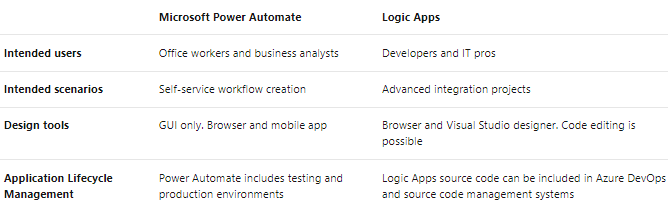
**PRACTICAL EXERCISES:**

1. Function HTTP Trigger for the Temperature
2. Function to Read values from Cosmos DB
3. Function to Write values to Cosmos DB and Queue
4. Durable Functions – Had Problem
5. Creating functions locally – Had Problem
6. Creating and publishing functions in VS
7. Creating a webhook in GitHub – Most exercises did no work

**[Choose the best Azure service to automate your business processes](https://docs.microsoft.com/en-us/learn/modules/choose-azure-service-to-integrate-and-automate-business-processes/?ns-enrollment-type=LearningPath&ns-enrollment-id=learn.create-serverless-applications)**

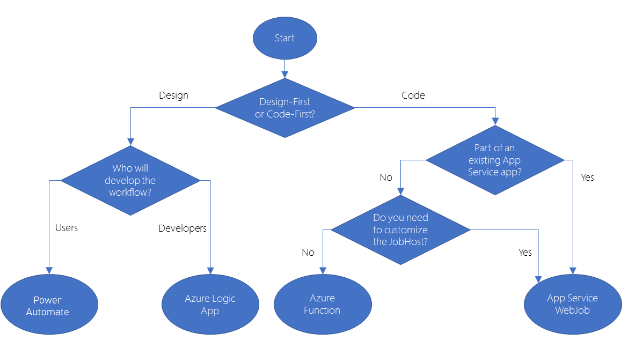
1. Common Business Issues
   1. Business processes modelled in software are often called **workflows**
   2. Azure includes four different for implement workflows that integrate multiple systems:
      1. Logic Apps
      2. Microsoft Power Automate
      3. Web Jobs
      4. Azure Functions
   3. These four technologies have some similarities**:**
      1. They can all accept inputs
      2. They can all run actions.
      3. They can all include conditions
      4. They can all produce outputs
2. Design-first technologies
   1. Logic Apps
      1. **Logic** **Apps** is a service within **Azure** to **automate**, **orchestrate**, and **integrate** **disparate** **components** of a **distributed** **application**
      2. Design-first approach in Logic Apps
      3. You can use a  **Logic** **Apps** **Designer** and **design** **canvas** or edit **JSON** notation
      4. Has over 200 connectors
      5. **Connector** is a **Logic** **Apps** component that provides an interface to an external service
   2. Microsoft Power Automate
      1. Is a service to create workflows even when you have no development
      2. There are four different types of flow:
         1. **Automated** – Starts with a trigger
         2. **Button** – Single click runs a repetitive task
         3. **Scheduled** – Runs on a regular basis, eg week
         4. **Business** **Processes** – Models a business process
   3. Design-First technologies compared



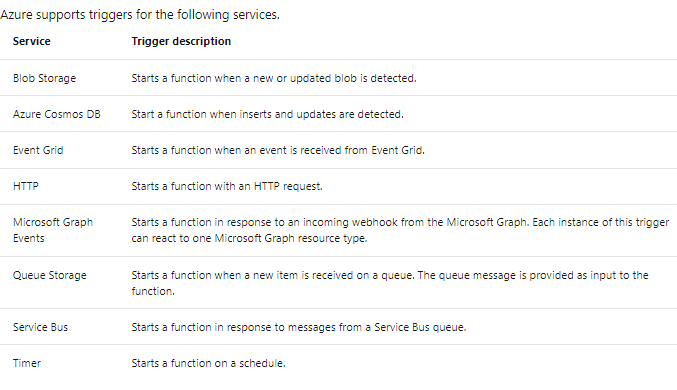
* 1. Code-First technologies
     1. WebJobs and WebJobs SDK
        1. The Azure App Service is a cloud-based hosting service for web applications, mobile back-ends, and RESTful APIs.
        2. **WebJobs** are a part of the Azure App Service that you can use to run a program or script automatically Azure Functions
        3. **Two kinds of Web Jobs:**
           1. Continuous – Runs in a continuous loop
           2. Triggered – Runs when started or on a schedule

1. Azure Functions
   1. An Azure Function is a simple way for you to run small pieces of code in the cloud
   2. Without having to worry about the infrastructure required to host that code
   3. You can write the Function in C#, Java, JavaScript, PowerShell, Python, or any of the languages that are listed in the Supported languages in Azure Functions article
   4. When you create an Azure Function, you can start by writing the code for it in the portal
   5. **Some of the templates available to you:**
      1. **HTTPTrigger** - code to execute in response to a request sent through the HTTP protocol
      2. **TimerTrigger** - code to execute according to a schedule
      3. **BlobTrigger** - code to execute when a new blob is added to an Azure Storage account
      4. **CosmosDBTrigger -** Code to execute in response to new or updated documents in a NoSQL database
2. Code-First Technologies Compared
   1. Simple administration and more flexible coding model provided by Azure Functions may lead you to choose them in preference to WebJobs
   2. **You may choose WebJobs for the following reasons:**
      1. You want the code to be a part of an existing App Service application and to be managed as part of that application
      2. You need close control over the object that listens for events that trigger the code

