Microsoft

AZ-204T00A

Learning Path 02: Implement Azure Functions

Agenda

- Explore Azure Functions
- Develop Azure Functions

Module 1: Explore Azure Functions

Learning objectives

- Explain functional differences between Azure Functions, Azure Logic Apps, and WebJobs
- Describe Azure Functions hosting plan options
- Describe how Azure Functions scale to meet business needs

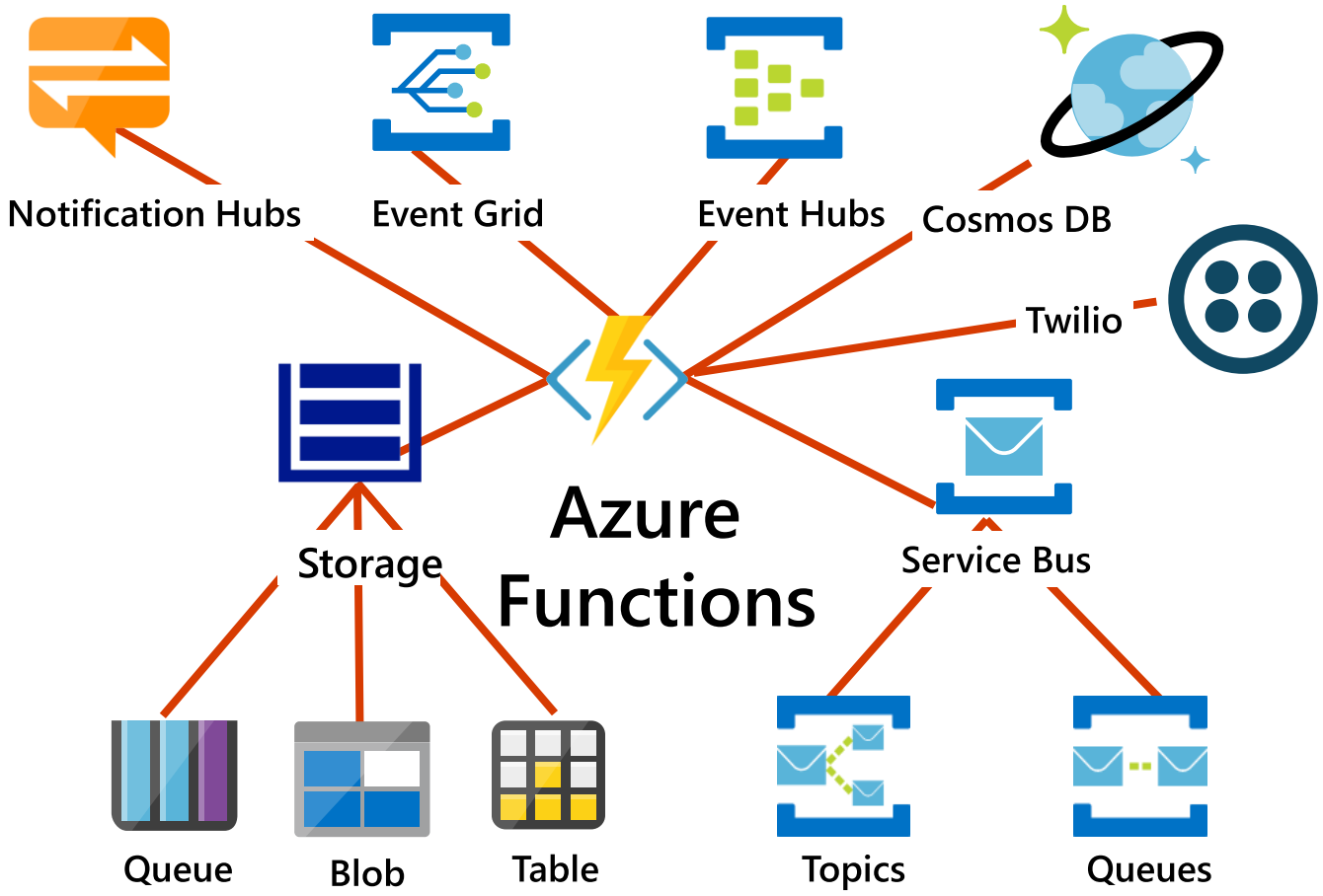
Introduction

- Azure Functions is a serverless solution
- The cloud infrastructure provides all the up-to-date resources needed to keep your applications running
- Azure Functions scale with demand

Discover Azure Functions (1 of 3)

Overview

- Consider Functions for tasks like image or order processing, file maintenance, or for any tasks that you want to run on a schedule.
- Azure Functions supports *triggers*, which start execution of your code, and *bindings*, which simplify coding for input and output data.



