Microsoft

Module 02: Implement Azure Functions

Microsoft

Topics

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

Lesson 01: Azure Functions overview

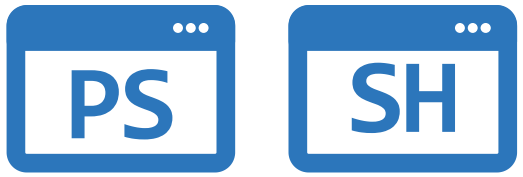
What can Azure Functions do?

- 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

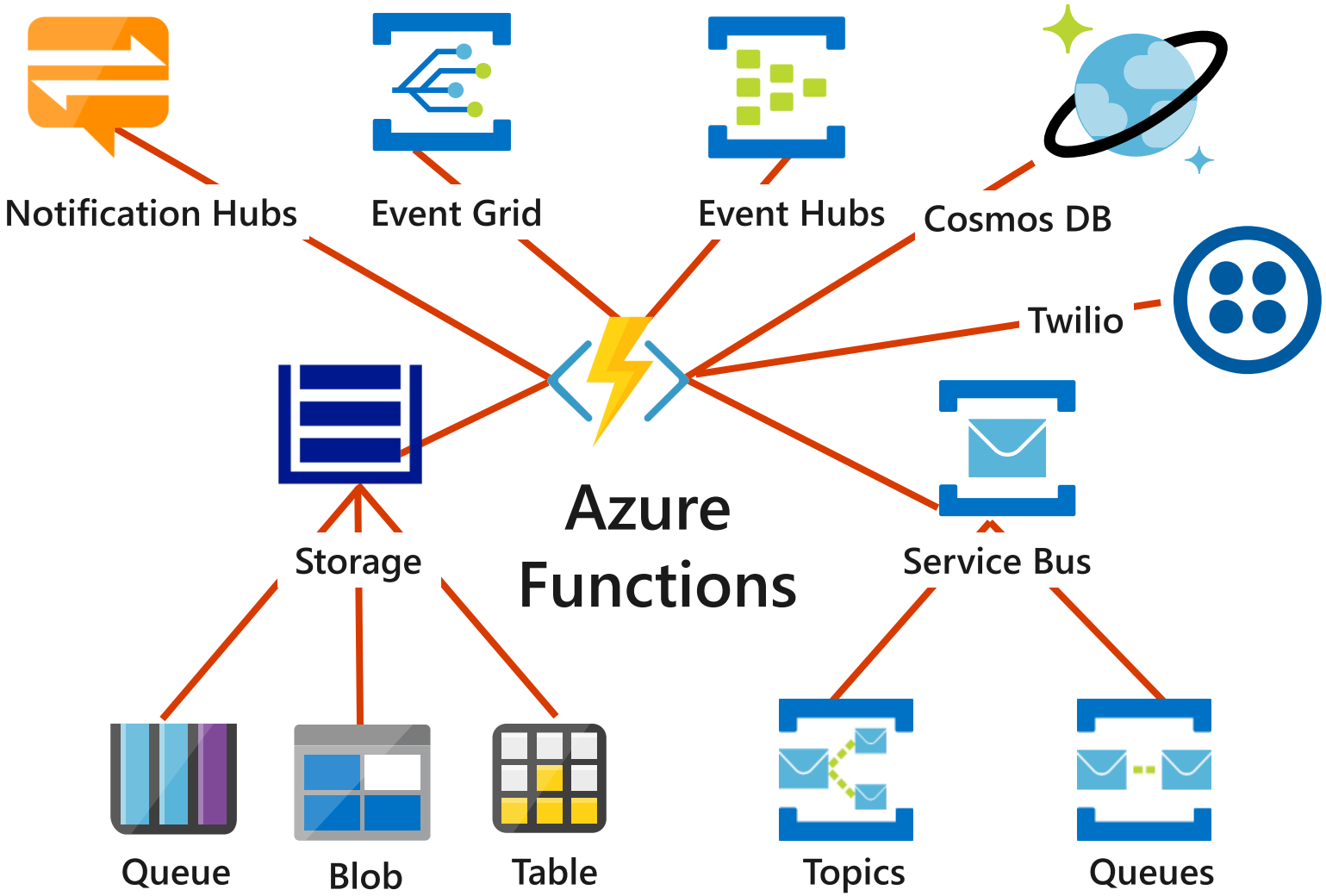
Azure Functions

- 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:

- Even supports scripting languages, such as:



Function integrations



Azure Function (Java program – Function.java)

```
public class Function {
    public String echo(
        @HttpTrigger(
            name = "request",
                methods = {"post"},
            authLevel = AuthorizationLevel.ANONYMOUS
        )
        String request, ExecutionContext context) {
        return String.format(request);
    }
}
```

Azure Function (Python script – __init__.py)

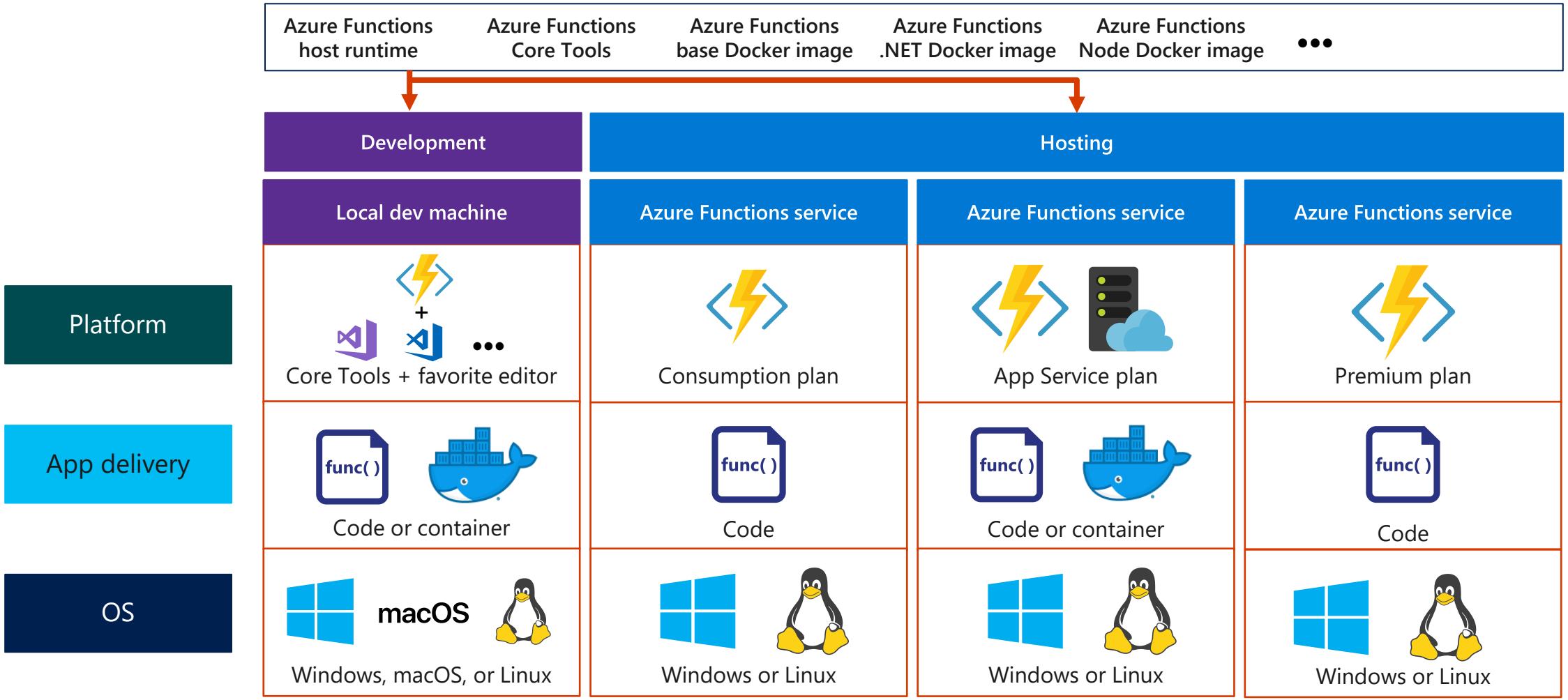
```python
import logging

import azure.functions as func

def main(myblob: func.InputStream):
    logging.info(f"Python blob trigger function processed\n"
        f"Name: {myblob.name}\n"
        f"Blob Size: {myblob.length} bytes")
```

