Azure Durable ⚡ Functions

Vaibhav Gujral @ Capgemini
Microsoft Azure MVP
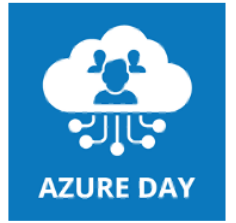
1

Thanks to

2

About Me

- Director, Microsoft Cloud CoE at Capgemini
- Born and brought up in India and based out of Omaha, NE
- Microsoft Azure MVP
- Leader, Omaha Azure User Group(https://omahaazure.org)
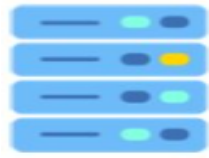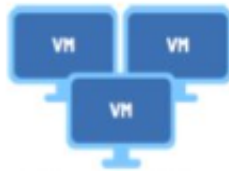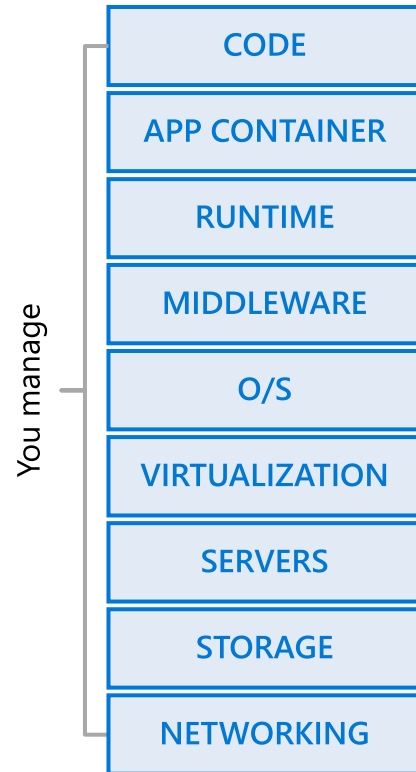- 15+ cloud certifications and counting…

Agenda

- What is Serverless?
- What are Azure Functions?
- Challenges with Azure Functions
- What are Durable Functions?
- Different Patterns
- Demos
- Q&A

https://vaibhavgujral.com/

Service Models

Bare Metal

You manage
- CODE
- APP CONTAINER
- RUNTIME
- MIDDLEWARE
- O/S
- VIRTUALIZATION
- SERVERS
- STORAGE
- NETWORKING

Virtual Machines

You manage
- CODE
- APP CONTAINER
- RUNTIME
- MIDDLEWARE
- O/S

Managed by vendor
- VIRTUALIZATION
- SERVERS
- STORAGE
- NETWORKING

App Services

You manage
- CODE
- APP CONTAINER

Managed by vendor
- RUNTIME
- MIDDLEWARE
- O/S
- VIRTUALIZATION
- SERVERS
- STORAGE
- NETWORKING

Serverless Functions

You manage
- CODE

Managed by vendor
- APP CONTAINER
- RUNTIME
- MIDDLEWARE
- O/S
- VIRTUALIZATION
- SERVERS
- STORAGE
- NETWORKING

https://vaibhavgujral.com/

- Doesn't mean No-Server

- Rather, think of it as Less-Server

- Abstraction of servers, infrastructure, and operating systems

- Function-as-a-Service (FaaS)
  - Execute independent code pieces as functions
  - Examples-
    - Azure Functions
    - AWS Lambda
    - Google Cloud Functions

- Serverless compute service that can run code on-demand

- Run small pieces of code in Azure (call them as "functions")

- Provides out of the box templates for some of the most common scenarios

- Useful in common scenarios like –
  - Connecting to Storage
  - Image processing
  - Exposing HTTP based APIs
  - IoT
  - Running a script or code in response to a variety of events etc

- Azure Functions is a serverless evolution of Azure WebJobs

Azure Functions Features

- Choice of Language –
  - Supports C#, F#, JavaScript, Java, Powershell, Python, Typescript
- Pay per use (only for the time the code is executed)
- Nuget and NPM support
- Integrated Security – Oauth support for Http-triggered functions
- Seemless integration with other Azure Services
- Flexible Development
- Azure Functions runtime is open-source

Language	1.x	2.x	3.x	4.x
C#	GA (.NET Framework 4.8)	GA (.NET Core 2.1[1])	GA (.NET Core 3.1) GA (.NET 5.0)	GA (.NET 6.0) Preview (.NET 7) Preview (.NET Framework 4.8)
JavaScript	GA (Node.js 6)	GA (Node.js 10 & 8)	GA (Node.js 14, 12, & 10)	GA (Node.js 14) GA (Node.js 16) Preview (Node.js 18)
F#	GA (.NET Framework 4.8)	GA (.NET Core 2.1[1])	GA (.NET Core 3.1)	GA (.NET 6.0)
Java	N/A	GA (Java 8)	GA (Java 11 & 8)	GA (Java 11 & 8) Preview (Java 17)
PowerShell	N/A	GA (PowerShell Core 6)	GA (PowerShell 7.0 & Core 6)	GA (PowerShell 7.0, 7.2)
Python	N/A	GA (Python 3.7 & 3.6)	GA (Python 3.9, 3.8, 3.7, & 3.6)	GA (Python 3.9, 3.8, 3.7)
TypeScript[2]	N/A	GA	GA	GA

https://docs.microsoft.com/en-us/azure/azure-functions/supported-languages

https://vaibhavgujral.com/

Azure Functions Integrations

- Functions can be integrated with various Azure and 3rd party services

- These services can either trigger the function execution or serve as input/output for function code

- Azure Functions can be integrated with these services:

Cosmos DB	Event Hub	Event Grid	Service Bus	Storage Account	Notification Hub
IoT Hub	Twilio	SendGrid	Kafka	dapr	Webhooks

