

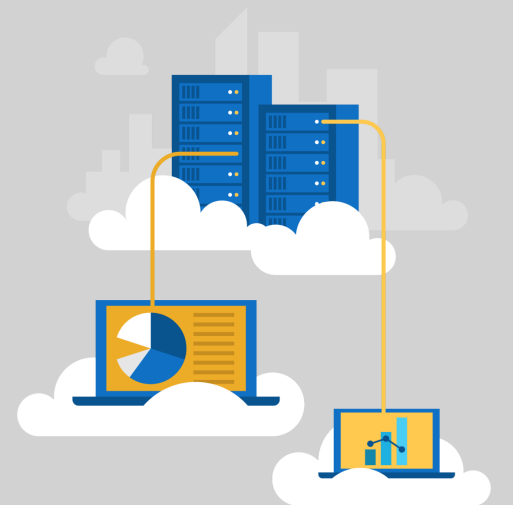
Module 03: Implement Azure Functions



Topics

- Azure Functions overview
- Developing Azure Functions
- Implement Durable Functions

Lesson 01: Azure Functions overview



Azure Functions

- Solution for running small pieces of code, or "functions," in the cloud:
 - Write only code that is relevant to business logic
 - Removes the necessity to write "plumbing" code to connect or host application components
- Build on open-source WebJobs code
- Supports a wide variety of programming languages, for instance:



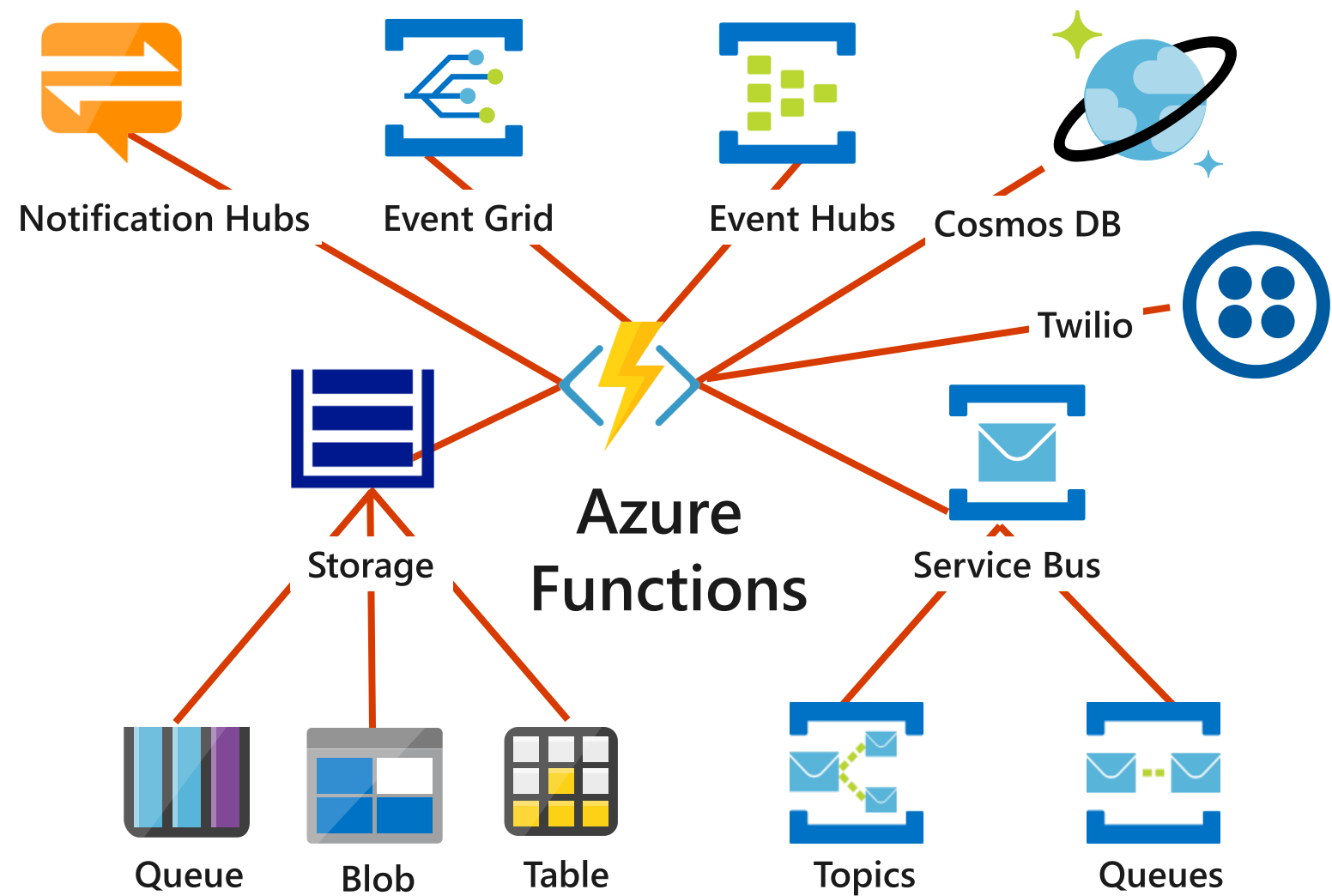
What can Azure Functions do?

- Run code based on HTTP requests
- Schedule code to run at predefined times
- Process new and modified:
 - Azure Cosmos DB documents
 - Azure Storage blobs
 - Azure Queue storage messages
- Respond to Azure Event Grid events by using subscriptions and filters
- Respond to high volumes of Azure Event Hubs events
- Respond to Azure Service Bus queue and topic messages

Use Cases

- Implement Simple APIs
- Migration / Reusing Parts from old Monoliths
 - PDF Generator with Custom *.dll that works just fine
- Security
 - Hiding Keys from SPA UIs -> Google Maps Key ...
- Integration(s)
 - Responding to Microsoft / Office 365 Webhook (SharePoint, Graph, ... whatever)
 - Logic Apps, Event Hub, Blob Storage, ...

