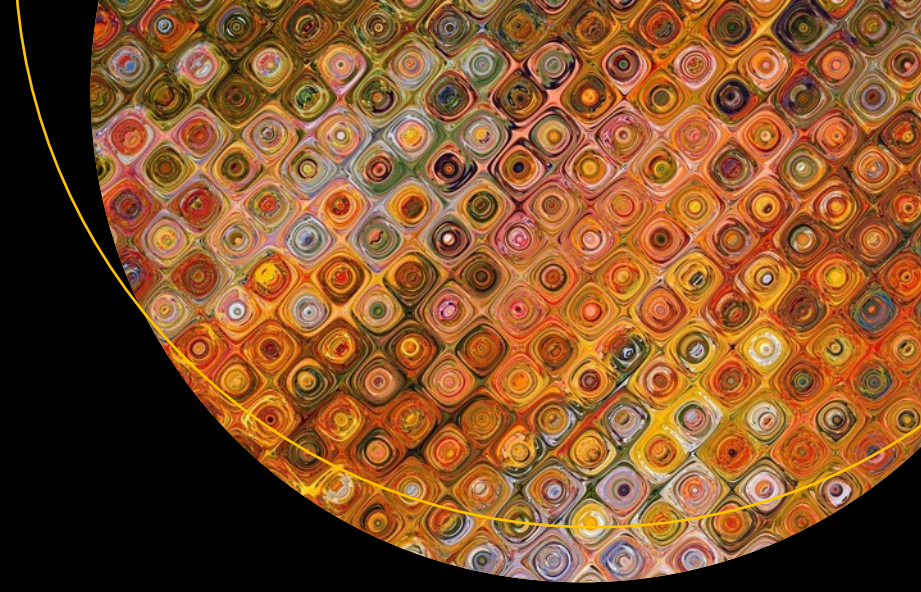

Hands-on Azure Functions with C#

Build Function as a Service (FaaS) Solutions

Ashirwad Satapathi
Abhishek Mishra

Apress®

Hands-on Azure Functions with C#

Build Function as a Service (FaaS) Solutions

Ashirwad Satapathi
Abhishek Mishra

Apress®

Hands-on Azure Functions with C#: Build Function as a Service (FaaS) Solutions

Ashirwad Satapathi
Gajapati, Odisha, India

Abhishek Mishra
Mumbai, Maharashtra, India

ISBN-13 (pbk): 978-1-4842-7121-6
ISBN-13 (electronic): 978-1-4842-7122-3
https://doi.org/10.1007/978-1-4842-7122-3

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Smriti Srivastava
Development Editor: Laura Berendson
Coordinating Editor: Shrikant Vishwakarma

Cover designed by eStudioCalamar

Cover image designed by Pexels

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 1 New York Plaza, Suite 4600, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-7121-6. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

This book is dedicated to my father, Mr. Upendra Satapathi, and mother, Mrs. Sabita Panigrahi, for supporting me through each and every phase of my life. Without your support I wouldn't have been able to complete this book.

—Ashirwad Satapathi

This book is dedicated to my super dad, Mr. Balabhardra Mishra, and loving mom, Mrs. Pragyan Mishra.

—Abhishek Mishra

Table of Contents

About the Authors

Ashirwad Satapathi works as a software developer with a leading IT firm and has expertise in building scalable applications with .NET Core. He has a deep understanding of building full-stack applications using .NET Core along with Azure PaaS and serverless offerings. He is an active blogger in the C# Corner developer community. He was awarded the C# Corner MVP in September 2020 for his contributions to the developer community.

Abhishek Mishra is an architect with a leading multinational software company and has deep expertise in designing and building enterprise-grade intelligent Azure and .NET-based architectures. He is an expert in .NET full stack, Azure (PaaS, IaaS, serverless), infrastructure as code, Azure machine learning, intelligent Azure (Azure Bot Services and Cognitive Services), and robotics process automation. He has a rich 15+ years of experience working in top organizations in the industry. He loves blogging and is an active blogger in the C# Corner developer community. He was awarded the C# Corner MVP in December 2018, December 2019, and December 2020 for his contributions to the developer community.

About the Technical Reviewer

Carsten Thomsen is primarily a back-end developer but works with smaller front-end bits as well. He has authored and reviewed a number of books and created numerous Microsoft Learning courses, all focused on software development. He works as a freelancer/contractor in various countries in Europe, using Azure, Visual Studio, Azure DevOps, and GitHub. He is an exceptional troubleshooter, asking the right questions in a most logical to least logical fashion; he also enjoys working in the areas of architecture, research, analysis, development, testing, and bug fixing. Carsten is a good communicator with great mentoring and team-lead skills and is skilled at researching and presenting new material.