AZURE DAY

An event triggers a function

Function execution

Output is returned

https://vaibhavgujral.com/
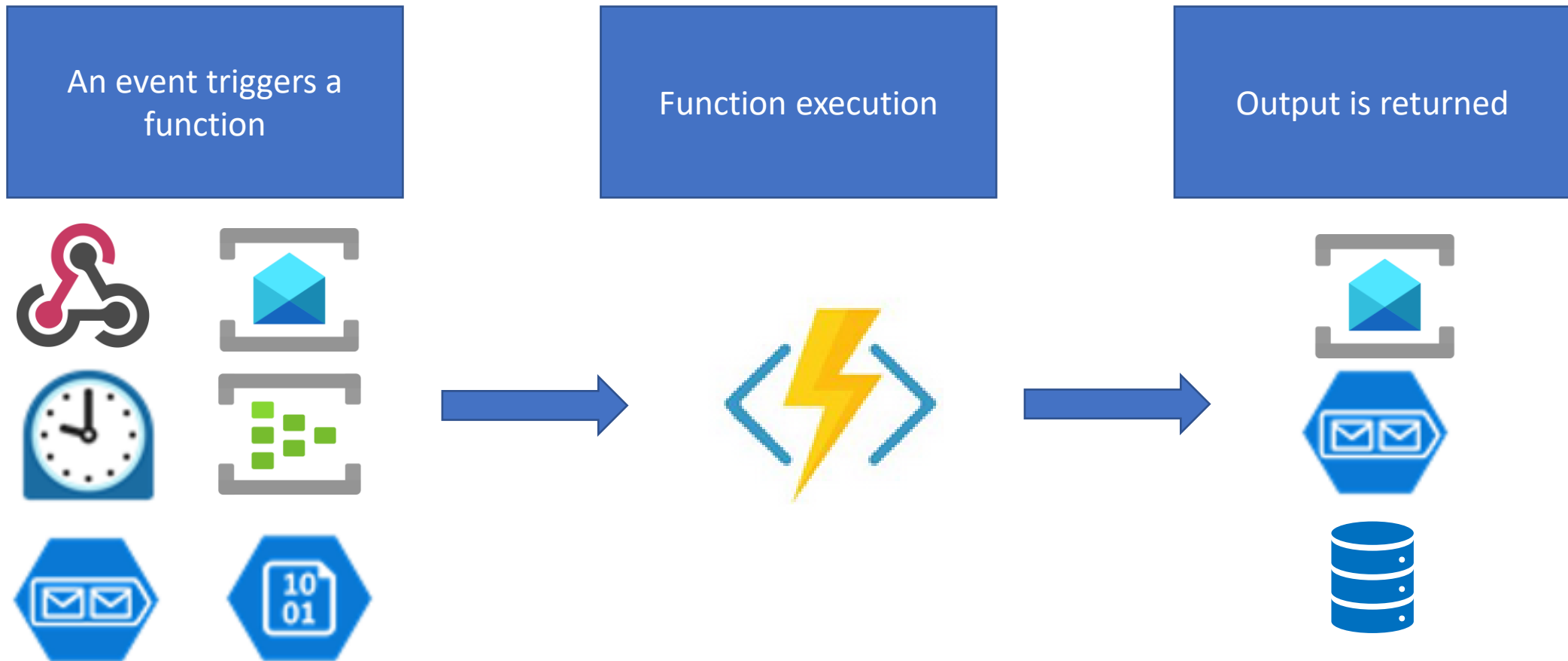
- One of the Merriam-Webster's definition says –
  - "A Trigger is something that acts like a mechanical trigger in initiating a process or reaction"
- Defines how a function is invoked
- Out-of-the-box templates to trigger execution of an Azure function
- A function can have exactly one trigger
- A trigger can have an associated data, which is usually the payload that triggered the function
- Binding direction for triggers in always in

https://vaibhavgujral.com/

- HTTPTrigger

- TimerTrigger

- CosmosDBTrigger

- BlobTrigger

- QueueTrigger

- EventGridTrigger

- EventHubTrigger

- ServiceBusQueueTrigger

- ServiceBusTopicTrigger

- Declarative way to make data from external services available to function code

- Bindings are optional

- Two types of bindings
  - Input
  - Output

- A function can have multiple input and output bindings

- Input and output bindings use in and out binding directions

- Some bindings support special binding direction – inout

- Defined in function.json file.

```json
{
  "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"
    }
  ]
}
```