Scale and hosting

- You can choose between three types of plans:
  - Consumption:
    - Instances are dynamically instanced and you are charged based on compute time
  - Premium
    - Instances of the Azure Functions host are added and removed based on the number of incoming events just like the Consumption plan, but provides additional features like: VNet connectivity; unlimited execution duration; and more predictable pricing
  - 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

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

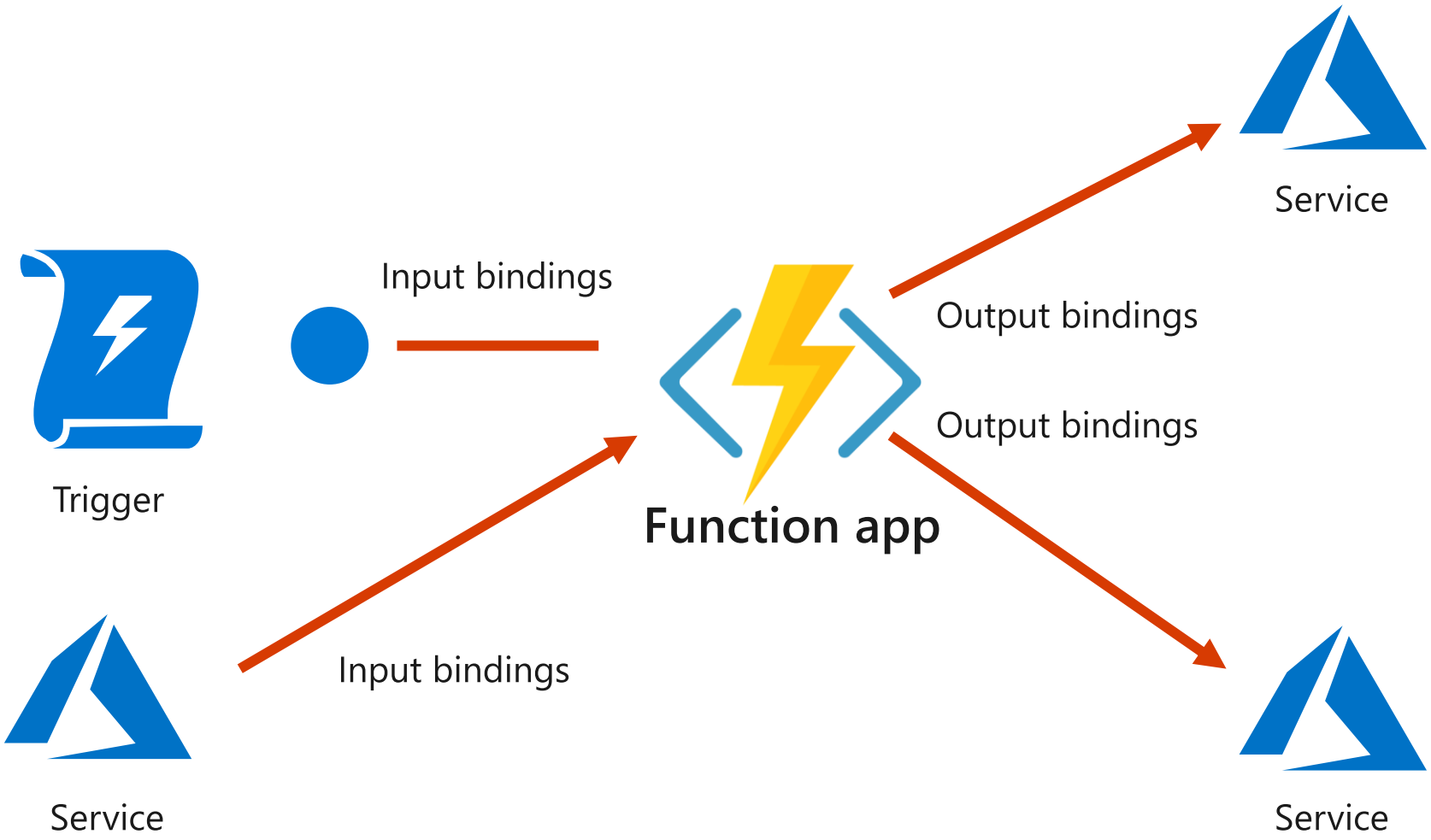
	IoT devices	Additional Azure hosts	Non-Azure hosts	On-premises
Platform	Azure IoT Edge	AKS, Service Fabric Mesh, ...	K8s, raw VMs, & more	App Service on Azure Stack Hub
App delivery	Container	Container	Container	Code
OS	Linux	Linux	Linux	Windows

Triggers

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...