Acknowledgments

We would like to thank the Apress team for giving us the opportunity to work on this book. Also thanks to the technical reviewer and the editors for helping us deliver this manuscript.

Introduction

Azure Functions is a function as a service (FaaS) offering on the Azure Platform. In this book, you will explore Azure Functions in detail and learn how to work with Azure Functions using a practical and example-based approach that will help you grasp the subject with ease.

The book will start with the essential topics. You will learn how to set up the application development environment for Azure Functions. Then you will get example-based steps for building a serverless solution using a combination of bindings and triggers in C#. The book will then dive into areas that will help you learn how to create custom bindings, connect with various data sources, ingest telemetry data for Azure Functions into Application Insights, and learn various ways to deploy the functions to the Azure environment.

You will also explore advanced areas such as running Azure Functions in an Azure Kubernetes Service cluster using Kubernetes Event Driven Autoscaling (KEDA). You will learn the DevOps way of working with Azure Functions using Azure DevOps, as well as the best practices you should follow while using Azure Functions.

This book provides production-like scenarios and provides labs that will deliver the right set of hands-on experience. The practical approach in the book will help you gain deep proficiency in the subject.

This book is intended for experienced developers, cloud architects, and tech enthusiasts looking forward to building scalable and efficient serverless solutions using Azure Functions. Anyone having a prior experience with C# and knowing the Azure basics can use this book to start their journey in building serverless solutions with Azure Functions.

CHAPTER 1

Introduction to Azure Functions

Function as a service (FaaS) is getting more popular every day on all the major cloud platforms. With FaaS, you can build small chunks of code that run for a short time and host them on the FaaS cloud offering. You get billed for the time your function runs, and you do not need to bother about the hosting infrastructure and the scaling aspects.

Microsoft Azure provides Azure Functions as an FaaS offering. You build your function code and host it on Azure Functions, part of Azure App Service. The underlying platform takes care of all the hosting and scaling needs. Executing your code on Azure Functions is cost-effective most of the time compared to other hosting services available in the cloud.

In this chapter, you will get a basic understanding of Azure Functions that will help you grasp the next set of chapters with ease.

Structure of the Chapter

In this chapter, we will explore the following aspects of Azure Functions:

- Introduction to Azure Functions
- Introduction to serverless
- Azure WebJobs vs. Azure Functions
- Advantages and disadvantages of Azure Functions
- Hosting plan for Azure Functions
- Use cases for Azure Functions

A. Satapathi and A. Mishra, *Hands-on Azure Functions with C#*, https://doi.org/10.1007/978-1-4842-7122-3_1

Objectives

After studying this chapter, you will be able to do the following:

- Understand the fundamentals of serverless computing and Azure Functions
- Identify scenarios where you can use Azure Functions

Introduction to Azure Functions

Azure Functions is a serverless computing service on the Microsoft Azure platform and is based on the FaaS computing model. You need to build your code, spin up a function, and host your code on Azure Functions. The underlying cloud platform manages the hosting infrastructure and hosting software. You do not need to worry about the scaling aspects of your hosted code. The underlying Azure platform manages all the scaling aspects for your code running on Azure Functions. You get billed when the function is active and doing its work. You do not get billed whenever Azure Functions is idle.

Azure Functions hosts code that runs for a short time interval. However, you can increase the execution time by choosing an appropriate hosting plan for the function. A function gets invoked and starts running using triggers. Azure Functions supports a wide range of triggers. For example, a timer can trigger a function in predefined time intervals, a new message in the Queue Storage can trigger it, or a simple HTTP call can trigger a function. Azure Functions interacts with a wide range of services, such as Blob Storage, Table Storage, Queue Storage, Event Grid, Cosmos DB, Service Bus Queue, and many more, using bindings. You can declare both the triggers and the bindings declaratively without writing any code.

Azure Functions supports three runtime versions and an array of programming languages based on the runtime you select. 3.*x* is the newest runtime, and 1.*x* is the oldest runtime available. You can build your code using any of the programming languages in Table 1-1.