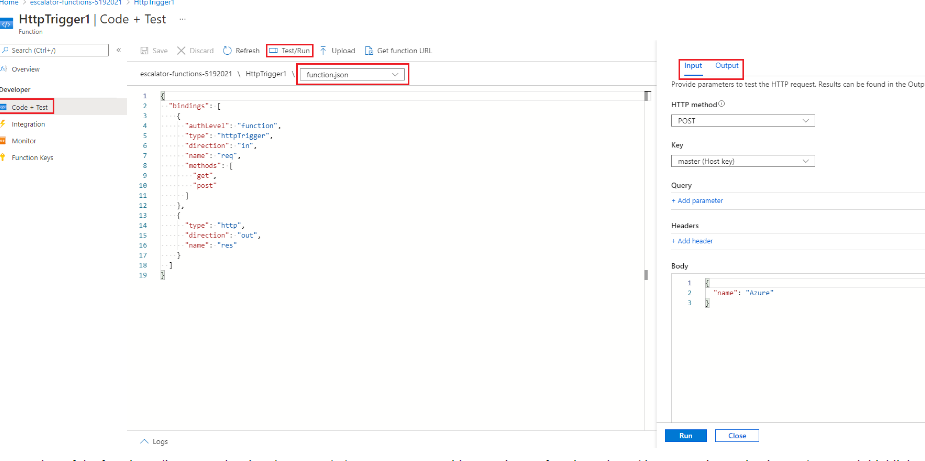


1. Analyse the decision criteria
   1. How to choose a service
   2. Choosing a design-first technology
      1. If you choose to use a design-first approach, you must also choose from Microsoft Power Automate and Azure Logic Apps.
      2. Who will design the workflow: will it be **developers** or **users**?
      3. Logic Apps: there is a GUI designer on which you draw out the workflow. Designed for people with development skills
      4. In Microsoft Power Automate, extra help and templates are provided for common types of workflow. Designed for users who have a good understanding of the business process but no coding skills
   3. Choosing a code-first technology
      1. Extra features that are included with Azure Functions:
         1. Trigger events and supported languages
         2. Ability to develop test code in the browser
         3. Pay-per-use price model
      2. There are two situations in which WebJobs might be a better choice:
         1. You have an existing Azure App Service application
         2. You have specific customizations that you want to make to the JobHost
         3. Webjobs only supports C# on Microsoft Windows
   4. **Mixing technologies**
      1. You can also call one workflow from another
      2. Microsoft Power Automate can easily call another that is built as an Azure Function
      3. Reason to mix the technologies used in your business processes would be to give users control over a small section of a complete workflow
2. Choose the best design-first technology to automate your business process
   1. Scenario
   2. **Business Process** – Bike Shop
   3. Choose a technology
   4. Design First or code first
      1. If you choose a design-first approach, the workflow is visualized in an easy-to-understand design surface. In addition, that diagram is not a separate document, but a picture of the process as it is implemented. The benefit is that there's no possibility that the diagram is not updated when the process is changed.
   5. Microsoft Power Automate or Azure Logic Apps?Following factors influence the decision:
      1. It seems sensible that the development of the custom connector and the workflow should be done by the same person or team. Since these must be developers, it's best to use Azure Logic Apps.
3. When to choose Azure Functions to run your business logic
   1. Scenario
   2. Business Process
   3. Choose a technology
      1. Design-first or code-first?
      2. To implement this workflow using just Logic Apps or Power Automate would be difficult**. While we haven't heard too many low-level details**, it's clear that this process needs to access an inventory system and place orders with a third-party parts company.
      3. Azure Functions or Azure Apps Service WebJobs?
         1. Cost: With Web Jobs, you pay for the entire VM or App Service Plan. Azure functions run on consumption plan.
         2. Integrations: Although it is possible to call a WebJob from a Logic App, the integration between Logic Apps and Functions is closer

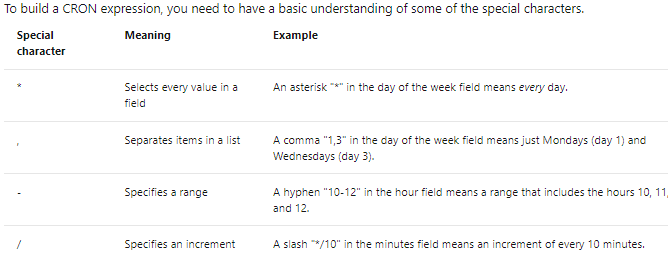
### [Create serverless logic with Azure Functions](https://docs.microsoft.com/en-us/learn/modules/create-serverless-logic-with-azure-functions/?ns-enrollment-type=LearningPath&ns-enrollment-id=learn.create-serverless-applications)

1. Decide if serverless computing is right for your business needs
   1. What is serverless compute?
      1. [Serverless compute](https://azure.microsoft.com/solutions/serverless/) can be thought of as a function as a service (FaaS)
      2. Logic runs as functions and you don't have to manually provision or scale infrastructure
      3. Your app is automatically scaled out or down depending on load
      4. Two most common approaches are Azure Logic Apps and Azure Functi
   2. What is Azure Functions?
      1. Azure Functions is a serverless application platform
      2. You can write your function code in the language of your choice, including C#, F#, JavaScript, Python, and PowerShell Core
      3. Support for package managers like NuGet and NPM
   3. Benefits of a serverless compute solution
      1. Avoids over-allocation of infrastructure
      2. Stateless logic - Function instances are created and destroyed on demand
      3. Event driven – Functions are event driven. Run only in response to an event (called a "trigger")
      4. Functions can be used in traditional compute environments - Should the needs of your app change, you can take your project and deploy it in a non-serverless environment
   4. Drawbacks of a serverless compute solution
      1. **Execution time** - Functions have a timeout of 5 minutes. Timeout is configurable to a maximum of 10 minutes. If your service is initiated through an HTTP request, get additional 2.5 minutes.  [**Durable Functions**](https://docs.microsoft.com/en-us/azure/azure-functions/durable) that allows you to orchestrate the executions of multiple functions without any timeout
      2. **Execution frequency** – If function will be called by multiple clients, might be better to host on a VM. While scaling, only one function app instance can be created every 10 seconds, for up to 200 total instances
   5. **Exercise - Create a function app in the Azure portal – Same Sandbox? Publish Code – Code or Docker Container. Creating function App -.net 3.1**
      1. JavaScript?
      2. Powershell?
      3. What is a function app?
         1. Functions are hosted in an execution context called a function app
         2. Function apps are used to logically group and structure your functions and a compute resource in Azure.
      4. Choose a service plan
         1. The first service plan is the Consumption service plan -  Azure serverless application platform. The Consumption service plan provides automatic scaling and bills you when your functions are running. Default timeout is 5 minutes, to as long a 10 minutes.
         2. The second plan is called the Azure App Service plan - Allows you to avoid timeout periods. Because it runs continuously on a VM.  You are responsible for managing the app resources the function runs on
      5. Storage account requirements
         1. A function app must be linked to a storage account
         2. Function app uses this storage account for internal operations such as logging function executions and managing execution triggers
   6. Run your code on-demand with Azure Functions
      1. Functions are event driven
      2. Event that starts a function is a trigger
   7. Bindings
      1. Bindings are a declarative way to connect data and services to your function
      2. Bindings know how to talk to different services, which means you don't have to write code in your function to connect to data sources and manage connections.
      3. Each **binding** has a direction - your code reads data from **input** **bindings**, and writes data to **output** **bindings**.
      4. A trigger is a special type of input binding that has the additional capability of initiating execution.
   8. Define a sample binding
      1. Let's say we 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.