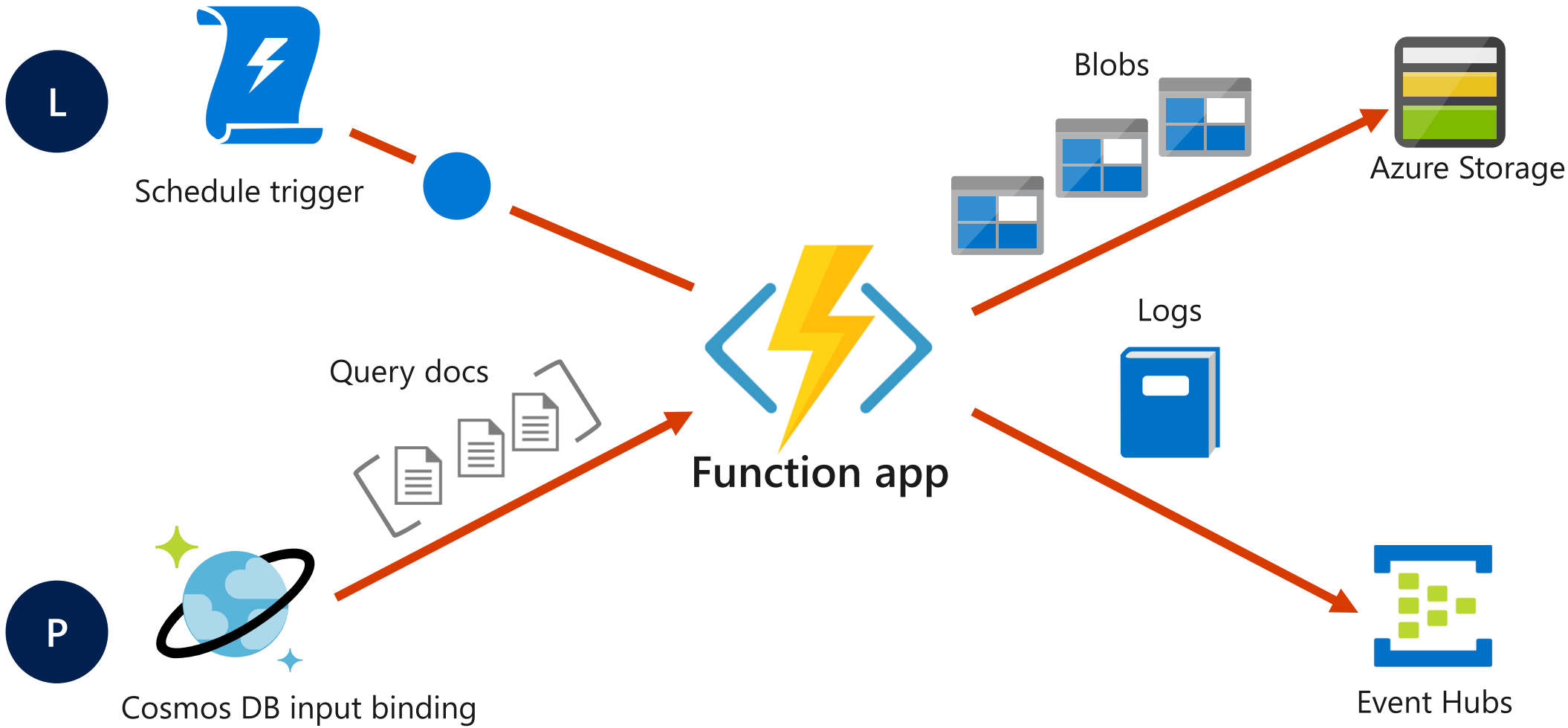
Input and Output Bindings



Bindings

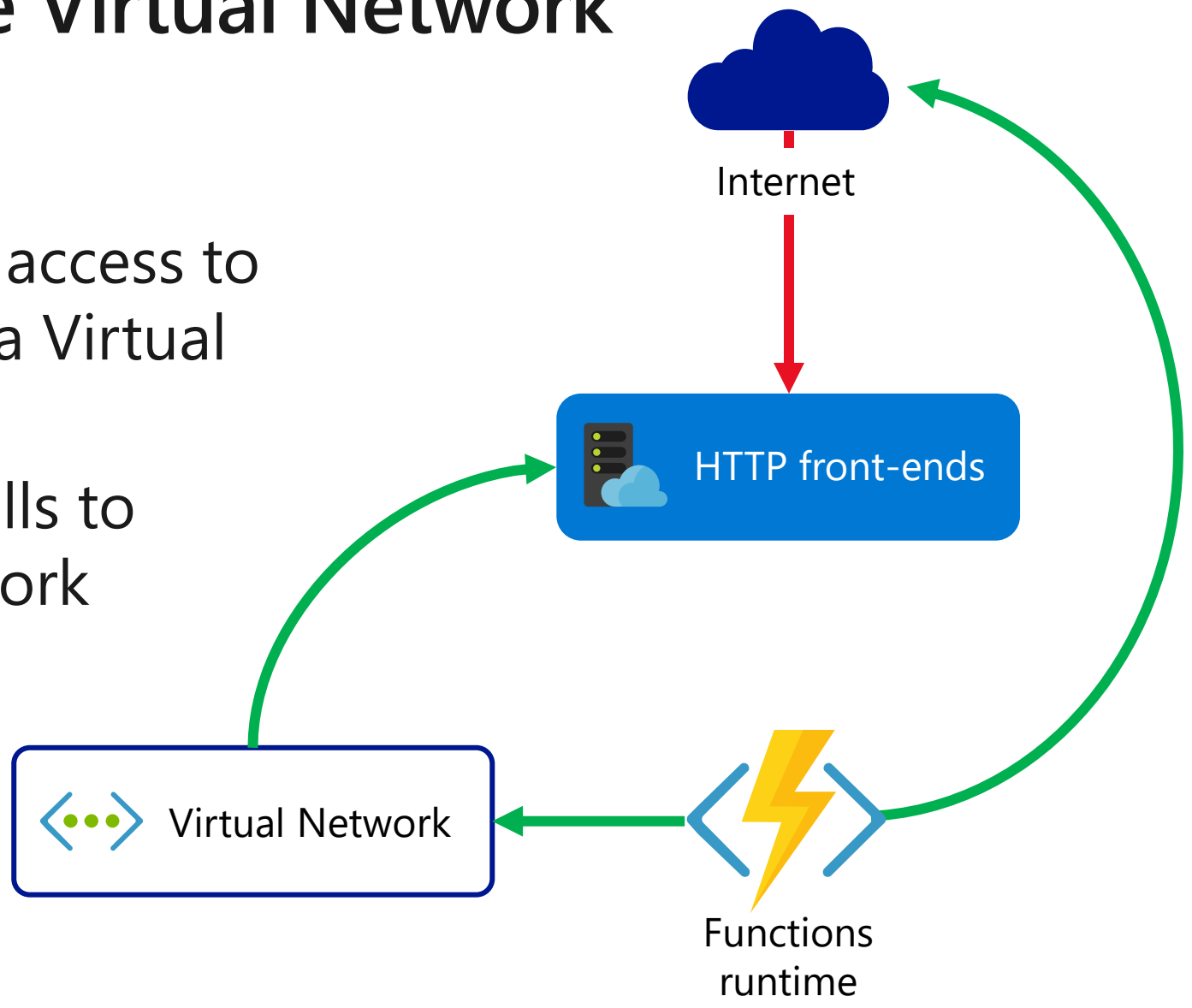
- Declarative way to connect to data from your code:
  - Connect to services without writing plumbing code
  - Service credentials are not stored in code
  - Bindings are optional
- Function can have multiple input and output bindings
- Output bindings can send data to Azure services such as:
  - Storage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