Table 1-1. *Azure Functions Runtimes and Supported Programming Languages*

Language	Runtime 1.*x*	Runtime 2.*x*	Runtime 3.*x*
C#	.NET 4.7	.NET Core 2.2	.NET Core 3.1
JavaScript	Node 6	Node 10 and 8	Node 12 and 10
F#	.NET 4.7	.NET Core 2.2	.NET Core 3.1
Java	Not supported	Java 8	Java 11 and 8
PowerShell	Not supported	PowerShell Core 6	PowerShell 7 and Core 6
Python	Not supported	Python 3.7 and 3.6	Python 3.8, 3.7, and 3.6
TypeScript	Not supported	Supported	Supported

A function executes whenever it gets invoked by a trigger. The function runs for a particular time interval and gets into an idle state. It wakes up whenever it gets invoked again by a trigger. The function takes some time to get warmed up and start executing whenever it gets triggered.

Introduction to Serverless

You start getting billed for cloud services as soon as you spin them up. You get billed even if you do not use the services. Also, you need to plan and configure the scaling strategy for these services. Some services give you the flexibility to set autoscaling, and for others, you need to set the scaling configuration manually. In either case, you end up providing the necessary settings so that the services can scale.

In the serverless cloud services case, you get billed when the service is running and is executing your hosted code, and you do not get billed when the service is idle and is not executing anything. You pay the cloud vendor on an actual consumption basis, which saves you money. The underlying platform manages all the scaling aspects of your application running inside the serverless service. You need not configure any scaling settings for the serverless service. The serverless services are intelligent enough to add new instances to handle incoming traffic and remove the additional instances when the incoming traffic decreases.

Serverless does not mean that the cloud services are not hosted on any server. You cannot run any code without a server. In the case of serverless services, you do not have control over the server hosting your code. You need to bring your code and host it on the serverless services without worrying about the underlying infrastructure. The cloud vendor manages the underlying infrastructure.

The following are a few of the popular serverless offerings provided by Microsoft Azure:

- Azure Functions
- Azure Logic Apps
- Azure Event Grid
- Serverless Azure Kubernetes Service
- Serverless SQL Database

Note In the case of serverless services and platform as a service (PaaS), you can get your code and host it on the service without managing the underlying infrastructure. The cloud vendor manages the infrastructure. However, you need to manage the scaling aspects in the case of PaaS. The cloud vendor manages the scaling for the serverless service. In the case of PaaS, you get billed as soon as you spin up the service. However, in a serverless service, you get billed when the service is active and executes your code.

Azure WebJobs vs. Azure Functions

You create a WebJobs job in an App Service Plan. A web job works as a background worker for your applications hosted on Azure App Service. For example, you can host an application that facilitates users to upload files in Azure Blob Storage. Usually, these files will be in a user-specific format. Before the application processes the files, the files should be transformed into a standard format that the application can understand. In such scenarios, you can create a web job in the same App Service Plan. This web job will run as a background worker, pick up the user-uploaded file, and transform it into a format that the application can understand. Web jobs can get triggered using a wide

variety of triggers such as Azure Queue Storage, Cosmos DB, Azure Blob Storage, Azure Service Bus, Azure Event Hub, and many more. Azure WebJobs meets all the necessary developer needs for background processing. However, it shares the same App Service Plan as Azure App Service. Sharing the same App Service Plan means sharing the same underlying computing infrastructure. This sharing of the underlying infrastructure leads to performance bottlenecks at times.

Functions are not just meant to process background tasks. They can host business logic for applications as well. However, they are well suited to host code that runs for a short time interval. The functions are serverless offerings and scale independently. The underlying infrastructure manages all the scaling aspects for the function. Web jobs are tied to the Azure App Service instances and scale as and when the Azure App Service instance scales. You need to set scaling configurations explicitly for each web job. Functions can run as and when triggered using consumption-based plans, or they can run continuously using a Dedicated Plan. Web jobs are always tied to the App Service Plan that is a dedicated hosting plan. However, you are not charged separately for web jobs. They come with the App Service Plan. The Azure portal provides a browser-based editor that you can use to build, test, and deploy functions inside the Azure portal. This feature enhances the productivity of the developer. You can integrate Azure Functions with Azure Logic Apps with ease and build enterprise-grade solutions on Azure. Azure Functions supports various triggers such as HTTP WebHooks (GitHub/Slack) and Azure Event Grid that Azure WebJobs does not support.