https://vaibhavgujral.com/

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

public class Person
{
    public string PartitionKey { get; set; }
    public string RowKey { get; set; }
    public string Name { get; set; }
    public string MobileNumber { get; set; }
}
```
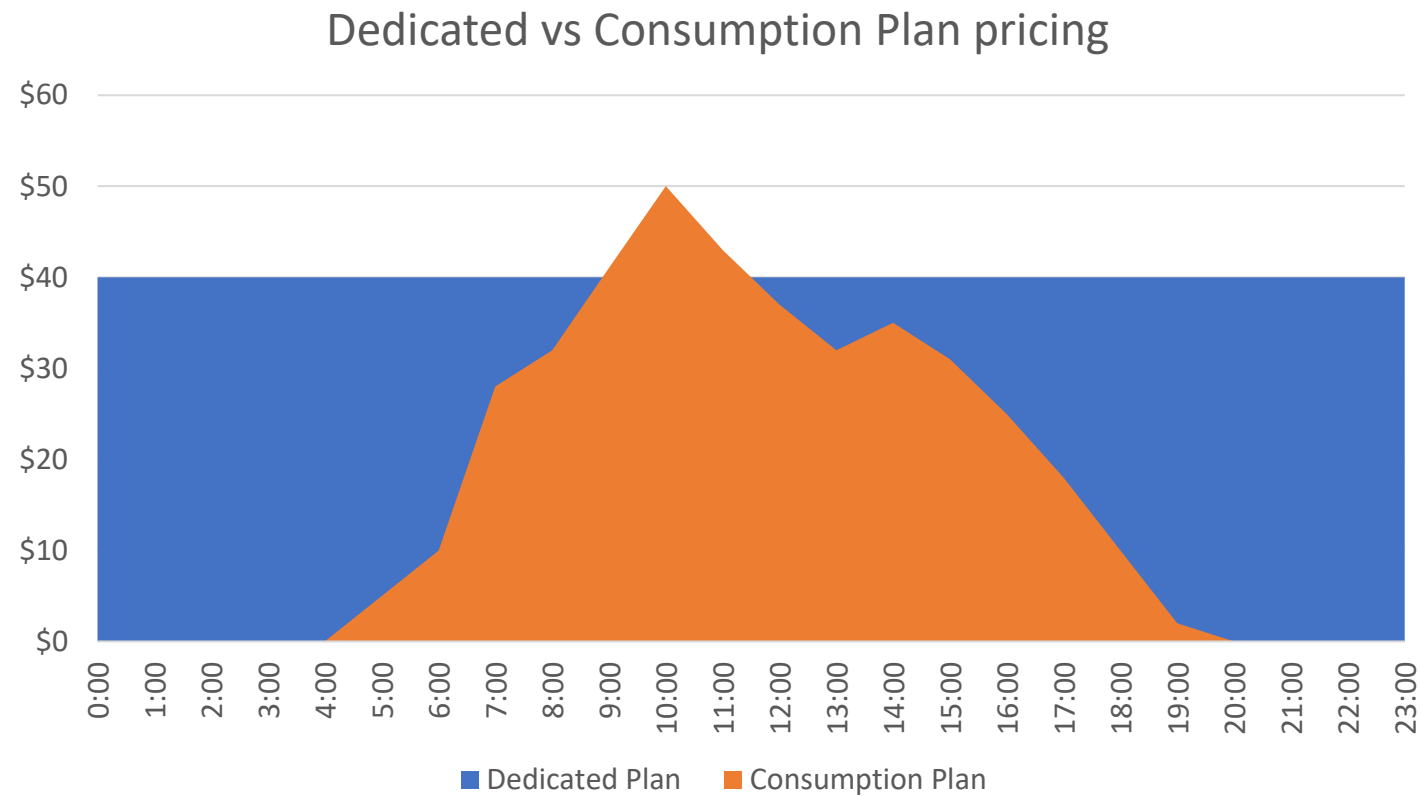
https://vaibhavgujral.com/

Three pricing plans:

- Consumption Plan
  - Pay for the time the code is executed
  - Default Hosting plan

- Premium Plan
  - Similar to consumption plan
  - Perpetually warm instances to avoid any cold start
  - VNET connectivity

- Dedicated Plan
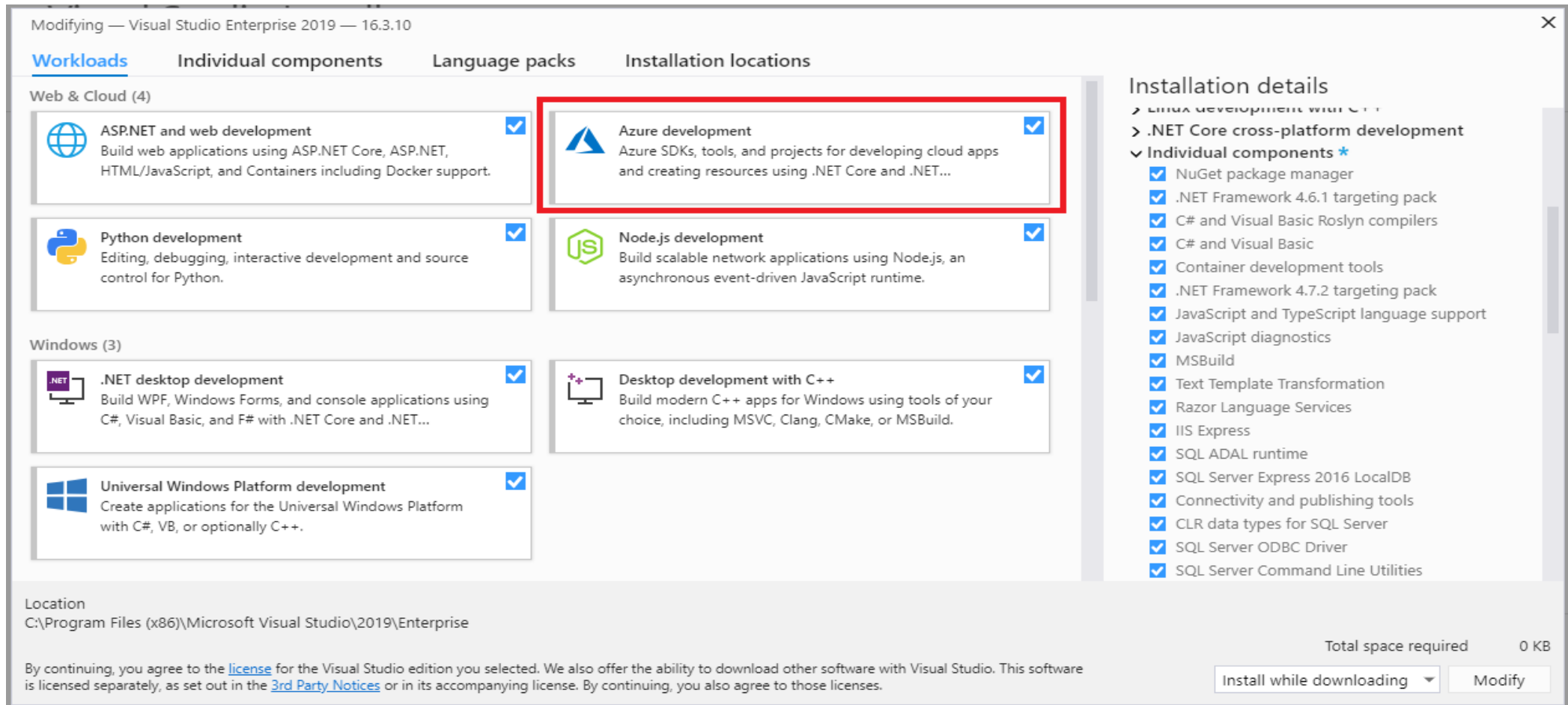 - Same as an app service plan for a web app
  - Enable always on

Azure Functions Pricing

Dedicated vs Consumption Plan pricing