Discover Azure Functions (2 of 3)

Compare Azure Functions and Azure Logic Apps

	Functions	Logic Apps
Development	Code-first (imperative)	Designer-first (declarative)
Connectivity	About a dozen built-in binding types, write code for custom bindings	Large collection of connectors, Enterprise Integration Pack for B2B scenarios, build custom connectors
Actions	Each activity is an Azure function; write code for activity functions	Large collection of ready-made actions
Monitoring	Azure Application Insights	Azure portal, Azure Monitor logs
Management	REST API, Visual Studio	Azure portal, REST API, PowerShell, Visual Studio
Execution context	Can run locally or in the cloud	Supports run-anywhere scenarios

Discover Azure Functions (3 of 3)

Compare Functions and WebJobs

	Functions	WebJobs with WebJobs SDK
Serverless app model with automatic scaling	Yes	No
Develop and test in browser	Yes	No
Pay-per-use pricing	Yes	No
Integration with Logic Apps	Yes	No
Trigger events	Timer Azure Storage queues and blobs Azure Service Bus queues and topics Azure Cosmos DB Azure Event Hubs HTTP/WebHook (GitHub Slack) Azure Event Grid	Timer Azure Storage queues and blobs Azure Service Bus queues and topics Azure Cosmos DB Azure Event Hubs File system

Compare Azure Functions hosting options (1 of 3)

Three basic hosting plans:

- Consumption plan
- Premium plan
- Dedicated plan (App Service)

Additional options for highest control and isolation:

- App Service Environment
- Kubernetes

Hosting plans dictate:

- How your function app is scaled.
- The resources available to each function app instance.
- Support for advanced functionality, such as Azure Virtual Network connectivity.

Compare Azure Functions hosting options (2 of 3)

Consumption plan

- The default hosting plan.
- Scales automatically and you only pay when your functions are running.
- Instances of the Functions host are dynamically added and removed based on the number of incoming events.

Premium plan

- Automatically scales based on demand using pre-warmed workers.
- Runs on more powerful instances and connects to virtual networks.

Dedicated plan

- Run your functions within an App Service plan at regular App Service plan rates.
- Best for long-running scenarios where Durable Functions can't be used.

Compare Azure Functions hosting options (3 of 3)

Function app timeout duration

- The `functionTimeout` property in the *host.json* project file specifies the timeout duration.
- **Consumption** plan has a default timeout of 5 minutes, and a maximum timeout of 10 minutes.
- **Premium** and **Dedicated** plans have a default timeout of 30 minutes and no maximum duration.

Storage account requirements

- On any plan, a function app requires a general Azure Storage account.
- Azure Functions relies on Azure Storage for operations such as managing triggers and logging function executions.

Scale Azure Functions (1 of 3)

- The unit of scale for Azure Functions is the function app.
- A *scale controller* monitors the rate of events to determine whether to scale out or scale in.
- Instances may be "scaled in" to zero when no functions are running within a function app.
- The next request has the latency – a *cold start* – of scaling from zero to one

Scale Azure Functions (2 of 3)

Scaling behaviors

Scaling can vary on a number of factors, and scale differently based on the trigger and language selected.

- **Maximum instances**: A single function app only scales out to a maximum of 200 instances. A single instance may process more than one message or request at a time though, so there isn't a set limit on number of concurrent executions.
- **New instance rate**: For HTTP triggers, new instances are allocated, at most, once per second. For non-HTTP triggers, new instances are allocated, at most, once every 30 seconds.

Scale Azure Functions (3 of 3)

Limit scale out

- You can restrict the maximum number of instances an app used to scale out.
- Common for cases where a downstream component like a database has limited throughput.
- Set the *functionAppScaleLimit* value to zero for unrestricted, or to a value between one and app maximum

Scaling in a Dedicated plan

- Using an App Service plan, you can manually scale out by adding more VM instances.
- You can also enable autoscale.