* + 1. Our JSON configuration specifies that our function will be triggered when a message is added to a queue named **myqueue-items**. The return value of our function is then written to the **outTable** table in Azure Table storage.
    2. To view and edit the contents of function.json in the Azure portal, from Home, select your function app, and in the right pane, select **JSON View**
  1. Create a function in the Azure portal
     1. Azure provides several pre-made function templates for common scenario
        1. Quickstart
        2. Custom functions
     2. Add Function Templates
        1. When adding your first function, you are presented with the Add function pane
        2. When adding your first function, you are presented with the Add function pane
     3. Custom Function Templates
        1. Azure provides over 30 additional templates you can start with
     4. Navigate to your function and files
        1. For example:
           1. A configuration file, **function.json**, and a source code file, **index.js**.
        2. In the left menu pane, under Developer, select Code + Test
        3. As you can see, the pane on the right has tabs for Input and Output. Selecting the Input tab provides parameters to test the HTTP request for your function
  2. Test your Azure function
     1. Manual execution - For instance, if you are using an HTTP trigger, you can use a tool, such as Postman or cURL
     2. Azure Portal - The portal also provides a convenient way to test your functions. selecting Test/Run from the top menu bar, on the right side of the code box,
  3. Monitoring and Application Insights dashboard
     1. The Azure portal provides a monitoring dashboard if you turn on the Application Insights integration
     2. In the left menu pane of your function app, under Monitoring, select Logs and open Application Insights
  4. Streaming Logs Pane
     1. You're also able to add logging statements to your function for debugging in the Azure portal
     2. Calls are passed to a logging object
     3. 
  5. Errors and Warnings
     1. You can locate the errors and warnings window tab in the same flyout menu as the log window
  6. Exercise - Add logic to the function app
     1. **cURL** is a command line tool that can be used to send or receive files.
     2. It's included with Linux, macOS, and Windows 10, and can be downloaded for most other operating systems. cURL supports numerous protocols like HTTP, HTTPS
     3. Secure HTTP triggers
        1. API keys to block unknown callers
        2. When you create a function, you select the authorization level
        3. By default, it's set to Function, but can be set to Admin
        4. Use a global "master" key, or Anonymous to indicate that no key is required
        5. Can supply key as HTTP Request

### [Execute an Azure Function with triggers](https://docs.microsoft.com/en-us/learn/modules/execute-azure-function-with-triggers/?ns-enrollment-type=LearningPath&ns-enrollment-id=learn.create-serverless-applications)

1. Determine the best trigger for your Azure function
   1. What is a trigger?
      1. Defines how an Azure Function is invoked
   2. Types of triggers
      1. Timer, HTTP, BLOB, Queue, Azure Cosmos DB, Event Hub
   3. What is a binding?
2. Run an Azure Function on a schedule
   1. What is a timer trigger?
      1. A timer trigger is a trigger that executes a function at a consistent interval
      2. You supply two pieces of information:
         1. Timestamp parameter name
         2. A schedule – CRON expression
   2. What is a CRON expression?
      1. Is a string that consists of six fields that represent a set of times
      2. The order of the six fields in Azure is: {**second**} {**minute**} {**hour**} {**day**} {**month**} {**day of the week**}.
      3. For example, a CRON expression to create a trigger that executes every five minutes looks like: 0 \*/5 \* \* \* \*
      4. First, the asterisk (\*) means "select every value within the field." Because this field represents minutes, the possible values are 0-59. The second special character is the slash (/), which represents an increment. When you combine these characters together, it means for all values 0-59, select every fifth value. An easier way to say that is simply "every five minutes."
3. How to Create a Timer Trigger
   1. You can create a timer trigger in the Azure portal.
4. Exercise - Create a timer trigger
5. Execute an Azure function with an HTTP request
   1. Azure Functions allows us to quickly create a piece of logic to execute when an HTTP request is received
   2. What is an HTTP trigger?
      1. HTTP triggers have many capabilities and customizations, including:
         1. Provide authorized access by supplying keys
         2. Restrict which HTTP verbs are supported
         3. Return data back
         4. Receive data through a string parameter
         5. Support URL route templates
      2. When you create an HTTP trigger, you need to select a programming language, provide a trigger name, and select an Authorization level
   3. What is an HTTP trigger Authorization Level?
      1. An HTTP trigger Authorization level is a flag
      2. There are three levels:
         1. Function
         2. Anonymous
         3. Admin
      3. An HTTP trigger Authorization level is a flag
      4. There are two types of keys: function and host
      5. The **Anonymous** level means that authentication is not required
   4. How to create an HTTP Trigger
      1. Just like a timer trigger, you can create an HTTP trigger through the Azure portal
      2. One setting that's important to understand is **Request parameter name.** By default, the name of the parameter is req.
   5. How to Invoke an HTTP Trigger
      1. To invoke an HTTP trigger, you send an HTTP request to the URL for your function
      2. After you have the URL for your function, you can send HTTP requests
      3. An HTTP trigger invokes an Azure function when it receives an HTTP request to its function URL. HTTP triggers allow you to receive data and return data back to the caller
   6. Execute an Azure function when a blob is created
      1. What is Azure Storage?
         1. Azure Storage is Microsoft's cloud storage solution that supports all types of data, including: blobs, queues, and NoSQL
         2. The goal of Azure Storage is to provide data storage that's:
            1. High Availability
            2. Secure
            3. Scalable
            4. Managed
         3. What is Azure Blob storage?
            1. Is an object storage solution that's designed to store large amounts of unstructured data
            2. Good For:

Storing Files

Serving Files

Streaming Video and Audio

Logging Data

* + - * 1. Types of BLOBS:

Block Blobs: Most common. store text or binary data efficiently

Append Blobs: Like block blobs, but  designed more for append operations like creating a log file that's being constantly updated

Page Blobs: Made up of pages and are designed for frequent random read and write operations

* + - * 1. What is a blob trigger?

Is a trigger that executes a function when a file is uploaded or updated in Azure Blob storage

To create a blob trigger, you create an Azure Storage account and provide a location that the trigger monitors

* + - * 1. How to create a blob trigger

One setting that's important to understand is the **Path**

The **Path** tells the blob trigger where to monitor

By default, the **Path** value is: samples-workitems/{name}

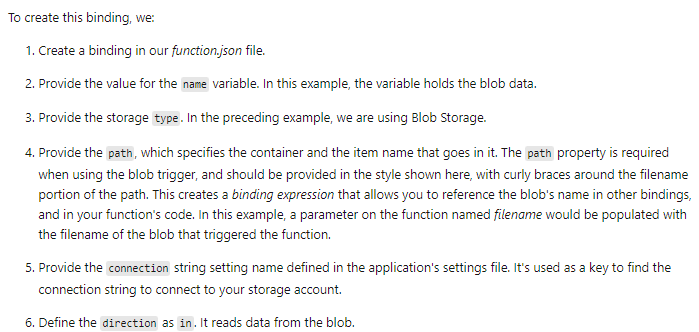
samples-workitems, represents the blob container that the trigger monitors

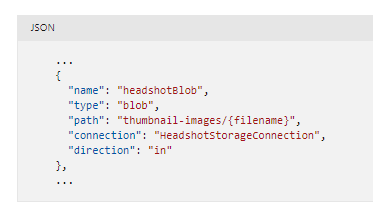
The second part, {name} means that every type of file will cause the trigger to invoke the function

 For example, we could make the trigger invoke the function only when a PNG file is added by using syntax like:

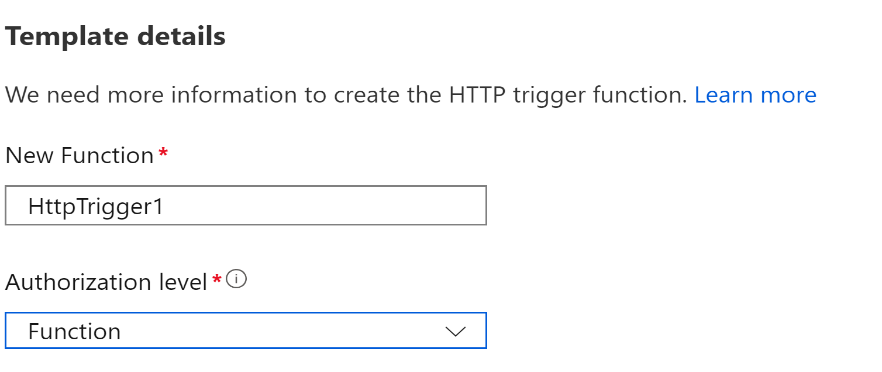
samples-workitems/{name}.png

### [Chain Azure Functions together using input and output bindings](https://docs.microsoft.com/en-us/learn/modules/chain-azure-functions-data-using-bindings/?ns-enrollment-type=LearningPath&ns-enrollment-id=learn.create-serverless-applications)