https://vaibhavgujral.com/

Azure Function Tooling – Visual Studio

https://vaibhavgujral.com/

https://vaibhavgujral.com/

Azure Function Tooling – VS Code

https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions

https://vaibhavgujral.com/
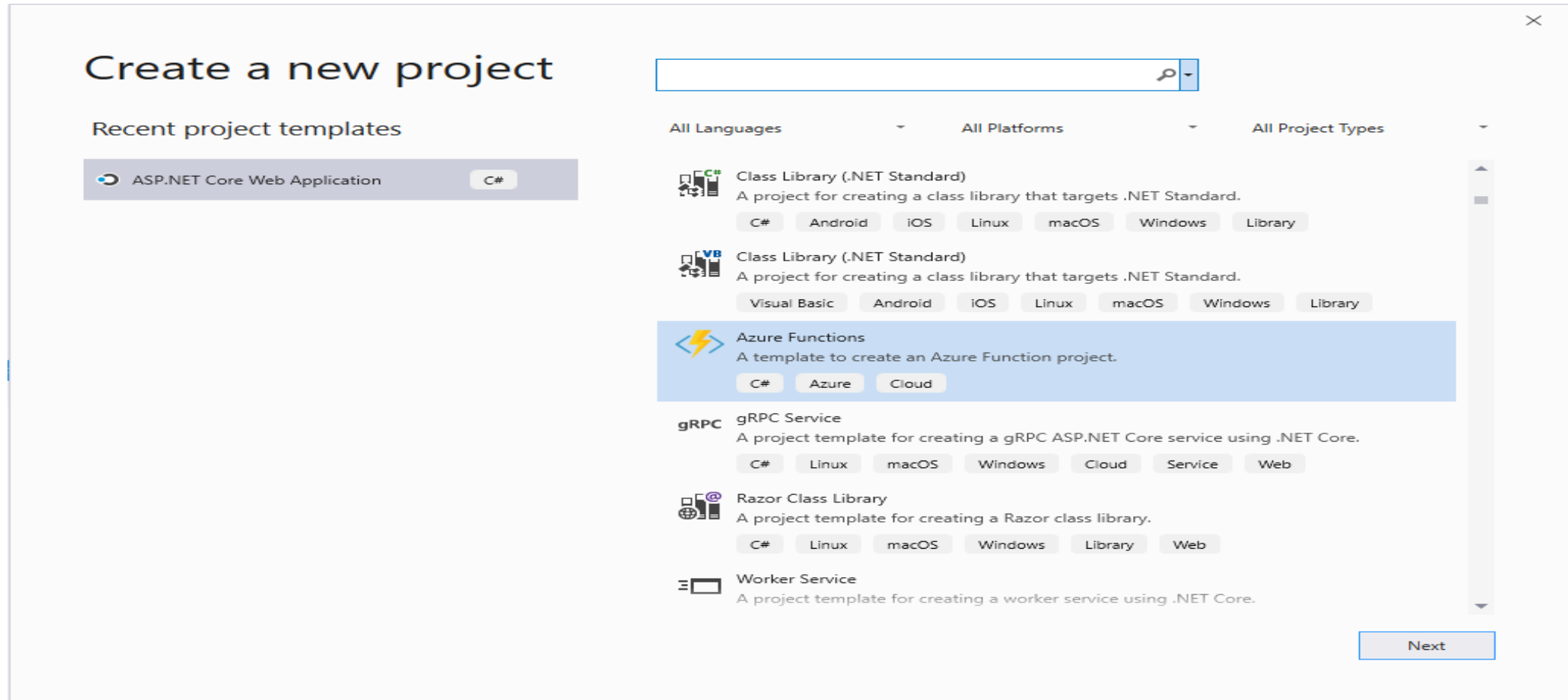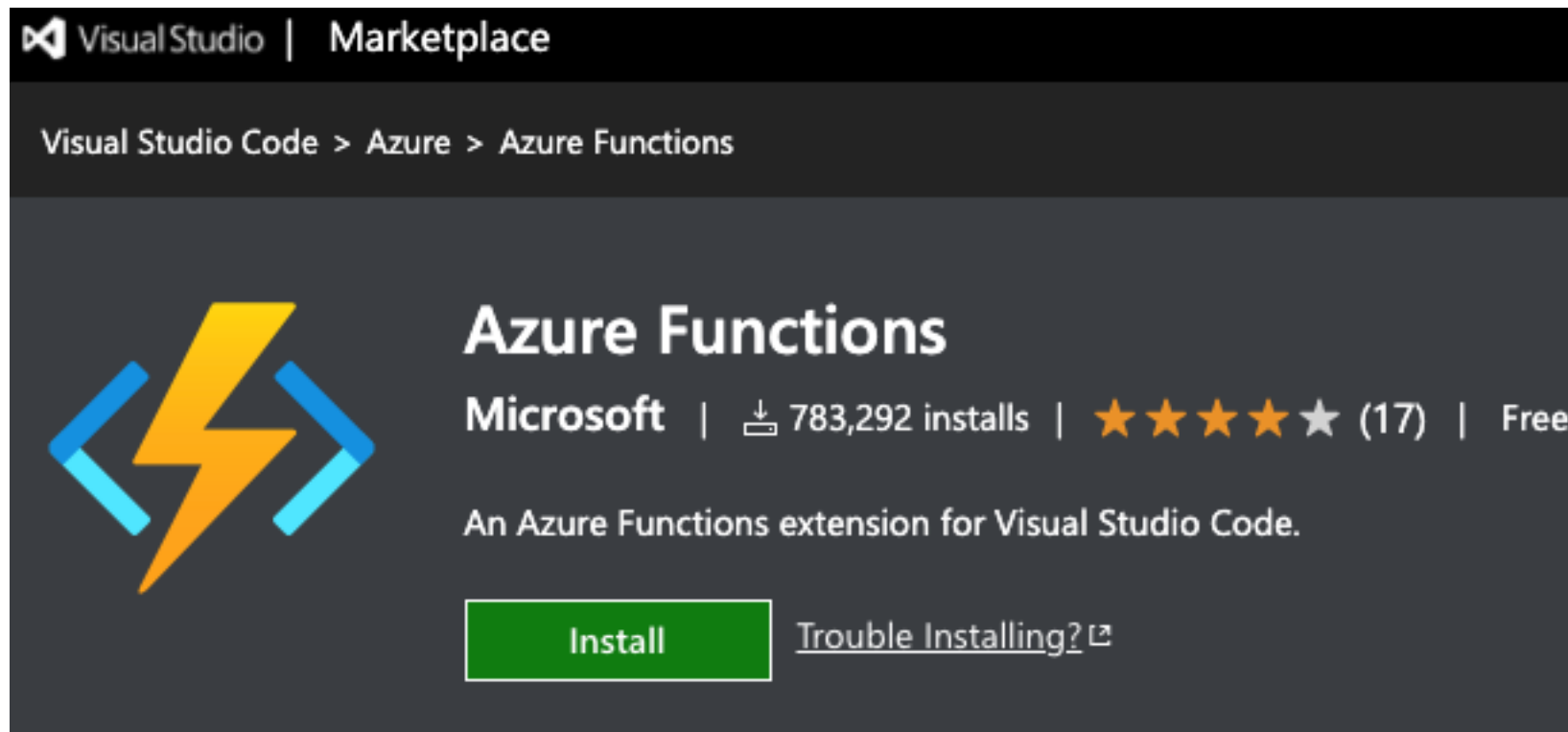
Azure Functions Core Tools

- Azure Functions Core Tools
  - Develop/test/debug/deploy functions on local computer from command prompt or terminal
- v1.x
  - Supported on Windows
- v2.x/3.x/4.x
  - Supports Windows, macOS and Linux
  - Needs .Net core 2.0/3.0 and Node.js (including npm)

https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local

https://vaibhavgujral.com/

1. cu ate compl flows
