you to have watched on Azure service updates as well. In the coming chapter, I will work on another important aspect of cloud development, an interesting database offering from Microsoft Azure, and Azure CosmosDB.

Happy Azure Learning!

Questions

1. What are the different deployment options offered by Azure Web App?
2. What are Web App Deployment Slots?
3. What is the ‘AlwaysOn’ feature in the Web App?
4. Can be a web application be deployed via OneDrive or DropBox?
5. Explain the App Service Plan?
6. List the basic difference between Scale-Up and Scale-Out?

CHAPTER 3

Application Backend With Azure Cosmos DB

In the previous chapters, you studied the Azure ecosystem and have also completed hands-on with .NET core web application and deploying it to Azure App Service. In this chapter, we will be introducing you to an interesting database offering from Microsoft Azure, Azure CosmosDB. We will dive you more with an introduction to the service, creating the resource along with points to remember during the course. Later part, we will make you through exercise on using Azure Cosmos DB resource into your application.

Structure

- Introduction to Azure Cosmos DB
- Working with Cosmos DB
- Blogs development
- Basic CRUD Operation

Objective

After reading this chapter, you will learn:

- Introduction to Azure CosmosDB – Types and Feature
- Designing Azure Table or SQL as Database for Application.
- Managing and Implementing in application
- CRUD operation

Introduction to Azure Cosmos DB

Cosmos DB is a globally distributed database available on Azure. It is the most important introduction line of Cosmos DB. By using novel multi-master replication protocol, data in Cosmos DB can be replicated in several geographical areas. It supports unlimited elastic read and write scalability.

Cosmos DB is a non-relational database. We can call it as No SQL as well. Data in Cosmos DB is termed as a document. Each document is represented as a JSON object. There is no predefined format for the document. Each document may contain different attributes.

These documents can be logically grouped into containers. The group of containers is nothing but a **database**. All attributes are indexed by default.

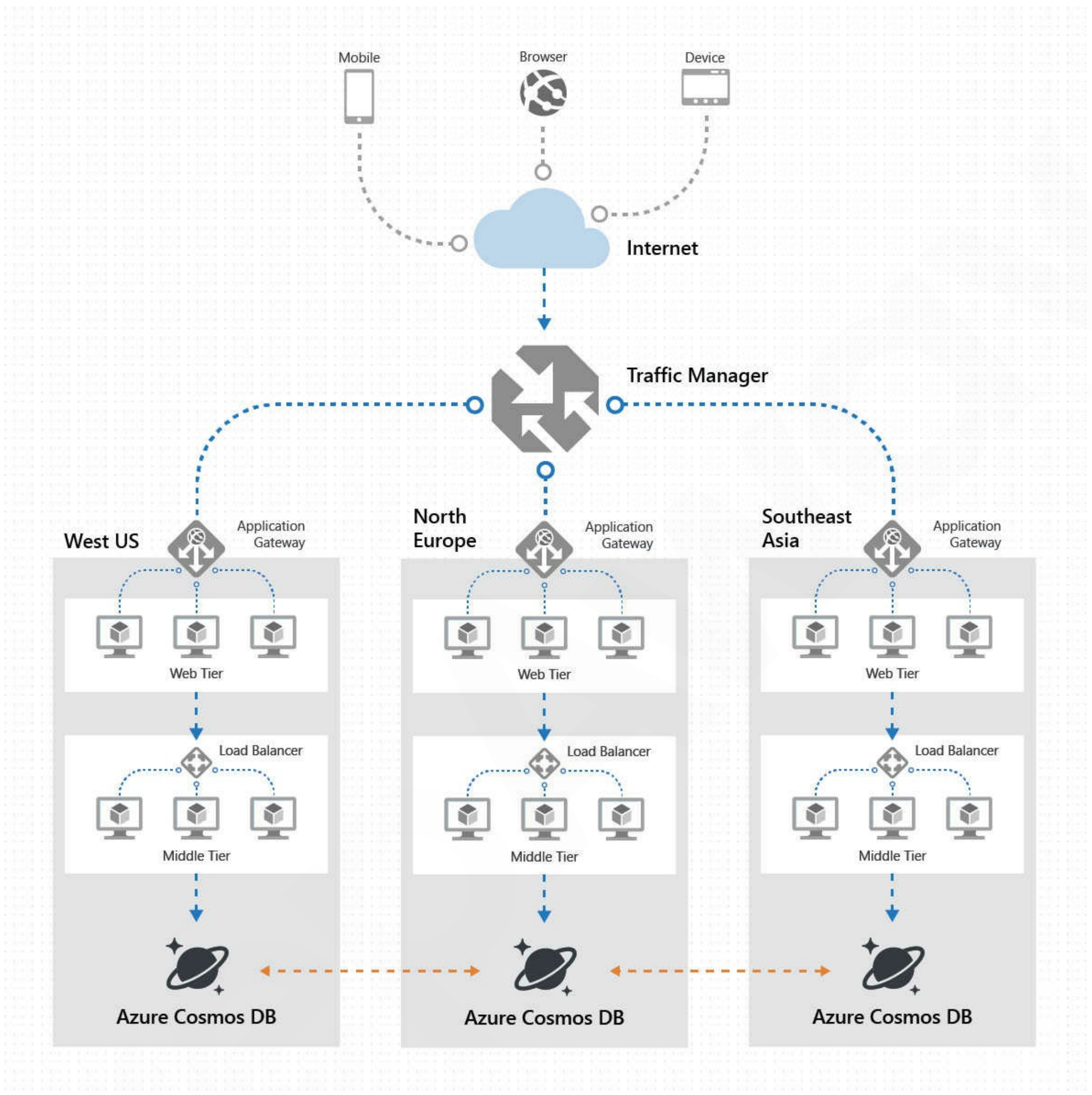


Figure 3.1: *Microsoft Docs – Azure CosmosDB – Broader look*

Cosmos DB is a multitenant **platform as a service (PaaS)** offering from Azure. Hence it comes with scalability and redundancy features of cloud service. DB provides comprehensive **service level agreements (SLAs)** for throughput, latency, availability, and consistency guarantees, something no other database service offers.

To access data from Cosmos DB, Azure provides SQL API, MongoDB API, Cassandra API, Tables API, and Gremlin API. They function as follows:

- SQL API can be used if we want to query a non-relational database using familiar SQL syntax.
- MongoDB API can be used if we have migrated existing mongo db data to managed Azure Cosmos DB.
- Similarly, Table API can be used if we have migrated existing Azure Table data to managed Azure Cosmos DB.
- Cassandra API can be used when we are migrating the existing Cassandra database to Azure Cosmos DB data.
- Azure Cosmos DB Gremlin API is used to store and operate on graph data. Gremlin API supports modeling graph data and provides APIs to traverse through the graph data.

Working with Cosmos DB

Now I will create Azure Cosmos DB account. In this account, I am going to create collections for models and add items to collections.

In a browser window, sign in to the Azure portal. Select **`Create a resource | Databases | Azure Cosmos DB`**.

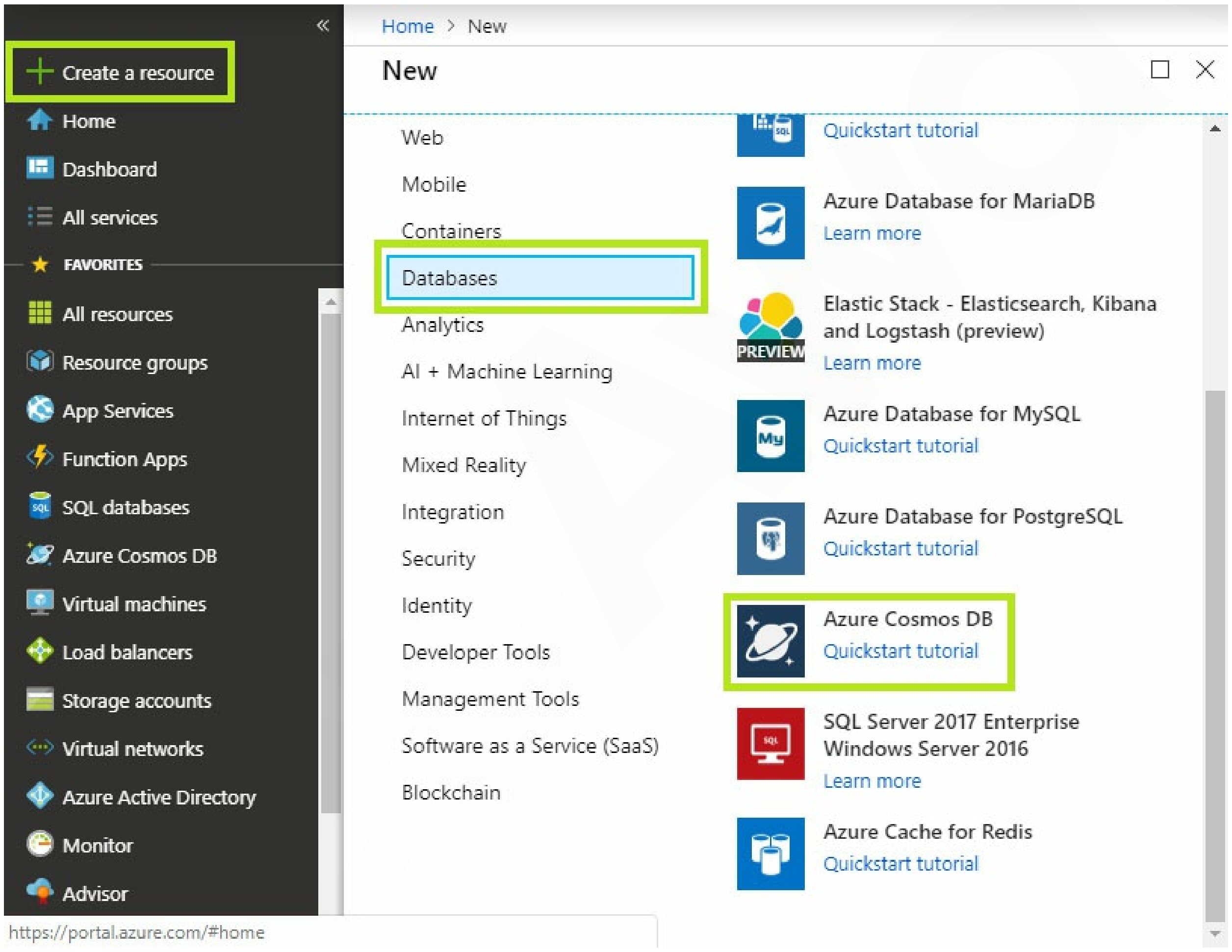


Figure 3.2

On the Create Azure Cosmos DB Account page, I will enter settings in the basics tab.

In Subscription dropdown, it will show all subscriptions assigned to the account, which I have used to login to the Azure portal. I have a visual studio enterprise subscription assigned to our account. If you use a trial subscription, it will show a limited trial option under dropdown. Here I will select Visual

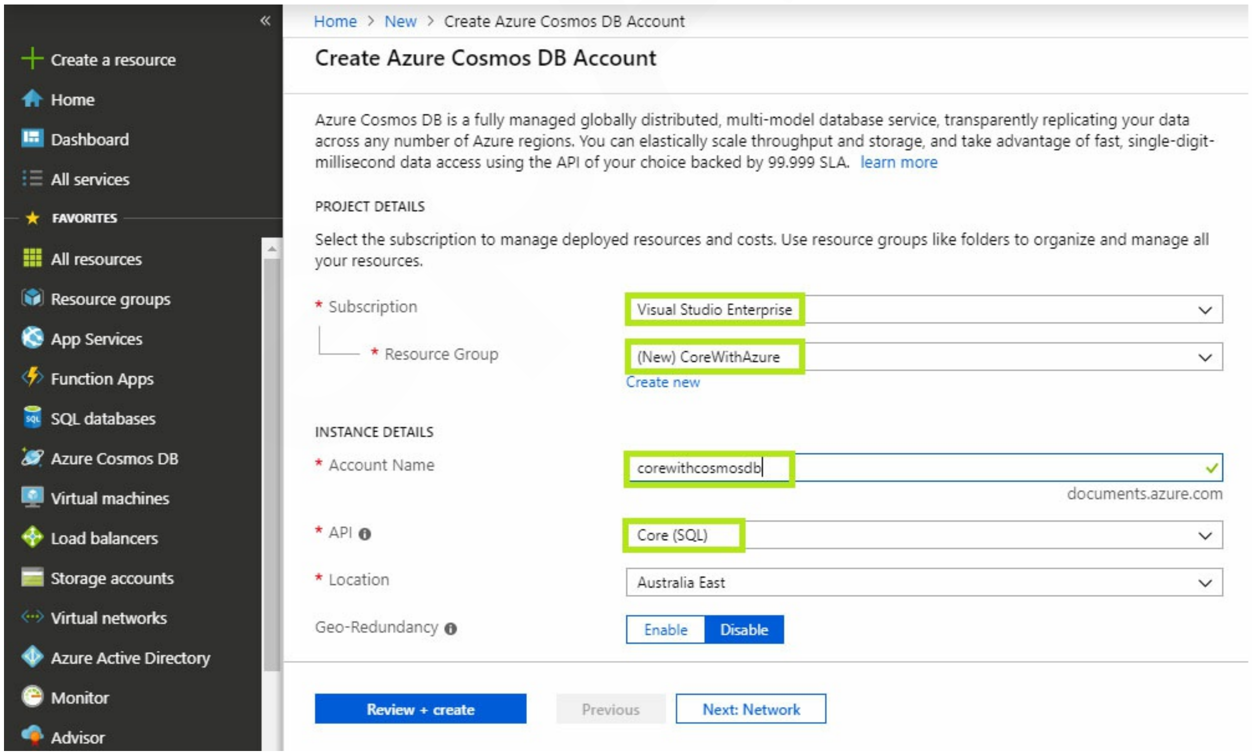
Studio Subscription.

We studied in the previous chapter that the resource group is used to group resources that are connected or from similar modules/applications. It eases the management of resources under the same application. Here I will select the same resource group which we have created in the previous section – *CoreWithAzure*.

The account name is a unique name for Cosmos DB account resource. The ID can only contain lowercase letters, numbers, and the hyphen (-) character. It must be between 3 and 31 characters in length.

Azure Cosmos DB provides five APIs to interact with data, namely Core(SQL) for document databases, Gremlin for graph databases, MongoDB for document databases, Azure Table, and Cassandra. I am going to create documents database and query data using SQL like syntax, and we will select **`Core SQL`**.

We can host resources in Azure datacenter at any location. To ensure faster retrieval of data, it is always recommended to select the location which is nearer to the users.



Create Azure Cosmos DB Account

Azure Cosmos DB is a fully managed globally distributed, multi-model database service, transparently replicating your data across any number of Azure regions. You can elastically scale throughput and storage, and take advantage of fast, single-digit-millisecond data access using the API of your choice backed by 99.999 SLA. learn more

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription: Visual Studio Enterprise

* Resource Group: (New) CoreWithAzure
Create new

INSTANCE DETAILS

* Account Name: corewithcosmosdb
documents.azure.com

* API: Core (SQL)

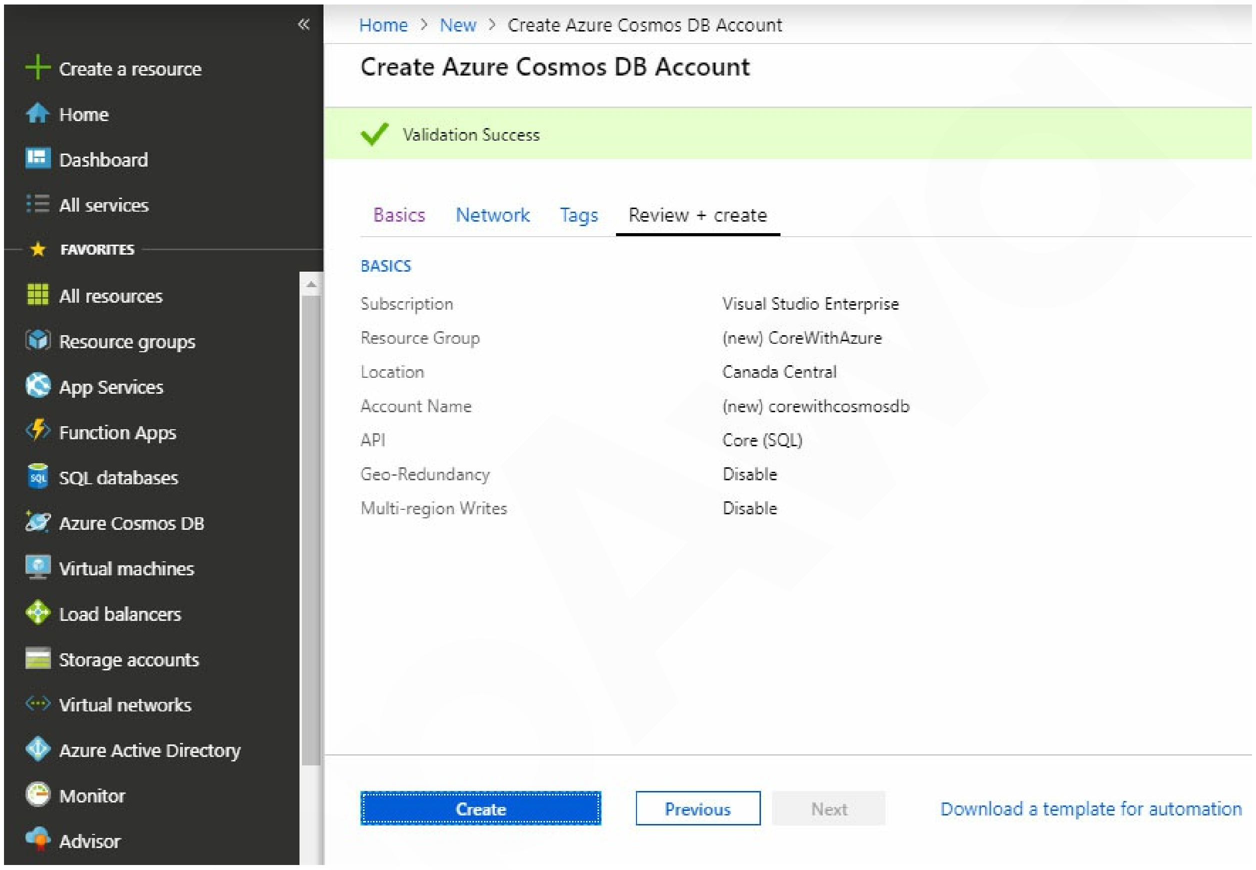
* Location: Australia East

Geo-Redundancy: Enable [✓ Disable]

Figure 3.3

Select **`Review + Create`** to create the Cosmos DB account. For now, skip **`Network`** and **`Tags`** settings.

Azure will validate all information entered and show validation evaluation messages. I have provided all the valid information. Hence, it will show the **`Validation Success`** message. Select **`Create`**.



Create Azure Cosmos DB Account

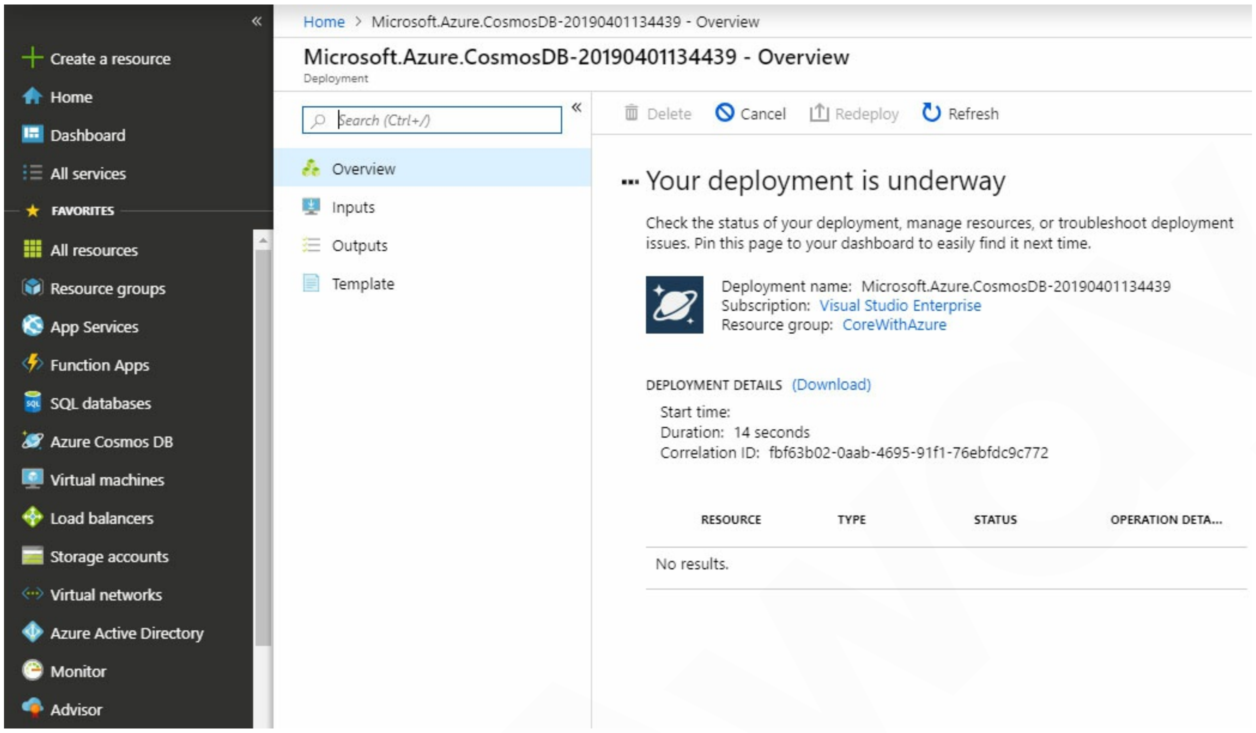
Validation Success

BASICS

Subscription	Visual Studio Enterprise
Resource Group	(new) CoreWithAzure
Location	Canada Central
Account Name	(new) corewithcosmosdb
API	Core (SQL)
Geo-Redundancy	Disable
Multi-region Writes	Disable

Figure 3.4

When I select **`Create`**, deployment requests will be submitted, and Azure started working on it.



Microsoft.Azure.CosmosDB-20190401134439 - Overview
Deployment

... Your deployment is underway

Check the status of your deployment, manage resources, or troubleshoot deployment issues. Pin this page to your dashboard to easily find it next time.

Deployment name: Microsoft.Azure.CosmosDB-20190401134439
Subscription: Visual Studio Enterprise
Resource group: CoreWithAzure

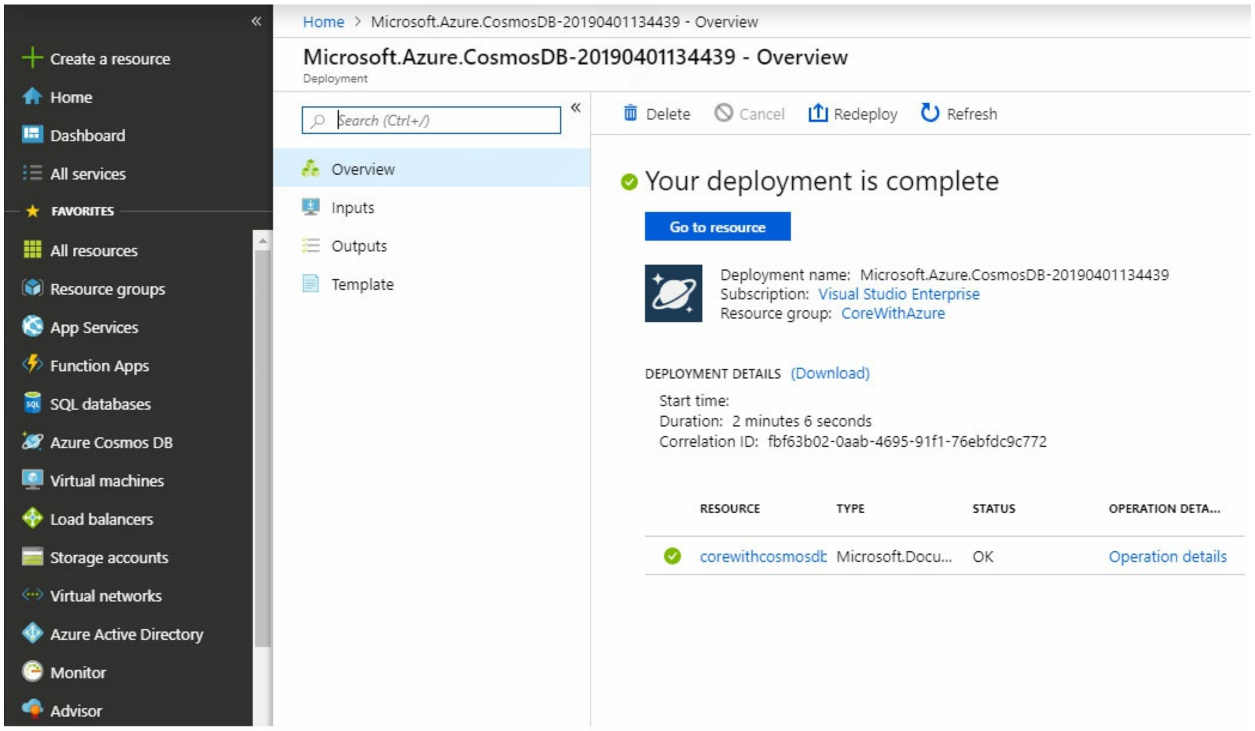
DEPLOYMENT DETAILS (Download)

Start time:
Duration: 14 seconds
Correlation ID: fbf63b02-0aab-4695-91f1-76ebfdc9c772

RESOURCE	TYPE	STATUS	OPERATION DETA...
No results.			

Figure 3.5

After deployment, the Azure portal will show **`Your deployment is Complete`** message along with deployment and resource details.



Microsoft.Azure.CosmosDB-20190401134439 - Overview
Deployment

Your deployment is complete

Deployment name: Microsoft.Azure.CosmosDB-20190401134439
Subscription: Visual Studio Enterprise
Resource group: CoreWithAzure

DEPLOYMENT DETAILS (Download)

Start time:
Duration: 2 minutes 6 seconds
Correlation ID: fbf63b02-0aab-4695-91f1-76ebfdc9c772

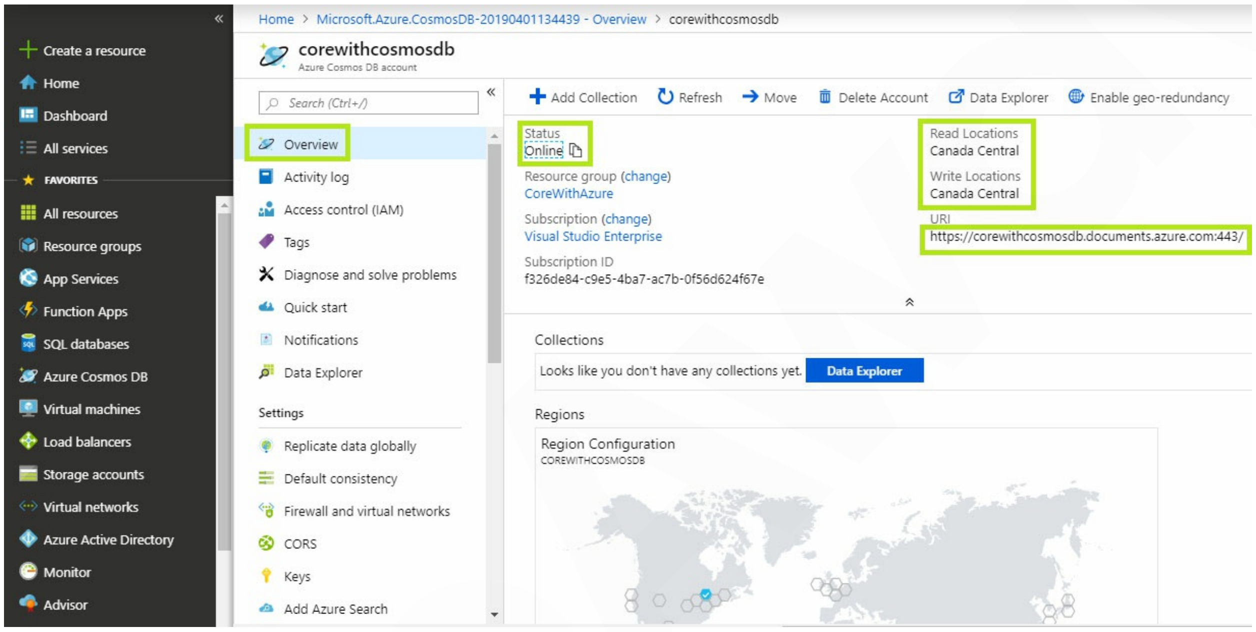
	RESOURCE	TYPE	STATUS	OPERATION DETA...
✓	corewithcosmosdt	Microsoft.Docu...	OK	Operation details

Figure 3.6

I will navigate to the Cosmos DB account we just created by hitting the go-to resource button, as can be seen in the preceding image.

Under the **`Overview`** page, you can see **`Status of Cosmos DB`** account, read and write locations, and URL of account.

Also, in the image *Figure 3.6*, below with **`Regions`** heading, locations where data will be replicated are shown highlighted.



Status
Online

Resource group (change)
CoreWithAzure

Subscription (change)
Visual Studio Enterprise

Subscription ID
f326de84-c9e5-4ba7-ac7b-0f56d624f67e

Read Locations
Canada Central

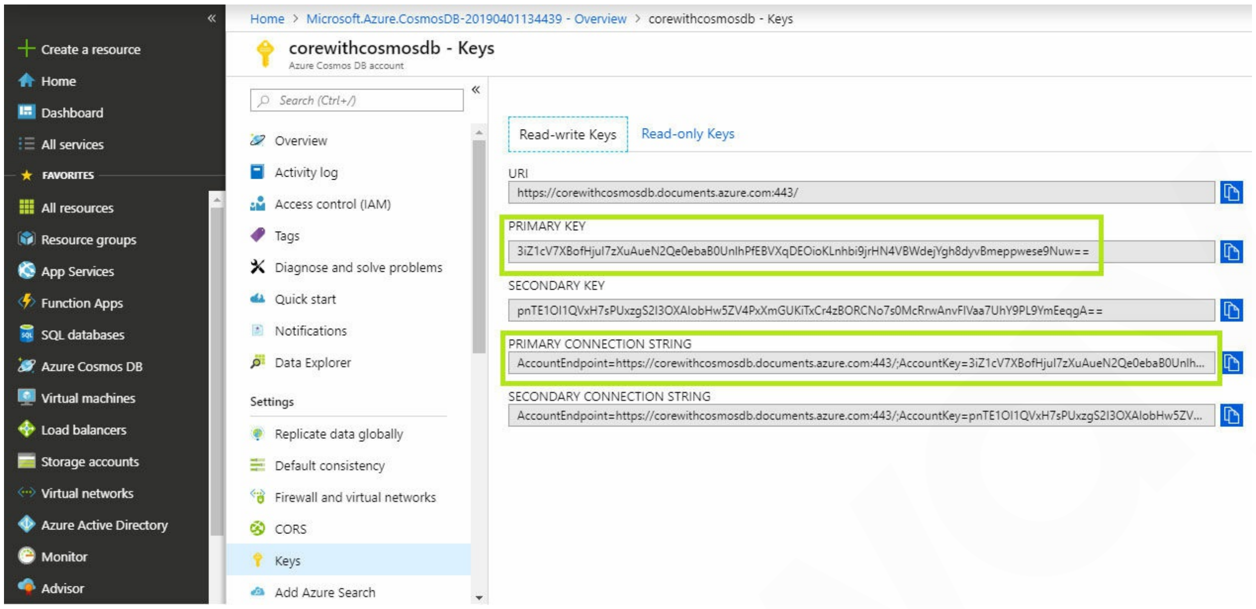
Write Locations
Canada Central

URI
https://corewithcosmosdb.documents.azure.com:443/

Looks like you don't have any collections yet.

Figure 3.7

Now I will click on keys from the left-hand side menu. Azure provides primary and secondary keys that are required to connect with the Azure Cosmos DB account. These keys will be used in our web application to perform CRUD operations on Cosmos DB collection.



URI
https://corewithcosmosdb.documents.azure.com:443/

PRIMARY KEY
3iZ1cV7XBofHjul7zXuAueN2Qe0ebaB0UnlhPfEBVXqDEOioKLnhbi9jrHN4VBWdejYgh8dyvBmeppwese9Nuw==

SECONDARY KEY
pnTE1Ol1QVxH7sPUxzgS2l3OXAlobHw5ZV4PxXmGUKiTxCr4zBORCNo7s0McRrwAnvFlVaa7UhY9PL9YmEeqgA==

PRIMARY CONNECTION STRING
AccountEndpoint=https://corewithcosmosdb.documents.azure.com:443/;AccountKey=3iZ1cV7XBofHjul7zXuAueN2Qe0ebaB0Unlh...

SECONDARY CONNECTION STRING
AccountEndpoint=https://corewithcosmosdb.documents.azure.com:443/;AccountKey=pnTE1Ol1QVxH7sPUxzgS2l3OXAlobHw5ZV...

Figure 3.8

Now we are ready with Cosmos DB account creation. Now I will show you how you can utilize Cosmos DB as **`Document`** storage.

I am going to use the same **Model View Controller (MVC)** web application we have created in the previous chapter. I am going to show you **Create, Read, Update,** and **Delete (CRUD)** operation on Blog data, which I am going to save in the Cosmos DB account, which we have created.

Let us create a model for Blog in our MVC application. Go to **`Project`**, right-click on **`Models`** folder. Select **`Add | New Item.`**

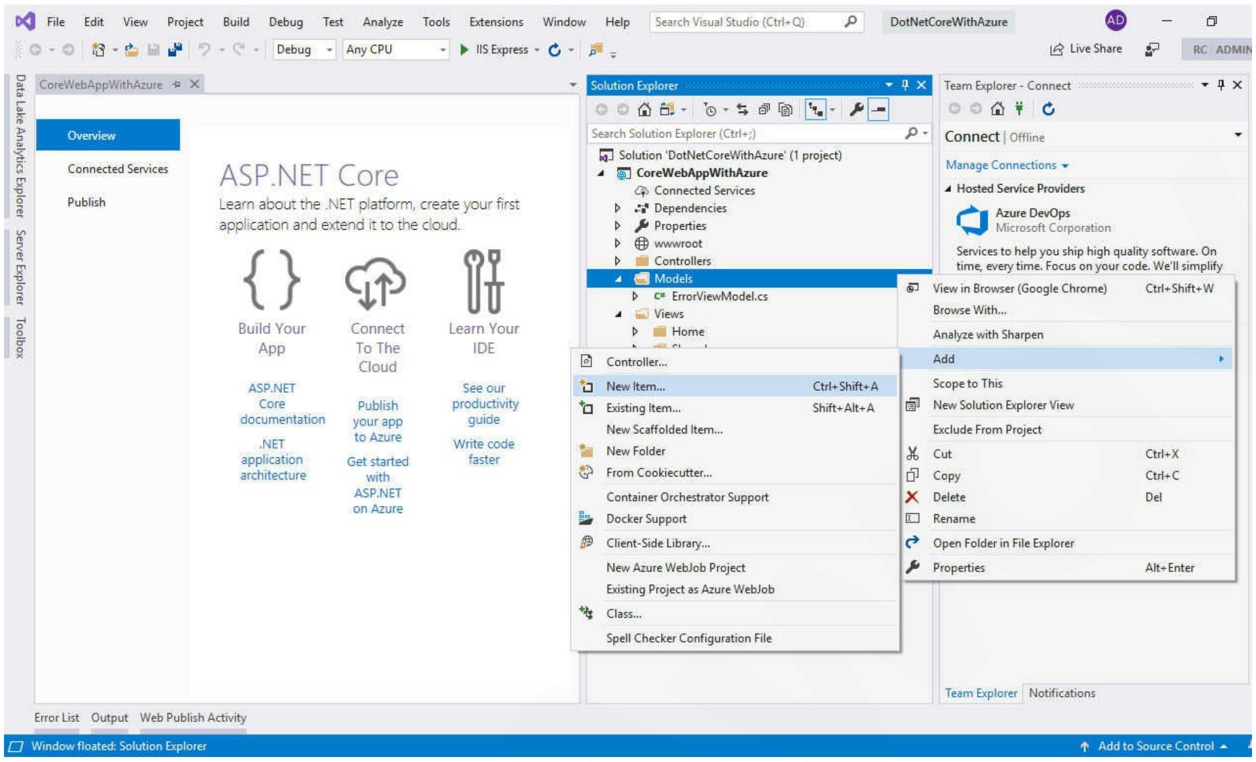
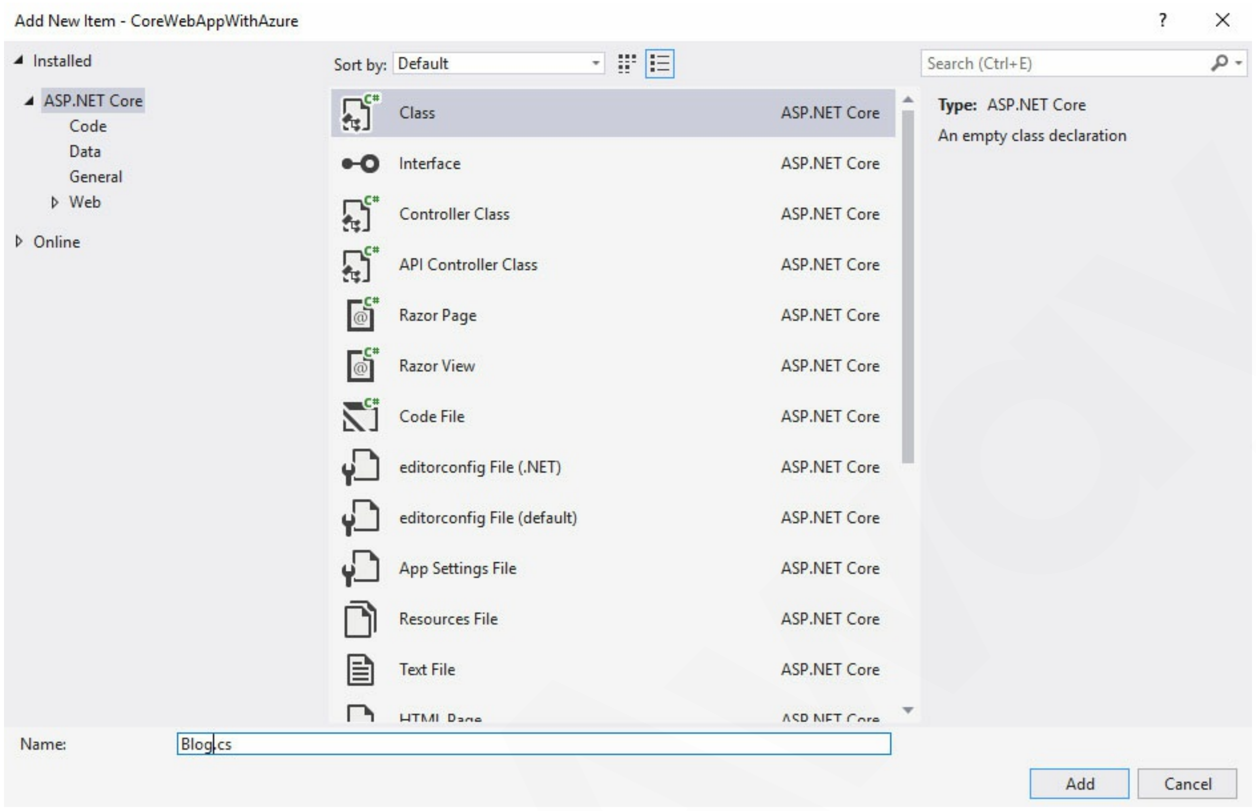


Figure 3.9

In **`Add New Item`** pane select **`Class`**. I will provide the **`Model`** name as **`Blog`** and click on **`Add`**.



Add New Item - CoreWebAppWithAzure

Installed

ASP.NET Core
Code
Data
General
Web

Online

Sort by: Default

Search (Ctrl+E)

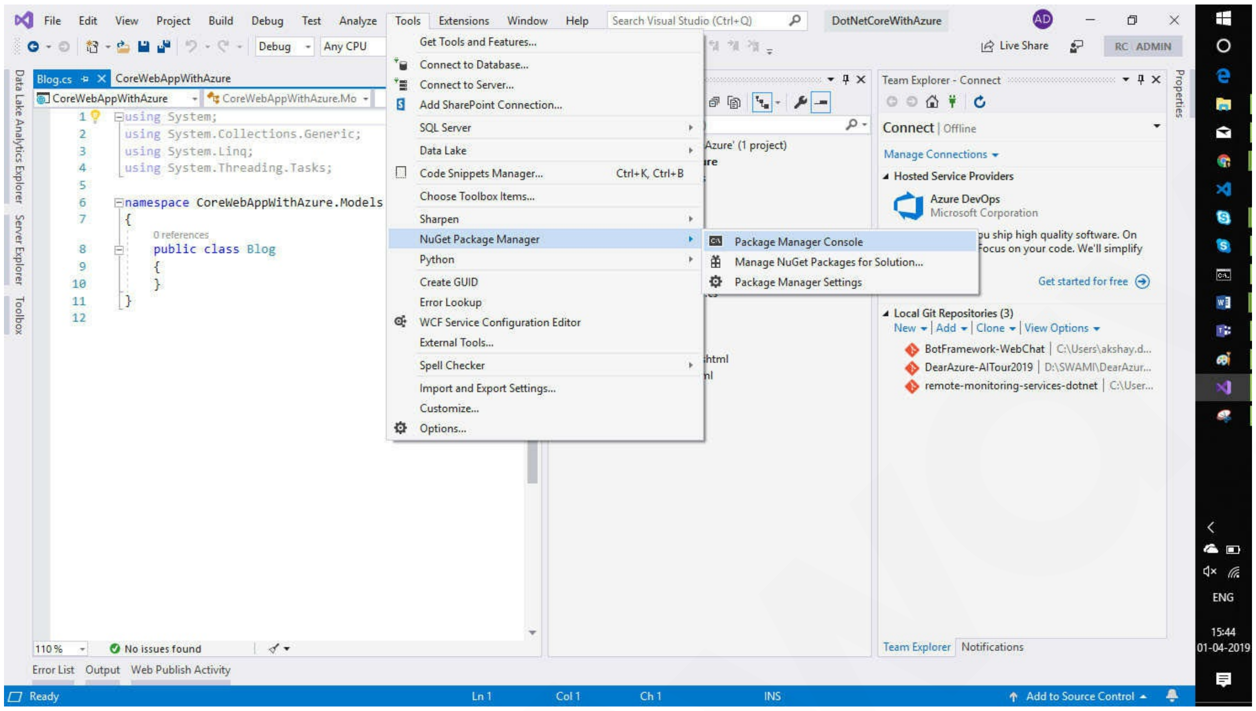
Class	ASP.NET Core
Interface	ASP.NET Core
Controller Class	ASP.NET Core
API Controller Class	ASP.NET Core
Razor Page	ASP.NET Core
Razor View	ASP.NET Core
Code File	ASP.NET Core
editorconfig File (.NET)	ASP.NET Core
editorconfig File (default)	ASP.NET Core
App Settings File	ASP.NET Core
Resources File	ASP.NET Core
Text File	ASP.NET Core
HTML Page	ASP.NET Core

Type: ASP.NET Core
An empty class declaration

Name: Blog.cs

Figure 3.10

I will go to **`Tools | NuGet Package Manager | Package Manager Console`**. It will open the command prompt. I am going to install required nugget packages using the console.

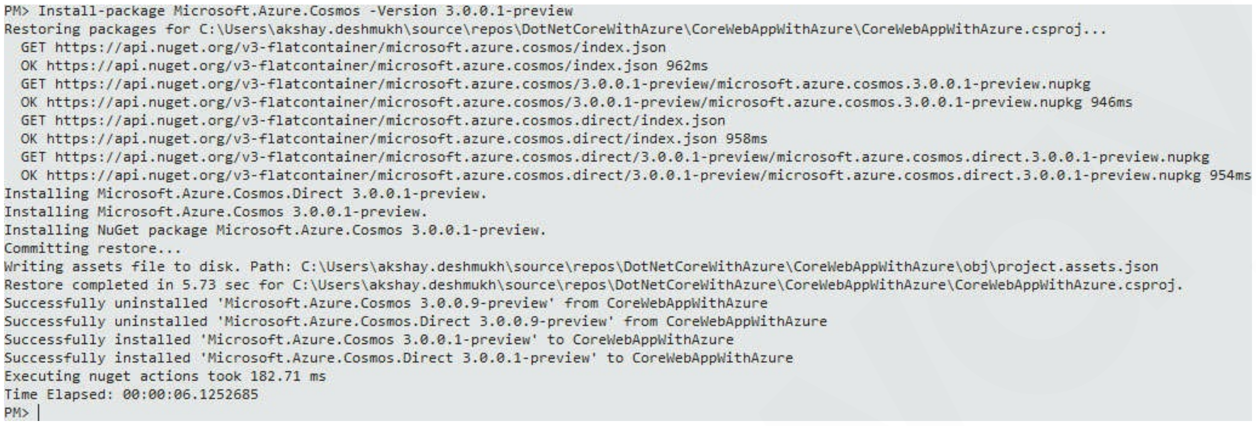


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace CoreWebAppWithAzure.Models
{
    public class Blog
    {
    }
}
```

Figure 3.11

I am going to install `Microsoft.Azure.Cosmos` and `Newtonsoft.Json` package to project. We will run the following commands, which will install packages and all its dependencies. It will also add references to `Microsoft.Azure.Cosmos.Client` and `Newtonsoft.Json` to our project.

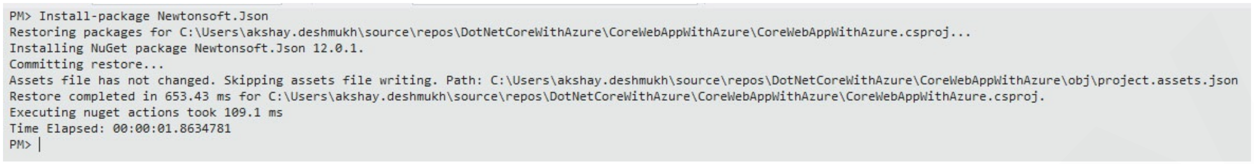
Install-package Microsoft.Azure.Cosmos -Version 3.0.0.1-preview



```
PM> Install-package Microsoft.Azure.Cosmos -Version 3.0.0.1-preview
Restoring packages for C:\Users\akshay.deshmukh\source\repos\DotNetCoreWithAzure\CoreWebAppWithAzure\CoreWebAppWithAzure.csproj...
  GET https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos/index.json
  OK https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos/index.json 962ms
  GET https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos/3.0.0.1-preview/microsoft.azure.cosmos.3.0.0.1-preview.nupkg
  OK https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos/3.0.0.1-preview/microsoft.azure.cosmos.3.0.0.1-preview.nupkg 946ms
  GET https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos.direct/index.json
  OK https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos.direct/index.json 958ms
  GET https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos.direct/3.0.0.1-preview/microsoft.azure.cosmos.direct.3.0.0.1-preview.nupkg
  OK https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos.direct/3.0.0.1-preview/microsoft.azure.cosmos.direct.3.0.0.1-preview.nupkg 954ms
Installing Microsoft.Azure.Cosmos.Direct 3.0.0.1-preview.
Installing Microsoft.Azure.Cosmos 3.0.0.1-preview.
Installing NuGet package Microsoft.Azure.Cosmos 3.0.0.1-preview.
Committing restore...
Writing assets file to disk. Path: C:\Users\akshay.deshmukh\source\repos\DotNetCoreWithAzure\CoreWebAppWithAzure\obj\project.assets.json
Restore completed in 5.73 sec for C:\Users\akshay.deshmukh\source\repos\DotNetCoreWithAzure\CoreWebAppWithAzure\CoreWebAppWithAzure.csproj.
Successfully uninstalled 'Microsoft.Azure.Cosmos 3.0.0.9-preview' from CoreWebAppWithAzure
Successfully uninstalled 'Microsoft.Azure.Cosmos.Direct 3.0.0.9-preview' from CoreWebAppWithAzure
Successfully installed 'Microsoft.Azure.Cosmos 3.0.0.1-preview' to CoreWebAppWithAzure
Successfully installed 'Microsoft.Azure.Cosmos.Direct 3.0.0.1-preview' to CoreWebAppWithAzure
Executing nuget actions took 182.71 ms
Time Elapsed: 00:00:06.1252685
PM> |
```

Figure 3.12

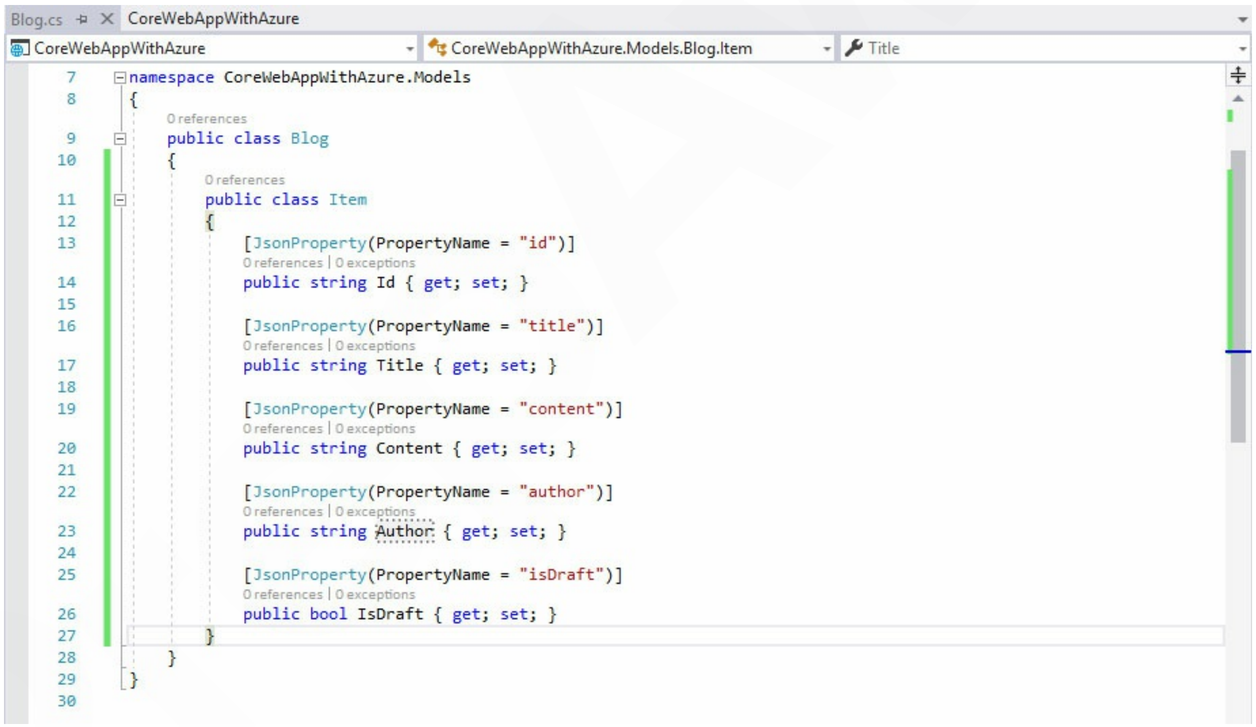
Install-package Newtonsoft.Json



```
PM> Install-package Newtonsoft.Json
Restoring packages for C:\Users\akshay.deshmukh\source\repos\DotNetCoreWithAzure\CoreWebAppWithAzure\CoreWebAppWithAzure.csproj...
Installing NuGet package Newtonsoft.Json 12.0.1.
Committing restore...
Assets file has not changed. Skipping assets file writing. Path: C:\Users\akshay.deshmukh\source\repos\DotNetCoreWithAzure\CoreWebAppWithAzure\obj\project.assets.json
Restore completed in 653.43 ms for C:\Users\akshay.deshmukh\source\repos\DotNetCoreWithAzure\CoreWebAppWithAzure\CoreWebAppWithAzure.csproj.
Executing nuget actions took 109.1 ms
Time Elapsed: 00:00:01.8634781
PM> |
```

Figure 3.13

Now I will open `Blog.cs` file and start creating properties in blog class. I will add **`Tile, Content, Author`**, and is the **`Draft`** property for blog class. I have added the `JsonProperty` attribute for each property of the class. JSON.NET controls serialization and deserialization of objects while sending or receiving it to/from Cosmos DB. To make property names `json` object compatible, I have used `JsonProperty`.



```
namespace CoreWebAppWithAzure.Models
{
    0 references
    public class Blog
    {
        0 references
        public class Item
        {
            [JsonProperty(PropertyName = "id")]
            0 references | 0 exceptions
            public string Id { get; set; }

            [JsonProperty(PropertyName = "title")]
            0 references | 0 exceptions
            public string Title { get; set; }

            [JsonProperty(PropertyName = "content")]
            0 references | 0 exceptions
            public string Content { get; set; }

            [JsonProperty(PropertyName = "author")]
            0 references | 0 exceptions
            public string Author { get; set; }

            [JsonProperty(PropertyName = "isDraft")]
            0 references | 0 exceptions
            public bool IsDraft { get; set; }
        }
    }
}
```

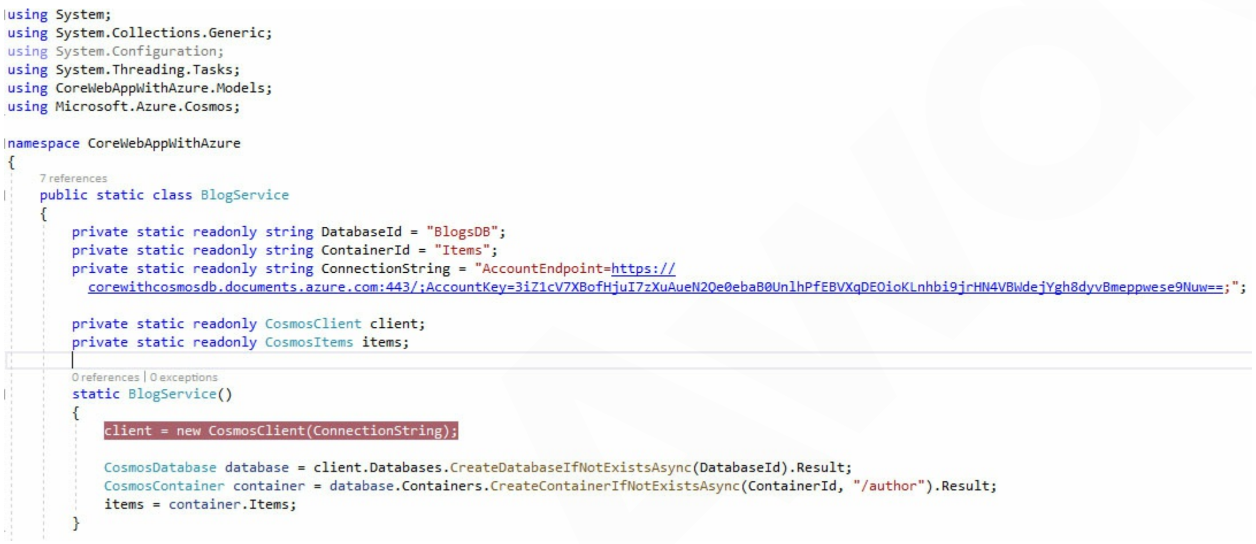
Figure 3.14

I will create a service for `Blog`, naming it as BlogService. Add a `static` class to the project and name it as `BlogService`. I will add static properties required to connect with `CosmosDB` and fill them from `Web.config` settings values.

I have provided a database as `BlogsDB`. The container in that database will be

`Items;` it is nothing but a collection in terms of No SQL terminology. The connection string is copied from the keys section of the `CosmosDB` account resource.

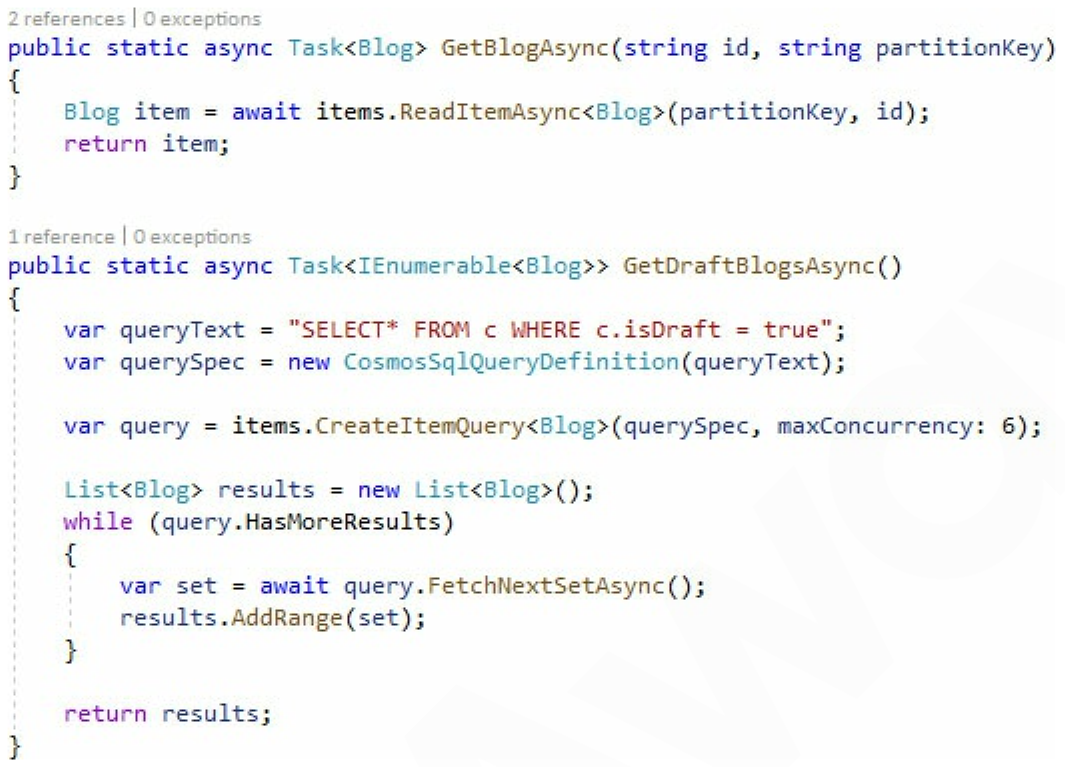
In the constructor of service, I have created the `CosmosClient` object using the connection string. I have created a database in the `CosmosDB` account if not exists. I have also created container/collection if it does not exist under `BlogsDb`.



```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Threading.Tasks;
using CoreWebAppWithAzure.Models;
using Microsoft.Azure.Cosmos;

namespace CoreWebAppWithAzure
{
    7 references
    public static class BlogService
    {
        private static readonly string DatabaseId = "BlogsDB";
        private static readonly string ContainerId = "Items";
        private static readonly string ConnectionString = "AccountEndpoint=https://
          corewithcosmosdb.documents.azure.com:443/;AccountKey=3iZ1cV7XBofHjuI7zXuAueN2Qe0ebaB0UnlhPfEBVXqDEOioKLnhbi9jrHN4VBWdejYgh8dyvBmeppwese9Nuw==;";

        private static readonly CosmosClient client;
        private static readonly CosmosItems items;

        0 references | 0 exceptions
        static BlogService()
        {
            client = new CosmosClient(ConnectionString);

            CosmosDatabase database = client.Databases.CreateDatabaseIfNotExistsAsync(DatabaseId).Result;
            CosmosContainer container = database.Containers.CreateContainerIfNotExistsAsync(ContainerId, "/author").Result;
            items = container.Items;
        }
```

Figure 3.15

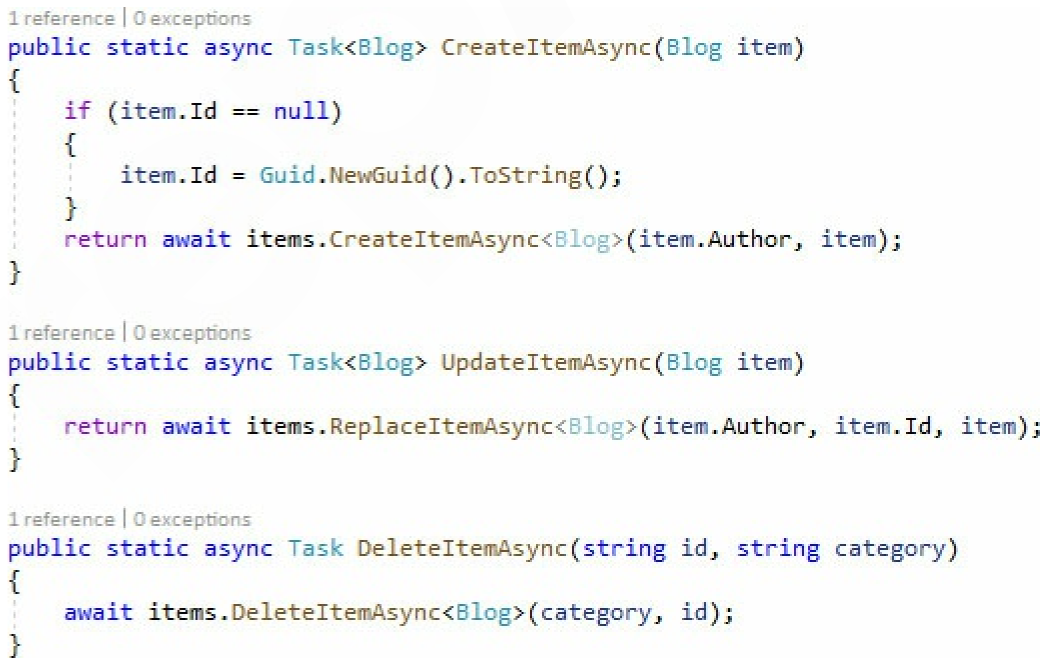
Now let's add method `GetBlogAsync` to get a single `Blog` item on the basis of Id and partition key. I have partitioned the data on the basis of the `Author` of the blog. Then add a method to get all blogs that are in draft mode. I am using SQL API to get data from `CosmosDB`. Max concurrency parameter controls the max number of partitions that our client will `query` in parallel:



```
2 references | 0 exceptions
public static async Task<Blog> GetBlogAsync(string id, string partitionKey)
{
    Blog item = await items.ReadItemAsync<Blog>(partitionKey, id);
    return item;
}

1 reference | 0 exceptions
public static async Task<IEnumerable<Blog>> GetDraftBlogsAsync()
{
    var queryText = "SELECT* FROM c WHERE c.isDraft = true";
    var querySpec = new CosmosSqlQueryDefinition(queryText);

    var query = items.CreateItemQuery<Blog>(querySpec, maxConcurrency: 6);

    List<Blog> results = new List<Blog>();
    while (query.HasMoreResults)
    {
        var set = await query.FetchNextSetAsync();
        results.AddRange(set);
    }

    return results;
}
```

Figure 3.16

Now let us add methods to create, update, and delete `Blog` item from `CosmosDB`. It calls methods against an `item` object of container/ collection.



```
1 reference | 0 exceptions
public static async Task<Blog> CreateItemAsync(Blog item)
{
    if (item.Id == null)
    {
        item.Id = Guid.NewGuid().ToString();
    }
    return await items.CreateItemAsync<Blog>(item.Author, item);
}

1 reference | 0 exceptions
public static async Task<Blog> UpdateItemAsync(Blog item)
{
    return await items.ReplaceItemAsync<Blog>(item.Author, item.Id, item);
}

1 reference | 0 exceptions
public static async Task DeleteItemAsync(string id, string category)
{
    await items.DeleteItemAsync<Blog>(category, id);
}
```

Figure 3.17

Now I will add the controller for our blog model. Go to **`Solution Explorer`**. Then right-click on the `Controller` folder. Select **`Add | Controller`**. I will name the controller as **`BlogController`** and select **`OK`**.

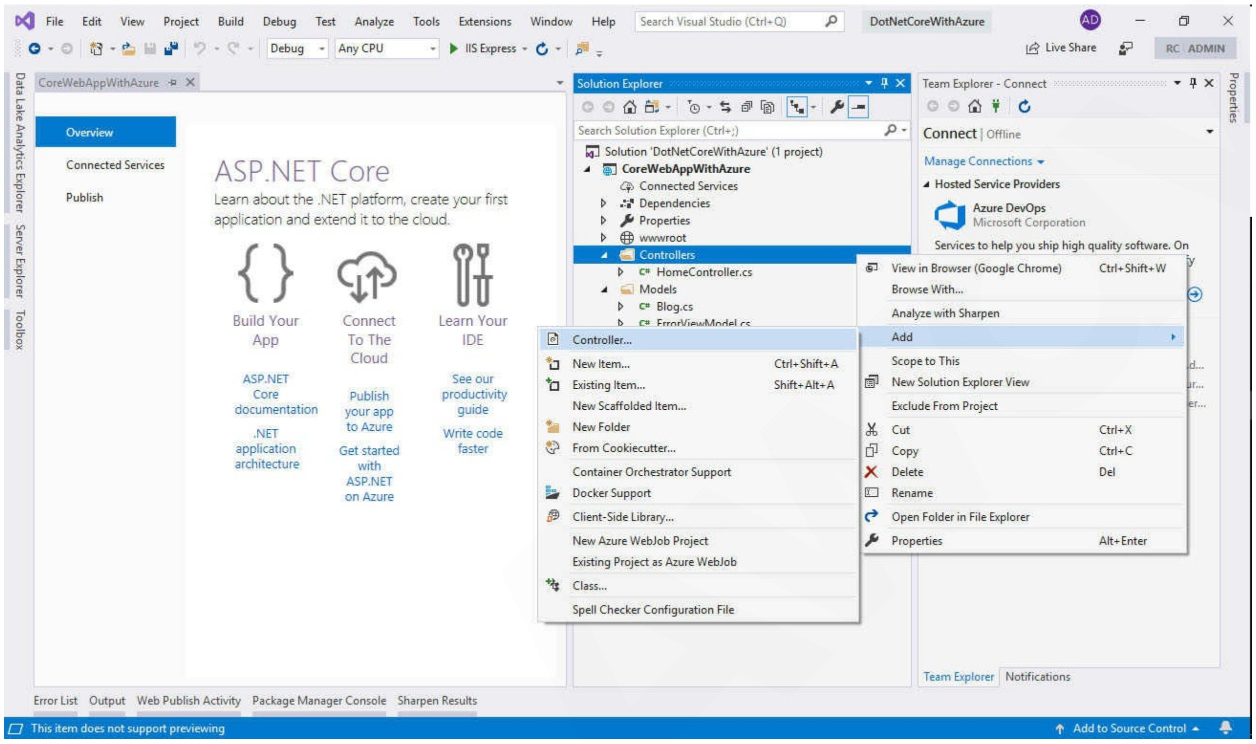
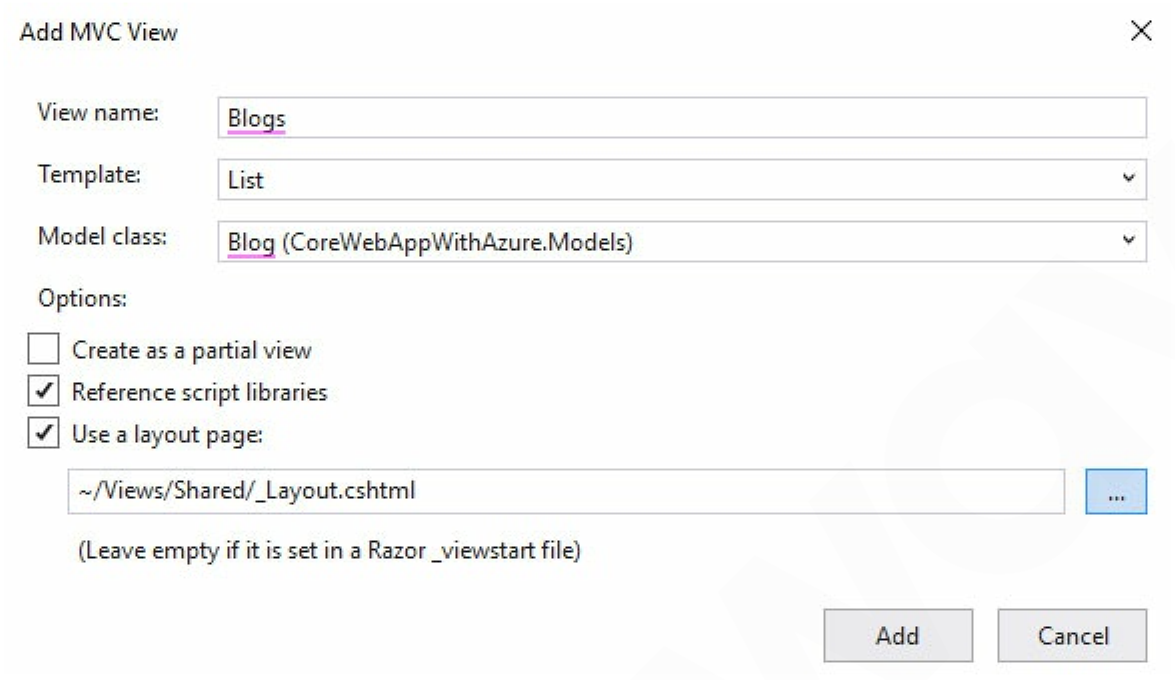

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search Visual Studio (Ctrl+Q)
DotNetCoreWithAzure
Debug
Any CPU
IIS Express
Live Share
CoreWebAppWithAzure
Overview
Connected Services
Publish
ASP.NET Core
Learn about the .NET platform, create your first application and extend it to the cloud.
Build Your App
Connect To The Cloud
Learn Your IDE
ASP.NET Core documentation
.NET application architecture
Publish your app to Azure
Get started with ASP.NET on Azure
See our productivity guide
Write code faster
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'DotNetCoreWithAzure' (1 project)
CoreWebAppWithAzure
Connected Services
Dependencies
Properties
wwwroot
Controllers
HomeController.cs
Models
Blog.cs
ErrorViewModel.cs
Controller...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
From Cookiecutter...
Container Orchestrator Support
Docker Support
Client-Side Library...
New Azure WebJob Project
Existing Project as Azure WebJob
Class...
Spell Checker Configuration File
View in Browser (Google Chrome) Ctrl+Shift+W
Browse With...
Analyze with Sharpen
Add
Scope to This
New Solution Explorer View
Exclude From Project
Cut Ctrl+X
Copy Ctrl+C
Delete Del
Rename
Open Folder in File Explorer
Properties Alt+Enter
Team Explorer - Connect
Connect | Offline
Manage Connections
Hosted Service Providers
Azure DevOps
Microsoft Corporation
Services to help you ship high quality software. On
Team Explorer Notifications
Error List Output Web Publish Activity Package Manager Console Sharpen Results
This item does not support previewing
Add to Source Control

Figure 3.18

Now I will create a view for `BlogController`. Add a folder named `Blog` to view folder. Right-click on the `Blog` folder. Select **`Add |View`**. In Add MVC View pane, I will provide **`View name`** as `Blogs`, **`Template`** for the view as List, **`Model class`** as Blog class, which I have created earlier. I will `select _layouts.cshtml` from `Shared` view as layout page of view:



View name: Blogs

Template: List

Model class: Blog (CoreWebAppWithAzure.Models)

Options:

- [ ] Create as a partial view
- [x] Reference script libraries
- [x] Use a layout page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

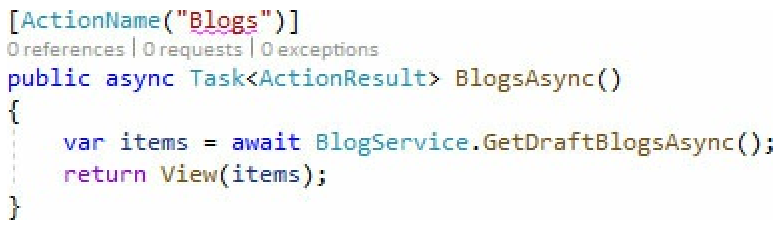
Figure 3.19

Now I will click on **`Add`**, and the visual studio starts creating a new view for the **`Blog`** model. Wizard will open the `cshtml` file of view. Update code in `cshtml` and I will close it:

```
@model IEnumerable<CoreWebAppWithAzure.Models.Blog>

@{
 ViewData["Title"] = "Blogs";
 Layout = "~/Views/Shared/_Layout.cshtml";
}

<h1>Blogs</h1>

<p>
<a asp-action="AddBlog">Create New</a>
</p>
<table class="table">
<thead>
<tr>
<th>
@Html.DisplayNameFor(model => model.Id)
</th>
<th>
@Html.DisplayNameFor(model => model.Title)
</th>
<th>
@Html.DisplayNameFor(model => model.Content)
```

```
</th>
<th>
@Html.DisplayNameFor(model => model.Author)
</th>
<th>
@Html.DisplayNameFor(model => model.IsDraft)
</th>
<th></th>
</tr>
</thead>
<tbody>
@foreach (var item in Model)
   {
<tr>
<td>
@Html.DisplayFor(modelItem => item.Id)
</td>
<td>
@Html.DisplayFor(modelItem => item.Title)
</td>
<td>
@Html.DisplayFor(modelItem => item.Content)
</td>
<td>
@Html.DisplayFor(modelItem => item.Author)
</td>
<td>
@Html.DisplayFor(modelItem => item.IsDraft)
</td>
<td>
@Html.ActionLink(“Edit”, “EditBlog”, new { id = item.Id, author =
item.Author }) |
@Html.ActionLink(“Delete”, “DeleteBlog”, new { id = item.Id,
author = item.Author })
</td>
</tr>
   }
</tbody>
</table>
```

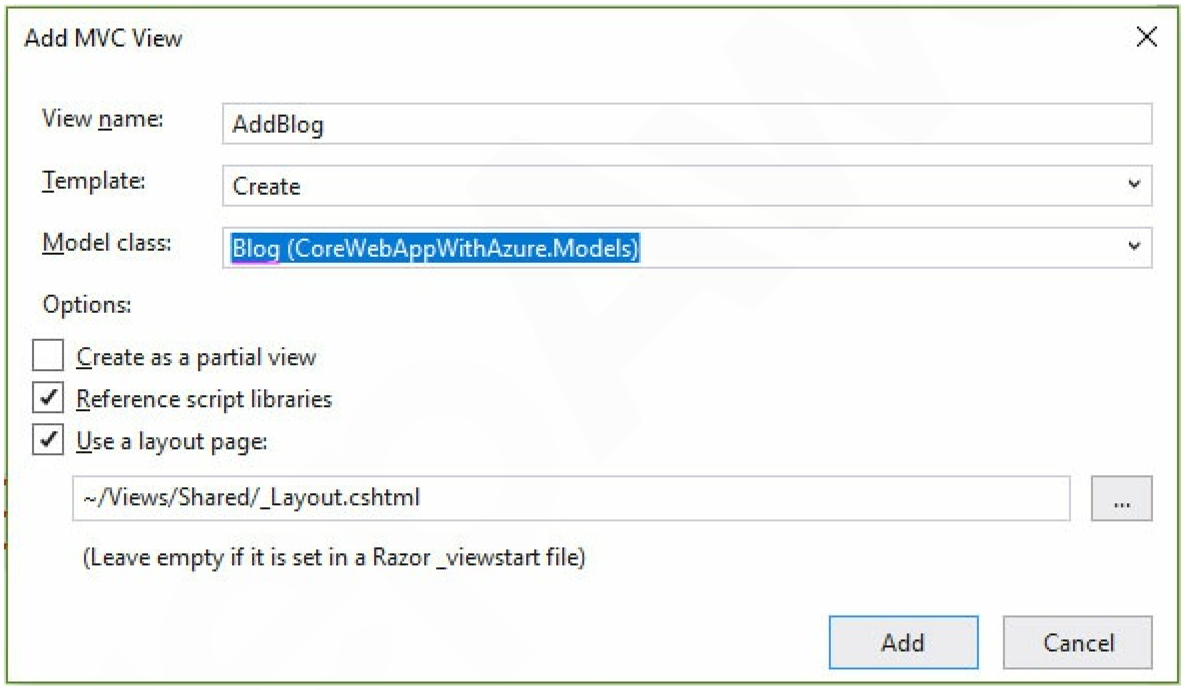
Let us add action for `Blogs` view in `BlogController`. I will create `BlogAsync` method. I will get blogs in draft mode using BlogService class. Action method will return a View and `blogitems` in view bag.