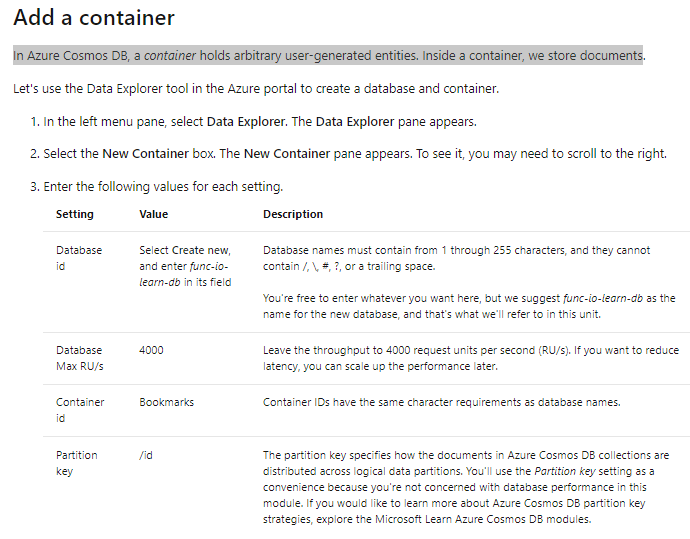
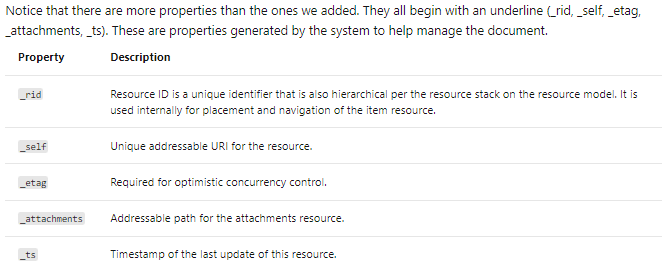
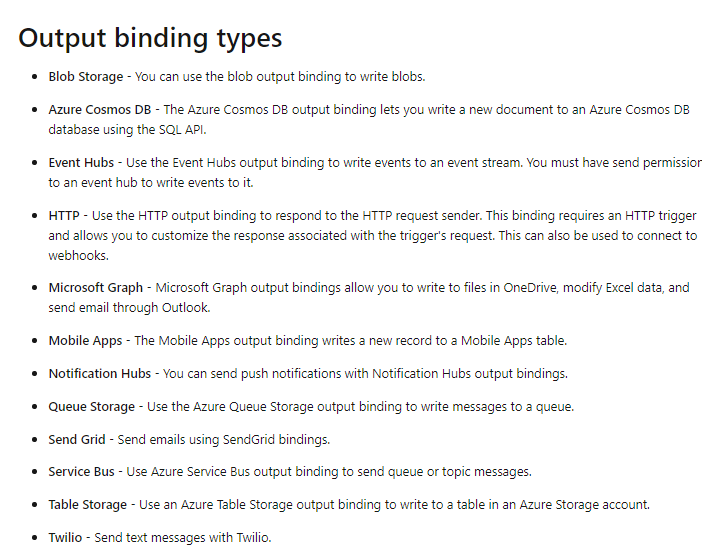
1. Explore input and output binding types
   1. What is a binding?
      1. Declarative way to connect to data from within your code
      2. Make it easier to integrate with data streams consistently in a function
      3. Types of bindings:
         1. Input Binding – Connects to data source
         2. Output Binding – Connects to data destination
      4. There are also triggers, which are special types of input bindings that cause a function to run
      5. Types of supported bindings:
         1. Blob Storage
         2. Azure Service Bus Queues
         3. Azure Cosmos DB
         4. Azure Event Hubs
         5. External Files
         6. External Tables
         7. HTTP End Points
      6. Binding Properties
         1. Name - Defines the function parameter through which you access the data
         2. Type - Identifies the type of binding. Eg, data or service we want to interact with
         3. Direction - Indicates the direction data is flowing, input or output?
         4. Connection: Additional parameter for connection strings to store app settings.
      7. Create a binding
         1. Bindings are defined in JSON.
         2. In a file names function.json
         3. Example:



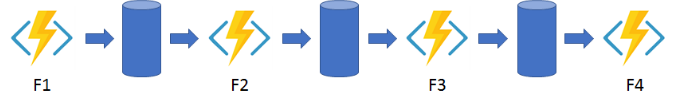
1. Exercise - Explore input and output binding types
   1. In this exercise, we'll create a function that will start when it receives an HTTP request, and will respond to each request by sending back a message
   2. The authorization level option determines what kind of key is used to securely access your function. Choosing Function requires callers of your function to provide a function-specific key with their requests

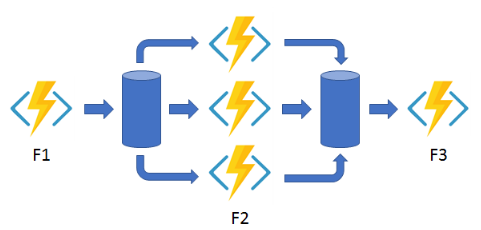


* 1. You can’t add more than one trigger to a function

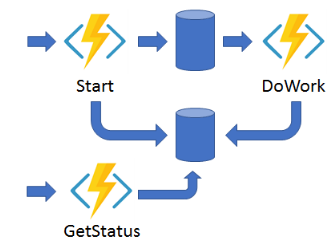
1. Read data with input bindings
   1. To connect to a data source, you have to configure an input binding
   2. Input binding types
      1. Azure Blob Storage
      2. Azure Cosmos DB
      3. Mobile Apps
      4. Azure Table Storage
   3. What is a binding expression?
      1. A binding expression is specialized text in function.json, function parameters, or code that is evaluated when the function is invoked
   4. Types of binding expressions
      1. App Settings
      2. Trigger filename
      3. Trigger metadata
      4. JSON Payloads
      5. New GUID
      6. Current date and time
   5. Most expressions are identified by being wrapped in curly braces.
   6. However, app setting binding expressions are wrapped in percent signs rather than curly brace
   7. For example, if the blob output binding path is %Environment%/newblob.txt, and the Environment app setting value is Development, a blob will be created in the Development container
2. **Exercise - Read data with input bindings – Create a Function to connect to CosmosDB**
   1. Azure Cosmos DB provides five APIs:
      1. Core SQL
      2. MongoDB
      3. Cassandra
      4. Azure Table
      5. Gremlin (Graph Database)
   2. At this time, the Azure Cosmos DB trigger, input bindings, and output bindings only work with Core SQL API and Graph API accounts
   3. In Azure Cosmos DB, a container holds arbitrary user-generated entities. Inside a container, we store documents
3. Write data with output bindings
   1. Not all types support both input and output
   2. It's possible to apply multiple bindings to a single function. This allows you to define both input and output bindings
4. Add an Azure Queue Storage output binding
   1. Azure Queue storage is a service for storing messages that can be accessed from anywhere in the world
   2. The size of a single message can be as much as 64 KB, and a queue can contain millions of messages