2. ta g ent
3. No err ling in wor
4. Short e n duration

Function 1
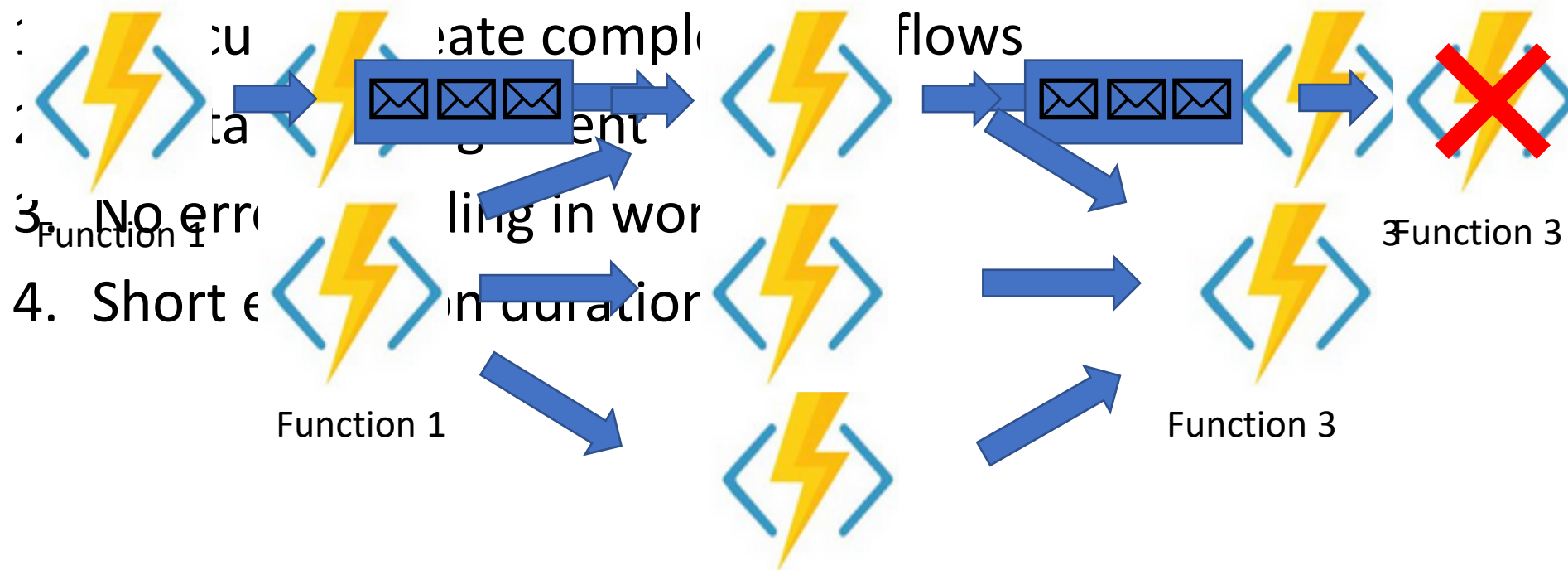
Function 1

Function 3

Function 3

- An extension of Azure Functions
- Enables stateful functions in server-less environment
- Simplifies complex, stateful coordination problems in serverless applications
- Provides stateful orchestration of function execution
- Built on top of Durable Task Framework
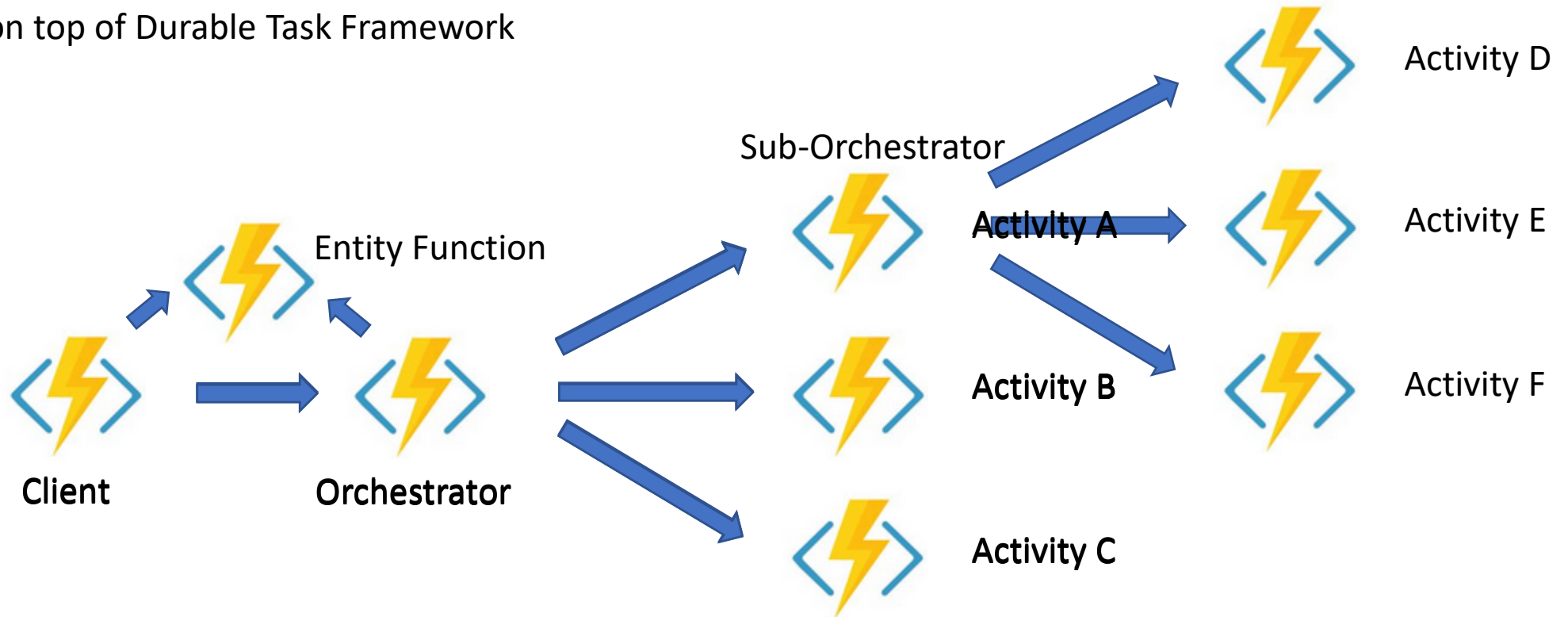
Entity Function

Sub-Orchestrator

Activity D

Activity A

Activity E

Activity B

Activity F

Client

Orchestrator

Activity C

Orchestration client binds the activity function to the functions using OrchestrationTriggerAttribute
Activity trigger triggers delegated to orchestrator function