Trigger and Bindings example

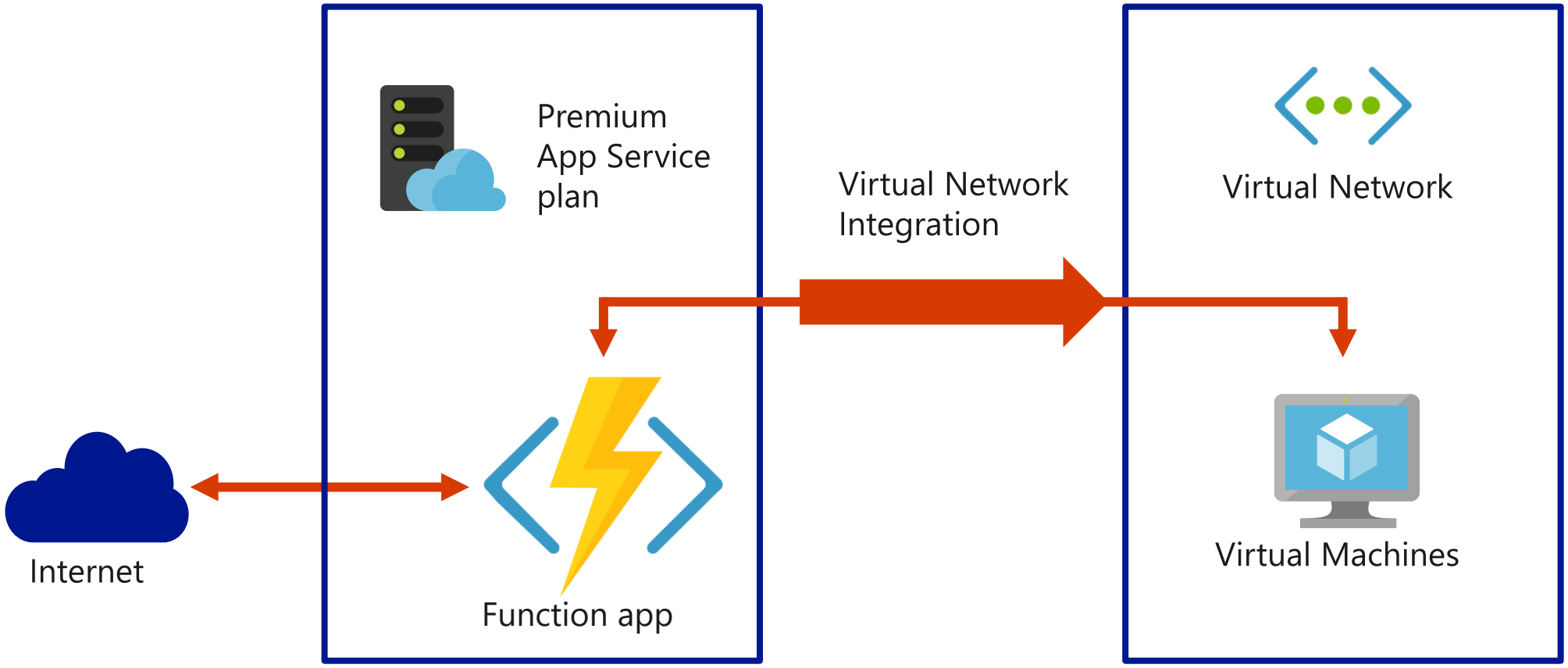


Integrating with Azure Virtual Network

- 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

Lesson 02: Developing Azure Functions

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

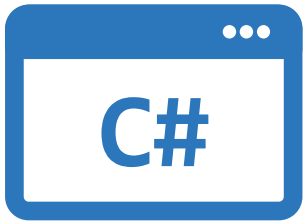
Azure Functions in Visual Studio

- 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

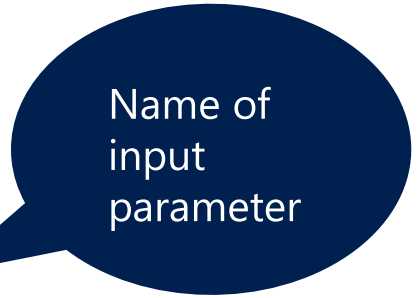
Demonstration: Creating an HTTP trigger function by using the Azure portal

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}");
        }
    }
}
```



Binding configuration

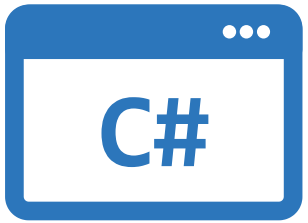



```
{
    "bindings": [
        {
            "name": "order",
            "type": "queueTrigger",
            "direction": "in",
            "queueName": "myqueue-items",
            "connection": "MY_STORAGE_ACCT_APP_SETTING"
        },
        {
            "name": "$return",
            "type": "table",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

