Create serverless applications

Azure Functions enable the creation of event driven, compute-on-demand systems that can be triggered by various external events. Learn how to leverage functions to execute server-side logic and build serverless architectures. This learning path can help you prepare for the Microsoft Certified: Azure Developer Associate certification.

Choose the best Azure service to automate your business processes

Microsoft Azure provides several different ways to host and execute code or workflows without using Virtual Machines (VMs) including Azure Functions, Microsoft Power Automate, Azure Logic Apps, and Azure WebJobs. In this module, you will learn about these technologies and how to choose the right one for a given scenario.

Business process may involve multiple steps, multiple people, and multiple software packages. Each process may run in a simple line of activities that workers perform one after the other or they may branch or loop. Each process may also run quickly or take days or weeks to complete. Business processes modeled in software are often called workflows.

Technologies on Azure:

- Logic Apps
- Microsoft Power Automate
- WebJobs
- Azure Functions

Similarities:

- They can all accept inputs. An input is a piece of data or a file that is supplied to the workflow.
- They can all run actions. An action is a simple operation that the workflow executes and may often modify data or cause another action to be performed.
- They can all include conditions. A condition is a test, often run against an input, that may decide which action to execute next.
- They can all produce outputs. An output is a piece of data or a file that is created by the workflow.
- In addition, workflows created with these technologies can either start based on a schedule or they can be triggered by some external event.

Design-first approach

Design-first approach has a user interface in which you can draw out the workflow.

Logic apps

Logic apps is a service that can be used to automate, orchestrate and integrate different components of a distributed applications. You can draw out complex workflows that model

comples business processes. You can either use designer to simply model the or use JSON notation to configure the flow.

Advantages:

- no line of code required
- create business processes and workflows visually
- integrate with SaaS and enterprise applications
- unlock the value from on-premises and cloud applications
- automate EAI,B2B/EDI, and business premises
- take advantage of the Microsoft Cloud to enhance your integration solutions

Integration with Logic Apps is easy because there are already 325+ connectors available. Connector is a Logic Apps component that provides an interface to an external service. Basically it is a proxy or a wrapper around the API. They define actions and triggers.

Actions are changes directed by user (send an email).

Triggers can notify your app when specific events occurs. (new email arrives)
- polling triggers: call the service at a specific frequency. When new data available, it causes a new run of your workflow instance with the data as an input
- push triggers: these triggers listen to data on an endpoint (they wait for a new event), then run your workflow instance.

If you have an unique system, you can create a custom connector if your system exposes a REST API.

1. build API
    a. communication REST oR SOAP
    b. public (visible on the public internet) or private (visible only to your network)
2. secure API with OAuth 2, Azure AD, Basic auth, API key,
3. define and describe the custom connector
    a. import the definition from swagger, postman collection or do it from scratch
4. use in logic apps or so
5. share with users in same organization
6. certify in order to share with all users

Each connector defines

- general and throttling limits (email size, api calls)
- actions e.g.:

Delete email
This operation is used to delete a specific email permanently.
Reply to email (V2)
This operation is used to reply to a specific email.
Reply to email [DEPRECATED]
This action has been deprecated. Please use Reply to email (V2) instead.
This operation is used to reply to a specific email.
- triggers, e.g.:

When a new email arrives
This operation triggers when a new email matching the specified criteria arrives.

Microsoft Power Automate

Microsoft Power Automate is a service that you can use to create workflows even you have no development or IT Pro experiencce. You can create workflows that integrate and orchestrate many different components by using the website or a mobile app.

4 different types of flow:

- automated: a flow that is started by a trigger (new tweet)
- button: just click to run the task
- scheduled: a flow that executes in defined period
- business process: a flow that models a business process (stock ordering process)

easy-to-use design surface, no coding needed is built onLogic Apps, so it supports the same connectors and actions customer connector is available.

Design-first comparison

	Microsoft Power Automate	Logic Apps
users	office workers and business analysts	developers and IT pros
scenarios	self-service workflows	advanced integration projects
design tools	GUI: browser and mobile app	Browser and Visual studio designer, code editing possible
app lifecycle	includes testing and prod envs	can be included in Azure DevOps and source code management systems

Code-first technologies

Write code to orchestrate and integrate different business applications into a single workflow when you need more control over performance or need to write custom code.