```
[FunctionName("QueueStart")]
public static Task Run(

[FunctionName("SayHello")]
public static string SayHello([ActivityTrigger] IDurableActivityContext helloContext)
{
    string name = helloContext.GetInput<string>();
    return $"Hello {name}!";
}
```

26

https://vaibhavgujral.com/

https://vaibhavgujral.com/
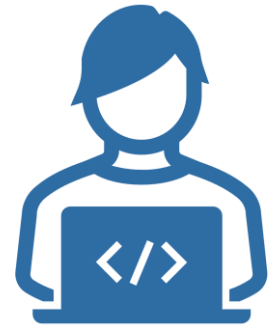
Durable Functions Language Support

Language stack	Azure Functions Runtime versions	Language worker version	Minimum bundles version
.NET / C# / F#	Functions 1.0+	In-process (GA) Out-of-process (preview ⧉)	n/a
JavaScript/TypeScript	Functions 2.0+	Node 8+	2.x bundles
Python	Functions 2.0+	Python 3.7+	2.x bundles
PowerShell	Functions 3.0+	PowerShell 7+	2.x bundles
Java (preview)	Functions 3.0+	Java 8+	4.x bundles

https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview

https://vaibhavgujral.com/

- Durable functions utilize event sourcing pattern and uses Azure storage as the backend.
- Logical container for Azure Storage resources used for orchestrators.
- Orchestrators and activities must belong to the same task hub to interact

https://vaibhavgujral.com/
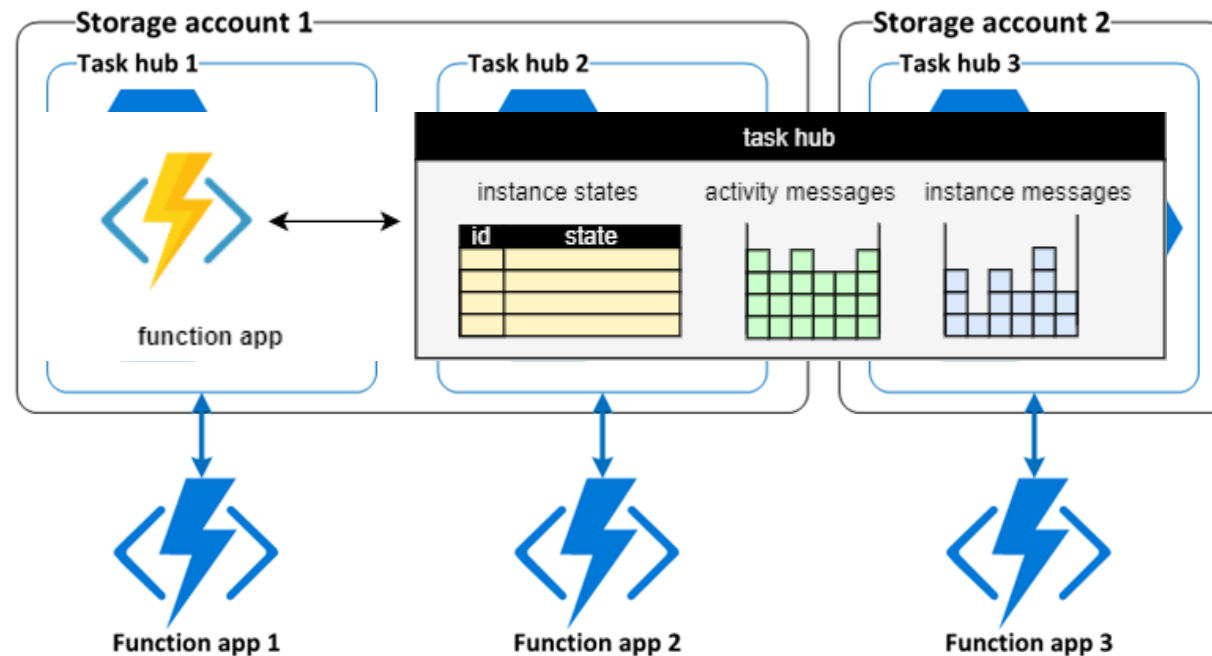
Durable Functions workflow

Orchestrator Started	**Orchestrator Wakes up & Completed**
↓	↑
Activity 1 Started	**Activity 2 Completed**
↓	↑
Orchestrator goes to sleep	**Orchestrator goes to sleep**
↓	↑
Activity 1 Completed	**Activity 2 Started**

Orchestrator wakes up

https://vaibhavgujral.com/
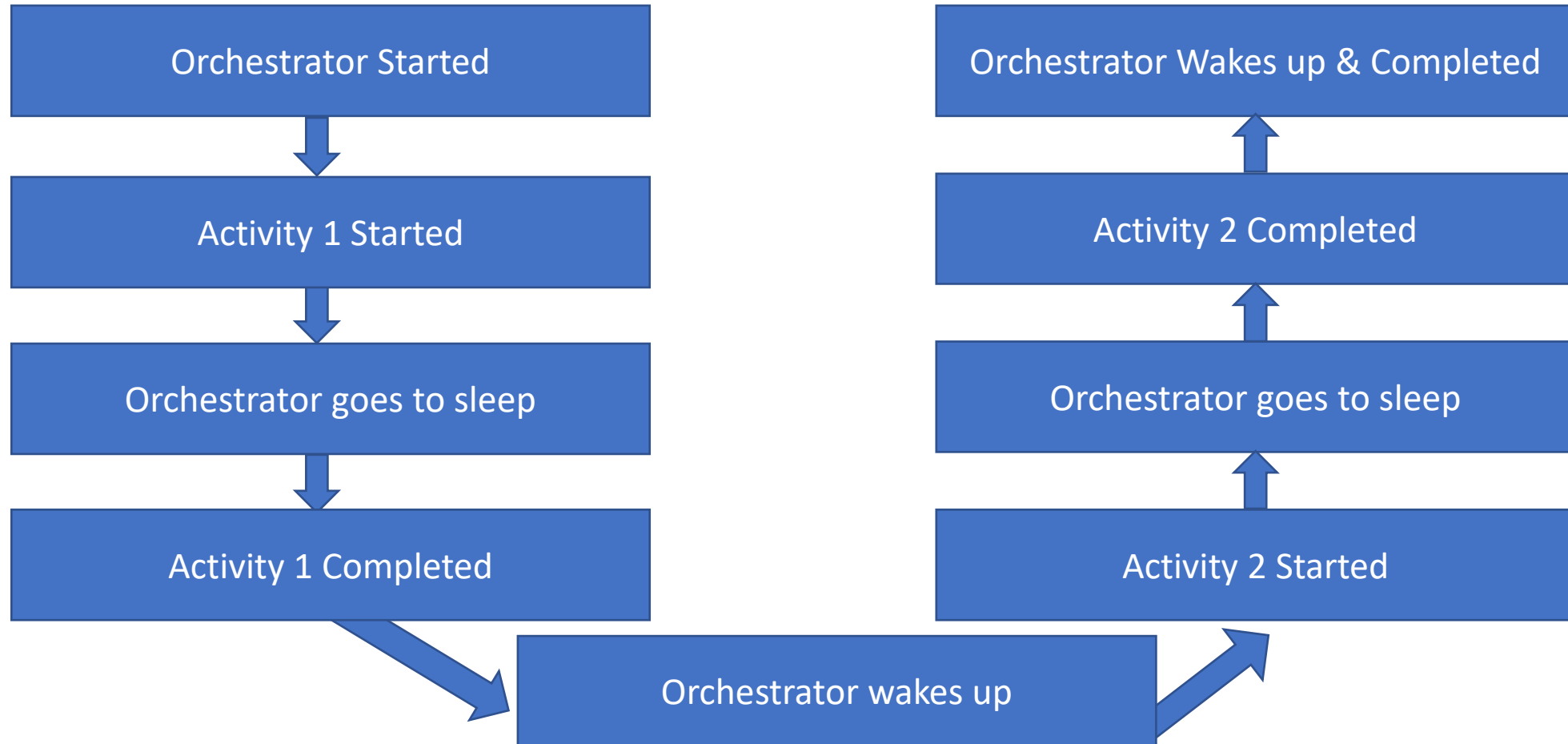
Orchestrator replays from the start whenever it wakes up