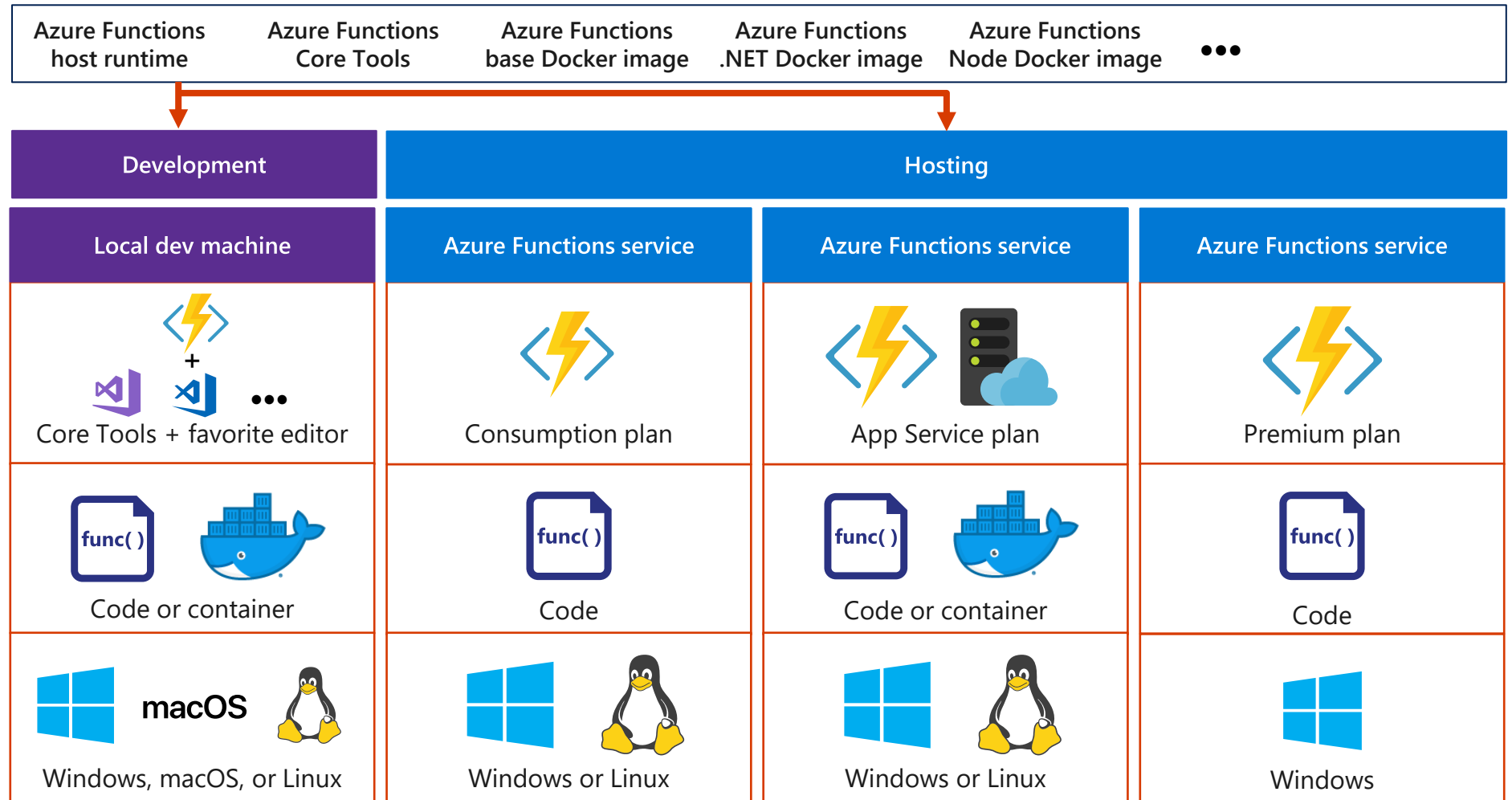
Function integrations



Azure Functions hosting



<https://github.com/azure/azure-functions-host> (+other repos)



Azure Functions hosting (continued)



<https://github.com/azure/azure-functions-host> (+other repos)

Azure Functions
host runtime

Azure Functions
Core Tools

Azure Functions
base Docker image

Azure Functions
.NET Docker image

Azure Functions
Node Docker image

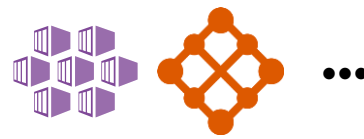
...

Hosting

Platform



Azure IoT Edge



AKS, Service Fabric Mesh, ...



...

K8s, raw VMs, & more



App Service on Azure Stack

App delivery



Container



Container



Container



Code

OS



Linux



Linux



Linux



Windows

Scale and hosting

- You can choose between two types of plans:
 - Consumption:
 - Instances are dynamically instanced and you are charged based on compute time
 - App Service plan:
 - Traditional App Services model used with Web Apps, API Apps, and Mobile Apps
- The type of plan controls:
 - How host instances are scaled out
 - The resources that are available to each host

Lesson 02: Developing Azure Functions



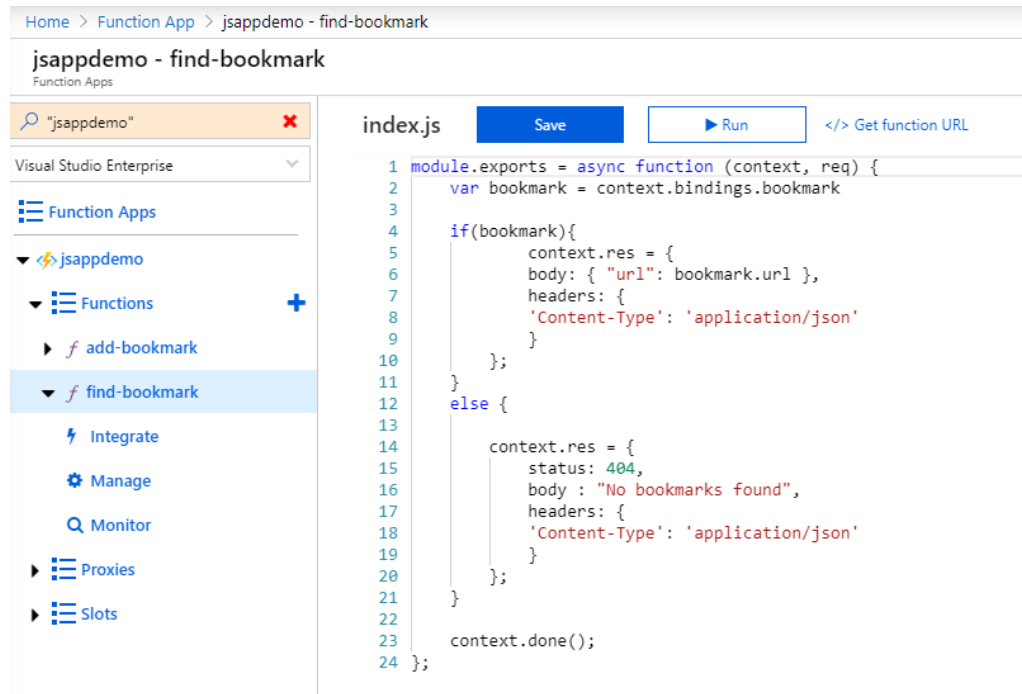
Editing / Scaffolding

- Azure Portal
 - Uses *.csx
- VS Code or VS Professional
 - Uses *.cs compiled to assembly
- Azure Function Tools
 - npm install -g azure-functions-core-tools
- Azure Function Extension

[illegible]

Function Edition Azure Portal

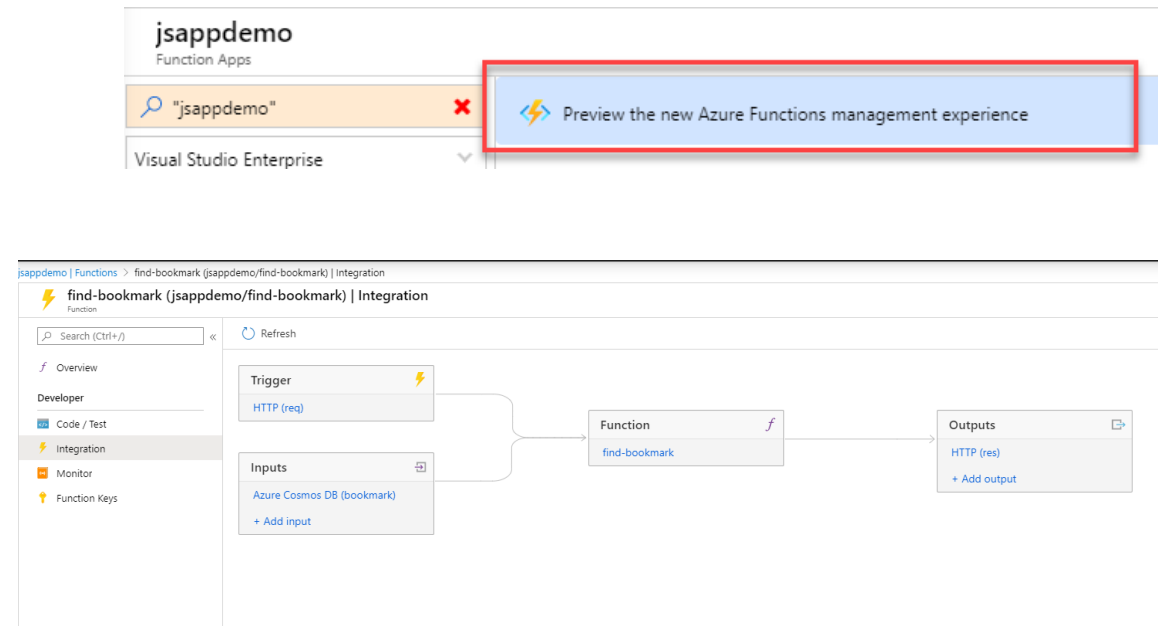
- 2 Versions: Old vs Preview
- Set Bindings here -> export



The screenshot shows the Azure Portal interface for a Function App named 'jsappdemo'. The left sidebar contains a navigation menu with 'Function Apps' and 'Functions' sections. The 'Functions' section is expanded, showing 'add-bookmark' and 'find-bookmark'. The 'find-bookmark' function is selected. The main area displays the 'index.js' file with the following code:

```
1 module.exports = async function (context, req) {
2   var bookmark = context.bindings.bookmark
3
4   if(bookmark){
5     context.res = {
6       body: { "url": bookmark.url },
7       headers: {
8         'Content-Type': 'application/json'
9       }
10    };
11  }
12  else {
13
14    context.res = {
15      status: 404,
16      body : "No bookmarks found",
17      headers: {
18        'Content-Type': 'application/json'
19      }
20    };
21  }
22  context.done();
23 }
24
```

Buttons for 'Save', 'Run', and 'Get function URL' are visible above the code editor.



The screenshot shows the 'Integration' view for the 'find-bookmark' function. A red box highlights a banner that says 'Preview the new Azure Functions management experience'. The function configuration is as follows:

- Trigger:** HTTP (req)
- Inputs:** Azure Cosmos DB (bookmark), + Add input
- Function:** find-bookmark
- Outputs:** HTTP (res), + Add output

The left sidebar shows the 'Integration' tab selected under the 'Developer' section.