WebJobs and the WebJobs SDK

WebJobs are a part of Azure App Service (cloud-based hosting service for web applications, mobile back-ends and RESTful APIs) that you can use to run a program or script automatically. Write code in SHell Scripts, PHP, Python, Node.js, Java, .NET

two kinds:

- continuous: run in a loop (polling a shader folder for a new photo)
- triggered: run when you manually start them or on a scheduled

Azure functions

Simple way to run small pieces of code in the cloud, without having to worry about the infrastructure. Supports many languages. Can be automatically scaled.

templates:

- HTTPTrigger: in response to a request sent through the HTTP protocol
- TimeTrigger: according to schedule
- BlobTrigger: a new blob is added to an Azure Storage account
- CosmosDBTrigger: in response to new or updated documents in a NoSQL db

Can be integrated with many services within Azure and third parties. These services can trigger the function.

Code-first comparison

	Azure WebJobs	Azure Functions
supported langs	C# when using SDK	many
automatic scaling	no	yes
dev and test in browser	no	yes
pay-per-use pricing	no	yes
integration with LA	no	yes
package managers	NuGet when SDK	NuGet and NPM
part of App Service app	yes	no
close control JobHost	yes	no

Links

Choose the right integration and automation services in Azure
What is Azure Logic Apps?
What is Azure Logic Apps?
Run Background tasks with WebJobs in Azure App Service
An introduction to Azure Functions
Create a function that integrates with Azure Logic Apps

Create serverless logic with Azure Functions

Serverless compute can be thought of as a function as a service (FaaS), or a microservice that is hosted on a cloud platform. Your business login ruc runs as functions and you don't have to manually scaled out or down depending on load. Azure has several ways to build this sort of architecture. The two most common approaches are Azure Logic Apps and Azure Functions.

Serverless compute is a great option for hosting business logic code in the cloud. With serverless offerings such as Azure Functions. You can write your business logic in the language of your choice. You get automatic scaling, you have no servers to manage, and you are charged based on what is used - not on reserved time.

Authentication of HTTP triggers by a query param called **code** or by a custom HTTP header **x-functions-key.**

Summary

Which of the following best defines serverless logic?
Code you write that runs on servers a cloud provider manages.
Serverless doesn't mean there are no servers - it just means the developer doesn't have to worry about servers. Instead, a cloud provider such as Azure, manages servers.

The container that groups functions into a logical unit for easier management, deployment, and sharing of resources is called?
Function app
A function app is a way to organize and collectively manage your functions. A function app is comprised of one or more individual functions that are managed together by Azure App Service. All the functions in a function app share the same pricing plan, continuous deployment, and runtime version.

We secured our function against unknown HTTP callers by requiring a function-specific API key be passed with each call. Which of the following fields is the name header in the HTTP requests that needs to contain this key?
x-functions-key
The API can be included in a query string variable named "code", or it can be included in an x-functions-key HTTP header.

Links

Serverless compute

Execute an Azure Function with triggers

A trigger is responsible for executing an Azure function and there are dozens of triggers to choose from. This module will you show you some of the most common types of triggers and how to configure them to execute your logic.

A trigger is an object that defines how an Azure function is invoked. For example, if you want a function to execute every 10 minutes, you could use a timer trigger. Every function must have exactly one trigger associated with it. If you want to execute a piece of logic that runs under multiple conditions, you need to create multiple functions that share the same core function code.

Type of triggers

Type	Purpose
Timer	Execute a function at a set interval.
HTTP	Execute a function when an HTTP request is received.
Blob	Execute a function when a file is uploaded or updated in Azure Blob storage.
Queue	Execute a function when a message is added to an Azure Storage queue.
Azure Cosmos DB	Execute a function when a document changes in a collection.
Event Hub	Execute a function when an event hub receives a new event.

Timer trigger

Timer trigger is a trigger that executes a function at a consistent interval. To create a timer trigger, you need to supply two pieces of information:

1. A Timestamp parameter name, which is simply an identifier to access the trigger in code.
2. A Schedule, which is a CRON expression that sets the interval for the timer.

A *CRON expression* is a string that consists of six fields that represent a set of times.The order of the six fields in Azure is: {second} {minute} {hour} {day} {month} {day of the week}.