Advantages and Disadvantages of Azure Functions

You build your code and host it on Azure Functions without worrying about the underlying hosting infrastructure. The cloud vendor takes care of all the hosting aspects such as the hosting server and the hosting software. As a developer, you get more time to focus on building your application code and working on its functionality. The underlying infrastructure scales your application without needing you to configure the scale settings. Also, you get billed when a function gets triggered and the code gets executed. This feature saves you money. You can use Azure Functions with Logic Apps and build truly enterprise-grade applications. In fact, you can integrate Azure Functions with a wide range of Azure services with ease. Azure Functions is well suited to execute code that runs for a short time interval. You can break down your application functionality into smaller chunks and host it on Azure Functions. This will help you bring in the

single responsibility pattern at a more granular level. The single responsibility pattern states that a module or a component of a software program should perform a single functionality of the program. For example, in a calculator application, you should have a component or a module that performs an add operation, a different component that performs a subtract operation, and so on. The component of the application should be designed to perform a single functionality instead of doing everything for the application.

However, functions execute when they get triggered and move into an idle state when they do not do any work. Whenever a function is idle, it will take some time for the function to spring into action whenever triggered. This is because it will take some time for the underlying infrastructure to get warmed up and start executing the code. This phenomenon is referred to as a *cold-start* issue that you must consider while designing solutions for Azure Functions. At times, Azure Functions can cost more compared to hosting your code on Azure Web App. The underlying platform spins up new instances for Azure Functions whenever the load increases, and you do not have any control over the scaling aspect. Spinning more instances will increase the cost of your solution. You should predict the user concurrency for your application and have the right cost estimate for your solution. In addition, you should devise an appropriate strategy to control or manage the user concurrency using queues or some other techniques and control the Azure Functions' degree of scalability in your solution.

Hosting Plans for Azure Functions

The hosting plan helps you choose the underlying infrastructure specification for the function, define how the function should scale, and set up any other advanced features such as virtual network support that the function will need. You get billed based on the hosting plan you choose for Azure Functions. The following are the hosting plans supported by Azure Functions:

- Consumption Plan
- Premium Plan
- Dedicated Plan

Consumption Plan

In the Consumption Plan case, you do not have control over how the functions scale. The underlying Azure platform adds or removes instances on the fly based on the incoming traffic that the functions receive. You do not have any control over the underlying hosting infrastructure. You get billed when the function runs. This hosting plan is an ideal serverless plan, but you may encounter a cold-start phenomenon. It takes a while for the Azure Functions instances to warm up and spring into action whenever triggered. Your code does not run instantaneously when the function is triggered as it takes some time to wake up from its idle state. This phenomenon is referred to as the cold-start phenomenon. In the Consumption Plan case, the function can execute for a maximum of ten minutes and has a default value of five minutes. The default value of five minutes refers to the amount of time the function will execute before timing out without explicitly setting the timeout value for the function.

Premium Plan

In the Premium Plan case, you can have prewarmed Azure Functions instances that can spring into action and execute the code as soon the function is triggered. The prewarmed instances help you overcome the cold-start phenomenon. Like with the Consumption Plan, you do not have any control over how Azure Functions scales or over the underlying hosting infrastructure in the Premium Plan case. However, you get options to choose an SKU (EP1, EP2, or EP3) that will meet the memory and CPU requirements for your application. The underlying Azure platform manages all the scaling aspects. You get support for a virtual network. In the Premium Plan case, you can configure a function to run for a longer duration without timing out. By default, the function execution will time out after 30 minutes. Your functions can run continuously or nearly continuously.

Dedicated Plan

The Dedicated Plan in Azure Functions is the same as the App Service Plan in Azure WebApp. You get to choose from a wide range of SKUs and sizes compared to the Premium Plan that will meet the application's memory and CPU requirements. You can configure manual scaling or automatic scaling for your functions. You also get virtual network support. This hosting plan is best suited for long-running applications.

Use Cases for Azure Functions

The Azure Functions service can fit into any modern application patterns and use cases. The following are a few of the best-fit scenarios where you can use Azure Functions:

- You can build an *n*-tier application using Azure Functions. You can break the business and data access logic into smaller chunks and host each of these chunks in a function.
- You can run background processing jobs in Azure Functions.
- You can use Azure Functions and Durable Functions to build workflow-based applications where you can orchestrate each of the workflow steps using Azure Durable Functions and Azure Functions.
- You can use Azure Functions to build microservices-based applications. Each function can host a business service.
- You can use Azure Functions to build schedule-based applications that run on particular time intervals or during a particular time of day or month or year.
- You can build notification systems to trigger a function to notify an end user or a system based on conditions and events.
- You can use Azure Functions in Internet of Things (IoT) scenarios to implement functions to perform a business activity or process the ingested data and put it in storage or send it to the next set of processing.
- You can use Azure Functions and Azure Event Grid in event-driven scenarios where these functions can get triggered and perform a task.