Azure Functions in Visual Studio Code

- Use the Azure Functions extension for Visual Studio Code to:
 - Build and run functions locally
 - Publish functions to Azure
 - Build C# pre-compiled class libraries
 - Build C# scripts by adjusting the extension settings
- Use the many built-in features and extensions for Visual Studio Code to make development easier

Azure Functions in Visual Studio

- Visual Studio project type:
 - Develop, test, and deploy C# functions to Azure
 - Requires an Azure development workload installation
- Use WebJobs attributes to configure functions in C#
- Precompile C# functions:
 - Better cold-start performance

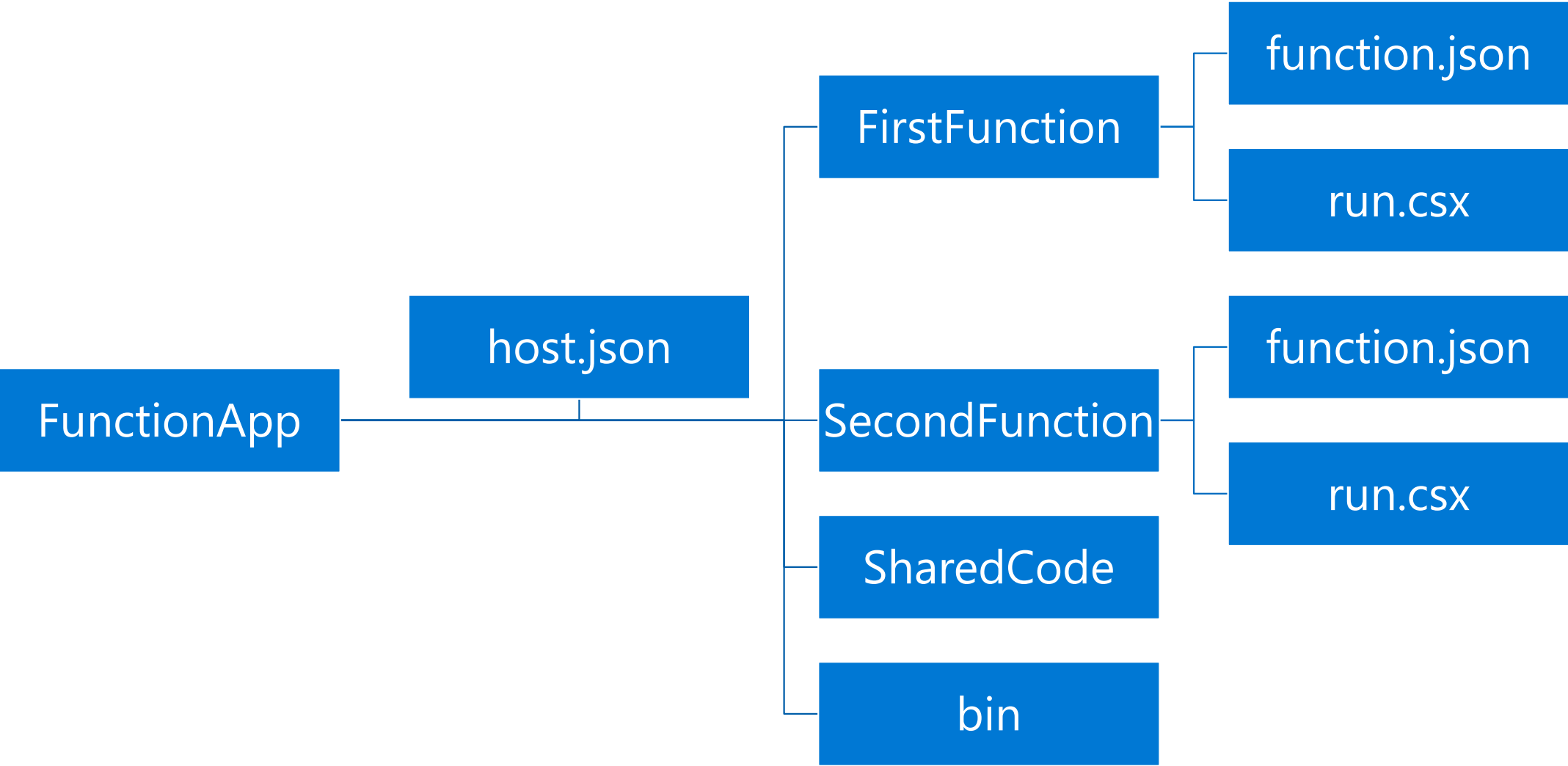
Develop Functions using Core Tools

- The Core Tools feature a variety of functions-related capabilities, but their primary purpose is to:
 - Generate the files and folders you need to develop functions on your local computer
 - Run your functions locally so you can test and debug them
 - Publish your functions to Azure
- `npm i -g azure-functions-core-tools`

```
user@Azure:~/myFunctions$ func init
Select a worker runtime:
1. dotnet
2. node
3. python
4. powershell
Choose option: 2
node
Select a Language:
1. javascript
2. typescript
Choose option: 1
javascript
Writing package.json
Writing .gitignore
Writing host.json
Writing local.settings.json
Writing /home/user/myFunctions/.vscode/extensions.json
```

```
user@Azure:~/myFunctions$ func new
Select a template:
1. Azure Blob Storage trigger
2. Azure Cosmos DB trigger
3. Durable Functions activity
4. Durable Functions HTTP starter
5. Durable Functions orchestrator
6. Azure Event Grid trigger
7. Azure Event Hub trigger
8. HTTP trigger
9. IoT Hub (Event Hub)
10. Azure Queue Storage trigger
11. SendGrid
12. Azure Service Bus Queue trigger
13. Azure Service Bus Topic trigger
14. Timer trigger
Choose option: 8
HTTP trigger
Function name: [HttpTrigger] myHttpTriggeredFunction
Writing /home/user/myFunctions/myHttpTriggeredFunction/index.js
Writing /home/user/myFunctions/myHttpTriggeredFunction/function.json
The function "myHttpTriggeredFunction" was created successfully from the "HTTP trigger" template.
```


Function folder structure

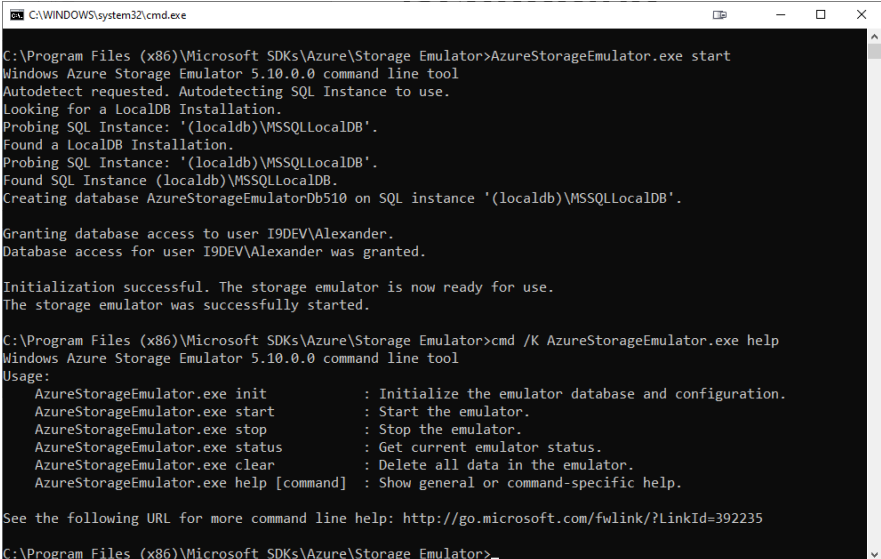


Config Files

- host.json
 - Global configuration options that affect all functions for a function app.
- function.json
 - Defines the function's trigger, bindings, and other configuration settings
- local.settings.json
 - Stores app settings, connection strings, and settings used by local development tools