```csharp
[FunctionName("FunctionChaining")]
0 references
public static async Task<List<string>> RunOrchestrator(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    try
    {
        var outputs = new List<string>();

        outputs.Add(await context.CallActivityAsync<string>("Activity1", "Tokyo"));
        outputs.Add(await context.CallActivityAsync<string>("Activity2", "Seattle"));
        outputs.Add(await context.CallActivityAsync<string>("Activity3", "London"));

        return outputs;
    }
    catch (System.Exception)
    {
        return new List<string>();
    }
}
```

- Error handling can be implemented in orchestrator and activity functions using programming language's native error-handling features.

- Exception from an activity is thrown as ***FunctionFailedException*** to the orchestrator.

- An automatic retry policy can be used to call activities or sub-orchestrators to come out of transient failures.

- Use ***du*** orchest

```
[FunctionName("Function")]
[FunctionName("TimerOrchestratorWithRetry")]
public static async Task Run([OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var retryOptions = new RetryOptions(
        firstRetryInterval: TimeSpan.FromSeconds(5),
        maxNumberOfAttempts: 3);

    await context.CallActivityWithRetryAsync("FlakyFunction", retryOptions, null);

    // ...
}
```

https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-error-handling

https://vaibhavgujral.com/

- Used to perform management tasks on orchestrations, entities and task hubs

- Webhooks authorized by Azure Functions host but handled by Durable functions extension

1. Start orchestration
2. Get (all) instance(s) status
3. Purge single instance history
4. Purge multiple instance histories
5. Raise event
6. Terminate instance
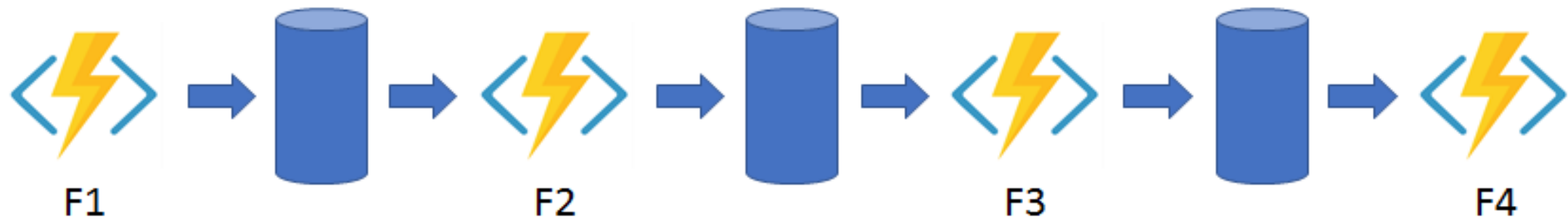
7. Signal entity
8. Get entity
9. List entities

Response Codes –
- HTTP 202 (Accepted)
- HTTP 400 (Bad request)

Parameter	Description
taskHub	The name of the task hub.
connection	The name of the connection string for the storage account.
systemKey	The authorization key required to invoke the API

https://vaibhavgujral.com/
https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-http-api
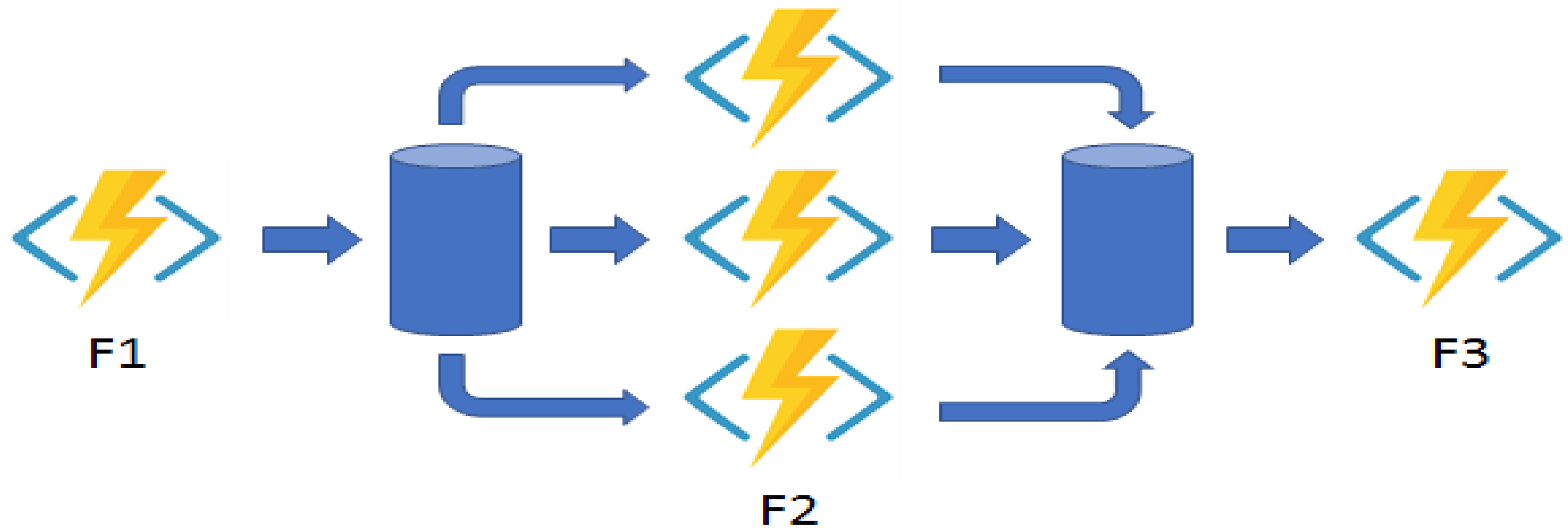
F1   F2   F3   F4

Function Chaining