# What is Durable Functions?

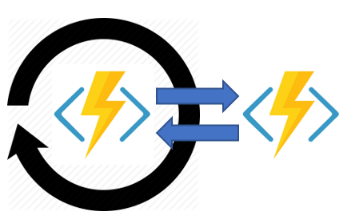
1. Durable Functions enables you to implement complex stateful functions in a serverless-environment
2. Durable Functions
   1. Durable Functions is an extension of Azure Functions
   2. Whereas Azure Functions operate in a stateless environment
   3. Durable Functions can retain state between function calls
   4. Some benefits of using Durable Functions include:
      1. They enable you to write event driven code
      2. You can chain functions together
      3. You can orchestrate and coordinate functions
      4. The state is managed for you
   5. Durable functions allows you to define stateful workflows using an orchestration function
   6. An orchestration function provides these extra benefits:
      1. You can define the workflows in code
      2. Functions can be called both synchronously and asynchronously
      3. Azure checkpoints the progress of a function automatically when the function awaits
3. Function Types
   1. **Client** -  Entry point for creating an instance of a Durable Functions orchestration
   2. **Orchestrator -**  Describe how actions are executed, and the order in which they are run
   3. **Activity -** Are the basic units of work in a durable function orchestration
4. Application Patterns
   1. **Function Chaining:** Workflow executes a sequence of functions
   2. **Fan out/Fan in:** Pattern runs multiple functions in parallel and then waits for all the functions to finish.



* 1. **Async HTTP API** - Addresses the problem of coordinating state of long-running operations with external clients.



* 1. **Monitor:** Implements a recurring process in a workflow, possibly looking for a change in state

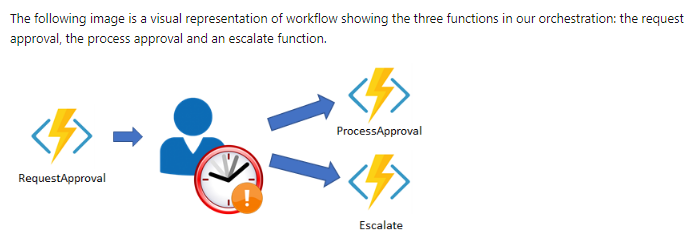


* 1. **Human Interaction:** Combines automated processes that also involve some human interaction



1. Comparison with Logic Apps
   1. Durable Functions and Logic Apps are both Azure services that enable serverless workload
   2. **Azure Durable Functions** is intended as a powerful serverless compute option to run custom logic
   3. **Azure Logic Apps** is better suited for integrating Azure services and components
   4. With **Azure Durable Functions**, you develop **orchestrations** by writing code
   5. With **Logic Apps**, you create **orchestrations** by using the **design** **surface** or **editing configuration files**

# Design a workflow based on Durable Functions

1. You can use Durable Functions to orchestrate a long-running workflow as a set of activities
2. **Escalation** **steps** are useful to the business, as they move along a task when a deadline has been reached
3. They ensure tasks are completed, and not forgotten
4. Description of the design approval process

# Exercise - Create a workflow using Durable Functions

1. Had an error running exercise. Attempt this again.

# Control long-running tasks using timers

1. Timers in Durable Functions
   1. Durable Functions provides timers for use in the orchestrator functions
   2. You should use durable timers in orchestrator functions instead of the **setTimeout()** and **setInterval()** functions
   3. You create a durable timer by calling the **createTimer()** method of the **DurableOrchestrationContext**
2. Using Timers for Delay
   1. You should always use **currentUtcDateTime** to obtain the current date and time, instead of **Date**.**now** or **Date**.**UTC**

# Exercise - Add a durable timer to manage a long-running task

1. Add moment npm package to your function app
   1. Did not work

Summary

1. Durable Functions enables you to implement long-running workflows without requiring that you maintain state information manually

# Develop, test, and publish Azure Functions by using Azure Functions Core Tools

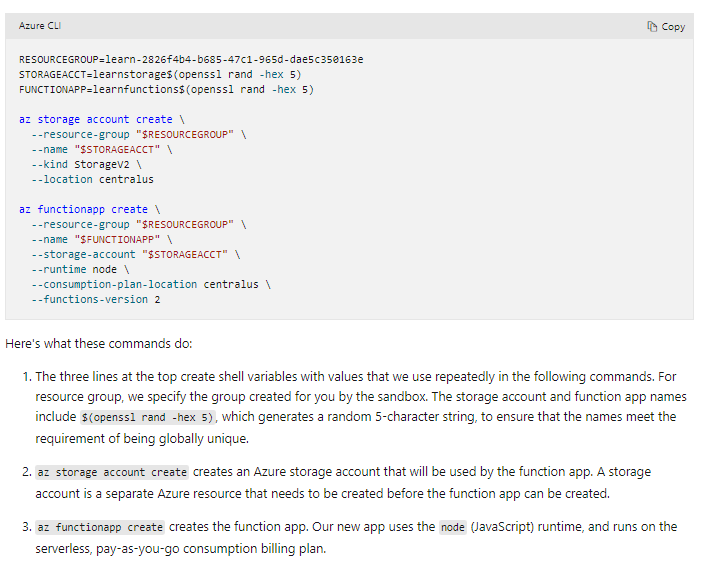
1. Create and run Azure Functions locally by using the Core Tools
   1. The Azure Functions Core Tools let you develop and run functions on your local computer from the command line
   2. What are the Azure Functions Core Tools?
      1. The Azure Functions Core Tools are a set of command-line tools
      2. you can use to develop and test Azure Functions on your local computer
      3. The Core Tools primary purpose:
         1. Generate the files and folders you need
         2. Run your functions locally so you can test and debug them
         3. Publish functions to Azure
      4. The Core Tools are packaged as a single command-line utility named func
      5. If you run func from the command line without any additional commands, it will display version information and a usage guide
      6. The Core Tools are standalone utilities
      7. In practice, you'll also need the Azure CLI or Azure PowerShell to log in to Azure and perform other management tasks
   3. Core Tools versions
      1. Two versions 1.x and 2.x
      2. Latest is recommended
   4. Local development vs. Azure portal development
      1. Once you start using a local development workflow based on the Core Tools, don't expect to be able to use the portal to make changes to your functions.
2. Create functions locally
   1. Function apps and functions projects
      1. Every function published to Azure belongs to a function app
      2. **Function** **app**: a collection of functions that are published together into the same environment.
      3. When you develop functions locally, you work within a functions project
      4. Functions Project: A folder that contains the code and configuration files that define your functions
      5. With the Azure Functions Core Tools, you can use the Core Tools to generate function projects and functions from scratch
   2. Create a new functions project with func init
      1. Regardless of the runtime you choose, the two most critical project files are always present:
         1. host.json
         2. local.settings.json
   3. Create a new function with func new
      1. **func new** will generate a complete, publish-ready starter implementation in your function project's chosen language
   4. Run Functions Locally
      1. Functions aren't programs that can be run on their own: they must be hosted by the **functions** **host**.
      2. The **host** is what **powers** everything outside of your function code: it **loads** **configuration**, **listens** for **triggers** and HTTP requests, **starts** the **worker** **process**
      3. To start the functions host locally, run **func** **start** from a functions project folder
      4. Use **HTTP** calls, like **curl**, to interact with your functions