Debugging

- Code of the Function App is Stored in the Storage Group
 - Run from zip option available
- Local Debugging is supported using Azure Storage Emulator
 - Can be accessed using Storage Explorer
 - Storage Explorer uses LocalDB by default



```
C:\WINDOWS\system32\cmd.exe

C:\Program Files (x86)\Microsoft SDKs\Azure\Storage Emulator>AzureStorageEmulator.exe start
Windows Azure Storage Emulator 5.10.0.0 command line tool
Autodetect requested. Autodetecting SQL Instance to use.
Looking for a LocalDB Installation.
Probing SQL Instance: '(localdb)\MSSQLLocalDB'.
Found a LocalDB Installation.
Probing SQL Instance: '(localdb)\MSSQLLocalDB'.
Found SQL Instance (localdb)\MSSQLLocalDB.
Creating database AzureStorageEmulatorDb510 on SQL instance '(localdb)\MSSQLLocalDB'.

Granting database access to user I9DEV\Alexander.
Database access for user I9DEV\Alexander was granted.

Initialization successful. The storage emulator is now ready for use.
The storage emulator was successfully started.

C:\Program Files (x86)\Microsoft SDKs\Azure\Storage Emulator>cmd /K AzureStorageEmulator.exe help
Windows Azure Storage Emulator 5.10.0.0 command line tool
Usage:
  AzureStorageEmulator.exe init           : Initialize the emulator database and configuration.
  AzureStorageEmulator.exe start          : Start the emulator.
  AzureStorageEmulator.exe stop           : Stop the emulator.
  AzureStorageEmulator.exe status         : Get current emulator status.
  AzureStorageEmulator.exe clear         : Delete all data in the emulator.
  AzureStorageEmulator.exe help [command] : Show general or command-specific help.

See the following URL for more command line help: http://go.microsoft.com/fwlink/?LinkId=392235

C:\Program Files (x86)\Microsoft SDKs\Azure\Storage Emulator>
```











Demonstration: Creating an HTTP trigger function by using the Azure portal



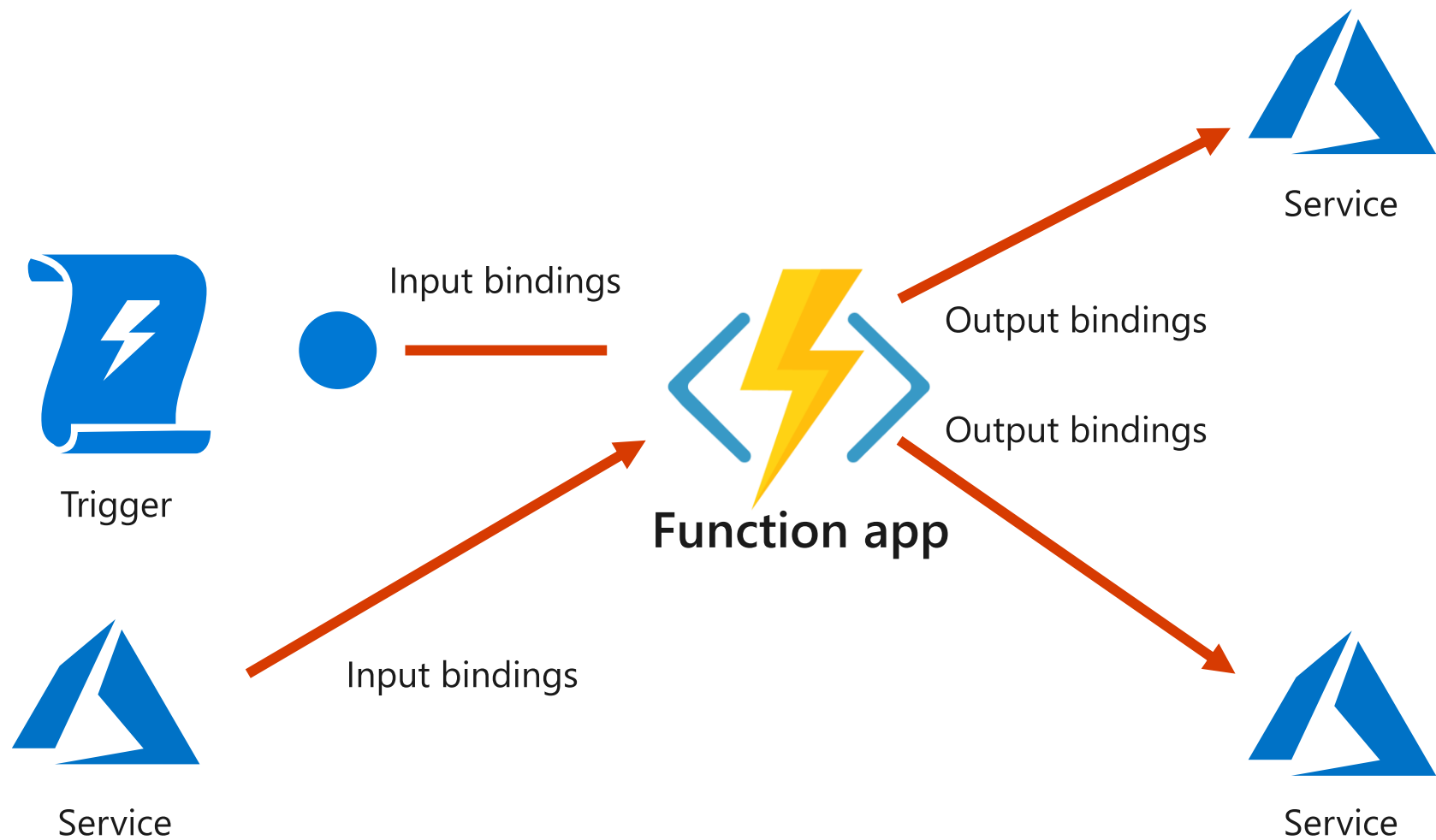
Trigger types

- Triggers based on Azure services
 - Cosmos DB
 - Blob and queues
 - Service Bus
 - Event Hub
- Triggers based on common scenarios
 - HTTP request
 - Scheduled timer
- Triggers based on third-party services
 - GitHub
- And more...

[Templates](#) [Details](#)

 HTTP trigger Eine Funktion, die bei jedem Empfang einer HTTP-Anforderung ausgeführt wird und basierend auf Daten im Textkörper oder der Abfragezeichenfolge antwortet	 Timer trigger Eine Funktion, die nach einem angegebenen Zeitplan ausgeführt wird
 Azure Queue Storage trigger Eine Funktion, die immer dann ausgeführt wird, wenn einer angegebenen Azure Storage-Warteschlange eine Nachricht hinzugefügt wird	 Azure Service Bus Queue trigger A function that will be run whenever a message is added to a specified Service Bus queue
 Azure Service Bus Topic trigger A function that will be run whenever a message is added to the specified Service Bus topic	 Azure Blob Storage trigger Eine Funktion, die immer dann ausgeführt wird, wenn einem angegebenen Container ein Blob hinzugefügt wird
 Azure Event Hub trigger Eine Funktion, die immer dann ausgeführt wird, wenn ein Event Hub ein neues Ereignis empfängt	 Azure Cosmos DB trigger A function that will be run whenever documents change in a document collection
 IoT Hub (Event Hub) A function that will be run whenever an IoT Hub receives a new event from IoT Hub (Event Hub)	 SendGrid Eine Funktion, die eine Bestätigungs-E-Mail sendet, wenn einer bestimmten Warteschlange ein neues Element hinzugefügt wird