```
[ActionName("Blogs")]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult> BlogsAsync()
{
    var items = await BlogService.GetDraftBlogsAsync();
    return View(items);
}
```

Figure 3.20

Now I will add view for adding a `Blog`. Right click on `Blog` folder. Select **`Add | View`**. In **`Add MVC View`** pane, I will provide **`View Name`** as `AddBlog`, **`Template`** for view as **`Create`**, **`Model class`** as **`Blog`** class which I have created earlier. I will `select _layouts`.cshtml from shared view as layout page of view.



Add MVC View

View name: AddBlog

Template: Create

Model class: Blog (CoreWebAppWithAzure.Models)

Options:

- [ ] Create as a partial view
- [x] Reference script libraries
- [x] Use a layout page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

Figure 3.21

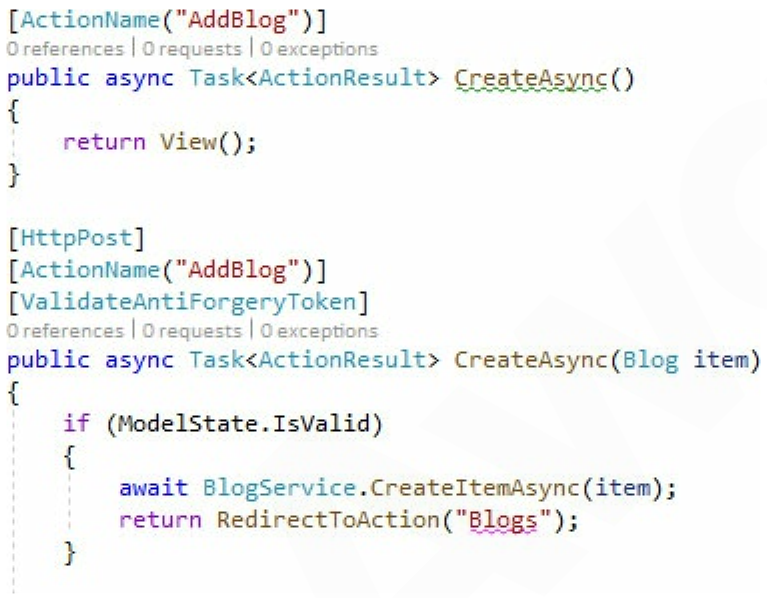
Click on **`Add`** to create a view. I will update code in `cshtml`.

```
@model CoreWebAppWithAzure.Models.Blog

@{
 ViewData[“Title”] = “AddBlog”;
 Layout = “~/Views/Shared/_Layout.cshtml”;
}

<h1>AddBlog</h1>

<h4>Blog</h4>
```

```
<hr />
<div class="row">
<div class="col-md-4">
<form asp-action="AddBlog">
<div asp-validation-summary="ModelOnly" class="text-danger">
</div>
<div class="form-group">
<label asp-for="Id" class="control-label"></label>
<input asp-for="Id" class="form-control" />
<span asp-validation-for="Id" class="text-danger"></span>
</div>
<div class="form-group">
<label asp-for="Title" class="control-label"></label>
<input asp-for="Title" class="form-control" />
<span asp-validation-for="Title" class="text-danger"></span>
</div>
<div class="form-group">
<label asp-for="Content" class="control-label"></label>
<input asp-for="Content" class="form-control" />
<span asp-validation-for="Content" class="text-danger"></span>
</div>
<div class="form-group">
<label asp-for="Author" class="control-label"></label>
<input asp-for="Author" class="form-control" />
<span asp-validation-for="Author" class="text-danger"></span>
</div>
<div class="form-group form-check">
<label class="form-check-label">
<input class="form-check-input" asp-for="IsDraft"
/>@Html.DisplayNameFor(model => model.IsDraft)
</label>
</div>
<div class="form-group">
<input type="submit" value="Create" class="btn btn-primary" />
</div>
</form>
</div>
</div>

<div>
<a asp-action="Index">Back to List</a>
</div>

@section Scripts {
@{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Now I will add the action method for the **`AddBlog`** view. **`CreateAsync`** action method with no parameters will get called when the user clicks on **`CreateBlog`** link. It will return a view to provide the **`Blog`** item information.

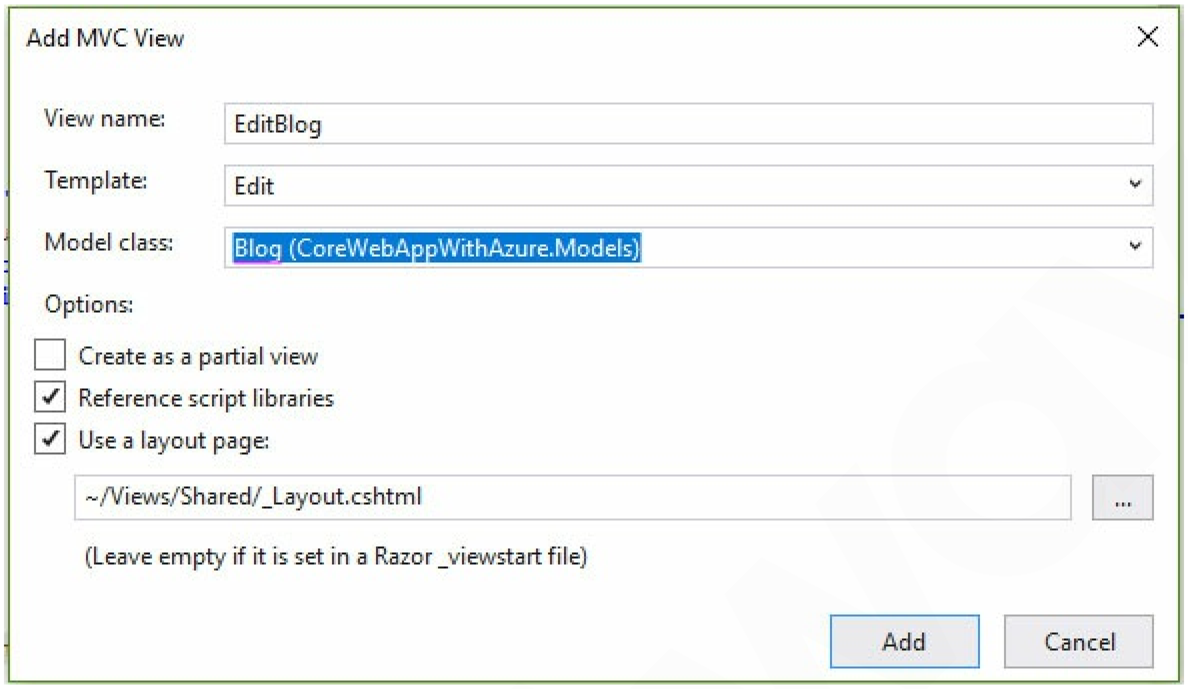
The user will provide **`Blog`** information and submit the form. It will post blog information to another **`CreateAsync`** method. It will validatethe model and call **`BlogService`** to create a new **`Blog`** item in the container:



```
[ActionName("AddBlog")]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult> CreateAsync()
{
    return View();
}

[HttpPost]
[ActionName("AddBlog")]
[ValidateAntiForgeryToken]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult> CreateAsync(Blog item)
{
    if (ModelState.IsValid)
    {
        await BlogService.CreateItemAsync(item);
        return RedirectToAction("Blogs");
    }
```

Figure 3.22

Now I will add a view for editing a `Blog` item. Right-click on the `Blog` folder. Select **`Add | View`**. In **`Add MVC View`** pane, I will provide **`View Name`** as `EditBlog`. **`Template`** for a view as `Edit`. **`Model class`** as `Blog` class, which we have created earlier. I will select `_layouts.cshtml` from Shared view as a layout page of view:


Add MVC View

View name: EditBlog

Template: Edit

Model class: Blog (CoreWebAppWithAzure.Models)

Options:

☐ Create as a partial view

☑ Reference script libraries

☑ Use a layout page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

Figure 3.23

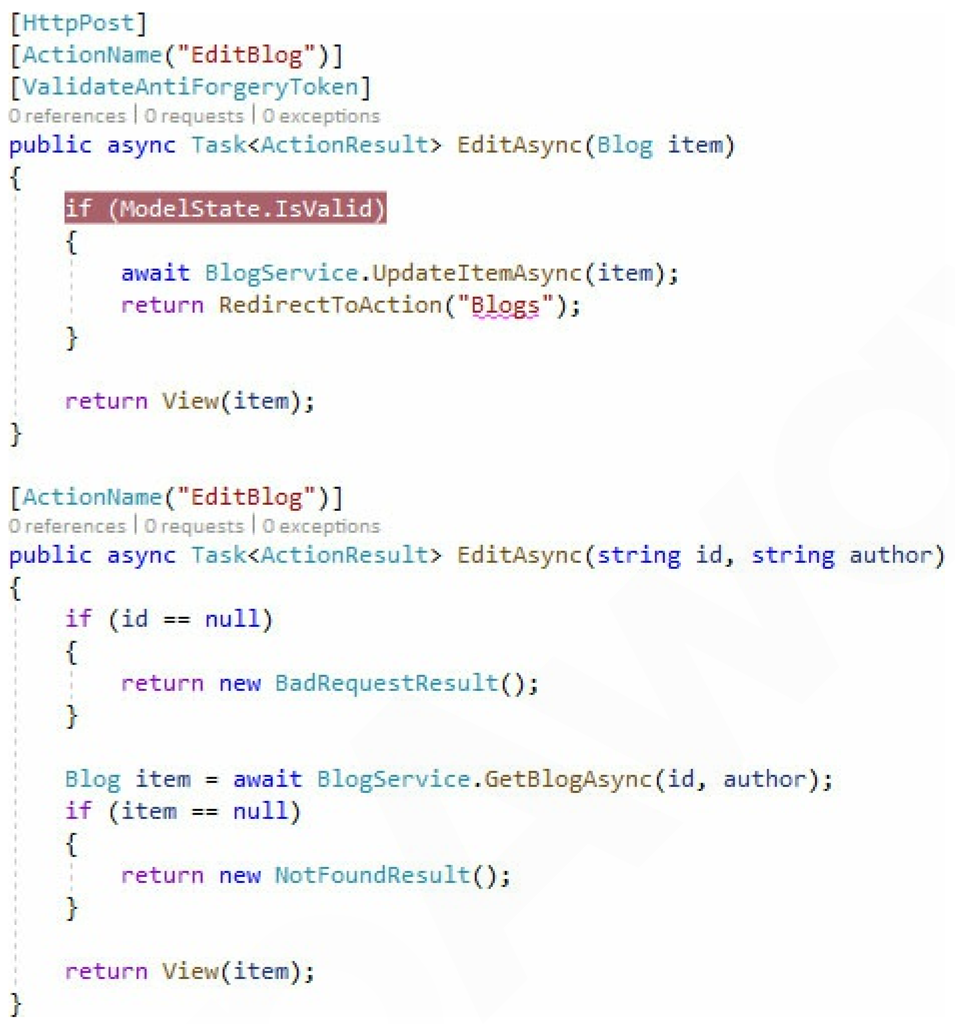
Click on **`Add`** to create **`Edit`** view of a **`Blog`** item. Update code in **`cshtml`**.

```
@model CoreWebAppWithAzure.Models.Blog

@{
 ViewData[“Title”] = “EditBlog”;
 Layout = “~/Views/Shared/_Layout.cshtml”;
}

<h1>EditBlog</h1>

<h4>Blog</h4>
<hr />
<div class=”row”>
<div class=”col-md-4”>
<form asp-action=”EditBlog”>
<div asp-validation-summary=”ModelOnly” class=”text-danger”>
</div>
<div class=”form-group”>
<label asp-for=”Id” class=”control-label”></label>
<input asp-for=”Id” class=”form-control” />
<span asp-validation-for=”Id” class=”text-danger”></span>
</div>
<div class=”form-group”>
<label asp-for=”Title” class=”control-label”></label>
<input asp-for=”Title” class=”form-control” />
<span asp-validation-for=”Title” class=”text-danger”></span>
```

```
</div>
<div class="form-group">
<label asp-for="Content" class="control-label"></label>
<input asp-for="Content" class="form-control" multiple />
<span asp-validation-for="Content" class="text-danger"></span>
</div>
<div class="form-group">
<label asp-for="Author" class="control-label"></label>
<input asp-for="Author" class="form-control" />
<span asp-validation-for="Author" class="text-danger"></span>
</div>
<div class="form-group form-check">
<label class="form-check-label">
<input class="form-check-input" asp-for="IsDraft"
/>@Html.DisplayNameFor(model => model.IsDraft)
</label>
</div>
<div class="form-group">
<input type="submit" value="Save" class="btn btn-primary" />
</div>
</form>
</div>
</div>

<div>
<a asp-action="Index">Back to List</a>
</div>

@section Scripts {
@{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Now I will add an action method for the `EditBlog` view. `EditAsync` action method with id and author parameter will get called when the user clicks on the `EditBlog` link. It will check `Blog` item with ID in a container and, if found, return a view to providing `Blog` item information.

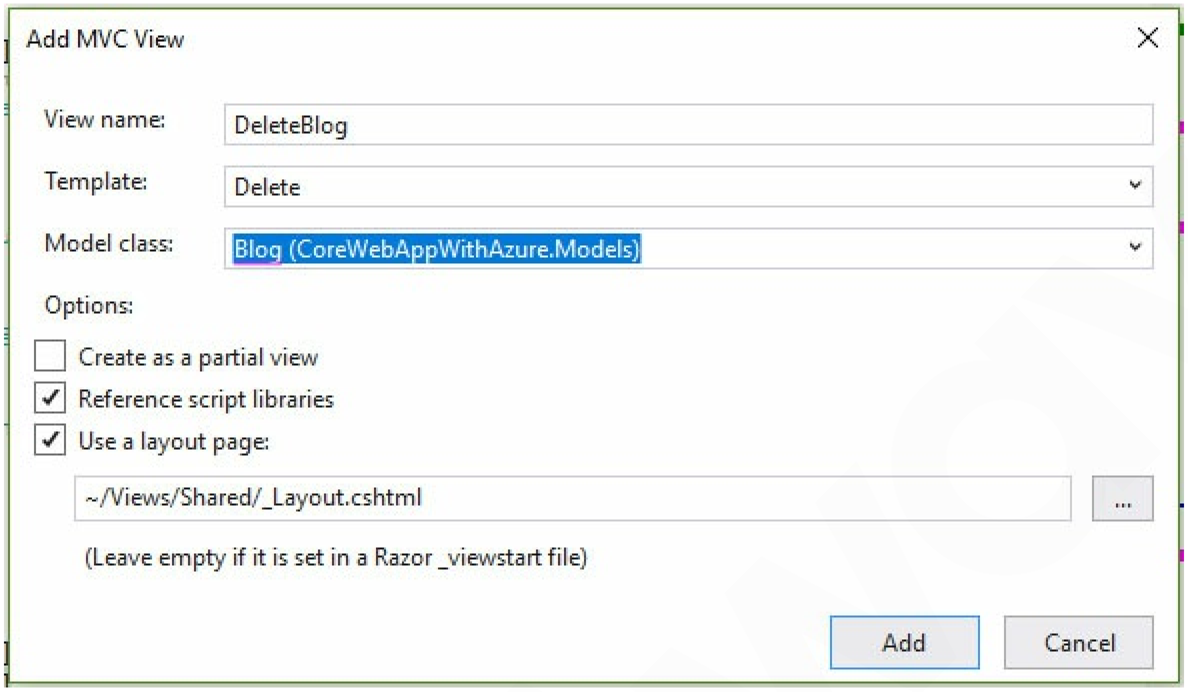
The user will update `Blog` information and submit the form. It will post blog information to another `EditAsync` method. It will validate the model and call `BlogService` to update the respective `Blog` item in the container.



```
[HttpPost]
[ActionName("EditBlog")]
[ValidateAntiForgeryToken]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult> EditAsync(Blog item)
{
    if (ModelState.IsValid)
    {
        await BlogService.UpdateItemAsync(item);
        return RedirectToAction("Blogs");
    }

    return View(item);
}

[ActionName("EditBlog")]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult> EditAsync(string id, string author)
{
    if (id == null)
    {
        return new BadRequestResult();
    }

    Blog item = await BlogService.GetBlogAsync(id, author);
    if (item == null)
    {
        return new NotFoundResult();
    }

    return View(item);
}
```

Figure 3.24

Now I will add the view for deleting a `Blog` item. Right-click on the `Blog` folder. Select **`Add | View`**. In **`Add MVC View`** pane, I will provide **`View Name`** as **`DeleteBlog`**, **`Template`** for the view as **`Delete`**, **`Model class`** as **`Blog`** class, which we have created earlier. I will select `_layouts.cshtml` from **`Shared`** view as a layout page of view.



View name: DeleteBlog
Template: Delete
Model class: Blog (CoreWebAppWithAzure.Models)
Options:
- [ ] Create as a partial view
- [✓] Reference script libraries
- [✓] Use a layout page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

Figure 3.25

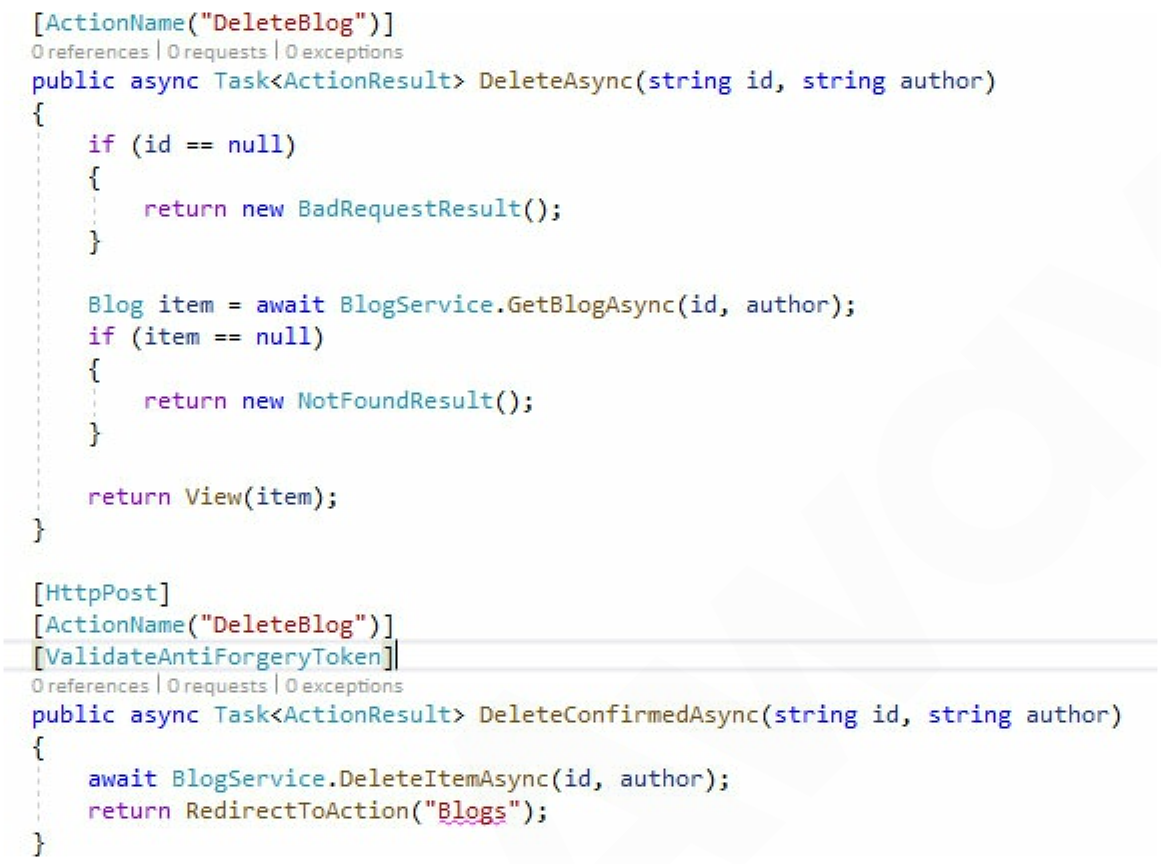
Update code in view:

```
@model CoreWebAppWithAzure.Models.Blog

@{
 ViewData["Title"] = "DeleteBlog";
 Layout = "~/Views/Shared/_Layout.cshtml";
}

<h1>DeleteBlog</h1>

<h3>Are you sure you want to delete this?</h3>
<div>
<h4>Blog</h4>
<hr />
<dl class="row">
<dt class = "col-sm-2">
@Html.DisplayNameFor(model => model.Id)
</dt>
<dd class = "col-sm-10">
@Html.DisplayFor(model => model.Id)
</dd>
<dt class = "col-sm-2">
@Html.DisplayNameFor(model => model.Title)
</dt>
<dd class = "col-sm-10">
@Html.DisplayFor(model => model.Title)
</dd>
```

```
<dt class = “col-sm-2”>
@Html.DisplayNameFor(model => model.Content)
</dt>
<dd class = “col-sm-10”>
@Html.DisplayFor(model => model.Content)
</dd>
<dt class = “col-sm-2”>
@Html.DisplayNameFor(model => model.Author)
</dt>
<dd class = “col-sm-10”>
@Html.DisplayFor(model => model.Author)
</dd>
<dt class = “col-sm-2”>
@Html.DisplayNameFor(model => model.IsDraft)
</dt>
<dd class = “col-sm-10”>
@Html.DisplayFor(model => model.IsDraft)
</dd>
</dl>
<form asp-action=”DeleteBlog”>
<input type=”submit” value=”Delete” class=”btn btn-danger” /> |
<a asp-action=”Index”>Back to List</a>
</form>
</div>
```

Now close all `cshtml` files of views. I will add action method for `DeleteBlog` view. When user clicks on delete, `DeleteConfirmAsync` action method will get called. It will return a view to provide confirmation before deleting an item in container.

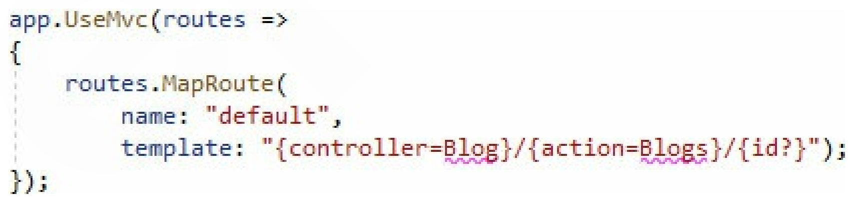
When delete is confirmed `DeleteAsync` method will be called. It will validate the model and call `BlogService` to `Deleteblog` item in the container:



```
[ActionName("DeleteBlog")]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult> DeleteAsync(string id, string author)
{
    if (id == null)
    {
        return new BadRequestResult();
    }

    Blog item = await BlogService.GetBlogAsync(id, author);
    if (item == null)
    {
        return new NotFoundResult();
    }

    return View(item);
}

[HttpPost]
[ActionName("DeleteBlog")]
[ValidateAntiForgeryToken]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult> DeleteConfirmedAsync(string id, string author)
{
    await BlogService.DeleteItemAsync(id, author);
    return RedirectToAction("Blogs");
}
```

Figure 3.26

We have completed all the development parts. Now I will configure `Blogs` as the default route for our web application. I will open the `Startup.cs` and navigate to the `Configure` method. Locate the `ap.usemvc` method call. Update template attribute controller to `Blog` and action to `Blogs`.

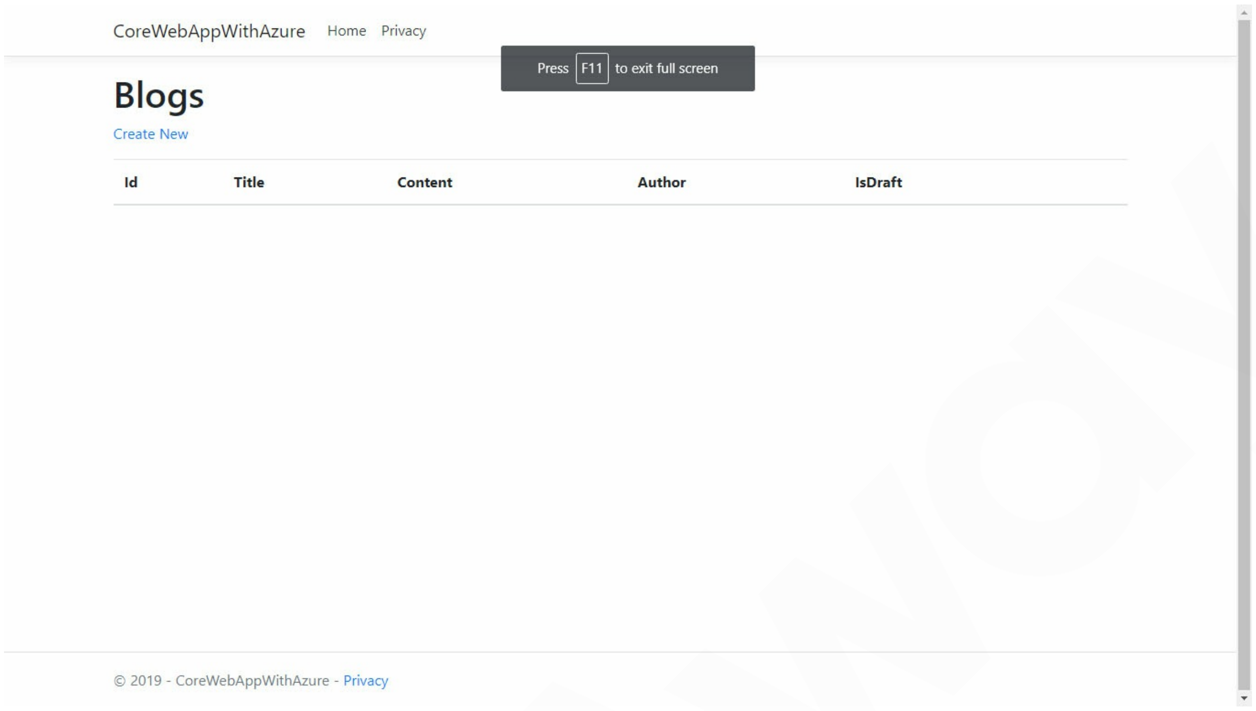