# Exercise - Create a function locally by using the Core Tools

1. Cloud Shell system has the Core Tools and Azure CLI preinstalled
2. Code . opens Cloud Shell Editor
3. To start function host silently: func start &> ~/output.txt &
4. To stop a background function: pkill func
5. To view the hosts log output: code ~/output.txt

# Publish a function to Azure by using the Core Tools

1. Create a function app
   1. Before you can use the Core Tools to publish a project, you need to create a function app in Azure. This is not a feature in Core Tools.
2. Publish to Azure
   1. The Core Tools don't ask you to sign in to Azure
   2. Instead, they access your subscriptions and resources by loading your session information from the Azure CLI or Azure PowerShell
3. Things to know
   1. The Core Tools do not validate or test your functions. Run func start before publishing
   2. When you publish, any functions already present in the target app are stopped and deleted
   3. Publishing to Azure does not create any kind of relationship between the local project and the target function app
   4. The invocation URLs displayed after you publish may include a code parameter in the query string



# Create and test a simple Azure Function locally with Visual Studio

1. You can write, debug, and deploy an Azure Function from within the Azure portal
2. This is not always required as it edits the dev, staging and production environments.
3. Developing functions with Visual Studio enables you to manage Azure Functions code with code of other services inside the same projects
4. Use Visual Studio Installer to add Azure functionality

## Azure Functions Tools extension for Visual Studio

1. The Azure Functions Tools are a Visual Studio extension that enable you to create, test, and deploy Azure Functions in your local development environment

## Azure Function App

1. An Azure Function App hosts one or more Azure Functions
2. It provides the environment and runtime for the functions
3. An Azure Function is triggered by an event
4. You specify the type of event that will trigger the functions
5. **The events available include:** Blob trigger, Event hub trigger, Azure Cosmos DB trigger, Http trigger, Queue trigger, Service Bus Queue trigger, Service Bus Topic trigger, Timmer trigger
6. Azure currently provides three versions of the runtime environment required to run Azure Functions:
   1. Version 1 (v1) uses the .NET Framework 4.7
   2. Version 2 (v2x) runs using .NET Core 2
   3. Version 3 (v3x) contains JavaScript and .NET changes
7. Using v2 triggers enables you to develop and host the trigger in different environments
8. Version 1 triggers can only be created using Windows
9. An Azure Function App stores management information, code, and logs in Azure Storage
10. Create a Storage Account to hold this data
11. Storage account must support Azure Blob, Queue, Files, and Table storage; use a general Azure Storage account for this purpose
12. An Azure Function can perform privileged or sensitive operations
13. You protect an Azure Function by specifying the access rights required to trigger the function
14. An Azure Function triggered by an HTTP request supports three levels of access rights:
    1. **Anonymous**: No authentication is required
    2. **Function**: The HTTP request must provide a key
    3. **Admin**: Similar to Function, except uses an admin key

## Structure of an Azure Function

1. An **Azure** **Function** is implemented as a **static** class
2. The class provides a **static**, **asynchronous** method named **Run**, which acts as the **entry** **point** for the class
3. The **parameters** passed to the **Run** method provide the **context** for the **trigger**.
4. You can access the data in the request using the same techniques available in any HTTP app
5. The attributes applied to this function specify the authorization requirements
6. The function returns a value containing any output data and results, wrapped in an **IActionResult** object
7. In all cases, an Azure Function is passed an **ILogger** parameter
8. Use to write log messages
9. You can modify metadata using the **HttpTrigger**, **BlobTrigger**
10. If you open a browser and enter the URL it will trigger the function

# Exercise - Create and test a simple Azure Function locally with Visual Studio

# Publish a simple Azure Function

1. An Azure Function runs in the cloud in the context of an Azure Function App
2. A Function App is a container that specifies the operating system for running an Azure Function, with memory, computing power and disk space
3. Azure Function App also provides the public URL

## Deploy from Visual Studio

1. Azure Functions tools for Visual Studio enable you to deploy an Azure Function directly from Visual Studio
2. The Azure Functions template provides a Publish wizard
3. The **Publish** wizard requires that you either have **access** to the **Azure** **Functions** **App** that will host your functions, or you have an **Azure** **subscription**

## Continuous deployment

1. Azure Functions integrates with BitBucket, Dropbox, GitHub, and Azure DevOps, One Drive
2. Continuous deployment is a great option for projects where multiple and frequent contributions are being integrated
3. You can configure continuous deployment from the Azure portal, using the **Deployment Center**