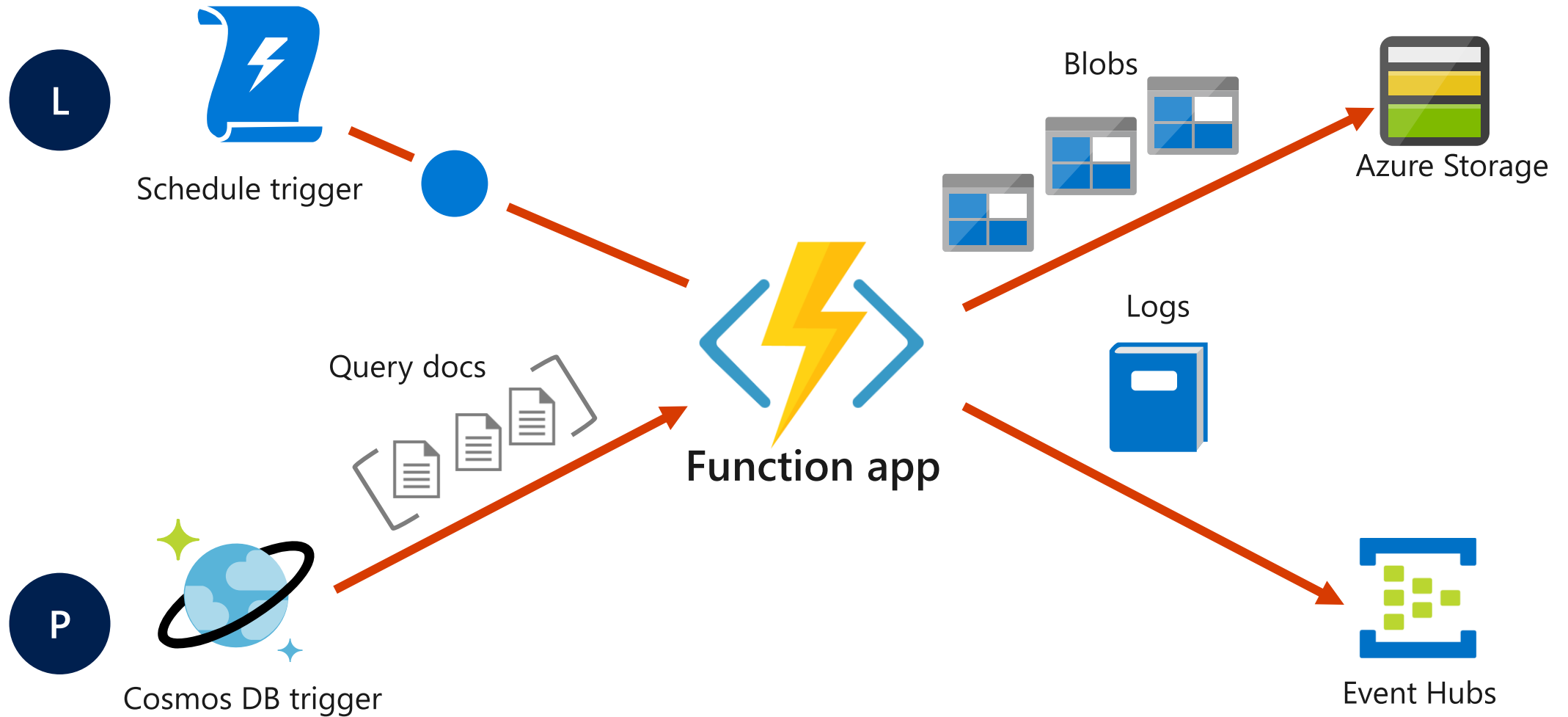
Input and Output Bindings



Bindings

- Declarative way to connect to data from your code
 - Connect to services without writing plumbing code
 - Service credentials are not stored in code
 - Bindings are optional
- Function can have multiple input and output bindings
- Output bindings can send data to Azure services such as
 - Storage
 - Azure Cosmos DB
 - Service Bus

Trigger and Bindings example



Triggers & Bindings

Example scenario	Trigger	Input binding	Output binding						
A new queue message arrives which runs a function to write to another queue.	Queue*	None	Queue*	S	are supported in the major versions of the Azure Functions runtime:				
					1.x	2.x and higher ¹	Trigger	Input	Output
A scheduled job reads Blob Storage contents and creates a new Cosmos DB document.	Timer	Blob Storage	Cosmos DB	✓	✓	✓	✓	✓	
The Event Grid is used to read an image from Blob Storage and a document from Cosmos DB to send an email.	Event Grid	Blob Storage and Cosmos DB	SendGrid	✓	✓	✓	✓	✓	
				✓	✓	✓		✓	
				✓	✓	✓		✓	
				✓	✓	✓		✓	
				✓	✓	✓		✓	
A webhook that uses Microsoft Graph to update an Excel sheet.	HTTP	None	Microsoft Graph		✓		✓	✓	
		Microsoft Graph OneDrive files		✓		✓	✓		
		Microsoft Graph Outlook email		✓			✓		

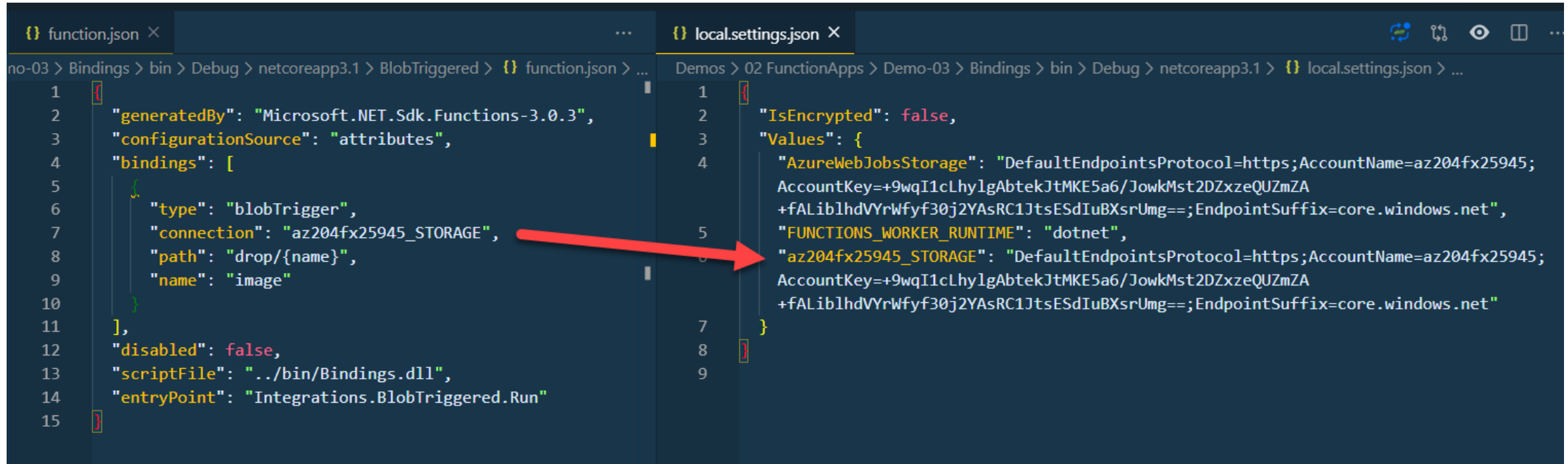
Function code

```
using System;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;

namespace FunctionApp1
{
    public static class Function1
    {
        [FunctionName("QueueTriggerCSharp")]
        public static void Run([QueueTrigger("myqueue-items", Connection =
"QueueStorage")]string myQueueItem, TraceWriter log)
        {
            log.Info($"C# Queue trigger function processed: {myQueueItem}");
        }
    }
}
```



Bindings – function.json



```
function.json
1 {
2   "generatedBy": "Microsoft.NET.Sdk.Functions-3.0.3",
3   "configurationSource": "attributes",
4   "bindings": [
5     {
6       "type": "blobTrigger",
7       "connection": "az204fx25945_STORAGE",
8       "path": "drop/{name}",
9       "name": "image"
10    }
11  ],
12  "disabled": false,
13  "scriptFile": "../bin/Bindings.dll",
14  "entryPoint": "Integrations.BlobTriggered.Run"
15 }

local.settings.json
1 {
2   "IsEncrypted": false,
3   "Values": {
4     "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=az204fx25945;AccountKey=+9wqI1cLhylgAbtekJtMKE5a6/JowkMst2DZxzeQUZmZA+fALib1hdVYrWfyf30j2YAsRC1JtsESdIuBXsrUmg==;EndpointSuffix=core.windows.net",
5     "FUNCTIONS_WORKER_RUNTIME": "dotnet",
6     "az204fx25945_STORAGE": "DefaultEndpointsProtocol=https;AccountName=az204fx25945;AccountKey=+9wqI1cLhylgAbtekJtMKE5a6/JowkMst2DZxzeQUZmZA+fALib1hdVYrWfyf30j2YAsRC1JtsESdIuBXsrUmg==;EndpointSuffix=core.windows.net"
7   }
8 }
```