Summary and knowledge check

In this module, you learned how to:

- Explain functional differences between Azure Functions, Azure Logic Apps, and WebJobs
- Describe Azure Functions hosting plan options
- Describe how Azure Functions scale to meet business needs

1. Which Azure Functions hosting plan is best when predictive scaling and costs are required?

2. Which serverless workflow would you choose if the solution requires a designer-first development model?

Module 2: Develop Azure Functions

Learning objectives

- Explain the key components of a function and how they are structured
- Create triggers and bindings to control when a function runs and where the output is directed
- Connect a function to services in Azure
- Create a function by using Visual Studio Code and the Azure Functions Core Tools

Introduction

A function contains two important pieces:

- Your **code**, which can be written in a variety of languages.
- Some **config**, the *function.json* file.
- Depending on your chosen language the *function.json* file might be automatically generated

Explore Azure Functions development (1 of 3)

- The *function.json* file defines the function's trigger, bindings, and other configuration settings.
- Every function has one and only one trigger.
- The runtime uses this config file to determine the events to monitor and how to pass data into and return data from a function execution.
- The bindings property is where you configure both triggers and bindings.

```
{
  "disabled":false,
  "bindings":[
      // ... bindings here
      {
          "type": "bindingType",
          "direction": "in",
          "name": "myParamName",
          // ... more depending on binding
      }
  ]
}
```

Explore Azure Functions development (2 of 3)

A function app provides an execution context in Azure to run your functions.

- It is the unit of deployment and management for your functions.
- A function app is comprised of one or more individual functions that are managed, deployed, and scaled together.
- All functions in a function app share the same pricing plan, deployment method, and runtime version.

Explore Azure Functions development (3 of 3)

Local development environments

- Functions makes it easy to use your favorite code editor and development tools to create and test functions on your local computer.
- Your local functions can connect to live Azure services, and you can debug them on your local computer using the full Functions runtime.
- The way in which you develop functions on your local computer depends on your language and tooling preferences.

Create triggers and bindings (1 of 7)

Overview

- Triggers are what cause a function to run. A trigger defines how a function is invoked and a function must have exactly one trigger.
- Binding to a function is a way of declaratively connecting another resource to the function
- Bindings may be connected as input bindings, output bindings, or both.
- You can mix and match different bindings to suit your needs.
- Triggers and bindings let you avoid hardcoding access to other services.

Create triggers and bindings (2 of 7)

Trigger and binding definitions

- Triggers and bindings are defined differently depending on the development language.
- C# class library - decorating methods and parameters with C# attributes
- Java - decorating methods and parameters with Java annotations
- JavaScript/PowerShell/Python/TypeScript - updating *function.json* schema

```
{
  "dataType": "binary",
  "type": "httpTrigger",
  "name": "req",
  "direction": "in"
}
```

Create triggers and bindings (3 of 7)

Binding direction

- All triggers and bindings have a direction property in the *function.json* file
- For triggers, the direction is always `in`
- Input and output bindings use `in` and `out`
- Some bindings support a special direction `inout`. If you use `inout`, only the **Advanced editor** is available via the **Integrate** tab in the portal.
- When you use attributes in a class library to configure triggers and bindings, the direction is provided in an attribute constructor or inferred from the parameter type.

Create triggers and bindings (4 of 7)

Azure Functions trigger and binding example

Scenario: You want to write a new row to Azure Table storage whenever a new message appears in Azure Queue storage.

This scenario can be implemented using an Azure Queue storage trigger and an Azure Table storage output binding.

```
"bindings": [
    {
      "type": "queueTrigger",
      "direction": "in",
      "name": "order",
      "queueName": "myqueue-items",
      "connection": "STORAGE_ACCT_SETTING"
    },
    {
      "type": "table",
      "direction": "out",
      "name": "$return",
      "tableName": "outTable",
      "connection": "STORAGE_ACCT_SETTING"
    }
```

Create triggers and bindings (5 of 7)