Special character	Meaning	Example
*	Selects every value in a field	An asterisk "*" in the day of the week field means *every* day.
,	Separates items in a list	A comma "1,3" in the day of the week field means just Mondays (day 1) and Wednesdays (day 3).
-	Specifies a range	A hyphen "10-12" in the hour field means a range that includes the hours 10, 11, and 12.
/	Specifies an increment	A slash "*/10" in the minutes field means an increment of every 10 minutes.

HTTP trigger

An HTTP trigger is a trigger that executes a function when it receives an HTTP request. HTTP triggers have many capabilities and customizations, including:

- Provide authorized access by supplying keys.
- Restrict which HTTP verbs are supported.
- Return data back to the caller.
- Receive data through query string parameters or through the request body.
- Support URL route templates to modify the function URL.

There are three Authorization levels:

- **function**: a key that is specific to a function. (host keys apply to all functions inside the function app.)
- **admin**: a key that requires host key
- **anonymous**: no authorization

Blob trigger

Azure Blob storage is an object storage solution that's designed to store large amounts of unstructured data.

For example, Azure Blob storage is great at doing things like:

- Storing files
- Serving files
- Streaming video and audio
- Logging data

There are three types of blobs: block blobs, append blobs, and page blobs. Block blobs are the most common type. They allow you to store text or binary data efficiently. Append blobs are like block blobs, but they're designed more for append operations like creating a log file that's being constantly updated. Finally, page blobs are made up of pages and are designed for frequent random read and write operations.

A blob trigger is a trigger that executes a function when a file is uploaded or updated in Azure Blob storage.

Summary

A CRON expression is a string that consists of six fields that represent a set of times. The order of the six fields in Azure is: {second} {minute} {hour} {day} {month} {day of the week}. Suppose you needed a CRON expression that meant "every day", what special character would you put in the {day of the week} position?

*

An asterisk specifies that every possible value should be selected. Having an asterisk in the {day of the week} field means that every day should be selected.

Suppose your Azure Function has a blob trigger associated with it and you want it to execute only when images are uploaded. Which of the following blob trigger *Path* values should you use?

samples-workitems/{name}.png

The Path tells the blob trigger where it should monitor for changes, and if there are any filters applied. Adding a file extension to the Path specifies that uploaded files must have that file extension in order for the trigger to invoke the function.

True or false: an Azure Function can have multiple triggers associated with it?

False

Every Azure Function must have exactly one trigger associated with it. If you want to use multiple triggers, you must create multiple functions.

Chain Azure Functions together using input and output bindings

Azure Functions makes it easy for your function code to integrate with data and services. Through the power of bindings, you declare the data sources to read and write, and let Azure Functions take care of the rest.

Bindings provide a declarative way to connect to data from within your code.

There are two kinds of bindings you can use with functions:

1. **Input binding** - An input binding is a connection to a data source. Our function can read data from these inputs.
2. **Output binding** - An output binding is a connection to a data destination. Our function can write data to these destinations.

There are also triggers, which are special types of input bindings that cause a function to execute. For example, an Azure Event Grid notification can be configured as a trigger. When an event occurs, the function will run.
A binding type can be used as an input, an output or both. For example, a function can write to Azure Blob Storage output binding, but a blob storage update could trigger another function.

Three properties are required in all bindings. You may have to supply additional properties based on the type of binding and storage you are using.

1. **Name** - Defines the function parameter through which you access the data. For example, in a queue input binding, this is the name of the function parameter that receives the queue message content
2. **Type** - Identifies the type of binding, i.e., the type of data or service we want to interact with.
3. **Direction** - Indicates the direction data is flowing, i.e., is it an input or output binding?

Additionally, most binding types also need a fourth property:

4. **Connection** - Provides the name of an app setting key that contains the connection string. Bindings use connection strings stored in app settings to keep secrets out of the function code. This makes your code more configurable and secure.

Input bindings allow you to connect your function to a data source. You can connect to several types of data sources, and the parameters for each vary. To resolve values from various sources, you can use binding expressions in the *function.json* file, function parameters, or code.

Input binding types

There are multiple types of input. However, not all types support both input and output. You'll use them whenever you want to ingest data of that type. Here, we'll look at the types that support input bindings and when to use them.