Binding-based code

```
#r "Newtonsoft.Json"

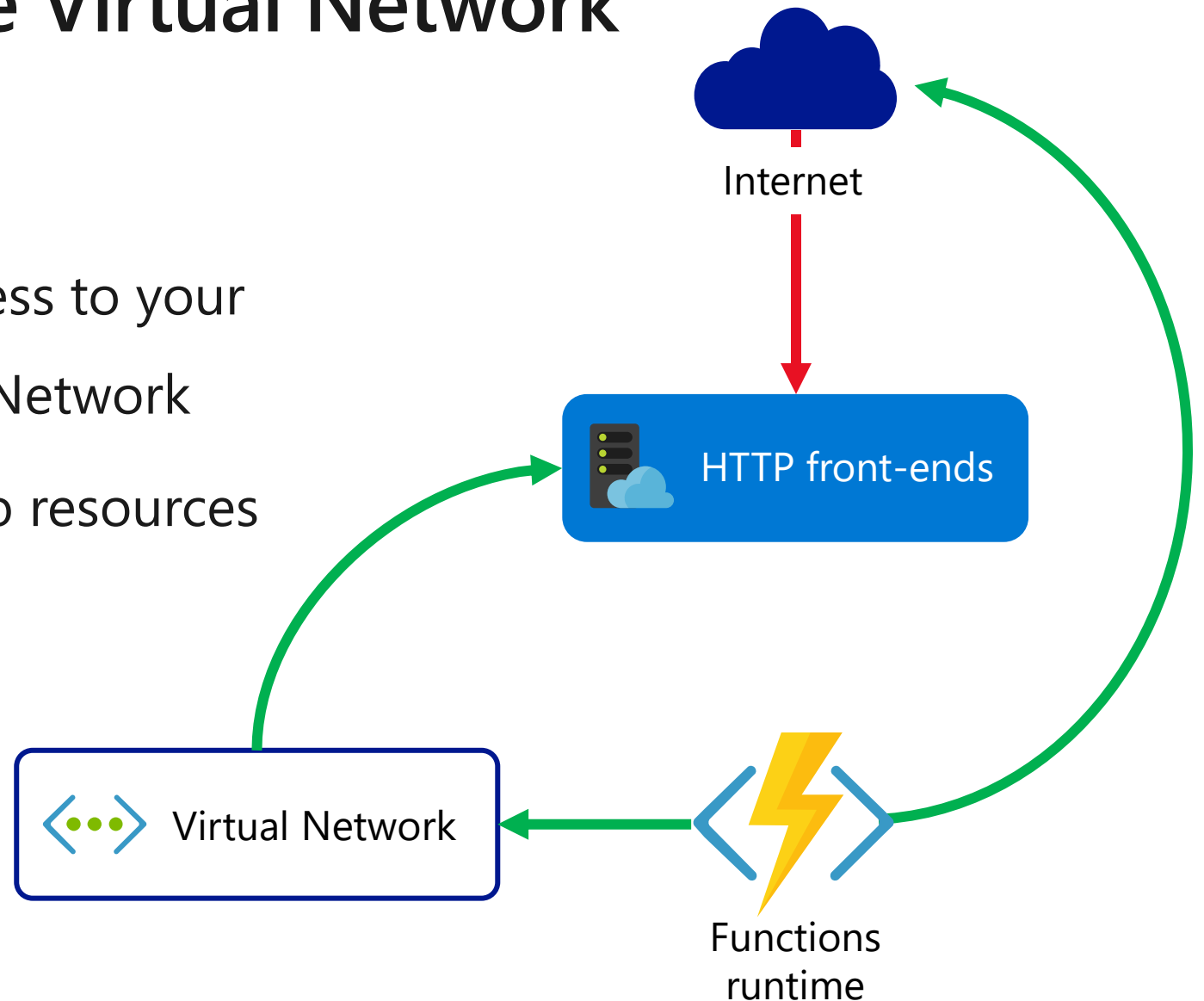
using Microsoft.Extensions.Logging;
using Newtonsoft.Json.Linq;

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()
    };
}
```

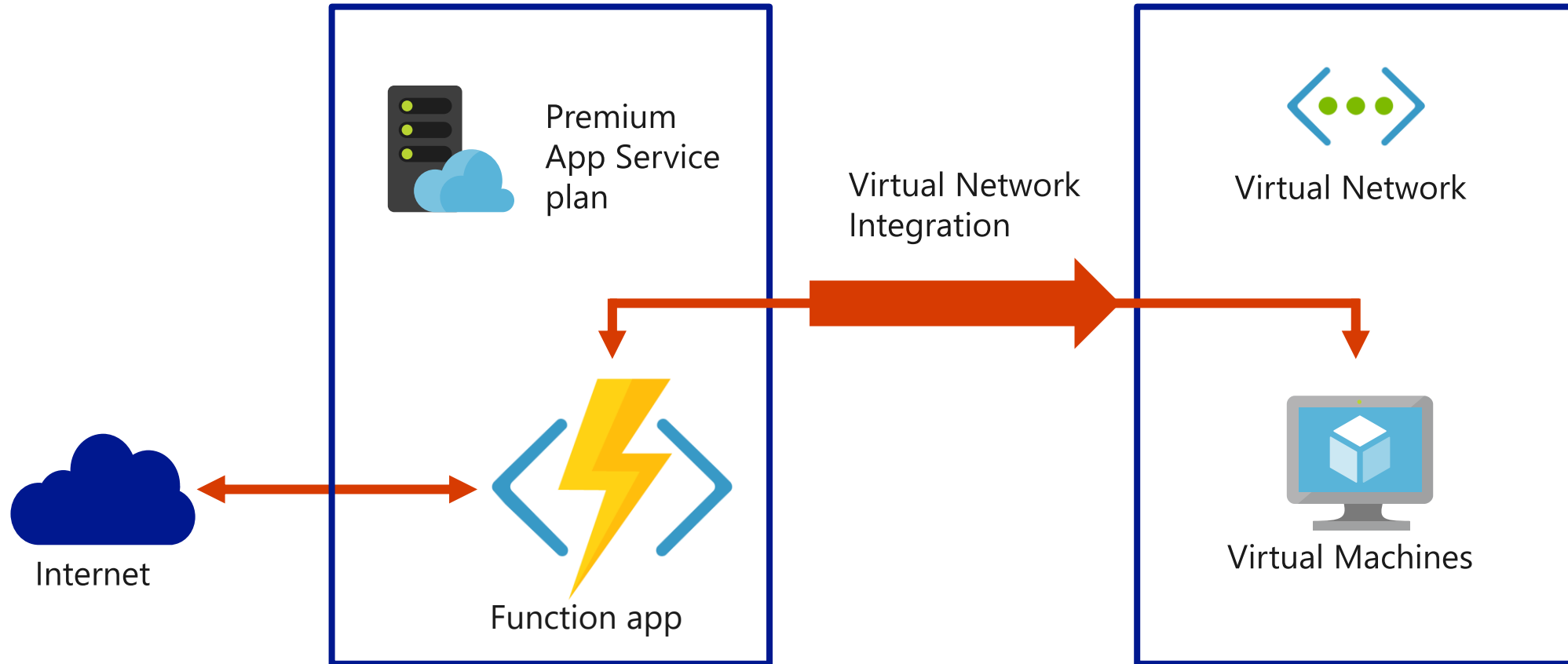


Integrating with Azure Virtual Network

- Requires the Premium plan
- Secures the inbound HTTP access to your app to one subnet in a Virtual Network
- Allows secure outbound calls to resources in a Virtual Network



Azure Virtual Network integration example



Best practices

- Avoid long-running functions:
 - Functions that run for a long time can time out
- Use queues for cross-function communication:
 - If you require direct communication, consider Durable Functions or Azure Logic Apps
- Write stateless functions:
 - Functions should be stateless and idempotent
 - State data should be associated with your input and output payloads
- Code defensively:
 - Assume that your function might need to continue from a previous fail point

Demo: Creating an Azure Functions project



Lesson 03: Durable Functions



Durable Functions

- Durable Functions enables you to implement complex stateful functions in a serverless-environment.
 - They enable you to write event driven code.
 - You can chain functions together
 - Manages state, checkpoints, and restarts
- Durable functions allows you to define stateful workflows using an orchestration function
 - Workflows are defined in code
 - Calls other functions synchronously or asynchronously
 - Checkpoint progress whenever function awaits

Durable Function types

- Installation

- `npm install durable-functions`

- Client functions

- Entry point for creating an instance of a Durable Functions orchestration.
 - Run in response to an event from many sources, such as a new HTTP request arriving, a message being posted to a message queue, ...

- Orchestrator functions

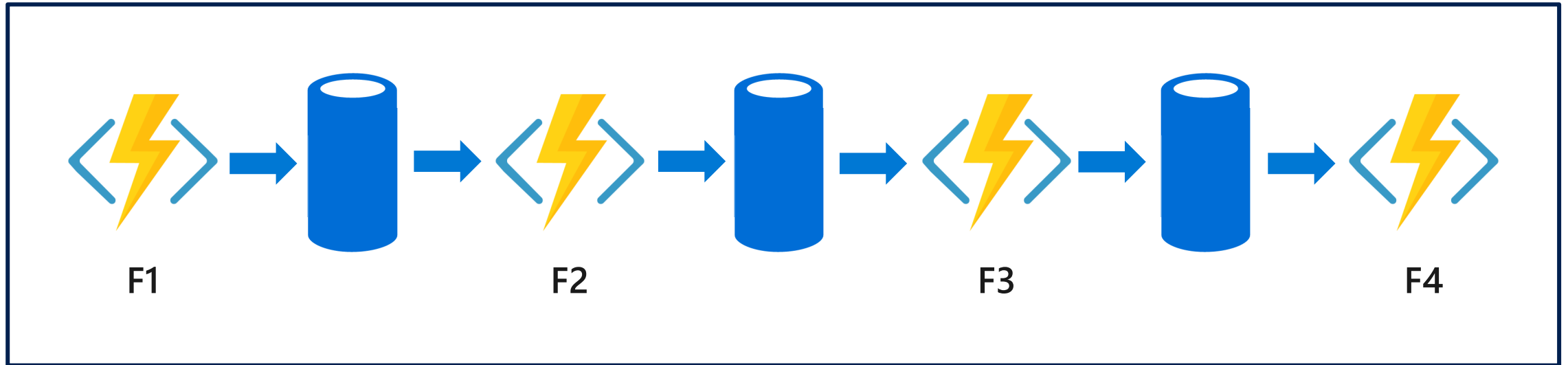
- Describe how actions are executed, and the order in which they are run.
 - You write the orchestration logic in code (C# or JavaScript).