```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Blog}/{action=Blogs}/{id?}");
});
```

Figure 3.27

Our application is ready to add, update, and delete blog items. I will go to solution explorer, right-click on the project, and select **`debug`**. It will host our application to the IIS express and launch the browser.

It will show the **`Blogs`** list view. I have not added any blog to the container; hence the table is empty. I will click on the **`Create New`** link to create a blog.


CoreWebAppWithAzure Home Privacy

Press F11 to exit full screen

Blogs

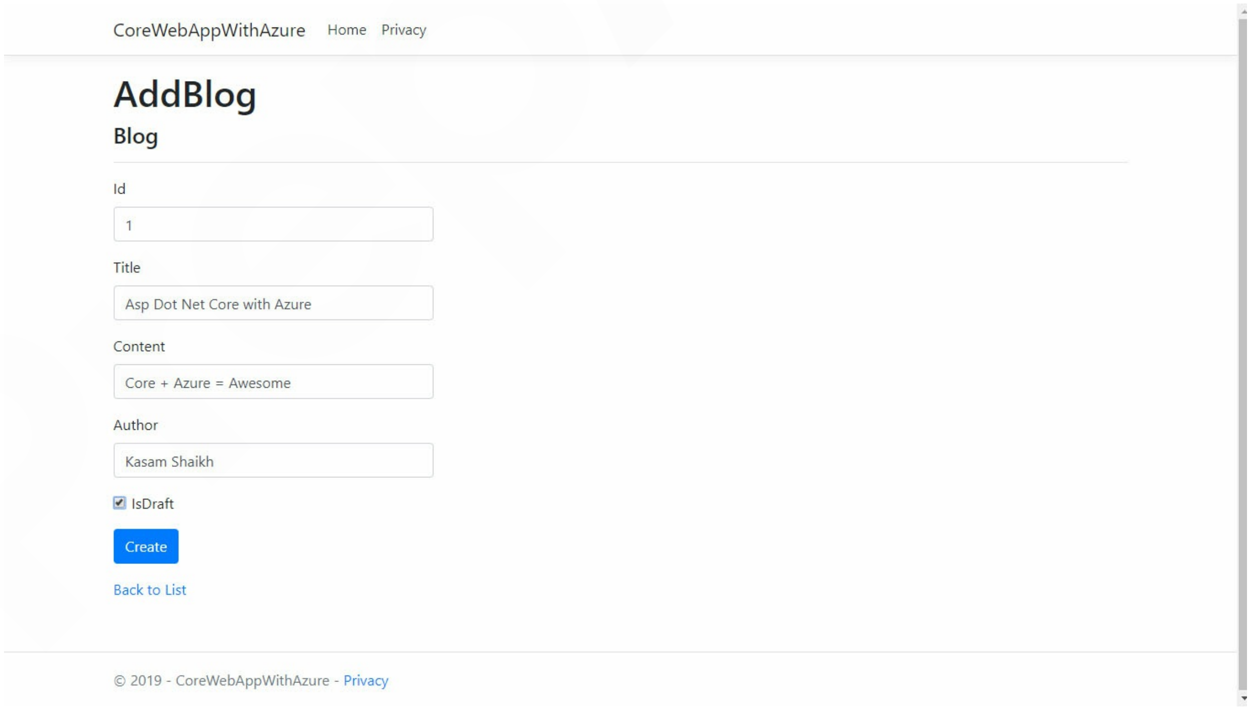
Create New

Id	Title	Content	Author	IsDraft

© 2019 - CoreWebAppWithAzure - Privacy

Figure 3.28

It will redirect us to the **`AddBlog`** view. I will add **`Blog`** information. I will check the **`IsDraft`** checkbox to keep the blog in draft mode. Click on **`Create`** to add Blog Item to the container.


CoreWebAppWithAzure Home Privacy

AddBlog

Blog

Id: 1

Title: Asp Dot Net Core with Azure

Content: Core + Azure = Awesome

Author: Kasam Shaikh

[✓] IsDraft

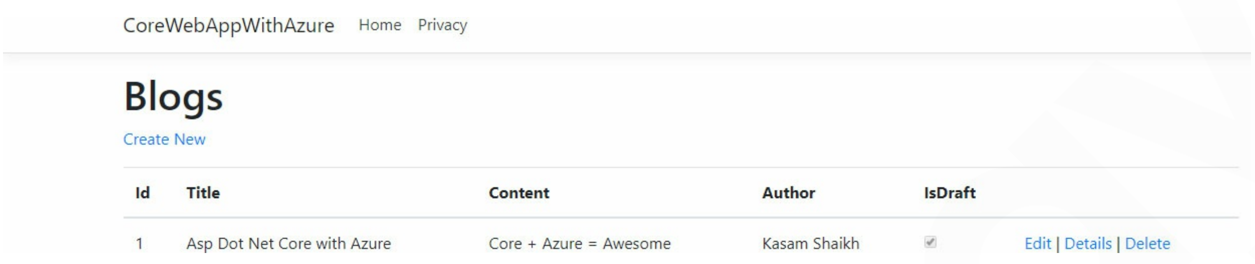
Create

Back to List

© 2019 - CoreWebAppWithAzure - Privacy

Figure 3.29

It will add the blog item to the container and redirect us to the **`Blogs`** list view. Here you can view **`Blog`** item that I have added. In the last column, **`Edit`** and **`Delete`** links are available. I will click on **`Edit`** to update blog information:


CoreWebAppWithAzure Home Privacy

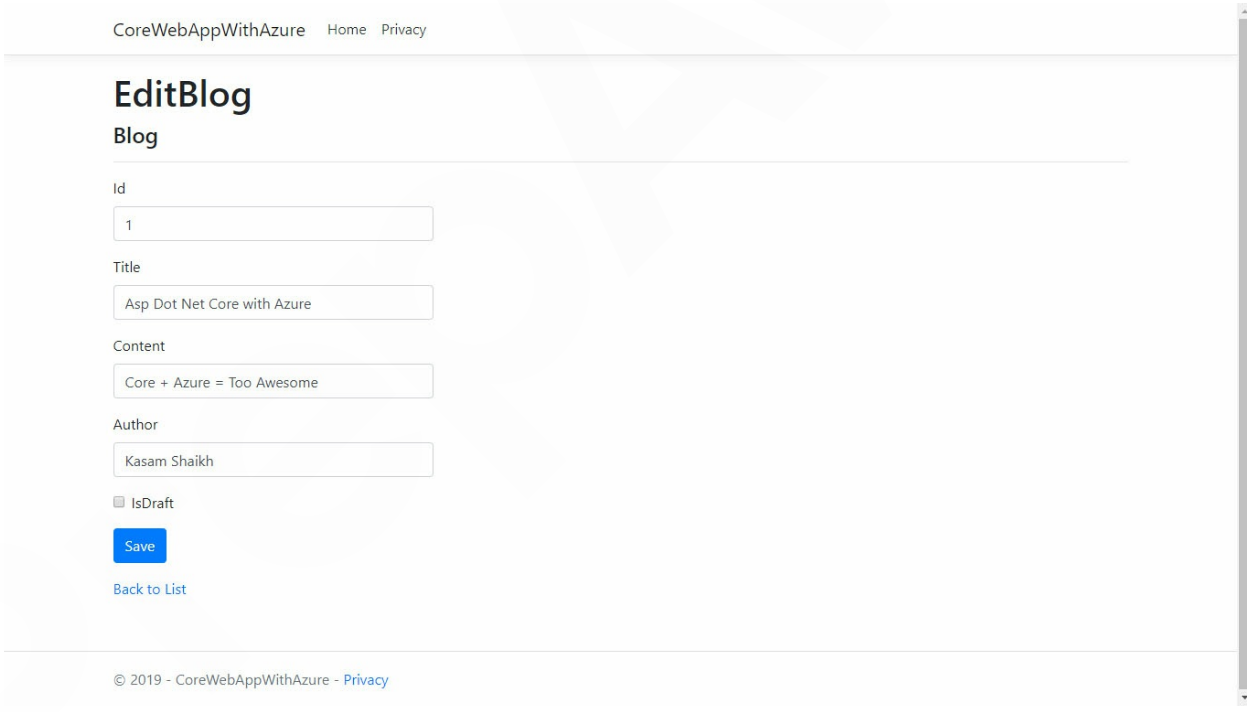
Blogs

Create New

Id	Title	Content	Author	IsDraft	
1	Asp Dot Net Core with Azure	Core + Azure = Awesome	Kasam Shaikh	✓	Edit | Details | Delete

Figure 3.30

On the edit page, I will update content to **`Core + Azure = Too Awesome`** and uncheck the **`IsDraft`** checkbox. I will click on **`Submit`**. It will call the `EditBlogAction` method, and the blog item will be updated in the container. It will redirect us to the **`Blog`** lists page.


CoreWebAppWithAzure Home Privacy

EditBlog

Blog

Id: 1

Title: Asp Dot Net Core with Azure

Content: Core + Azure = Too Awesome

Author: Kasam Shaikh

IsDraft

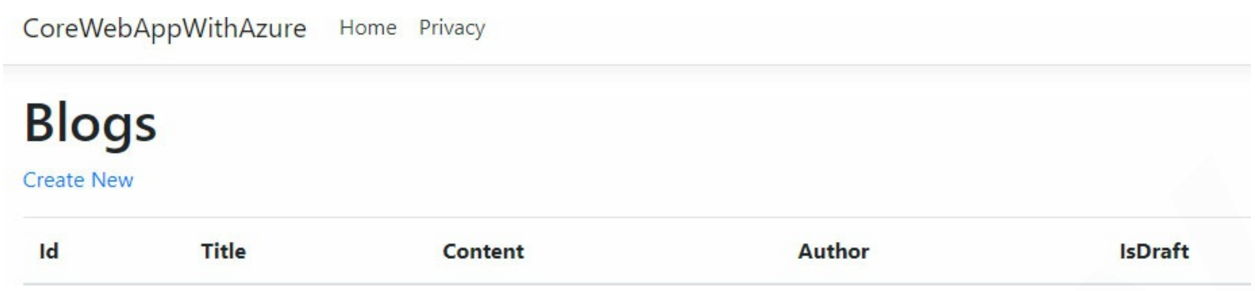
Save

Back to List

© 2019 - CoreWebAppWithAzure - Privacy

Figure 3.31

On the **`Blogs`** list view, no blog Items will be shown. I have unchecked the **`IsDraft`** checkbox of a blog item. We are showing only draft mode **`Blogs`** on list view:


CoreWebAppWithAzure Home Privacy

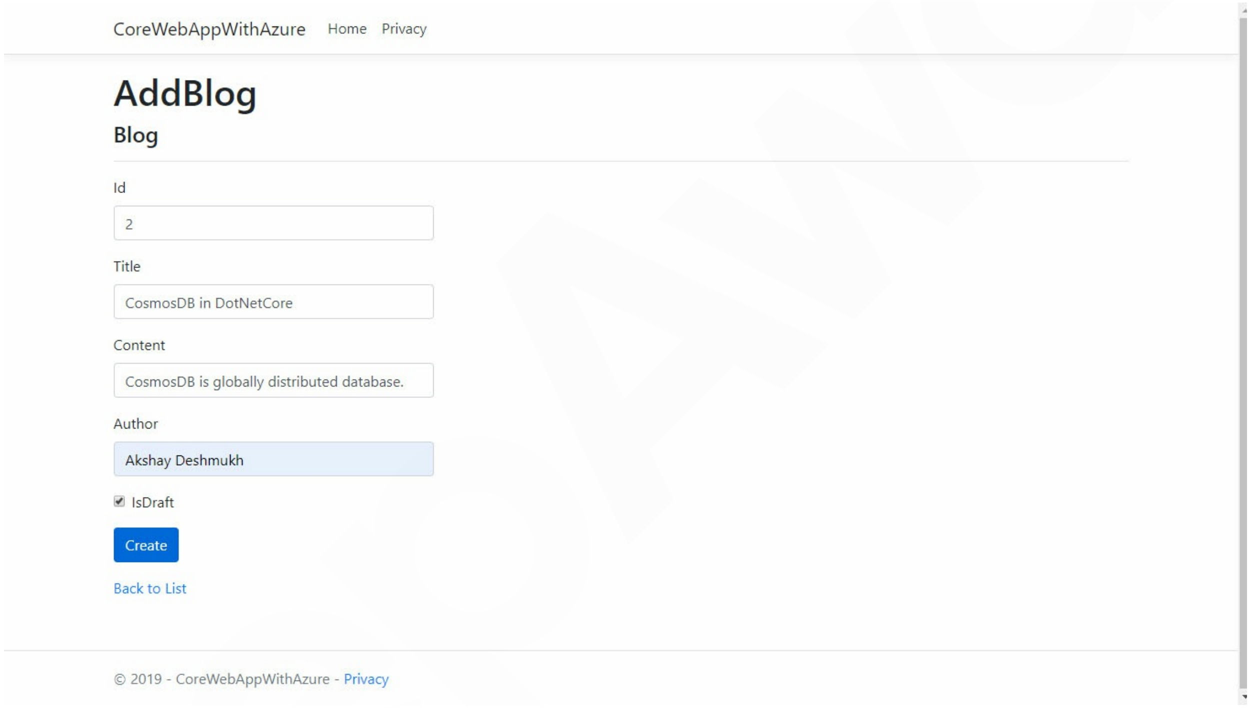
Blogs

Create New

Id	Title	Content	Author	IsDraft

Figure 3.32

Similarly, let’s add one blog in the container using the **`AddBlog`** view. I will add information on the page and click on **`Create`**.


CoreWebAppWithAzure Home Privacy

AddBlog

Blog

Id: 2

Title: CosmosDB in DotNetCore

Content: CosmosDB is globally distributed database.

Author: Akshay Deshmukh

☑ IsDraft

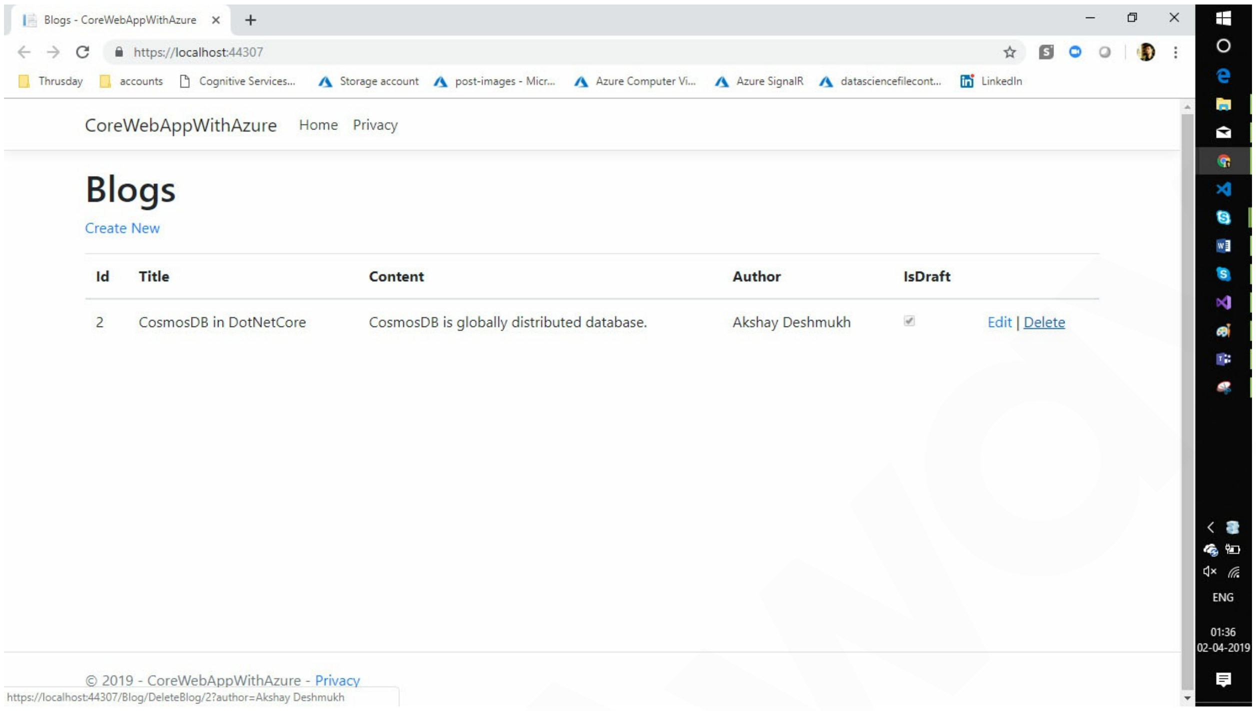
Create

Back to List

© 2019 - CoreWebAppWithAzure - Privacy

Figure 3.33

The added blog is in draft mode; hence it will be shown in the list view. Now let us click on **`Delete`** link in front of the blog item added. It will redirect us to **`DeleteBlog`** confirmation view:



Blogs

Create New

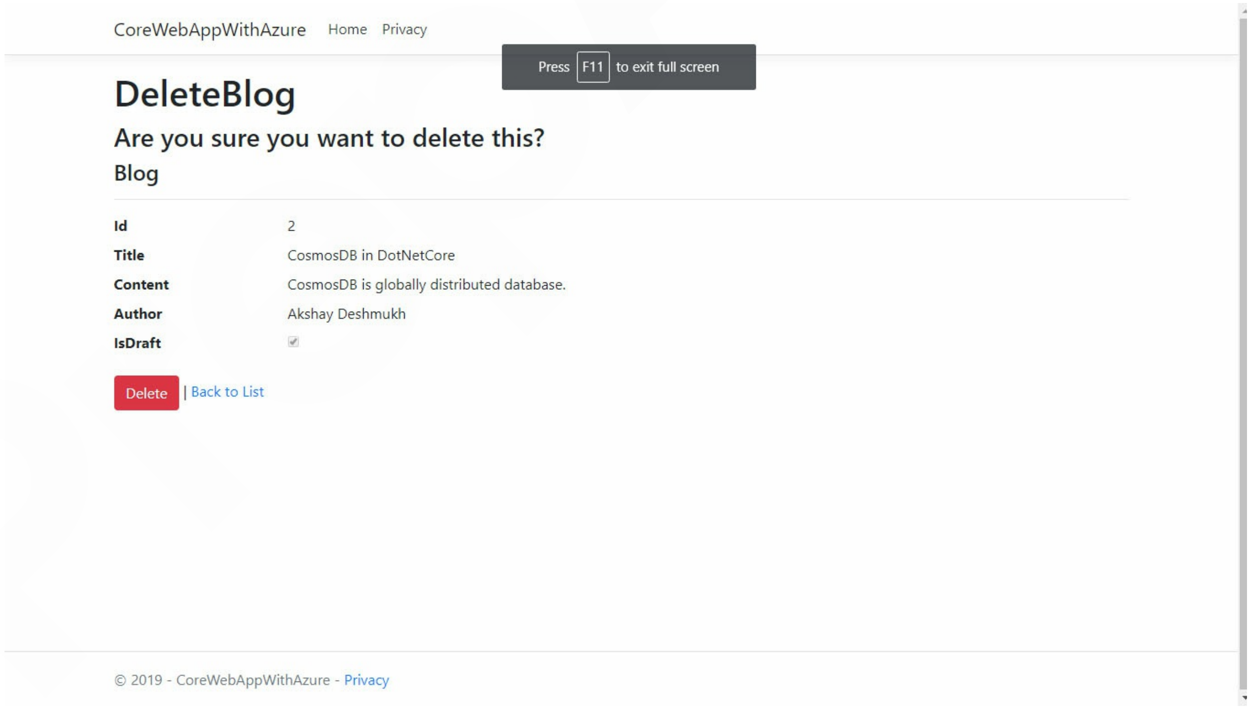
Id	Title	Content	Author	IsDraft	
2	CosmosDB in DotNetCore	CosmosDB is globally distributed database.	Akshay Deshmukh	✓	Edit | Delete

© 2019 - CoreWebAppWithAzure - Privacy

https://localhost:44307/Blog/DeleteBlog/2?author=Akshay Deshmukh

Figure 3.34

To confirm the delete operation, I will click on the **`Delete`** button. It will call delete action and delete a blog item from the container:



DeleteBlog

Are you sure you want to delete this?

Blog

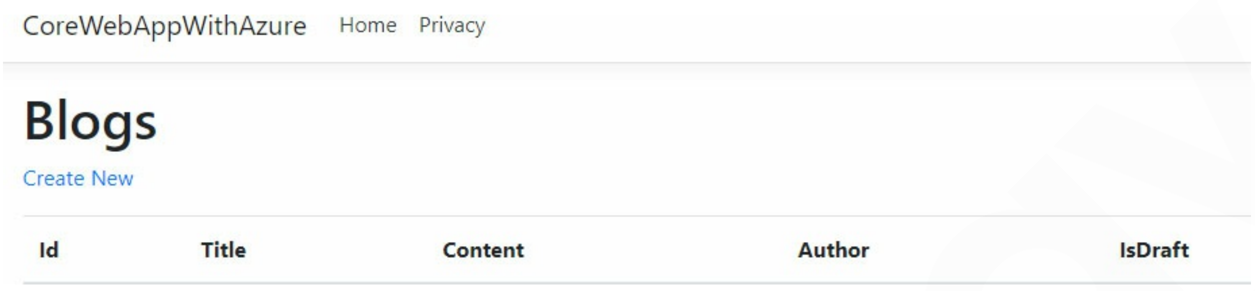
Id	2
Title	CosmosDB in DotNetCore
Content	CosmosDB is globally distributed database.
Author	Akshay Deshmukh
IsDraft	✓

Delete | Back to List

© 2019 - CoreWebAppWithAzure - Privacy

Figure 3.35

After deleting **`Blog`** information, we will be redirected to the list view page. It will show blank list because we have deleted blog item and previous blog item is not in draft mode:



Blogs

Id	Title	Content	Author	IsDraft

Figure 3.36

Things to explore more

In this chapter, we went through the steps and learned how seamlessly we could perform CRUD operation with Azure Cosmos Database. Now another important thing for you to explore is about Azure Cosmos DB consistency levels. Though this topic is more relevant to Architect and professionals involved in designing the solution architecture, I would recommend you to go through this topic. But as a reader, I would suggest touching all the aspects of the topic, and therefore we have provided such a detailed explanation.

Azure Cosmos DB approaches the data consistency with the range of options. Starting with consistency as **`Strong`**, it reaches the extreme end by having consistency as **`Eventual`**. It also includes bounded staleness, session, and consistent prefix. The strongest being the **`Strong Consistency`** model and the eventual to be the much **`Weaker Consistency`**. The following image depicts the models with respect to its trade-offs pertaining to high availability and performance.

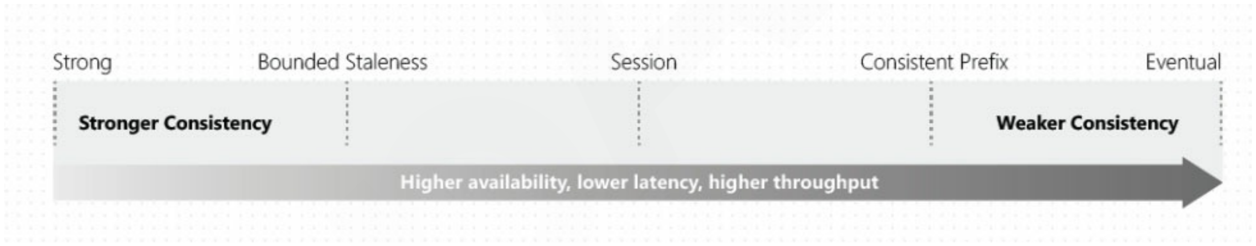
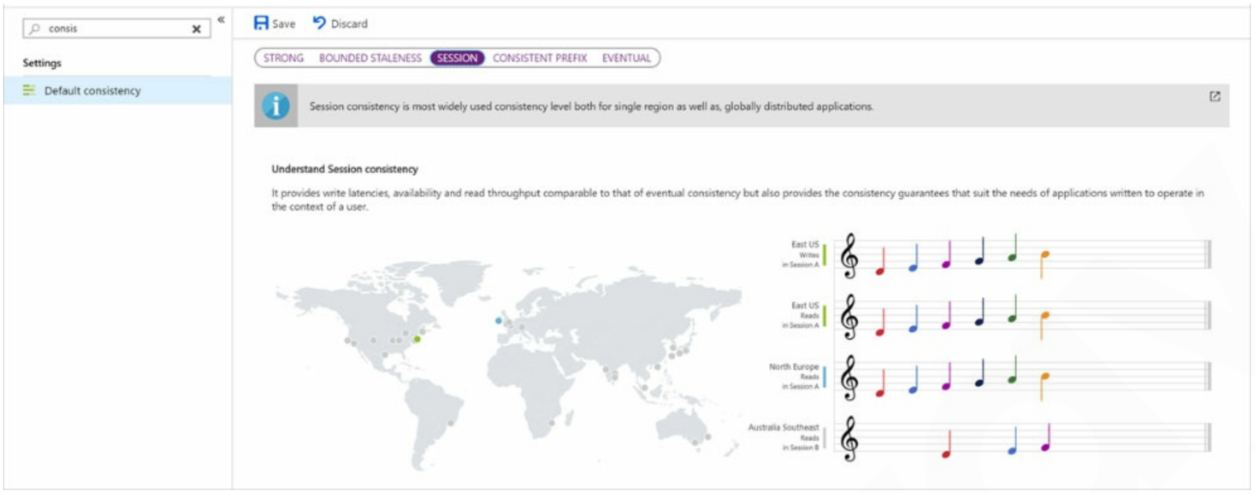


Figure 3.37: *Image source: Microsoft Docs.*

You can use among these available consistency options to make your flow highly available with high performing.

For configuring the same, you have to just navigate to **Settings | Default** consistency from the left-hand pane. And just hit from the desired model options:



Settings

Default consistency

STRONG BOUNDED STALENESS [✓ SESSION] CONSISTENT PREFIX EVENTUAL

Session consistency is most widely used consistency level both for single region as well as, globally distributed applications.

Understand Session consistency

It provides write latencies, availability and read throughput comparable to that of eventual consistency but also provides the consistency guarantees that suit the needs of applications written to operate in the context of a user.

Figure 3.38

The important thing here to note is, **`Default consistency`**, when you create an Azure Cosmos DB resource is **`Session`**. Remember, this is one of my favorite interview questions asked for Azure Developer.

Conclusion

In this chapter, you studied what Azure Cosmos DB is, how to create and manage the resource and work with CRUD operations. I will recommend you to explore more on consistency levels in Azure Cosmos DB along with an interview tip. Also, advice to you will be to repeat the exercise detailed in the chapter to make yourself more confident to work with Azure Cosmos DB. In the next chapter, I will explore more about another interesting offering from Microsoft Azure, that is, Microsoft Azure Storage.

Questions

1. What are the Key benefits of Azure CosmosDB?
2. What are the different Consistency levels in Azure CosmosDB?
3. What is the default consistency level in Azure CosmosDB?
4. What do you mean by Sharding of database?
5. Can you use the Graph database in Azure CosmosDB?
6. Explain the difference between NoSQL vs. Relational database?

CHAPTER 4

Working With Microsoft Azure Storage

In the previous chapter, we studied all about Azure Cosmos DB, its creation, management along with working with the service. And I believe you must have followed the given exercise as well. In this chapter, I will walk you through one of the most important and widely used Azure services as a storage option, name **Microsoft Azure Storage.**

When we have an application, it comes with having a storage solution to be designed with it. Now storage can be anything document, image files, and so on. Microsoft Azure offers a dynamic storage option with Azure Storage service. This service can be used as a storage for data, storage for diagnostic settings for applications, storage for Azure **Virtual Machines (VMs),** storage as a drive for your application hosted in Azure VMs, along with hosting the static contents. To be more precise, Microsoft Azure Storage is highly available, secure, scalable, easily managed, and securely accessible.

Structure

- Azure Storage services
- Create a Microsoft Azure Storage account
- Working with Azure SDKs
- Managing Azure Blob from application

Objectives

After reading this chapter, you will learn:

- Introduction to Microsoft Azure Storage
- Creating and Configuring Azure Storage
- Implementation of Azure Storage service in .NET core application
- Manging the functional flow in the application

Azure Storage services

When you have a storage account resource created, it offers the main four data services, popularly known and explained as follows:

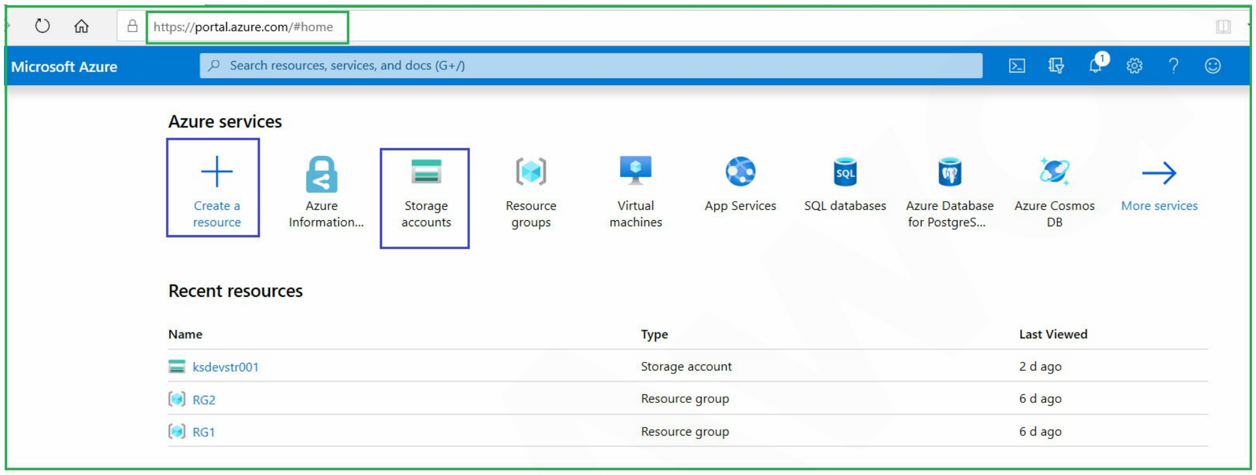
- **Azure Blobs:** Azure Blob Storage is massively used for saving unstructured data, such as binary data, files, image files, and so on. As this is one of the widely used services, I will dive you more on Blob storage in later part of this chapter.
- **Azure Files:** Azure Files, also known as **Azure Files** shares, as the name **depicts**, are widely used as a network share solution, access by using the **Server Message Block (SMB)** protocol. These shared files can be used by multiple VMs just by mounting it as a drive-in any given machine. The most common scenarios where Azure File services are used are when the solution demands a hybrid architecture on-premises and Azure VMs to share the same file share as storage. Another most used scenario where the dev team is spread across different geographical locations, and they need to share common utilities, dev tools, and so on. In this case, a common file share is the best option to go with.

  The important point here to note is, what makes Azure File share different from other corporate file share options. Answer to this is, Azure File share is accessible via Rest APIs, via URLs globally, and securely using **Shared Access Signature (SAS)** token. Any files in the file share can also be accessed via URL securely, and this makes Azure File share different from all other storage network share options. Again this is our favorite interview question being always asked any Azure developer.
- **Azure Queues:** Queues are used to store a message that needs to be asynchronously processed, more like storing and retrieving the messages. Each message can be of size up to 64 KB. And millions of messages can be injected in a queue.
- **Azure Tables:** Table Storage, though it comes as the offering from Azure Storage, is now part of Azure Cosmos DB.

We will learn more about Azure Storage in the later section during the exercise of Azure Storage creation.

Create a Microsoft Azure Storage account

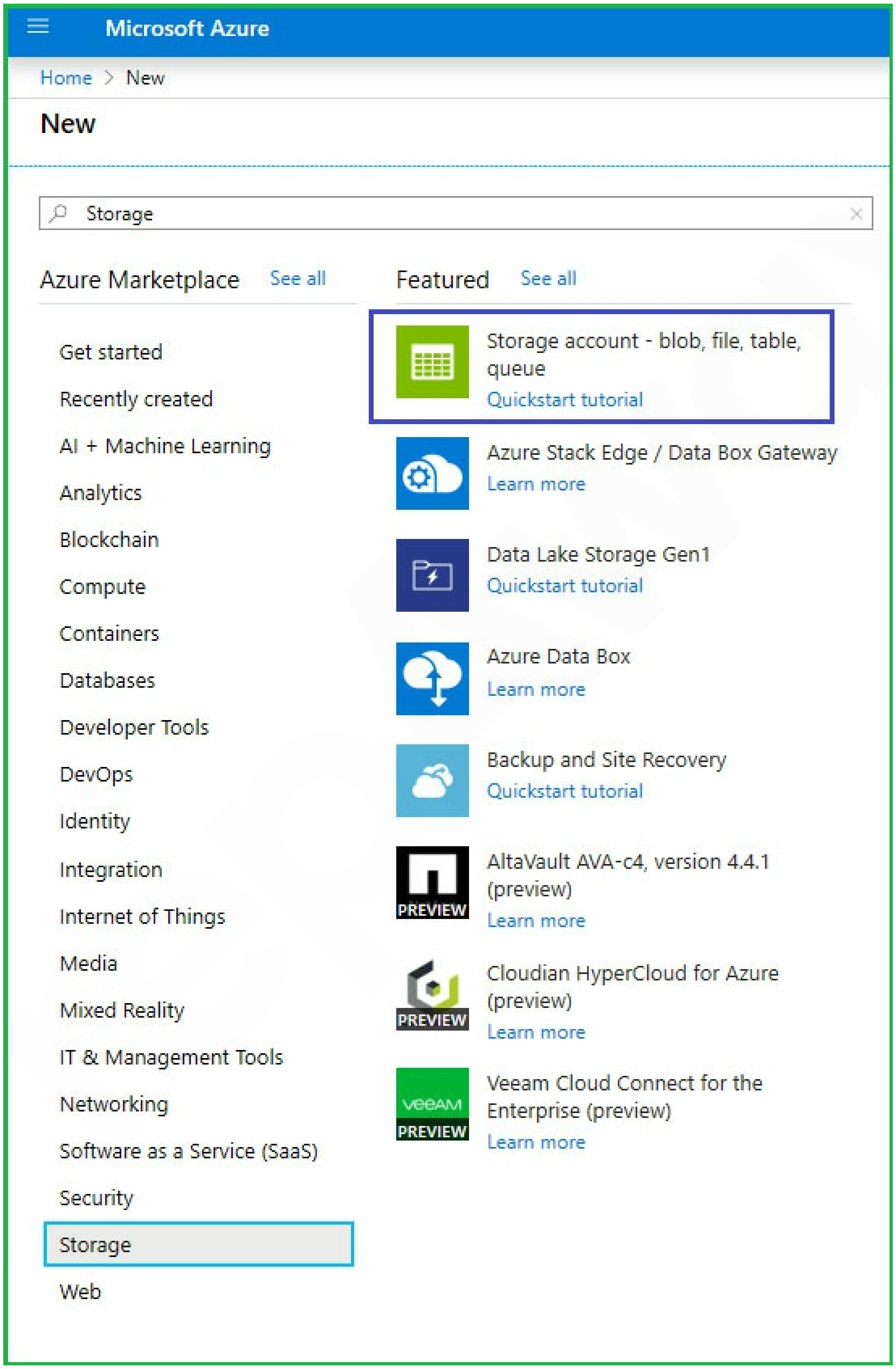
For any resource creation, open up Microsoft Azure portal in any browser, and click on **`Create a resource`** or **`Storage`** from listed Azure services. The new UI at the time of writing this book comes with easy access to most used **`Azure services, Recent resources`**, and so on.



Name	Type	Last Viewed
ksdevstr001	Storage account	2 d ago
RG2	Resource group	6 d ago
RG1	Resource group	6 d ago

Figure 4.1

Now, from services listed in **`Azure Marketplace`**, select **`Storage`** from **`left pane | Storage account`**.



Microsoft Azure

Home > New

New

Storage

Azure Marketplace See all

Get started
Recently created
AI + Machine Learning
Analytics
Blockchain
Compute
Containers
Databases
Developer Tools
DevOps
Identity
Integration
Internet of Things
Media
Mixed Reality
IT & Management Tools
Networking
Software as a Service (SaaS)
Security
Storage
Web

Featured See all

Storage account - blob, file, table, queue
Quickstart tutorial

Azure Stack Edge / Data Box Gateway
Learn more

Data Lake Storage Gen1
Quickstart tutorial

Azure Data Box
Learn more

Backup and Site Recovery
Quickstart tutorial

AltaVault AVA-c4, version 4.4.1 (preview)
Learn more

Cloudian HyperCloud for Azure (preview)
Learn more

Veeam Cloud Connect for the Enterprise (preview)
Learn more

Figure 4.2

Once you select the **`Storage`** service, you will be presented with a form in the blade to enter details with respect to your storage account.

It comes with five tabs to configure details with, namely, **`Basics, Networking, Advanced, Tags, Review + Create`**.

We will have a look into these as we proceed further.

The **`Basic`** requires mandatory project details as follows:

- **`Subscription:`** Here, you need to select your subscription. I have selected our `free trial` account. Now I purposely selected the free trial, as I wanted to ensure you the power of free accounts. To show you that you can very well start with your journey of learning Azure.
- **`Resource group:`** It's a logical grouping of your Azure resources. You can select from an existing resource group or can create a new resource group, as per your choice. For this article, I have created a new resource group name, `rg-AzurewithCore`.

Next comes the instance details:

- **`Storage Account Name:`** The name must be unique across all existing storage account names in Azure. It must be 3 to 24 characters long and can contain only lowercase letters and numbers. For this chapter, I am giving the name as `azurewithcore`.
- **`Location:`** Region, your storage azure resource, will get deployed. This can be from widely available azure regions and as per your target location. For this chapter, I have selected as the European region.
- **`Performance:`** This comes with two options, Standard, backed by magnetic drives, and **`Premium`**, backed with SSD drives. This could be considered as one of the pricing factors. For this chapter, I have selected as Standard, the default one.
- **`Account Kind:`** This is a very important detail to configure. This comes with three different options to select with:
  - **`General-purpose v2 accounts:`** This storage is mostly recommended account type to go for any application. This storage account type for used for **`blobs, files, queues`**, and **`tables`**.
  - **`General-purpose v1 accounts:`** It is a very legacy kind account

type for blobs, files, queues, and tables.

- **`BlobStorage accounts`:** Can be defined again as a legacy Blob-only storage account.

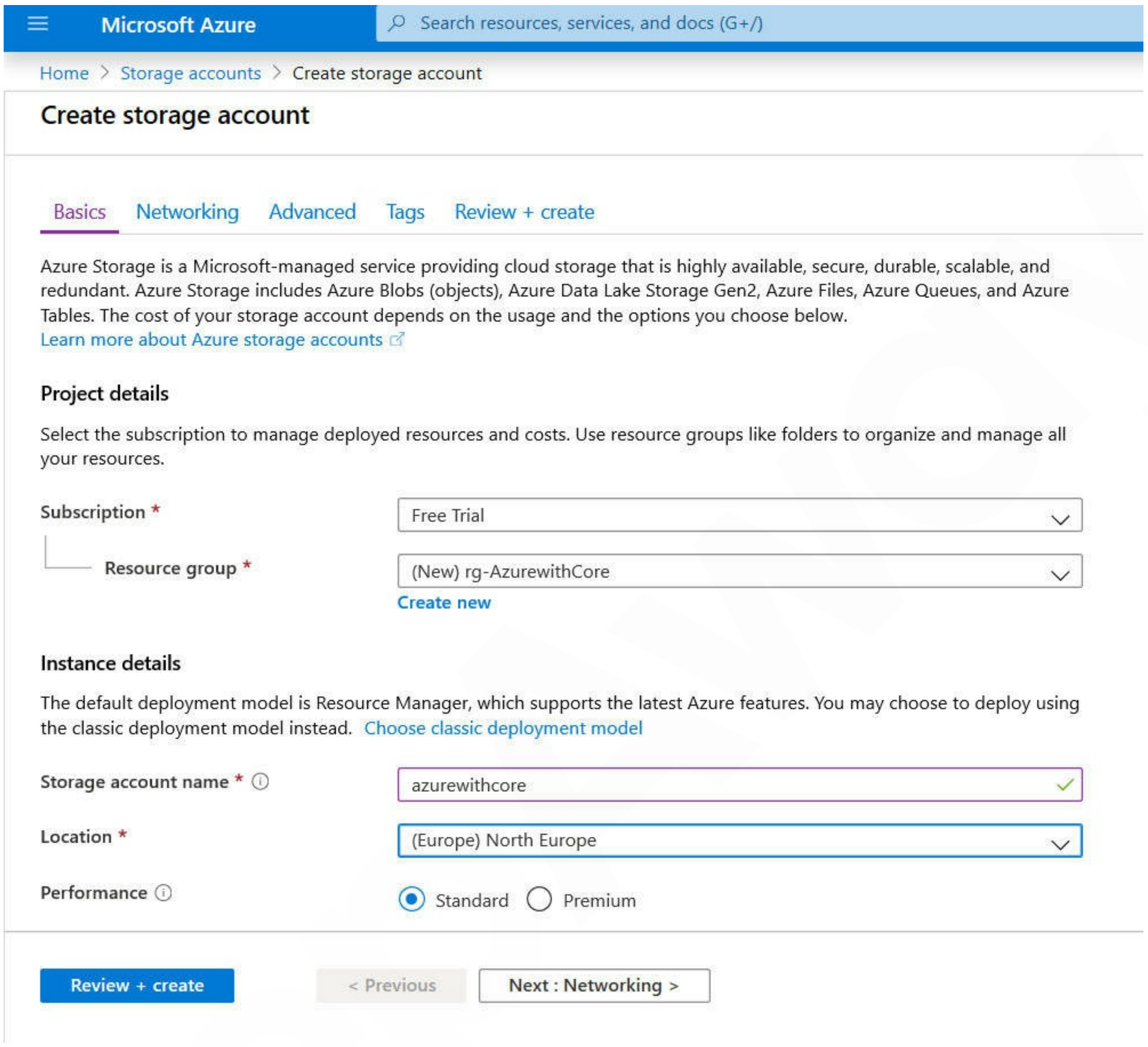
For this chapter, I will be selecting the most recommended storage kind, **`General-purpose v2`** accounts:

- **`Replication`:** This is the pattern of data replication being carried out with the storage by Azure, mainly used to ensure durability and high availability. This comes with the following different replication options for storage.
    - **Locally-redundant storage (LRS):** Data here are replicated synchronously thrice within the primary region. It is considered as a simple and low-cost strategy for replication.
    - **Zone-redundant storage (ZRS):** Data here are replicated synchronously across there Azure availability zones in the primary region. It is considered when the application demands high availability.
    - **Geo-redundant storage (GRS):** Data here is replicated synchronously thrice in the primary region, then replicated asynchronously to the secondary region. It is considered to deal with protection against regional outages. The secondary region can be read by enabling **read-access geo-redundant storage (RA-GRS)**.
    - **Geo-zone-redundant storage (GZRS) (preview):** Data here is replicated synchronously across three Azure availability zones in the primary region, then replicated asynchronously to the secondary region. It is considered when the application demands both availabilities along with maximum durability. At the time of writing this chapter, this option is in preview.

For this chapter, I will go with the default option selected:

- **`Access tier`:** This option only comes in when you select account kind as **`General-purpose v2 accounts`**. Again, it is one of the factors incurring costing. It all decides the nature of your use with data stored in the storage account. Cool tier is mainly used for archive type of data, rarely used. Hot comes with the data used frequently. And note, this is one of the interview questions for Azure developer. For this chapter, I will go with default as **`Hot`** tier.

Once all the details are entered, click on **`Next`** for entering details required in other tabs, click on **`Networking`** to proceed:


Microsoft Azure
Search resources, services, and docs (G+/)
Home > Storage accounts > Create storage account

Create storage account

Basics Networking Advanced Tags Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below.
Learn more about Azure storage accounts

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Free Trial

Resource group * (New) rg-AzurewithCore

Create new

Instance details

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. Choose classic deployment model

Storage account name * azurewithcore

Location * (Europe) North Europe

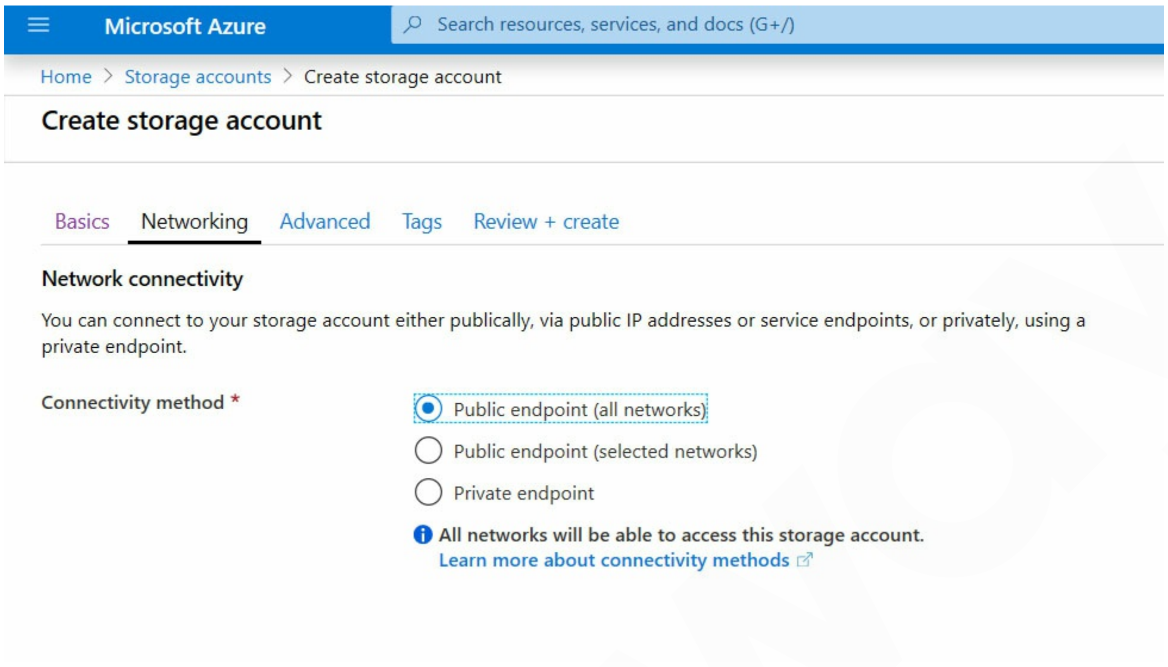
Performance [✓ Standard] Premium

Figure 4.3

`Networking` mainly deals with the connectivity of your Azure Storage account with other resources. You can connect to your storage account either publically, via public IP addresses or service endpoints, or privately, using a private endpoint.

The option will present you with three different connectivity methods:

- **`Public endpoint (all networks):`** Allows all networks to connect the storage. For this chapter, I will go with this option, which comes as default:



Microsoft Azure | Search resources, services, and docs (G+/)

Home > Storage accounts > Create storage account

Create storage account

Basics | Networking | Advanced | Tags | Review + create

Network connectivity

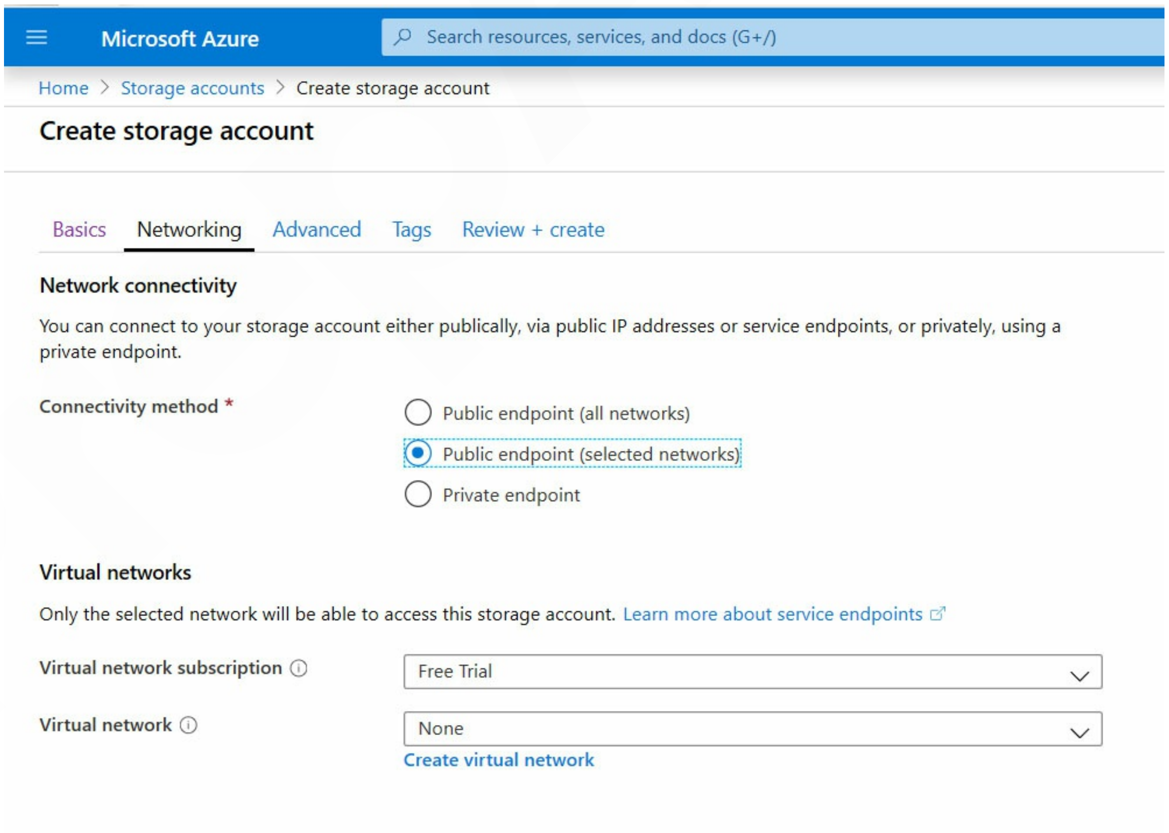
You can connect to your storage account either publically, via public IP addresses or service endpoints, or privately, using a private endpoint.

Connectivity method * [✓ Public endpoint (all networks)] Public endpoint (selected networks) Private endpoint

All networks will be able to access this storage account.
Learn more about connectivity methods

Figure 4.4

- **`Public endpoint (selected networks):`** Allows you to provide selected virtual network to access your storage.



Microsoft Azure | Search resources, services, and docs (G+/)

Home > Storage accounts > Create storage account

Create storage account

Basics | Networking | Advanced | Tags | Review + create

Network connectivity

You can connect to your storage account either publically, via public IP addresses or service endpoints, or privately, using a private endpoint.

Connectivity method * Public endpoint (all networks) [✓ Public endpoint (selected networks)] Private endpoint

Virtual networks

Only the selected network will be able to access this storage account. Learn more about service endpoints

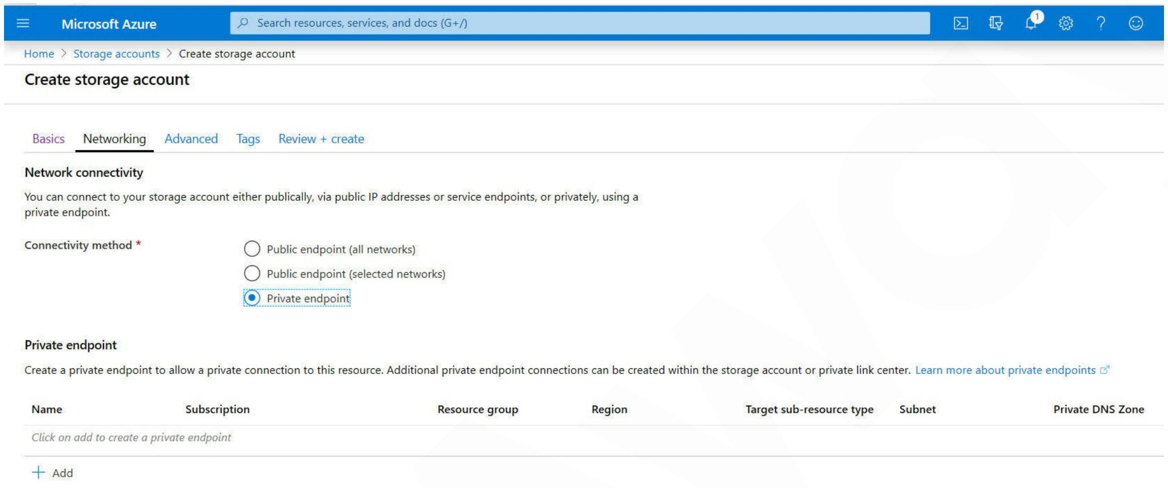
Virtual network subscription: Free Trial

Virtual network: None

Create virtual network

Figure 4.5

- **`Private endpoint:`** It enables you to create a private endpoint to allow a private connection to this resource. Additional private endpoint connections can be created within the storage account or private link center as well.



Microsoft Azure

Home > Storage accounts > Create storage account

Create storage account

Basics | Networking | Advanced | Tags | Review + create

Network connectivity

You can connect to your storage account either publically, via public IP addresses or service endpoints, or privately, using a private endpoint.

Connectivity method *
- Public endpoint (all networks)
- Public endpoint (selected networks)
- [✓ Private endpoint]

Private endpoint

Create a private endpoint to allow a private connection to this resource. Additional private endpoint connections can be created within the storage account or private link center. Learn more about private endpoints

Name	Subscription	Resource group	Region	Target sub-resource type	Subnet	Private DNS Zone
Click on add to create a private endpoint						

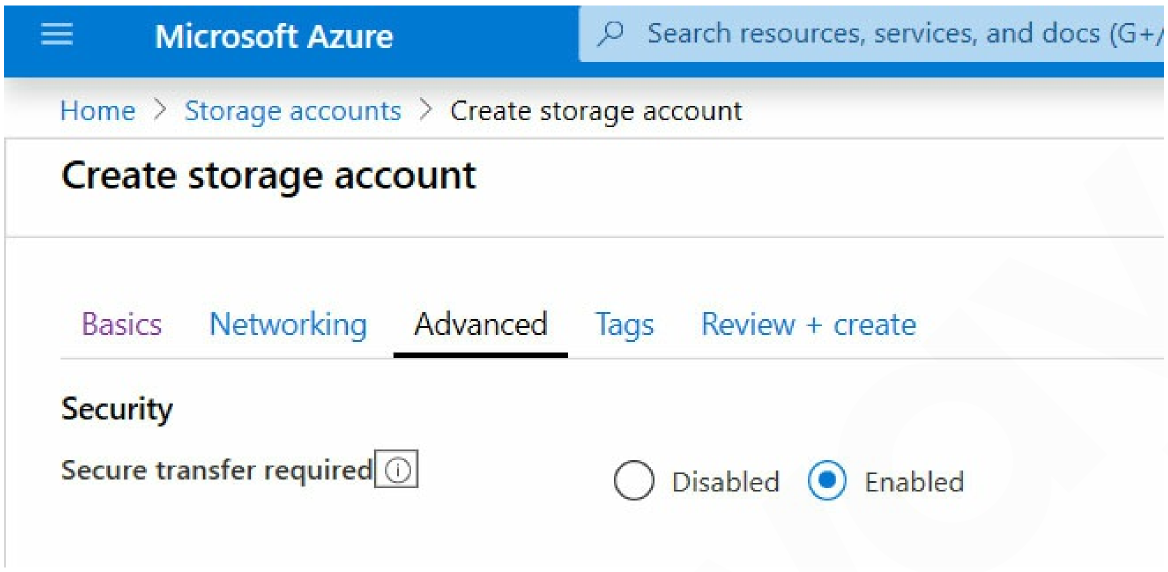
+ Add

Figure 4.6

Click on **`Advanced`** to proceed.

`Advanced` tab will present you with options to configure security and data protection aspects of your Azure Storage.

- **`Security transfer required:`** This option enhances the security of your storage account. It only allows the request to the storage account by a secure way of connection. For instance, any call from http will be rejected, and the only call from https will be allowed. By default, it comes as **`Enabled`**.


Microsoft Azure
Search resources, services, and docs (G+/
Home > Storage accounts > Create storage account
Basics
Networking
Advanced
Tags
Review + create

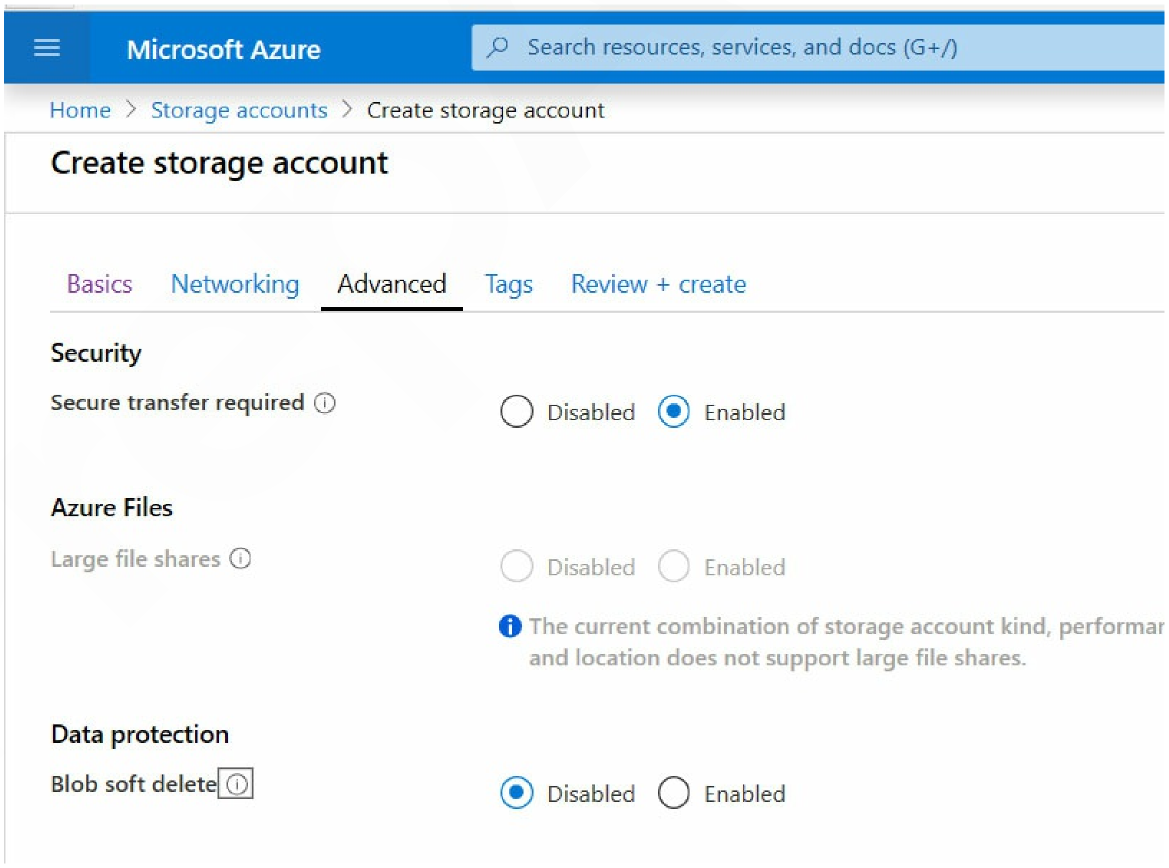
Create storage account

Security

Secure transfer required: Disabled [✓ Enabled]

Figure 4.7

- **`Blob soft delete:`** When enabled, soft delete presents you to save and recover your blob data in cases where blobs are deleted. By default, it comes as **`Disabled`**.


Microsoft Azure
Search resources, services, and docs (G+/)
Home > Storage accounts > Create storage account
Basics
Networking
Advanced
Tags
Review + create

Create storage account

Security

Secure transfer required: Disabled [✓ Enabled]

Azure Files

Large file shares: Disabled Enabled

The current combination of storage account kind, performar and location does not support large file shares.

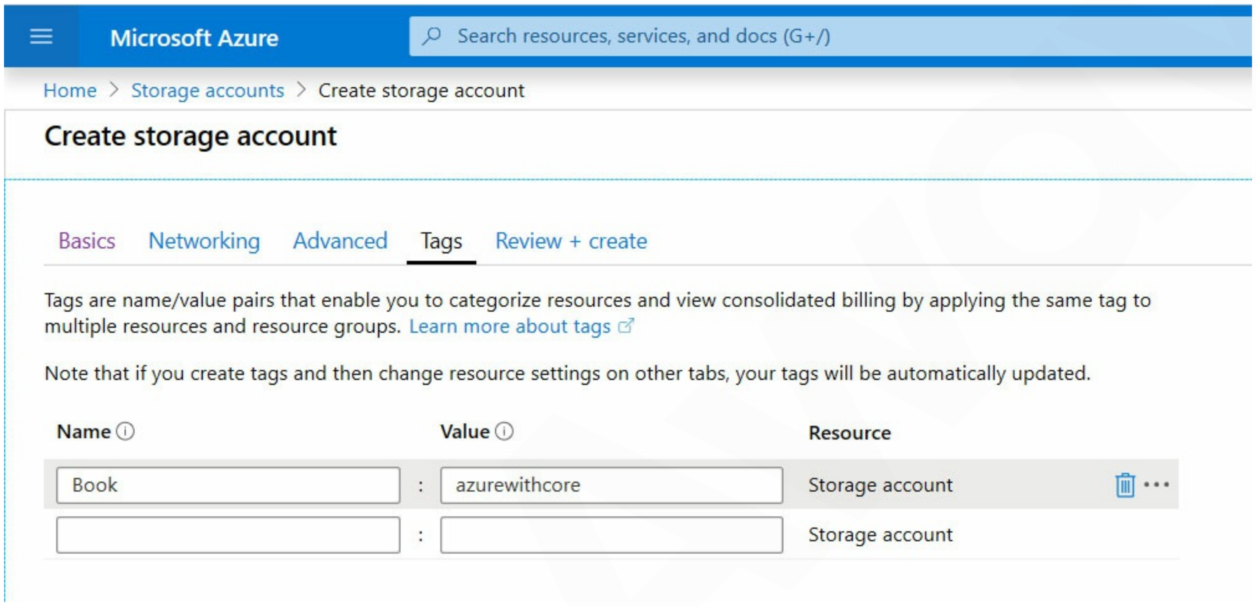
Data protection

Blob soft delete: [✓ Disabled] Enabled

Figure 4.8

Next comes the **`Tags`** tab.

Now it is always advisable to tag your resources when you create them. It helps you to have control over managing your resources. You can always query or fetch your azure resources using this tag. I will tag it, **`Name`** as Book, and **`Value`** as `azurewithcore`.


Microsoft Azure
Search resources, services, and docs (G+/)
Home > Storage accounts > Create storage account
Create storage account
Basics
Networking
Advanced
Tags
Review + create

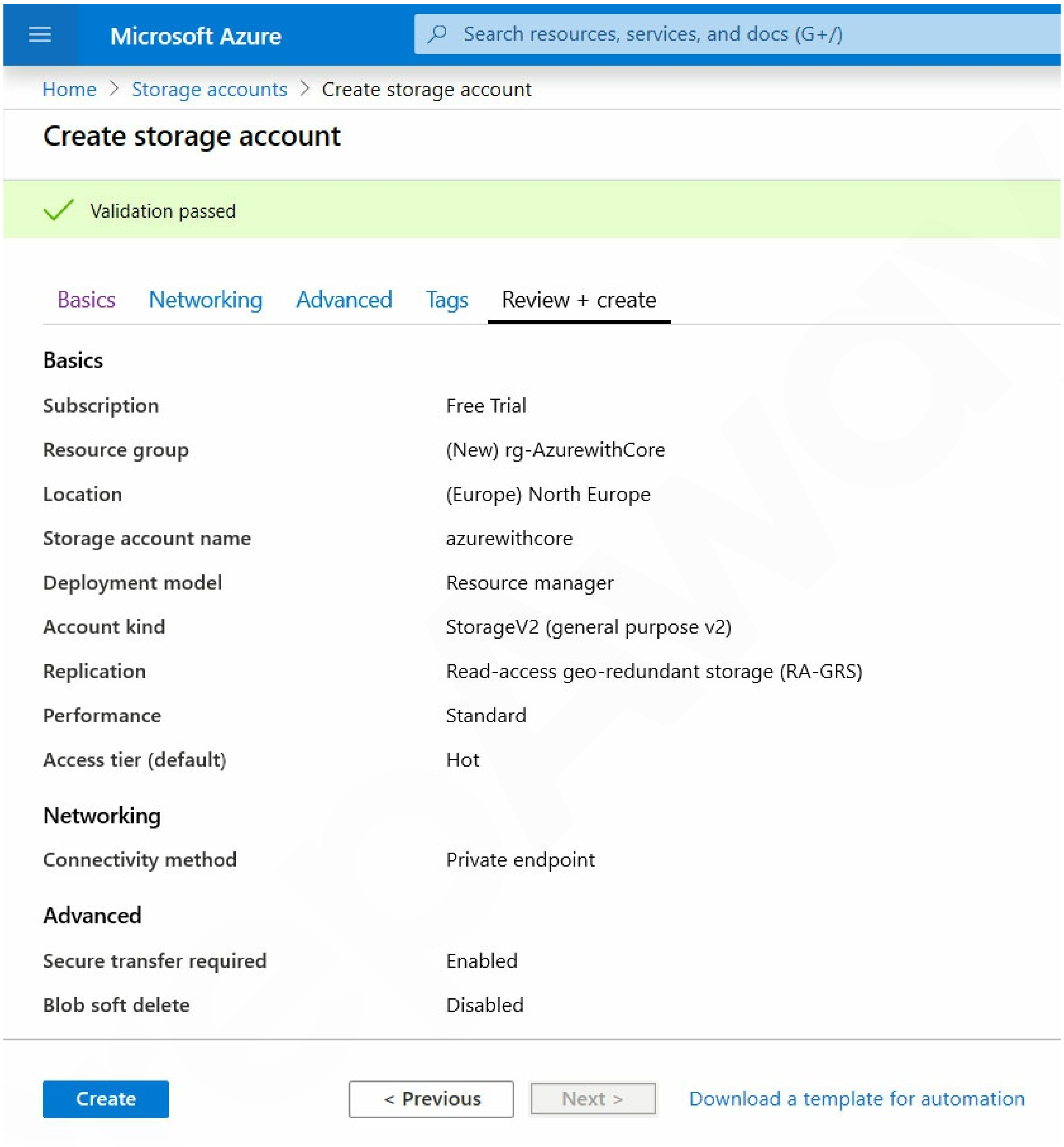
Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. Learn more about tags

Note that if you create tags and then change resource settings on other tabs, your tags will be automatically updated.

Name		Value	Resource
Book	:	azurewithcore	Storage account
	:		Storage account

Figure 4.9

Next comes to click on **`Review + Create`**. This will present you with all the options you selected, validate, and have a **`Create`** button in place to proceed with creating the storage resource.


Microsoft Azure
Search resources, services, and docs (G+/)
Home > Storage accounts > Create storage account

Create storage account

Validation passed

Basics
Networking
Advanced
Tags
Review + create

Basics

Subscription	Free Trial
Resource group	(New) rg-AzurewithCore
Location	(Europe) North Europe
Storage account name	azurewithcore
Deployment model	Resource manager
Account kind	StorageV2 (general purpose v2)
Replication	Read-access geo-redundant storage (RA-GRS)
Performance	Standard
Access tier (default)	Hot

Networking

Connectivity method	Private endpoint

Advanced

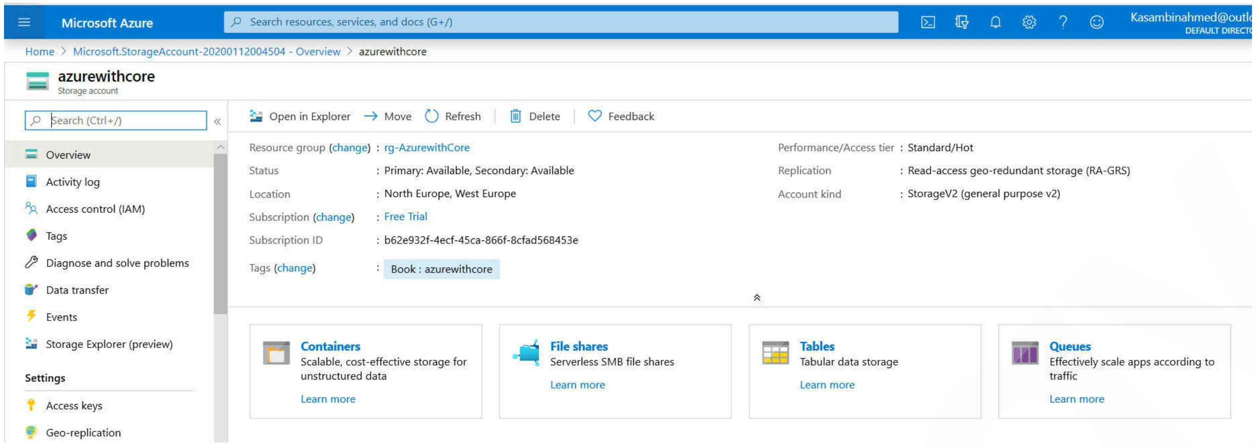
Secure transfer required	Enabled
Blob soft delete	Disabled

Figure 4.10

Click on **`Create`** to initiate the azure resource creation.

After fewer seconds, you will be notified of the status of resource deployment. Once notified, just click on the **`go to resource`** link to open the Azure Storage.

And you will be on your newly created Azure Storage **`Overview`** section.



Resource group (change) : rg-AzurewithCore
Status : Primary: Available, Secondary: Available
Location : North Europe, West Europe
Subscription (change) : Free Trial
Subscription ID : b62e932f-4ecf-45ca-866f-8cfad568453e
Tags (change) : Book : azurewithcore

Performance/Access tier : Standard/Hot
Replication : Read-access geo-redundant storage (RA-GRS)
Account kind : StorageV2 (general purpose v2)

Containers
Scalable, cost-effective storage for unstructured data
Learn more

File shares
Serverless SMB file shares
Learn more

Tables
Tabular data storage
Learn more

Queues
Effectively scale apps according to traffic
Learn more

Figure 4.11

This storage can now be used for all the data services it offers, as listed in the preceding screenshot.

Working with Azure SDKs

Let's work with Container and blobs with ASP.NET Core 3.1 for quick and simple operations. This will help you to have a gist of working with Azure Storage in .NET Core applications.

As a pre-requisite for this exercise, we need:

- Visual Studio 2019
- ASP.NET Core 3.1 SDK
- Valid Azure subscription
- Internet to run and test

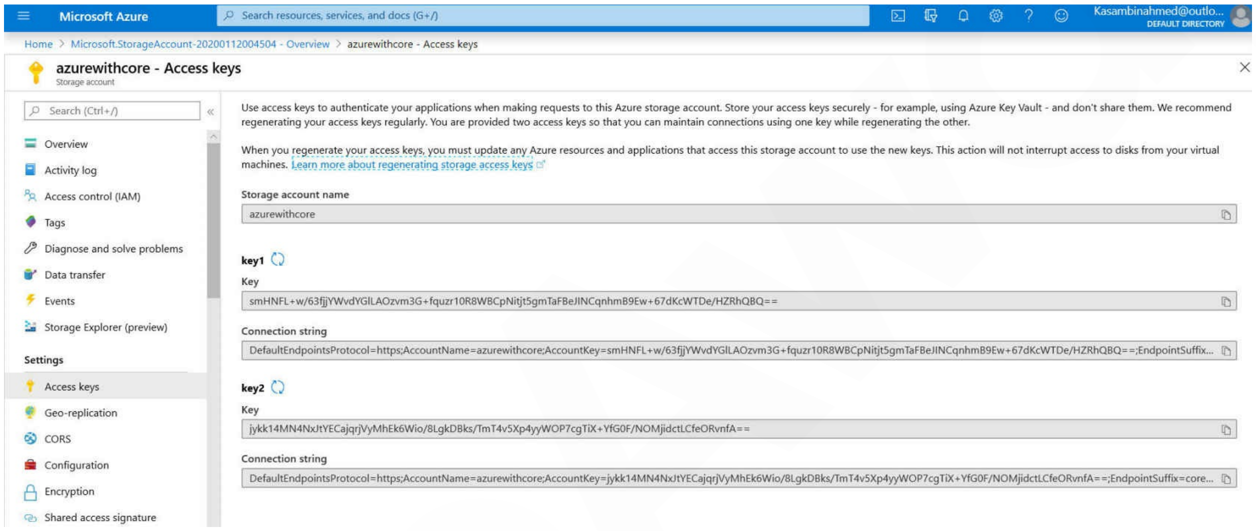
To talk about your application with Azure Storage resource, you need to have the connection string added in the application. These keys and connection strings will authenticate your application when making a request to the Azure Storage account.

Retrieve Storage connection string

Open up **`Azure Portal`** and navigate to the **`Azure Storage account resource`** page.

Under **`Settings | Access Keys.`**

Here you will find two sets of **`Keys`** and **`Connection`** String. Copy any one connection string from the page.



Use access keys to authenticate your applications when making requests to this Azure storage account. Store your access keys securely - for example, using Azure Key Vault - and don't share them. We recommend regenerating your access keys regularly. You are provided two access keys so that you can maintain connections using one key while regenerating the other.

When you regenerate your access keys, you must update any Azure resources and applications that access this storage account to use the new keys. This action will not interrupt access to disks from your virtual machines. Learn more about regenerating storage access keys

Storage account name
azurewithcore

key1

Key
smHNFL+w/63fjjYWvdYGlLAOzvm3G+fquzr10R8WBCpNitjt5gmTaFBeJINCqnhmB9Ew+67dKcWTDe/HZRhQBQ==

Connection string
DefaultEndpointsProtocol=https;AccountName=azurewithcore;AccountKey=smHNFL+w/63fjjYWvdYGlLAOzvm3G+fquzr10R8WBCpNitjt5gmTaFBeJINCqnhmB9Ew+67dKcWTDe/HZRhQBQ==;EndpointSuffix...

key2

Key
jykk14MN4NxJtYECajqrjVyMhEk6Wio/8LgkDBks/TmT4v5Xp4yyWOP7cgTiX+YfG0F/NOMjidctLCfeORvnfA==

Connection string
DefaultEndpointsProtocol=https;AccountName=azurewithcore;AccountKey=jykk14MN4NxJtYECajqrjVyMhEk6Wio/8LgkDBks/TmT4v5Xp4yyWOP7cgTiX+YfG0F/NOMjidctLCfeORvnfA==;EndpointSuffix=core...

Figure 4.12

Setting up the application settings

Very first, get this .NET Core 3.1 SDK (latest version) by following this link https://dotnet.microsoft.com/download/dotnet-core/3.0?WT.mc_id=microsoft-azuredevtips-micrum downloaded, and install in your machine.

Once installed, open up Visual Studio 2019, I am using a community edition. And click on `Create a new Project`:

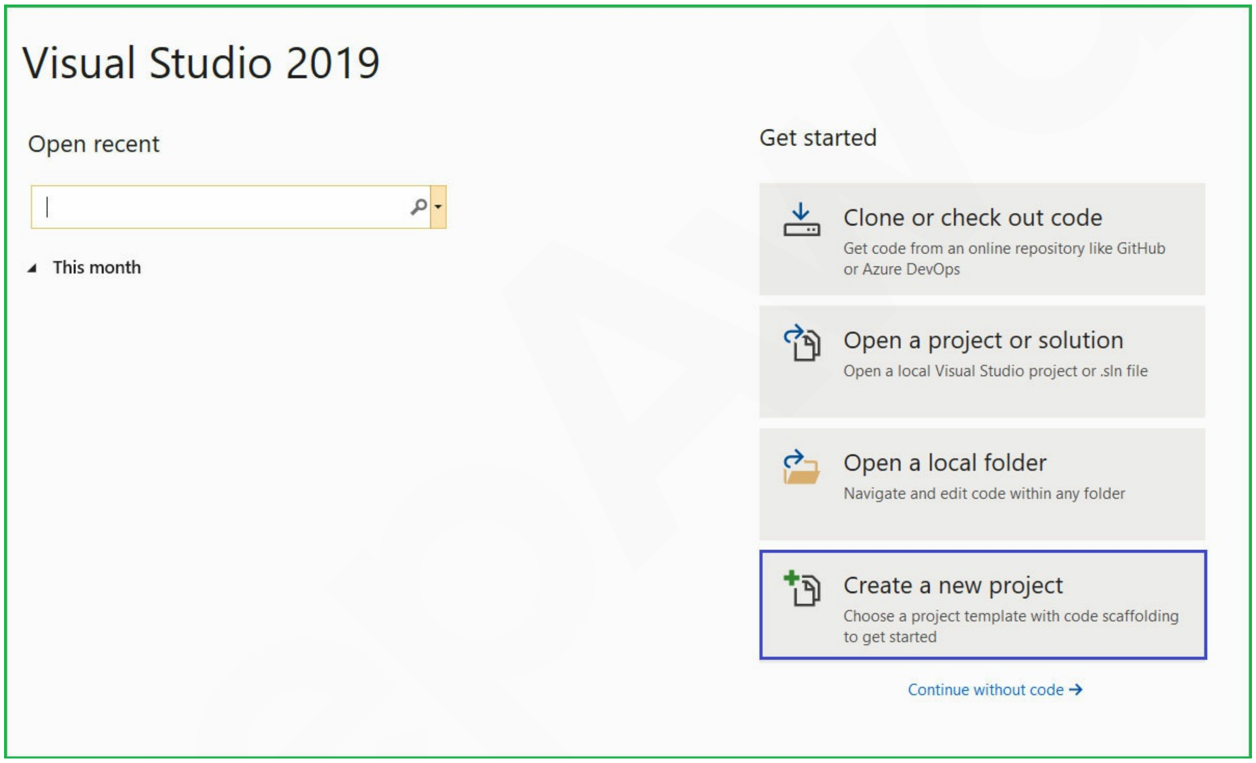
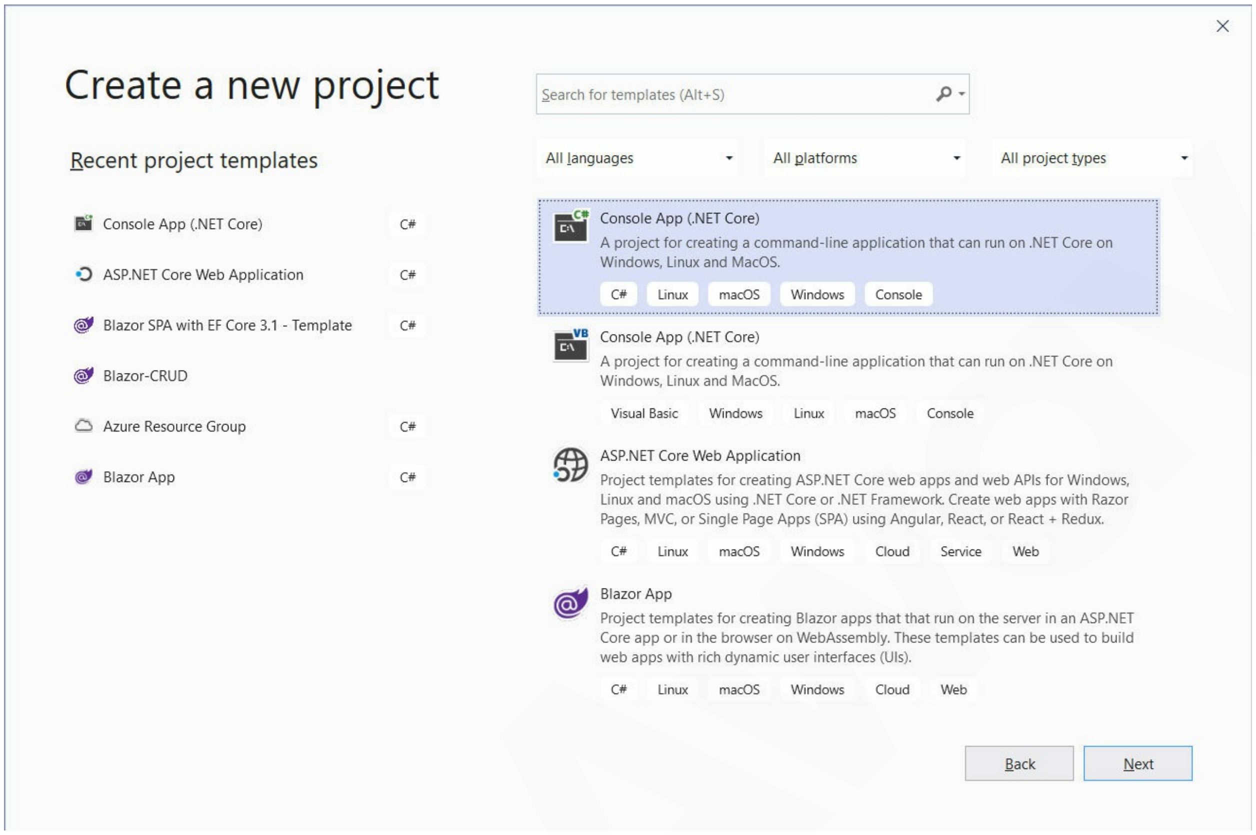

Visual Studio 2019
Open recent
This month
Get started
Clone or check out code
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

Figure 4.13

We will be electing an ASP.NET Core Console application with C#, a project for having a Command Line application that can run on .NET Core on Windows, Linux, and macOS.

And click on `Next` to proceed:



Create a new project

Recent project templates

Console App (.NET Core)	C#
ASP.NET Core Web Application	C#
Blazor SPA with EF Core 3.1 - Template	C#
Blazor-CRUD	
Azure Resource Group	C#
Blazor App	C#

Search for templates (Alt+S)

All languages | 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

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
Visual Basic Windows Linux macOS 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

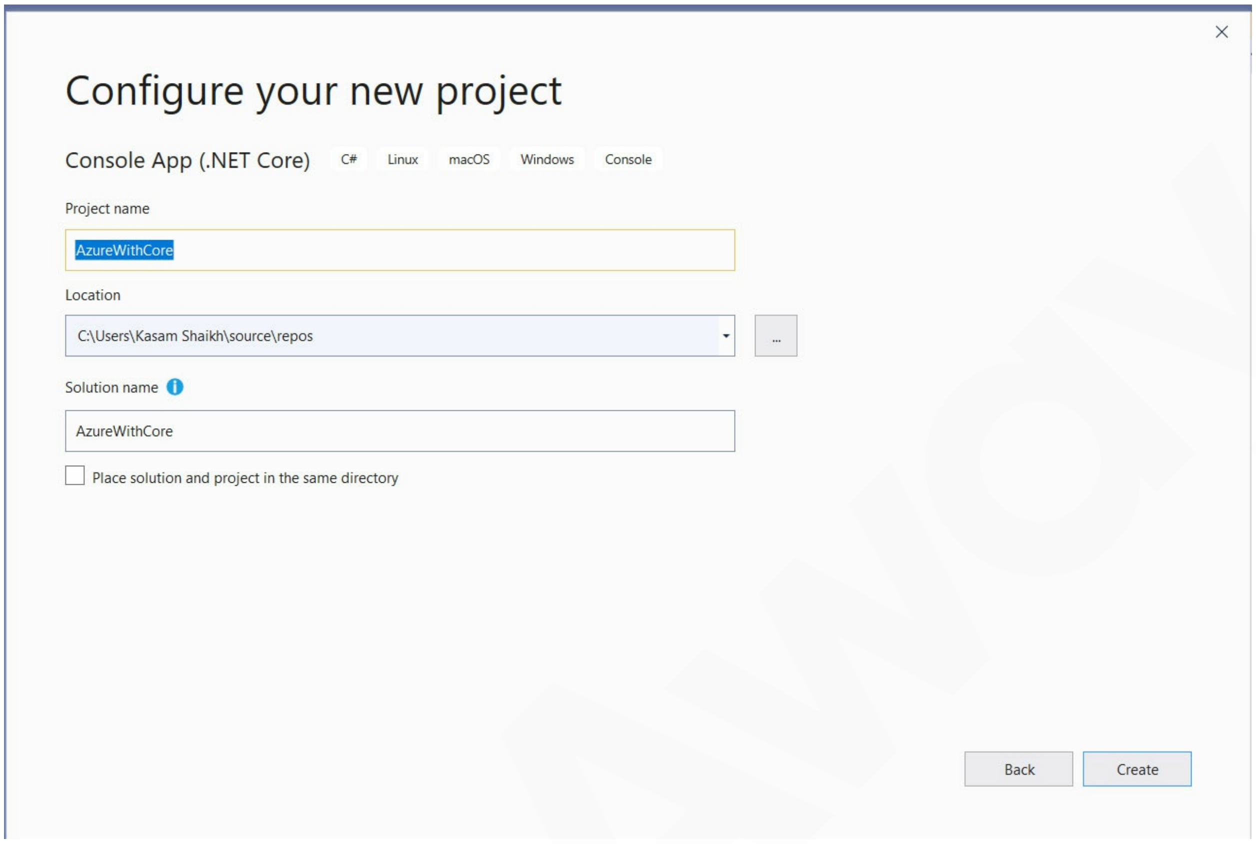
Blazor App
Project templates for creating Blazor apps that that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

Back | Next

Figure 4.14

Now it is the time to configure the application, with **`Project name`**, **`Location`**, and **`Solution name`**. For this chapter, I am giving it as `AzurewithCore`.

Click on **`Create`** to proceed with having the application:


Configure your new project

Console App (.NET Core)

Project name: AzureWithCore

Location: C:\Users\Kasam Shaikh\source\repos

Solution name: AzureWithCore

☐ Place solution and project in the same directory

Figure 4.15

Now, the very first thing will be to add the Azure Storage package using Nuget to the application for performing Storage operations,

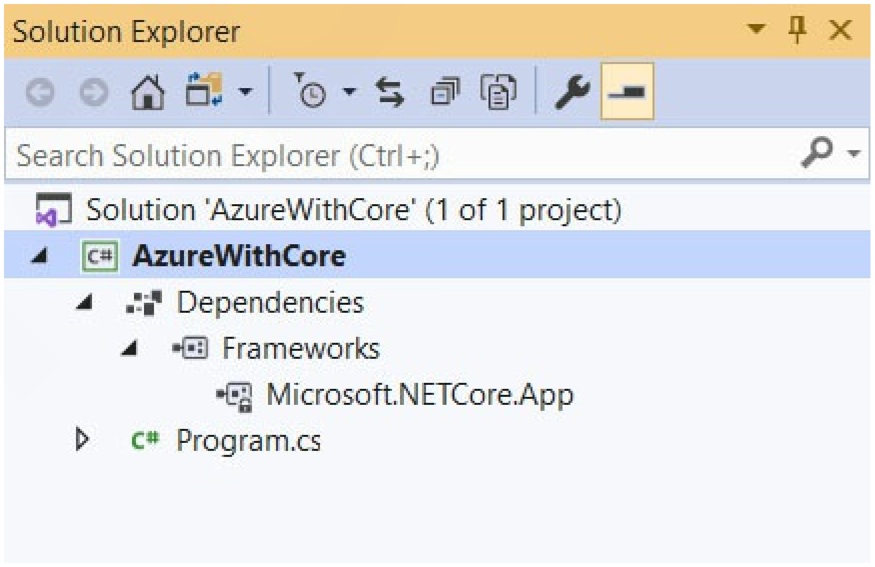


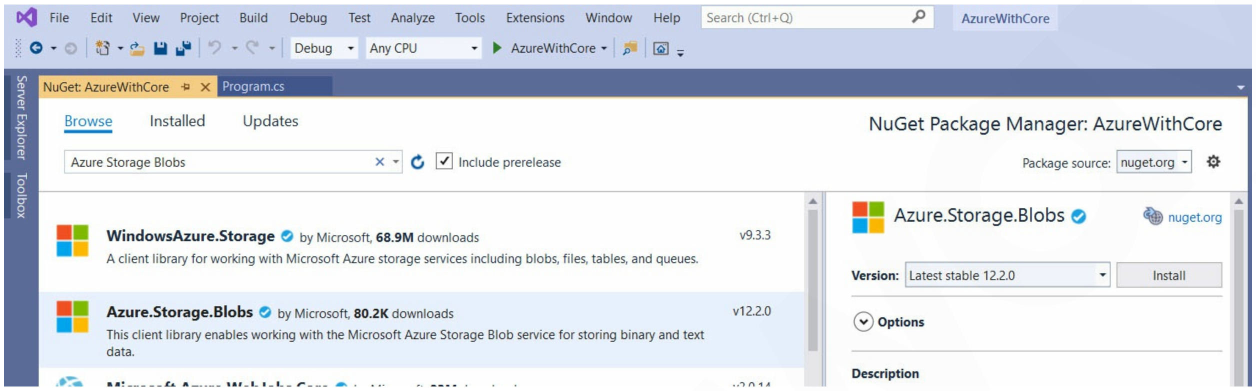
Figure 4.16

Right-click on the project in **`Solution Explorer`**, and select **`Manage Nuget`**

`Packages.`

Navigate to the **`Browse`** section, and enter `Azure Storage Blobs` in the text box and hit **Enter**.

Select the one listed as shown in the following screenshot, and click on **`Install`** from right-hand pane:



Azure Storage Blobs

WindowsAzure.Storage by Microsoft, **68.9M** downloads v9.3.3
A client library for working with Microsoft Azure storage services including blobs, files, tables, and queues.

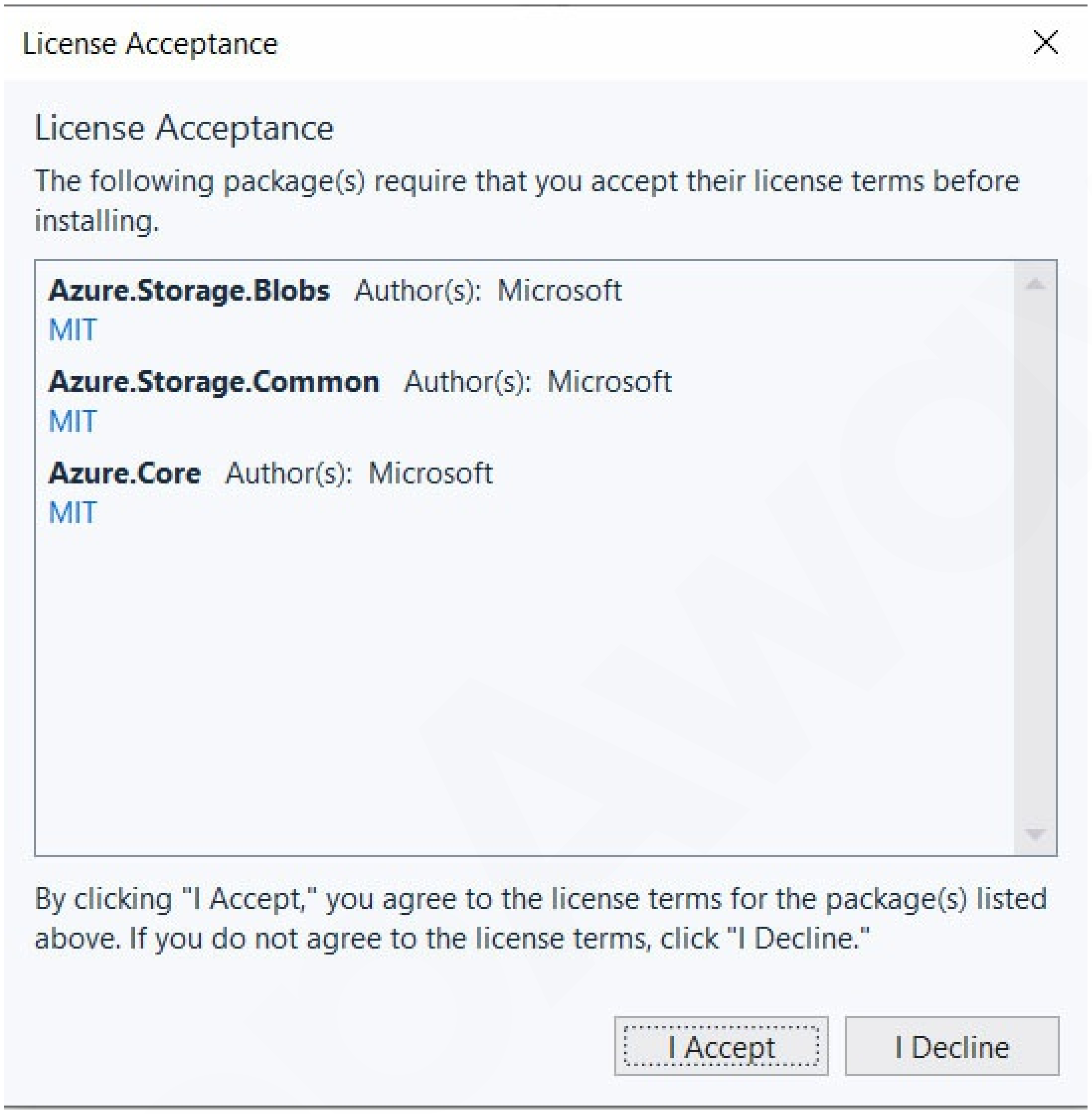
Azure.Storage.Blobs by Microsoft, **80.2K** downloads v12.2.0
This client library enables working with the Microsoft Azure Storage Blob service for storing binary and text data.

Azure.Storage.Blobs nuget.org

Version: Latest stable 12.2.0 Install

Figure 4.17

Read and accept the terms for the packages to complete the installation.


License Acceptance

License Acceptance

The following package(s) require that you accept their license terms before installing.

Azure.Storage.Blobs Author(s): Microsoft
MIT

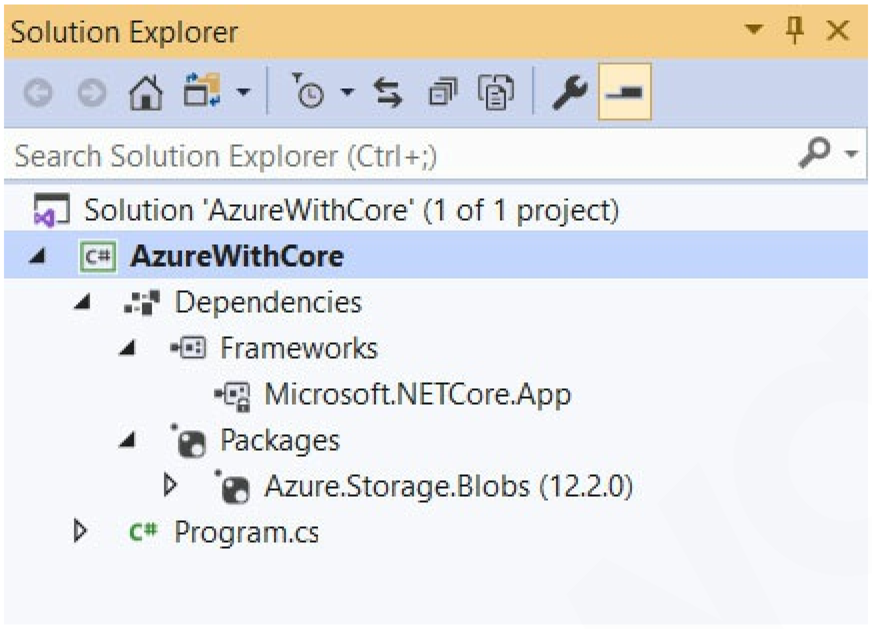
Azure.Storage.Common Author(s): Microsoft
MIT

Azure.Core Author(s): Microsoft
MIT

By clicking "I Accept," you agree to the license terms for the package(s) listed above. If you do not agree to the license terms, click "I Decline."

Figure 4.18

After successful installation, you will have the package listed in **`Solution Explorer`** as Packages. The notification of its completion can also be seen in the **`Visual Studio Output`** window.



Solution 'AzureWithCore' (1 of 1 project)
AzureWithCore
Dependencies
Frameworks
Microsoft.NETCore.App
Packages
Azure.Storage.Blobs (12.2.0)
Program.cs

Figure 4.19

Here I am done with setting up the things.

Let’s start with the coding part.

Managing Azure Blob from application

Open up Program.cs page and have the code written over here.

Very first is to add the `using` reference:

```
using System;
using System.IO;
using System.Threading.Tasks;
```

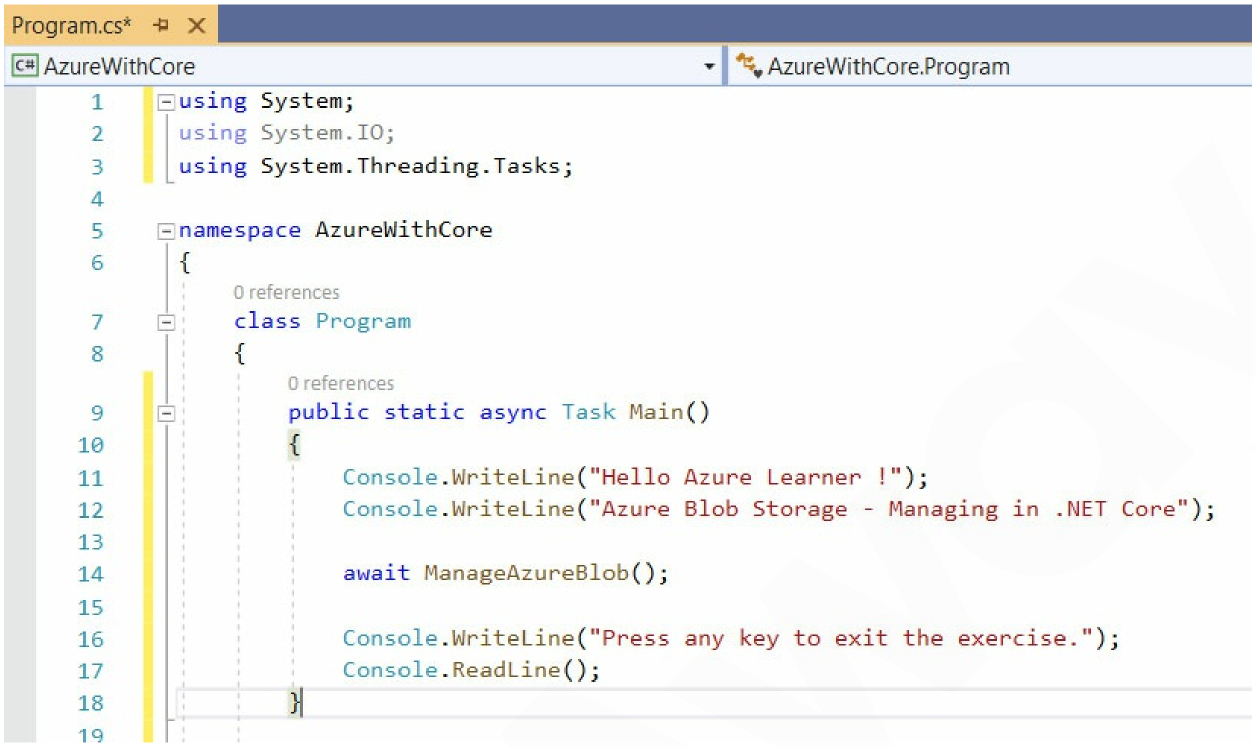
You need to perform `async` operations and need to set the method for the same.

Create a `private static async` method and call it from `Main Class` to initiate. I am creating a method name `ManageAzureBlob`.

Adding a few messages to have the application presentable and self-explanatory during the execution:

```
public static async Task Main()
   {
    Console.WriteLine(“Hello Azure Learner !”);
    Console.WriteLine(“Azure Blob Storage - Managing in .NET
    Core”);

    await ManageAzureBlob();

    Console.WriteLine(“Press any key to exit the exercise.”);
    Console.ReadLine();
   }
```

Now here I assume you are aware of basic C# coding language. Hence, I will not be covering in detail on why I used the await keyword, and so on. I will focus more on code with respect to the Azure Blob storage package.



Program.cs* | AzureWithCore | AzureWithCore.Program

```
using System;
using System.IO;
using System.Threading.Tasks;

namespace AzureWithCore
{
    0 references
    class Program
    {
        0 references
        public static async Task Main()
        {
            Console.WriteLine("Hello Azure Learner !");
            Console.WriteLine("Azure Blob Storage - Managing in .NET Core");

            await ManageAzureBlob();

            Console.WriteLine("Press any key to exit the exercise.");
            Console.ReadLine();
        }
```

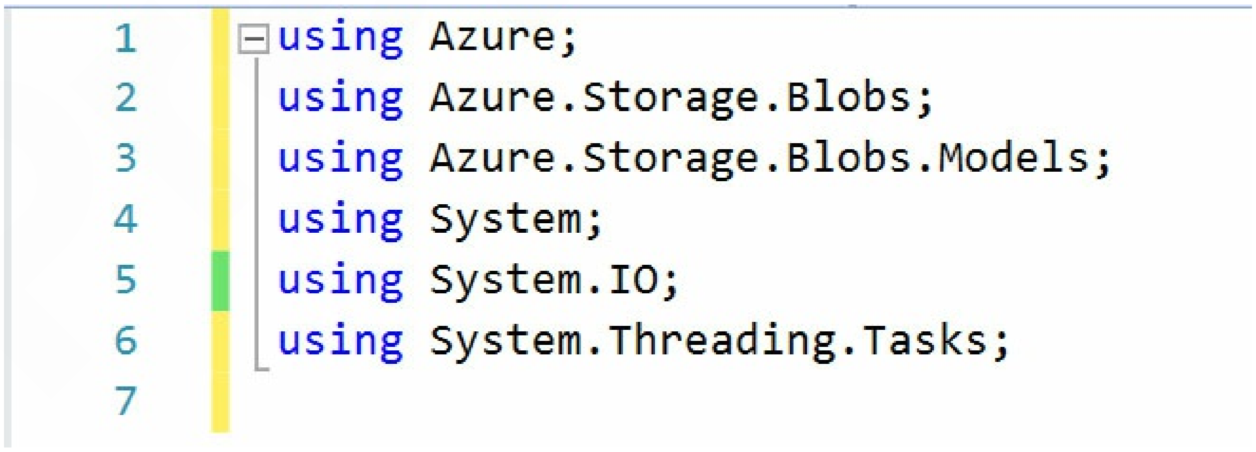
Figure 4.20

Now let’s work on the `ManageAzureBlob` method, as seen in the preceding screenshot.

To start with, add a few more using statements to deal with `Azure` operations.

```
using Azure;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
```

So now your code with have following using statements:



```
using Azure;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
using System;
using System.IO;
using System.Threading.Tasks;
```

Figure 4.21

I will explain later in code exercise on why I added these three Using statements.

Create a Container

You have to first create the **`Blob Client`** object to start with. Blob Client takes two arguments, Azure Storage account connection string and Container name, to create the container.

Under the method,`ManageAzureBlob`, create a `blobContainerClient`.

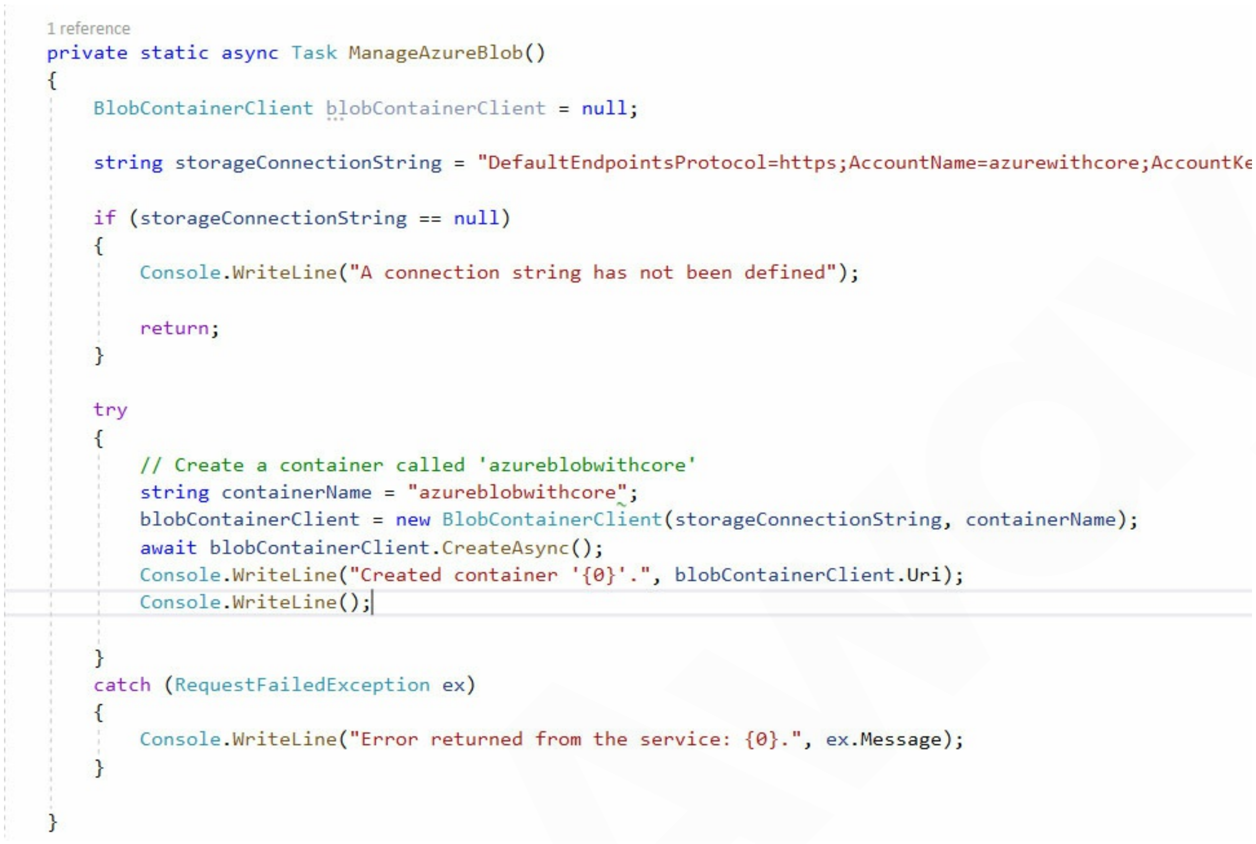
It's a part of `Azure.Blobs.Storage` class and allows you to manipulate **`Azure Storage Containers`** and their **`Blobs`**.

Another important step is to have the Azure Storage account connection string added in the application. This you can use as **`Environment Variable`**, but for making this exercise simple, have added the connection string in the method as a string, naming as `storageConnectString`. This connection string is the same we retrieved earlier from the Access Key section from the Azure portal.

A simple null check is for the connection string.

Following is the code for creating the container. It sets a `Container` name, and pass it as an argument to `blobContainerClient` along with storage account connection string. A later line of code creates the container at to Storage account.

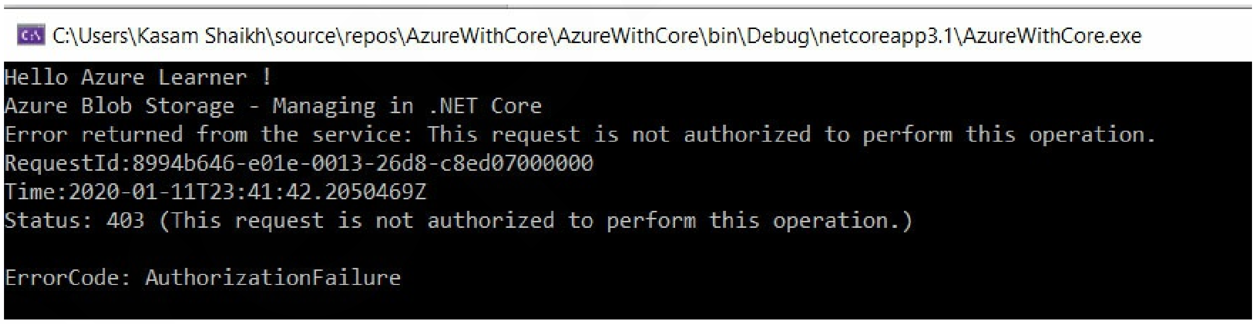
And after the container creation prints the primary endpoint for `blobContainer`.



```
1 reference
private static async Task ManageAzureBlob()
{
    BlobContainerClient blobContainerClient = null;

    string storageConnectionString = "DefaultEndpointsProtocol=https;AccountName=azurewithcore;AccountKe

    if (storageConnectionString == null)
    {
        Console.WriteLine("A connection string has not been defined");

        return;
    }

    try
    {
        // Create a container called 'azureblobwithcore'
        string containerName = "azureblobwithcore";
        blobContainerClient = new BlobContainerClient(storageConnectionString, containerName);
        await blobContainerClient.CreateAsync();
        Console.WriteLine("Created container '{0}'.", blobContainerClient.Uri);
        Console.WriteLine();

    }
    catch (RequestFailedException ex)
    {
        Console.WriteLine("Error returned from the service: {0}.", ex.Message);
    }

}
```

Figure 4.22

Let’s run it, and it throws an **`AuthorizationFailure`** error!



```
C:\Users\Kasam Shaikh\source\repos\AzureWithCore\AzureWithCore\bin\Debug\netcoreapp3.1\AzureWithCore.exe
Hello Azure Learner !
Azure Blob Storage - Managing in .NET Core
Error returned from the service: This request is not authorized to perform this operation.
RequestId:8994b646-e01e-0013-26d8-c8ed07000000
Time:2020-01-11T23:41:42.2050469Z
Status: 403 (This request is not authorized to perform this operation.)

ErrorCode: AuthorizationFailure
```

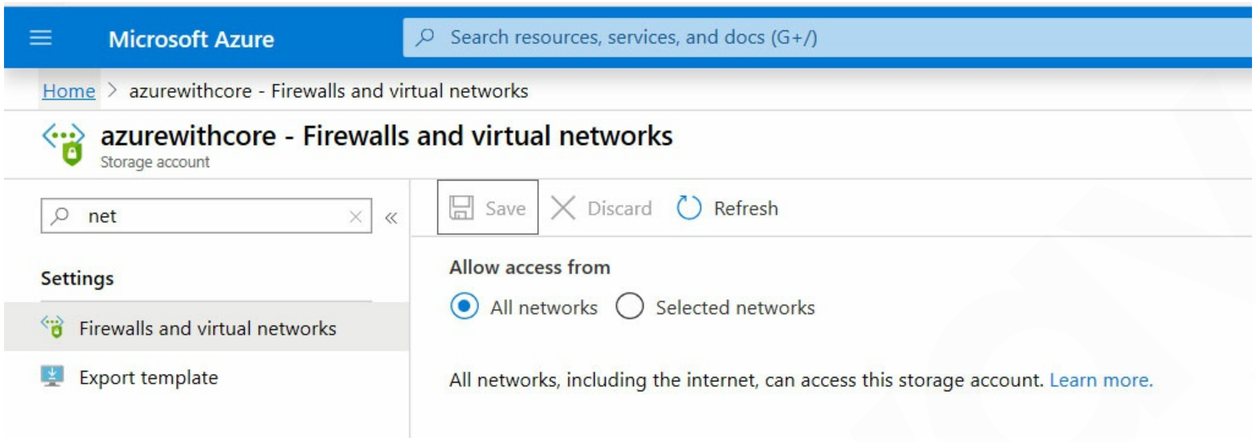
Figure 4.23

Any guess why?

If you notice the image of the storage account validation screen, you will see the connectivity configured is as a private endpoint, and hence this call is not being authorized.

Now to resolve this authorization error, navigate to **`Azure Portal | Azure Storage Account | under settings|Click on Firewalls and Virtual Network.`**

Click on the radio button with the option to allow **`All networks`**, and **`Save`** it.


Microsoft Azure
Search resources, services, and docs (G+/)
Home > azurewithcore - Firewalls and virtual networks
Storage account
net
Settings
Firewalls and virtual networks
Export template
Save
Discard
Refresh

azurewithcore - Firewalls and virtual networks

Allow access from

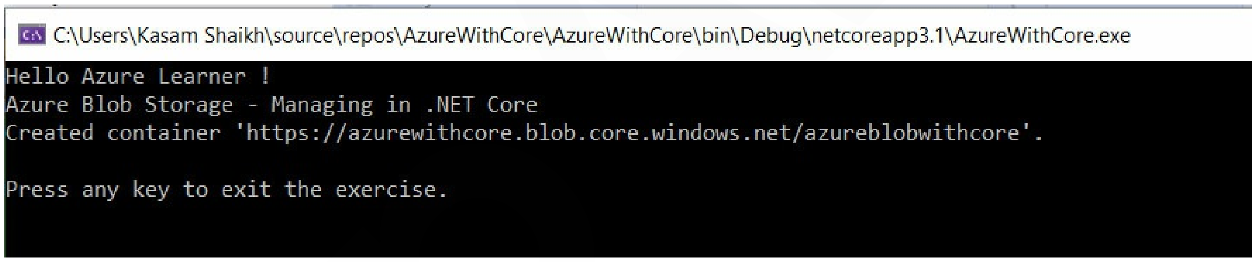
(● All networks) ○ Selected networks

All networks, including the internet, can access this storage account. Learn more.

Figure 4.24

After performing this activity, run the application again.

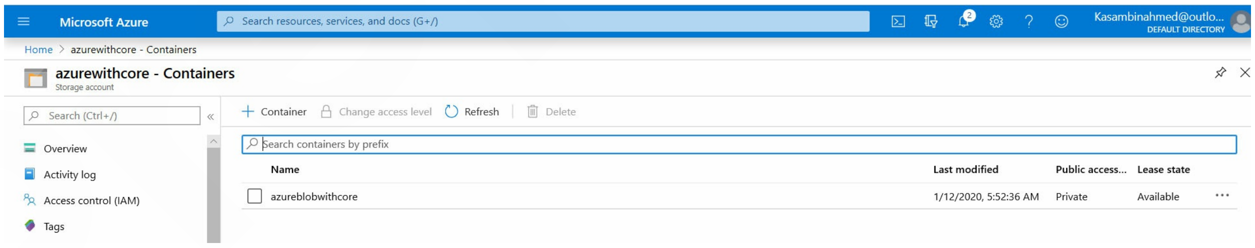
You can see now that it allows the creation and provided the newly created Blob URI.



```
C:\Users\Kasam Shaikh\source\repos\AzureWithCore\AzureWithCore\bin\Debug\netcoreapp3.1\AzureWithCore.exe
Hello Azure Learner !
Azure Blob Storage - Managing in .NET Core
Created container 'https://azurewithcore.blob.core.windows.net/azureblobwithcore'.

Press any key to exit the exercise.
```

Figure 4.25

In the Azure portal, the newly created container can be seen:



Name	Last modified	Public access...	Lease state
azureblobwithcore	1/12/2020, 5:52:36 AM	Private	Available

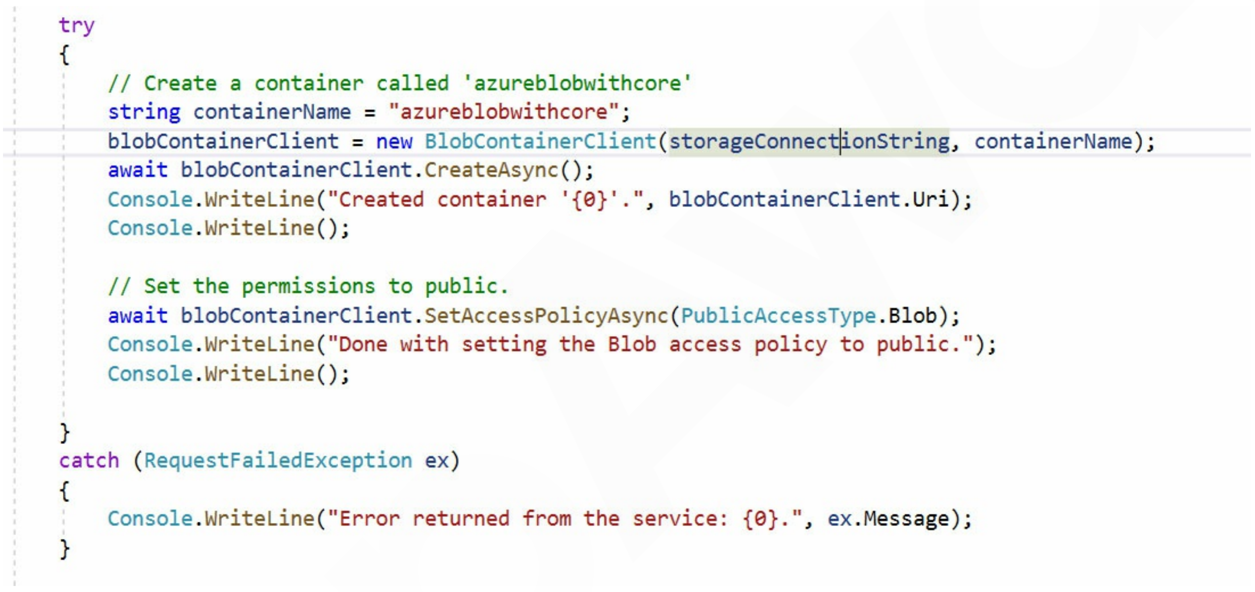
Figure 4.26

Setting access policy

If you notice the preceding screenshot, you can see the access level to the newly created container is listed as **`Private`**.

`blobContainerClient` and `SetAccessPolicyAsync` does the required settings, passing `publicAccessType` enum for `Blob`. This is part of `Azure.Storage.Blob.Model`.

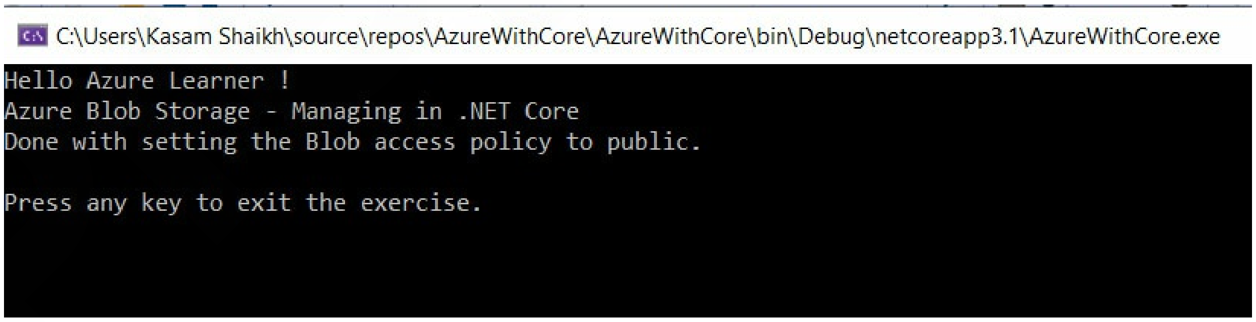
One line of code performs the required operation.