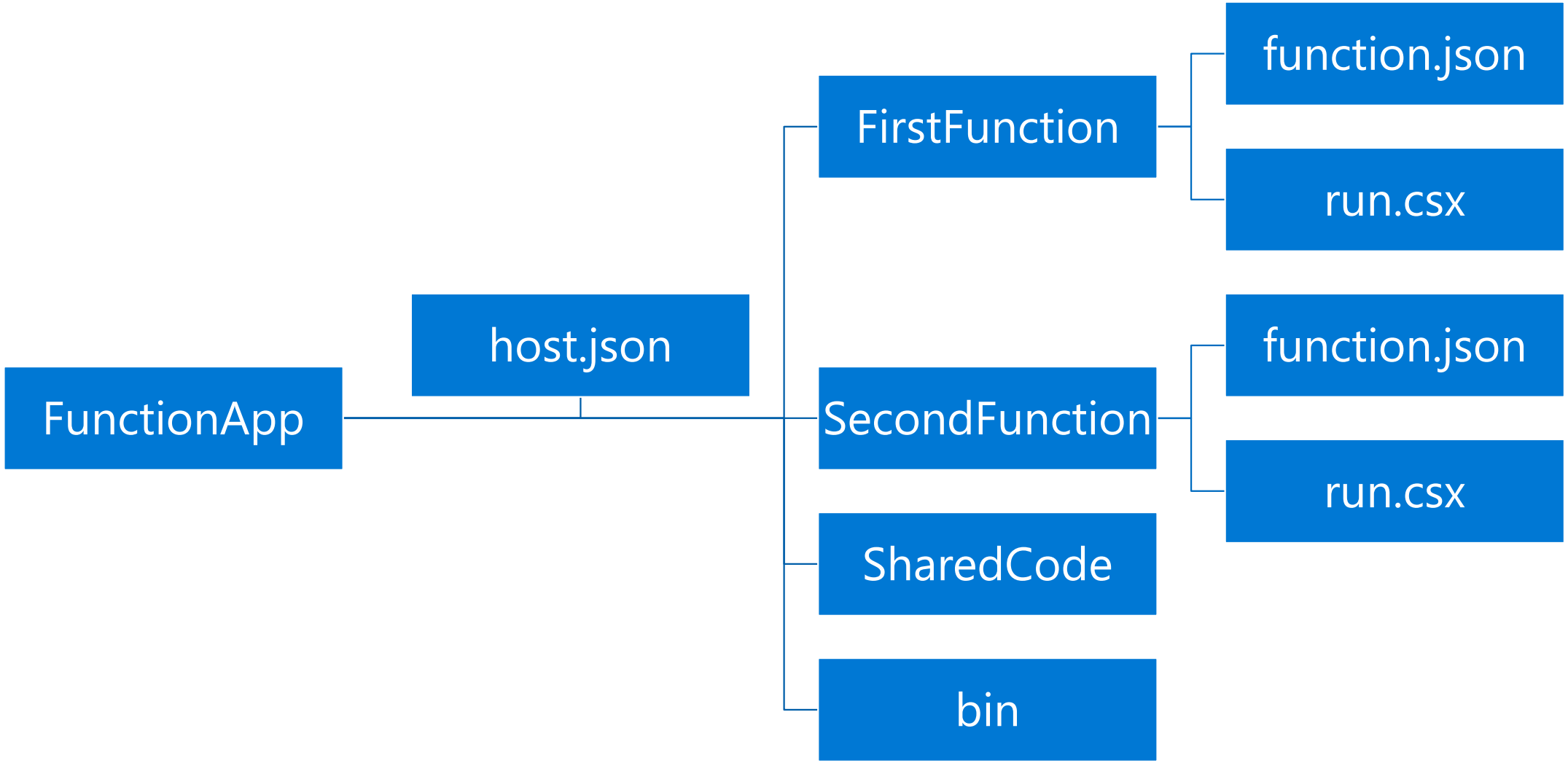
Binding-based code

```
#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()
    };
}
```

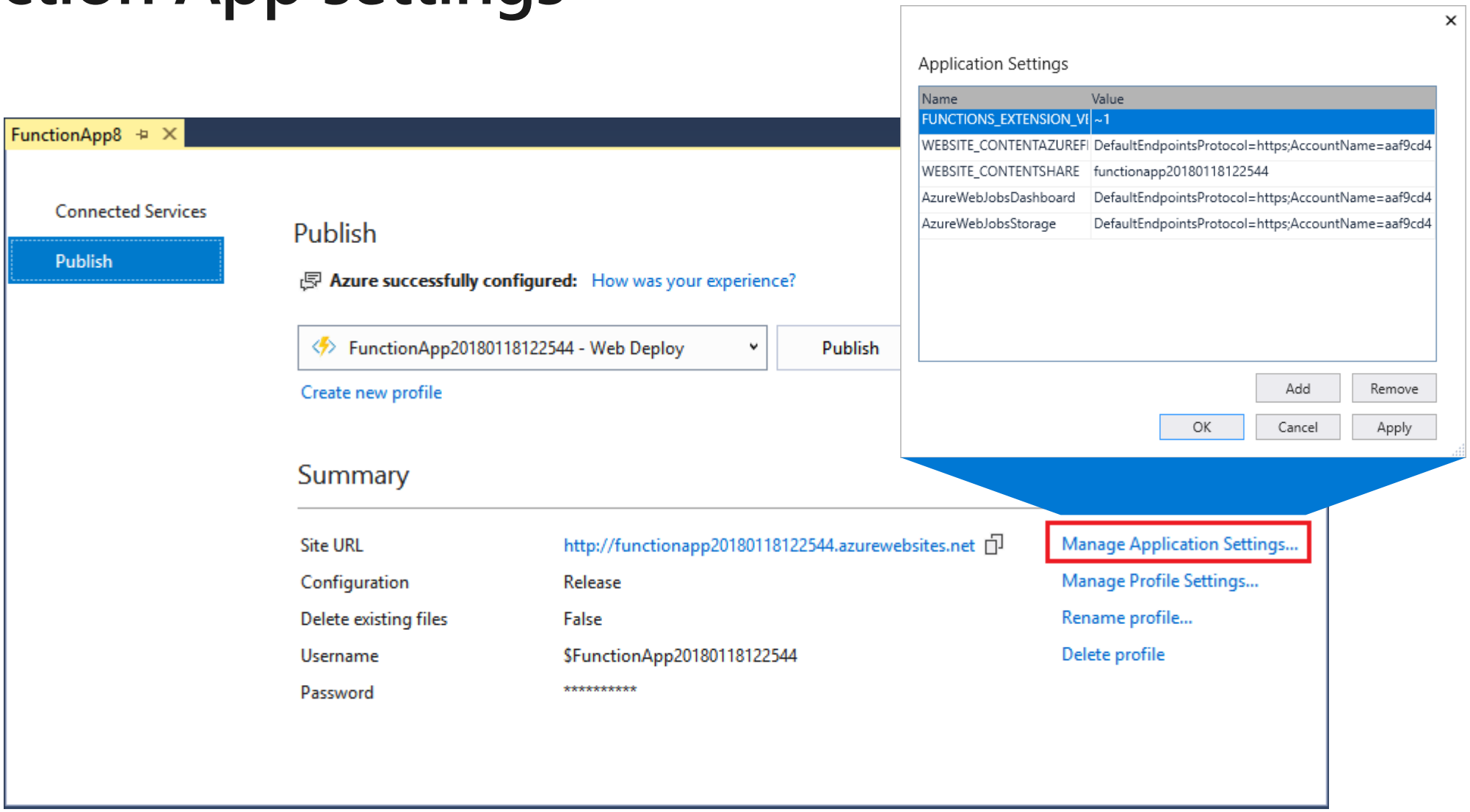
Name of input parameter



Function folder structure



Function App settings



Lesson 03: Implement Durable Functions

Durable Functions

- Write stateful functions in a stateless environment
- Manages state, checkpoints, and restarts
- Defines an Orchestrator function
  - Workflows are defined in code
  - Calls other functions synchronously or asynchronously
  - Checkpoint progress whenever function awaits