- 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.

Durable Function scenario - Chaining

Function chaining refers executing a sequence of functions in a particular order. Often, the output of one function needs to be applied to the input of another function.



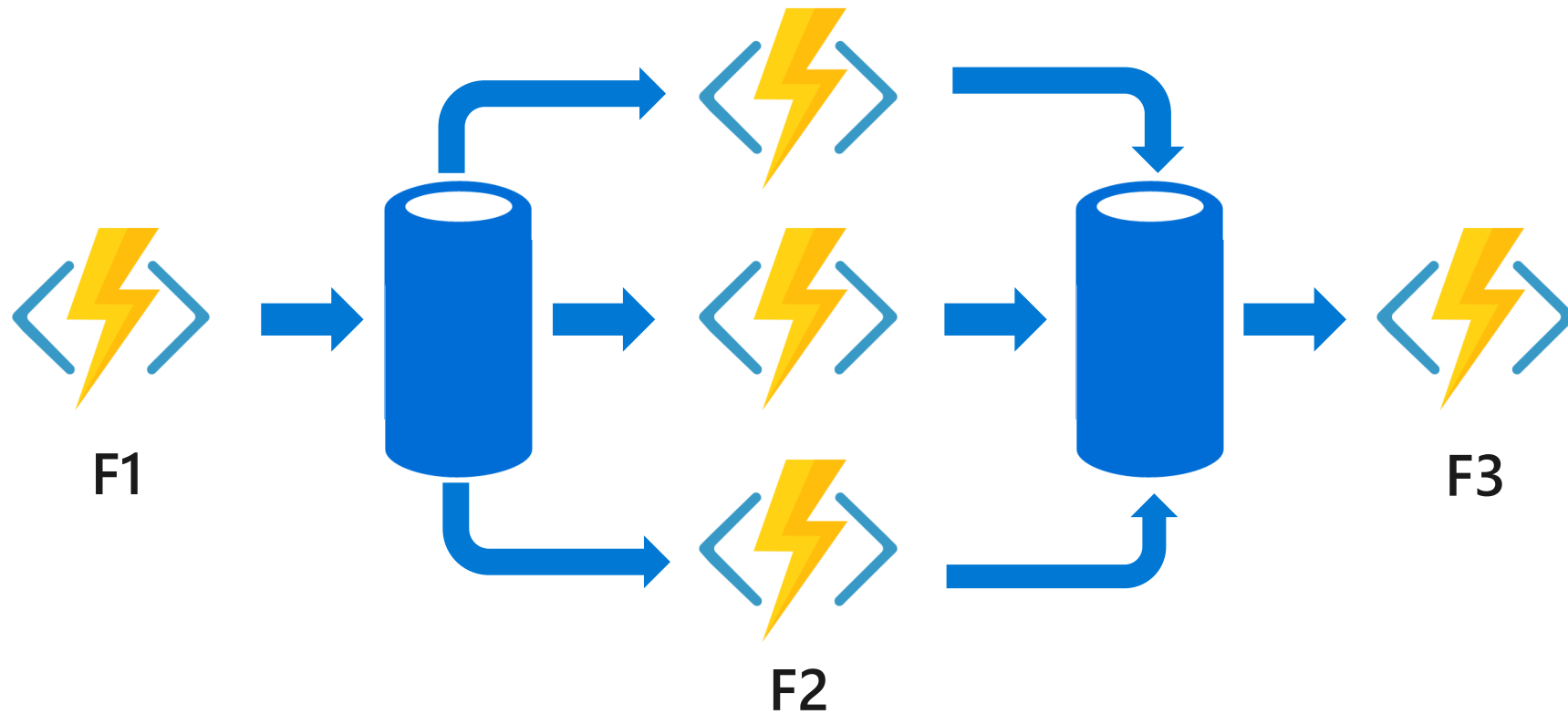
Durable Function scenario - Chaining code

```
public static async Task<object> Run(DurableOrchestrationContext ctx)
{
    try
    {
        var x = await ctx.CallActivityAsync<object>("F1");
        var y = await ctx.CallActivityAsync<object>("F2", x);
        var z = await ctx.CallActivityAsync<object>("F3", y);
        return await ctx.CallActivityAsync<object>("F4", z);
    }
    catch (Exception)
    {
        // error handling/compensation goes here
    }
}
```



Durable Function scenario - Fan-out/fan-in

Fan-out/fan-in refers to the pattern of executing multiple functions in parallel, and then waiting for all to finish

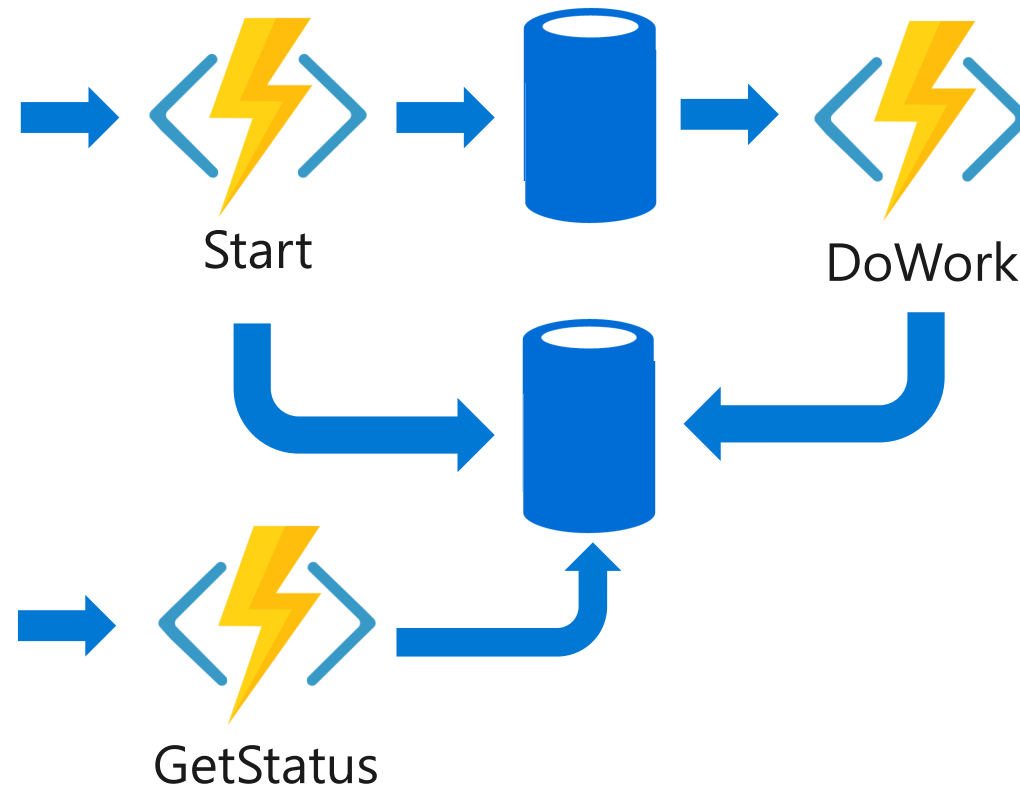


Durable Function scenario - Fan-out/fan-in code

```
public static async Task Run(DurableOrchestrationContext ctx)
{
    var parallelTasks = new List<Task<int>>();
    // get a list of N work items to process in parallel
    object[] workBatch = await ctx.CallActivityAsync<object[]>("F1");
    for (int i = 0; i < workBatch.Length; i++)
    {
        Task<int> task = ctx.CallActivityAsync<int>("F2", workBatch[i]);
        parallelTasks.Add(task);
    }
    await Task.WhenAll(parallelTasks);
    // aggregate all N outputs and send result to F3
    int sum = parallelTasks.Sum(t => t.Result);
    await ctx.CallActivityAsync("F3", sum);
}
```