```
try
{
    // Create a container called 'azureblobwithcore'
    string containerName = "azureblobwithcore";
    blobContainerClient = new BlobContainerClient(storageConnectionString, containerName);
    await blobContainerClient.CreateAsync();
    Console.WriteLine("Created container '{0}'.", blobContainerClient.Uri);
    Console.WriteLine();

    // Set the permissions to public.
    await blobContainerClient.SetAccessPolicyAsync(PublicAccessType.Blob);
    Console.WriteLine("Done with setting the Blob access policy to public.");
    Console.WriteLine();

}
catch (RequestFailedException ex)
{
    Console.WriteLine("Error returned from the service: {0}.", ex.Message);
}
```

Figure 4.27

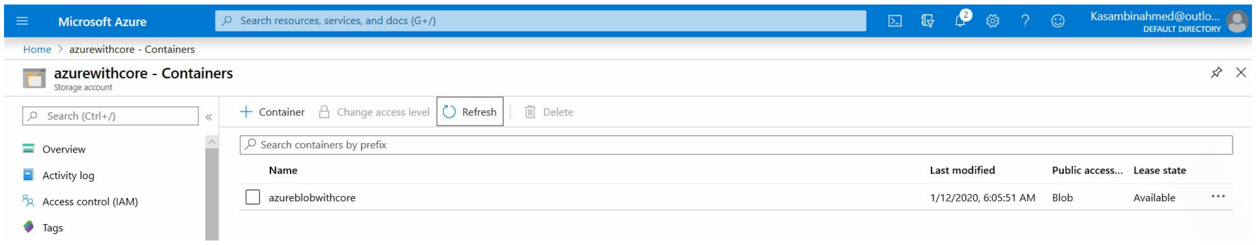
Run the code:



```
C:\Users\Kasam Shaikh\source\repos\AzureWithCore\AzureWithCore\bin\Debug\netcoreapp3.1\AzureWithCore.exe
Hello Azure Learner !
Azure Blob Storage - Managing in .NET Core
Done with setting the Blob access policy to public.

Press any key to exit the exercise.
```

Figure 4.28

The change reflected in the Azure portal:



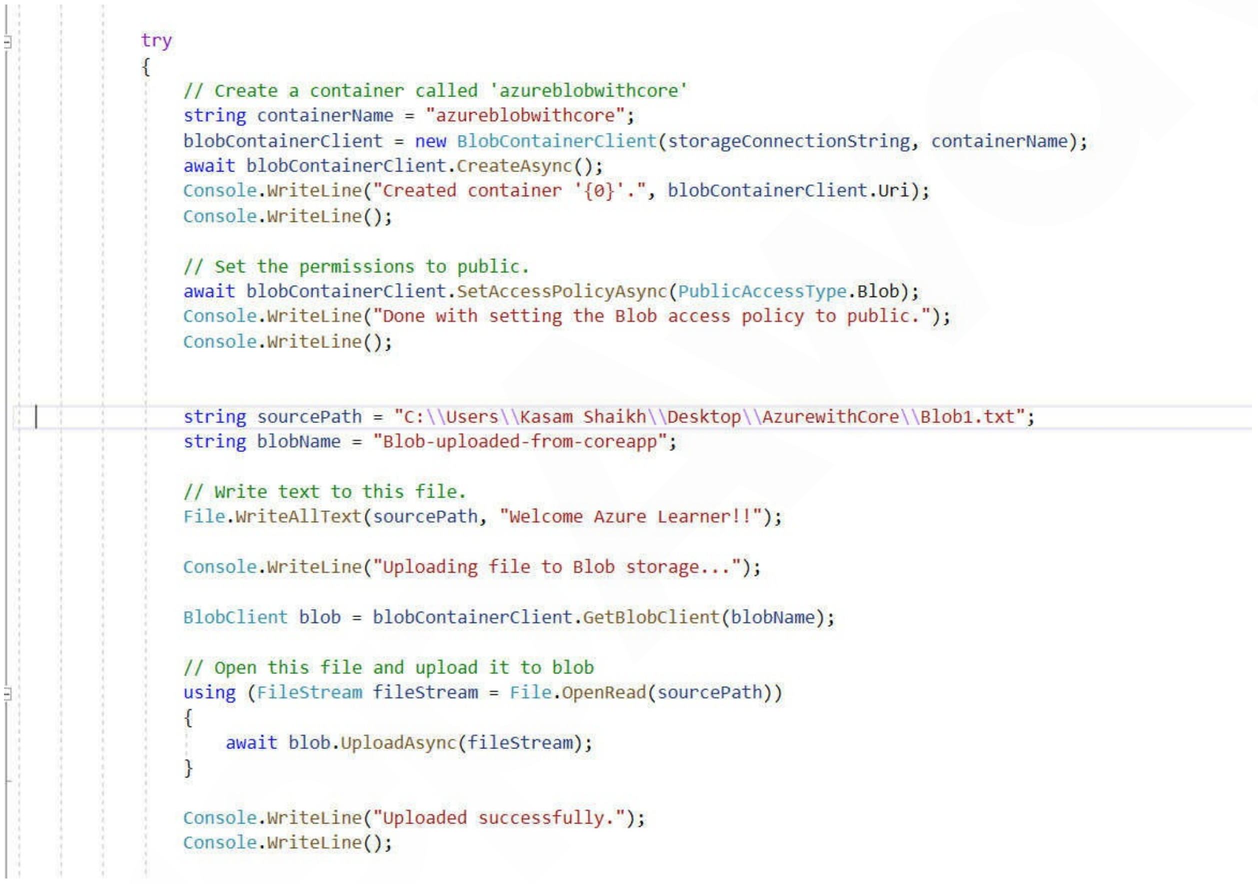
Name	Last modified	Public access...	Lease state
azureblobwithcore	1/12/2020, 6:05:51 AM	Blob	Available

Figure 4.29

Uploading file as Blob

We created the Storage account, then `Container`, and later changed it access policy as well. Now I will show you how to upload the file as `Blob`.

Have a look at the following screenshot. Now the method will look similar to the following screenshot:



```
try
{
    // Create a container called 'azureblobwithcore'
    string containerName = "azureblobwithcore";
    blobContainerClient = new BlobContainerClient(storageConnectionString, containerName);
    await blobContainerClient.CreateAsync();
    Console.WriteLine("Created container '{0}'.", blobContainerClient.Uri);
    Console.WriteLine();

    // Set the permissions to public.
    await blobContainerClient.SetAccessPolicyAsync(PublicAccessType.Blob);
    Console.WriteLine("Done with setting the Blob access policy to public.");
    Console.WriteLine();


    string sourcePath = "C:\\Users\\Kasam Shaikh\\Desktop\\AzurewithCore\\Blob1.txt";
    string blobName = "Blob-uploaded-from-coreapp";

    // Write text to this file.
    File.WriteAllText(sourcePath, "Welcome Azure Learner!!");

    Console.WriteLine("Uploading file to Blob storage...");

    BlobClient blob = blobContainerClient.GetBlobClient(blobName);

    // Open this file and upload it to blob
    using (FileStream fileStream = File.OpenRead(sourcePath))
    {
        await blob.UploadAsync(fileStream);
    }

    Console.WriteLine("Uploaded successfully.");
    Console.WriteLine();
```

Figure 4.30

Here, `sourcePath` is the path of the file, and I will be uploading as `Blob` to the storage account. But before uploading, we will write a text in the file.

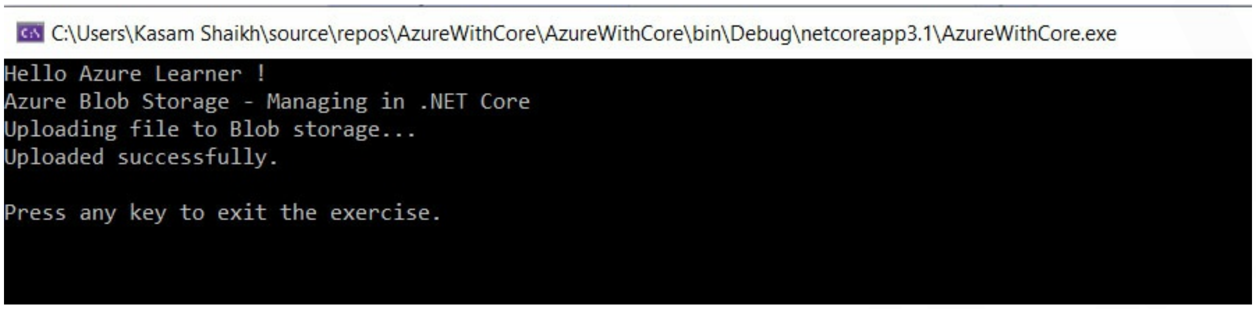
`blobName` is the name of the blob, and I will be uploading.

Writing the file performs a simple IO operation.

You need to first create a `blobClient` ad using `blobContainerClient` object's `GetBlobClient` method pass the name you wish to have your blob to get upload with.

Again, using file IO operation to read the file, method `UploadAsync` of `blobClient` object is used to upload the file as a blob to a specified Azure Storage account.

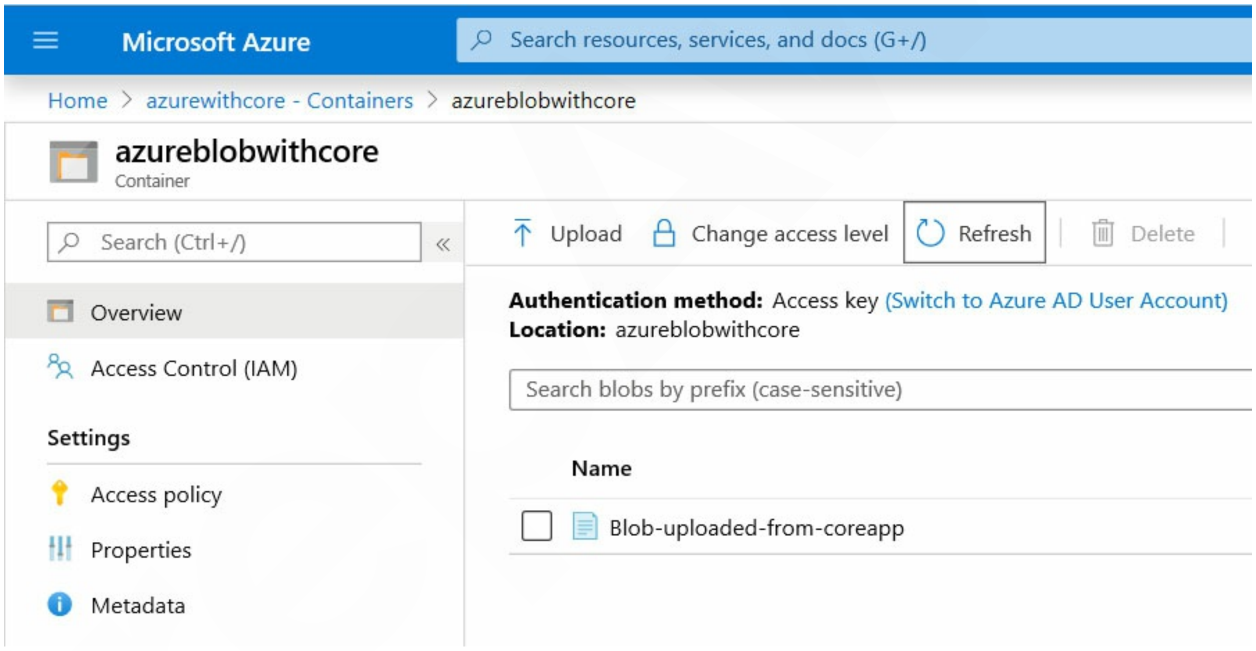
Now run the code, and the output will be a success as seen as follow:



```
C:\Users\Kasam Shaikh\source\repos\AzureWithCore\AzureWithCore\bin\Debug\netcoreapp3.1\AzureWithCore.exe
Hello Azure Learner !
Azure Blob Storage - Managing in .NET Core
Uploading file to Blob storage...
Uploaded successfully.

Press any key to exit the exercise.
```

Figure 4.31

Navigate to the portal, and here you can see the uploaded file:



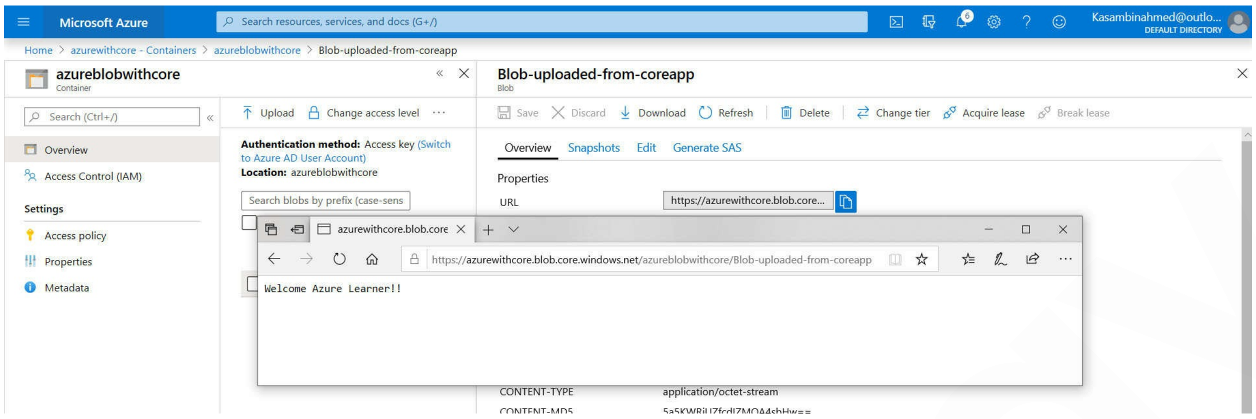
Authentication method: Access key (Switch to Azure AD User Account)
Location: azureblobwithcore

Search blobs by prefix (case-sensitive)

Name
Blob-uploaded-from-coreapp

Figure 4.32

Click on **`Blob`**, copy the public URL and open in browser, you can see the text we added via code:



Authentication method: Access key (Switch to Azure AD User Account)
Location: azureblobwithcore

URL	https://azurewithcore.blob.core...
CONTENT-TYPE	application/octet-stream
CONTENT-MD5	5a5KWRjUZfcdIZMQA4sbHw==

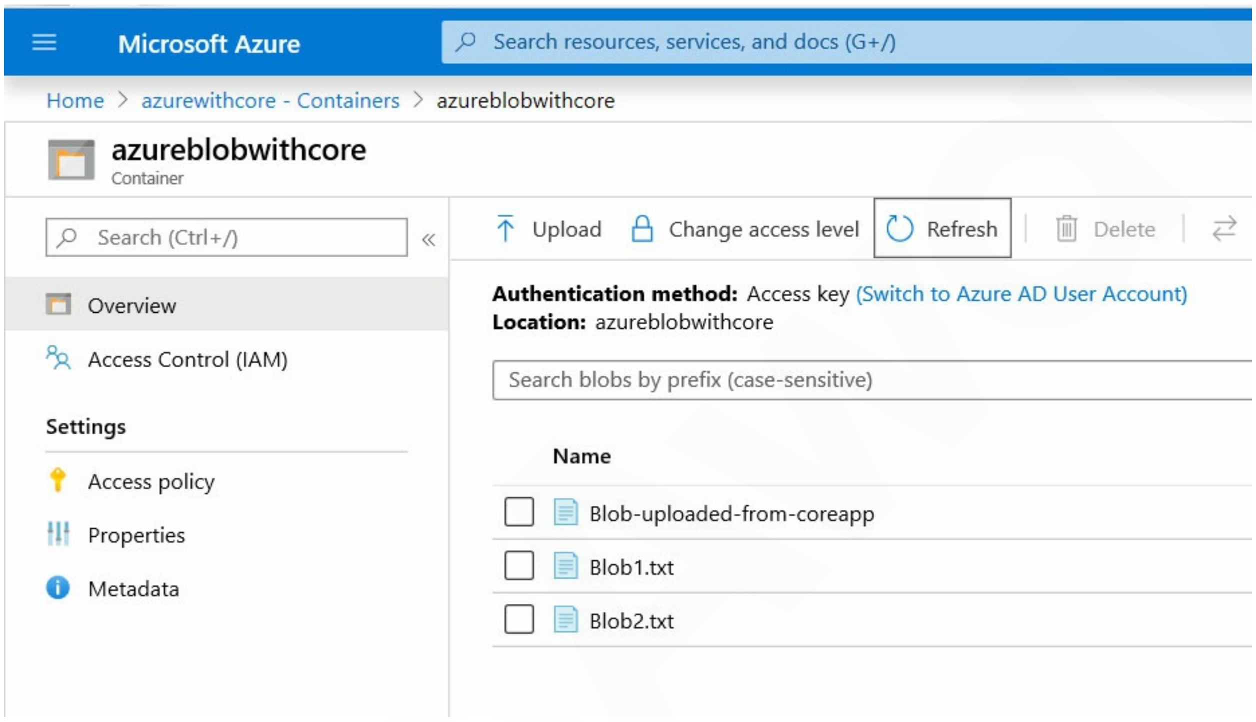
https://azurewithcore.blob.core.windows.net/azureblobwithcore/Blob-uploaded-from-coreapp

```
Welcome Azure Learner!!
```

Figure 4.33

Listing the Blobs in Container

For the exercise, I have added a couple of blobs to the **`Container`**, as seen in the following screenshot:



Authentication method: Access key (Switch to Azure AD User Account)
Location: azureblobwithcore

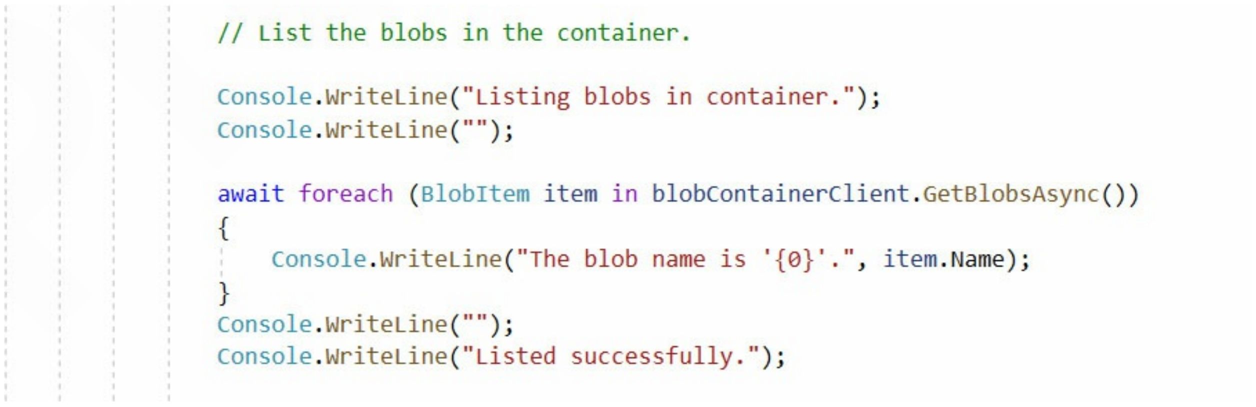
Search blobs by prefix (case-sensitive)

Name
Blob-uploaded-from-coreapp
Blob1.txt
Blob2.txt

Figure 4.34

You can list out the blobs in the given container in your application via code.

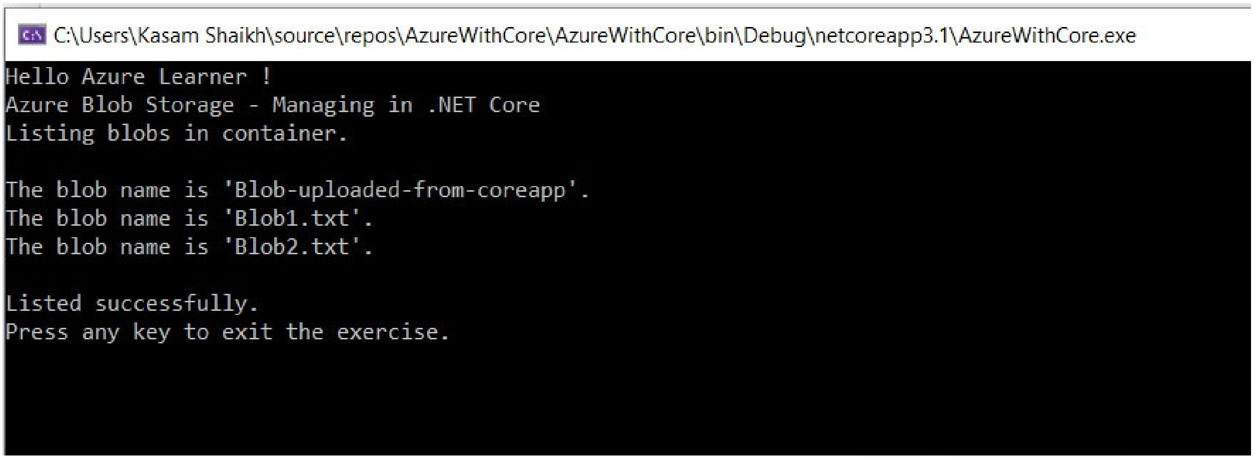
By looping each item as `blobItem` form `blobContainerClient's` `GetBlobAsync` method, you can list all the blobs residing in the given container:



```
// List the blobs in the container.

Console.WriteLine("Listing blobs in container.");
Console.WriteLine("");

await foreach (BlobItem item in blobContainerClient.GetBlobsAsync())
{
    Console.WriteLine("The blob name is '{0}'.", item.Name);
}
Console.WriteLine("");
Console.WriteLine("Listed successfully.");
```

Figure 4.35

And when you run the code:



```
C:\Users\Kasam Shaikh\source\repos\AzureWithCore\AzureWithCore\bin\Debug\netcoreapp3.1\AzureWithCore.exe
Hello Azure Learner !
Azure Blob Storage - Managing in .NET Core
Listing blobs in container.

The blob name is 'Blob-uploaded-from-coreapp'.
The blob name is 'Blob1.txt'.
The blob name is 'Blob2.txt'.

Listed successfully.
Press any key to exit the exercise.
```

Figure 4.36

We can see the list of blobs, as can be seen in the preceding screenshot.

Conclusion

In this chapter, you learned about one the most used, important, and a must known Azure service – Azure Storage. You also went through each option presented by Azure as a part of Azure Storage configuration, including security aspects. Further, you learned how to make the .NET Core app talk with Azure Storage services by performing the operations on Storage accounts using the available packages and code. In next chapter, I will continue with Azure Storage and learn about its features to host the application.

Do practice the exercise. Happy Azure Coding.

Questions

1. What are the different types of Blob Storage?
2. What is the difference between the Cool tier and the Hot tier?
3. What are different Performance tiers offering in Azure Storage?
4. What makes Azure File Share different from other File sharing system?
5. Explain Storage Data Redundancy?
6. Explain types of Data movement options available?

CHAPTER 5

Working With Microsoft Azure Storage as Hosting Option

In the previous chapter, we learned about the Microsoft Azure Storage account, a widely used Azure service as a storage option. We also learned about managing the Azure Blobs using .NET Core console application. This chapter can be considered as the continuation of the last chapter. Here I will make you go through an amazing feature of Azure Storage offered by Microsoft Azure for hosting the static contents.

Structure

- Initial setting up
- Files to deploy
- Azure setup
- Enable static website
- Azure CLI to enable Static website
- Uploading published files

Objectives

The aim of learning this chapter is to:

- Introduction to Microsoft Azure Storage Static website option.
- Enabling the Feature using Azure Portal &amp; Azure CLI .
- Azure Storage Explorer to manage Storage account
- Hosting .NET Core 3.1 Angular SPA.
- Hosting .NET Core 3.1 ReactJS SPA

Quick overview

Microsoft Azure offers a supercool feature with Microsoft Azure Storage named as **Static website**. It enables you to host static content in Blob. Content can be any files, HTML, JavaScript, CSS, or image files. Any files with client-side scripting can be used as content to host. At the time of writing, the server-side is not yet supported.

There comes occasion wherein you need to host an application or single page application with static files and contents, and that too for fewer days, a week time or so. Azure App service or any other hosting providers could be a little expensive here.

Think, if I say, you can host your any single page application or static files and content with minimal or almost free of charge on Azure? Shocked, but yes, it's true!

Microsoft Azure Storage offers a feature free of cost to host static content and files or any client-side scripting application on public cloud Azure, with publicly access endpoints, security, durability, and even with the use of the custom domain. This feature is known as the Static website, and at the time of writing this book, it comes with a **`General Purpose V2`** storage account only. You're billed only for the Blob storage that your site utilizes and operations costs.

For this chapter, I will make you go through hosting ASP.NET Core 3.1 Angular, ReactJS, and Redux ReactJS application host on Azure Storage account as a **`Static website`**.

Prerequisite

For achieving the objective, we need the following things in place:

- Visual Studio 2019
- ASP.NET Core 3.1 SDK
- Valid Azure Subscription
- Azure Storage Explorer (optional)
- Internet to run and test 😊

Initial settings up the things

Firstly, get this .NET Core 3.1 SDK (latest version) by going to the following link https://dotnet.microsoft.com/download/dotnet-core/3.0?WT.mc_id=microsoft-azuredevtips-micrum download, and install in your machine.

Once installed, open up Visual Studio 2019, I am using a community edition.

And click on **`Create a new Project`**:

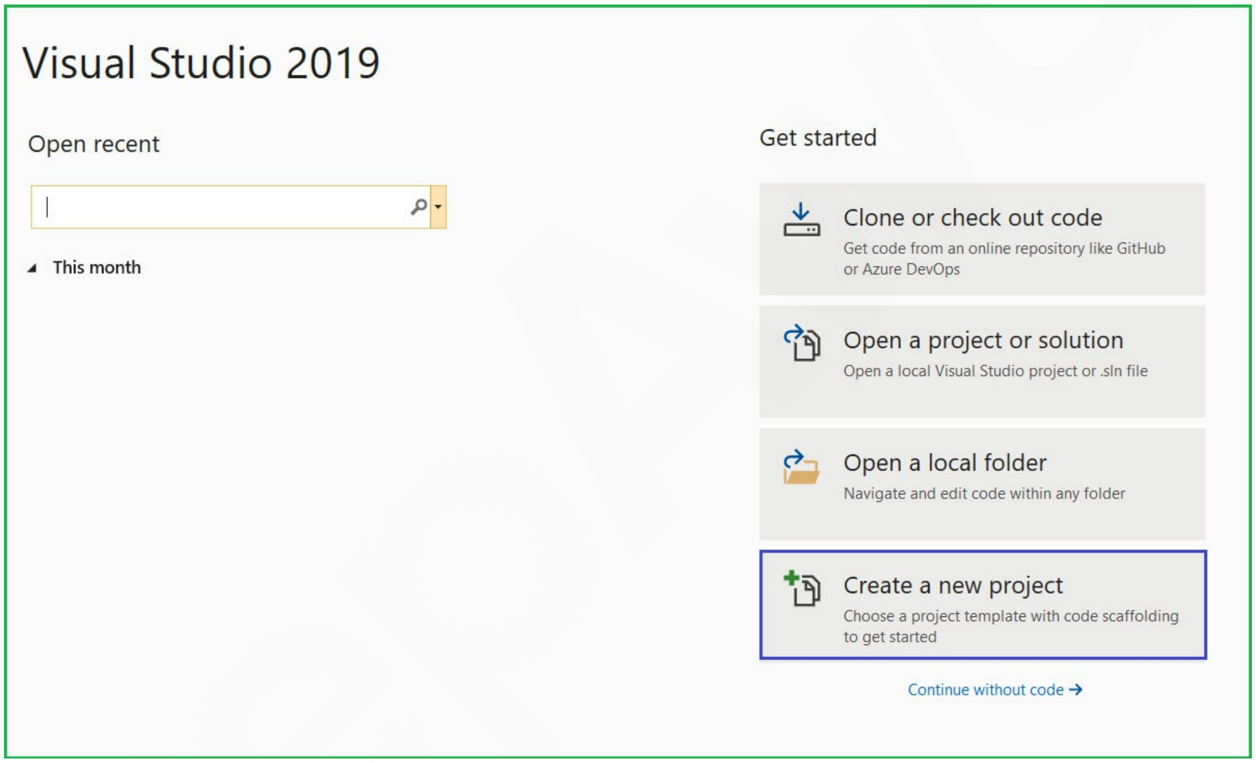
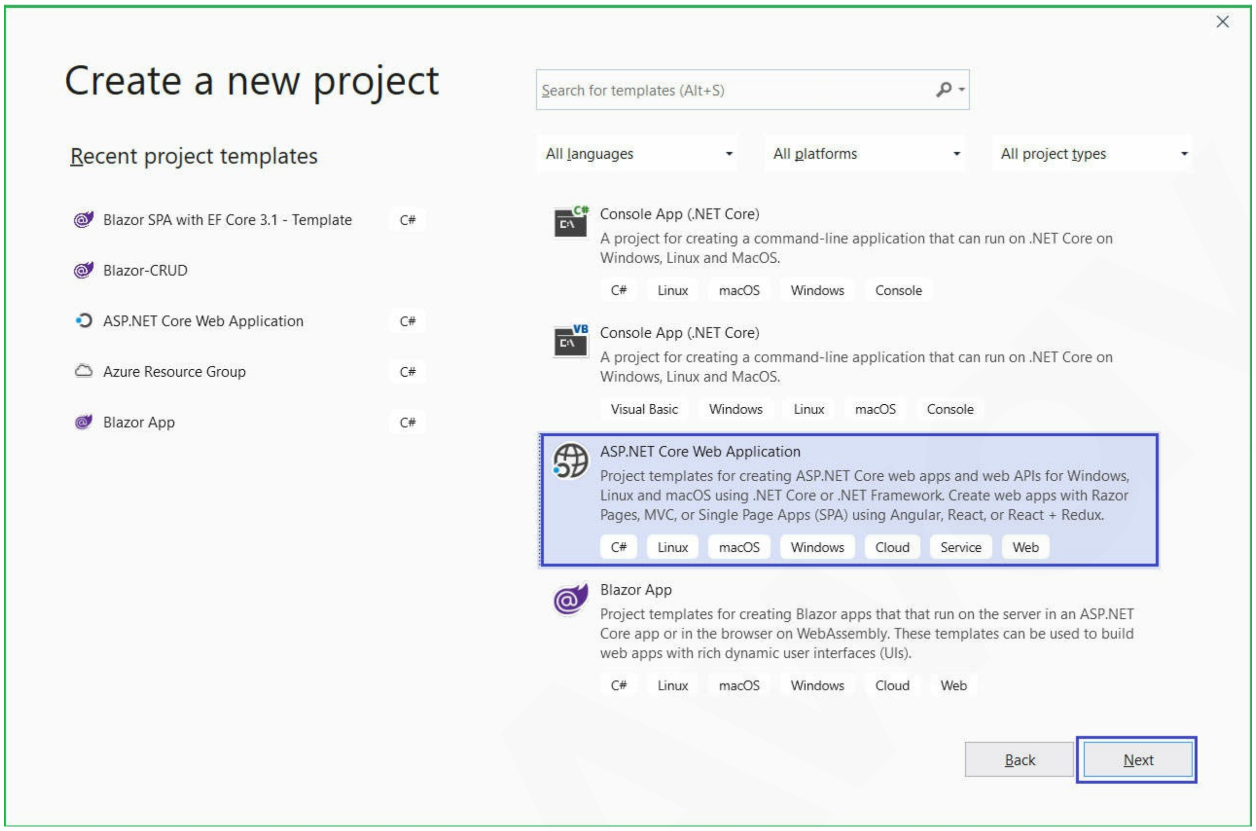

Visual Studio 2019
Open recent
This month
Get started
Clone or check out code
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

Figure 5.1

Select **`ASP.NET Core Web Application`**, and click on **`Next`**:


Create a new project

Search for templates (Alt+S)

Recent project templates

All languages | All platforms | All project types

Blazor SPA with EF Core 3.1 - Template C#

Blazor-CRUD

ASP.NET Core Web Application C#

Azure Resource Group C#

Blazor App C#

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

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
Visual Basic Windows Linux macOS 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

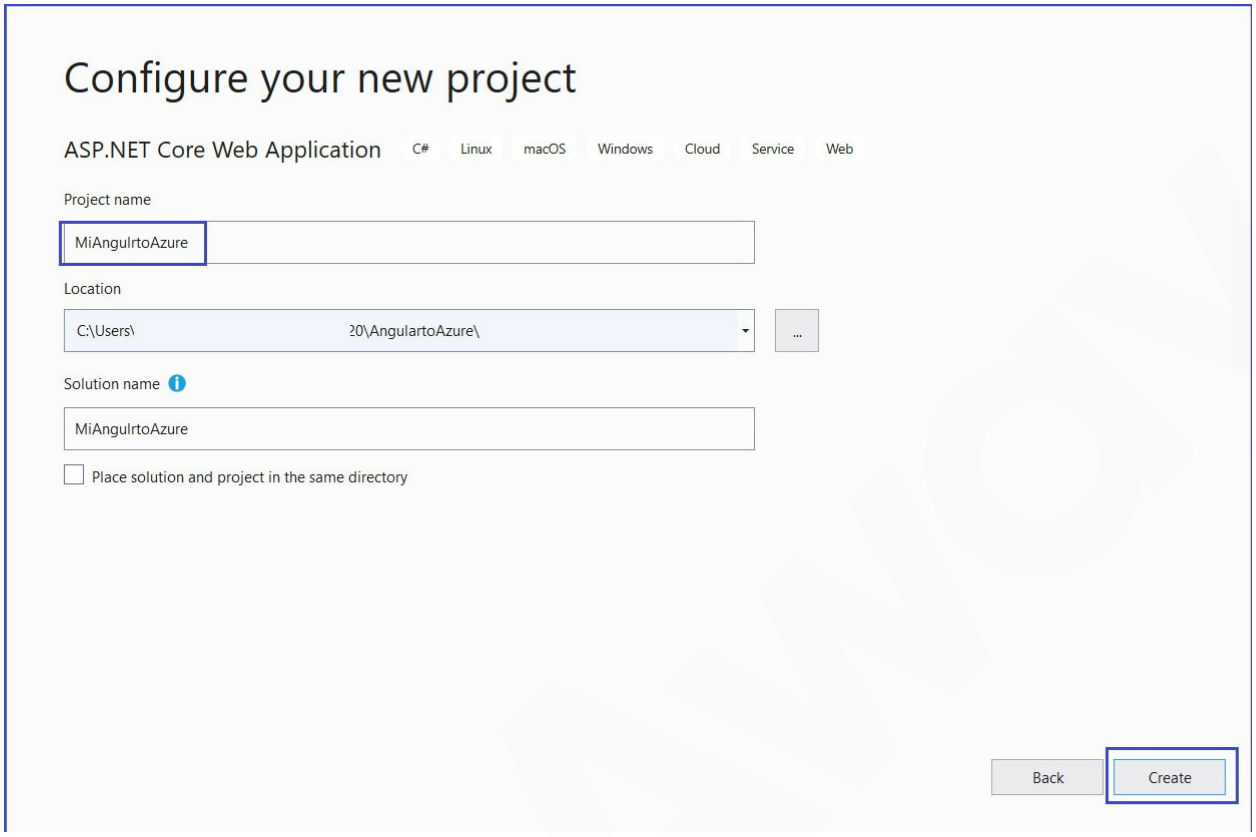
Blazor App
Project templates for creating Blazor apps that that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

Back | Next

Figure 5.2

Add a **`Project name`** for your **`Angular`** application, select the **`Location`**, and other information. For this article, have named app as `MiAngulartoAzure`, can be seen in the following image:

Click on **`Create`** to proceed:


Configure your new project

ASP.NET Core Web Application

Project name: MiAngulrtoAzure

Location: C:\Users\ 20\AngulartoAzure\

Solution name: MiAngulrtoAzure

☐ Place solution and project in the same directory

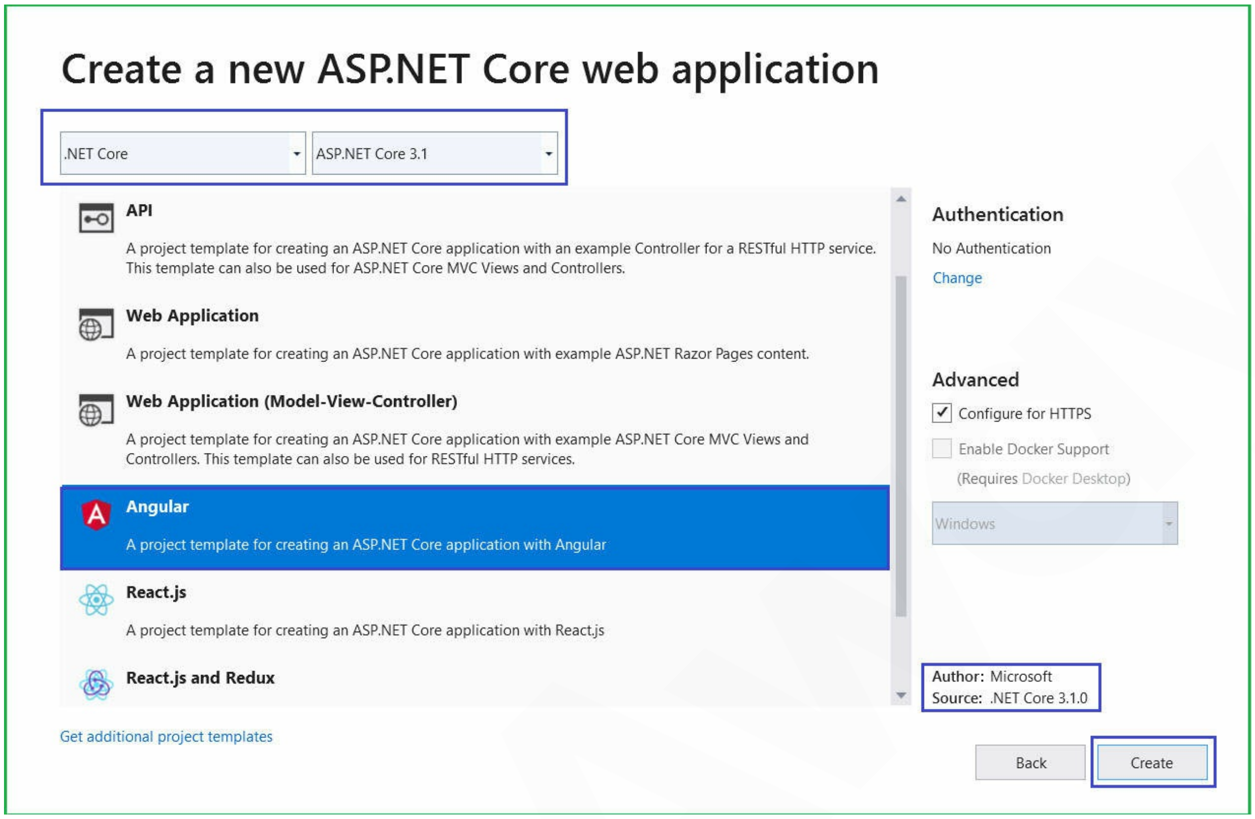
Figure 5.3

Now select stack as **`.NET Core`** and **`ASP.NET Core 3.1`** from the dropdown, to ensure the stack is used as required.

I will have an **`Angular`** application hosted first.

Select **`Angular`** from project templates listed. Make sure to check the source get displayed as **`.NET Core 3.1.0`**.

That’s it, click on **`Create`** to have the application created:



Create a new ASP.NET Core web application

.NET Core | ASP.NET Core 3.1

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.

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

Web Application (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.

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

React.js
A project template for creating an ASP.NET Core application with React.js

React.js and Redux

Authentication
No Authentication
Change

Advanced
✓ Configure for HTTPS
Enable Docker Support
(Requires Docker Desktop)
Windows

Author: Microsoft
Source: .NET Core 3.1.0

Figure 5.4

In fewer seconds, you can see the project ready in `Solution Explorer,` with a set of well-defined files and folders.

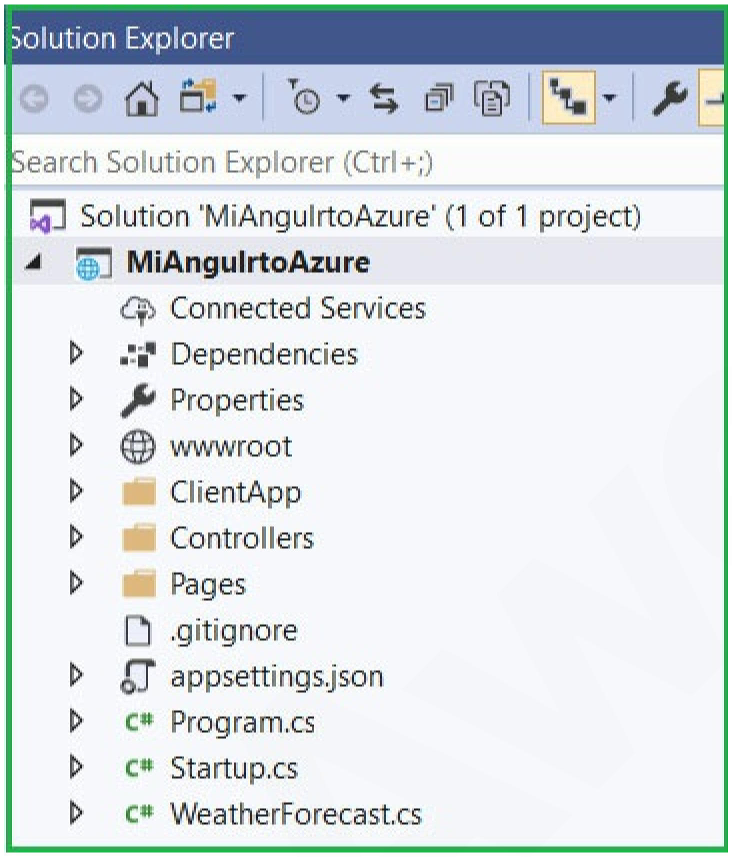
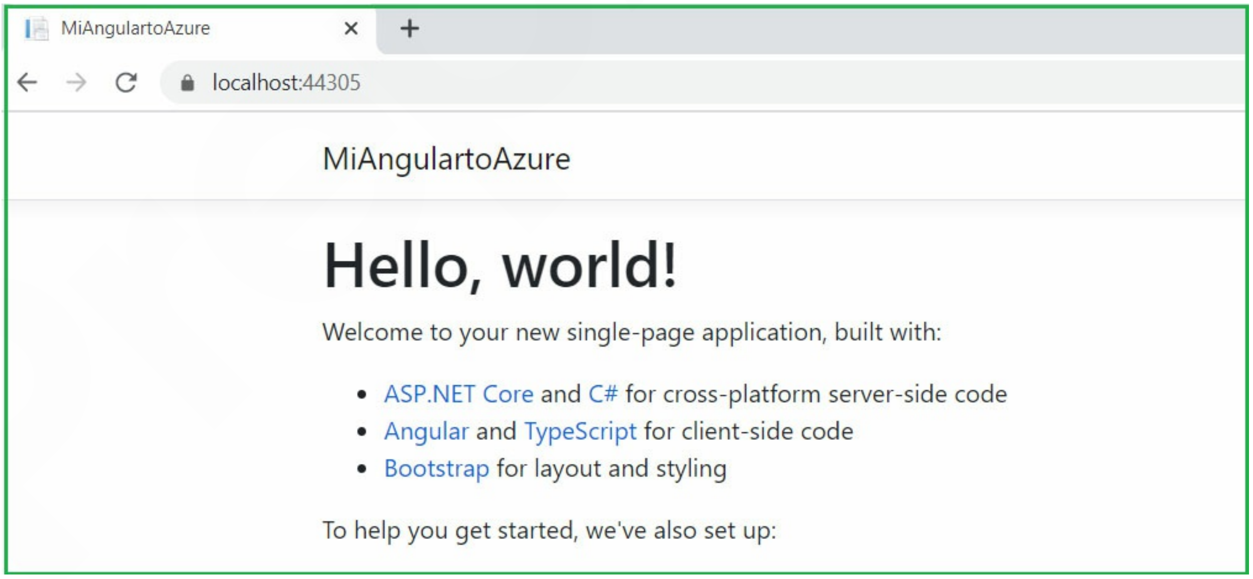


Figure 5.5

Let us run it:



Hello, world!

Welcome to your new single-page application, built with:

- ASP.NET Core and C# for cross-platform server-side code
- Angular and TypeScript for client-side code
- Bootstrap for layout and styling

To help you get started, we've also set up:

Figure 5.6

Files to deploy

Now in order to have this application on the cloud, we need to deploy or upload the project files to Microsoft Azure Storage Blob.

Select **`Project`** in **`Solution Explorer | Right Click | Publish`**:

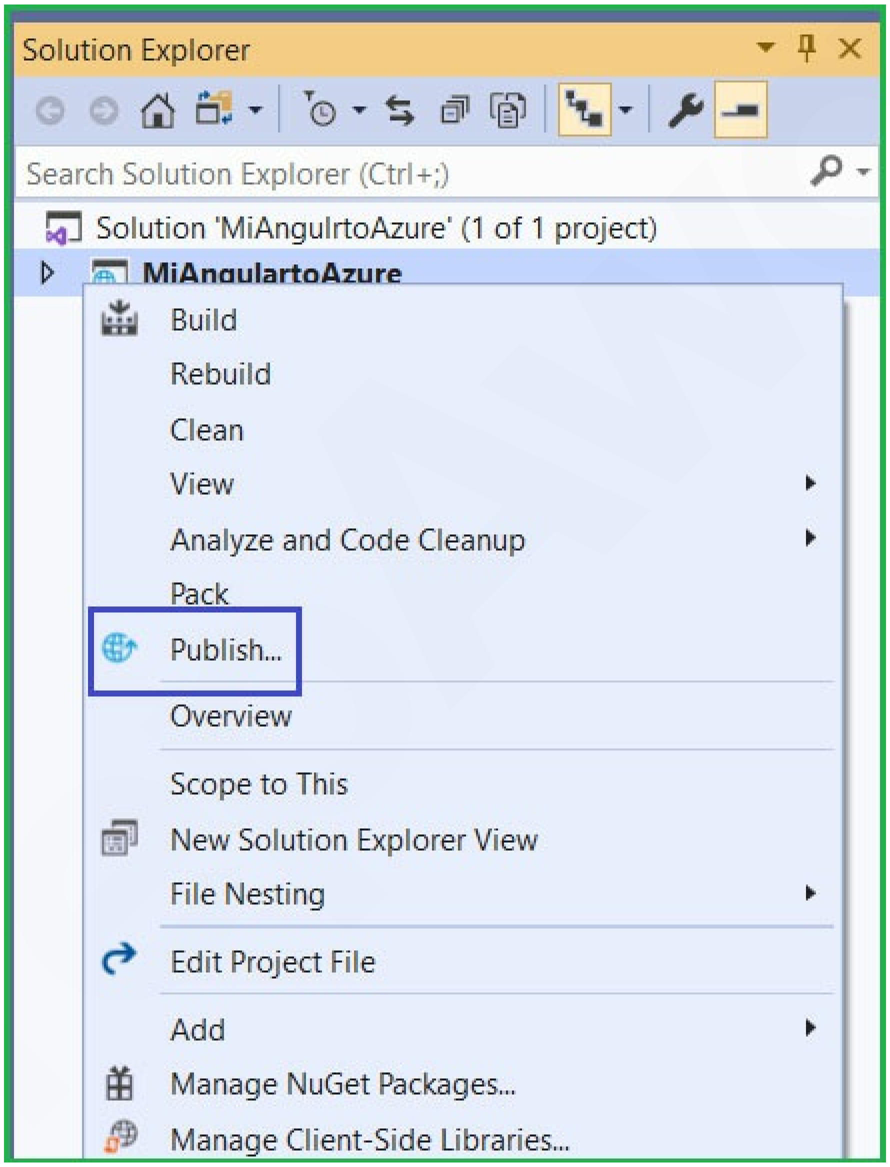


Figure 5.7

This will open a window with all available options.

Click on **`Publish`** target as Folder from left hand listed options. You can provide the path you want the application files to get published. We will keep the location as default.

Click on **`Create Profile`**:

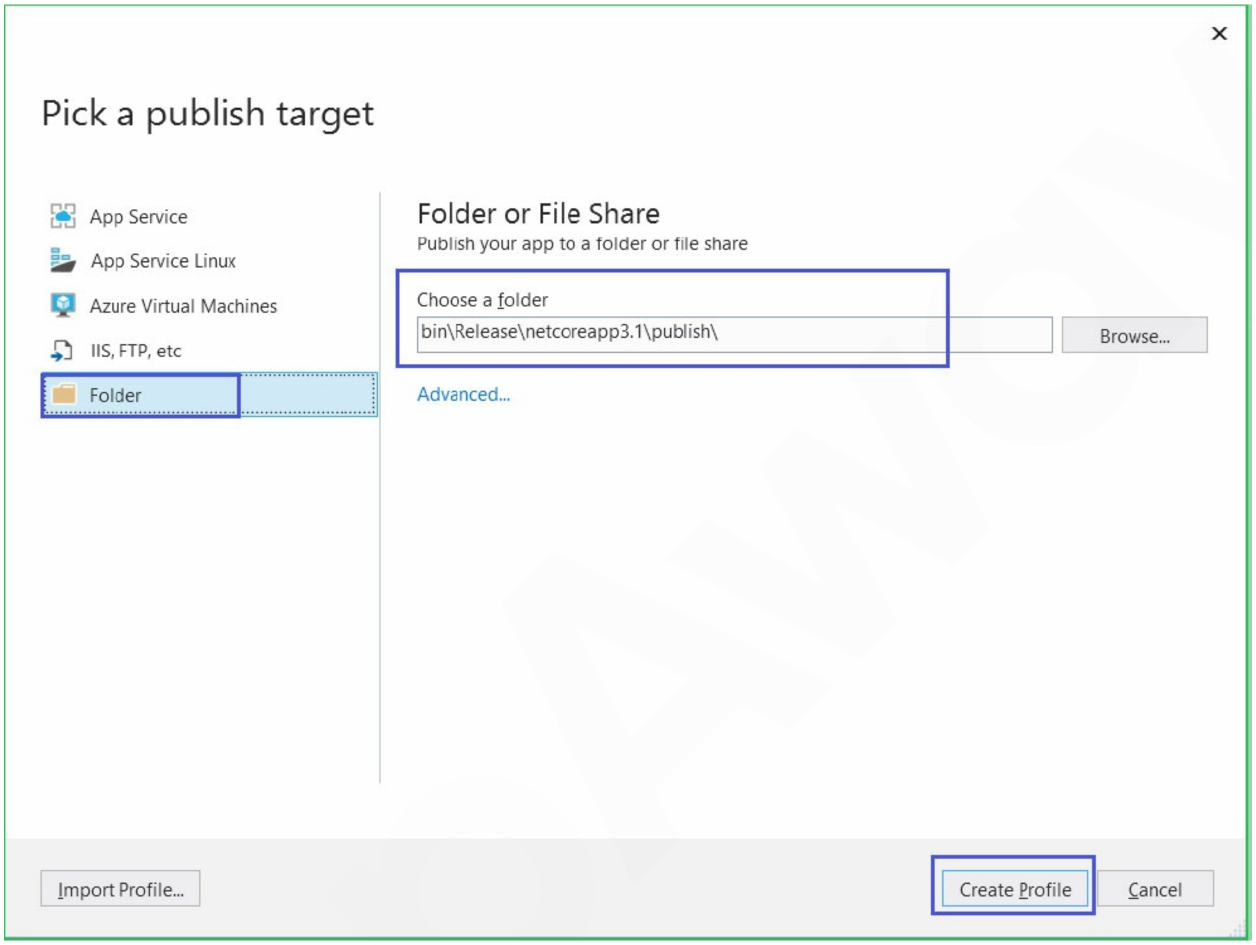
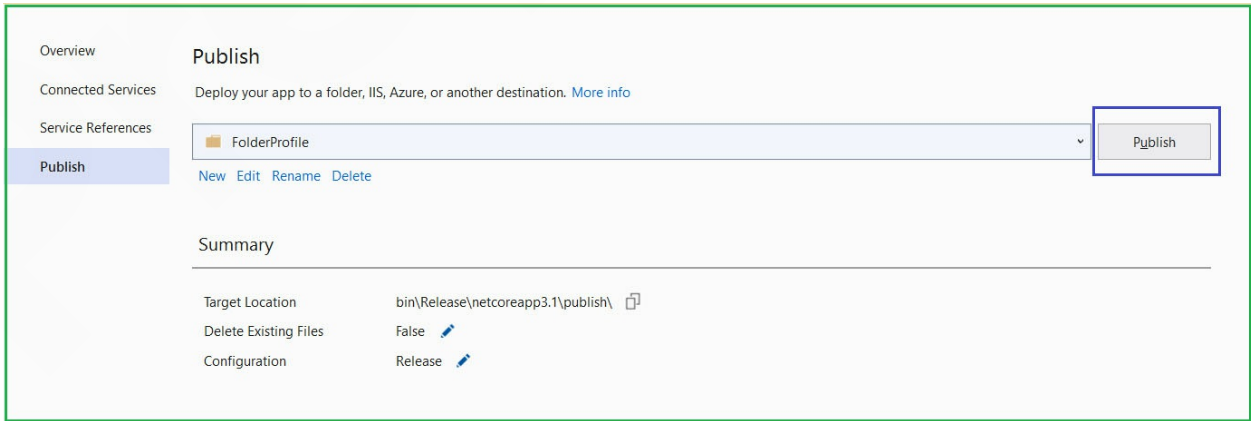


Figure 5.8

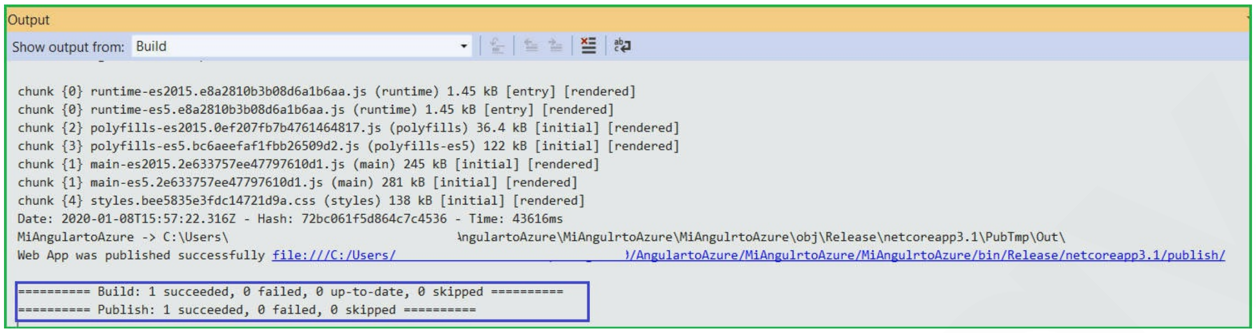
Once publish target is set, click on **`Publish`** to start with the activity:



Target Location	bin\Release\netcoreapp3.1\publish\
Delete Existing Files	False
Configuration	Release

Figure 5.9

Within fewer minutes you can the succeeded message in the output window:



```
Output
Show output from: Build

chunk {0} runtime-es2015.e8a2810b3b08d6a1b6aa.js (runtime) 1.45 kB [entry] [rendered]
chunk {0} runtime-es5.e8a2810b3b08d6a1b6aa.js (runtime) 1.45 kB [entry] [rendered]
chunk {2} polyfills-es2015.0ef207fb7b4761464817.js (polyfills) 36.4 kB [initial] [rendered]
chunk {3} polyfills-es5.bc6aeefaf1fbb26509d2.js (polyfills-es5) 122 kB [initial] [rendered]
chunk {1} main-es2015.2e633757ee47797610d1.js (main) 245 kB [initial] [rendered]
chunk {1} main-es5.2e633757ee47797610d1.js (main) 281 kB [initial] [rendered]
chunk {4} styles.bee5835e3fdc14721d9a.css (styles) 138 kB [initial] [rendered]
Date: 2020-01-08T15:57:22.316Z - Hash: 72bc061f5d864c7c4536 - Time: 43616ms
MiAngulartoAzure -> C:\Users\                    \ngulartoAzure\MiAngulrtoAzure\MiAngulrtoAzure\obj\Release\netcoreapp3.1\PubTmp\Out\
Web App was published successfully file:///C:/Users/                    )/AngulartoAzure/MiAngulrtoAzure/MiAngulrtoAzure/bin/Release/netcoreapp3.1/publish/

========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

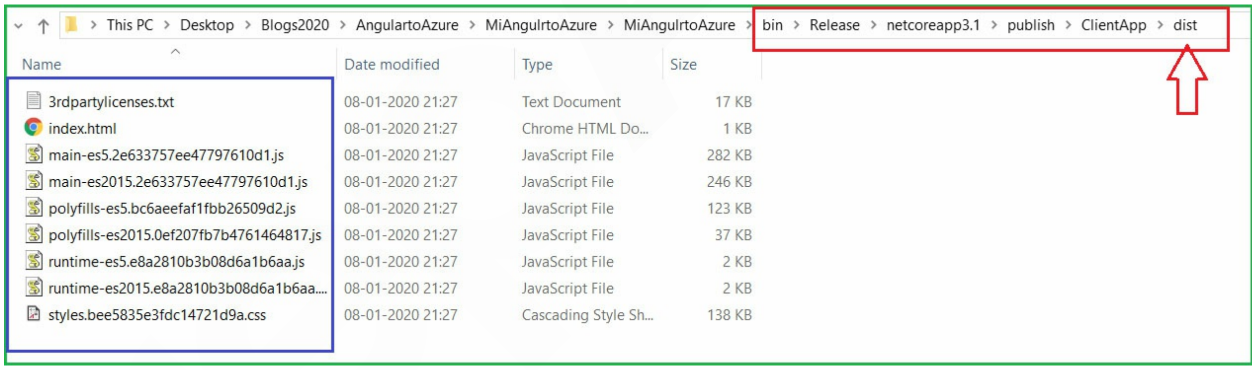
Figure 5.10

Now, click the location the files are published.

And navigate to **`Publish | ClientApp | dist folder.`**

Make sure only the files under the dist folder should be uploaded to the Microsoft Azure Storage Blob container.

Consider this as the most important step or take away from this article, to which files need to get deployed or uploaded to Azure for coming in as static content:



This PC > Desktop > Blogs2020 > AngulartoAzure > MiAngulrtoAzure > MiAngulrtoAzure > bin > Release > netcoreapp3.1 > publish > ClientApp > dist

Name	Date modified	Type	Size
3rdpartylicenses.txt	08-01-2020 21:27	Text Document	17 KB
index.html	08-01-2020 21:27	Chrome HTML Do...	1 KB
main-es5.2e633757ee47797610d1.js	08-01-2020 21:27	JavaScript File	282 KB
main-es2015.2e633757ee47797610d1.js	08-01-2020 21:27	JavaScript File	246 KB
polyfills-es5.bc6aeefaf1fbb26509d2.js	08-01-2020 21:27	JavaScript File	123 KB
polyfills-es2015.0ef207fb7b4761464817.js	08-01-2020 21:27	JavaScript File	37 KB
runtime-es5.e8a2810b3b08d6a1b6aa.js	08-01-2020 21:27	JavaScript File	2 KB
runtime-es2015.e8a2810b3b08d6a1b6aa....	08-01-2020 21:27	JavaScript File	2 KB
styles.bee5835e3fdc14721d9a.css	08-01-2020 21:27	Cascading Style Sh...	138 KB

Figure 5.11

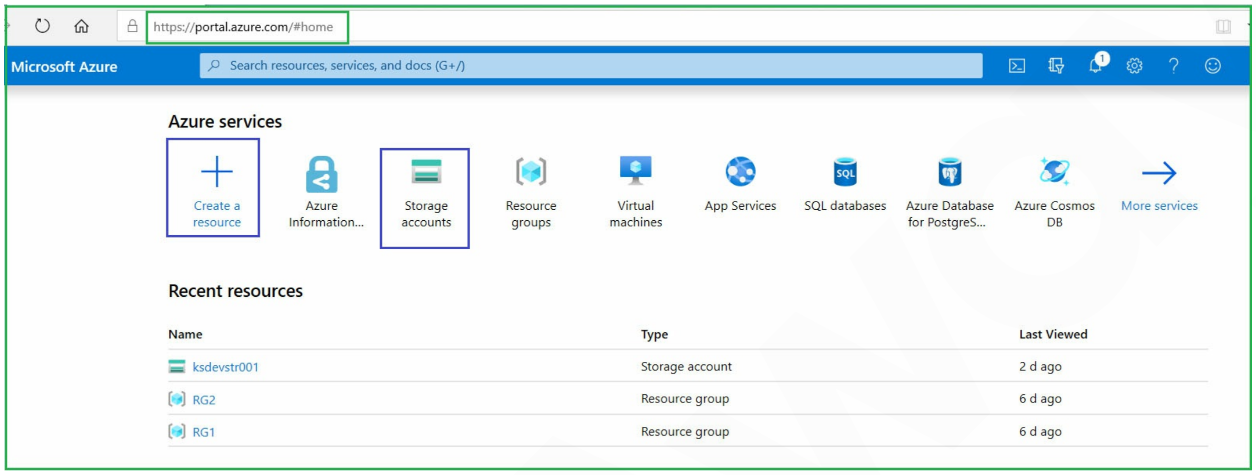
Azure setup

Here we first need to have:

- Microsoft Azure Storage account created
- Enable Static website feature
- Upload the preceding, published files

Create a Microsoft Azure Storage account.

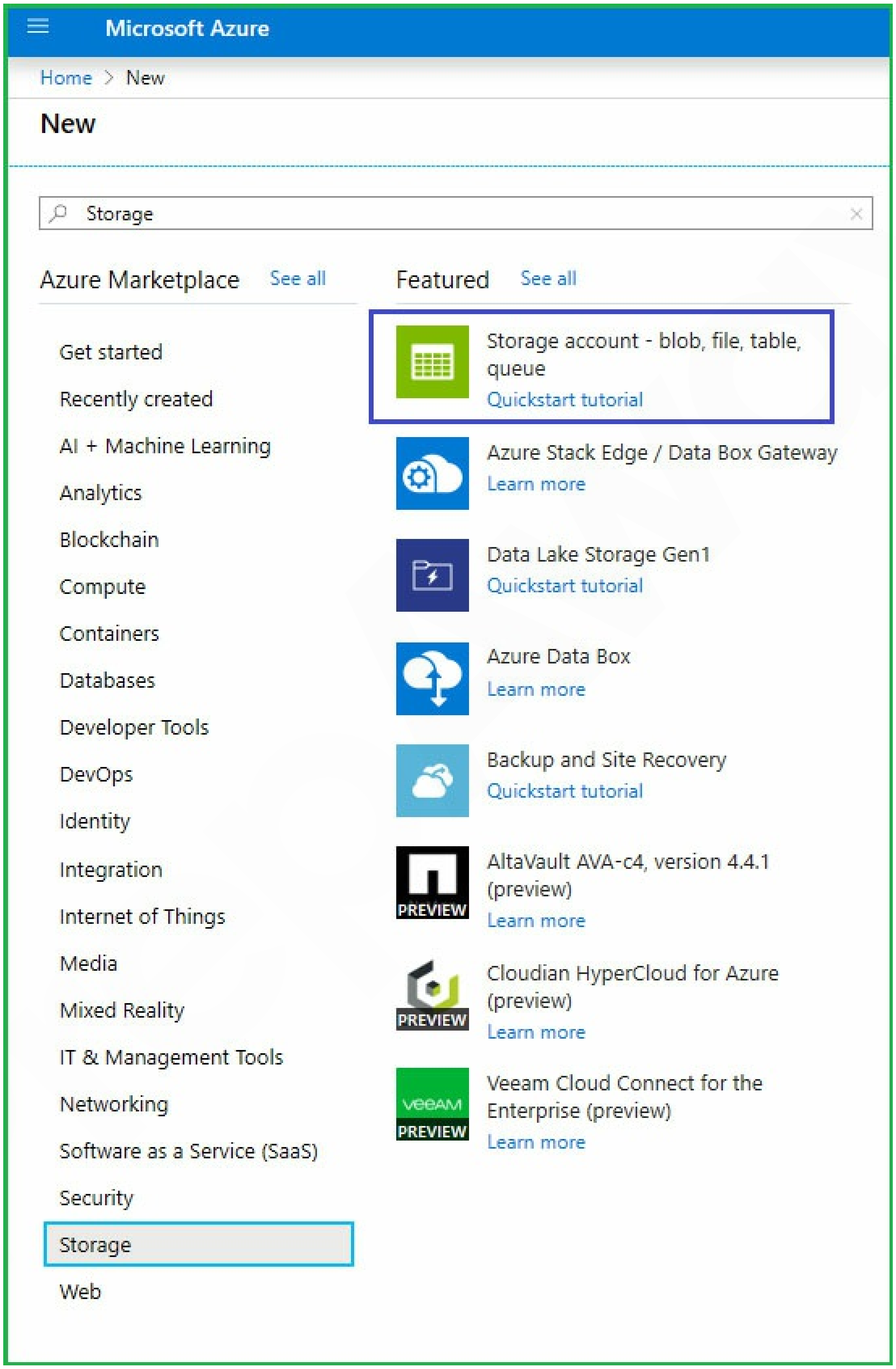
Open **`Microsoft Azure Portal`** in any browser, and click on **`Create a resource`** or **`Storage`** from listed **`Azure services`**:



Name	Type	Last Viewed
ksdevstr001	Storage account	2 d ago
RG2	Resource group	6 d ago
RG1	Resource group	6 d ago

Figure 5.12

From services listed in **`Azure Marketplace`**, select **`Storage`** from left **`pane | Storage account:`**



Microsoft Azure

Home > New

New

Storage

Azure Marketplace See all

- Get started
- Recently created
- AI + Machine Learning
- Analytics
- Blockchain
- Compute
- Containers
- Databases
- Developer Tools
- DevOps
- Identity
- Integration
- Internet of Things
- Media
- Mixed Reality
- IT & Management Tools
- Networking
- Software as a Service (SaaS)
- Security
- Storage
- Web

Featured See all

- Storage account - blob, file, table, queue
  Quickstart tutorial
- Azure Stack Edge / Data Box Gateway
  Learn more
- Data Lake Storage Gen1
  Quickstart tutorial
- Azure Data Box
  Learn more
- Backup and Site Recovery
  Quickstart tutorial
- AltaVault AVA-c4, version 4.4.1 (preview)
  Learn more
- Cloudian HyperCloud for Azure (preview)
  Learn more
- Veeam Cloud Connect for the Enterprise (preview)
  Learn more

Figure 5.13

Once you select the **`Storage`** service, you will be presented with a form in the blade to enter details with respect to your storage account.

It comes with five tabs to configure details with, namely: **`Basics, Networking, Advanced, Tags, Review + Create`**.

This chapter will focus more on entering the **`Basic`** details and will keep default values for rest required tabs.

Now, the **`Basic`** requires mandatory project details such as:

- **`Subscription:`** Here, you need to select your subscription. I have selected my free trails account.
- **`Resource group:`** It's a logical grouping of your Azure resources. You can select from an existing resource group or can create a new resource group, as per your choice. For this article, I have created a new resource group name, `MiAngulartoAzure`.

Next comes the instance details:

- **`Storage Account Name:`** The name must be unique across all existing storage account names in Azure. It must be 3 to 24 characters long and can contain only lowercase letters and numbers. For this article, I have given the name as miangularspa.

  Note: This name will be the part of your application public URL, so name it wisely.

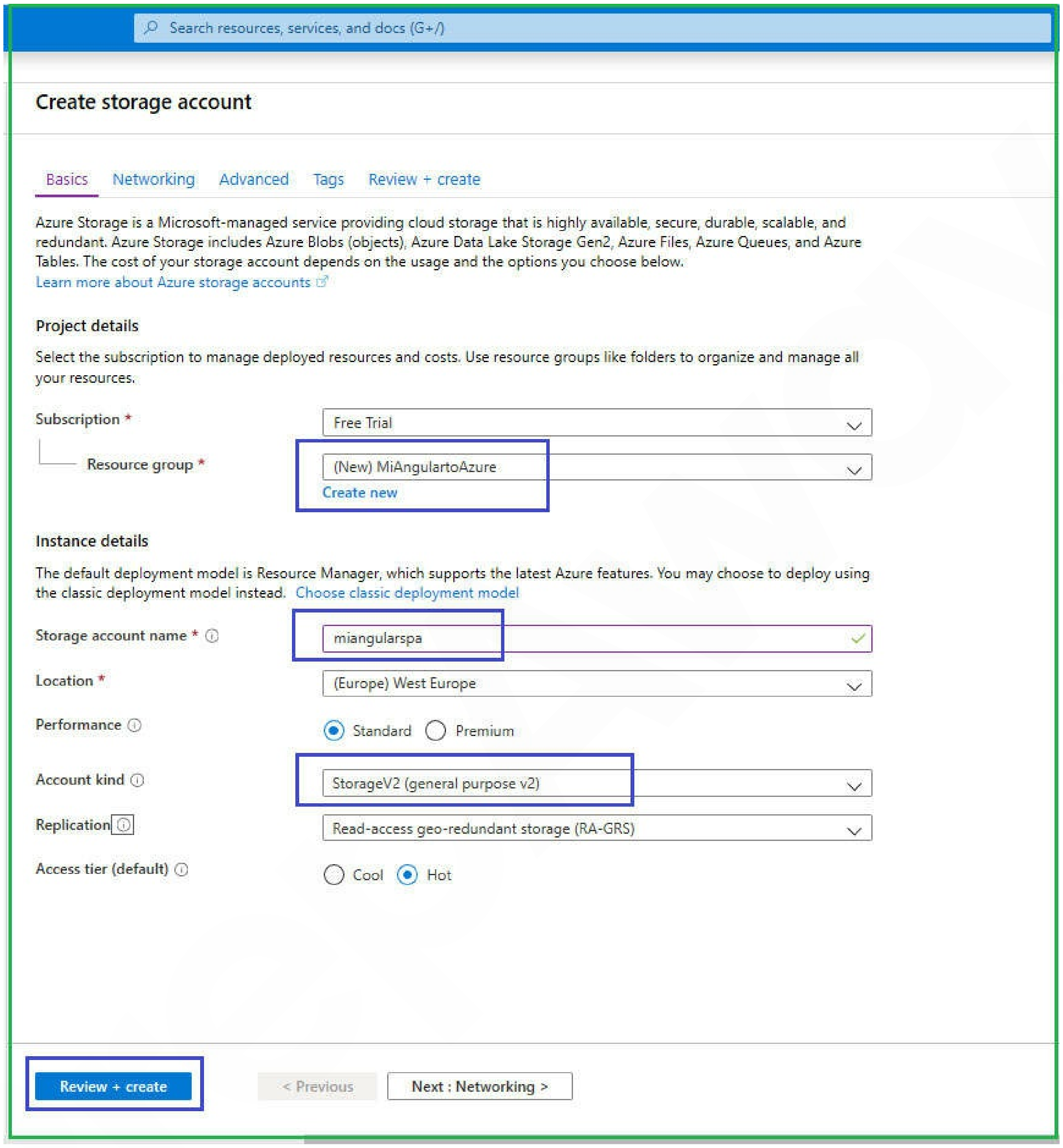
- **Location:** Region, your storage azure resource, will get deployed. This can be from widely available azure regions and as per your target location. For this article, I have selected the **`European`** region.
- **Performance:** This comes with two options, **`Standard`**, backed by magnetic drives, and **`Premium`**, backed with SSD drives. This could be considered as one of the pricing factors. For this article, I have selected as **`Standard`**, the default one, and the recommended one when using the **`Static website`**.
- **Account Kind:** It is a very important detail to configure. This comes with three different options to select from. As you need to use the storage as for hosting Static websites, you need to select **`StorageV2 (general purpose v2)`** only. This is because, at the time of writing this

article, it's only said storage supports the Static website feature.