1. **Blob storage** - Blob storage bindings allow you to read from a blob.
2. **Azure Cosmos DB** - The Azure Cosmos DB input binding uses the SQL API to retrieve one or more Azure Cosmos DB documents and passes them to the input parameter of the function. The document ID or query parameters can be determined based on the trigger that invokes the function.
3. **Microsoft Graph** - Microsoft Graph input bindings allow you to read files from OneDrive, read data from Excel, and get auth tokens so you can interact with any Microsoft Graph API.
4. **Mobile Apps** - The Mobile Apps input binding loads a record from a mobile table endpoint and passes it into your function.
5. **Table storage** - You can read data and work with Azure Table storage.

To create a binding as an input, you must define the direction as in. The parameters for each type of binding may vary.

Output binding types

1. **Blob Storage** - You can use the blob output binding to write blobs.
2. **Azure Cosmos DB** - The Azure Cosmos DB output binding lets you write a new document to an Azure Cosmos DB database using the SQL API.
3. **Event Hubs** - Use the Event Hubs output binding to write events to an event stream. You must have send permission to an event hub to write events to it.
4. **HTTP** - Use the HTTP output binding to respond to the HTTP request sender. This binding requires an HTTP trigger and allows you to customize the response associated with the trigger's request. This can also be used to connect to web hooks.
5. **Microsoft Graph** - Microsoft Graph output bindings allow you to write to files in OneDrive, modify Excel data, and send email through Outlook.
6. **Mobile Apps** - The Mobile Apps output binding writes a new record to a Mobile Apps table.
7. **Notification Hubs** - You can send push notifications with Notification Hubs output bindings.
8. **Queue Storage** - Use the Azure Queue storage output binding to write messages to a queue.
9. **Send Grid** - Send emails using SendGrid bindings.
10. **Service Bus** - Use Azure Service Bus output binding to send queue or topic messages.
11. **Table storage** - Use an Azure Table storage output binding to write to a table in an Azure Storage account.
12. **Twilio** - Send text messages with Twilio.

To create a binding as an output, you must define the direction as out. The parameters for each type of binding may vary.

Tasks

Enhance add-bookmark function.

1. Create another function to read from Blob storage and other input bindings that we haven't used in this module.
2. Create another function to write to more destinations by using other supported output binding types.
3. In the preceding unit, we introduced a queue and posted messages to it with an output binding. As a next step, consider adding another function that reads the messages in the queue and prints out the MESSAGE TEXT to the console with console.log().

Links

Azure Functions documentation
Azure Serverless Computing Cookbook
How to use Queue storage from Node.js
Introduction to Azure Cosmos DB: SQL API
A technical overview of Azure Cosmos DB
Azure Cosmos DB documentation

Summary

1. Which of the following is an advantage of using bindings in your Azure Functions to access data sources and data sinks?
They simplify the connection process; you don't need to code specific connection logic.
Bindings provide a declarative way to connect to data. They let you avoid the complexity of coding connection logic like making a database connection or invoking web API interfaces.
2. What is the name of the file that contains function configuration data?
function.json
The configuration is named function.json, which contains JSON configuration data such as binding declarations.
3. How many triggers must a function have?
One
A function must have exactly one trigger.

Create a long-running serverless workflow with Durable Functions

Durable functions are an extension of Azure Functions. Whereas Azure Functions operate in a stateless environment, Durable Functions can retain state between function calls. This approach enables you to simplify complex stateful executions in a serverless-environment.

Durable Functions scales as needed, and provides a cost effective means of implementing complex workflows in the cloud. Some benefits of using Durable Functions include:

- They enable you to write event driven code. A durable function can wait asynchronously for one or more external events, and then perform a series of tasks in response to these events.
- You can chain functions together. You can implement common patterns such as fan-out/fan-in, which uses one function to invoke others in parallel, and then accumulate the results.
- You can orchestrate and coordinate functions, and specify the order in which functions should execute.
- The state is managed for you. You don't have to write your own code to save state information for a long-running function.

Durable functions allows you to define stateful workflows using an Orchestration function. An orchestration function provides these extra benefits:

- You can define the workflows in code. You don't need to write a JSON description or use a workflow design tool.
- Functions can be called both synchronously and asynchronously. Output from the called functions is saved locally in variables and used in subsequent function calls.
- Azure checkpoints the progress of a function automatically when the function awaits. Azure may choose to dehydrate the function and save its state while the function waits, to preserve resources and reduce costs. When the function starts running again, Azure will rehydrate it and restore its state.