Summary

In this chapter, you learned the basics of Azure Functions. You explored what Azure Functions is and discussed the concepts of serverless computing. You then learned about how Azure WebJobs is different from Azure Functions and then explored the advantages and disadvantages of using Azure Functions and the scenarios in which to use the service. You also learned about the different hosting plans available for Azure Functions.

The following are the key takeaways from this chapter:

- Azure Functions is a serverless computing service on the Microsoft Azure platform and is based on the FaaS computing model.
- You need to build your code, spin up a function, and host your code on Azure Functions. The underlying cloud platform manages the hosting infrastructure and hosting software.
- The underlying platform manages the scaling aspects for Azure Functions, and you need not do any scaling configurations.
- You get billed when a function executes, and you do not incur any cost when a function is idle.
- Azure Functions supports the Consumption, Premium, and Dedicated Plans.
- You can use Azure Functions in current scenarios like the Internet of Things, microservices, event-driven applications, and many more.

CHAPTER 2

Build Your First Azure Function

You can create a function for Azure Functions using a wide variety of options. If you are comfortable with command-line interfaces, then you can use Azure PowerShell or the Azure command-line interface (CLI). You can use an integrated development environment (IDE) or a code editor like Visual Studio IDE or Visual Studio Code. You can also use the Azure portal to create a function.

In the previous chapter, you learned the basics of the Azure Functions service and explored some of its essential concepts. In this chapter, you will explore various options available to create a function. You will learn how to set up the development prerequisites and explore how Azure Functions works under the hood.

Structure of the Chapter

In this chapter, we will explore the following topics:

- Creating a function using the Azure portal
- Creating a function locally using the command line
- Creating a function using Visual Studio Code
- Creating a function using Visual Studio

A. Satapathi and A. Mishra, *Hands-on Azure Functions with C#*, https://doi.org/10.1007/978-1-4842-7122-3_2

Objectives

After studying this chapter, you will be able to do the following:

- Understand the core tools of Azure Functions
- Create a function using various tooling options

Create Functions Using the Azure Portal

In this section, you'll create a function in the Azure portal. The Azure portal provides an in-portal editor to create and customize functions. To create a function in the Azure portal, you will first have to create a *function app*. Then you can create multiple functions inside the function app.

To create a function app, visit `https://portal.azure.com` and log in to the portal using your credentials (Figure 2-1).

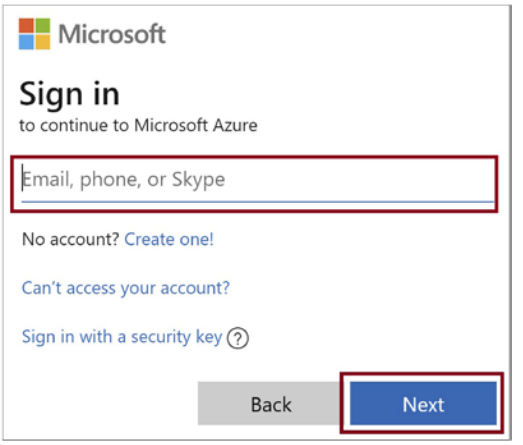

Figure 2-1. *Sign in to the Azure portal*

Once your login is successful, you will get redirected to the Azure portal dashboard. Type **function app** in the search bar and click the Function App option, as shown in Figure 2-2.

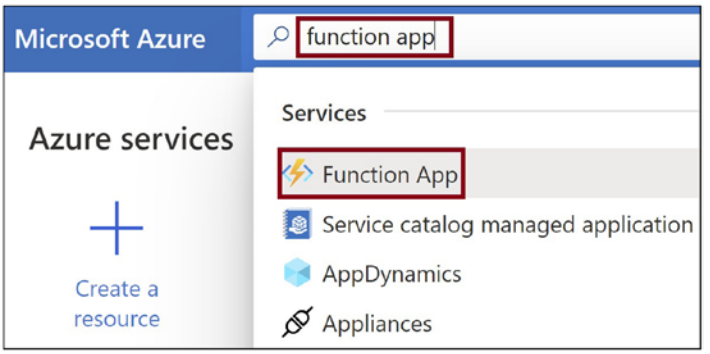

Figure 2-2. *Search for function app*

Now click the Create button, as shown in Figure 2-3, to create a new function app resource. If you have already created a function, you can see that listed in the portal.