- **`Replication:`** This is a pattern of data replication being carried out with the storage by Azure, mainly used to ensure durability and high availability. I will go with the default option selected.
- **`Access tier:`** Again, one of the factors incurring costing. It all decides the nature of your use with data stored in the storage account. **`Cool`** tier is mainly used for archive type of data, rarely used. Hot comes with the data used frequently. I will go with default as **`Hot`** tier.

Once all the details are entered, click on **`Next`** for entering details required in other tabs. As mentioned earlier, we will add only **`Basic`** details and will go with rest values as default.

Click on **`Review + create`** to proceed:



Search resources, services, and docs (G+/)

Create storage account

Basics | Networking | Advanced | Tags | Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below.
Learn more about Azure storage accounts

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Free Trial
Resource group *	(New) MiAngulartoAzure Create new

Instance details

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. Choose classic deployment model

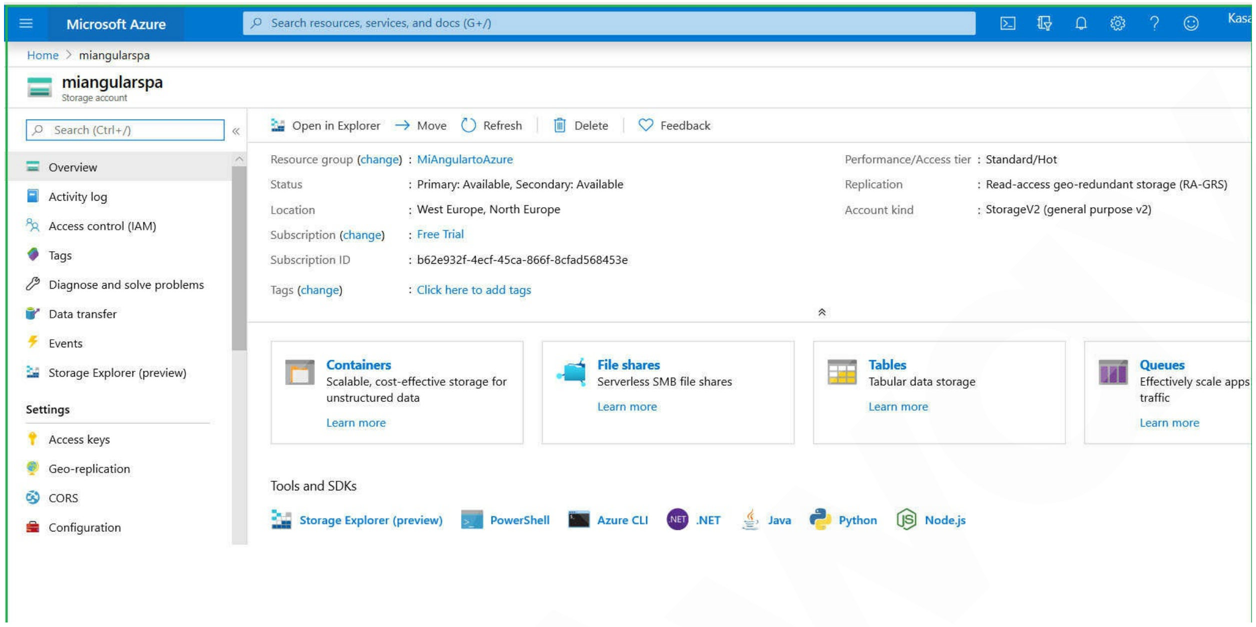
Storage account name *	miangularspa
Location *	(Europe) West Europe
Performance	[✓ Standard] Premium
Account kind	StorageV2 (general purpose v2)
Replication	Read-access geo-redundant storage (RA-GRS)
Access tier (default)	Cool [✓ Hot]

Figure 5.14

This will validate all the details provided, and once it passed will enable the **`Create`** button. Click on the **`Create`** button to start with the resource creation.

Keep an eye on notification, as you will be notified with the resource creation progress.

In fewer seconds, a **`Storage`** account will be getting created. Navigate to our storage account:



Resource group (change) : MiAngulartoAzure
Status : Primary: Available, Secondary: Available
Location : West Europe, North Europe
Subscription (change) : Free Trial
Subscription ID : b62e932f-4ecf-45ca-866f-8cfad568453e
Tags (change) : Click here to add tags

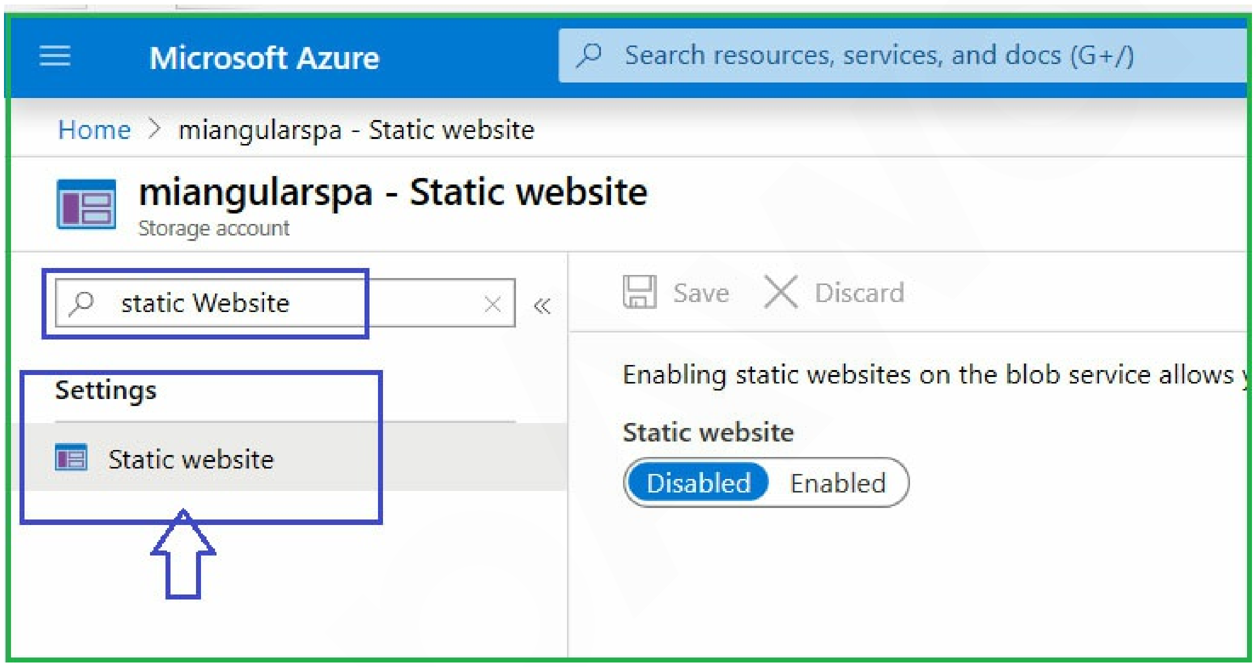
Performance/Access tier : Standard/Hot
Replication : Read-access geo-redundant storage (RA-GRS)
Account kind : StorageV2 (general purpose v2)

Figure 5.15

Enable Static website

Now once the storage account is created, it's time to enable the Static website feature for the blob storage.

You can type Static Website in the left blade search box. Alternately, you can navigate in the left blade under **`Settings |Static website.`** Click the **`option`**.


Microsoft Azure
Search resources, services, and docs (G+/)
Home > miangularspa - Static website
miangularspa - Static website
Storage account
static Website
Settings
Static website
Save
Discard
Static website
Disabled
Enabled

Enabling static websites on the blob service allows

Figure 5.16

This will lead to the **`Static website`** page, as seen in the preceding image.

By default, this comes as **`Disabled`**.

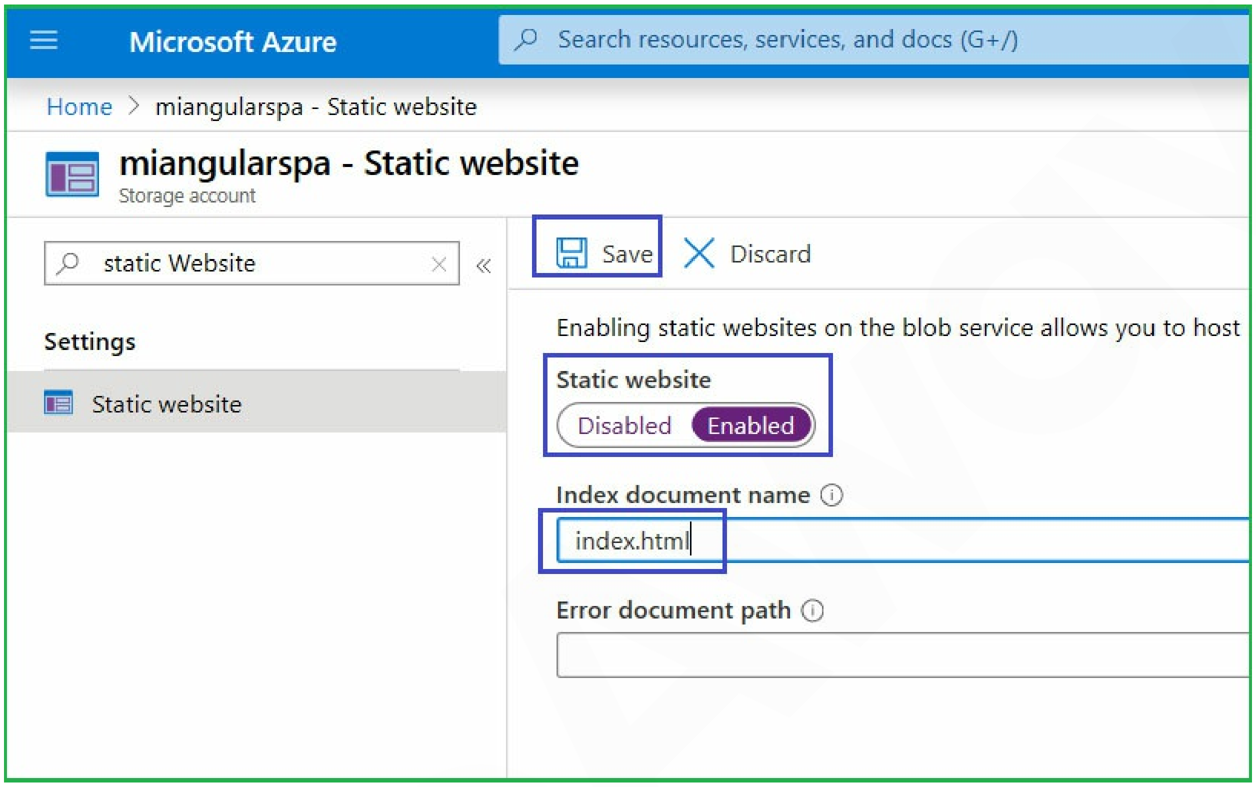
To enable the same, toggle the option to **`Enabled`**.

Once you made it to **`Enabled`**, you will be prompted with two more text areas to provide details as follows:

- **`Index document name:`** This is the name of the webpage that Azure Storage returns when a request is made to the root of the website or any subfolder. Here I will enter the page name as `Index.html`. Recall, when we published the application, we had one file with the same name.
- **`Error document path:`** Path of the 404 documents. The document will be displayed when Azure Storage returns a 404 error. I will keep this

option blank.

Click on **Save**:


Microsoft Azure
Search resources, services, and docs (G+/)
Home > miangularspa - Static website
miangularspa - Static website
Storage account
static Website
Settings
Static website
Save
Discard

Enabling static websites on the blob service allows you to host

Static website: Disabled [✓ Enabled]

Index document name: index.html

Error document path:

Figure 5.17

When you configure your storage account for **Static website** hosting, you will be presented by two public endpoints alias your public URL to access the static contents:

- **Primary endpoint:** The website can be accessed via this region-specific web endpoint. It is different from the blob storage endpoint, which you use to upload new content. Endpoint are named as, **https://<storage account name>.z6.web.core.windows.net/**. In our case, the endpoint is, https://miangularspa.z6.web.core.windows.net/.
- **Secondary endpoint:** The only difference here from the preceding endpoint, it's in the way it's formed. **https://<storage account name>-secondary.z6.web.core.windows.net/.** In our case, the secondary endpoint is https://miangularspa-secondary.z6.web.core.windows.net/.

Upload the preceding published files

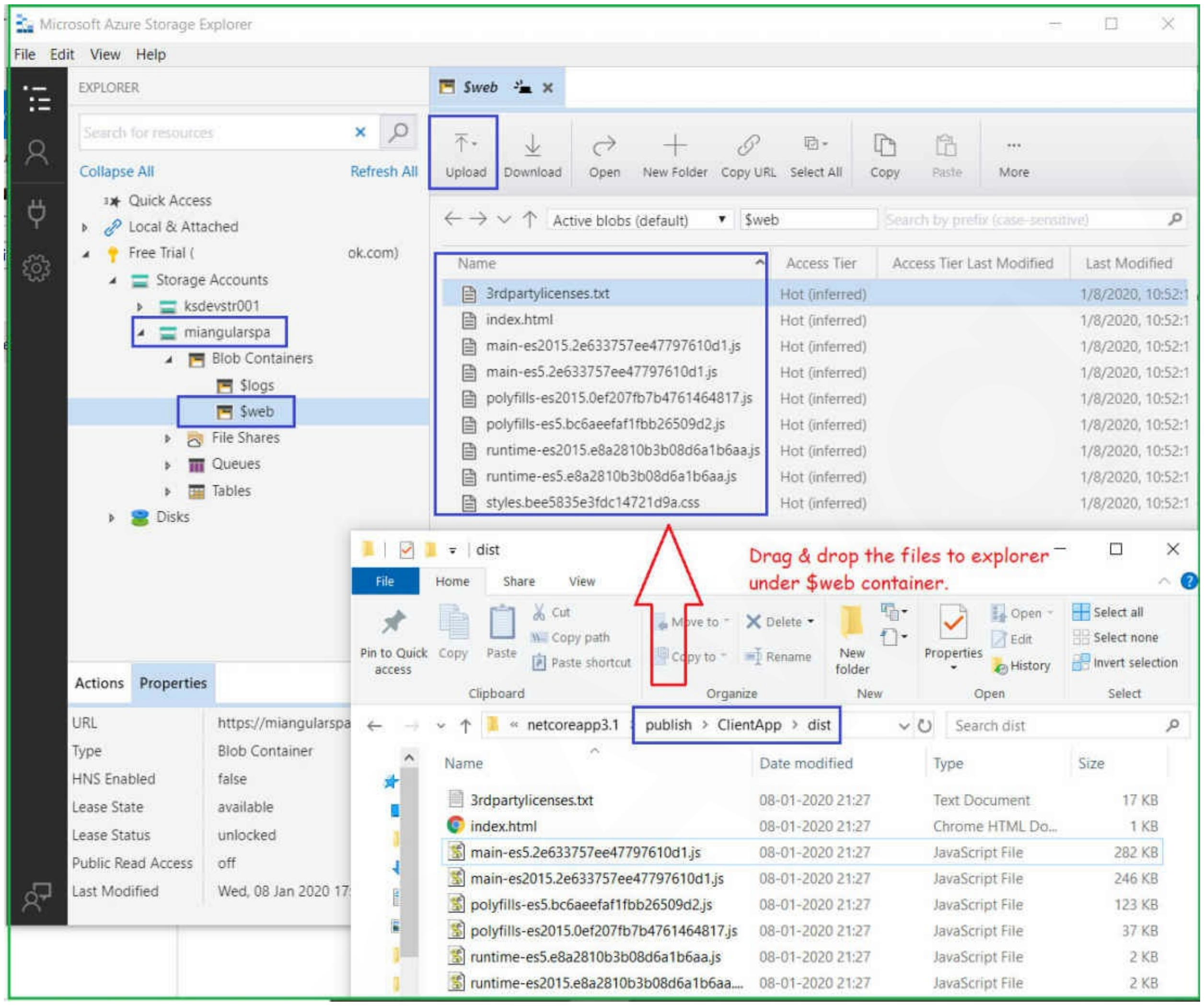
The next step and the last for this exercise is to upload the files to the blob storage. Here you can upload the files with different options available via Azure Portal, **`Azure Storage Explorer`**, Azure CLI, Rest API, and so on. For this article, I will use Azure Storage explorer:

Azure Storage explorer is free to download desktop client used to manage Azure Storage accounts. Working with Azure storage becomes too ease with storage explorer.

Open up **`Azure Storage Explorer`**, add your Azure account. This will list all the Storage accounts associated with your Azure account or subscription.

Navigate to Azure Storage account you created, and under **`Blob Containers`**, click on container name **`$web`**. This container was created when you enabled the **`Static website`** feature for Azure storage.

Click on **`Upload`** to select the published files (under folder name dist) or just drag and drop the files from the pc folder to **`Storage Explorer.`** Within a few seconds, all the files will be uploaded to the container, and this is the ease I mentioned earlier, comes in working with **`Microsoft Azure Storage Explorer`**:



Name	Access Tier	Access Tier Last Modified	Last Modified
3rdpartylicenses.txt	Hot (inferred)		1/8/2020, 10:52:1
index.html	Hot (inferred)		1/8/2020, 10:52:1
main-es2015.2e633757ee47797610d1.js	Hot (inferred)		1/8/2020, 10:52:1
main-es5.2e633757ee47797610d1.js	Hot (inferred)		1/8/2020, 10:52:1
polyfills-es2015.0ef207fb7b4761464817.js	Hot (inferred)		1/8/2020, 10:52:1
polyfills-es5.bc6aeefaf1fbb26509d2.js	Hot (inferred)		1/8/2020, 10:52:1
runtime-es2015.e8a2810b3b08d6a1b6aa.js	Hot (inferred)		1/8/2020, 10:52:1
runtime-es5.e8a2810b3b08d6a1b6aa.js	Hot (inferred)		1/8/2020, 10:52:1
styles.bee5835e3fdc14721d9a.css	Hot (inferred)		1/8/2020, 10:52:1

Actions Properties

URL	https://miangularspa
Type	Blob Container
HNS Enabled	false
Lease State	available
Lease Status	unlocked
Public Read Access	off
Last Modified	Wed, 08 Jan 2020 17:

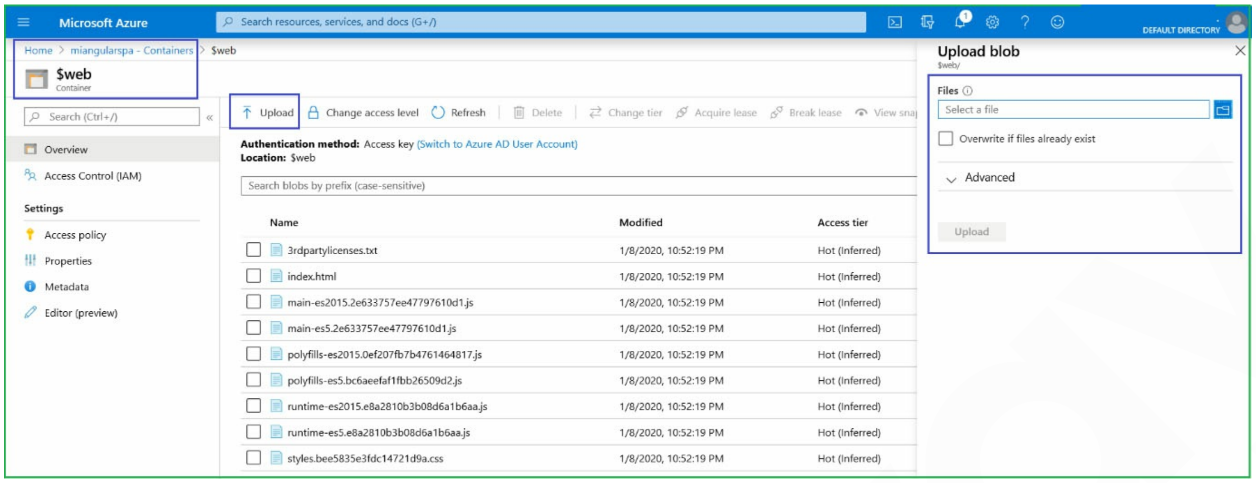
Name	Date modified	Type	Size
3rdpartylicenses.txt	08-01-2020 21:27	Text Document	17 KB
index.html	08-01-2020 21:27	Chrome HTML Do...	1 KB
main-es5.2e633757ee47797610d1.js	08-01-2020 21:27	JavaScript File	282 KB
main-es2015.2e633757ee47797610d1.js	08-01-2020 21:27	JavaScript File	246 KB
polyfills-es5.bc6aeefaf1fbb26509d2.js	08-01-2020 21:27	JavaScript File	123 KB
polyfills-es2015.0ef207fb7b4761464817.js	08-01-2020 21:27	JavaScript File	37 KB
runtime-es5.e8a2810b3b08d6a1b6aa.js	08-01-2020 21:27	JavaScript File	2 KB
runtime-es2015.e8a2810b3b08d6a1b6aa...	08-01-2020 21:27	JavaScript File	2 KB

Figure 5.18

If you don't have Azure Storage explorer or want to use **`Azure Portal`** to push the files, you can directly navigate **`to Storage Account section | Under Container | Click $Web container`**.

From top pane, click on the **`Upload`** option. A blade with options to upload the files will be open up at the right-hand side of the page.

Browse, select, and upload the file, as simple as that, as shown in the following image:



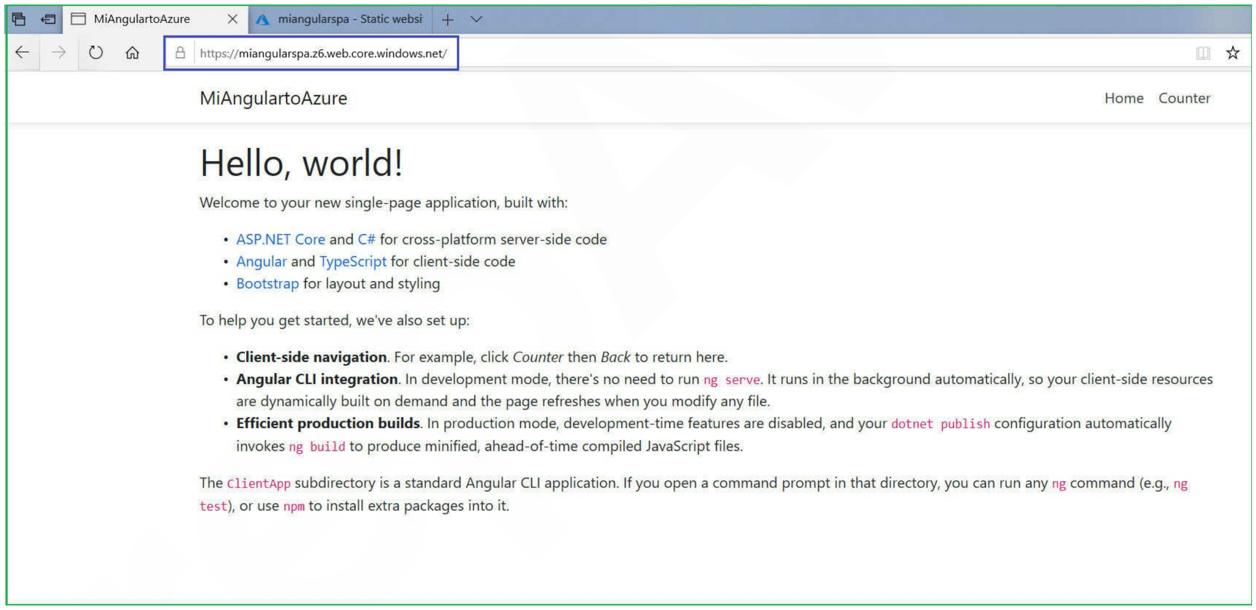
Authentication method: Access key (Switch to Azure AD User Account)
Location: $web

Search blobs by prefix (case-sensitive)

Name	Modified	Access tier
3rdpartylicenses.txt	1/8/2020, 10:52:19 PM	Hot (Inferred)
index.html	1/8/2020, 10:52:19 PM	Hot (Inferred)
main-es2015.2e633757ee47797610d1.js	1/8/2020, 10:52:19 PM	Hot (Inferred)
main-es5.2e633757ee47797610d1.js	1/8/2020, 10:52:19 PM	Hot (Inferred)
polyfills-es2015.0ef207fb7b4761464817.js	1/8/2020, 10:52:19 PM	Hot (Inferred)
polyfills-es5.bc6aeefaf1fbb26509d2.js	1/8/2020, 10:52:19 PM	Hot (Inferred)
runtime-es2015.e8a2810b3b08d6a1b6aa.js	1/8/2020, 10:52:19 PM	Hot (Inferred)
runtime-es5.e8a2810b3b08d6a1b6aa.js	1/8/2020, 10:52:19 PM	Hot (Inferred)
styles.bee5835e3fdc14721d9a.css	1/8/2020, 10:52:19 PM	Hot (Inferred)

Figure 5.19

After pushing all the files, let's open up the primary endpoint in the browser, presented after enabling the **`Static website`** feature:


MiAngulartoAzure
miangularspa - Static websi
https://miangularspa.z6.web.core.windows.net/
MiAngulartoAzure
Home
Counter

Hello, world!

Welcome to your new single-page application, built with:

- ASP.NET Core and C# for cross-platform server-side code
- Angular and TypeScript for client-side code
- Bootstrap for layout and styling

To help you get started, we've also set up:

- **Client-side navigation**. For example, click *Counter* then *Back* to return here.
- **Angular CLI integration**. In development mode, there's no need to run `ng serve`. It runs in the background automatically, so your client-side resources are dynamically built on demand and the page refreshes when you modify any file.
- **Efficient production builds**. In production mode, development-time features are disabled, and your `dotnet publish` configuration automatically invokes `ng build` to produce minified, ahead-of-time compiled JavaScript files.

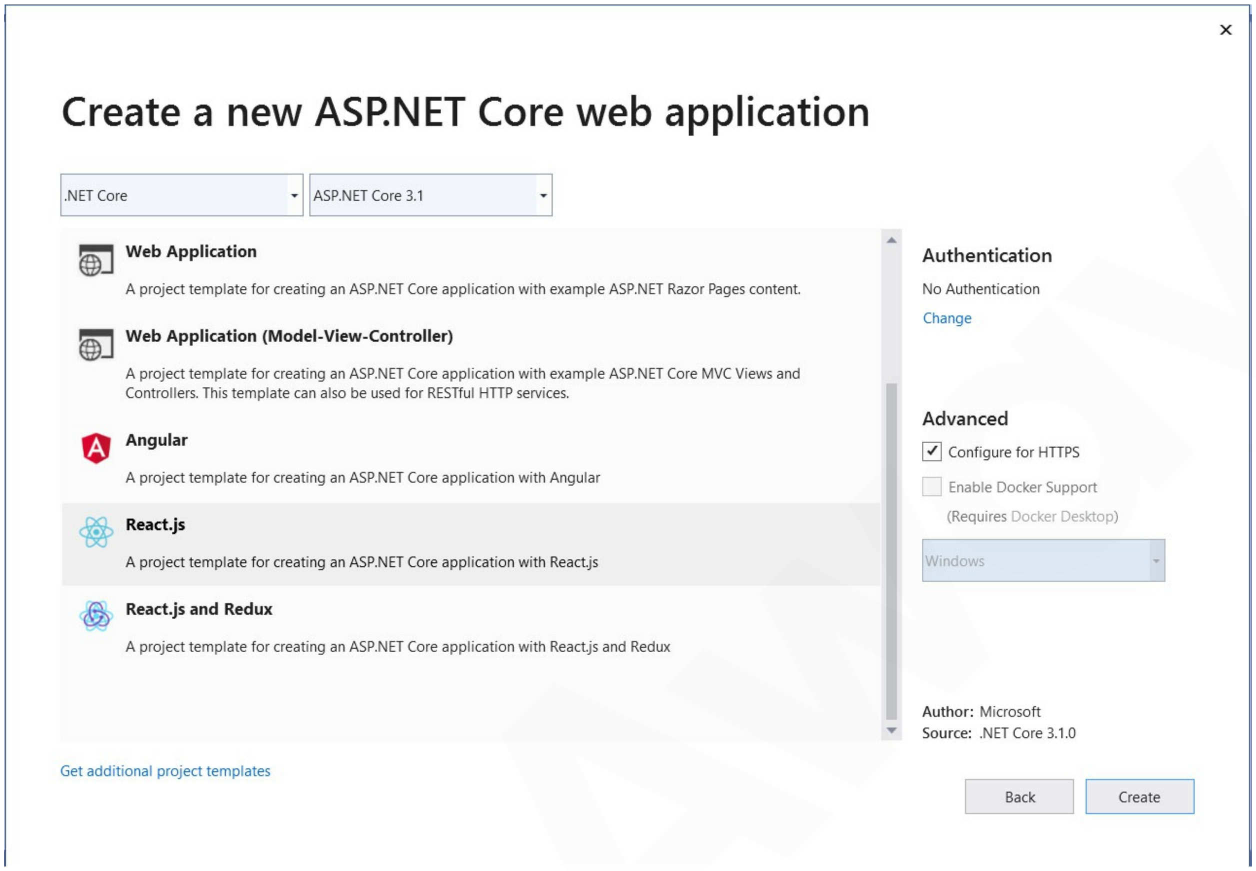
The `ClientApp` subdirectory is a standard Angular CLI application. If you open a command prompt in that directory, you can run any `ng` command (e.g., `ng test`), or use `npm` to install extra packages into it.

Figure 5.20

Cool, so your **`ASP.NET Core 3.1 Angular`** application is now live on public cloud, Azure:

Hosting **`.NET Core 3.1 ReactJS SPA.`**

In a similar fashion, you created an **`Angular`** application; let's create an **`ASP.NET Core 3.1 application with React.js`**:



Create a new ASP.NET Core web application

.NET Core | ASP.NET Core 3.1

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

Web Application (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.

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

React.js
A project template for creating an ASP.NET Core application with React.js

React.js and Redux
A project template for creating an ASP.NET Core application with React.js and Redux

Authentication
No Authentication
Change

Advanced
✓ Configure for HTTPS
Enable Docker Support
(Requires Docker Desktop)
Windows

Author: Microsoft
Source: .NET Core 3.1.0

Get additional project templates

Figure 5.21

In fewer seconds, you can see the project ready in **`Solution Explorer`**, with a set of well-defined files and folders for a **`React.js`** application:

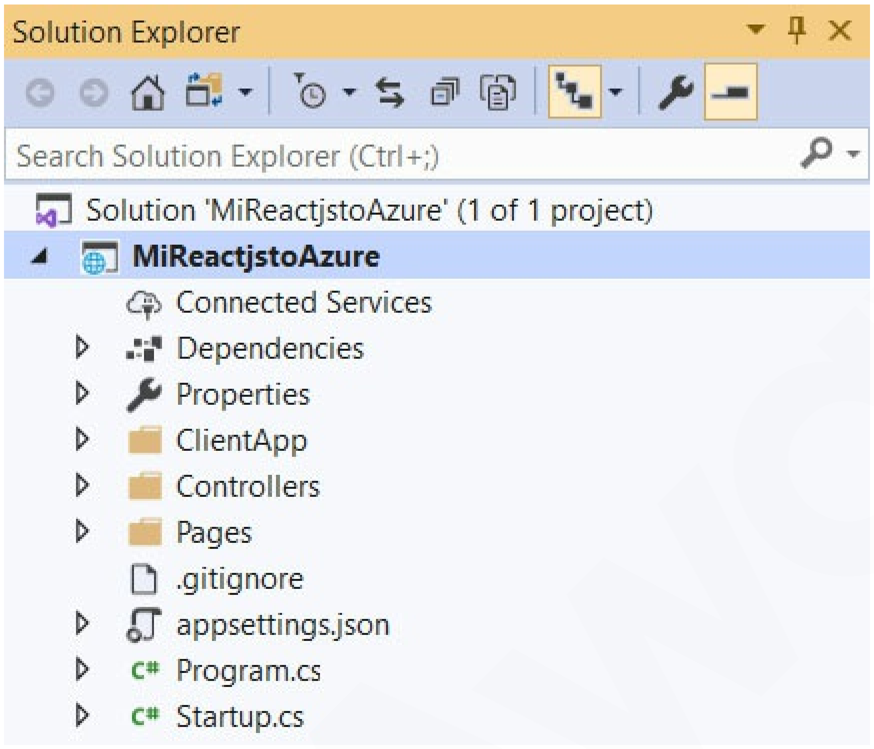
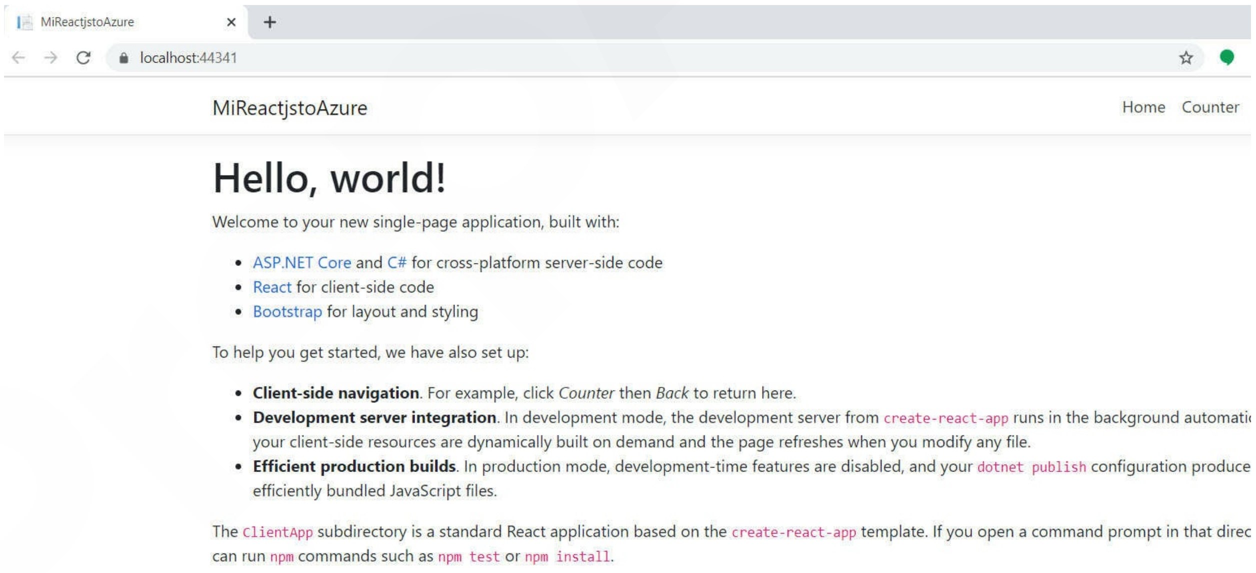


Figure 5.22

Let’s run it:



Hello, world!

Welcome to your new single-page application, built with:

- ASP.NET Core and C# for cross-platform server-side code
- React for client-side code
- Bootstrap for layout and styling

To help you get started, we have also set up:

- **Client-side navigation**. For example, click *Counter* then *Back* to return here.
- **Development server integration**. In development mode, the development server from `create-react-app` runs in the background automati your client-side resources are dynamically built on demand and the page refreshes when you modify any file.
- **Efficient production builds**. In production mode, development-time features are disabled, and your `dotnet publish` configuration produce efficiently bundled JavaScript files.

The `ClientApp` subdirectory is a standard React application based on the `create-react-app` template. If you open a command prompt in that direc can run `npm` commands such as `npm test` or `npm install`.

Figure 5.23

Files to deploy

Similarly, in order to have this application on the cloud, we need to deploy or upload the project files to **`Microsoft Azure Storage Blob`**.

Select **`Project`** in **`Solution explorer | Right Click | Publish.`**

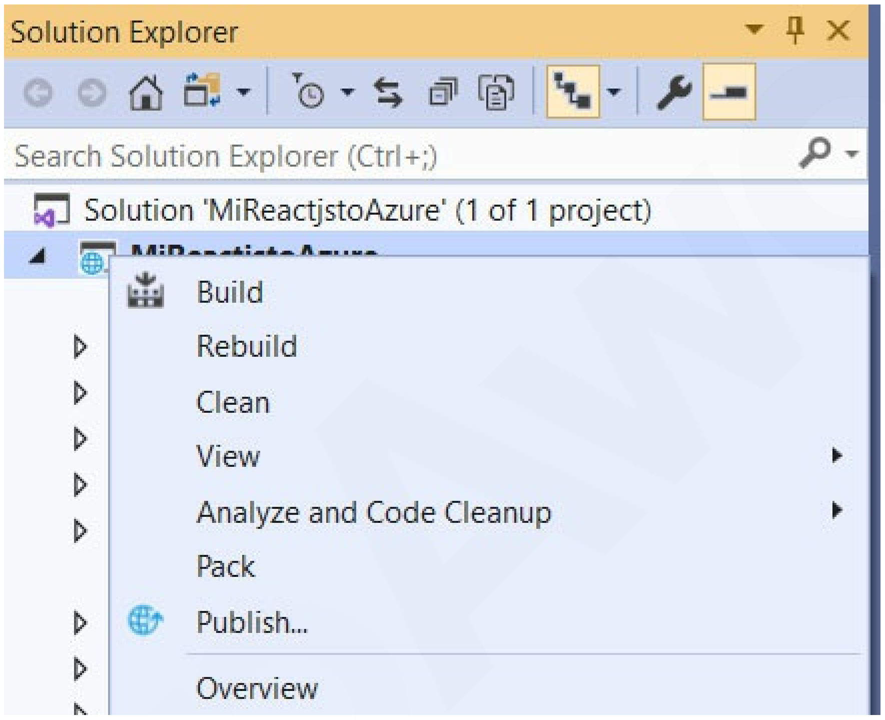
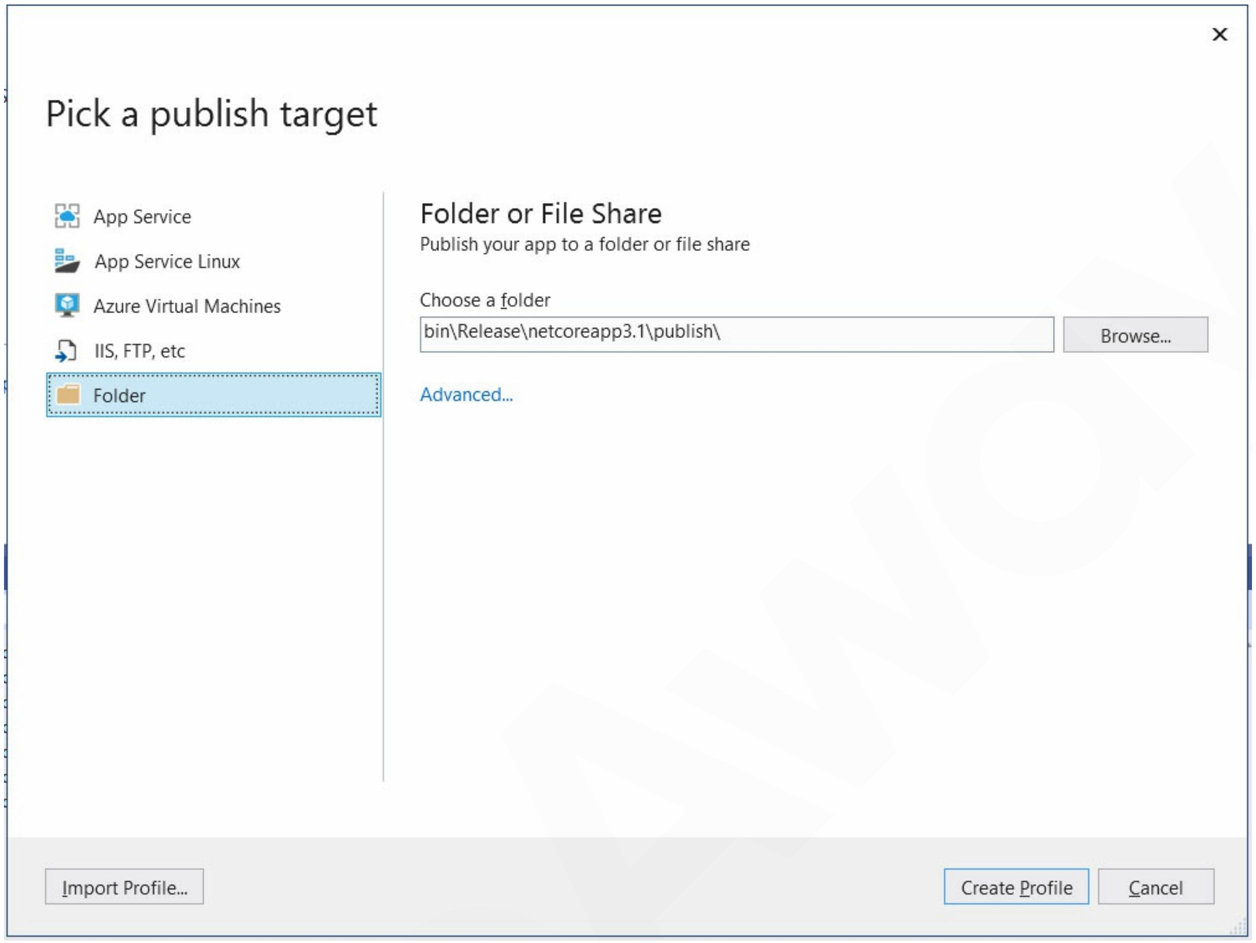


Figure 5.24

This will open a window with all available options.

Hit publish target as **`Folder`** from left hand listed options. You can provide the path you want the application files to get published. We will keep the location as default.

Click on **`Create Profile`**:

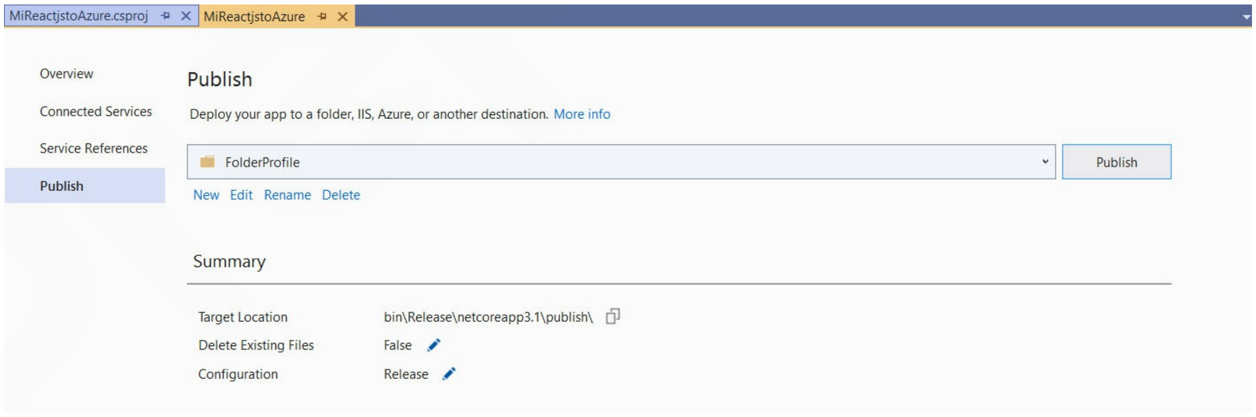

Pick a publish target

Folder or File Share
Publish your app to a folder or file share

Choose a folder: bin\Release\netcoreapp3.1\publish\

Figure 5.25

Once publish target is set, click on **`Publish`** to start with the activity:



Publish

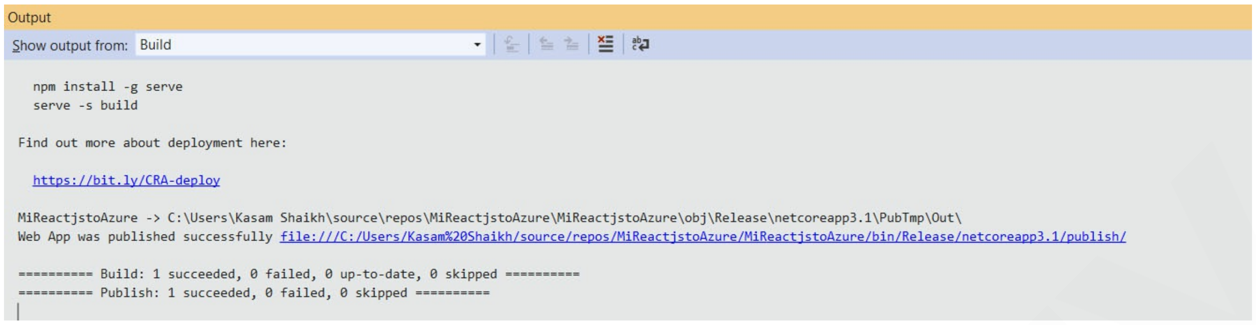
Deploy your app to a folder, IIS, Azure, or another destination. More info

Summary

Target Location	bin\Release\netcoreapp3.1\publish\
Delete Existing Files	False
Configuration	Release

Figure 5.26

Within fewer minutes you can the succeeded message in the output window:


Output
Show output from: Build

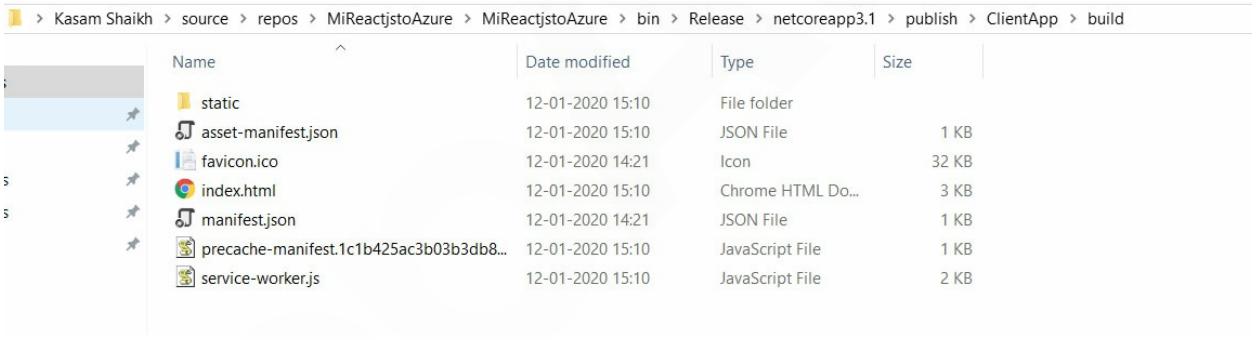
```
  npm install -g serve
  serve -s build

Find out more about deployment here:

  https://bit.ly/CRA-deploy

MiReactjstoAzure -> C:\Users\Kasam Shaikh\source\repos\MiReactjstoAzure\MiReactjstoAzure\obj\Release\netcoreapp3.1\PubTmp\Out\
Web App was published successfully file:///C:/Users/Kasam%20Shaikh/source/repos/MiReactjstoAzure/MiReactjstoAzure/bin/Release/netcoreapp3.1/publish/

========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

Figure 5.27

Now, click the location the files are published, and navigate to **`publish | ClientApp |build folder.`**

Make sure only the files under the build folder should be uploaded to the Microsoft Azure Storage Blob container.

Consider this as the most important step or take away from this article, to which files need to get deployed or uploaded to Azure for coming in as static content:

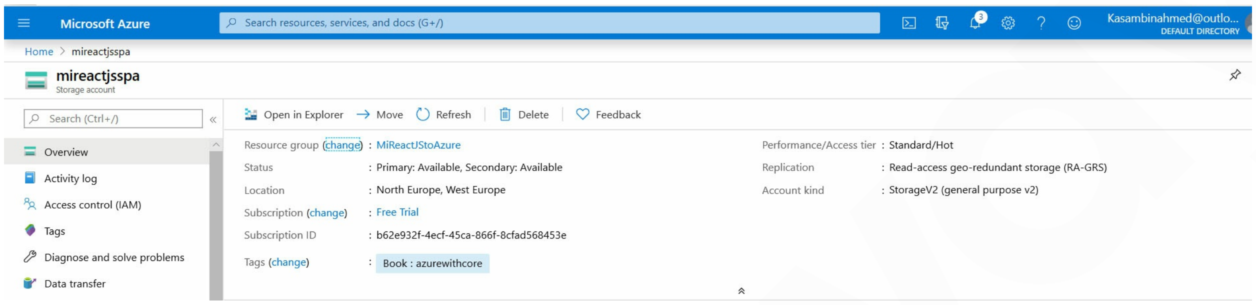

Kasam Shaikh > source > repos > MiReactjstoAzure > MiReactjstoAzure > bin > Release > netcoreapp3.1 > publish > ClientApp > build

Name	Date modified	Type	Size
static	12-01-2020 15:10	File folder	
asset-manifest.json	12-01-2020 15:10	JSON File	1 KB
favicon.ico	12-01-2020 14:21	Icon	32 KB
index.html	12-01-2020 15:10	Chrome HTML Do...	3 KB
manifest.json	12-01-2020 14:21	JSON File	1 KB
precache-manifest.1c1b425ac3b03b3db8...	12-01-2020 15:10	JavaScript File	1 KB
service-worker.js	12-01-2020 15:10	JavaScript File	2 KB

Figure 5.28

Azure setup

To host the preceding application, repeat the preceding said steps and create an Azure Storage **`General Purpose V2 account`**. I will create one, and name it as MiReactJSspa:



Resource group (change) : MiReactJStoAzure
Status : Primary: Available, Secondary: Available
Location : North Europe, West Europe
Subscription (change) : Free Trial
Subscription ID : b62e932f-4ecf-45ca-866f-8cfad568453e
Tags (change) : Book : azurewithcore
Performance/Access tier : Standard/Hot
Replication : Read-access geo-redundant storage (RA-GRS)
Account kind : StorageV2 (general purpose v2)

Figure 5.29

Azure CLI to enable Static website

Earlier in the chapter, I enabled the **`Static website`** feature through the Azure Portal. Now, I will show you how to enable this feature using the Azure command-line interface.

Azure CLI is one of the many available ways to work with Azure resources offered by Microsoft Azure. As an Azure developer, it is always advisable to have your hands dirty with different ways to manage the Azure resources. Till now, you learned how to work with Azure Portal, along with using .NET core SDKs and now will show how to achieve the task.

I will work in **`Azure Cloud Shell`** via Azure Portal.

Open **`Azure Portal`**, and click on the icon in the top menu bar in Azure portal:

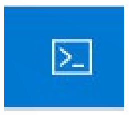
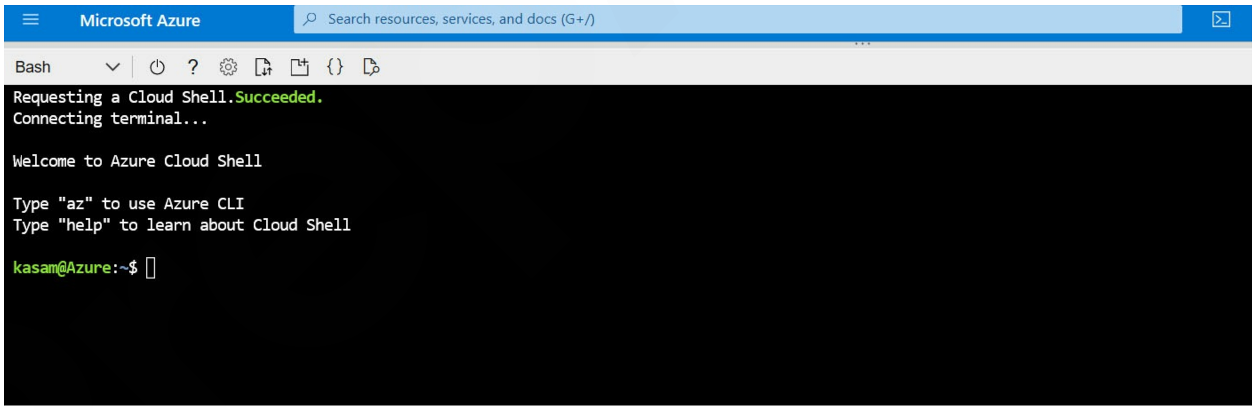


Figure 5.30

To open up the **`Azure Cloud Shell`** window:



```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI
Type "help" to learn about Cloud Shell

kasam@Azure:~$ 
```

Figure 5.31

If you are using for the first time, just follow the instruction. You can use both Bash and **`Powershell`** through this video. I will be using Bash. Now, I have already gone through creating Storage steps required for using **`Azure Cloud shell`**, and hence as seen in the preceding window, it's ready to start with executing the command.

The best command to start with is:

```
az help
```

And it will list all available subgroups, commands, the argument for Azure resources.

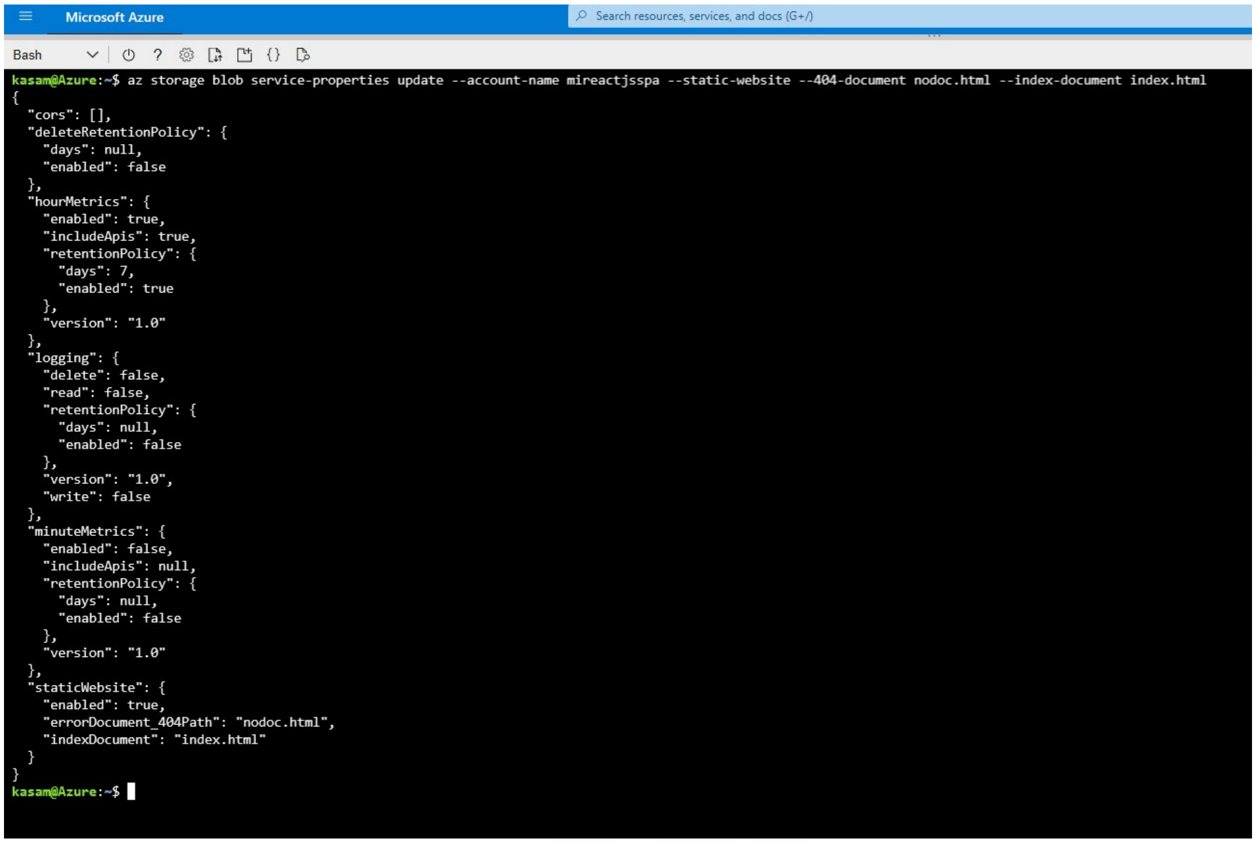
Here you need to enable the Static website feature for Azure Storage account, which you have already created. To do so, run the following command:

```
az storage blob service-properties update --account-name
<storage-account-name> --static-website --404-document <error-
document-name> --index-document <index-document-name>
```

Modifying this further to update Azure Storage account you created, change it to.

```
az storage blob service-properties update --account-name
mireactjsspa --static-website --404-document nodoc.html --index-
document index.html
```

Click **Enter**, and you will get the following output:



Microsoft Azure

```
kasam@Azure:~$ az storage blob service-properties update --account-name mireactjsspa --static-website --404-document nodoc.html --index-document index.html
{
  "cors": [],
  "deleteRetentionPolicy": {
    "days": null,
    "enabled": false
  },
  "hourMetrics": {
    "enabled": true,
    "includeApis": true,
    "retentionPolicy": {
      "days": 7,
      "enabled": true
    },
    "version": "1.0"
  },
  "logging": {
    "delete": false,
    "read": false,
    "retentionPolicy": {
      "days": null,
      "enabled": false
    },
    "version": "1.0",
    "write": false
  },
  "minuteMetrics": {
    "enabled": false,
    "includeApis": null,
    "retentionPolicy": {
      "days": null,
      "enabled": false
    },
    "version": "1.0"
  },
  "staticWebsite": {
    "enabled": true,
    "errorDocument_404Path": "nodoc.html",
    "indexDocument": "index.html"
  }
}
kasam@Azure:~$
```

Figure 5.32

Notice, preceding image states, **`Static website`**, enabled as true.

After enabling the feature, it's time to retrieve the Website URL, that is, the primary endpoint.

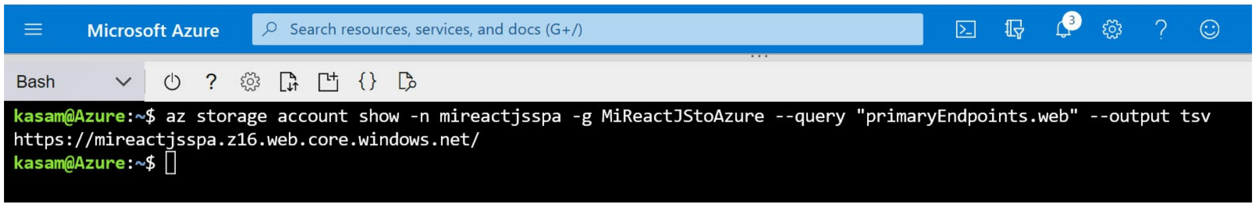
Following command is used to achieve the same:

```
az storage account show -n <storage-account-name> -g <resource-group-name> --query "primaryEndpoints.web" --output tsv
```

Changing it to:

```
az storage account show -n mireactjsspa -g MiReactJStoAzure --query "primaryEndpoints.web" --output tsv
```

And following is the output:



Microsoft Azure

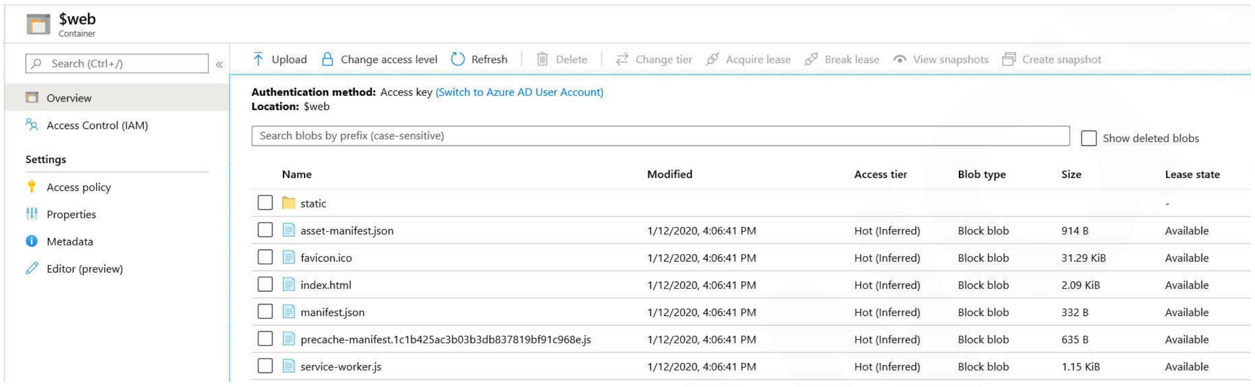
```
kasam@Azure:~$ az storage account show -n mireactjsspa -g MiReactJStoAzure --query "primaryEndpoints.web" --output tsv
https://mireactjsspa.z16.web.core.windows.net/
kasam@Azure:~$
```

Figure 5.33

Copy and save the URL.

Uploading published files

I will upload all the files using the Azure Portal. Note, uploading files too can be achieved using Azure CLI. If time permits, we will have an entire chapter on working with Azure CLI:

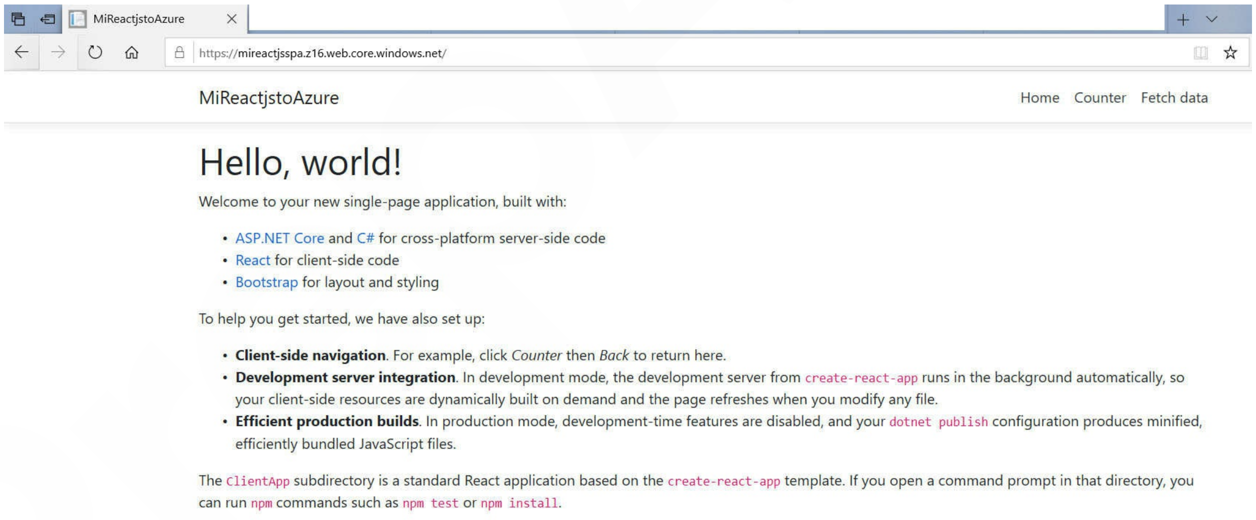


Authentication method: Access key (Switch to Azure AD User Account)
Location: $web

Name	Modified	Access tier	Blob type	Size	Lease state
static					-
asset-manifest.json	1/12/2020, 4:06:41 PM	Hot (Inferred)	Block blob	914 B	Available
favicon.ico	1/12/2020, 4:06:41 PM	Hot (Inferred)	Block blob	31.29 KiB	Available
index.html	1/12/2020, 4:06:41 PM	Hot (Inferred)	Block blob	2.09 KiB	Available
manifest.json	1/12/2020, 4:06:41 PM	Hot (Inferred)	Block blob	332 B	Available
precache-manifest.1c1b425ac3b03b3db837819bf91c968e.js	1/12/2020, 4:06:41 PM	Hot (Inferred)	Block blob	635 B	Available
service-worker.js	1/12/2020, 4:06:41 PM	Hot (Inferred)	Block blob	1.15 KiB	Available

Figure 5.34

After uploading all the required files for published **`Location`**, its time to open the browser with copied primary endpoint we retrieve via Azure CLI.



Hello, world!

Welcome to your new single-page application, built with:

- ASP.NET Core and C# for cross-platform server-side code
- React for client-side code
- Bootstrap for layout and styling

To help you get started, we have also set up:

- **Client-side navigation**. For example, click *Counter* then *Back* to return here.
- **Development server integration**. In development mode, the development server from `create-react-app` runs in the background automatically, so your client-side resources are dynamically built on demand and the page refreshes when you modify any file.
- **Efficient production builds**. In production mode, development-time features are disabled, and your `dotnet publish` configuration produces minified, efficiently bundled JavaScript files.

The `ClientApp` subdirectory is a standard React application based on the `create-react-app` template. If you open a command prompt in that directory, you can run `npm` commands such as `npm test` or `npm install`.

Figure 5.35

And we have our ASP.NET Core 3.1 ReactJS application live on Azure Cloud.

Note: Fetch Data menu in the app seen above, will not work, as it has server side calling.

You can repeat the same exercise for hosting an ASP.NET core 3.1 applications with `React.js` and Redux. Have it as your homework. As a hint, the file location I copied for the ReactJS application will be the same for ReactJS and Redux.

To end with, you can also host a **`Blazor Client`** app with ASP.NET Core 3.1 to Azure Storage account, try it out.

Conclusion

From the chapter, I hope you have learned something new, rather the easiest way to host your application with static content on public cloud Microsoft Azure. The best part is, this feature comes with no charge, expect the storage consumption charges. I would advise you to have this exercise at your end. In the next chapter, we will look more into security aspects when an ASP.NET Core application talks with the Azure ecosystem.

Happy Azure learning.

Questions

1. What is the Storage account type that offers hosting option for web applications?
2. Can be server-side application be hosted in Azure Storage?
3. Explain pricing for hosting a static website in Azure Storage?
4. What are the different ways to enable the Static Website feature in Azure Storage account?
5. When you enable the Static feature, under which Blob container, you should deploy the files?
6. Can you have a custom domain name enabled to Static website hosted in Azure Storage?

CHAPTER 6

Security Application Secrets Keys With Azure

In the previous chapter, we learned about using the Microsoft Azure Storage account as an amazing option to host your static content. We also learn about hosting .NET Core 3.1 Angular and React.JS single page application to public cloud Azure again using Azure Storage. Interestingly, we went through how to manage the storage accounts using the Azure Command Line interface. In this chapter, we will learn how to seamlessly secure your .NET core application secrets keys using Azure Key Vault and App service configuration.

Structure

- Create an Azure key vault
- Adding secret to Azure key vault
- Deploying the application to Azure Web App
- Manage service identity
- Key Vault access policies to Web App
- App service configuration
- .NET Core Application Code part

Objective

The takeaway from this chapter is:

- Working with Azure KeyVault
- Enabling .NET Core application to talk with Azure KeyVault
- How the Keys are managed in applications
- Working with App service Configuration
- How the Keys are managed in applications using App service Configuration

What all need to be in place

Secure key management is essential to protect data in the cloud. All keys use in an application hosted on the cloud should have to be securely communicated. What if you don’t write any password in application and still use the same through the application? Sounds interesting, isn’t it? The following exercise will achieve the same.

Azure Key Vault is a cloud service offered by Microsoft Azure, especially used to manage keys, secrets, and certificates. Key Vault eradicates the need for developers to store security information in their code. It also allows you to centralize the storage of your application secrets, which greatly reduces the chances that secrets may be leaked. It does come with secrets access, and usage logs help in further audits.

I will create an Azure Key Vault and have keys stored in it. We will have the core application to read the key from Azure Key Vault, without writing the password in the application. With few lines of code, this can be easily achieved.

Here I will start with having the steps required for achieving the objective, and during the exercise, we will explore the Azure services and features involved in the course. Exercise would be interesting, as you will be learning about many widely used terms such asManage Service Identity, Service Accounts, Least access privilege, and so on.

The agenda will be:

- To create the Azure Key Vault.
- Add Secret to Azure Key Vault.
- Create a Web App Service to host the MVC Core application
- Allow Web App to access secrets to Azure Key Vault.
- Create an ASP.NET Core MVC application.
- Add Code to the application to talk with Azure Key Vault.
- Test the application 😊

Create an Azure Key Vault

In this chapter, I will be using the Azure portal to create and manage the azure resources.

Open up **`Azure Portal | Click on Create Resource | Search for Azure Key Vault| Click Create to proceed:`**

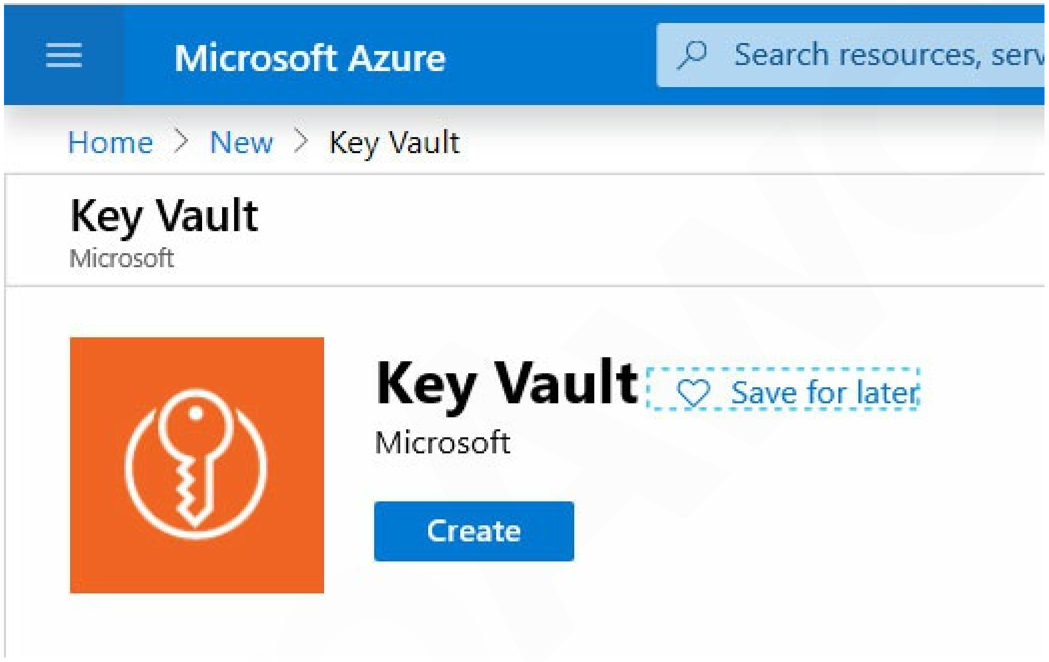


Figure 6.1

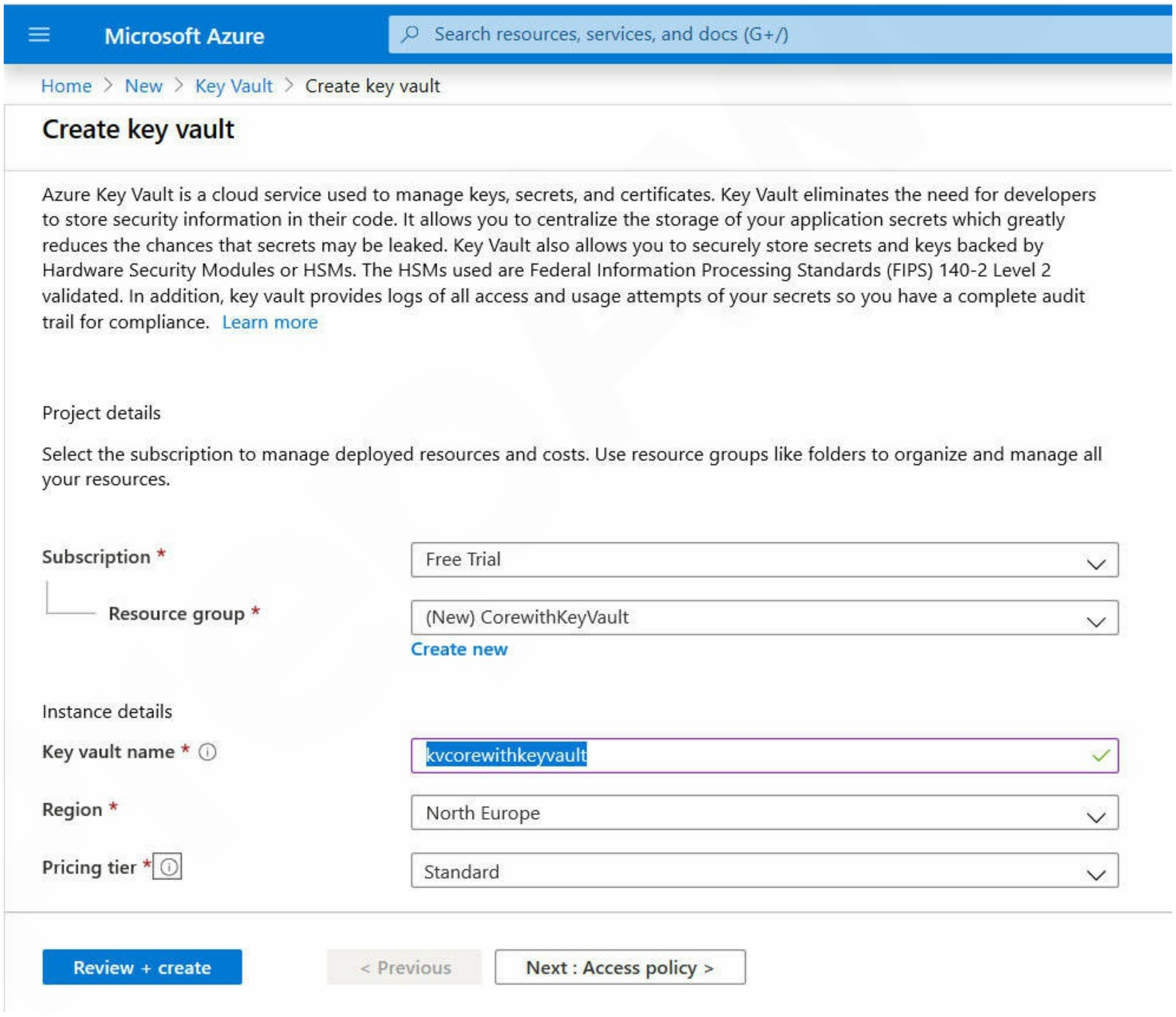
It presents with three main areas to configure, **`Basics, Access Policy, Networking`**, followed by **`Tags`**, and validating the configuration to proceed with key vault account creation.

Following details comes for **`Basic`** configurations:

- **`Subscription:`** Here, you need to select your subscription. I have selected my free trails account.
- **`Resource group:`** It's a logical grouping of your Azure resources. You can select from an existing resource group or can create a new resource group, as per your choice. For this article, I have created a new resource group name, `CorewithKeyVault`.

And next comes the instance details:

- **`Key vault name:`** Vault name must only contain alphanumeric characters and dashes and cannot start with a number. It's also part of the DNS name, so be careful in naming. For this exercise will name it as `kvcorewithKeyVault`.
- **`Region:`** The region's key vault resource will be deployed. Decide as per the other resources and your expected target users. We will go with Europe, as a region.
- **`Pricing Tier:`** It comes with two options, Standard and Premium tier. Premium comes with a key protected by the HSM - Hardware Security Module. I will go with the Standard tier.


Microsoft Azure
Search resources, services, and docs (G+/)
Home > New > Key Vault > Create key vault

Create key vault

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance. Learn more

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *: Free Trial

Resource group *: (New) CorewithKeyVault

Create new

Instance details

Key vault name *: kvcorewithkeyvault

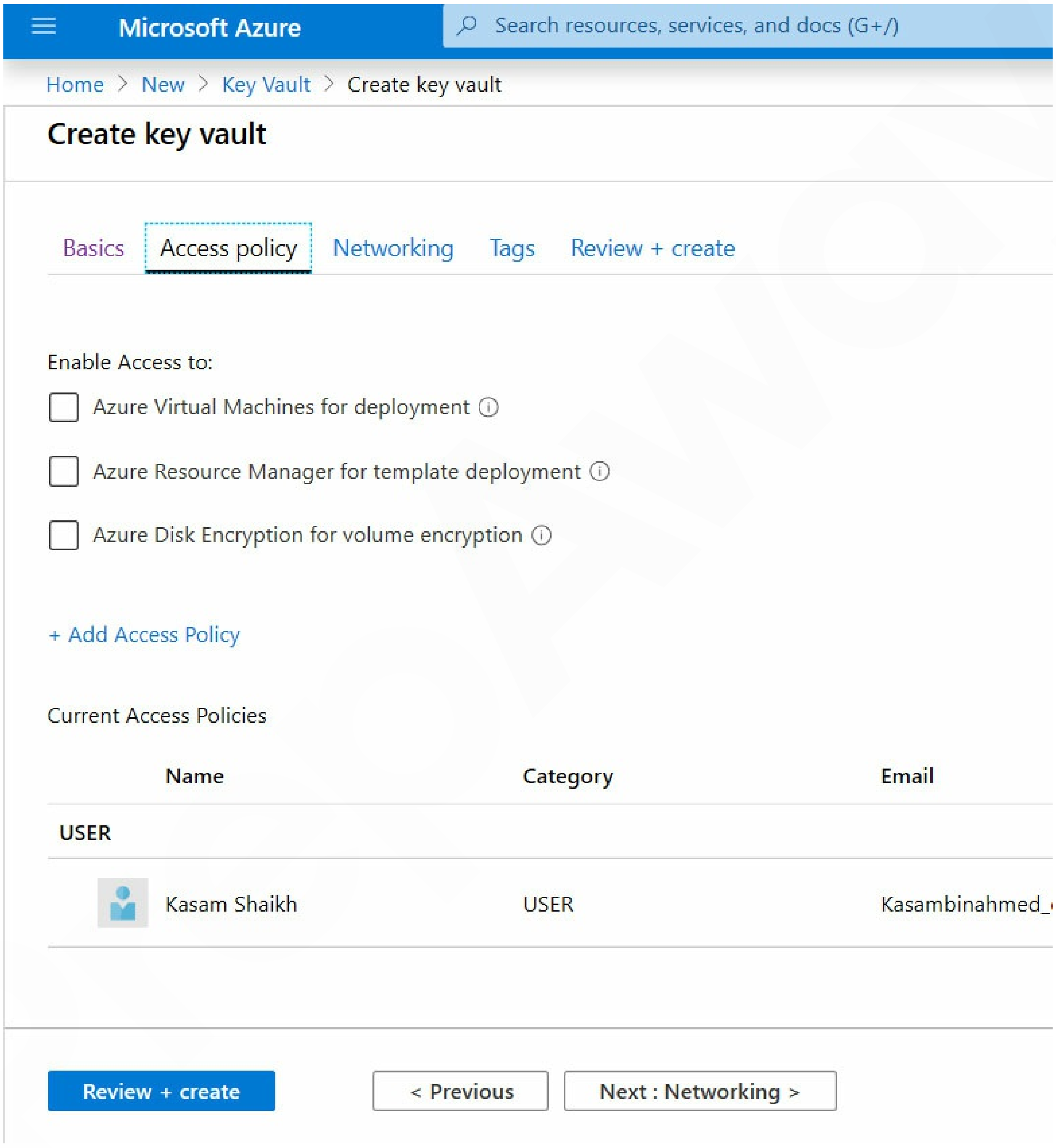
Region *: North Europe

Pricing tier *: Standard

Figure 6.2

Next comes the **`Access policy`**.

As I am the owner of the resource, our name is listed by default to have the access. I will configure the web app here to have the access, once it's created. So for now, leave it as is.



Enable Access to:

- [ ] Azure Virtual Machines for deployment
- [ ] Azure Resource Manager for template deployment
- [ ] Azure Disk Encryption for volume encryption

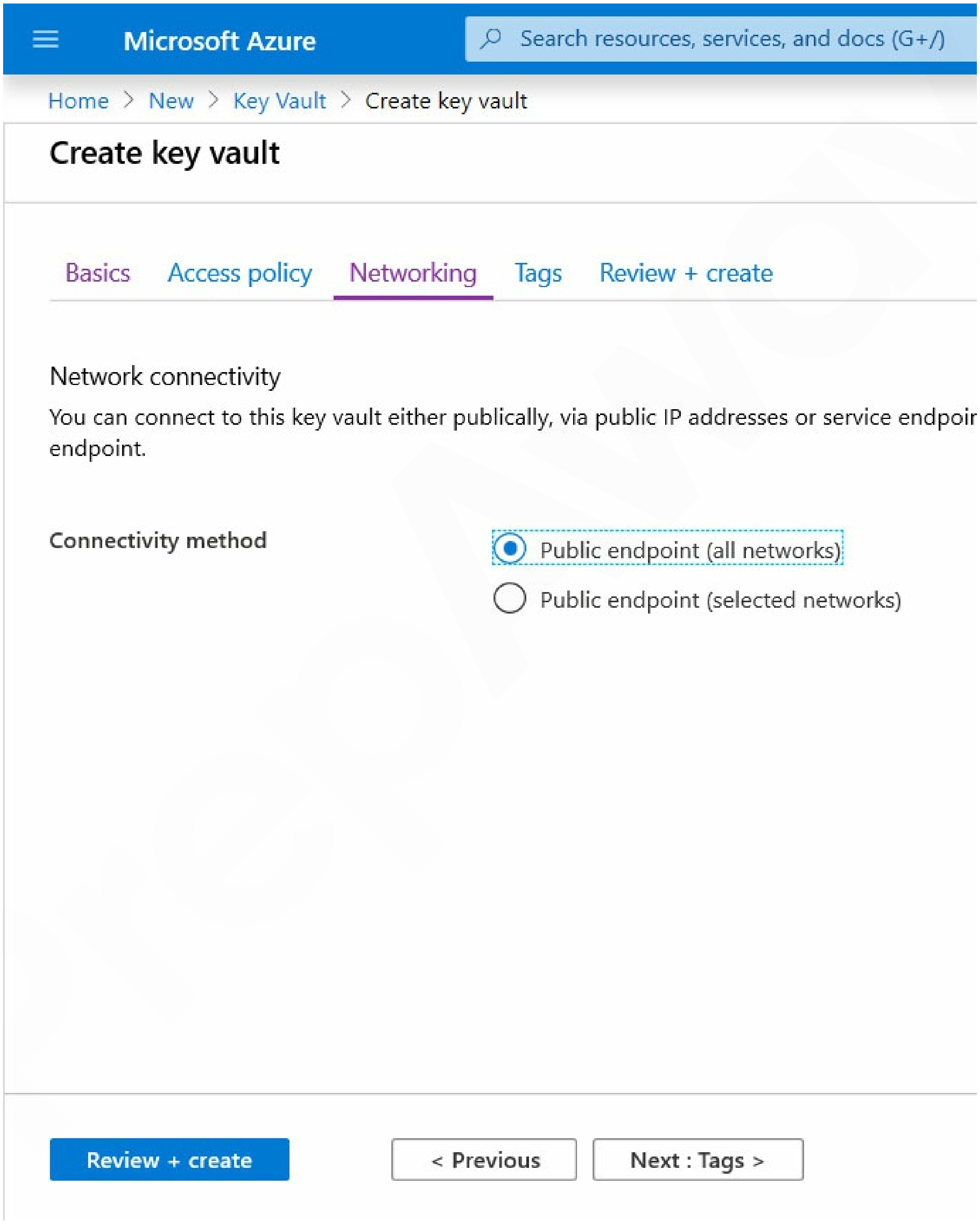
Current Access Policies

Name	Category	Email
USER		
Kasam Shaikh	USER	Kasambinahmed_

Figure 6.3

Click on **`Networking`** to proceed.

Here you can allow it's as to have a public endpoint or private endpoint with configuring the virtual network for allowing selected network. Let's go with default:



Microsoft Azure

Search resources, services, and docs (G+/)

Home > New > Key Vault > Create key vault

Create key vault

Basics | Access policy | Networking | Tags | Review + create

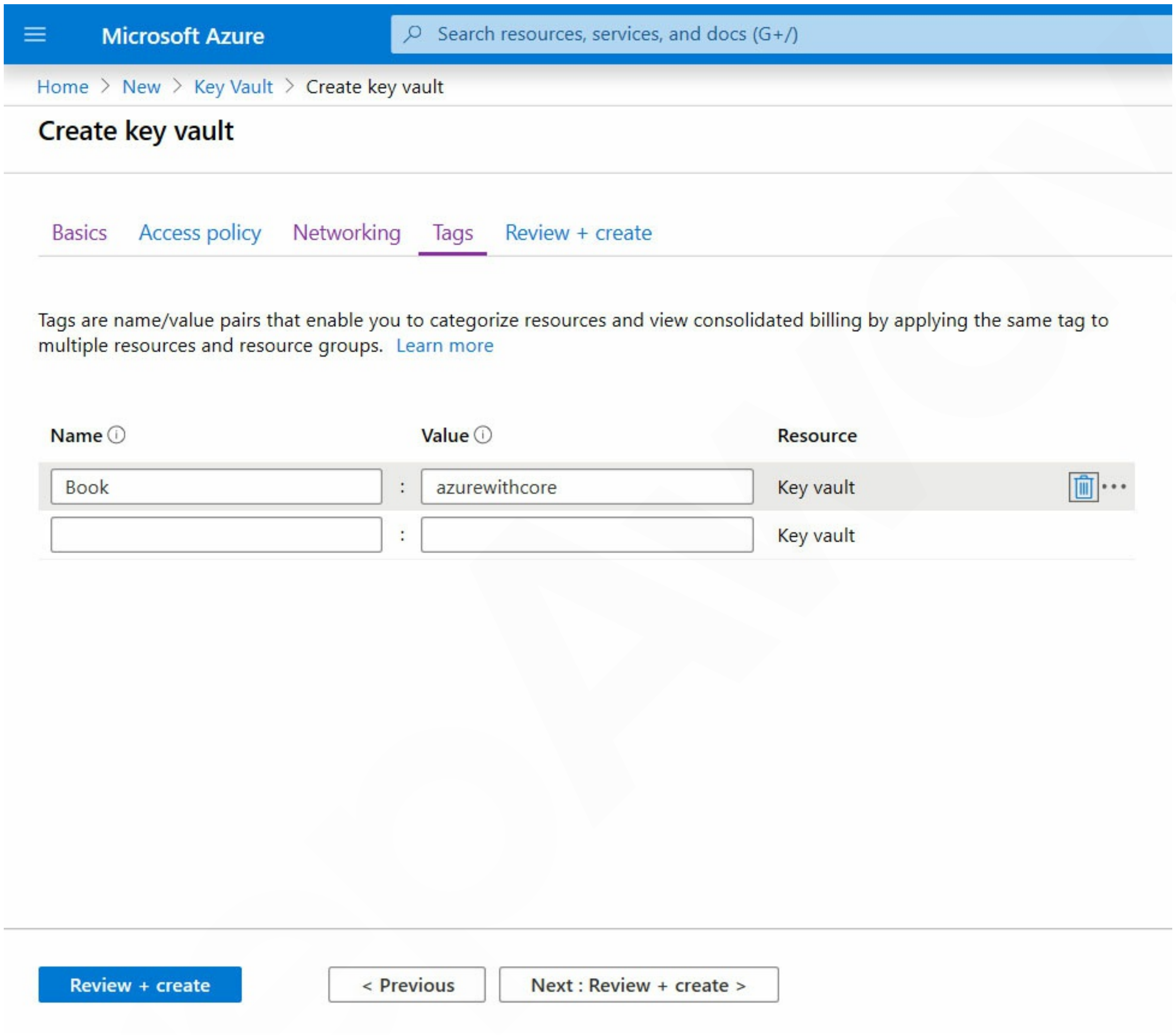
Network connectivity

You can connect to this key vault either publically, via public IP addresses or service endpoir
endpoint.

Connectivity method: [✓ Public endpoint (all networks)] Public endpoint (selected networks)

Figure 6.4

Next comes the **`Tags`** and will go with the same as used in the earlier chapter.



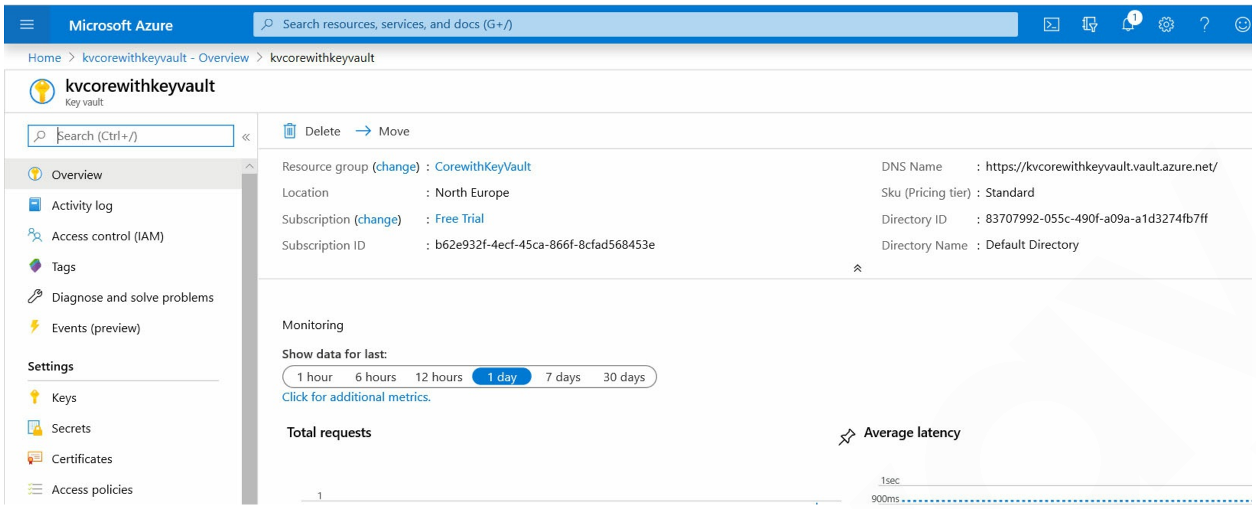
Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. Learn more

Name		Value	Resource
Book	:	azurewithcore	Key vault
	:		Key vault

Figure 6.5

Click on **`Review + Create`** to validate the configuration, and once it’s passed, click on **`Create`** to proceed.

After fewer minutes, Azure key vault account will be created:



Resource group (change) : CorewithKeyVault
Location : North Europe
Subscription (change) : Free Trial
Subscription ID : b62e932f-4ecf-45ca-866f-8cfad568453e
DNS Name : https://kvcorewithkeyvault.vault.azure.net/
Sku (Pricing tier) : Standard
Directory ID : 83707992-055c-490f-a09a-a1d3274fb7ff
Directory Name : Default Directory

Figure 6.6

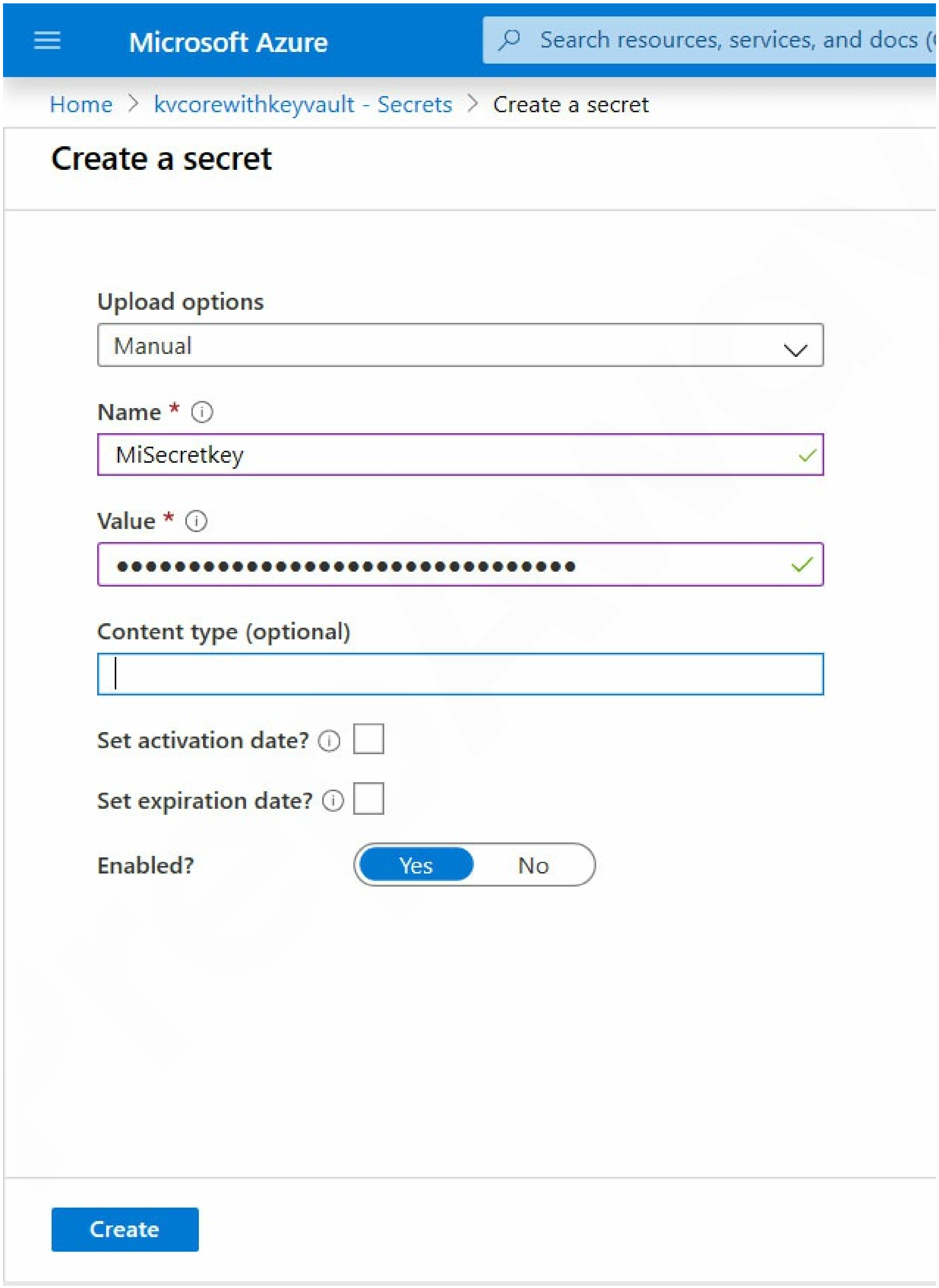
Adding Secret to Azure Key Vault

You can add your important keys in the key vault, and later it can be read from the application. To add the secrets, click from right-hand pane under **Settings | Secrets**|click on **`Generate / Import.`**

You will be presented by the blade with the following options:

- **`Upload options:`** With two options, **`Manual`** and **`Certificates`**. I will go with Manual.
- **`Name:`** Need to provide a valid secret name. Secret names can only contain alphanumeric characters and dashes. I will have it as `MiSecretkey`.
- **`Value:`** The Azure Portal currently only supports single-line secret values. Multi-lines can be added via PowerShell. I have added as **`I am coming from Azure Key Vault!`**.

`Name` and **`Value`** are mandatory, and others come as optional.



Microsoft Azure

Search resources, services, and docs

Home > kvcorewithkeyvault - Secrets > Create a secret

Create a secret

Upload options: Manual

Name *: MiSecretkey

Value *: ••••••••••••••••••••••••••••••••

Content type (optional):

Set activation date? ☐

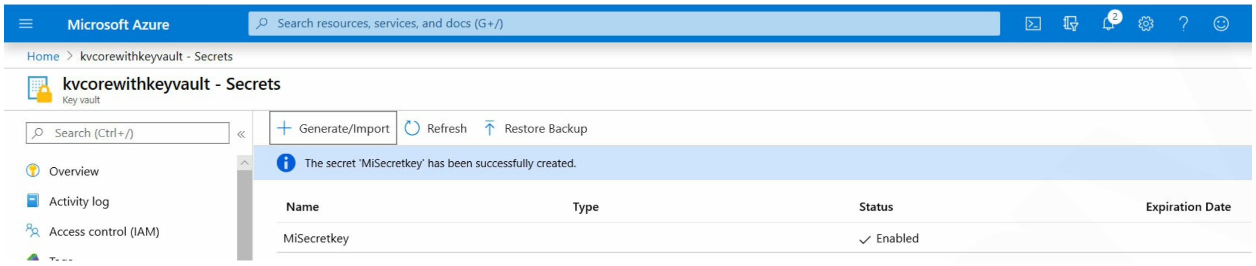
Set expiration date? ☐

Enabled? [✓ Yes] No

Create

Figure 6.7

Click on **`Create`**.



Microsoft Azure

Search resources, services, and docs (G+/)

Home > kvcorewithkeyvault - Secrets

kvcorewithkeyvault - Secrets

Key vault

Search (Ctrl+/)

Overview

Activity log

Access control (IAM)

+ Generate/Import   Refresh   Restore Backup

The secret 'MiSecretkey' has been successfully created.

Name	Type	Status	Expiration Date
MiSecretkey		✓ Enabled	

Figure 6.8

Create a .NET Core application

I have downloaded the code from GitHub, Azure samples, for the exercise.

Download the code and open up in Visual Studio 2019. The project name is the same name you will find it in GitHub. It comes with all required packages, SDKs, the app required to talk with Azure key vault resource:

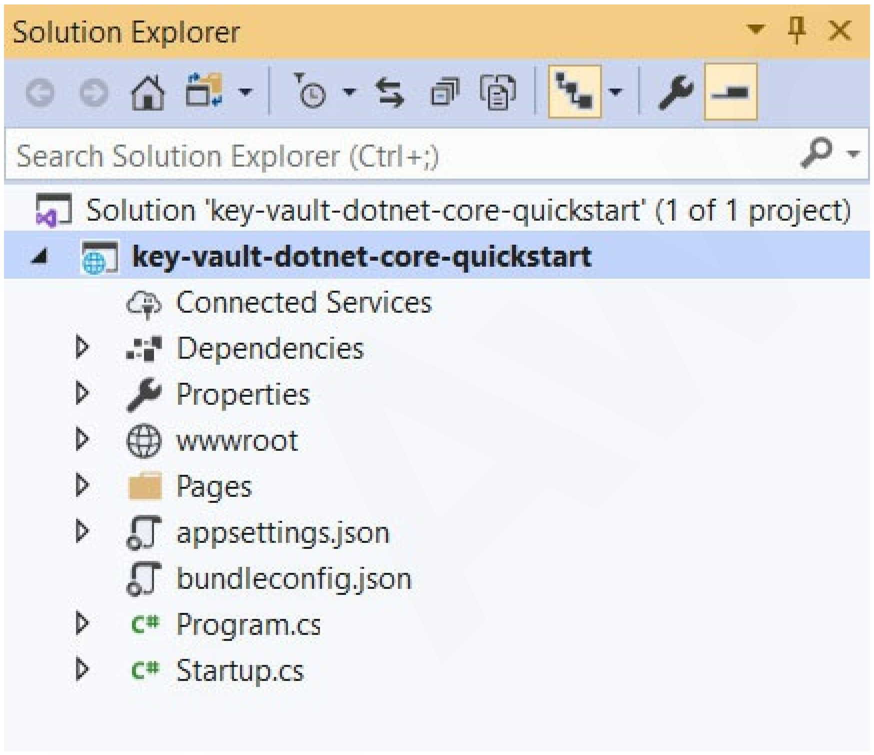
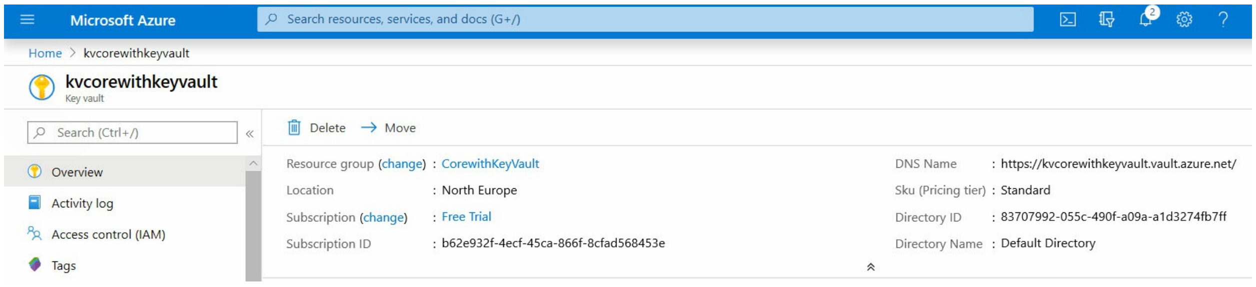

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'key-vault-dotnet-core-quickstart' (1 of 1 project)
key-vault-dotnet-core-quickstart
Connected Services
Dependencies
Properties
wwwroot
Pages
appsettings.json
bundleconfig.json
Program.cs
Startup.cs

Figure 6.9

Open up **`Program.cs`** page.

Copy the Azure key vault DNS Name, from the overview section of the Azure key vault account resource page.

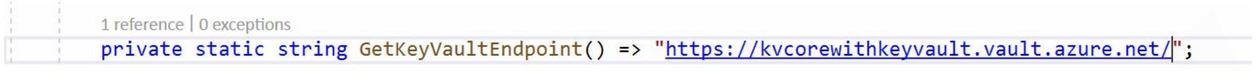
In our case DNS Name: **https://kvcorewithKeyVault.vault.azure.net/**



Resource group (change)	: CorewithKeyVault
Location	: North Europe
Subscription (change)	: Free Trial
Subscription ID	: b62e932f-4ecf-45ca-866f-8cfad568453e
DNS Name	: https://kvcorewithkeyvault.vault.azure.net/
Sku (Pricing tier)	: Standard
Directory ID	: 83707992-055c-490f-a09a-a1d3274fb7ff
Directory Name	: Default Directory

Figure 6.10

Add the DNS name, to private static string name `GetKeyVaultEndpoint`.

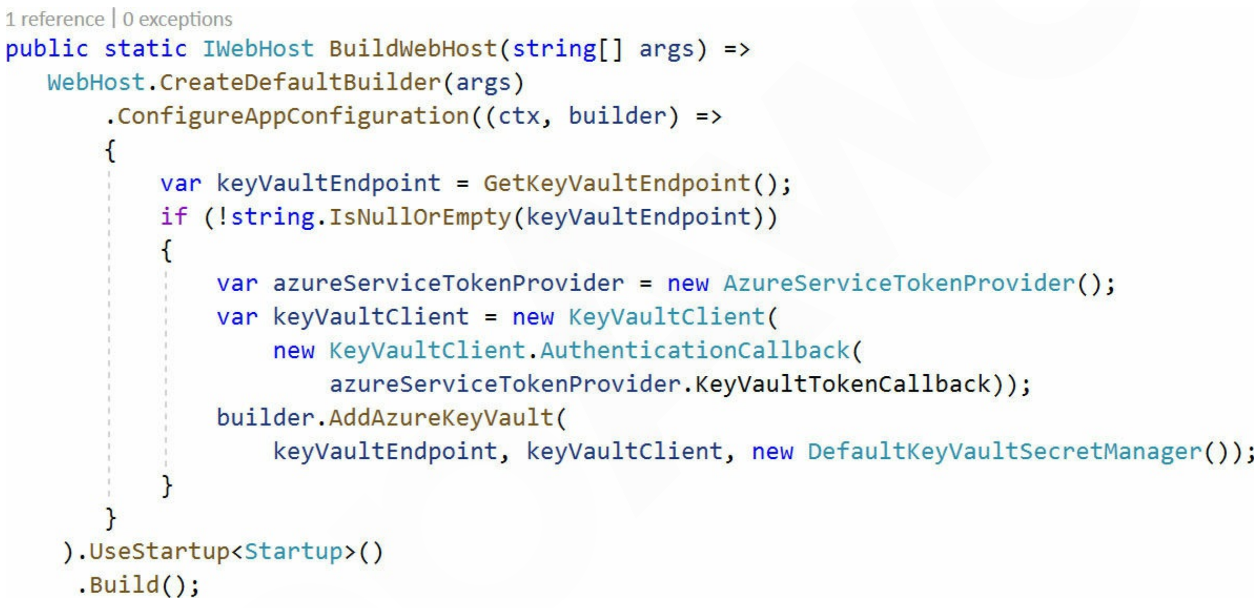


```
1 reference | 0 exceptions
private static string GetKeyVaultEndpoint() => "https://kvcorewithkeyvault.vault.azure.net/";
```

Figure 6.11

The key vault client is created, with service token and `Authenticationcallback`, and this DNS name is passed for final configuration.

Following lines of code, does the work:

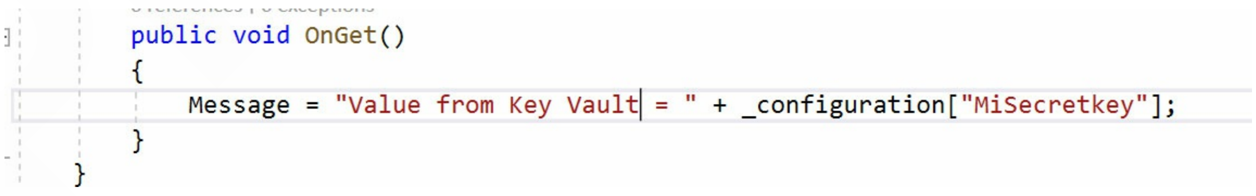


```
1 reference | 0 exceptions
public static IWebHost BuildWebHost(string[] args) =>
   WebHost.CreateDefaultBuilder(args)
       .ConfigureAppConfiguration((ctx, builder) =>
       {
           var keyVaultEndpoint = GetKeyVaultEndpoint();
           if (!string.IsNullOrEmpty(keyVaultEndpoint))
           {
               var azureServiceTokenProvider = new AzureServiceTokenProvider();
               var keyVaultClient = new KeyVaultClient(
                   new KeyVaultClient.AuthenticationCallback(
                       azureServiceTokenProvider.KeyVaultTokenCallback));
               builder.AddAzureKeyVault(
                   keyVaultEndpoint, keyVaultClient, new DefaultKeyVaultSecretManager());
           }
       }
    ).UseStartup<Startup>()
     .Build();
```

Figure 6.12

Navigate to **`About.cshtml.cs`** page for the retrieval code. This is where the sample code is added.

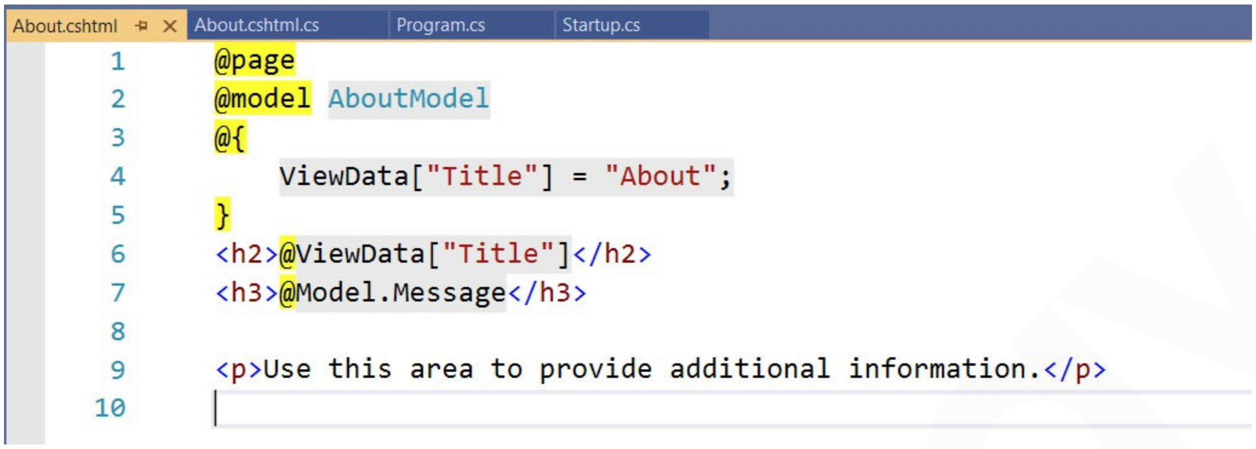
Add the secret key name you provided to fetch the key value.



```
public void OnGet()
{
    Message = "Value from Key Vault = " + _configuration["MiSecretkey"];
}
}
```

Figure 6.13

And can be seen in **`About`** page as:



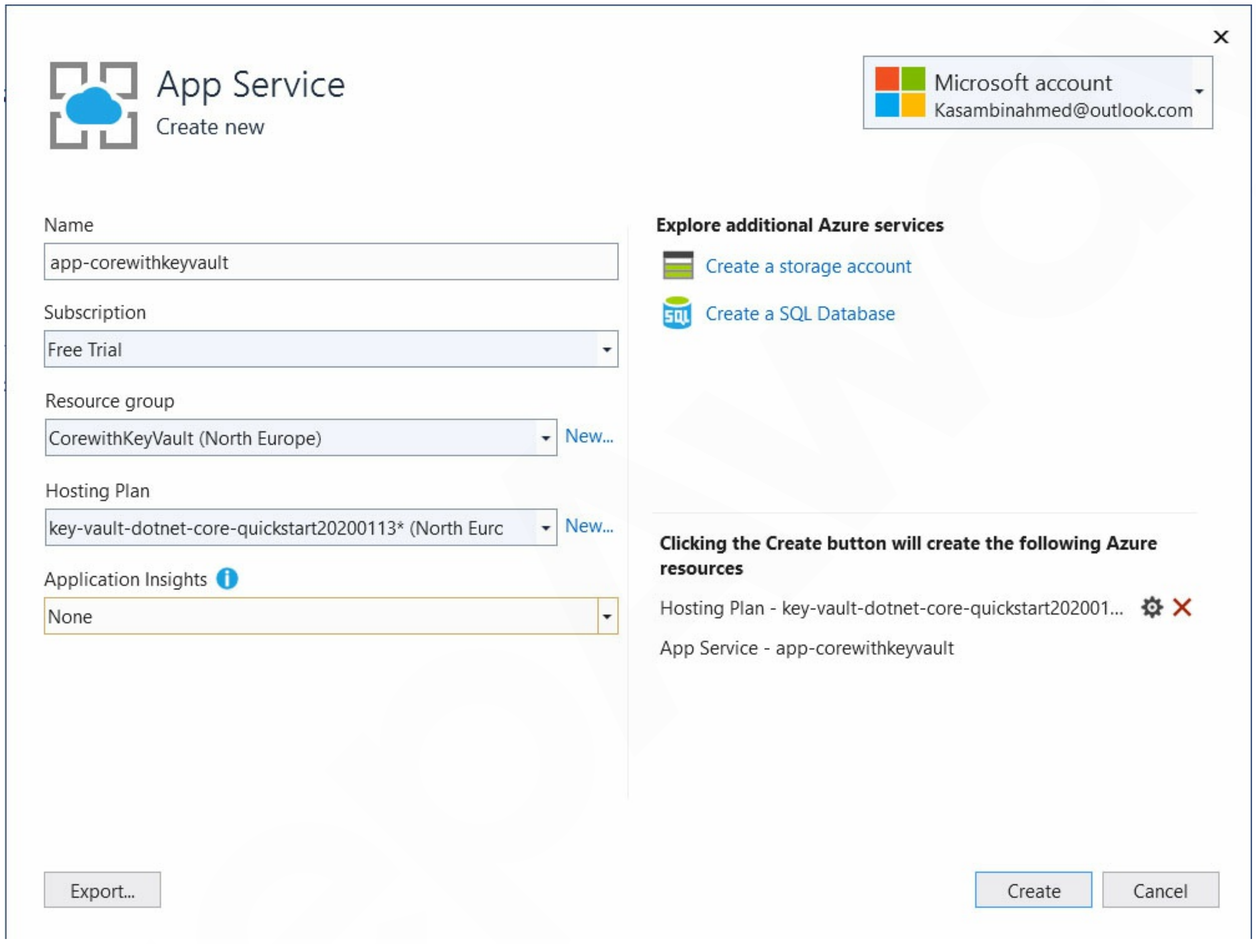
```
@page
@model AboutModel
@{
    ViewData["Title"] = "About";
}
<h2>@ViewData["Title"]</h2>
<h3>@Model.Message</h3>

<p>Use this area to provide additional information.</p>

```

Figure 6.14

Deploying the application to Azure Web App

Right-click on the project in **`Solution Explorer`**, and click **`Publish`**.

Provide the required details and click on **`Create | Publish`**.


App Service
Create new

Microsoft account
Kasambinahmed@outlook.com

Name: app-corewithkeyvault

Subscription: Free Trial

Resource group: CorewithKeyVault (North Europe)

Hosting Plan: key-vault-dotnet-core-quickstart20200113* (North Eurc

Application Insights: None

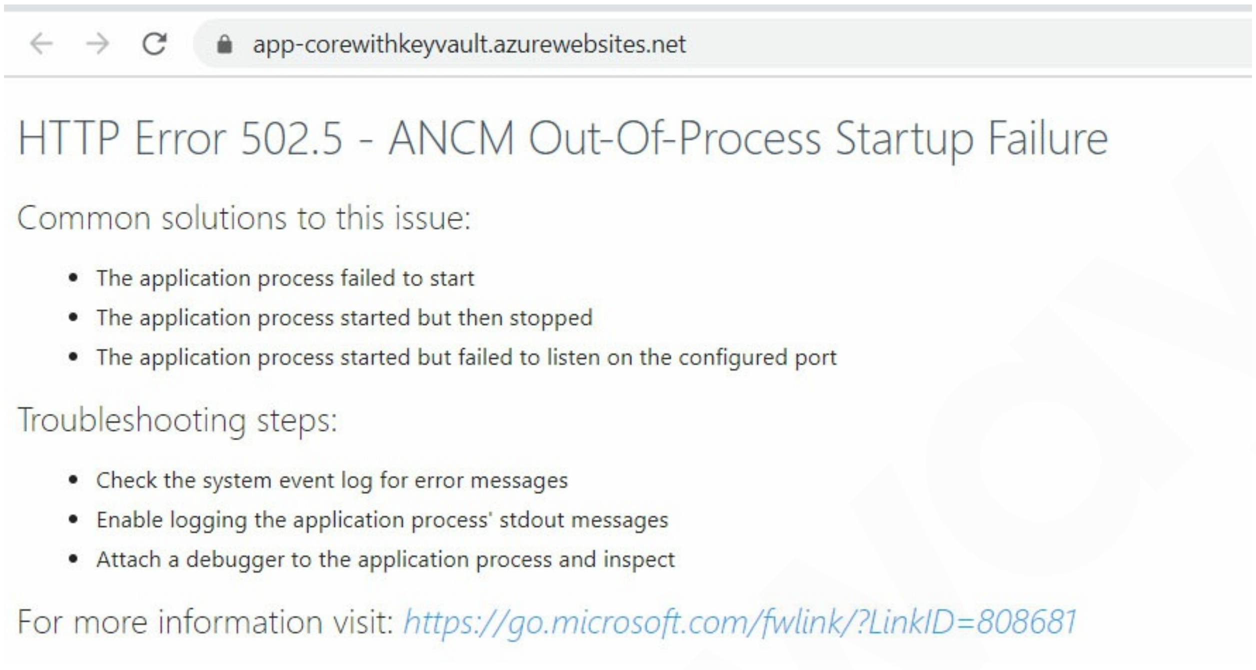
Clicking the Create button will create the following Azure resources

Hosting Plan - key-vault-dotnet-core-quickstart202001...

App Service - app-corewithkeyvault

Figure 6.15

Once the web app is created and publish code is deployed, open up the web app URL in the browser, and you will notice an error.



app-corewithkeyvault.azurewebsites.net

HTTP Error 502.5 - ANCM Out-Of-Process Startup Failure

Common solutions to this issue:

- The application process failed to start
- The application process started but then stopped
- The application process started but failed to listen on the configured port

Troubleshooting steps:

- Check the system event log for error messages
- Enable logging the application process' stdout messages
- Attach a debugger to the application process and inspect

For more information visit: https://go.microsoft.com/fwlink/?LinkID=808681

Figure 6.16

This error was expected, as you are yet to perform two more settings to the deployed resource. We have an entire chapter on the Web App, and hence I covered the app creation in short.

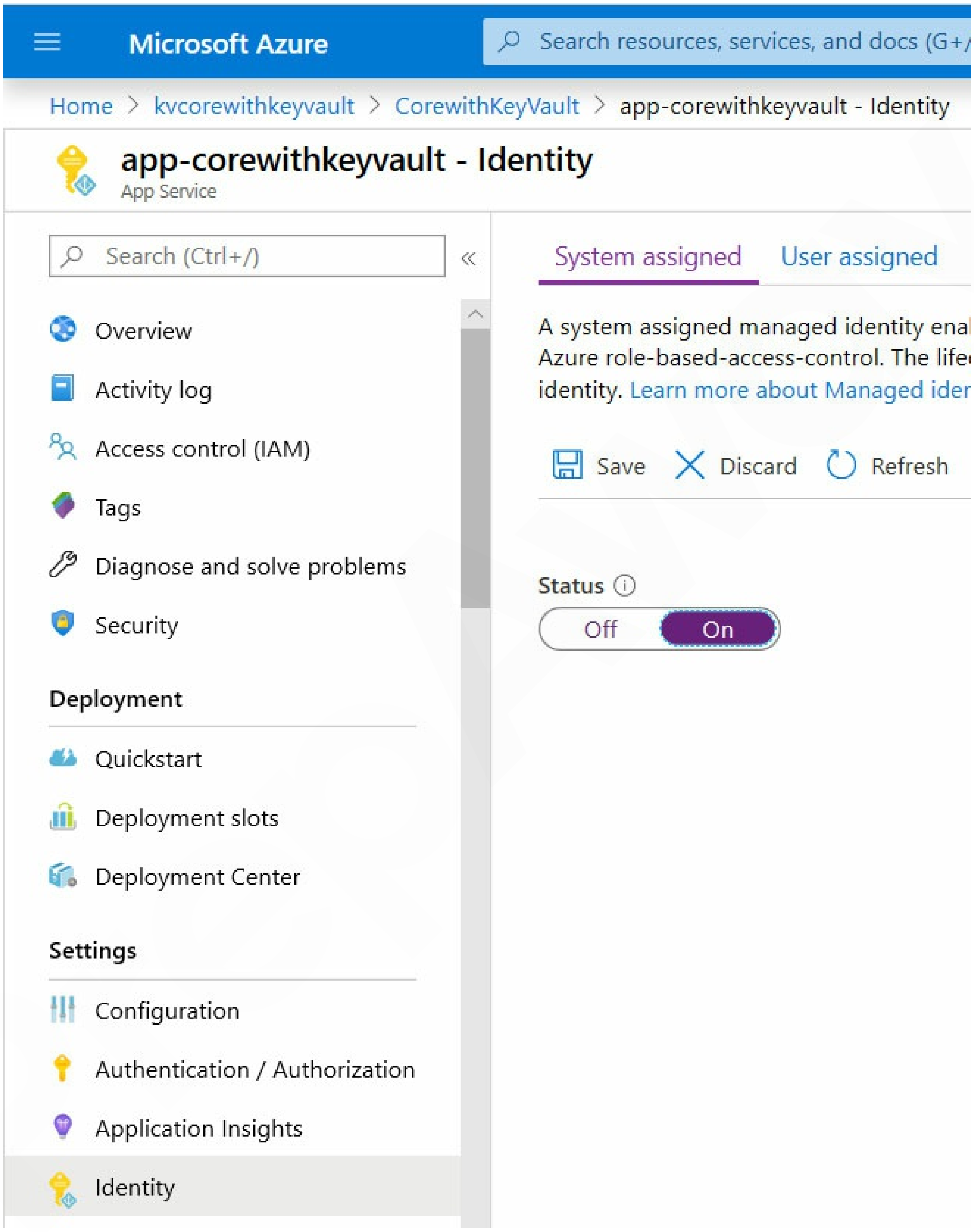
Manage service identity

To add a web app to Azure Key vault access, you need to have the web app added to Azure Active directory.

Navigate to Web App in the portal, and from the blade, under **`Settings`**|click on **`Identity`**.

A system assigned managed identity enables Azure resources to authenticate to cloud services, your case **`Azure Key Vault`**, without storing credentials in code. Once enabled, all necessary permissions can be granted via Azure role-based-access-control. The lifecycle of this type of managed identity is tied to the lifecycle of this resource. Here make the System assigned status as **`On`**, to enable the same.

And click **`Save`**.



A system assigned managed identity enal
Azure role-based-access-control. The life
identity. Learn more about Managed ider

Status: Off [✓ On]

Figure 6.17

Once enabled, `app-corewithKey Vault` will be registered with Azure Active

Directory. Once it is registered, `app-corewithKey Vault` can be granted permissions to access resources protected by Azure AD.

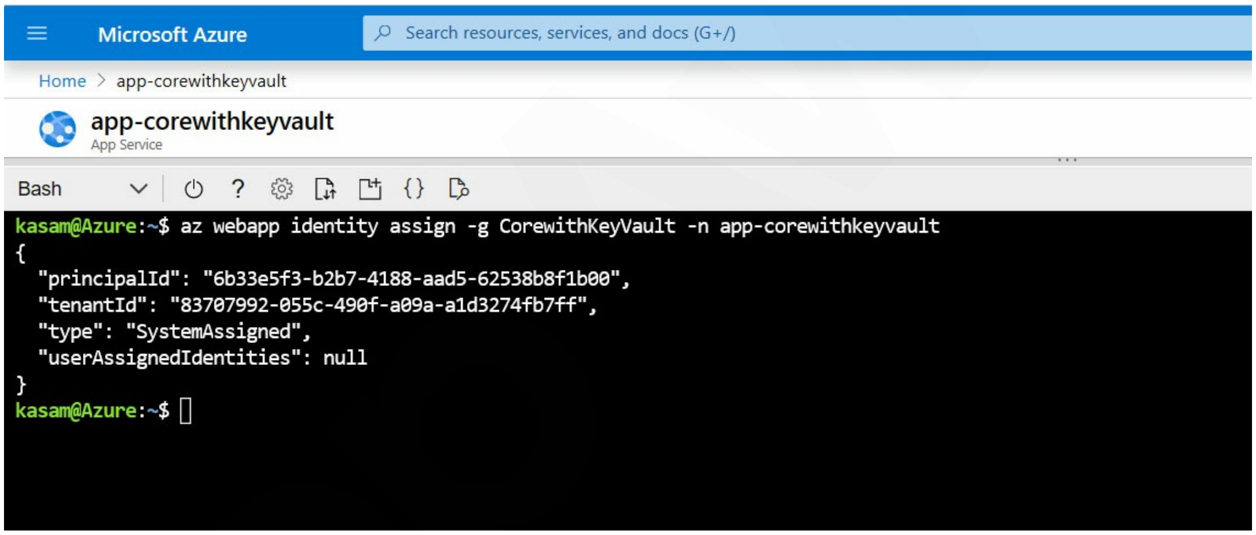
Now, once you enable the managed service identity, the web app can now be used as a service account and added under Azure key vault access policies.

I believe, being an Azure developer, you should know the different ways to perform any action on Azure resources. And hence will have the smartest way of doing, that is, via Azure CLI. Yes, you can also enable the identity by the Azure Command Line interface.

Azure CLI command to enable identity for the web app is:

```
az webapp identity assign -g ResourceGroupName -n WebAppName
```

Running the command replacing the actual values:


Microsoft Azure
Search resources, services, and docs (G+/)
Home > app-corewithkeyvault
app-corewithkeyvault
App Service
Bash