Function Types

You can use three durable function types: *Client*, *Orchestrator*, and *Activity*.

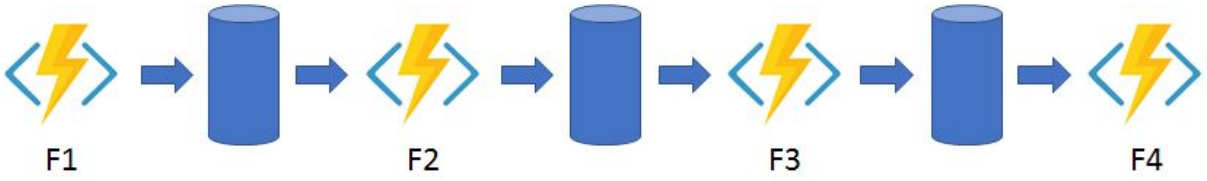
Client functions are the entry point for creating an instance of a Durable Functions orchestration. They can run in response to an event from many sources, such as a new HTTP request arriving, a message being posted to a message queue, an event arriving in an event stream. You can write them in any of the supported languages.

Orchestrator functions describe how actions are executed, and the order in which they are run. You write the orchestration logic in code (C# or JavaScript).

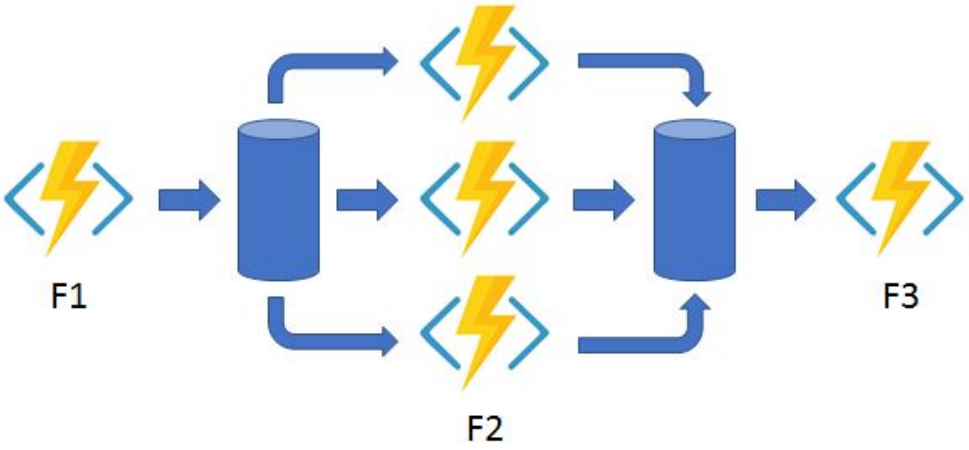
Activity functions are the basic units of work in a durable function orchestration. An activity function contains the actual work performed by the tasks being orchestrated.

Application patterns

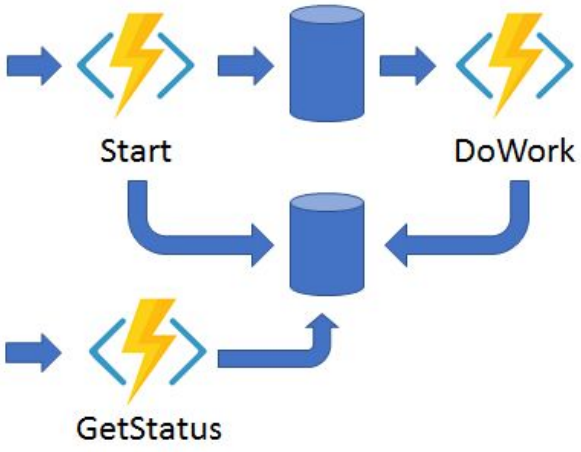
Function chaining - In this pattern, the workflow executes a sequence of functions in a specified order. The output of one function is applied to the input of the next function in the sequence. The output of the final function is used to generate a result.