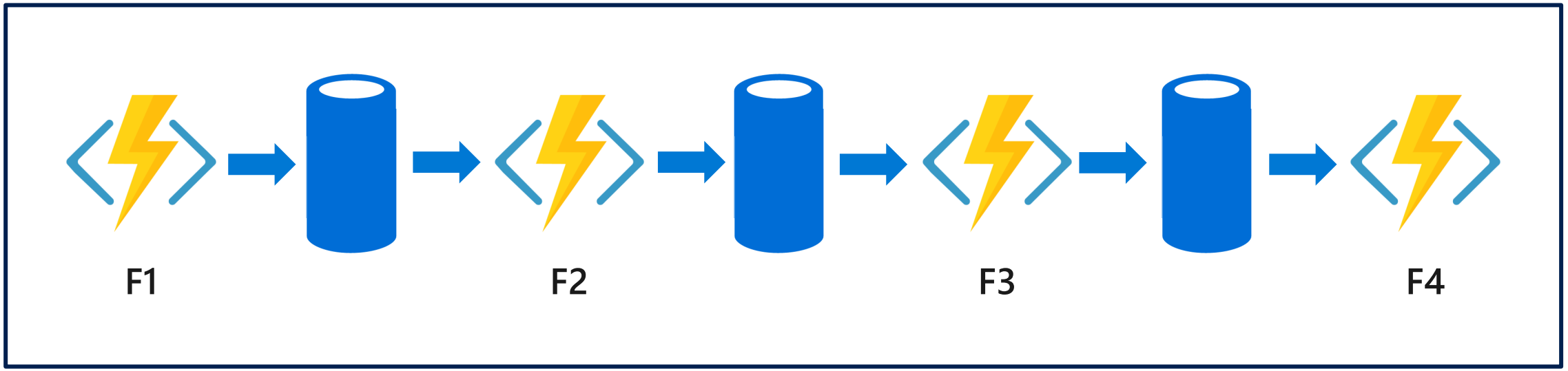
Durable Functions types

Orchestrator	Activity	Entity	Client
• Defines function workflows • Stateful	• The functions and tasks being orchestrated • Stateless	• Reads and updates small pieces of state • Stateful	• Sends messages to trigger Orchestrator and Entity functions • Stateless

- State is checkpointed and maintained in Azure Storage

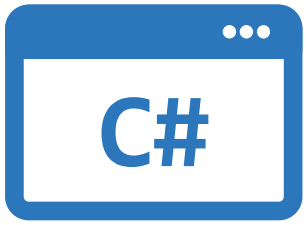
Durable Function scenario - Chaining

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.



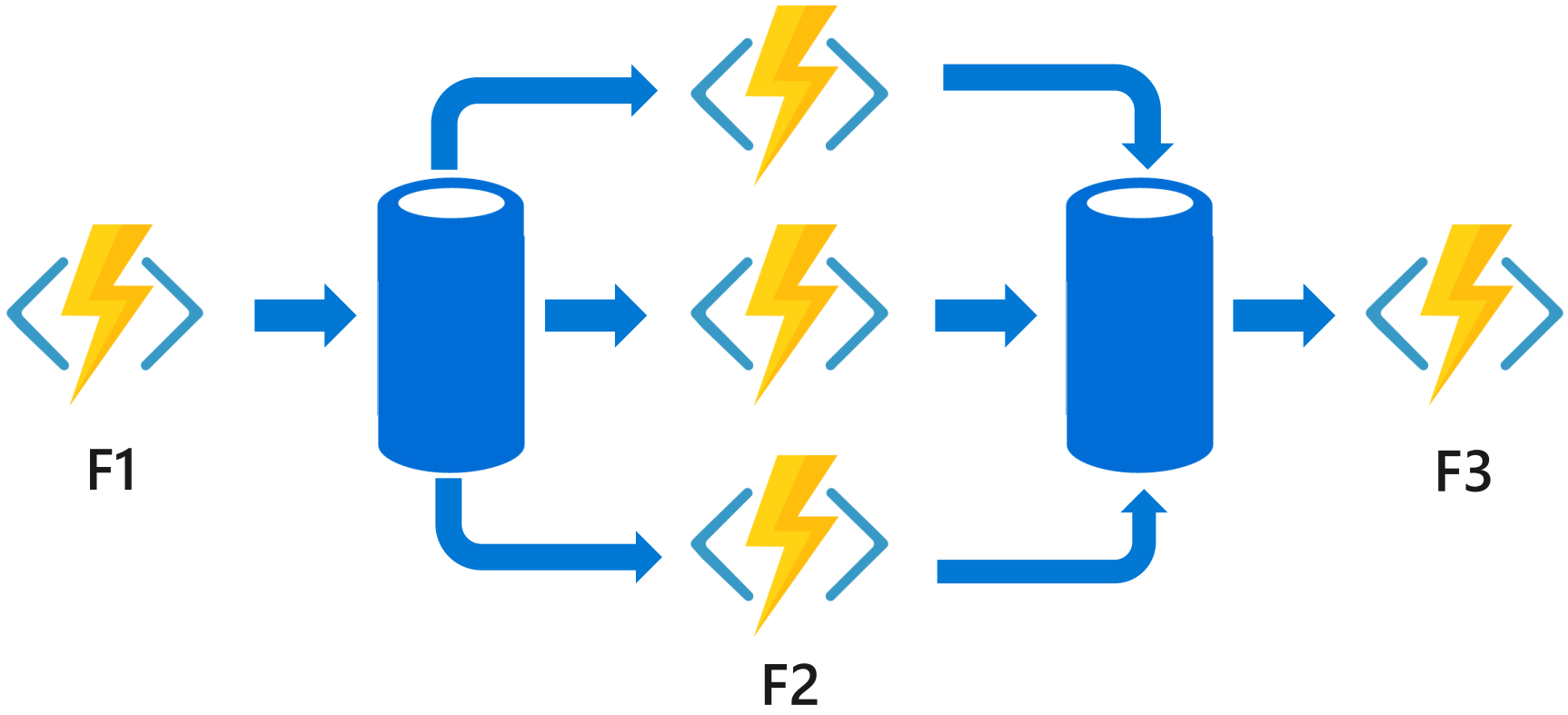
Durable Function scenario - Chaining code

```
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

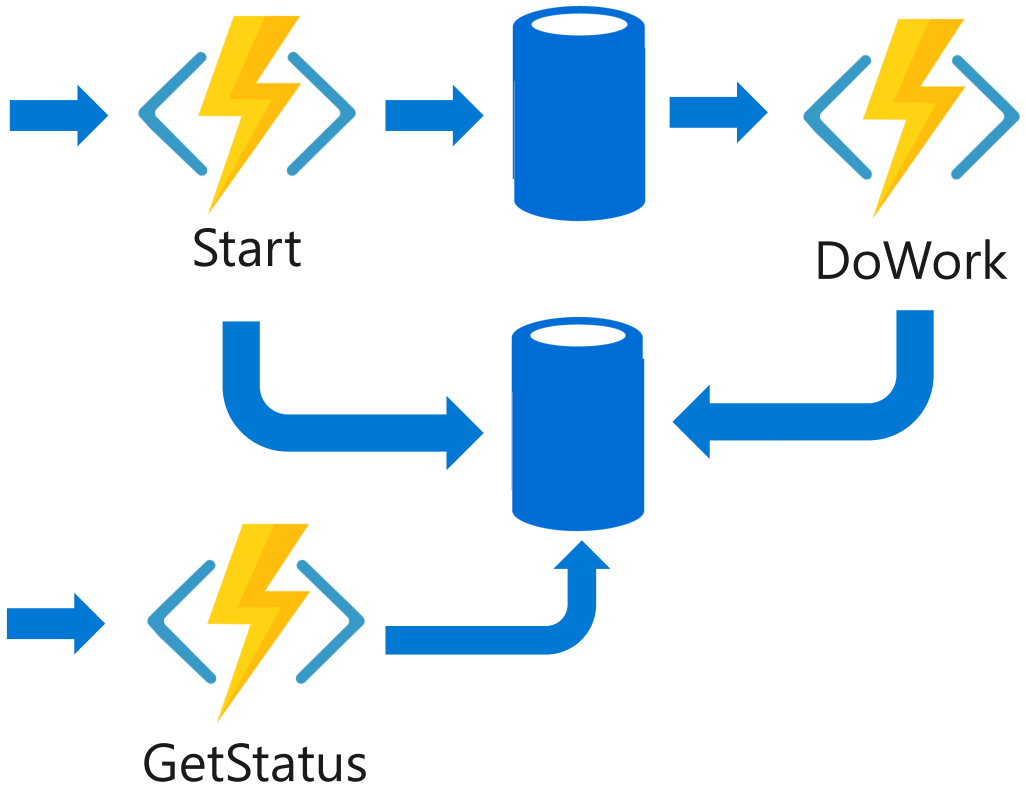


Durable Function scenario - Fan-out/fan-in code