```
kasam@Azure:~$ az webapp identity assign -g CorewithKeyVault -n app-corewithkeyvault
{
  "principalId": "6b33e5f3-b2b7-4188-aad5-62538b8f1b00",
  "tenantId": "83707992-055c-490f-a09a-a1d3274fb7ff",
  "type": "SystemAssigned",
  "userAssignedIdentities": null
}
kasam@Azure:~$ 
```

Figure 6.18

Key Vault access policies to Web App

To do so, navigate to the Key vault account page, and from left blade, under **`Settings`**|click **`Access Policies`**.

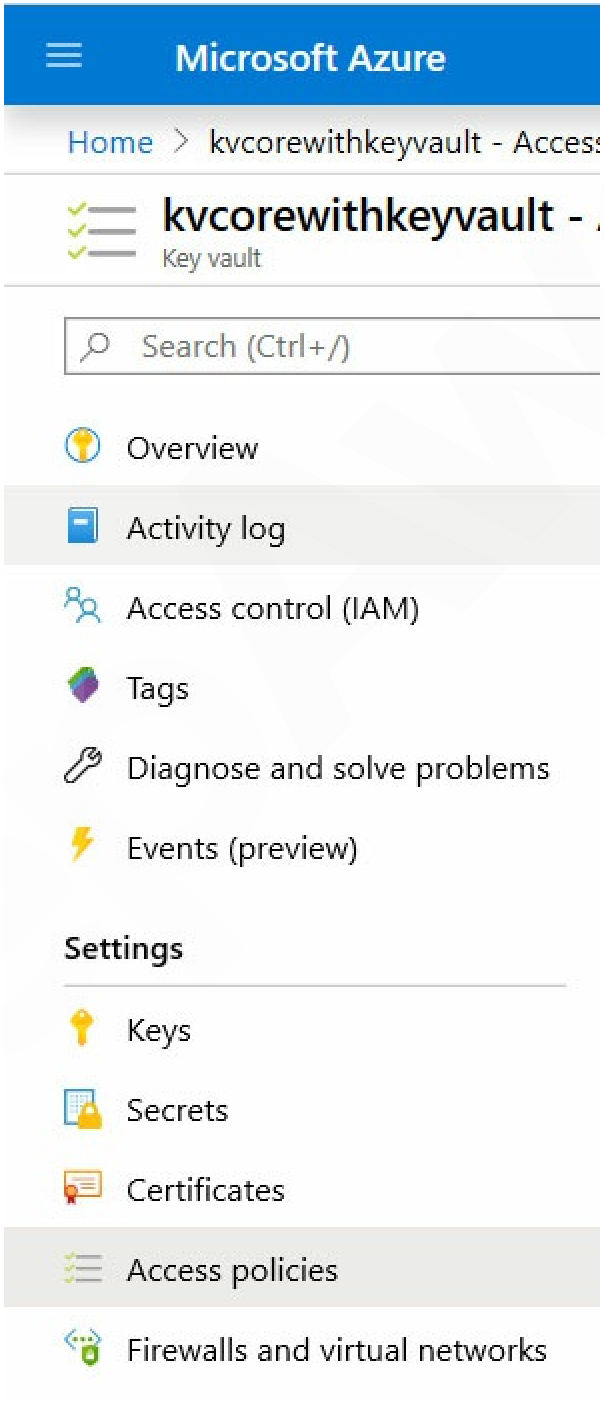
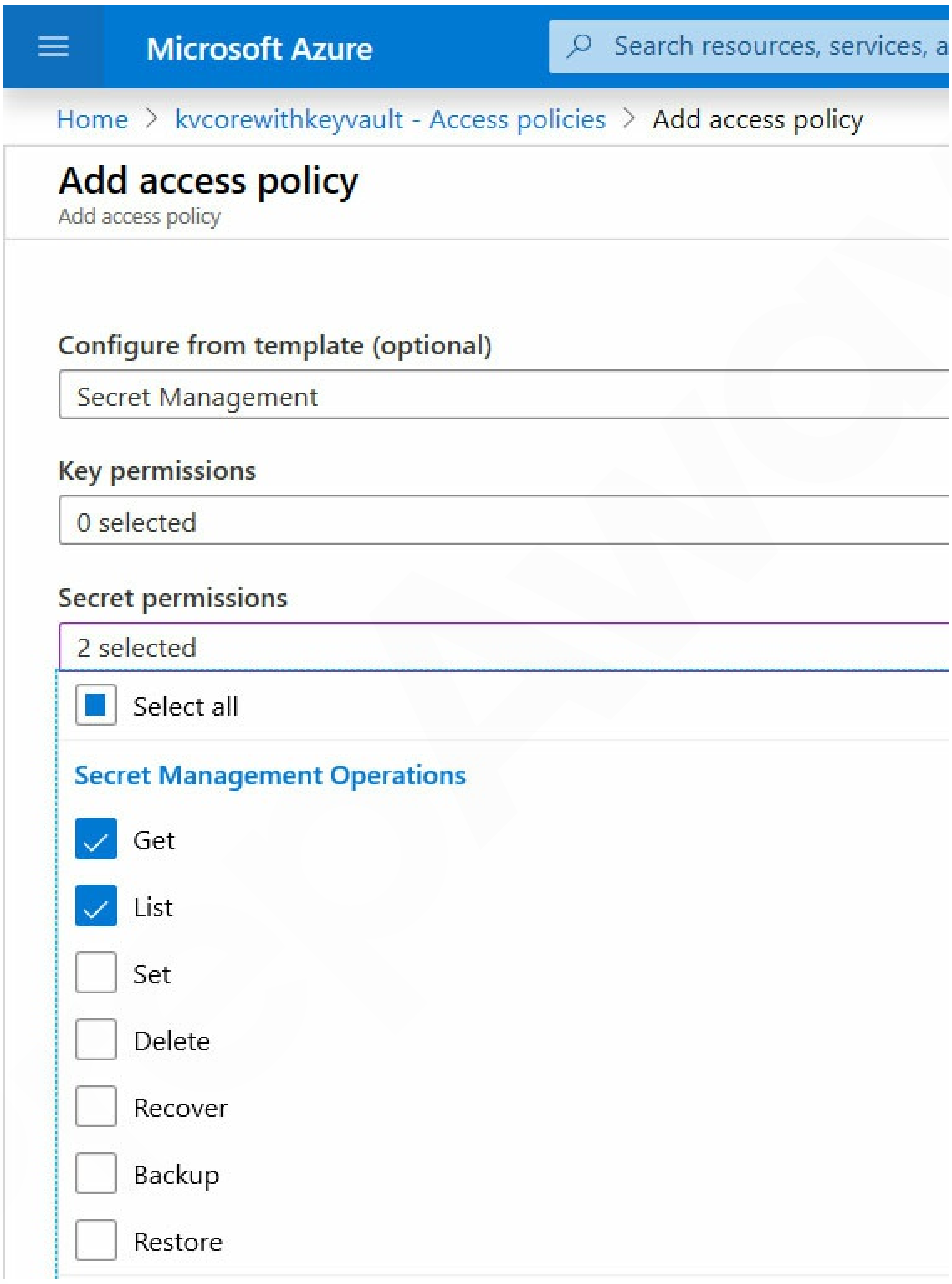


Figure 6.19

Now, click on **`+Add Access Policy`**. It will present you with a blade to provide details for adding it to the policy.

Configure from template–this provides you the list of management options available for Key vault resources. In this exercise, you are more concerned with managing **`Secrets`**. Hence select, **`Secret Management`** as an option.

Next comes for Secret Permission to be granted. Be default, all operations are checked. But here you have to follow, least access principle to go with. You want your application to get or list the key and no other operations to be performed, such as delete, and so on. Following this, uncheck all the boxes except **`Get`** and **`List`**, as can be seen in the following image.


Microsoft Azure

Search resources, services, a

Home > kvcorewithkeyvault - Access policies > Add access policy

Add access policy
Add access policy

Configure from template (optional)
Secret Management

Key permissions
0 selected

Secret permissions
2 selected

■ Select all

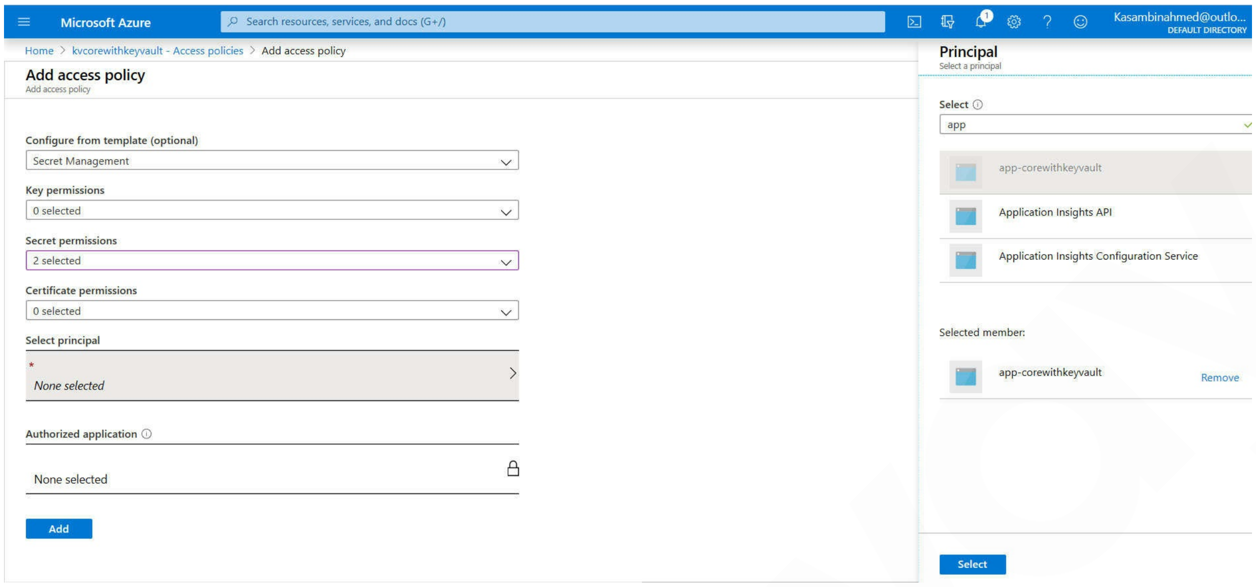
Secret Management Operations

[✓] Get
[✓] List
[ ] Set
[ ] Delete
[ ] Recover
[ ] Backup
[ ] Restore

Figure 6.20

Next comes in selecting principal, click on the option. It will present you with a blade on the left side of the screen.

Under the blade, type the name of the web application, select the same and click on the **`Select`** button.


Microsoft Azure
Search resources, services, and docs (G+/)
Kasambinahmed@outlo...
DEFAULT DIRECTORY
Home > kvcorewithkeyvault - Access policies > Add access policy

Add access policy

Add access policy

Configure from template (optional): Secret Management

Key permissions: 0 selected

Secret permissions: 2 selected

Certificate permissions: 0 selected

Select principal: * None selected

Authorized application: None selected

Add

Principal

Select a principal

Select: app

app-corewithkeyvault

Application Insights API

Application Insights Configuration Service

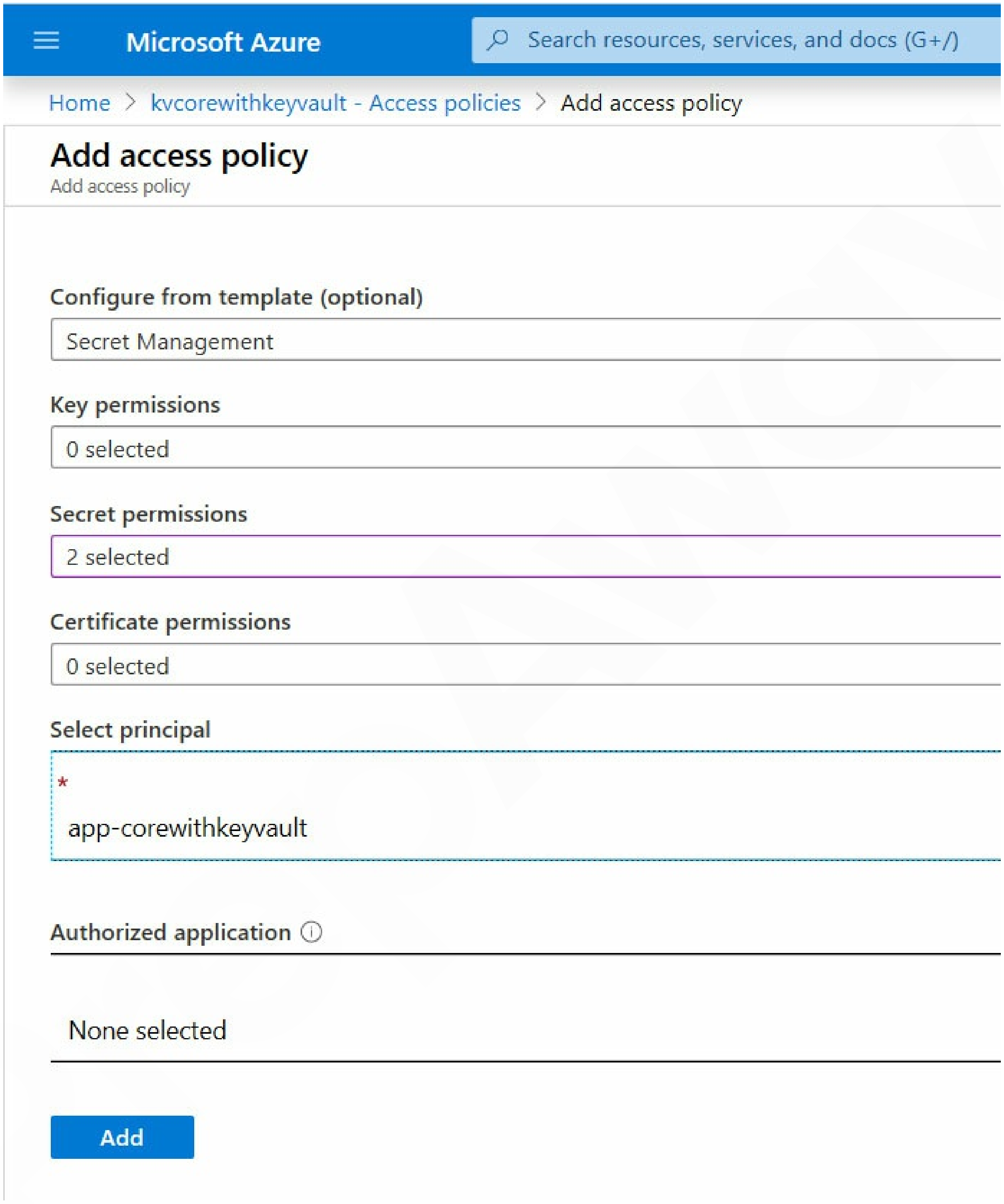
Selected member:

app-corewithkeyvault Remove

Select

Figure 6.21

Once all the values are added, go ahead and click on the **`Add`** button.


Microsoft Azure
Search resources, services, and docs (G+/)
Home > kvcorewithkeyvault - Access policies > Add access policy

Add access policy
Add access policy

Configure from template (optional): Secret Management

Key permissions: 0 selected

Secret permissions: 2 selected

Certificate permissions: 0 selected

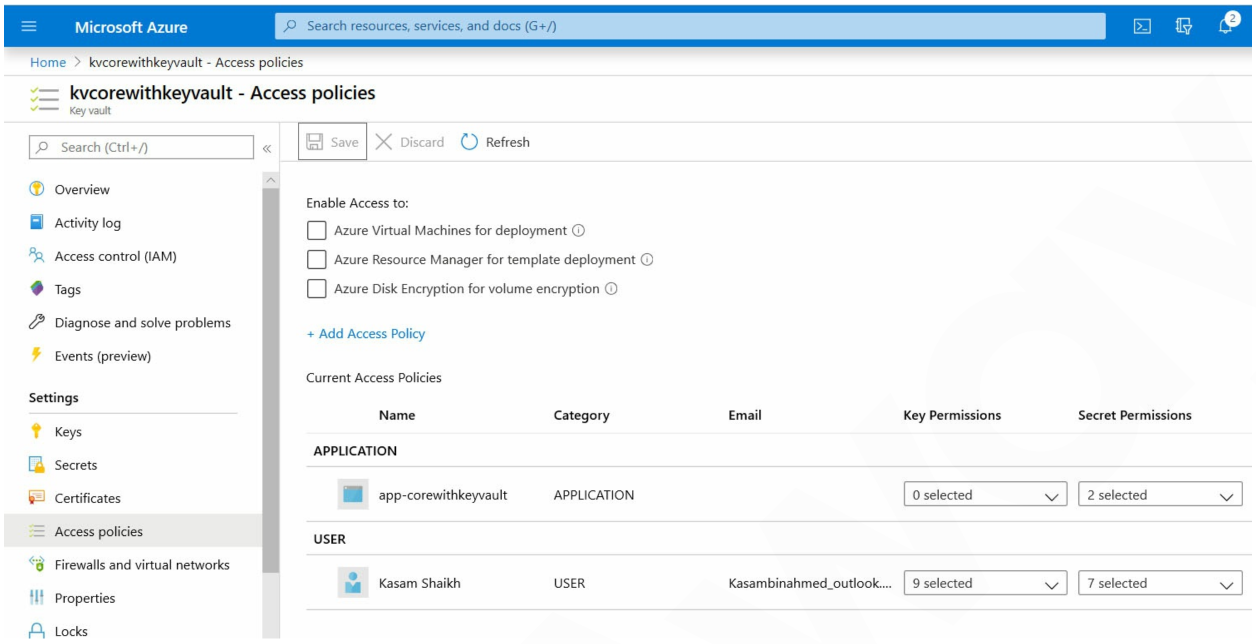
Select principal: * app-corewithkeyvault

Authorized application: None selected

Figure 6.22

And click on **Save**.

Now the web application will be listed under key vault access policies section with two secret permission checked with.



kvcorewithkeyvault - Access policies

Key vault

Enable Access to:

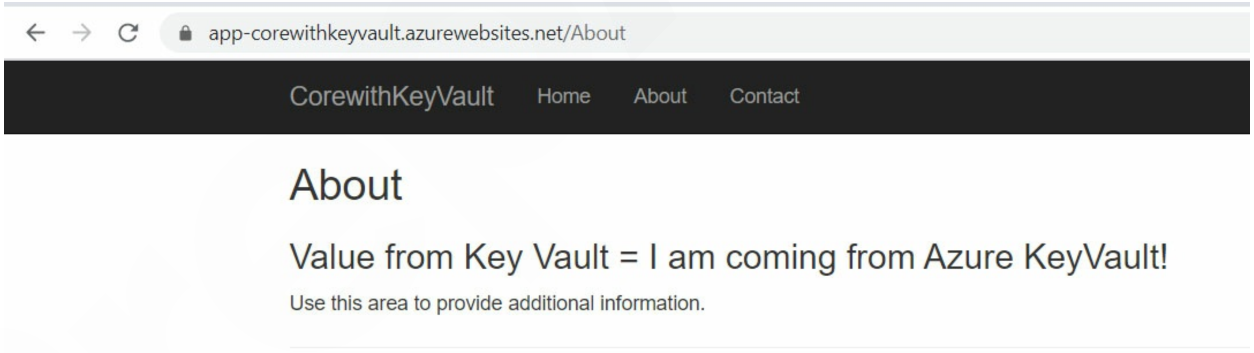
- [ ] Azure Virtual Machines for deployment
- [ ] Azure Resource Manager for template deployment
- [ ] Azure Disk Encryption for volume encryption

Current Access Policies

Name	Category	Email	Key Permissions	Secret Permissions
APPLICATION				
app-corewithkeyvault	APPLICATION		0 selected	2 selected
USER				
Kasam Shaikh	USER	Kasambinahmed_outlook....	9 selected	7 selected

Figure 6.23

After performing all the preceding steps, open up Web App in the browser, and open up the **`About Us`** page to verify, does the setting worked fine, and to read the Key vault value.



About

Value from Key Vault = I am coming from Azure KeyVault!

Use this area to provide additional information.

Figure 6.24

It is working as expected.

App service configuration

Azure App Configuration it’s another supercool service helping you to store, retrieve, and manage access to application settings all in one place. It is easy to set up and simple to use from any application. It also gives you the ability to modify an application's behavior on demand without having to redeploy the application.

App Configuration offers the following benefits:

- A fully managed service that can be set up in minutes.
- Flexible key representations and mappings.
- Versioning with labels.
- Point-in-time replay of settings.
- Comparison of two sets of configurations on custom-defined dimensions.
- Enhanced security through Azure-managed identities.
- Complete data encryption at rest or in transit.
- Native integration with popular frameworks including .NET and Java.

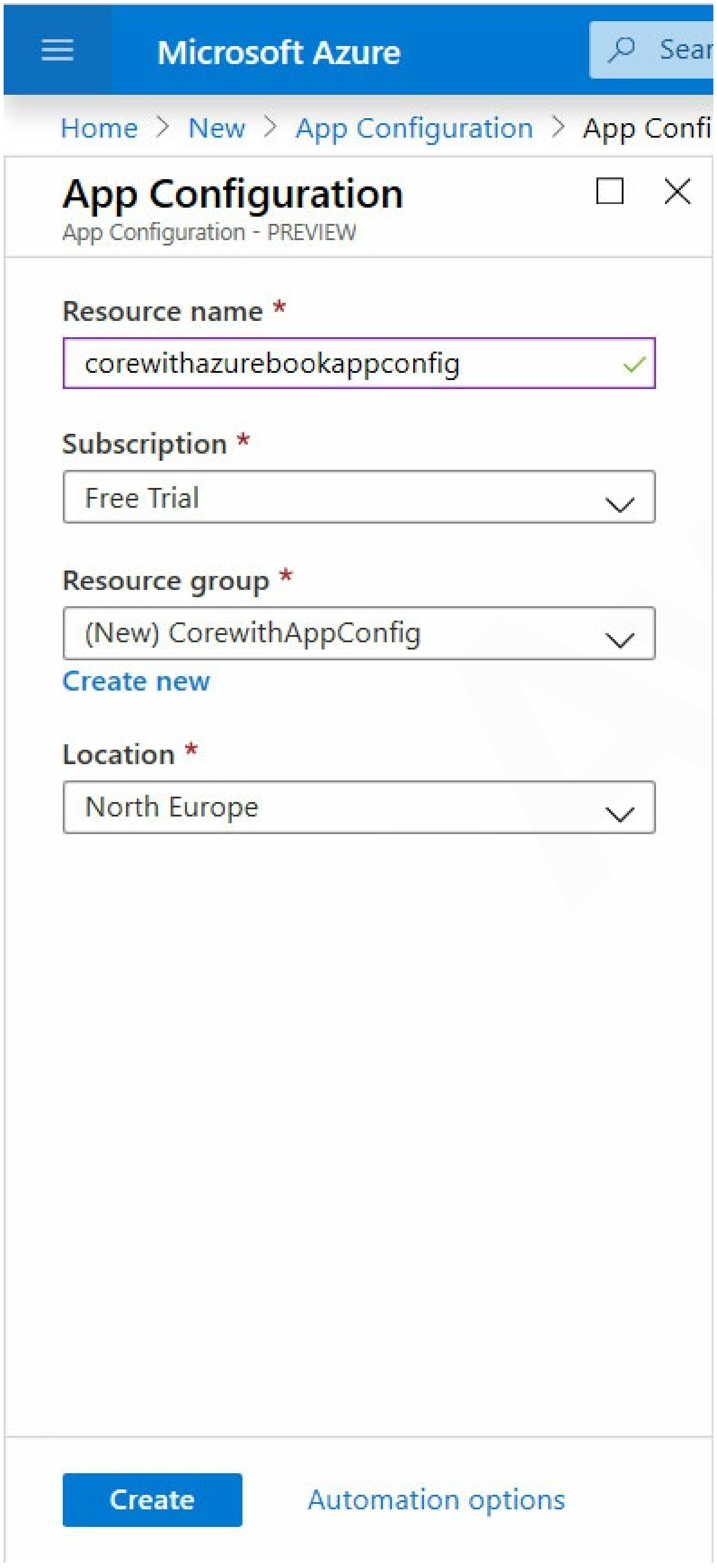
Working with the service

Open the Azure portal, and create a new resource. Search for **`App Configuration`** and click on **`Create`**.

It will present you with a blade to have the following details:

- **`Resource Name:`** Name of your app configuration service. For this exercise, I am giving it as `corewithazurebookappconfig`.
- **`Subscription:`** Select your valid subscription.
- **`Resource Group:`** A logical grouping for all azure resources. I will create a new resource group with the name `CorewithAppConfig`.
- **`Location:`** Location the resource will get deployed.

Click on **`Create`** to proceed and wait for fewer seconds to have resources created.



App Configuration

App Configuration - PREVIEW

Resource name *: corewithazurebookappconfig

Subscription *: Free Trial

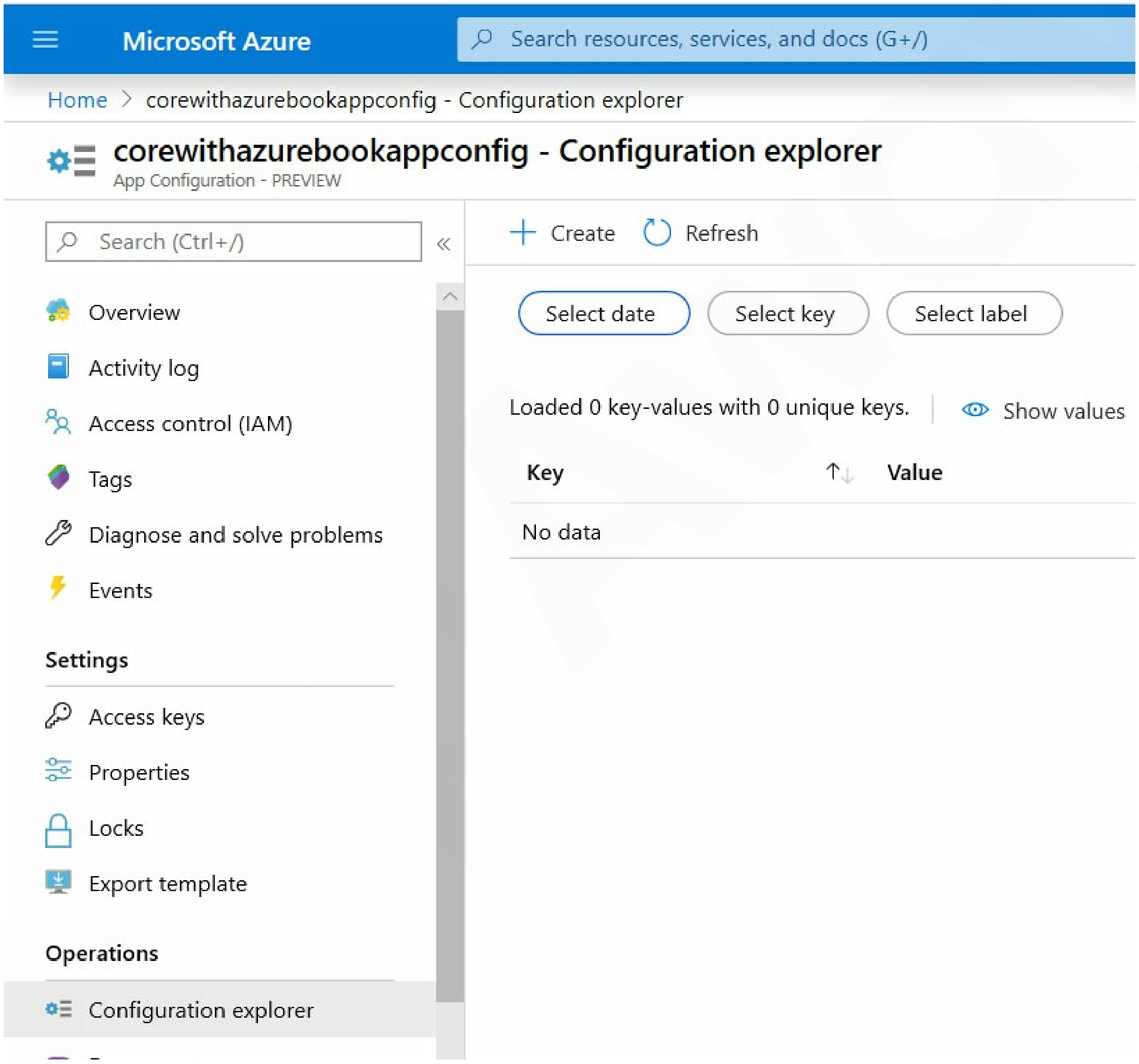
Resource group *: (New) CorewithAppConfig

Location *: North Europe

Figure 6.25

Once the resource is created, it's time to add some configuration to it.

Go to the resource page, from left-hand blade under **`Operations|`** select, **`Configuration Explorer`**.



Loaded 0 key-values with 0 unique keys.

No data

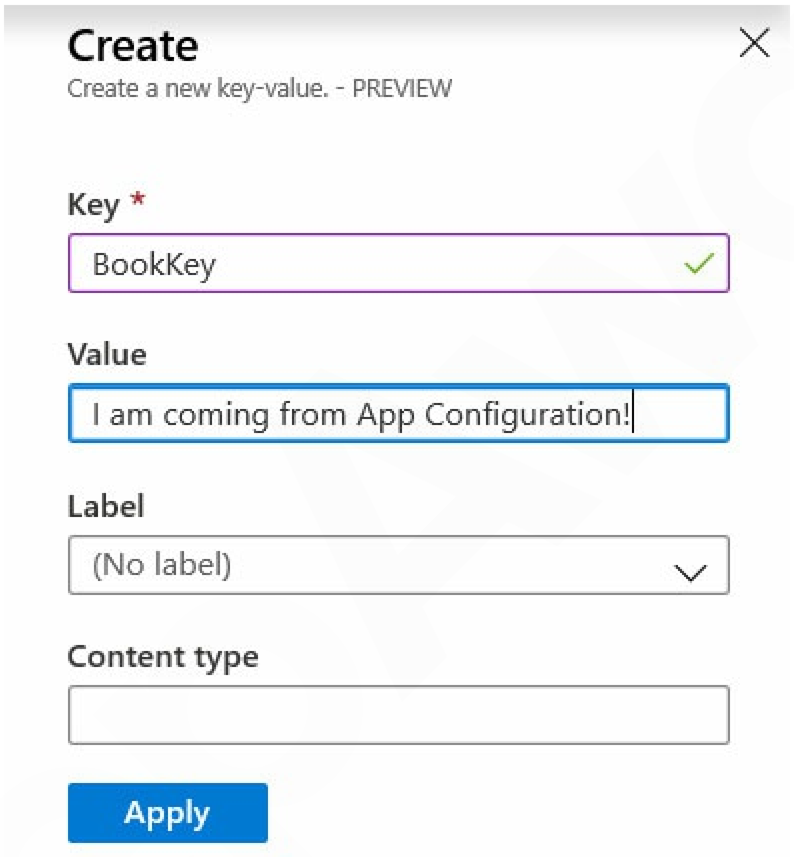
Figure 6.26

Click on **`+Create`** from the top menu, to have some key-value pair in the config. This key, later, I will show to retrieve in our .Net core application.

This will present you with the blade at the right-hand side of the screen for entering key-value details you need to store in app configuration:

- **`Key:`** Name of the Key
- **`Value:`** Value for the Key.
- **`Label:`** Labelling of the Key
- **`Content Type:`** Type of content associated with Key.

Following are the details, I entered:



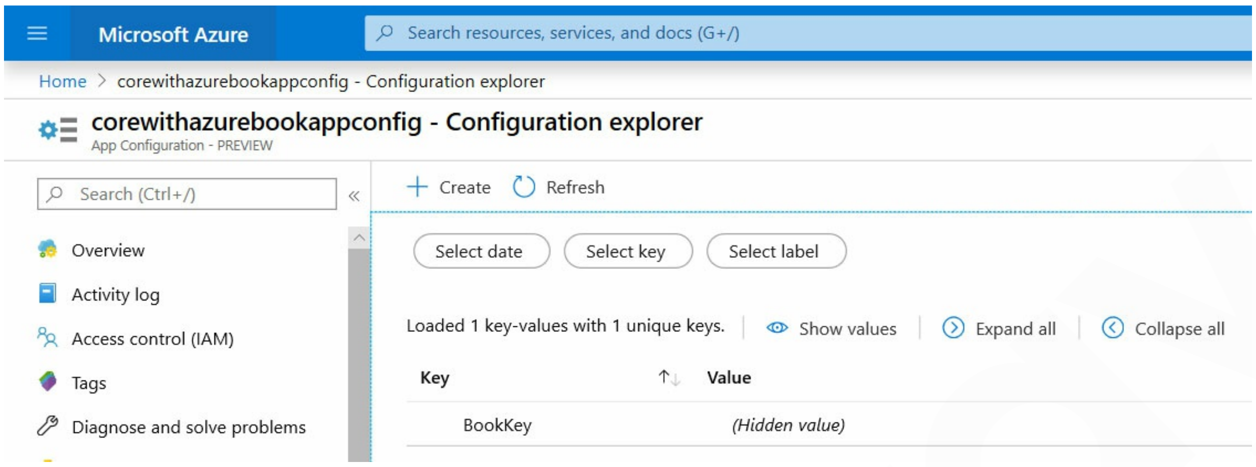
Create

Create a new key-value. - PREVIEW

Key *	BookKey
Value	I am coming from App Configuration!
Label	(No label)
Content type	

Figure 6.27

Entering the details, click on **`Apply`** to save the key. And later can be seen in **`Configuration Explorer`**:


Microsoft Azure
Search resources, services, and docs (G+/)
Home > corewithazurebookappconfig - Configuration explorer
corewithazurebookappconfig - Configuration explorer
App Configuration - PREVIEW
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Create
Refresh
Select date
Select key
Select label
Show values
Expand all
Collapse all

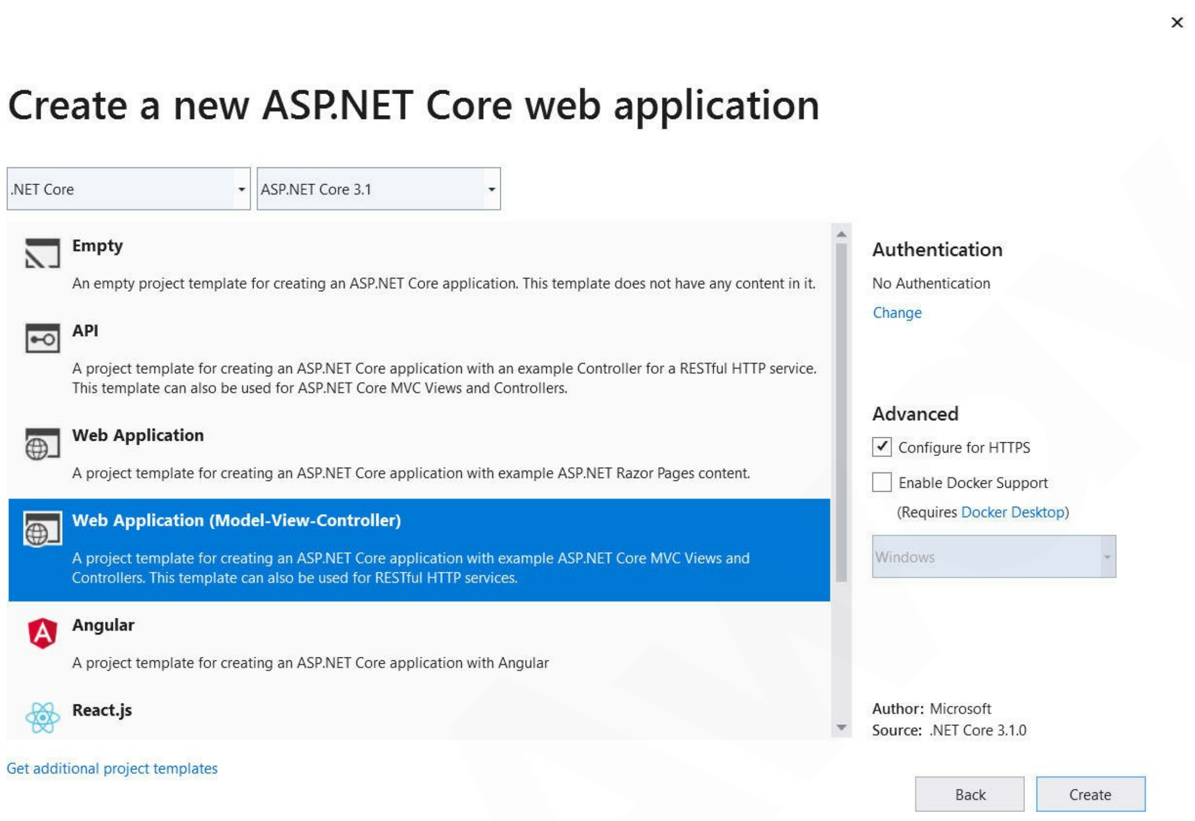
Loaded 1 key-values with 1 unique keys.

Key	Value
BookKey	(Hidden value)

Figure 6.28

.NET Core Application Code part

Create an ASP.NET Core 3.1 MVC application using Visual Studio 2019. Hope you have installed the .NET SDKs for core 3 and later, as a part of the exercise in an earlier chapter:



×

Create a new ASP.NET Core web application

.NET Core | ASP.NET Core 3.1

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

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.

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

Web Application (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.

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

React.js

Authentication
No Authentication
Change

Advanced
✓ Configure for HTTPS
☐ Enable Docker Support
(Requires Docker Desktop)
Windows

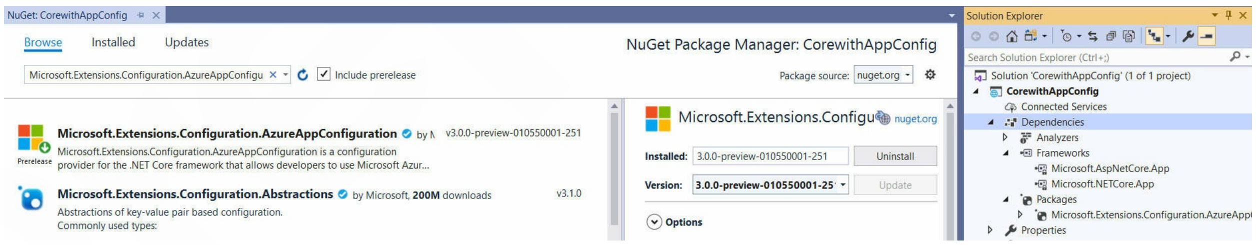
Author: Microsoft
Source: .NET Core 3.1.0

Get additional project templates

Back | Create

Figure 6.29

Install the following package from **`NuGet Package Manager`** to work with an App configuration: **`Microsoft.Extensions.Configuration.AzureAppConfiguration:`**



NuGet: CorewithAppConfig

Browse | Installed | Updates

Microsoft.Extensions.Configuration.AzureAppConfigu
✓ Include prerelease

Microsoft.Extensions.Configuration.AzureAppConfiguration by M v3.0.0-preview-010550001-251
Prerelease
Microsoft.Extensions.Configuration.AzureAppConfiguration is a configuration provider for the .NET Core framework that allows developers to use Microsoft Azur...

Microsoft.Extensions.Configuration.Abstractions by Microsoft, **200M** downloads v3.1.0
Abstractions of key-value pair based configuration.
Commonly used types:

NuGet Package Manager: CorewithAppConfig

Package source: nuget.org

Microsoft.Extensions.Configu nuget.org

Installed: 3.0.0-preview-010550001-251 | Uninstall
Version: 3.0.0-preview-010550001-25 | Update

Options

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'CorewithAppConfig' (1 of 1 project)
CorewithAppConfig
Connected Services
Dependencies
Analyzers
Frameworks
Microsoft.AspNetCore.App
Microsoft.NETCore.App
Packages
Microsoft.Extensions.Configuration.AzureApp
Properties

Figure 6.30

In order to read the keys form app configuration, we need to access it from an application using an app configuration connection string.

Navigate to **`Azure Portal | Azure AppCconfiguration`** resource.

From the left-hand pane, under **`Settings`**|select **`Access Keys:`**

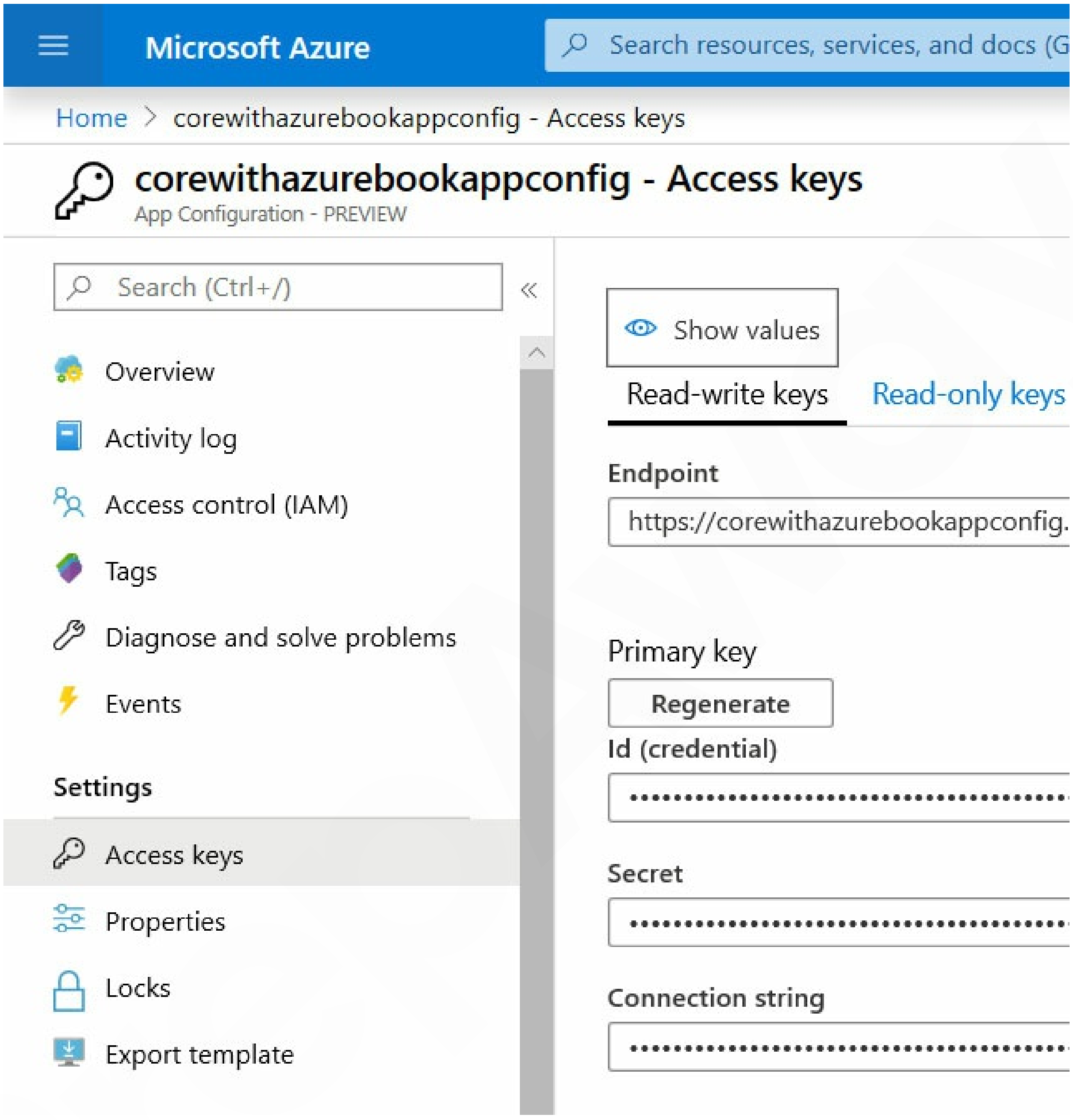


Figure 6.31

You will be presented by **`App Configuration`** resource endpoint and set of **`Primary key`** and **`Secondary key`**, with **`ID`**, **`Secret`**, and **`Connection string`**.

Copy the connection string, and will add the same in the application.

Right-click on **`Project`** in **`Solution Explorer`**, and select **`Manage User Secrets`**:

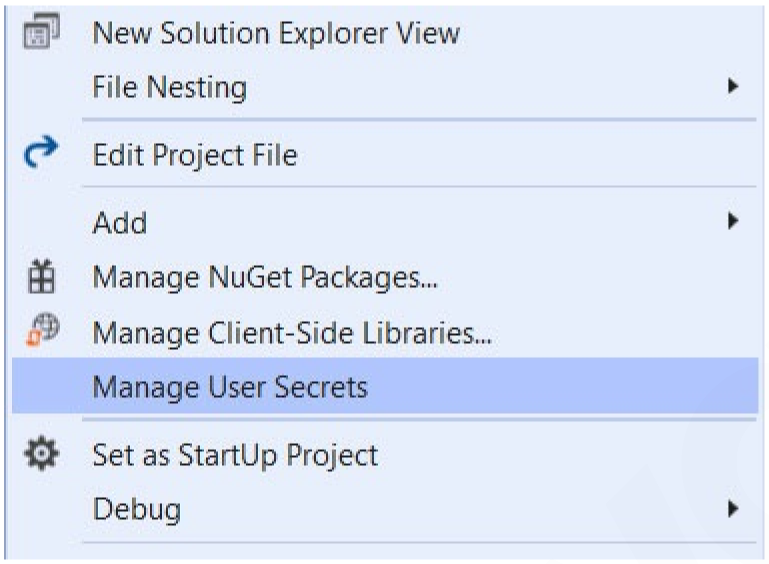
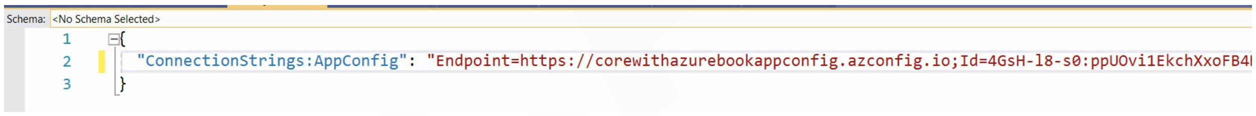


Figure 6.32

It will open a file name `Secret.json`, add the following lines:

```
"ConnectionStrings:AppConfig": "<enter the connectionstring to
App Configuration that we copied in the previous section>"
```

It will look as follows:

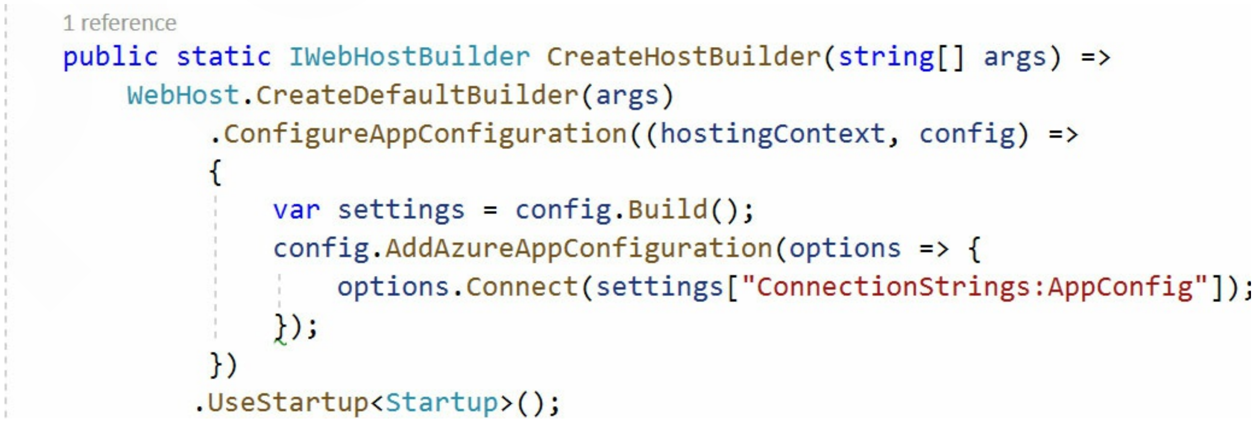


```
Schema: <No Schema Selected>
{
  "ConnectionStrings:AppConfig": "Endpoint=https://corewithazurebookappconfig.azconfig.io;Id=4GsH-l8-s0:ppUOvi1EkchXxoFB4
}
```

Figure 6.33

Open `program.cs` file, add update the `CreateHostBuilder` method with following lines of code.

This will be connecting the app configurations key using the connection string you added in the `secret.json` file.



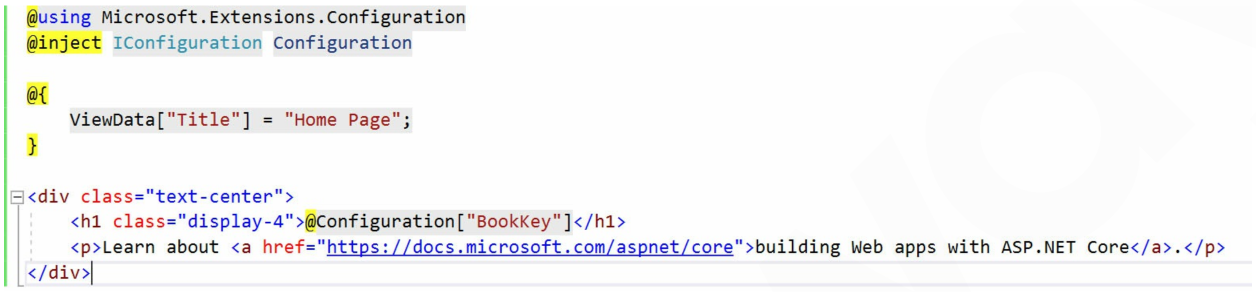
```
1 reference
public static IWebHostBuilder CreateHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((hostingContext, config) =>
        {
            var settings = config.Build();
            config.AddAzureAppConfiguration(options => {
                options.Connect(settings["ConnectionStrings:AppConfig"]);
            });
        })
       .UseStartup<Startup>();
```

Figure 6.34

Read the Key from App Configuration

Open up index.html file, and following lines:
`Notice we are using the Key from App configuration, as Configuration [“BookKey”],`

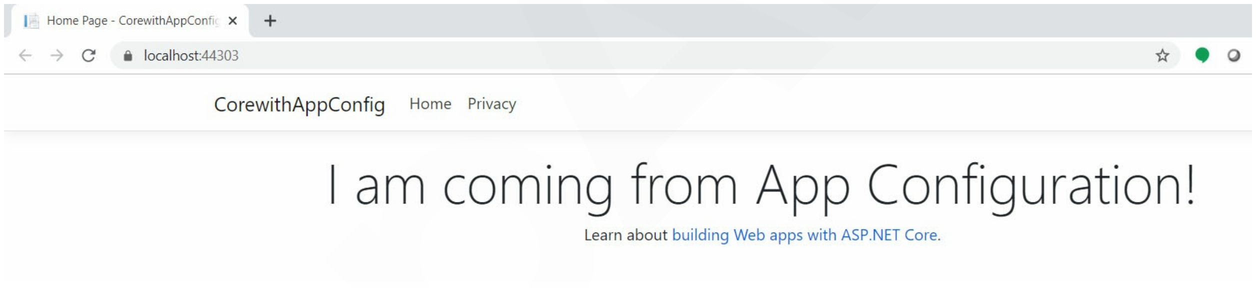
Here to fetch the value:



```
@using Microsoft.Extensions.Configuration
@inject IConfiguration Configuration

@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">@Configuration["BookKey"]</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>
```

Figure 6.35

Run the application:



I am coming from App Configuration!

Learn about building Web apps with ASP.NET Core.

Figure 6.36

The preceding message is now coming from `App configuration`, value for key name, `BookKey`.

Cool, right, now note this App configuration also comes with feature flags wherein you can enable any settings in your application to say, enabling an **`Update`** page, etc. by one click. Take it as homework and explore the feature.

Tip: It will require a few more tweaks in code to enable the feature flag in your application.

Conclusion

In this chapter, you learned about different ways of secure key management in Azure. How seamlessly you can implement in your .NET Core application with minimal lines of code. I hope you found this interesting and helpful to implement the same at your workplace. In the next chapter, you will see the working with Azure Serverless offering and Azure functions with Visual Studio 2019. Happy Azure Coding.

Questions

1. What are the service offerings available for central key management in Azure?
2. Explain the benefits of Azure KeyVault?
3. What do you mean by Managed Identities for Azure resources?
4. What is the Azure App Configuration service?
5. What are the .NET packages required to integrate KeyVault in an application?

CHAPTER 7

Step Towards Serverless Approach With Azure Functions

Welcome back! In the previous chapter, you learned about the services used for the centralization of secrets application keys with minimal code, enabling your application source code free from having keys. I strongly believe you must have tried with all the mentioned exercise and very much completed the homework. In this chapter, you will learn about working with Azure Serverless offering Azure functions with Visual Studio 2019.

Structure

- Azure function
- Debugging C# Azure functions
- Talk with Azure Storage
- Move on Cloud
- Testing function
- Live debugging of Azure function

Objectives

The objective of this chapter is:

- Introduction to Azure serverless offerings.
- Things developer should know about Azure functions
- Working with C# Azure functions on a local environment using Visual Studio 2019
- Debugging Live Azure Function on Cloud using Visual Studio 2019

Things you should know about Azure function

In simpler terms, Azure function is a piece of code that can be executed independently of any underlined infrastructure, a piece of code executed by a restful call. You have to be more focused on code than on how and where the code will get reside. When we say Serverless, it does not mean, **No Server** but, does mean **Less Server**. Server managed and controlled by Microsoft Azure, and not you.

Azure functions, Azure logic apps are few among services offered under Serverless offerings. An entire book can be written on this subject of service in the **Integration Platform as a Service offering (IPaaS)**. I will focus more on working with Azure function.

The azure function is the best choice for architects during solving microservices architecture. Azure functions are treated as a set of independent APIs used for achieving different tasks. By nature, we can build a Stateless architecture using Azure functions APIs. But it does allow building a stateful architecture, or with session management by using something called **Durable Functions**. Note, one of my favorite interview questions.😊

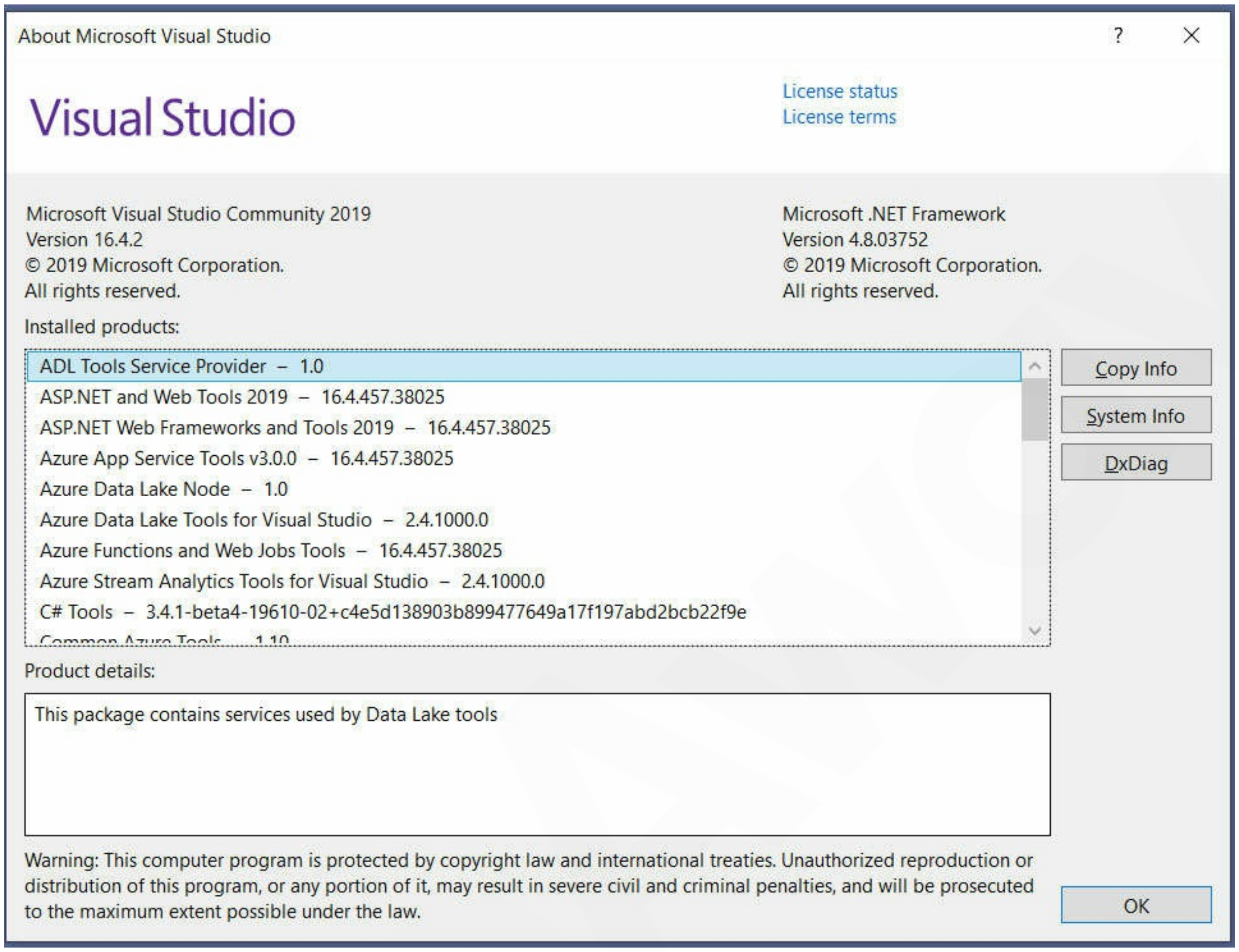
Create an Azure function

You can very well create the Azure functions using Azure Portal, Azure CLI, PowerShell, and ARM templates. But you can also work with Azure function by developing the function in our favorite IDE, Visual Studio 2019. The best part is:

- You can create Azure function in Visual Studio without an Azure Subscription
- You can debug the code line by line
- Few steps, seamlessly deploy the function to Azure
- If deployed the code to Azure, you can still debug the Azure functions on the cloud, using Visual Studio 2019

And many more. You can also leverage many intelligent features of IDE to accelerate function development.

For this exercise, I am using Visual Studio 2019 Community edition, version 16.4.2. At the time of writing this book, this was the latest version available.


About Microsoft Visual Studio

Visual Studio

License status
License terms

Microsoft Visual Studio Community 2019
Version 16.4.2
© 2019 Microsoft Corporation.
All rights reserved.

Microsoft .NET Framework
Version 4.8.03752
© 2019 Microsoft Corporation.
All rights reserved.

Installed products:

ADL Tools Service Provider – 1.0
ASP.NET and Web Tools 2019 – 16.4.457.38025
ASP.NET Web Frameworks and Tools 2019 – 16.4.457.38025
Azure App Service Tools v3.0.0 – 16.4.457.38025
Azure Data Lake Node – 1.0
Azure Data Lake Tools for Visual Studio – 2.4.1000.0
Azure Functions and Web Jobs Tools – 16.4.457.38025
Azure Stream Analytics Tools for Visual Studio – 2.4.1000.0
C# Tools – 3.4.1-beta4-19610-02+c4e5d138903b899477649a17f197abd2bcb22f9e

Product details:

This package contains services used by Data Lake tools

Warning: This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Figure 7.1

For working with Azure, do remember to; select **`Azure Workload`** while installing the Visual Studio 2019.

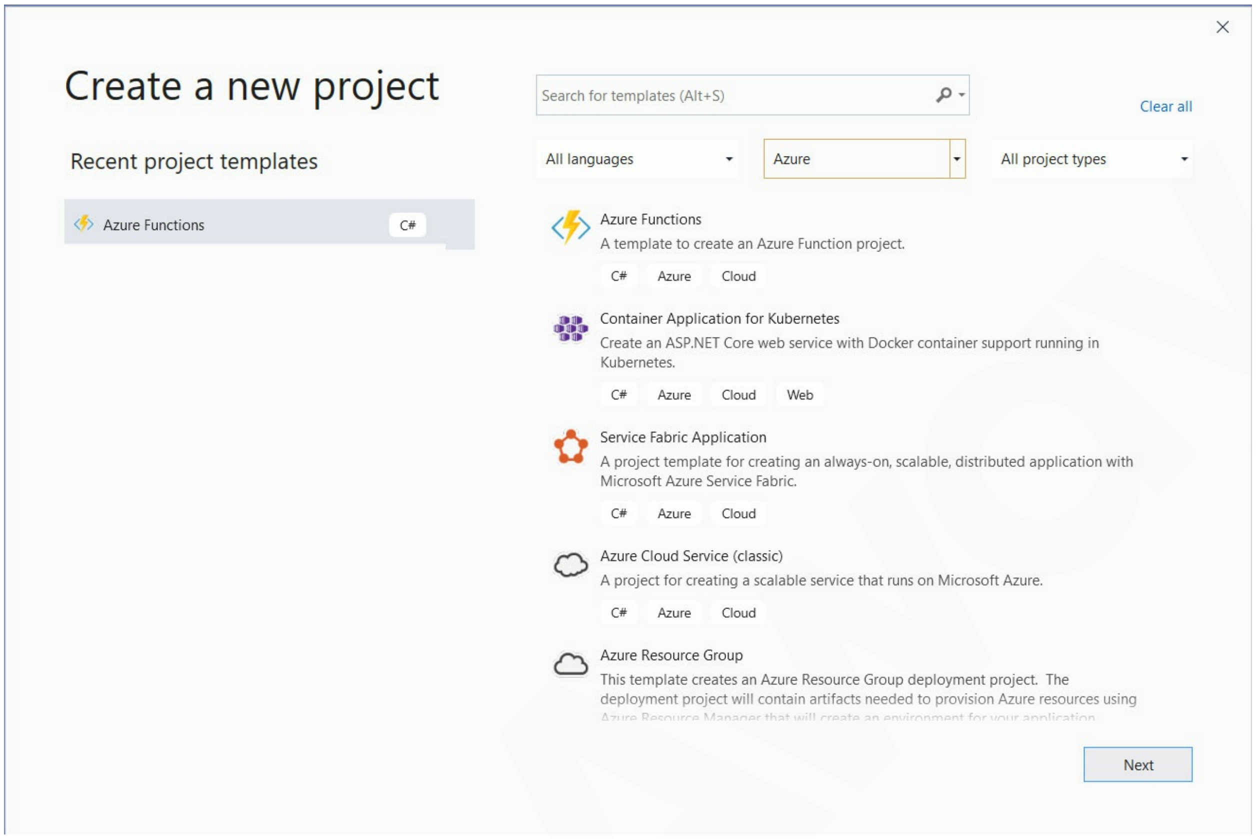
This exercise will be creating Azure functions with the runtime as 3.0.

Azure functions come with three runtimes as an option:

- **`V1:`** Supports .NET Frameworks
- **`V2:`** Support .NET Core 2.x
- **`V3:`** Supports .NET Core 3.x

This is the first book to have any exercise with Azure function runtime as 3 and .NET Core 3.1, as it just got live for production.

Click on **`Files | New Project | Azure`** from the dropdown, and it will present you with multiple project templates to start with. Select Azure function from the project template listed, and click on **`Next`**.



Create a new project

Recent project templates

- Azure Functions — C#

Search for templates (Alt+S)

Clear all

All languages | Azure | All project types

Azure Functions
A template to create an Azure Function project.
C# · Azure · Cloud

Container Application for Kubernetes
Create an ASP.NET Core web service with Docker container support running in Kubernetes.
C# · Azure · Cloud · Web

Service Fabric Application
A project template for creating an always-on, scalable, distributed application with Microsoft Azure Service Fabric.
C# · Azure · Cloud

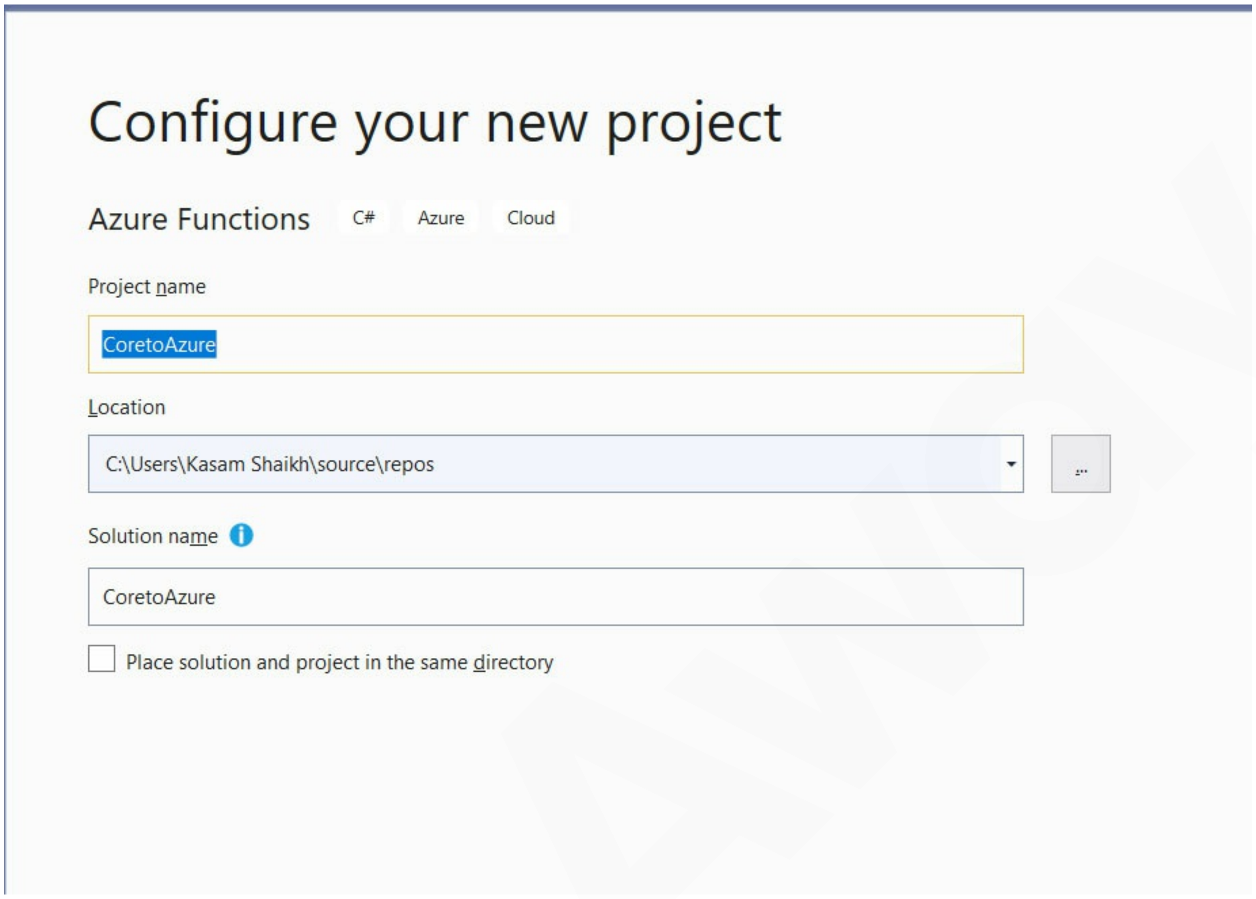
Azure Cloud Service (classic)
A project for creating a scalable service that runs on Microsoft Azure.
C# · Azure · Cloud

Azure Resource Group
This template creates an Azure Resource Group deployment project. The deployment project will contain artifacts needed to provision Azure resources using Azure Resource Manager that will create an environment for your application.

Next

Figure 7.2

Provide the `Project name`, `Location`, and `Solution name` of your solution:



Configure your new project

Azure Functions C# Azure Cloud

Project name: CoretoAzure

Location: C:\Users\Kasam Shaikh\source\repos

Solution name: CoretoAzure

☐ Place solution and project in the same directory

Figure 7.3

Next comes the interesting screen to choose for:

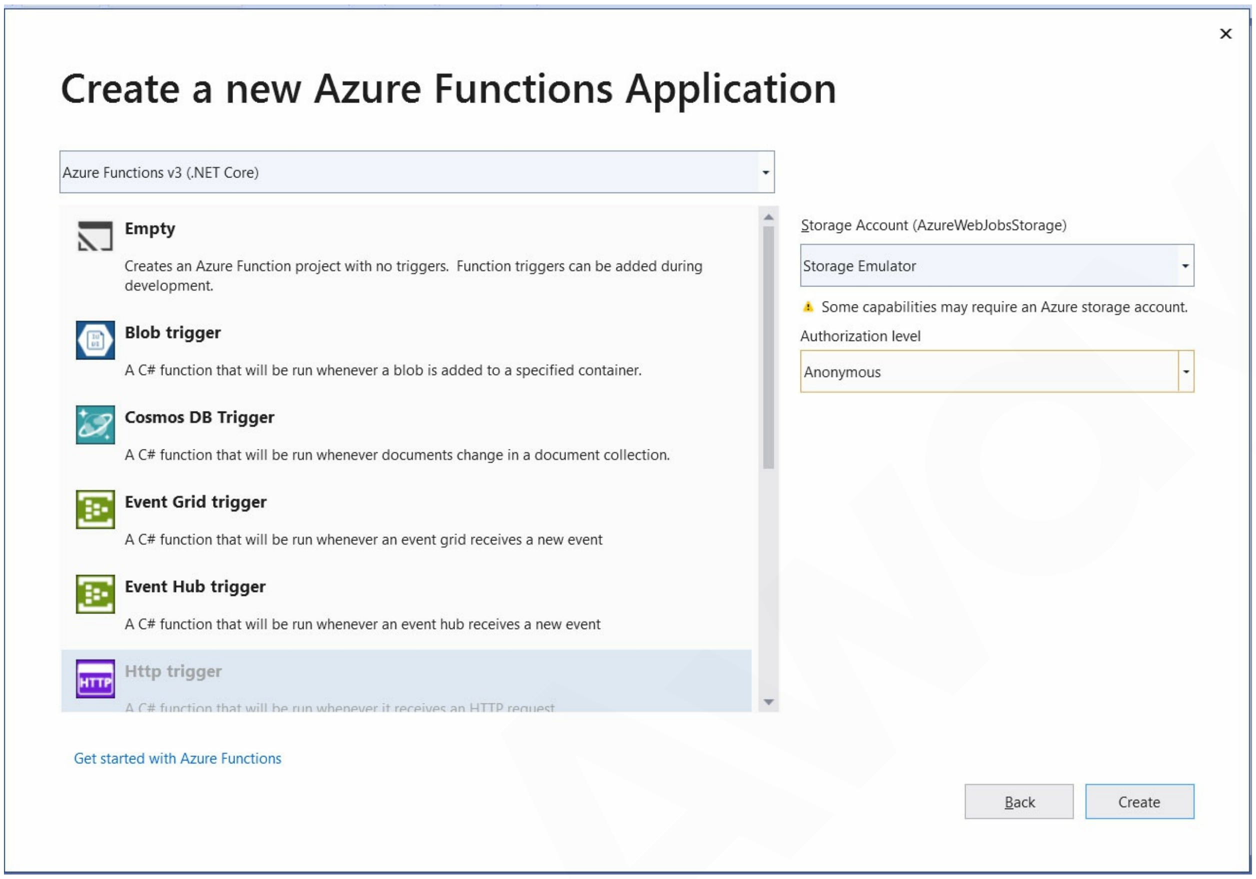
Azure Runtimes from the dropdown, and also to select with ready to start code projects.

When it comes to Authorization, there comes an optionsuch as:

- **`Functions:`** Where you need to provide the resource key.
- **`Anonymous:`** To select with and best to go within the dev environment.
- Admin.

Will go with options as seen in the following screenshot, **`Azure Functions v3, Http trigger`** function to start with, with **`Authorization level`** as **`Anonymous`** as will not connect it to Azure initially.

Click on **`Create`**, to proceed:



Create a new Azure Functions Application

Azure Functions v3 (.NET Core)

Empty
Creates an Azure Function project with no triggers. Function triggers can be added during development.

Blob trigger
A C# function that will be run whenever a blob is added to a specified container.

Cosmos DB Trigger
A C# function that will be run whenever documents change in a document collection.

Event Grid trigger
A C# function that will be run whenever an event grid receives a new event

Event Hub trigger
A C# function that will be run whenever an event hub receives a new event

Http trigger
A C# function that will be run whenever it receives an HTTP request

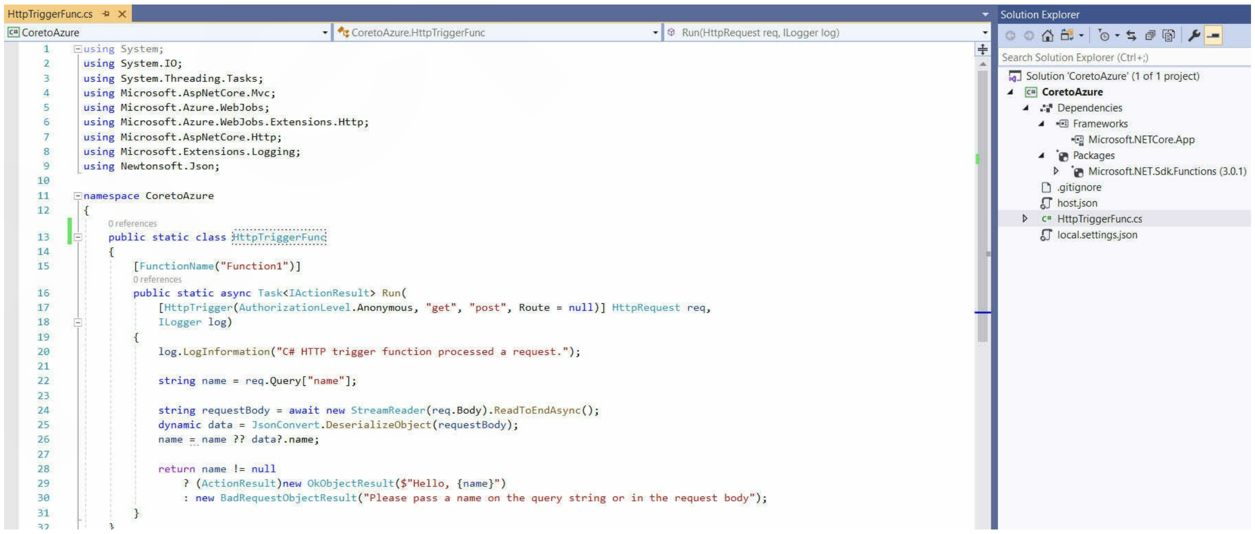
Storage Account (AzureWebJobsStorage): Storage Emulator

Some capabilities may require an Azure storage account.

Authorization level: Anonymous

Figure 7.4

It will present with a class file name Function1, with a piece of code. You can change the class name to any valid name of your choice. You can also see the part of the package of the function in **`Solution Explorer`**:



HttpTriggerFunc.cs
CoretoAzure
CoretoAzure.HttpTriggerFunc
Run(HttpRequest req, ILogger log)

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;

namespace CoretoAzure
{
    0 references
    public static class HttpTriggerFunc
    {
        [FunctionName("Function1")]
        0 references
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,
            ILogger log)
        {
            log.LogInformation("C# HTTP trigger function processed a request.");

            string name = req.Query["name"];

            string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
            dynamic data = JsonConvert.DeserializeObject(requestBody);
            name = name ?? data?.name;

            return name != null
                ? (ActionResult)new OkObjectResult($"Hello, {name}")
                : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
        }
    }
```

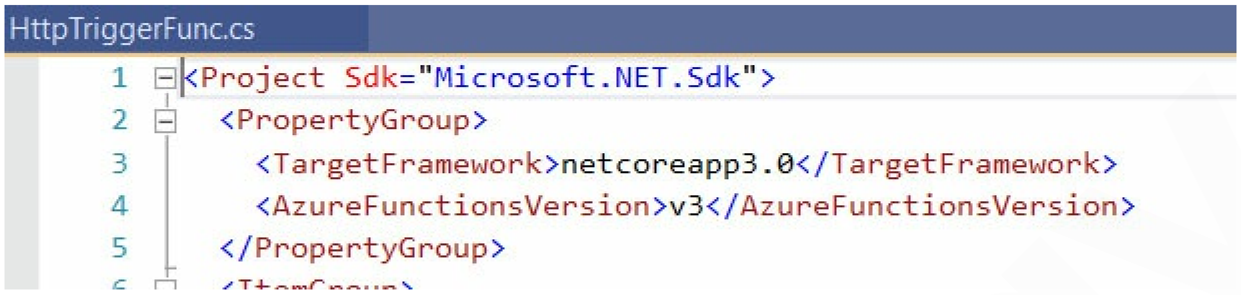
Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'CoretoAzure' (1 of 1 project)
  CoretoAzure
    Dependencies
      Frameworks
        Microsoft.NETCore.App
      Packages
        Microsoft.NET.Sdk.Functions (3.0.1)
    .gitignore
    host.json
    HttpTriggerFunc.cs
    local.settings.json
```

Figure 7.5

Click on **`Project`**, and you can verify the runtime and framework:



HttpTriggerFunc.cs

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netcoreapp3.0</TargetFramework>
    <AzureFunctionsVersion>v3</AzureFunctionsVersion>
  </PropertyGroup>
```

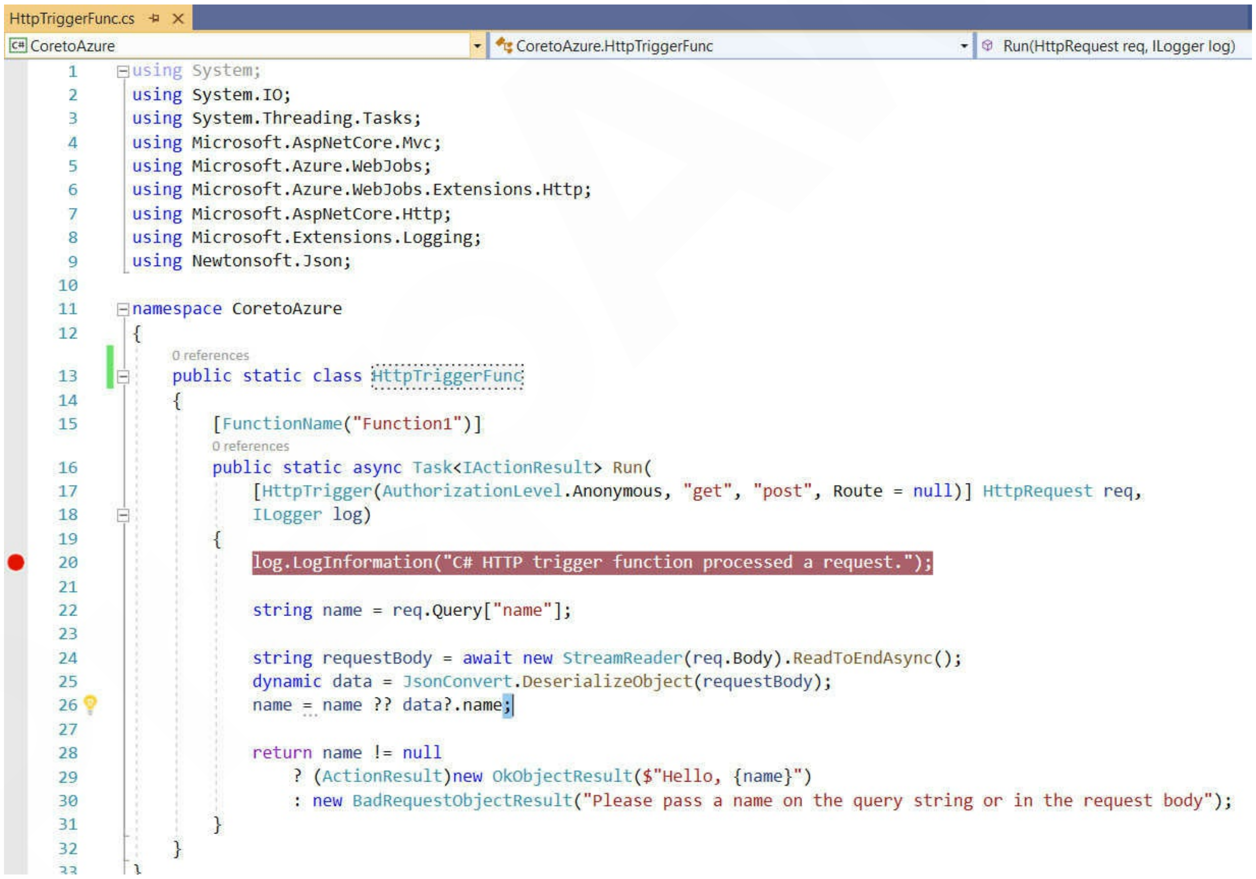
Figure 7.6

We have now successfully created a new `HTTPtriggered` function app using our favorite IDE Visual Studio 2019.

Debugging C# Azure functions on a local environment using Visual Studio 2019

One of the most common problems that developers face while developing any application on their local environment is that *everything works fine on my local machine but not on the production environment*. Now, you no need to worry about this while dealing with Azure function development. The Azure functions Runtime provided by the Azure CLI tools are exactly the same as the runtime available on Azure Cloud. You can always use and trigger an Azure service running on the cloud even during local development.

Let's debug the code, open `HttpTriggerFunc.cs` file and add a breakpoint by pressing *F9*:

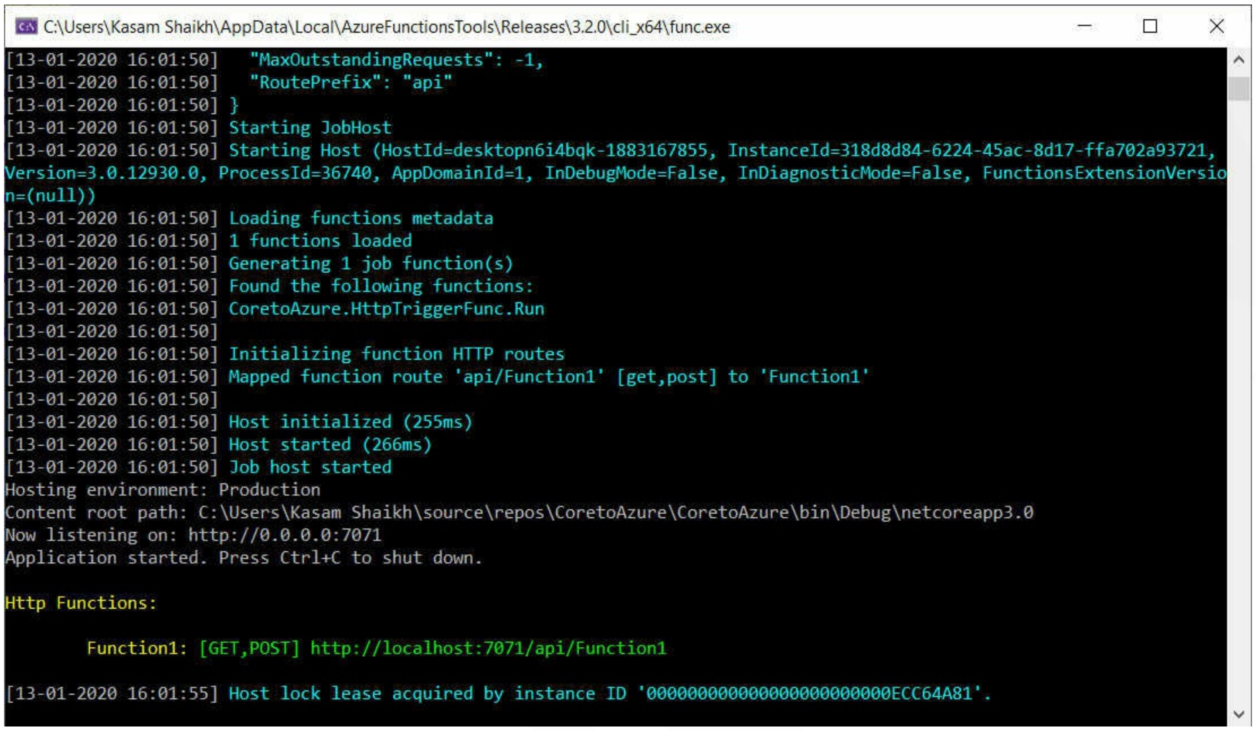


HttpTriggerFunc.cs
CoretoAzure
CoretoAzure.HttpTriggerFunc
Run(HttpRequest req, ILogger log)

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;

namespace CoretoAzure
{
    0 references
    public static class HttpTriggerFunc
    {
        [FunctionName("Function1")]
        0 references
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,
            ILogger log)
        {
            log.LogInformation("C# HTTP trigger function processed a request.");

            string name = req.Query["name"];

            string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
            dynamic data = JsonConvert.DeserializeObject(requestBody);
            name = name ?? data?.name;

            return name != null
                ? (ActionResult)new OkObjectResult($"Hello, {name}")
                : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
        }
    }
}
```

Figure 7.7

Build and run the project. Once the project starts, a job host will be created and started. It starts monitoring the requests on a specific port for all the functions of our function app. Can be seen in the following screenshot:

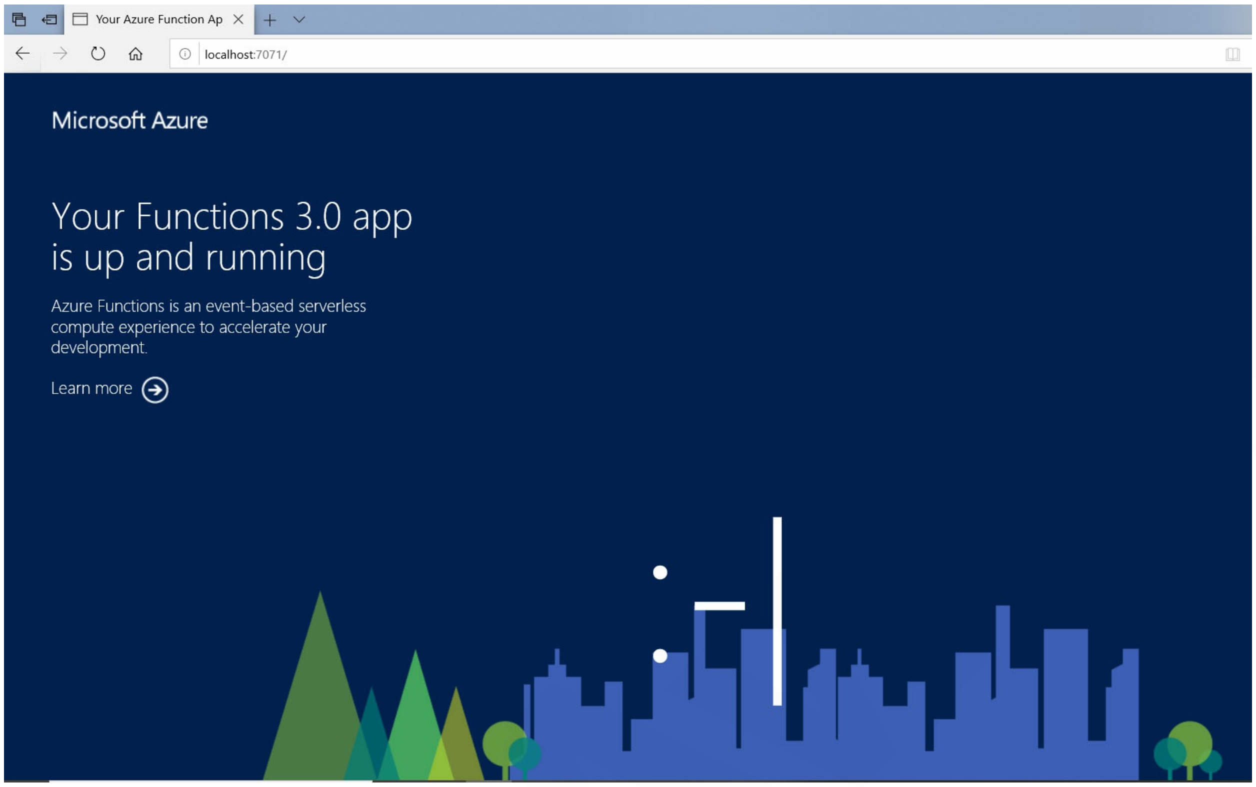


C:\Users\Kasam Shaikh\AppData\Local\AzureFunctionsTools\Releases\3.2.0\cli_x64\func.exe

```
[13-01-2020 16:01:50]   "MaxOutstandingRequests": -1,
[13-01-2020 16:01:50]   "RoutePrefix": "api"
[13-01-2020 16:01:50] }
[13-01-2020 16:01:50] Starting JobHost
[13-01-2020 16:01:50] Starting Host (HostId=desktopn6i4bqk-1883167855, InstanceId=318d8d84-6224-45ac-8d17-ffa702a93721,
Version=3.0.12930.0, ProcessId=36740, AppDomainId=1, InDebugMode=False, InDiagnosticMode=False, FunctionsExtensionVersio
n=(null))
[13-01-2020 16:01:50] Loading functions metadata
[13-01-2020 16:01:50] 1 functions loaded
[13-01-2020 16:01:50] Generating 1 job function(s)
[13-01-2020 16:01:50] Found the following functions:
[13-01-2020 16:01:50] CoretoAzure.HttpTriggerFunc.Run
[13-01-2020 16:01:50]
[13-01-2020 16:01:50] Initializing function HTTP routes
[13-01-2020 16:01:50] Mapped function route 'api/Function1' [get,post] to 'Function1'
[13-01-2020 16:01:50]
[13-01-2020 16:01:50] Host initialized (255ms)
[13-01-2020 16:01:50] Host started (266ms)
[13-01-2020 16:01:50] Job host started
Hosting environment: Production
Content root path: C:\Users\Kasam Shaikh\source\repos\CoretoAzure\CoretoAzure\bin\Debug\netcoreapp3.0
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.