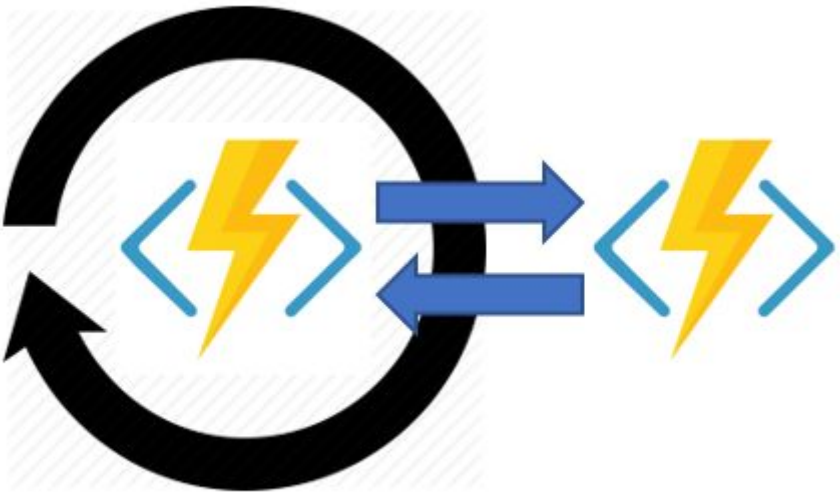
Fan out/fan in - This pattern runs multiple functions in parallel and then waits for all the functions to finish. The results of the parallel executions can be aggregated or used to compute a final result.



Async HTTP APIs - This pattern addresses the problem of coordinating state of long-running operations with external clients. An HTTP call can trigger the long-running action. Then, it can redirect the client to a status endpoint. The client can learn when the operation is finished by polling this endpoint.



Monitor - This pattern implements a recurring process in a workflow, possibly looking for a change in state. For example, you could use this pattern to poll until specific conditions are met.



Human interaction - This pattern combines automated processes that also involve some human interaction. A manual process within an automated process is tricky because people aren't as highly available and as responsive as most computers. Human interaction can be incorporated using timeouts and compensation logic that runs if the human fails to interact correctly within a specified response time. An approval process is an example of a process that involves human interaction.

Comparison with Logic Apps

Durable Functions and Logic Apps are both Azure services that enable serverless workload. Azure Durable Functions is intended as a powerful serverless compute option to run custom logic. Azure Logic Apps is better suited for integrating Azure services and components. You can use either technology to create complex orchestrations. With Azure Durable Functions, you develop orchestrations by writing code and using the Durable Functions extension. With Logic Apps, you create orchestrations by using the design surface or editing configuration files.

Azure Durable Functions	**Azure Logic Apps**	
Development	Code-first (imperative)	Design-first (declarative)
Connectivity	About a dozen built-in binding types. You can write code for custom bindings.	Large collection of connectors. Enterprise Integration Pack for B2B.You can also build custom connectors.
Actions	Each activity is an Azure Function. You write the code for activity functions.	Large collection of ready-made actions. You integrate custom logic through custom connectors.
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	Runs only in the cloud

Durable Functions provides timers for use in the orchestrator functions. They can implement delays or set up timeouts for asynchronous actions.

Links

Durable Functions patterns and technical concepts

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

The Azure Functions Core Tools are command-line utilities that enable you to develop and run functions locally and publish them to Azure.

Function apps and functions projects

Every function published to Azure belongs to a *function app*: a collection of functions that are published together into the same environment. All of the functions in an app share a common set of configuration values, and must all be built for the same language runtime. Each function app is an Azure resource that can be configured and managed independently.

When you develop functions locally, you work within a *functions project*: a folder that contains the code and configuration files that define your functions. A functions project on your computer is equivalent to a function app in Azure, and can contain multiple functions that use the same language runtime.

When you create a new functions project, the files included in the project folder depend on the language runtime you select. Regardless of the runtime you choose, the two most critical project files are always present:

- **host.json** stores runtime configuration values, such as logging options, for the function app. The settings stored in this file are used both when running functions locally and in Azure.
- **local.settings.json** stores configuration values that only apply to the function app when it is run locally with the Core Tools. This file contains two kinds of settings: local runtime settings, used to configure the local functions runtime itself, and custom application settings, which you can add and configure based on your app's needs and can be accessed and used by all the functions in the app.

Develop, test, and deploy an Azure Function with Visual Studio

Serverless architecture makes use of infrastructure provided by the cloud. You don't have to provision, manage, scale, and maintain any machinery or networks. Azure Functions is a fully managed PaaS service offered by Microsoft Azure to implement serverless architecture. Azure Functions is a fully scalable, resilient, reliable, and secure service.