## Zip deployment

1. **Azure** **Functions** can be **deployed** from a **zip** **file** using the **push** **deployment** **technique**
2. You can do this with the Azure CLI, or by using the REST interface
3. The zip file contains the executable code for your functions

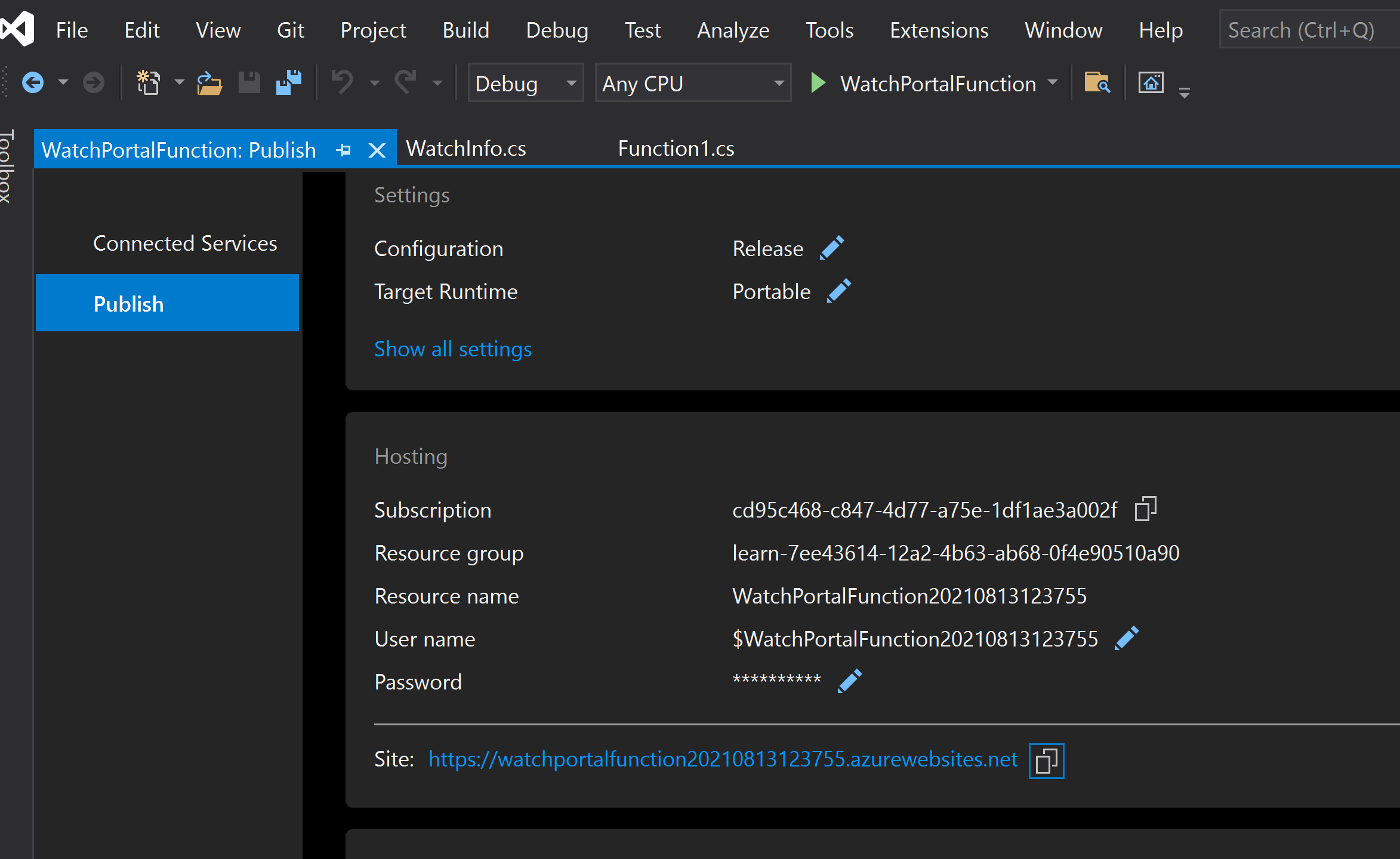


# Publish a simple Azure Function

1. The Publish wizard in the Azure Functions template provides the most straightforward way to deploy Azure Functions

## Create an Azure Function App using the Azure portal

1. The URL to use will be created and viewable in VS after the function has been published.



# Exercise - Unit test an Azure Function

1. Use the **xUnit** test framework with Visual Studio to test Azure Functions

# Create a function that is triggered by a webhook

1. What is a webhook?
   1. Webhooks are user-defined HTTP callbacks
   2. They're triggered by some event
   3. One common use of webhooks in a DevOps environment is to notify an Azure function that the code or configuration for an application has changed in GitHub
2. What is Azure Functions?
   1. This stuff is already known

# Exercise - Create an Azure function triggered by a webhook

# Set up a webhook for a GitHub repository

1. A webhook is triggered each time one or more subscribed events occur
2. For example, the Gollum event allows you to listen for wiki updates
3. Set up a webhook
   1. Setting up a webhook is a two-step process
   2. You specify how you want your webhook to behave through GitHub and what events it should listen to
   3. Then you set up your function in Azure Functions to receive and manage the payload received from the webhook
4. Content type
   1. Webhooks can be delivered using two different content types:
      1. The application/json content type delivers the JSON payload
      2. The application/x-www-form-urlencoded content type sends the JSON payload
5. Events
   1. Events are at the center of webhooks
   2. Events occur whenever actions are taken in the repository
   3. When the event occurs, the webhook fires and calls the URL that you specify, sending along the payload and event information to your URL

# Secure Webhook payloads with a secret

1. When you set a secret, you'll receive the x-hub-signature header in the webhook POST request
2. In GitHub, you can set the secret field by going to the repository where you have setup your webhook

## Validating payloads from GitHub

1. When your secret token is set, GitHub uses it to create a hash signature for each payload
2. The hash signature starts with the text sha1=