```
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);
}
```

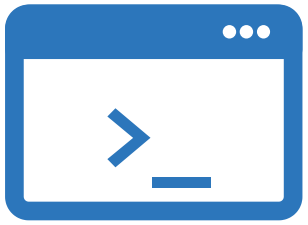
Durable Function scenario - Async HTTP APIs

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



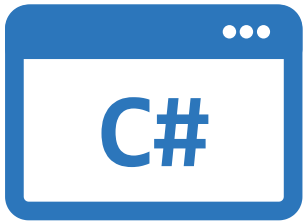
Durable Function scenario - Async HTTP APIs response

```
> curl -X POST https://myfunc.azurewebsites.net/orchestrators/DoWork -H "Content-Length: 0"
-i
HTTP/1.1 202 Accepted Content-Type: application/json
{ "id":"b79baf67f717453ca9e86c5da21e03ec", ... }

> curl
https://myfunc.azurewebsites.net/admin/extensions/DurableTaskExtension/b79baf67f717453ca9e86
c5da21e03ec -i
HTTP/1.1 202 Accepted Content-Type: application/json
{ "runtimeStatus":"Running", "lastUpdatedTime":"2017-03-16T21:20:47Z", ... }

> curl
https://myfunc.azurewebsites.net/admin/extensions/DurableTaskExtension/b79baf67f717453ca9e86
c5da21e03ec -i
HTTP/1.1 200 OK Content-Length: 175 Content-Type: application/json
{ "runtimeStatus":"Completed", "lastUpdatedTime":"2017-03-16T21:20:57Z", ... }
```



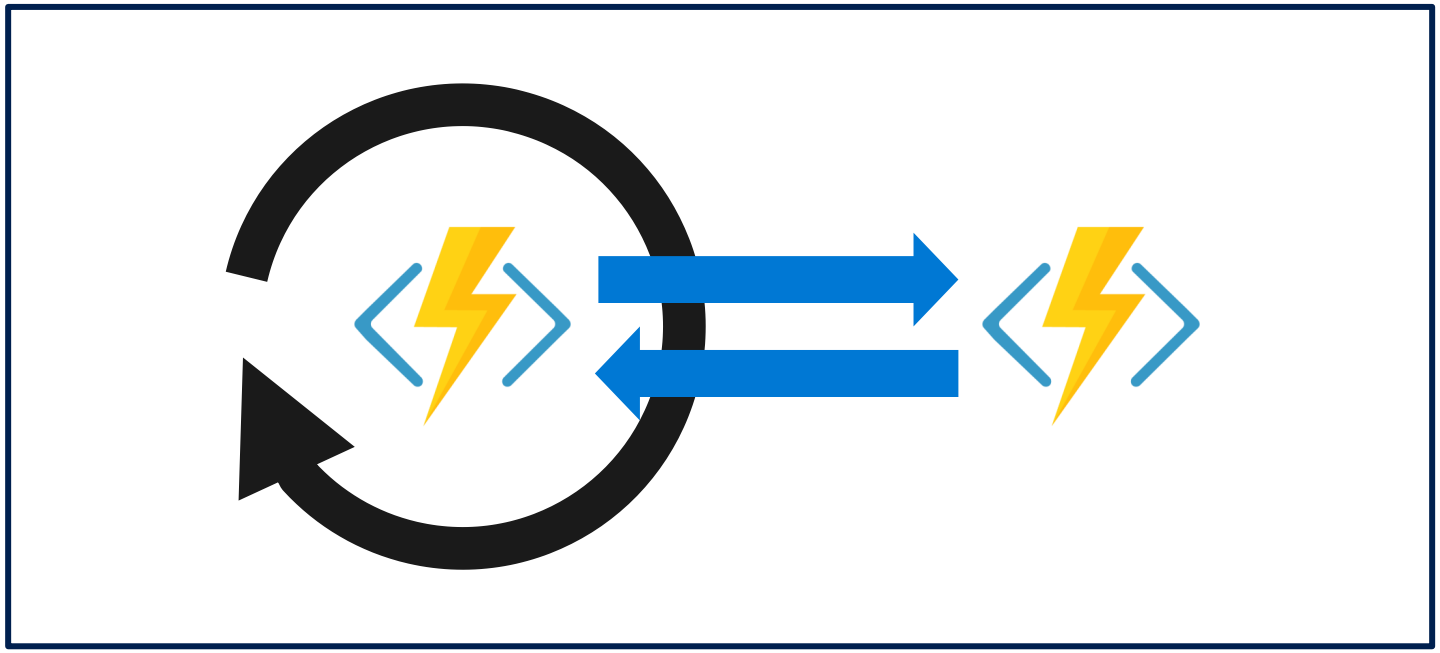
Durable Function scenario - Async HTTP APIs code