An Azure Function runs in the cloud in the context of an Azure Function App. A function app is a container that specifies the operating system for running an Azure Function, together with the resources available, such as the memory, computing power, and disk space. The Azure Function App also provides the public URL for running your functions. Behind the scenes, an Azure Function App is a collection of one or more virtual machines, running a web server. When you publish an Azure Function, you deploy it to these virtual machines.

Azure Functions makes it easy to deploy your function app using App Service continuous integration. Azure Functions integrates with BitBucket, Dropbox, GitHub, and Azure DevOps. This enables a workflow where function code updates made by using one of these integrated services triggers deployment to Azure.

Azure Function can be deployed from a zip file using the push deployment technique. You can do this with the Azure CLI, or by using the REST interface.

The zip file contains the executable code for your functions. Zip deployment copies these files to the wwwwroot folder in the function app. You can perform zip deployment using the functionapp deployment command in the Azure CLI.

```
az functionapp deployment source config-zip \
-g <resource-group> \
-n <function-app-name> \
--src <zip-file>
```

Function apps deployed from Visual studio or with Azure CLI, cannot be directly changed in Azure portal.

Links

An introduction to Azure Functions
Develop Azure Functions using Visual Studio
Create your first function using Visual Studio
Strategies for testing your code in Azure Functions

Monitor GitHub events by using a webhook with Azure Functions

Webhooks offer a lightweight mechanism for apps to be notified by another service when something of interest happens via an HTTP endpoint.

What is a webhook?

Webhooks are user-defined HTTP callbacks. They're triggered by some event, such as pushing code to a repo or updating a wiki page. When the event occurs, the source site makes an HTTP request to the URL configured for the webhook. With Azure Functions, we can define logic in a function that can be run when a webhook message is received.

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. The payload of the message sent through the webhook contains the details of the event. You can use the webhook with a function to perform a task such as deploying the updated version of the application.

What is Azure Functions?

Azure Functions is a serverless compute service. It enables you to run code without having to explicitly provision or manage any infrastructure. You can use Azure Functions to run script or code in response to a variety of events.

A Trigger causes a function to run. A trigger defines how a function is invoked. A function must have exactly one trigger. Triggers have associated data, which is often provided as the payload of the function.

A Binding is used to connect a resource to a function. You can define input bindings and output bindings. Data from a binding is provided to the function as parameters. For example, you can connect a database to your Azure Functions code by using a binding. Then you don’t need to wire up the database to the function with any connection code. Bindings are optional and a function might have one or multiple input and/or output bindings.

Webhook secrets

Setting a webhook secret allows you to ensure that POST requests sent to the payload URL are from GitHub. When you set a secret, you'll receive the x-hub-signature header in the webhook POST request.

Validating payloads from GitHub

When your secret token is set, GitHub uses it to create a hash signature for each payload. This hash signature is passed along with each request in the headers as x-hub-signature.

When your function receives a request, you need to compute the hash using your secret, and ensure that it matches the hash in the request header. GitHub uses an HMAC SHA1 hex digest to compute the hash, so you must calculate your hash in this same way, using the key of your secret and your payload body. The hash signature starts with the text sha1=.

Links

GitHub Webhooks
Gollum event
Azure Functions HTTP triggers and bindings
Securing your webhooks
HTTP and webhooks

Enable automatic updates in a web application using Azure Functions and SignalR Service

SignalR is an abstraction for a series of technologies that allows your app to enjoy two-way communication between the client and server. SignalR handles connection management automatically, and lets you broadcast messages to all connected clients simultaneously, like a chat room. You can also send messages to specific clients. The connection between the client and server is persistent, unlike a classic HTTP connection, which is re-established for each communication.

A key benefit of the abstraction provided by SignalR is the way it supports "transport" fallbacks. A transport is method of communicating between the client and server. SignalR connections begin with a standard HTTP request. As the server evaluates the connection, the most appropriate communication method (transport) is selected. Transports are chosen depending on the APIs available on the client.

For clients that support HTML 5, the WebSockets API transport is used by default. If the client doesn't support WebSockets, then SignalR falls back to Server Sent Events (also known as EventSource). For older clients, Ajax long polling or Forever Frame (IE only) is used to mimic a two-way connection.