C# script example

C# script code that works with the previous trigger and binding specified.

```
...
public static Person Run(JObject order, ILogger log)
{
    return new Person() {
        PartitionKey = "Orders",
        RowKey = Guid.NewGuid().ToString(),
        Name = order["Name"].ToString(),
        MobileNumber = order["MobileNumber"].ToString() };
}
public class Person
{
    public string PartitionKey { get; set; }
    public string RowKey { get; set; }
    public string Name { get; set; }
    public string MobileNumber { get; set; }
}
```

Create triggers and bindings (6 of 7)

JavaScript example

JavaScript code that works with the previous trigger and binding specified.

```
module.exports = async function (context, order) {
    order.PartitionKey = "Orders";
    order.RowKey = generateRandomId();
    context.bindings.order = order;
};
function generateRandomId() {
    return Math.random().toString(36).substring(2, 15) +
        Math.random().toString(36).substring(2, 15);
}
```

Create triggers and bindings (7 of 7)

Class library example

```
public static class QueueTriggerTableOutput
{
    [FunctionName("QueueTriggerTableOutput")]
    [return: Table("outTable", Connection = "CONNECTION")]
    public static Person Run(
        [QueueTrigger("myqueue-items", Connection = "CONNECTION")]JObject order,
        ILogger log)
    {
        return new Person() {
            PartitionKey = "Orders",
            RowKey = Guid.NewGuid().ToString(),
            Name = order["Name"].ToString(),
            MobileNumber = order["MobileNumber"].ToString() };
    }
...
```

Connect functions to Azure services (1 of 2)

Overview

- Your function project references connection information by name from its configuration provider.
- It does not directly accept the connection details, allowing them to be changed across environments.
- The default configuration provider uses environment variables. These might be set by Application Settings when running in the Azure Functions service, or from the local settings file when developing locally.

Connection values

- When the connection name resolves to a single exact value, the runtime identifies the value as a connection string, which typically includes a secret.
- The details of a connection string are defined by the service to which you wish to connect.
- A connection name can also refer to a collection of multiple configuration items.
- Environment variables can be treated as a collection by using a shared prefix that ends in double underscores __.

Connect functions to Azure services (2 of 2)

Configure an identity-based connection

- Some connections in Azure Functions are configured to use an identity instead of a secret. Support depends on the extension using the connection.
- In some cases, a connection string may still be required in Functions even though the service to which you are connecting supports identity-based connections.

Grant permission to the identity

- Whatever identity is being used must have permissions to perform the intended actions.
- This is typically done by assigning a role in Azure RBAC or specifying the identity in an access policy, depending on the service to which you are connecting.

Exercise: Create an Azure Function by using Visual Studio Code

In this exercise, you learn how to create a C# function that responds to HTTP requests. After creating and testing the code locally in Visual Studio Code, you'll deploy to Azure.

Objectives

- Create your local project
- Run the function locally
- Sign into Azure
- Publish the project to Azure
- Run the function in Azure
- Clean up resources

Summary and knowledge check

In this module, you learned how to:

- Explain the key components of a function and how they are structured
- Create triggers and bindings to control when a function runs and where the output is directed
- Connect a function to services in Azure
- Create a function by using Visual Studio Code and the Azure Functions Core Tools

1. What is required for a function to run. A trigger, binding, or both?

2. What supports the `in` and `out` direction settings. Triggers, bindings, or both?

Discussion and lab

Group discussion questions

- Contoso Inc built an Azure Function that is hosted using Consumption Plan. After analyzing the logs, you've noticed they are timing out after 5 minutes. What can you do to avoid this problem?
- Contoso Inc built an Azure Function to consume events from Event Hub. After some load testing, you've noticed they are not performing well. What can you do increase the performance?

Lab 02: Implement task processing logic by using Azure Functions

In this lab, you will demonstrate the ability to create a simple Azure function that echoes text that is entered and sent to the function by using HTTP POST commands. This will illustrate how the function can be triggered over HTTP.

http://aka.ms/az204labs

- Exercise 1: Create Azure resources
- Exercise 2: Configure a local Azure Functions project
- Exercise 3: Create a function that's triggered by an HTTP request
- Exercise 4: Create a function that triggers on a schedule
- Exercise 5: Create a function that integrates with other services
- Exercise 6: Deploy a local function project to an Azure Functions app

End of presentation