```
// HTTP-triggered function to start a new orchestrator function instance.
public static async Task<HttpResponseMessage> Run(
    HttpRequestMessage req,
    DurableOrchestrationClient starter,
    string functionName,
    ILogger log)
{
    // Function name comes from the request URL.
    // Function input comes from the request content.
    dynamic eventData = await req.Content.ReadAsAsync<object>();
    string instanceId = await starter.StartNewAsync(functionName, eventData);

    log.LogInformation($"Started orchestration with ID = '{instanceId}'.");

    return starter.CreateCheckStatusResponse(req, instanceId);
}
```



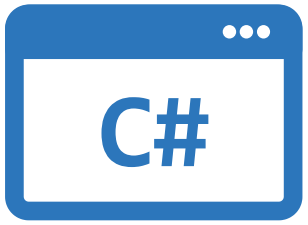
Durable Function scenario - Monitoring

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



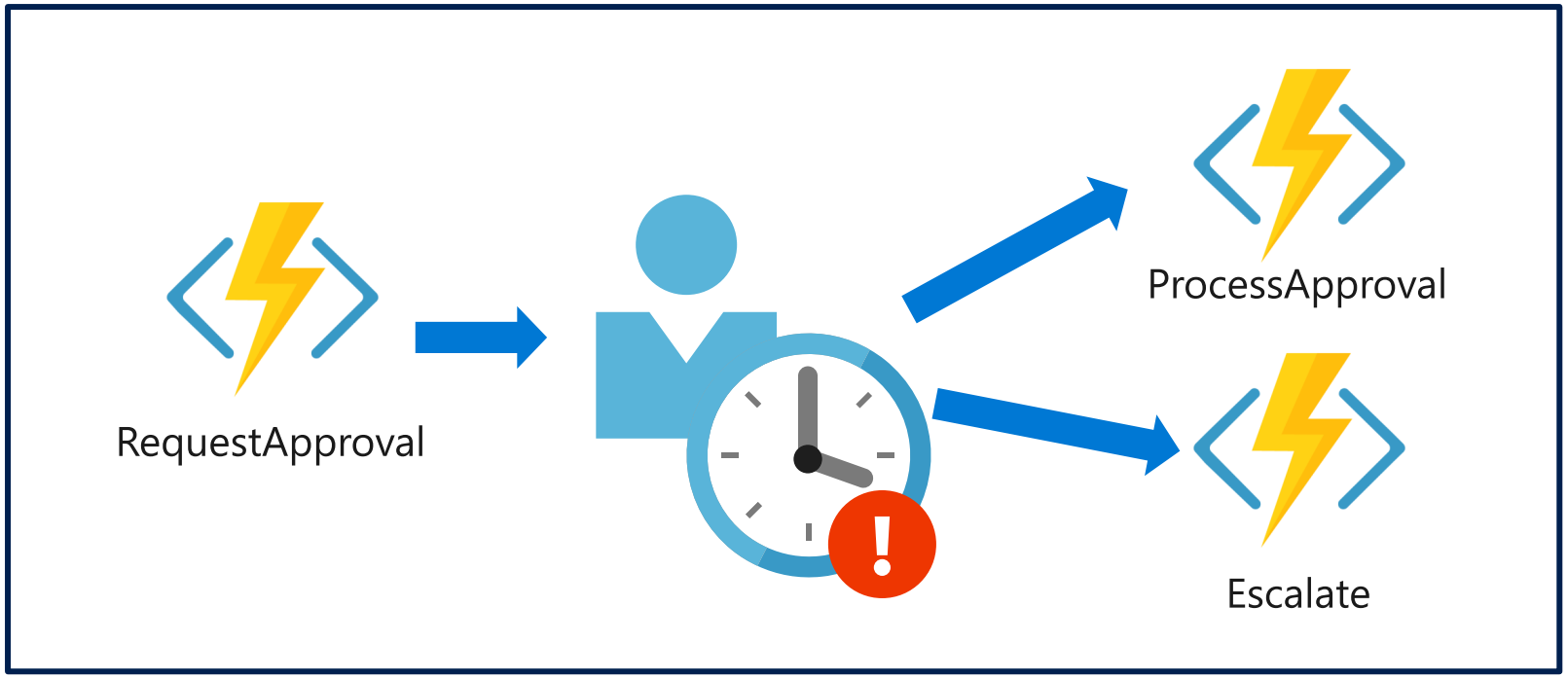
Durable Function scenario - Monitoring code

```
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
}
```



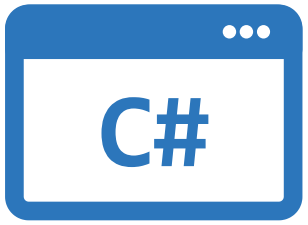
Durable Function scenario - Human interaction

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.



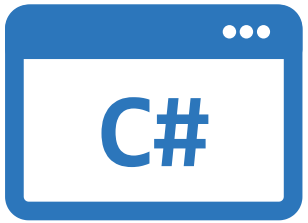
Durable Function scenario - Human interaction code

```
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)

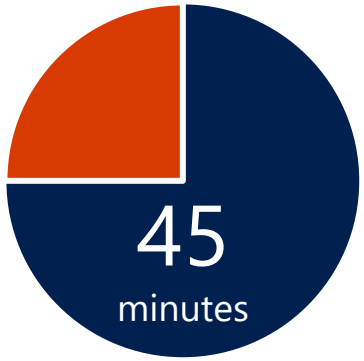
```
public static async Task Run(string instanceId, DurableOrchestrationClient client)
{
    bool isApproved = true;
    await client.RaiseEventAsync(instanceId, "ApprovalEvent", isApproved);
}
```



Lab: Implementing task processing logic by using Azure Functions

Lab: Implementing task processing logic by using Azure Functions

Duration



Lab sign-in information