The abstraction layer offered by SignalR provides two benefits to your application. The first advantage is future-proofing your app. As the web evolves and APIs superior to WebSockets become available, your application doesn't need to change. You could update to a version of SignalR that supports any new APIs and your application code won't need an overhaul.

The second benefit is that SignalR allows your application to gracefully degrade depending on supported technologies of the client. If it doesn't support WebSockets, then Server Sent Events are used. If the client can't handle Server Sent Events, then it uses Ajax long polling, and so on.

Links

Azure Cosmos DB input bindings for Azure Functions

Expose multiple Azure Function apps as a consistent API by using Azure API Management

Serverless architecture and microservices

Microservices has become a popular approach to the architecture of distributed applications in recent years. When you build an application as a collection of microservices, you create many different small services. Each service has a defined domain of responsibility, and is developed, deployed, and scaled independently. This modular architecture results in an application that is easier to understand, improve, and test. It also makes continuous delivery easier because, when you deploy a microservice, you change only a small part of the whole application.

Another complimentary trend in distributed software development is serverless architecture. In this approach, a host organization publishes a set of services that developers can use to run their code. The developers do not have to concern themselves with the supporting hardware, operating systems, underlying software, and other infrastructure. Instead the code is run in stateless computing resources that are triggered by requests. Costs are only incurred when the services execute so you don't pay much for services that are rarely used.

Azure Functions

Azure Functions is a service that enables serverless architectures in Azure. You can write functions, without worrying about the supporting infrastructure, in many different languages, including C#, Java, JavaScript, PowerShell, and Python. You can also use libraries from NuGet and the Node Package Manager (NPM) and authenticate users with the OAuth standard from providers such as Active Directory, Facebook, Google, and Microsoft Account.

When you write a function, you choose a template to use, depending on how you want to trigger your code. For example, if you want to execute the function in response to an HTTP request, use the HTTPTrigger template. You can use other templates to execute when there are new messages in a queue, a Blob storage container, or on a predefined schedule.

When you use Azure Functions in a Consumption Plan, you are charged only for the time that your code runs.

Azure API Management

Azure API Management (APIM) is a fully managed cloud service that you can use to publish, secure, transform, maintain, and monitor APIs. It helps organizations publish APIs to external, partner, and internal developers to unlock the potential of their data and services. API Management handles all the tasks involved in mediating API calls, including request authentication and authorization, rate limit and quota enforcement, request and response transformation, logging and tracing, and API version management. APIM enables you to create and manage modern API gateways for existing backend services no matter where they are hosted.

Because you can publish Azure Functions through API Management, you can use them to implement a microservices architecture: Each function implements a microservice. By adding many functions to a single API Management product, you can build those microservices into an integrated distributed application. Once the application is built, you can use API Management policies to implement caching or ensure security requirements.

APIM Consumption Tier

When you choose a usage plan for API Management, you can choose the consumption tier, because it's especially suited to microservice-based architectures and event-driven systems. For example, it would be a great choice for our online store web API.

The consumption tier uses the same underlying service components as the previous tiers but employs an entirely different architecture based on shared, dynamically allocated resources. It aligns perfectly with serverless computing models. There is no infrastructure to manage, no idle capacity, high-availability, automatic scaling, and usage-based pricing, all of which make it an especially good choice for solutions that involve exposing serverless resources as APIs.

By adding multiple APIs, functions, and other services to API Management, you can assemble those components into an integrated product that presents a single entry point to client applications. Composing an API using API Management has advantages that include:

- Client apps are coupled to the API expressing business logic, not the underlying technical implementation with individual microservices. You can change the location and definition of the services without reconfiguring or updating the client apps.
- API Management acts as an intermediary. It forwards requests to the right microservice, wherever it is located, and returns responses to users. Users never see the different URIs where microservices are hosted.
- You can use API Management policies to enforce consistent rules on all microservices in the product. For example, you can transform all XML responses into JSON, if that is your preferred format.
- Policies also enable you to enforce consistent security requirements.

API Management also includes helpful tools - you can test each microservice and its operations to ensure that they behave in accordance with your requirements. You can also monitor the behavior and performance of deployed services.

Azure API Management supports importing Azure Function Apps as new APIs or appending them to existing APIs. The process automatically generates a host key in the Azure Function App, which is then assigned to a named value in Azure API Management.

Links

An introduction to Azure Functions
API Management documentation
Import an Azure Function App as an API in Azure API Management