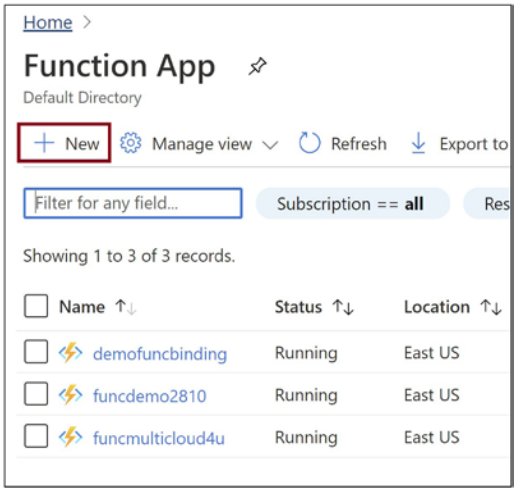

Figure 2-3. *Create a new function app*

Now you will get redirected to a new screen, as shown in Figure 2-4, where you need to fill in the required fields for the Basics section. These details are crucial to create your function app in your subscription.

Select the subscription in which you want to get billed for this Azure function app. After you select the subscription, choose or create the resource group where you need to create the function app. Then provide a unique name for your function app. This name needs to be globally unique, and no other function app should have the same name across Azure.

You can host either the application code or a container in Azure Functions. Select the option Code. The runtime stack refers to the language in which you need to create your functions. This book focuses on working with Azure Functions using C#, so for this example select .NET Core as the runtime stack.

Note You can write your application code and host it directly on function apps. Alternatively, you can containerize your functions and deploy the container in the function app.

Next, you need to select the version. We discussed the supported runtime stacks and supported language versions in Chapter 1; refer to Table 1-1. Finally, select the region where you need to create this function app. It is recommended that you select the nearest or same geographic region where the consuming services or applications are hosted.

Once you have filled in all the fields highlighted in Figure 2-4, click Next: Hosting to configure your function app's hosting plan–related configurations. Alternatively, you can click Review + Create to review your function configuration and then click Create to spin up the function app.

Note An Azure function app consists of multiple functions. All functions of a function app share the same resources, configurations, language runtime, and pricing plan.

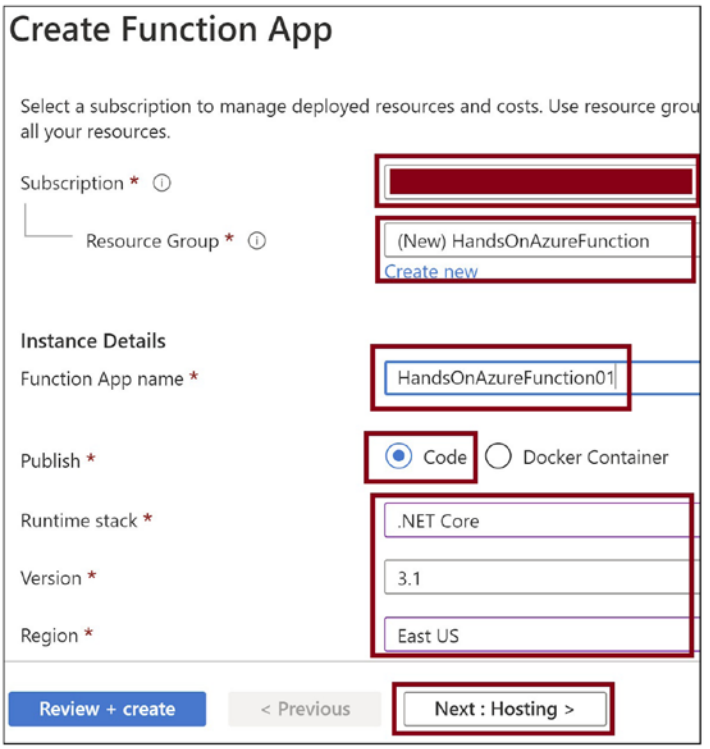

Figure 2-4. *Provide basic configuration details*

In the Hosting section, you need to fill in a few more details, as highlighted in Figure 2-5. You need to select an Azure storage account. You can select an existing storage account or create a new general-purpose Azure storage account. We need the storage account for monitoring and logging purposes. All the logs and metrics data gets stored in the storage account. The storage account also facilitates storing the code, as well as the binding configuration files for the functions created using the Consumption Plan or Premium Plan. So, we must have a storage account associated with a function to facilitate storing the code and the binding configuration files.

Note If you delete the function app's storage account, configured while creating the function app, all the functions that are part of that function app will stop working.