Http Functions:

        Function1: [GET,POST] http://localhost:7071/api/Function1

[13-01-2020 16:01:55] Host lock lease acquired by instance ID '0000000000000000000000000ECC64A81'.
```

Figure 7.8

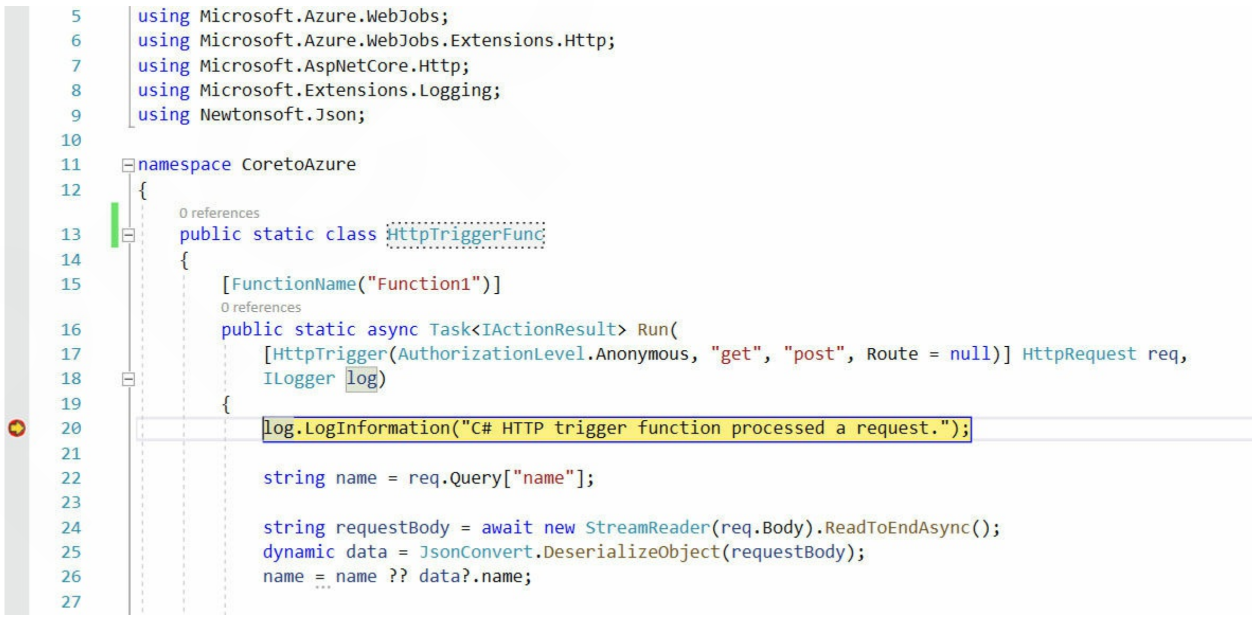
Copy the URL and hit in the browser, **http://localhost:7071**



Your Functions 3.0 app
is up and running

Azure Functions is an event-based serverless compute experience to accelerate your development.

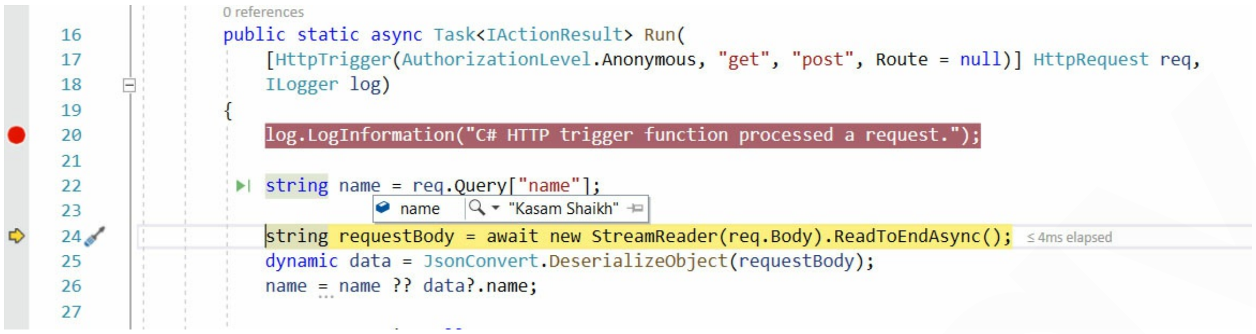
Figure 7.9



```
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;

namespace CoretoAzure
{
    0 references
    public static class HttpTriggerFunc
    {
        [FunctionName("Function1")]
        0 references
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,
            ILogger log)
        {
            log.LogInformation("C# HTTP trigger function processed a request.");

            string name = req.Query["name"];

            string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
            dynamic data = JsonConvert.DeserializeObject(requestBody);
            name = name ?? data?.name;

```

Figure 7.10

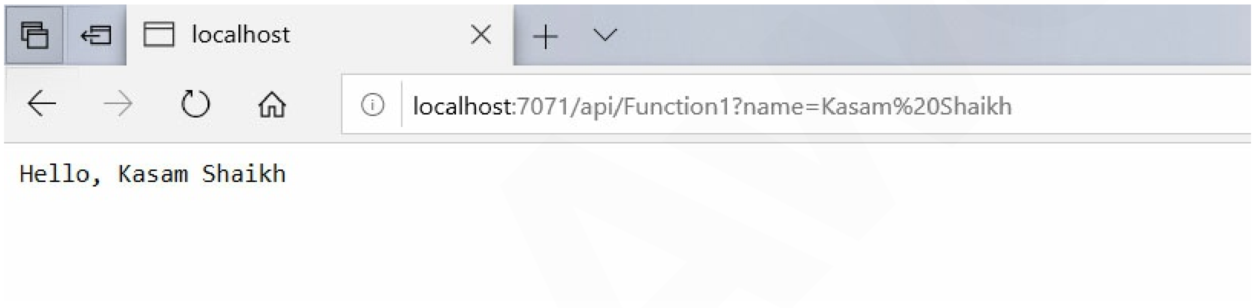
Also, moving further, you can see the passed variable:



```
0 references
public static async Task<IActionResult> Run(
    [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,
    ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    string name = req.Query["name"];
        name  "Kasam Shaikh"

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();  ≤4ms elapsed
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    name = name ?? data?.name;
```

Figure 7.11

And hit continue to process the request:



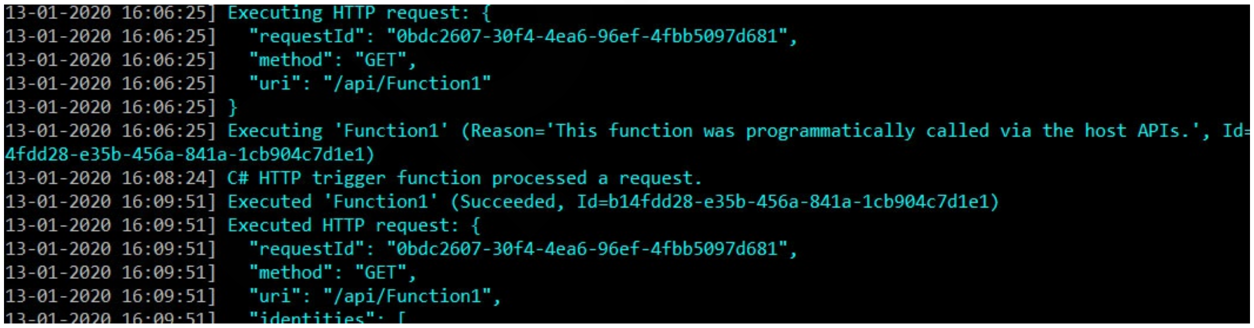
localhost

localhost:7071/api/Function1?name=Kasam%20Shaikh

```
Hello, Kasam Shaikh
```

Figure 7.12

You can see the processed request in Azure CLI:



```
13-01-2020 16:06:25] Executing HTTP request: {
13-01-2020 16:06:25]   "requestId": "0bdc2607-30f4-4ea6-96ef-4fbb5097d681",
13-01-2020 16:06:25]   "method": "GET",
13-01-2020 16:06:25]   "uri": "/api/Function1"
13-01-2020 16:06:25] }
13-01-2020 16:06:25] Executing 'Function1' (Reason='This function was programmatically called via the host APIs.', Id=
4fdd28-e35b-456a-841a-1cb904c7d1e1)
13-01-2020 16:08:24] C# HTTP trigger function processed a request.
13-01-2020 16:09:51] Executed 'Function1' (Succeeded, Id=b14fdd28-e35b-456a-841a-1cb904c7d1e1)
13-01-2020 16:09:51] Executed HTTP request: {
13-01-2020 16:09:51]   "requestId": "0bdc2607-30f4-4ea6-96ef-4fbb5097d681",
13-01-2020 16:09:51]   "method": "GET",
13-01-2020 16:09:51]   "uri": "/api/Function1",
13-01-2020 16:09:51]   "identities": [
```

Figure 7.13

Note: I haven't yet given any reference to Azure keys or strings anywhere in the code.

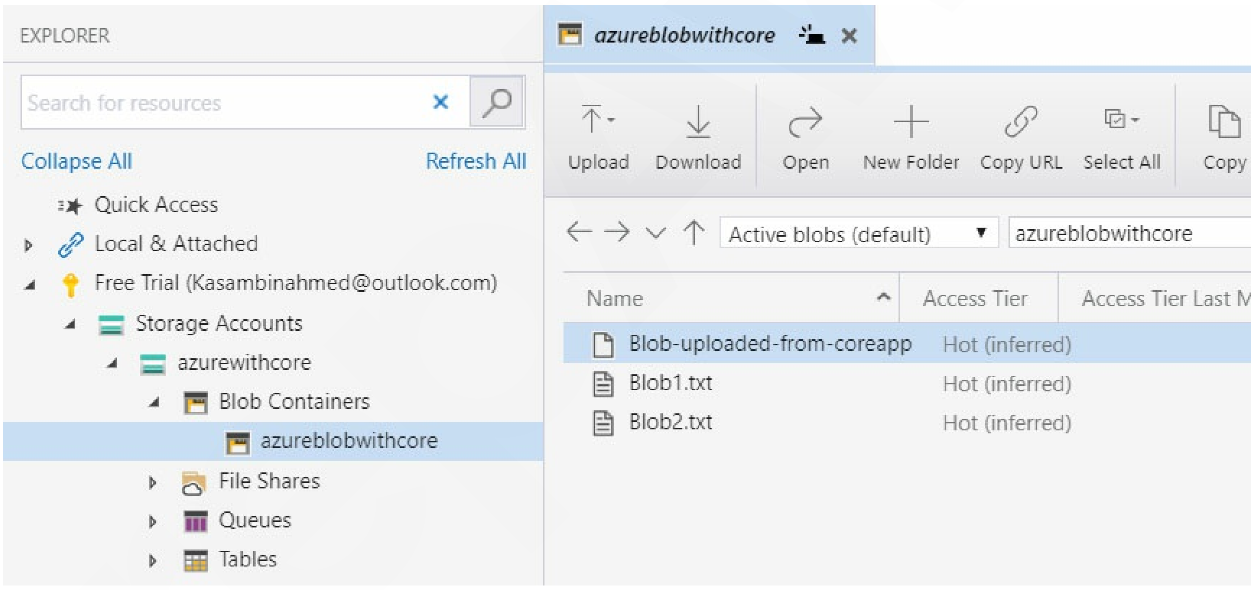
Also, when working with Visual Studio, you can have precompiled functions that can be added as a reference to other applications. On doing so, there will not only be a boost in performance and but also the best reusability of code too.

Talk with Azure Storage

Here I will walk you through steps detailing on how to connect with Azure storage from your local dev environment. In the first part of this chapter, I created Azure functions locally and ran it locally. Now, we will trigger the local function of an event occurring on Azure. To simplify more, when a Blob is created in the storage account, your function will get triggered on the local environment. Consider a scenario where you need to test your function locally on an event happening on the cloud, before getting it deployed on the production environment.

For this exercise, I will create a Storage account and a container. I will be using the same azure account and container I used in earlier chapters.

Also, will be using Azure **`Storage Explorer`**:



Name	Access Tier	Access Tier Last M
Blob-uploaded-from-coreapp	Hot (inferred)	
Blob1.txt	Hot (inferred)	
Blob2.txt	Hot (inferred)	

Figure 7.14

Now, let's create a new function in our **`CoreToAzure`** function project.

In **`Solution Explorer`** |right click on **`project | Click Add | New Azure`** function:

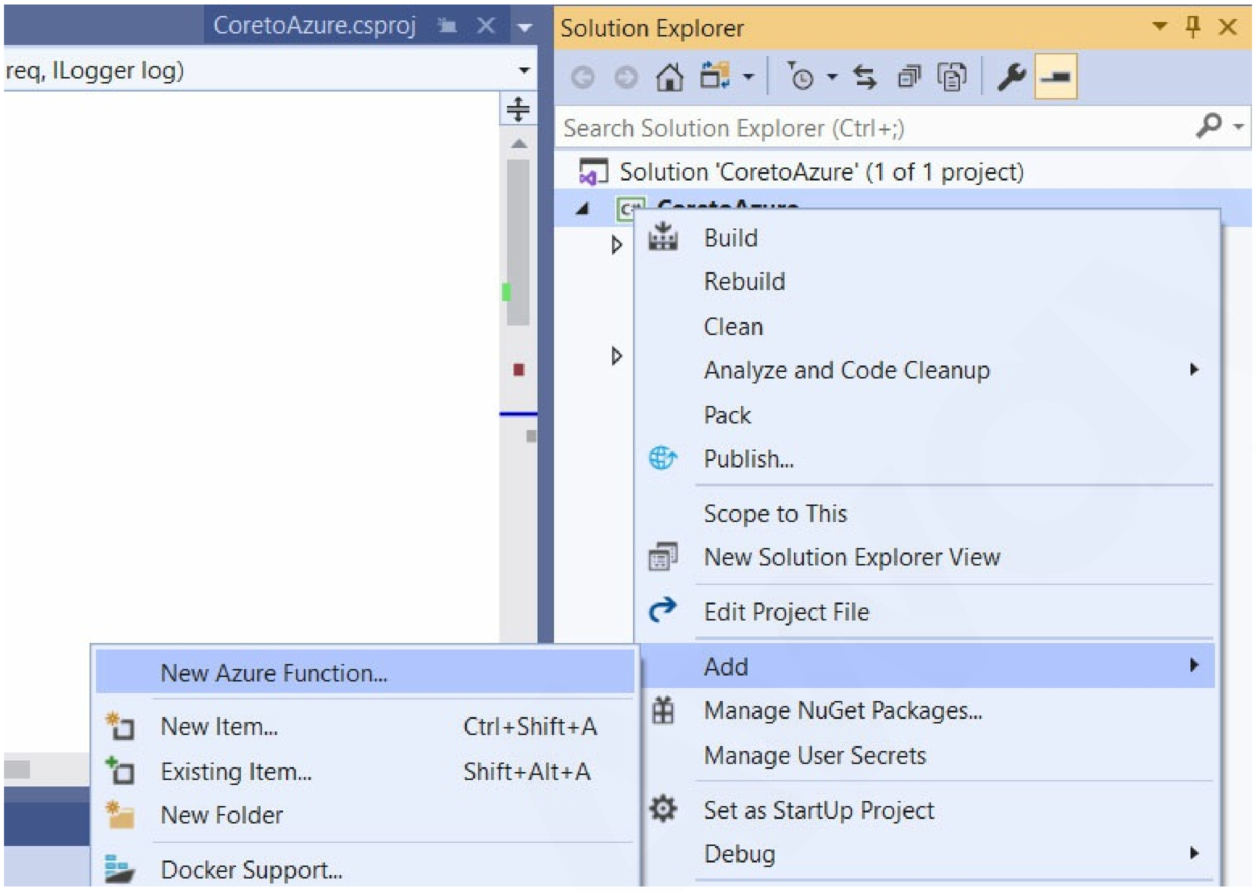
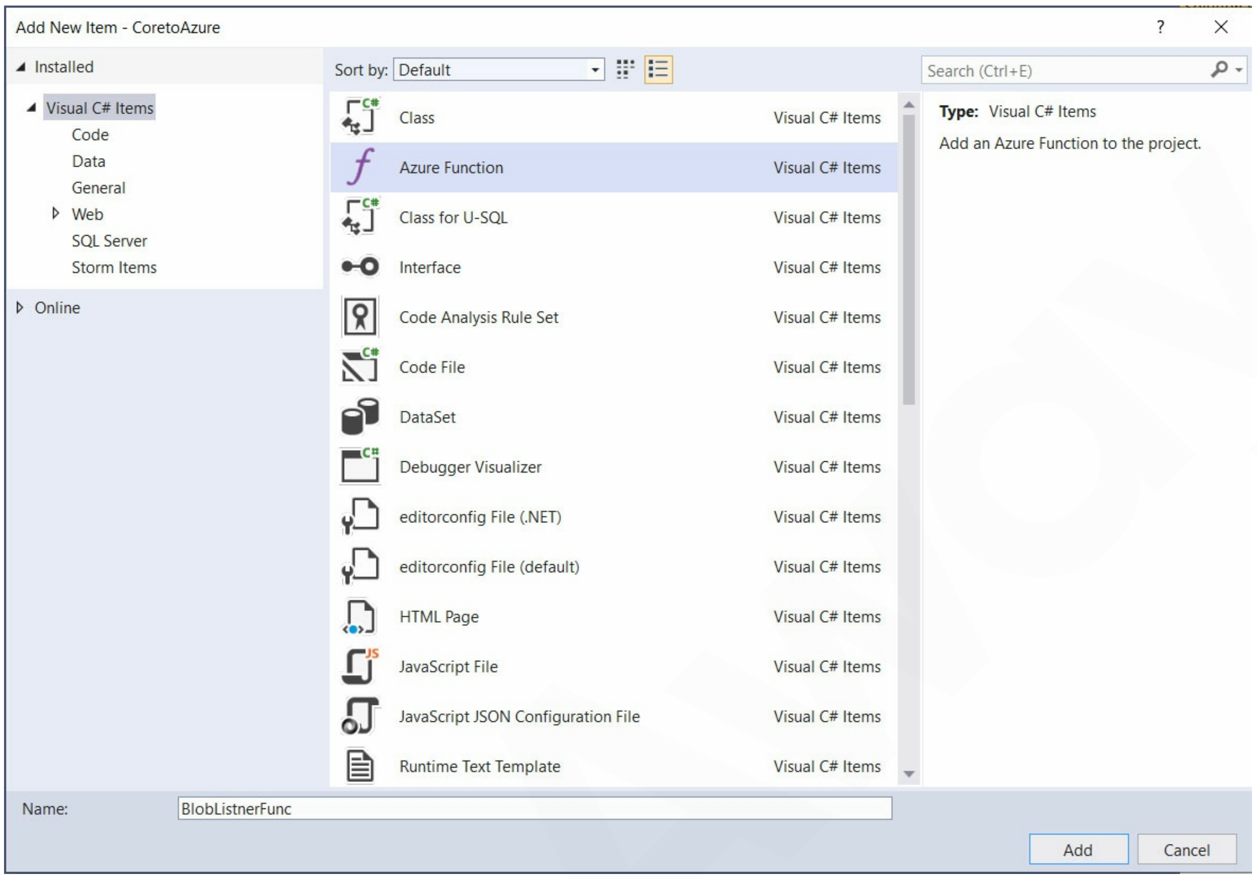


Figure 7.15

Select **`Azure Function`** and enter a nice cool name, Click on **`Add`** to proceed:


Add New Item - CoretoAzure
?
Installed
Visual C# Items
Code
Data
General
Web
SQL Server
Storm Items
Online
Sort by: Default
Search (Ctrl+E)
Class — Visual C# Items
Azure Function — Visual C# Items
Class for U-SQL — Visual C# Items
Interface — Visual C# Items
Code Analysis Rule Set — Visual C# Items
Code File — Visual C# Items
DataSet — Visual C# Items
Debugger Visualizer — Visual C# Items
editorconfig File (.NET) — Visual C# Items
editorconfig File (default) — Visual C# Items
HTML Page — Visual C# Items
JavaScript File — Visual C# Items
JavaScript JSON Configuration File — Visual C# Items
Runtime Text Template — Visual C# Items
Add
Cancel

Type: Visual C# Items

Add an Azure Function to the project.

Name: BlobListnerFunc

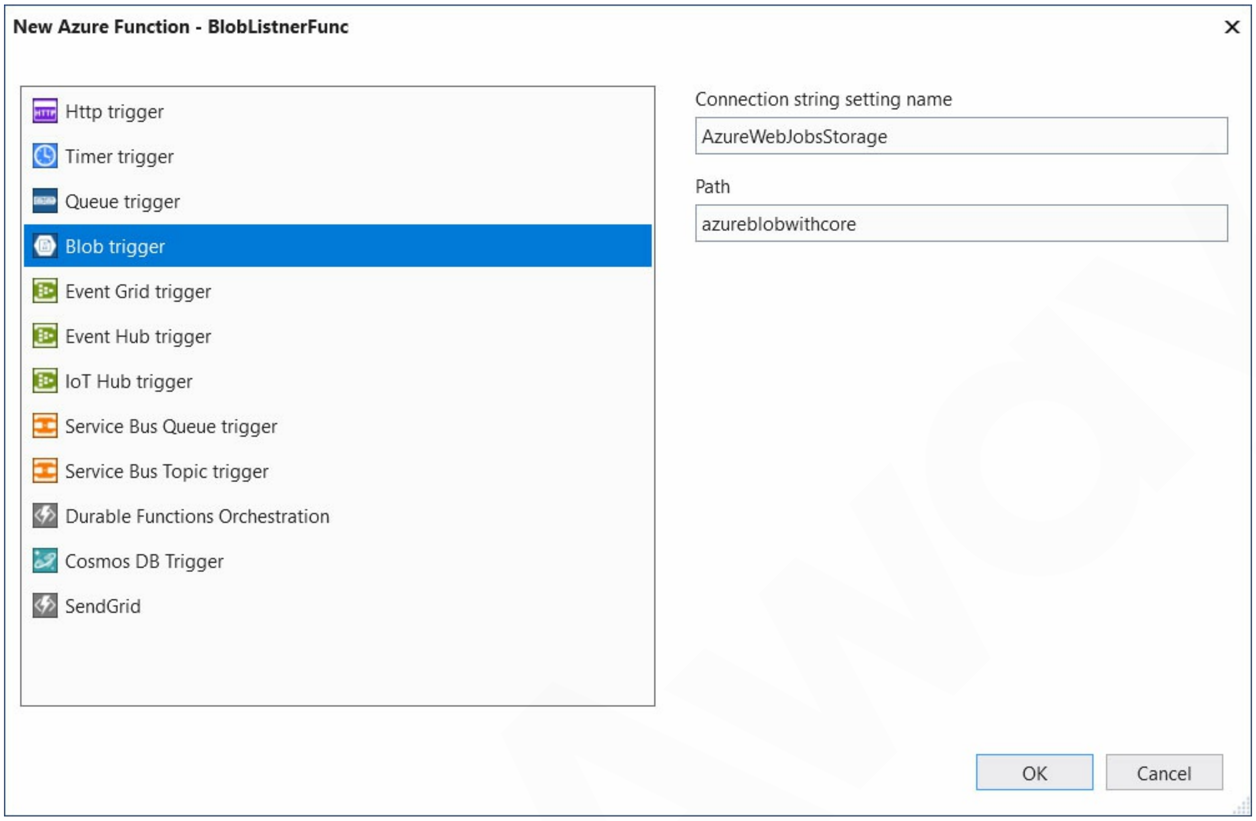
Figure 7.16

Select the **`Blob Trigger`** from the available list of functions.

Once you select the **`Blob Trigger`** function, you will be presented by:

- **`Connection String Setting Name:`** `AzureWebJobsStorage`, will be adding this sooner.
- **`Path:`** Name of the container from your storage account `azureblobwithCore`.

Click on **`Ok`** to proceed.



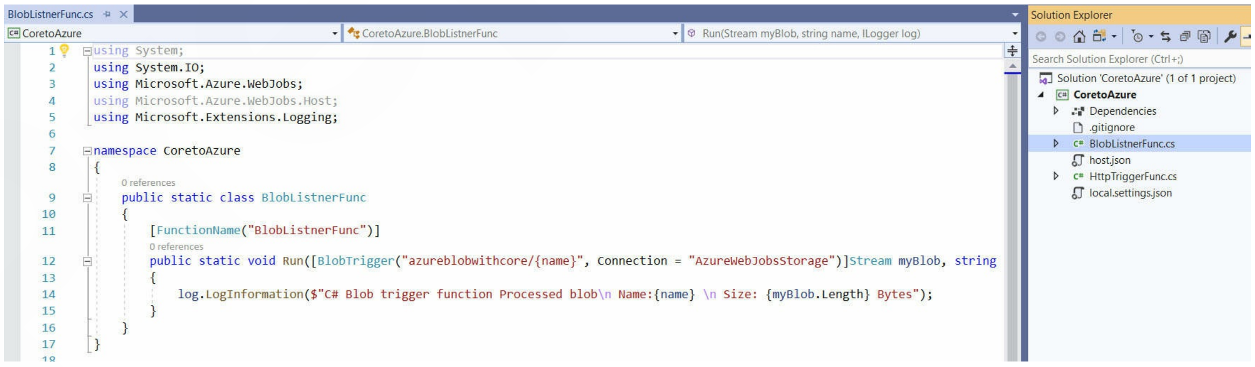
Connection string setting name: AzureWebJobsStorage

Path: azureblobwithcore

Figure 7.17

This will create the function. As you can see the code, shows the Blob name and size added to the storage account. Simple and clean code to start with.

Later you can append the code the way you want. Say, this file gets added to Blob, and here you will write a code to create a thumbnail screenshot of the file added or anything similar to it.

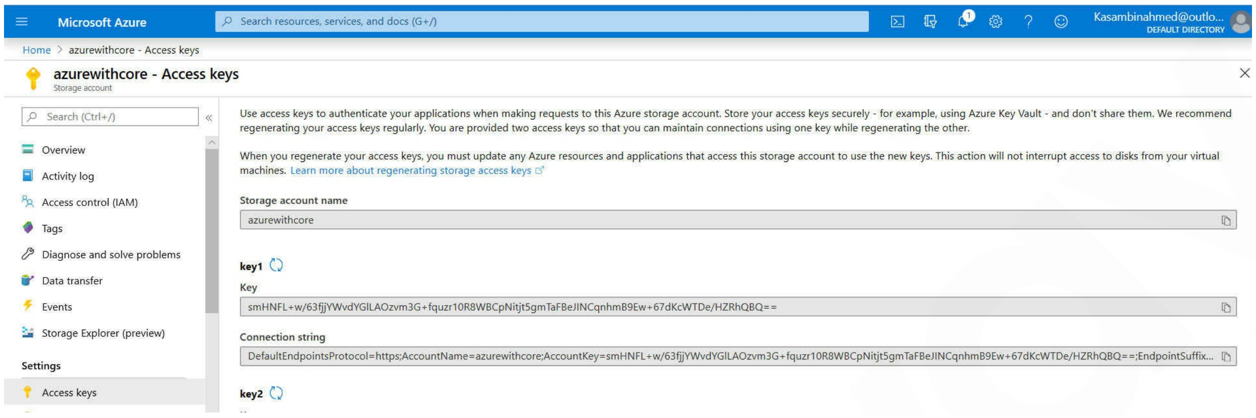

BlobListnerFunc.cs
CoretoAzure
CoretoAzure.BlobListnerFunc
Run(Stream myBlob, string name, ILogger log)
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'CoretoAzure' (1 of 1 project)
CoretoAzure
Dependencies
.gitignore
BlobListnerFunc.cs
host.json
HttpTriggerFunc.cs
local.settings.json

```
using System;
using System.IO;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Extensions.Logging;

namespace CoretoAzure
{
    0 references
    public static class BlobListnerFunc
    {
        [FunctionName("BlobListnerFunc")]
        0 references
        public static void Run([BlobTrigger("azureblobwithcore/{name}", Connection = "AzureWebJobsStorage")]Stream myBlob, string
        {
            log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Length} Bytes");
        }
    }
}
```

Figure 7.18

As the name suggests, connection string, you need to provide the connection string of Azure Storage account.

Navigate to Azure Storage account created in **`Azure Portal|`**from left pane under **`Settings`**|click on **`Access Keys`** and **`Copy the Connection String`**.



Use access keys to authenticate your applications when making requests to this Azure storage account. Store your access keys securely - for example, using Azure Key Vault - and don't share them. We recommend regenerating your access keys regularly. You are provided two access keys so that you can maintain connections using one key while regenerating the other.

When you regenerate your access keys, you must update any Azure resources and applications that access this storage account to use the new keys. This action will not interrupt access to disks from your virtual machines. Learn more about regenerating storage access keys

Storage account name
azurewithcore

key1

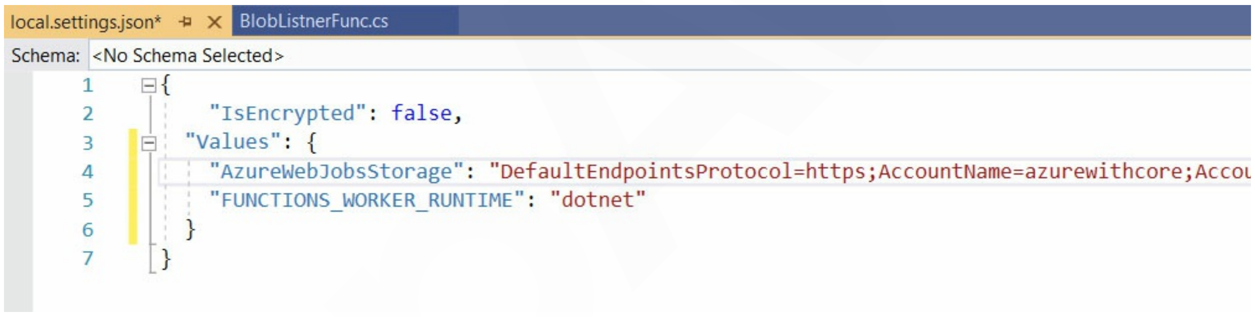
Key
smHNFL+w/63fjjYWvdYGlLAOzvm3G+fquzr10R8WBCpNitjt5gmTaFBeJINCqnhmB9Ew+67dKcWTDe/HZRhQBQ==

Connection string
DefaultEndpointsProtocol=https;AccountName=azurewithcore;AccountKey=smHNFL+w/63fjjYWvdYGlLAOzvm3G+fquzr10R8WBCpNitjt5gmTaFBeJINCqnhmB9Ew+67dKcWTDe/HZRhQBQ==;EndpointSuffix...

key2

Figure 7.19

Copy the connection string to local.settings.json file residing in the project root folder:

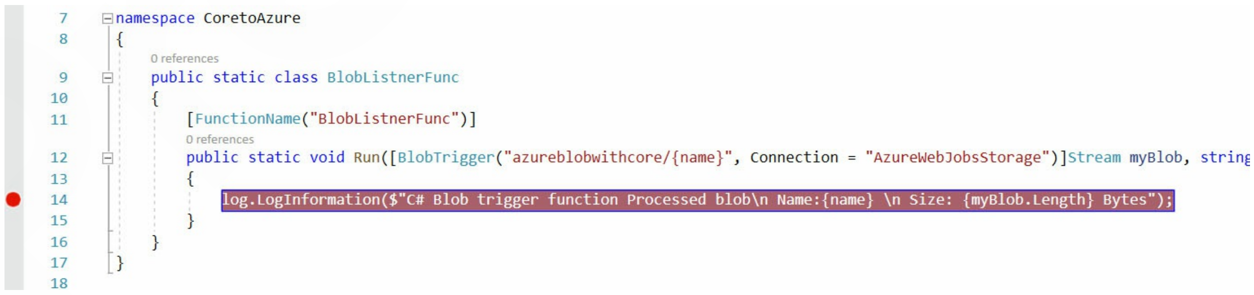


```
{
    "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=azurewithcore;Accou
    "FUNCTIONS_WORKER_RUNTIME": "dotnet"
  }
}
```

Figure 7.20

Time to test!

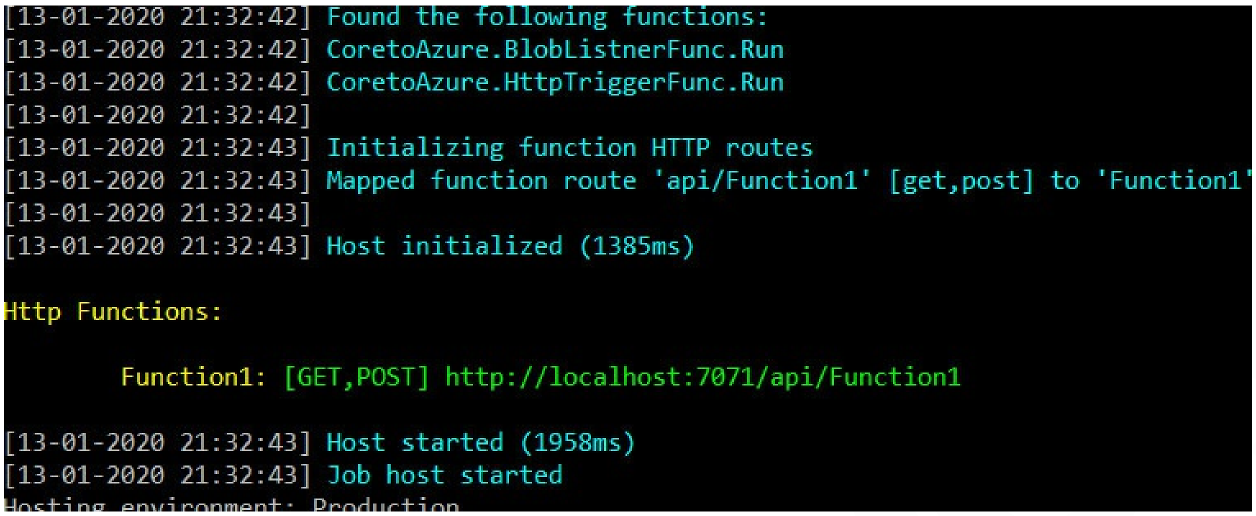
Let's have a breakpoint in the newly created `BlobListnerFunc` function:



```
namespace CoretoAzure
{
    0 references
    public static class BlobListnerFunc
    {
        [FunctionName("BlobListnerFunc")]
        0 references
        public static void Run([BlobTrigger("azureblobwithcore/{name}", Connection = "AzureWebJobsStorage")]Stream myBlob, string
        {
            log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Length} Bytes");
        }
    }
}
```

Figure 7.21

And press *F5*, to start the job host:



```
[13-01-2020 21:32:42] Found the following functions:
[13-01-2020 21:32:42] CoretoAzure.BlobListnerFunc.Run
[13-01-2020 21:32:42] CoretoAzure.HttpTriggerFunc.Run
[13-01-2020 21:32:42]
[13-01-2020 21:32:43] Initializing function HTTP routes
[13-01-2020 21:32:43] Mapped function route 'api/Function1' [get,post] to 'Function1'
[13-01-2020 21:32:43]
[13-01-2020 21:32:43] Host initialized (1385ms)

Http Functions:

        Function1: [GET,POST] http://localhost:7071/api/Function1

[13-01-2020 21:32:43] Host started (1958ms)
[13-01-2020 21:32:43] Job host started
Hosting environment: Production
```

Figure 7.22

Let's add a **`Blob`** file in the container. I am using here Azure **`Storage Explorer`** to do so:

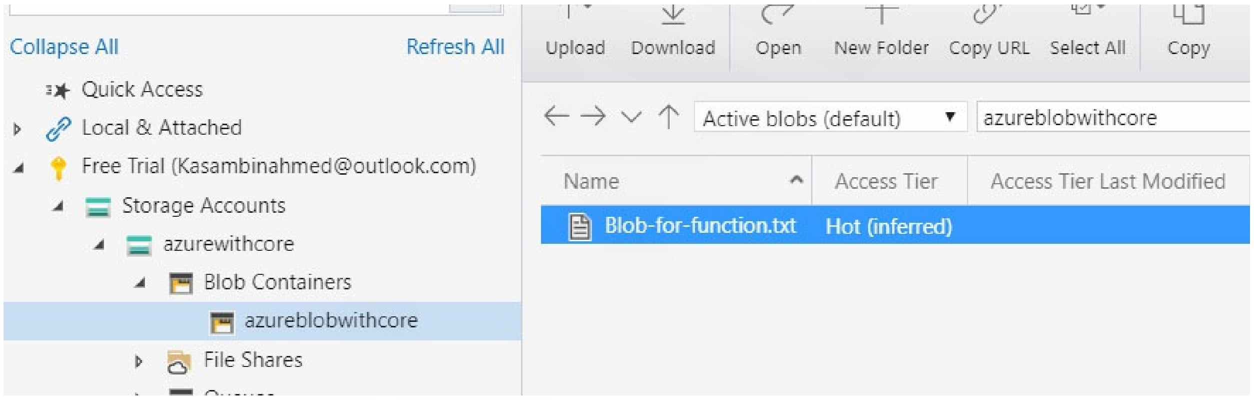
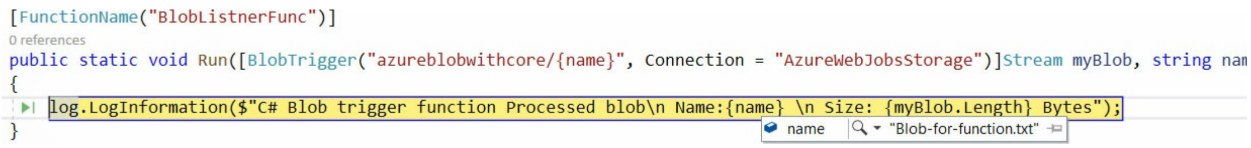


Figure 7.23

As soon as the blob is added to the container, the debugger will hit the debug point. Think, the Blob being added to Cloud Storage account residing to some other geolocation, and adding of the same triggered your function in the local environment:



```
[FunctionName("BlobListnerFunc")]
0 references
public static void Run([BlobTrigger("azureblobwithcore/{name}", Connection = "AzureWebJobsStorage")]Stream myBlob, string nam
{
    log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Length} Bytes");
}
```

Figure 7.24

When I started to host the job, it monitors the Azure Storage container using the connection string provided via `AzureWebJobsStorage`.When blobs get added to the container, it triggers the function.

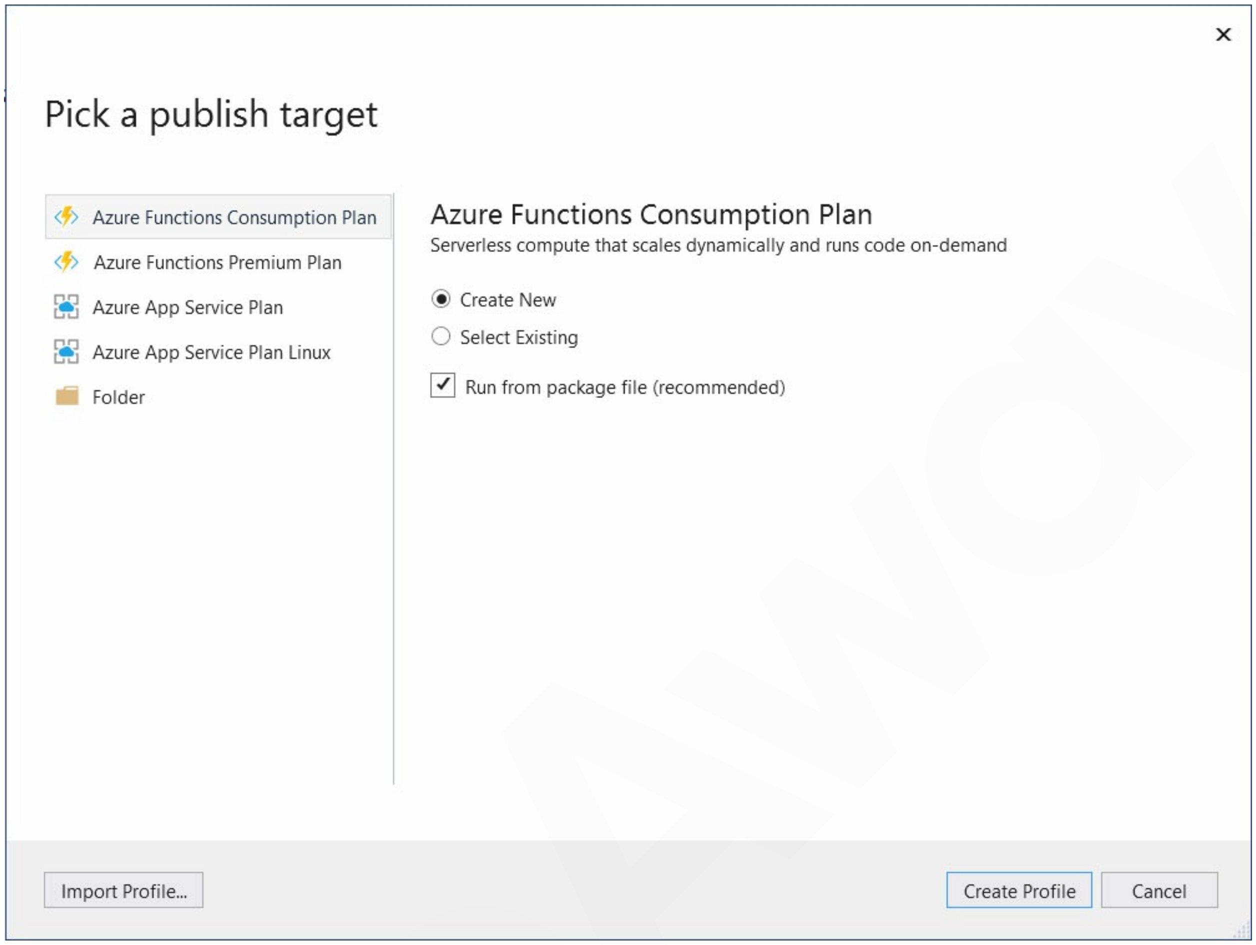
Move on Cloud

Now, you created the Azure function and tested it. It’s to deploy the function to the Azure function app. Azure function app is part of the Azure App service and comes with an amazing pricing option, Consumption plan. This plan ensures to get a bill only on execution time. Yes, when an Azure function is executed, you will be charged only for that period of time, unlike App service monthly plan. Albeit, you can run select App service plan too for having an Azure function.

We will be deploying the code using Visual Studio 2019. You can configure the required resources such as pricing plan, resource group, location, and so on from Visual Studio.

To start with, in **`Solution Explorer`**|right click on `project`|and click on **`Publish`**.

It will present you with an option to select the resource you need to deploy the code:


Pick a publish target
Azure Functions Consumption Plan
Azure Functions Premium Plan
Azure App Service Plan
Azure App Service Plan Linux
Folder
Import Profile...
Create Profile
Cancel

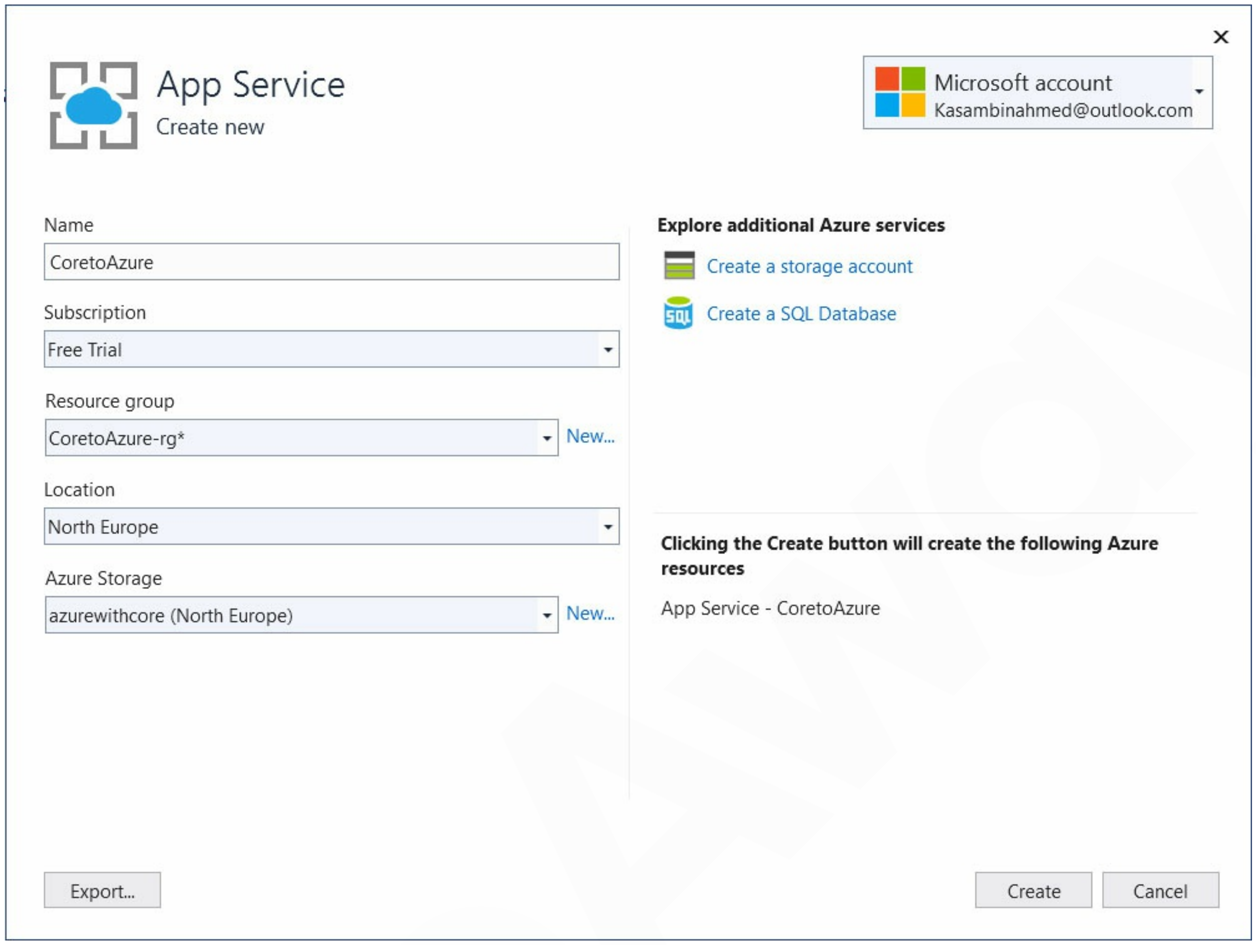
Azure Functions Consumption Plan
Serverless compute that scales dynamically and runs code on-demand

● Create New
○ Select Existing
☑ Run from package file (recommended)

Figure 7.25

Select **`Azure Functions with Consumption Plan`**. Create a new one as we are doing it for the first time. Click on **`Create Profile`**.

You will be presented with the window to provide the required details:



App Service
Create new

Microsoft account
Kasambinahmed@outlook.com

Name: CoretoAzure

Subscription: Free Trial

Resource group: CoretoAzure-rg* New...

Location: North Europe

Azure Storage: azurewithcore (North Europe) New...

Explore additional Azure services

Create a storage account

Create a SQL Database

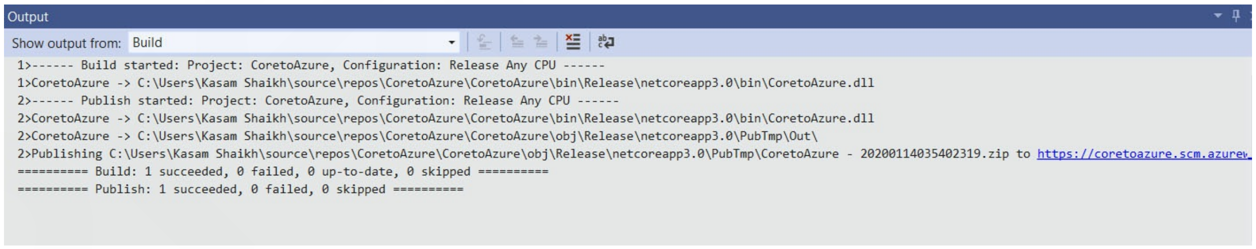
Clicking the Create button will create the following Azure resources

App Service - CoretoAzure

Figure 7.26

Click on **`Create`** to proceed.

Within fewer minutes, the function will complete its deployment.

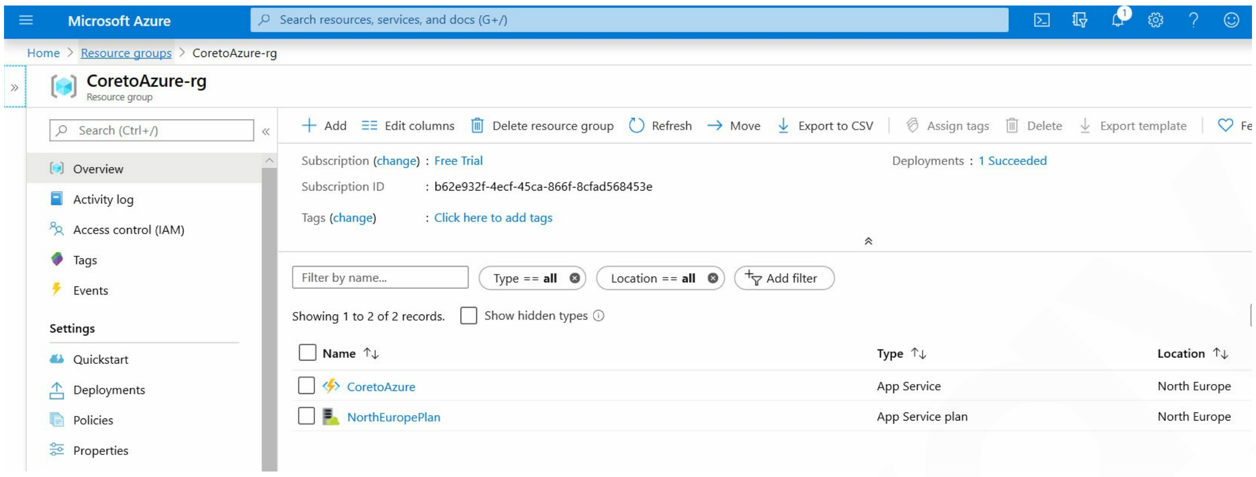


Show output from: Build

```
1>------ Build started: Project: CoretoAzure, Configuration: Release Any CPU ------
1>CoretoAzure -> C:\Users\Kasam Shaikh\source\repos\CoretoAzure\CoretoAzure\bin\Release\netcoreapp3.0\bin\CoretoAzure.dll
2>------ Publish started: Project: CoretoAzure, Configuration: Release Any CPU ------
2>CoretoAzure -> C:\Users\Kasam Shaikh\source\repos\CoretoAzure\CoretoAzure\bin\Release\netcoreapp3.0\bin\CoretoAzure.dll
2>CoretoAzure -> C:\Users\Kasam Shaikh\source\repos\CoretoAzure\CoretoAzure\obj\Release\netcoreapp3.0\PubTmp\Out\
2>Publishing C:\Users\Kasam Shaikh\source\repos\CoretoAzure\CoretoAzure\obj\Release\netcoreapp3.0\PubTmp\CoretoAzure - 20200114035402319.zip to https://coretoazure.scm.azurew
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
```

Figure 7.27

Go to **`Azure Portal`** to verify the newly created resources:



Subscription (change) : Free Trial

Subscription ID : b62e932f-4ecf-45ca-866f-8cfad568453e

Tags (change) : Click here to add tags

Deployments : 1 Succeeded

Showing 1 to 2 of 2 records.

Name	Type	Location
CoretoAzure	App Service	North Europe
NorthEuropePlan	App Service plan	North Europe

Figure 7.28

Click on the **`Function`** app, and you can see all the functions being created:

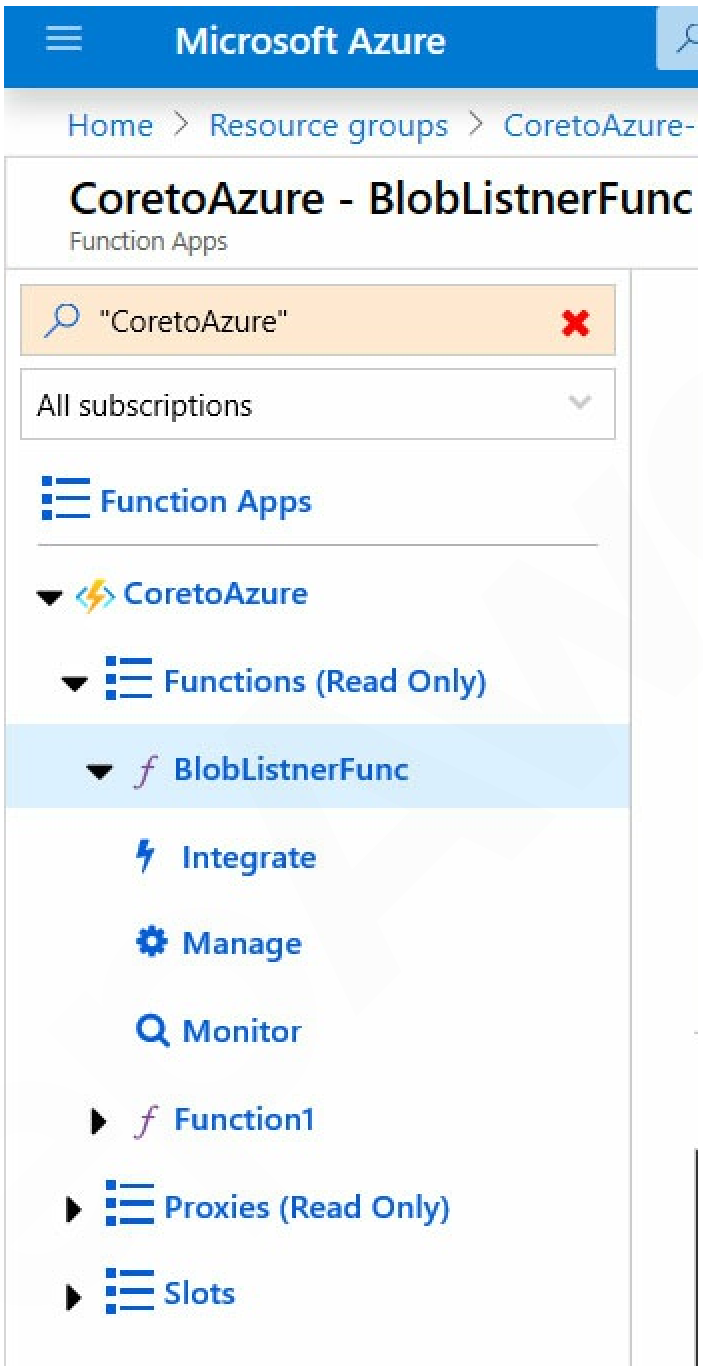
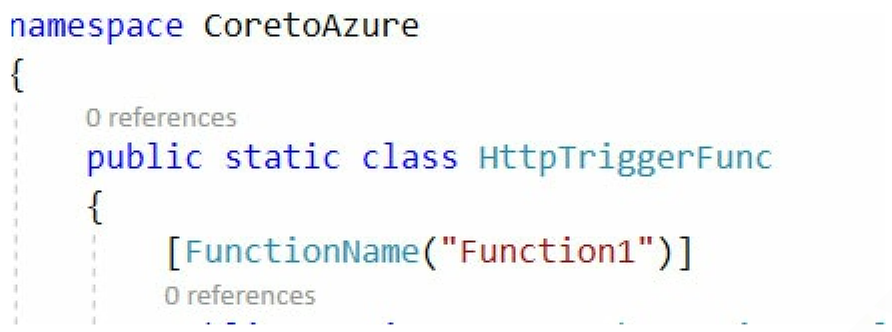


Figure 7.29

Now you must be thinking why the function **`a=name`** here is as `Function1` when I had already renamed it.

This is because; I didn't change the name over here:



```
namespace CoretoAzure
{
    0 references
    public static class HttpTriggerFunc
    {
        [FunctionName("Function1")]
        0 references
```

Figure 7.30

Make a note of this, a good lesson to learn, isn’t it!

To test the function on Azure

To test the function, do add a blob file name `Blob2.txtin` configured **`Storage Accounts`**. For ease, we will do it by Azure storage explorer.

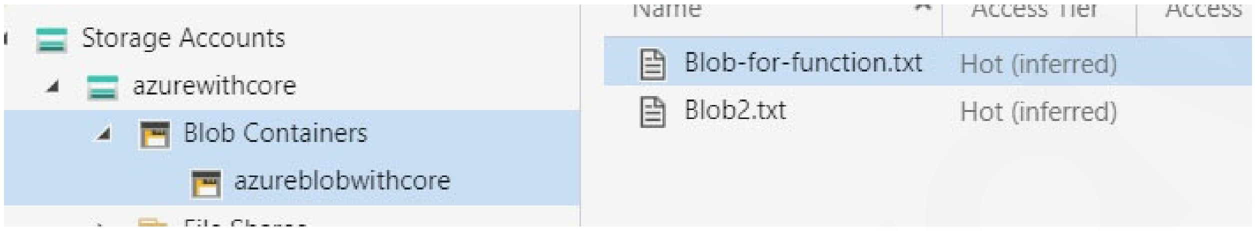
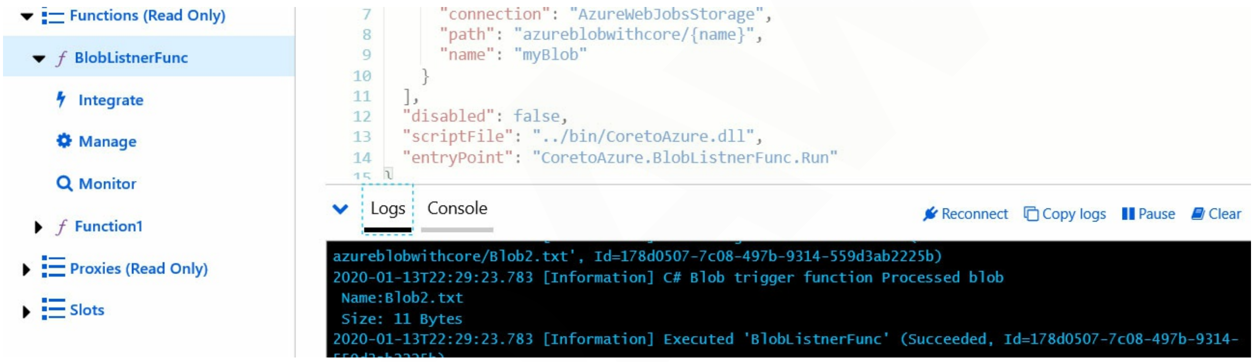


Figure 7.31

Now check the logs in Azure portal for `BlobListnerfunc`:



```
        "connection": "AzureWebJobsStorage",
        "path": "azureblobwithcore/{name}",
        "name": "myBlob"
      }
    ],
    "disabled": false,
    "scriptFile": "../bin/CoretoAzure.dll",
    "entryPoint": "CoretoAzure.BlobListnerFunc.Run"
```

```
azureblobwithcore/Blob2.txt', Id=178d0507-7c08-497b-9314-559d3ab2225b)
2020-01-13T22:29:23.783 [Information] C# Blob trigger function Processed blob
 Name:Blob2.txt
 Size: 11 Bytes
2020-01-13T22:29:23.783 [Information] Executed 'BlobListnerFunc' (Succeeded, Id=178d0507-7c08-497b-9314-
```

Figure 7.32

To verify the other function, click on the **`Function1`**|click on **`Get Function URL`**:

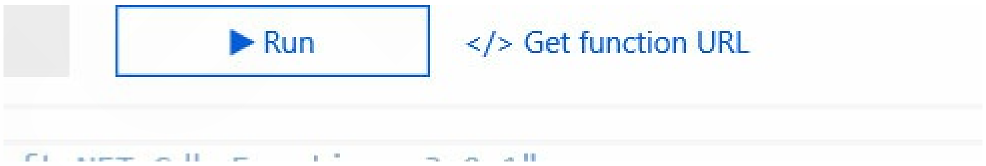
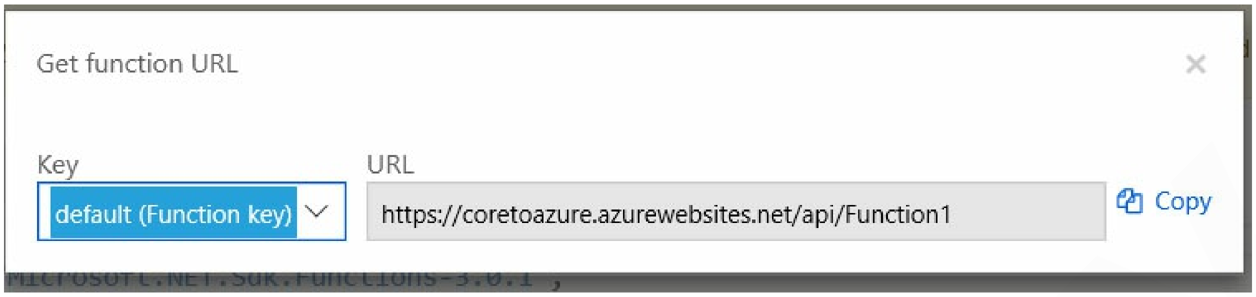


Figure 7.33

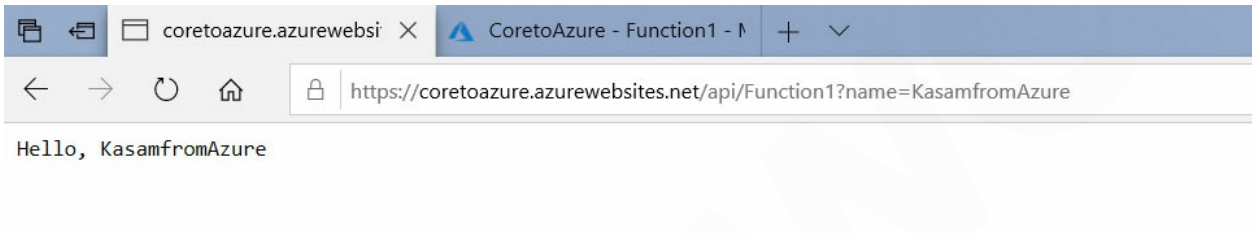
Clicking on **`Get Function URL`** will present you with a public URL you can call this function with. Copy the **`URL`**:



```
https://coretoazure.azurewebsites.net/api/Function1
```

Figure 7.34

And open it in the browser, passing the required parameter:



```
https://coretoazure.azurewebsites.net/api/Function1?name=KasamfromAzure
Hello, KasamfromAzure
```

Figure 7.35

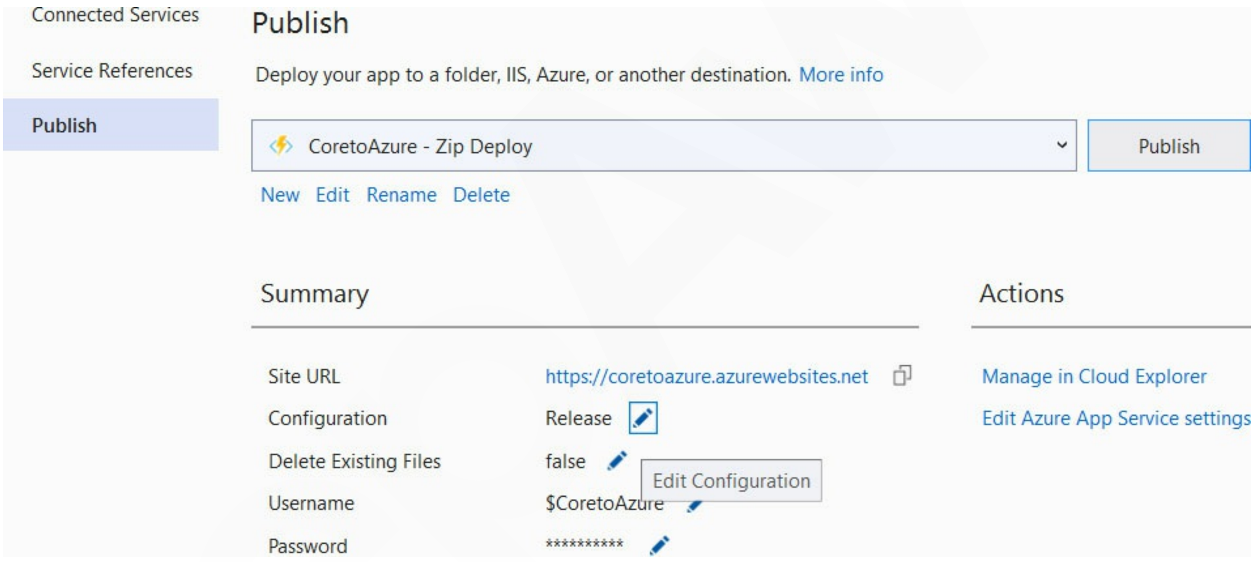
Live debugging of Azure function

Till now, I created an Azure function, tested it locally, connected to remote Azure event, and then deployed it to the Azure function App. I tested the public URL, and it worked fine as well. Now it's time to debug the live Azure function hosted on Azure Cloud using Visual Studio 2019.

To achieve the same, you need to have some tweaks in the earlier steps.

Very first, change the configuration for publishing your code from Release to Debug. You can achieve it by doing a right-click on **`Function`** project and click on **`Publish`**.

It presents with the publish window with all configurational details.



Deploy your app to a folder, IIS, Azure, or another destination. More info

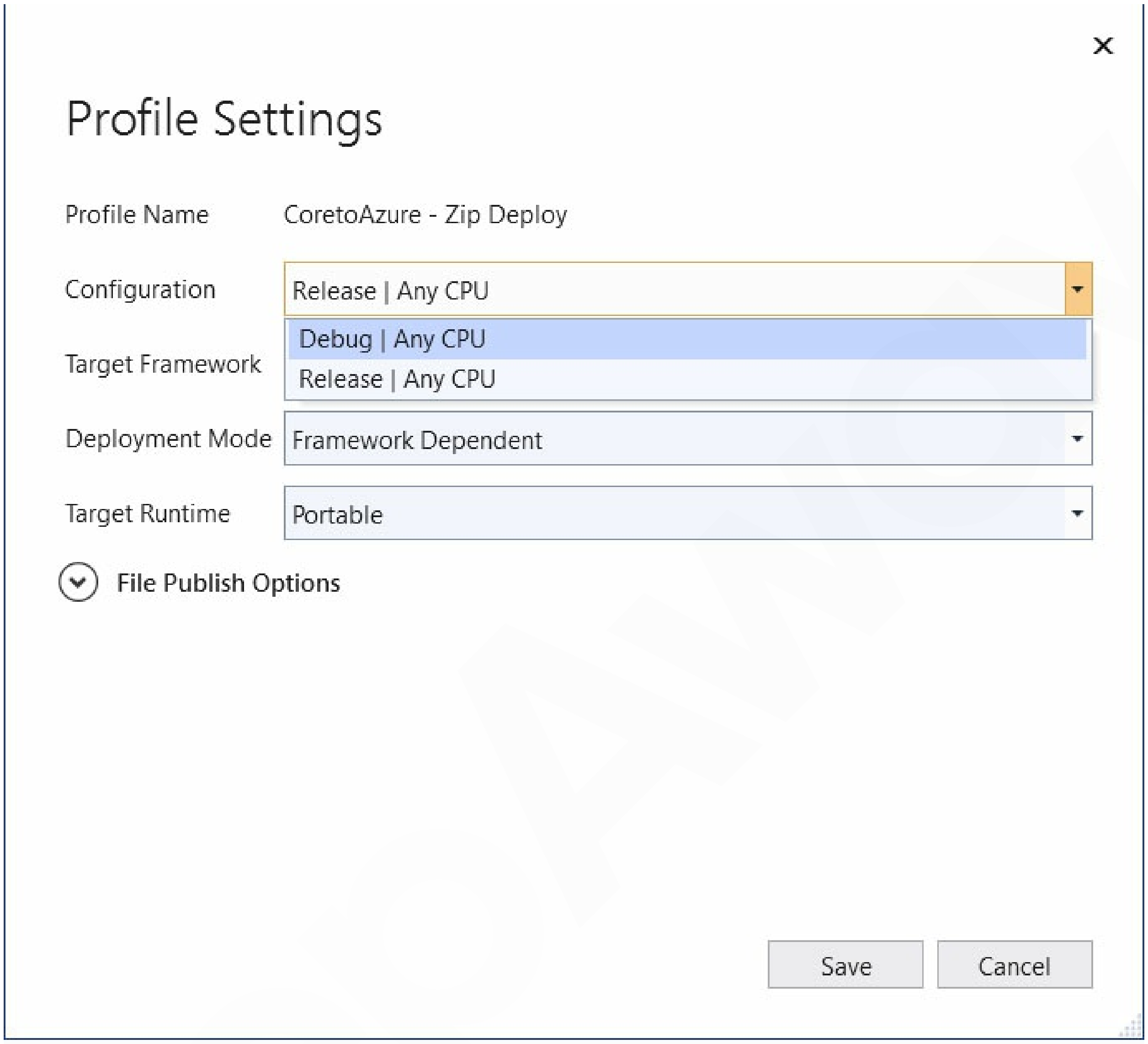
CoretoAzure - Zip Deploy

Site URL	https://coretoazure.azurewebsites.net
Configuration	Release
Delete Existing Files	false
Username	$CoretoAzure
Password	**********

Figure 7.36

Click on the **`Edit Configuration`** icon, and you will be presented with a window, to change the configuration.

Select **`Debug | Any CPU`** from the dropdown presented for Configuration.


Profile Settings

Profile Name	CoretoAzure - Zip Deploy
Configuration	Release | Any CPU
Target Framework	
Deployment Mode	Framework Dependent
Target Runtime	Portable

Figure 7.37

Click on **`Save`** and **`Publish`** the code.

Next, open up server explorer and connect with your Azure subscription. Once connected, it will list all the services associated with the subscription.

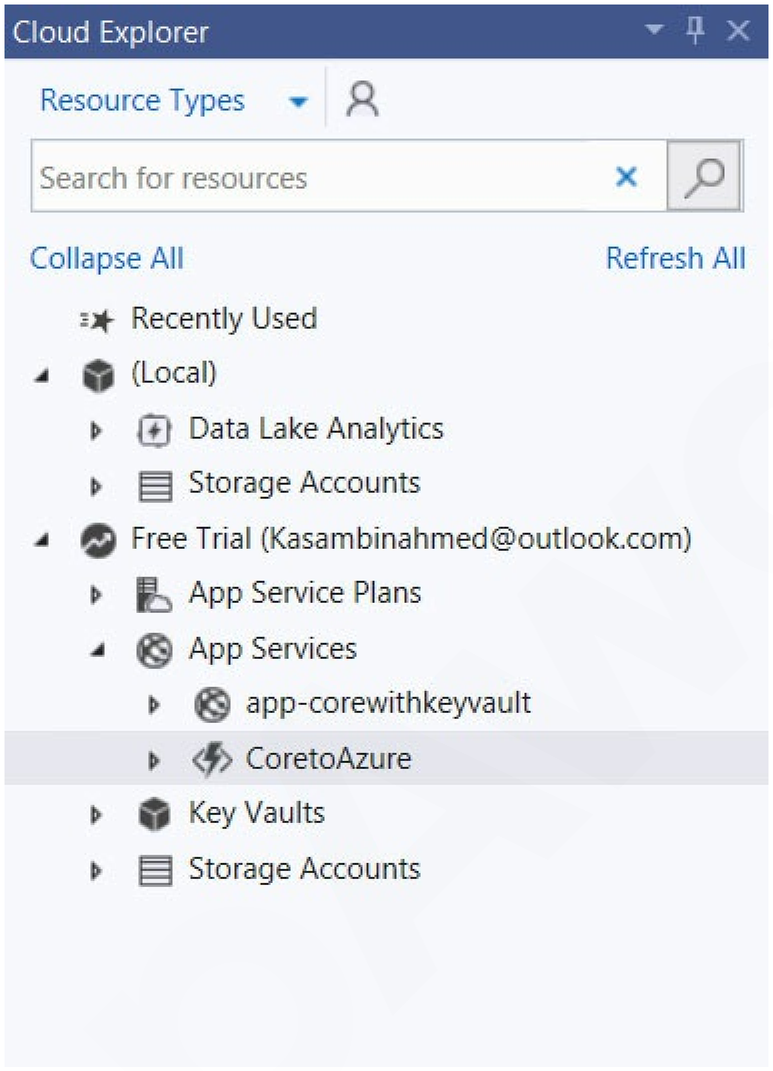


Figure 7.38

Right-click on the `Function` app `CoretoAzure` and click on `Attach Debugger`:

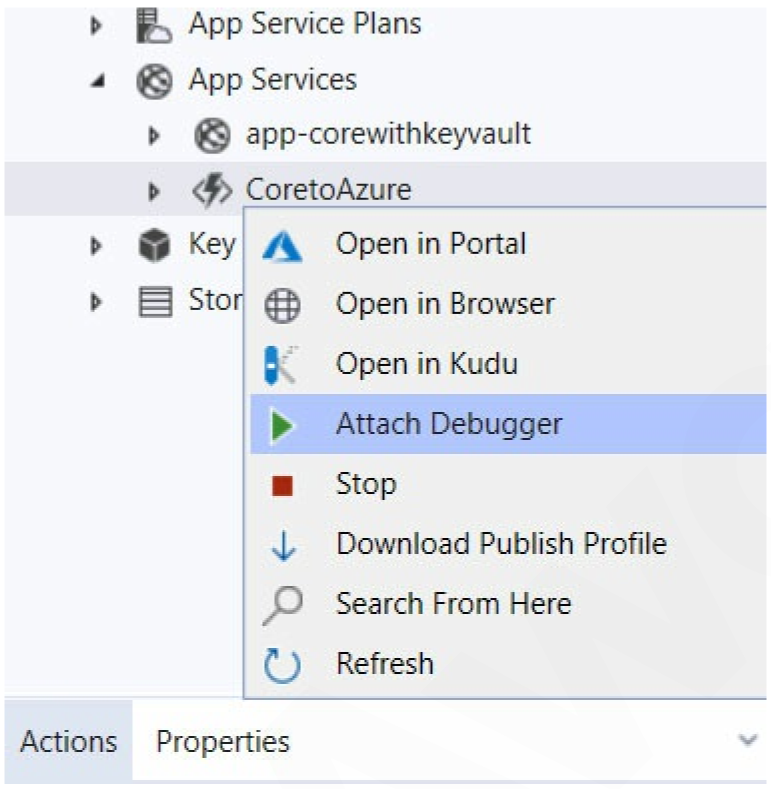


Figure 7.39

The process will take fewer minutes to enable remote debugging:

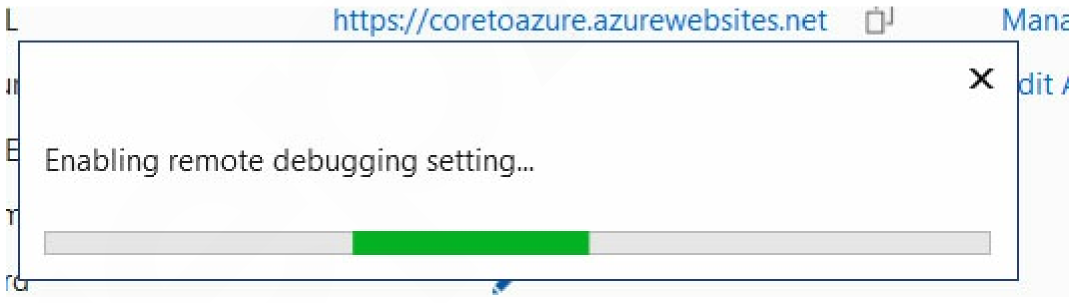
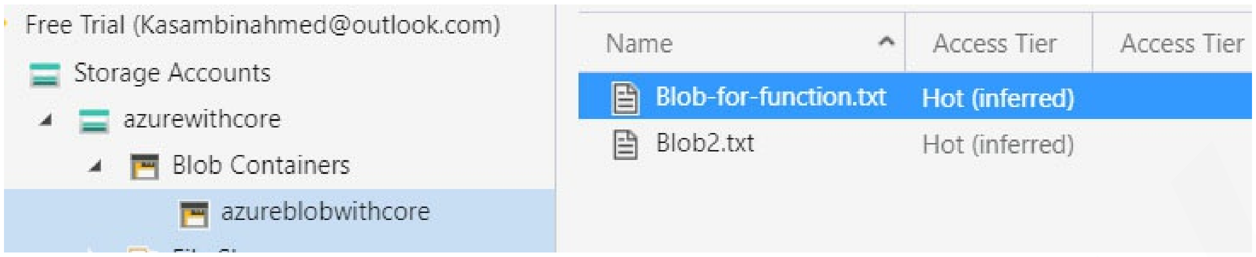

https://coretoazure.azurewebsites.net
Mana
dit
Enabling remote debugging setting...

Figure 7.40

The window will be closed once it gets enables.

I have kept the debugging on in the function file for `Blobtrigger`.

Add the blob with name `Blob-for-function`, in the storage using Azure storage explorer:

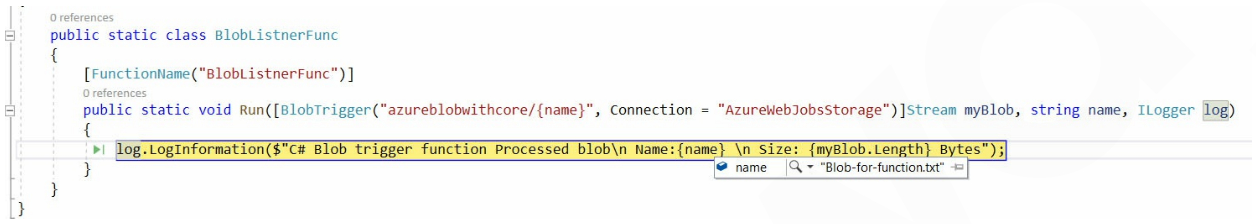


Free Trial (Kasambinahmed@outlook.com)
Storage Accounts
azurewithcore
Blob Containers
azureblobwithcore

Name	Access Tier	Access Tier
Blob-for-function.txt	Hot (inferred)	
Blob2.txt	Hot (inferred)	

Figure 7.41

And you can see debugger will hit the debug point:



```
0 references
public static class BlobListnerFunc
{
    [FunctionName("BlobListnerFunc")]
    0 references
    public static void Run([BlobTrigger("azureblobwithcore/{name}", Connection = "AzureWebJobsStorage")]Stream myBlob, string name, ILogger log)
    {
        log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Length} Bytes");
    }
}
}
```

name "Blob-for-function.txt"

Figure 7.42

And here we are, debugging Azure function Live.

Conclusion

In this chapter, you learned about Azure Serverless offering, Azure functions. And about working with Azure functions using Visual Studio 2019. We had a development in the local environment, later on, Azure Cloud. You also learned how seamlessly with a few steps, you can debug the azure function hosted on Cloud. Again, I will advise you to make your hands dirty with Azure functions. Happy Azure learning.

Questions

1. Explain Serverless?
2. What are Bindings & Triggers in Azure Function?
3. Can we have a stateful application in Azure Functions?
4. What are Durable Functions?
5. What are the different programming languages supported by Azure Functions?
6. Explain the Consumption Plan option in Pricing tier for Azure Functions?
7. Can you debug Azure Function deployed on Cloud?