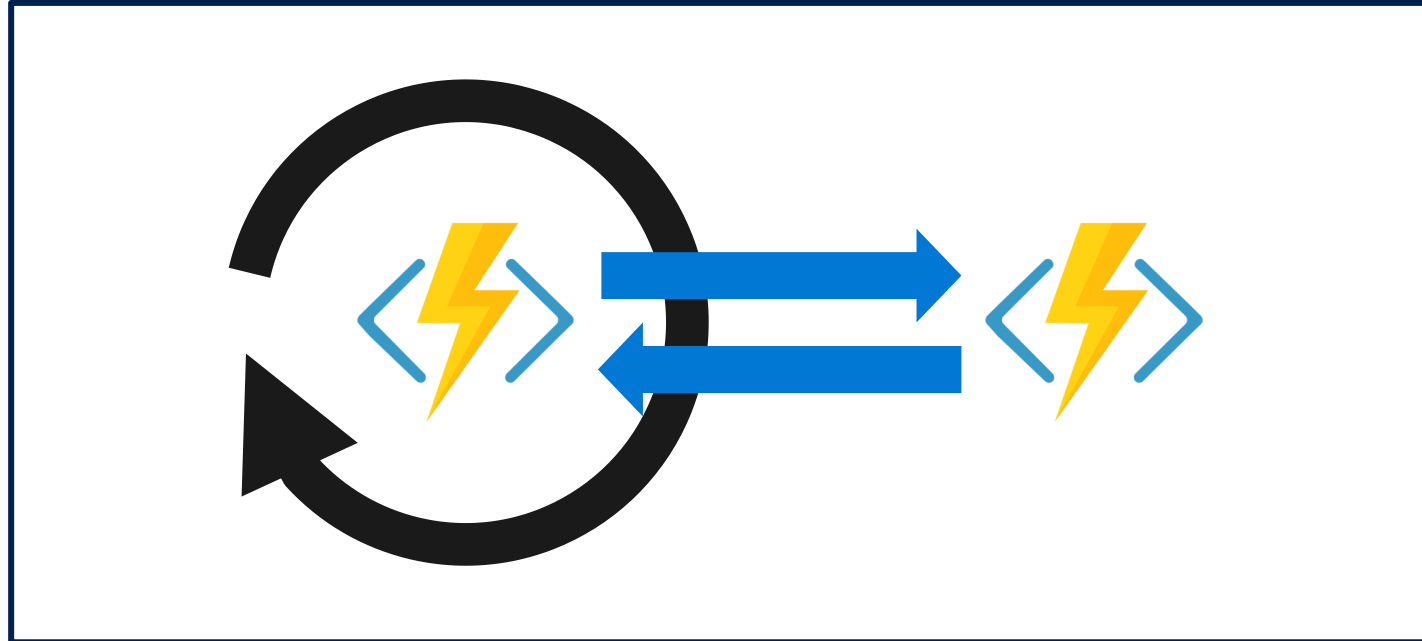
Durable Function scenario - Async HTTP APIs

Durable Functions provides built-in APIs that simplify the code that you write for interacting with long-running function executions



Durable Function scenario - Monitoring

The monitor pattern refers to a flexible recurring process in a workflow—for example, polling until certain conditions are met



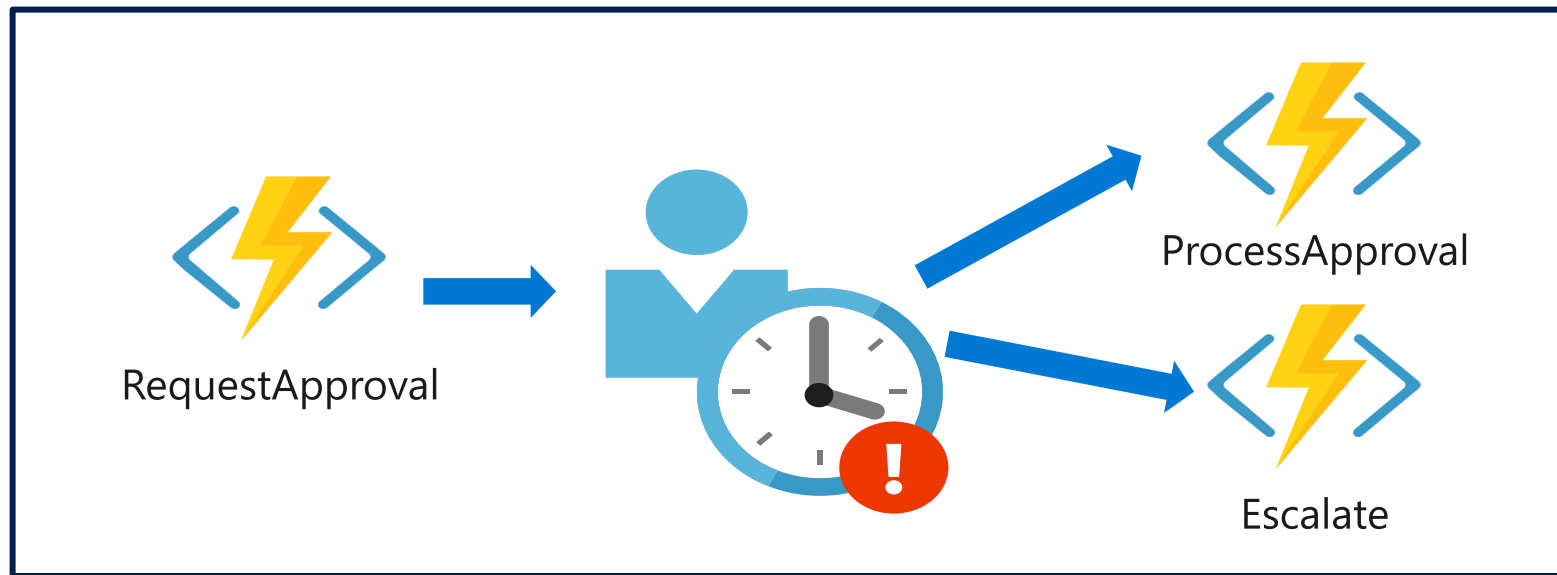
Durable Function scenario - Monitoring code

```
public static async Task Run(DurableOrchestrationContext ctx)
{
    int jobId = ctx.GetInput<int>(); int pollingInterval = GetPollingInterval();
    DateTime expiryTime = GetExpiryTime();
    while (ctx.CurrentUtcDateTime < expiryTime)
    {
        var jobStatus = await ctx.CallActivityAsync<string>("GetJobStatus", jobId);
        if (jobStatus == "Completed")
        { await ctx.CallActivityAsync("SendAlert", machineId); break; }
        // Orchestration will sleep until this time
        var nextCheck = ctx.CurrentUtcDateTime.AddSeconds(pollingInterval);
        await ctx.CreateTimer(nextCheck, CancellationToken.None);
    }
    // Perform further work here, or let the orchestration end
}
```



Durable Function scenario - Human interaction

Many processes involve human interaction. Automated processes must allow for human low availability, and they often do so by using time-outs and compensation logic.



Durable Function scenario - Human interaction code

```
public static async Task Run(DurableOrchestrationContext ctx)
{
    await ctx.CallActivityAsync("RequestApproval");
    using (var timeoutCts = new CancellationTokenSource())
    {
        DateTime dueTime = ctx.CurrentUtcDateTime.AddHours(72);
        Task durableTimeout = ctx.CreateTimer(dueTime, timeoutCts.Token);
        Task<bool> approval = ctx.WaitForExternalEvent<bool>("ApprovalEvent");
        if (approvalEvent == await Task.WhenAny(approvalEvent, durableTimeout))
        {
            timeoutCts.Cancel();
            await ctx.CallActivityAsync("ProcessApproval", approval.Result);
        } else
        {
            await ctx.CallActivityAsync("Escalate");
        }
    }
}
```



Durable Function scenario - Human interaction code (continued)

```
public static async Task Run(string instanceId, DurableOrchestrationClient client)
{
    bool isApproved = true;
    await client.RaiseEventAsync(instanceId, "ApprovalEvent", isApproved);
}
```

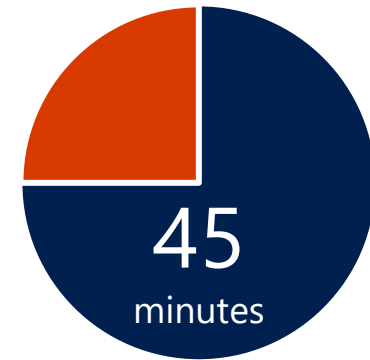


Durable Functions vs Logic Apps

	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

Lab: Implementing task processing logic by using Azure Functions

Duration



Lab sign-in information