```csharp
[FunctionName("FunctionChaining")]
0 references
public static async Task<List<string>> RunOrchestrator(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    try
    {
        var outputs = new List<string>();

        outputs.Add(await context.CallActivityAsync<string>("Activity1", "Tokyo"));
        outputs.Add(await context.CallActivityAsync<string>("Activity2", "Seattle"));
        outputs.Add(await context.CallActivityAsync<string>("Activity3", "London"));

        return outputs;
    }
    catch (System.Exception)
    {
        throw;
    }
}
```

https://vaibhavgujral.com/

```csharp
[FunctionName("E2_BackupSiteContent")]
0 references
public static async Task<long> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext backupContext)
{
    string rootDirectory = "D:\\Temp";

    string[] files = await backupContext.CallActivityAsync<string[]>("E2_GetFileList", rootDirectory);

    var tasks = new Task<long>[files.Length];
    for (int i = 0; i < files.Length; i++)
    {
        tasks[i] = backupContext.CallActivityAsync<long>("E2_CopyFileToBlob", files[i]);
    }

    await Task.WhenAll(tasks);

    long totalBytes = tasks.Sum(t => t.Result);
    return totalBytes;
}
```

https://vaibhavgujral.com/

https://vaibhavgujral.com/
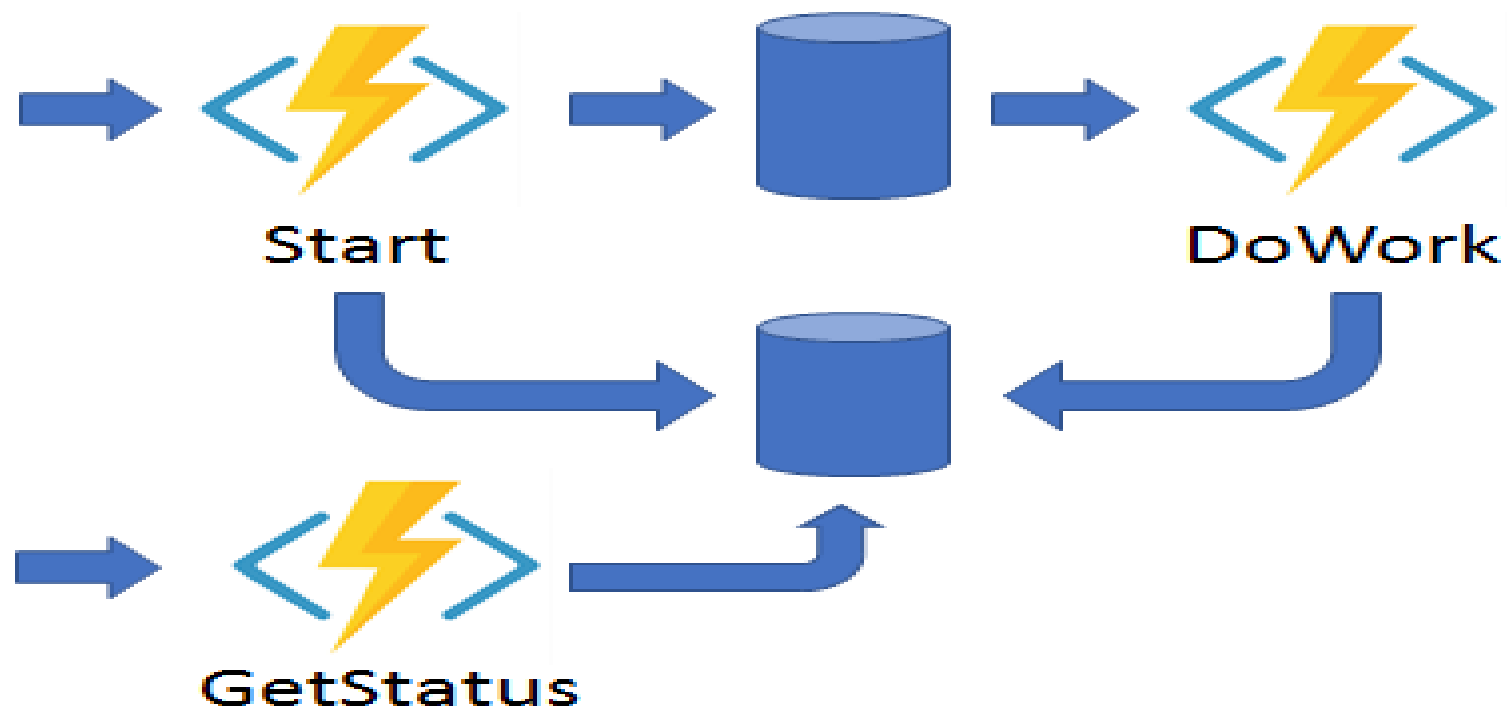
Async Http APIs

```
> curl -X POST https://myfunc.azurewebsites.net/api/orchestrators/DoWork -H "Content-Length: 0" -i
HTTP/1.1 202 Accepted
Content-Type: application/json
Location: https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/instances/b79baf67f717453ca9e86c5da21

{"id":"b79baf67f717453ca9e86c5da21e03ec", ...}
```

```
> curl https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/instances/b79baf67f717453ca9e86c5da21e03
HTTP/1.1 202 Accepted
Content-Type: application/json
Location: https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/instances/b79baf67f717453ca9e86c5da21

{"runtimeStatus":"Running","lastUpdatedTime":"2019-03-16T21:20:47Z", ...}
```
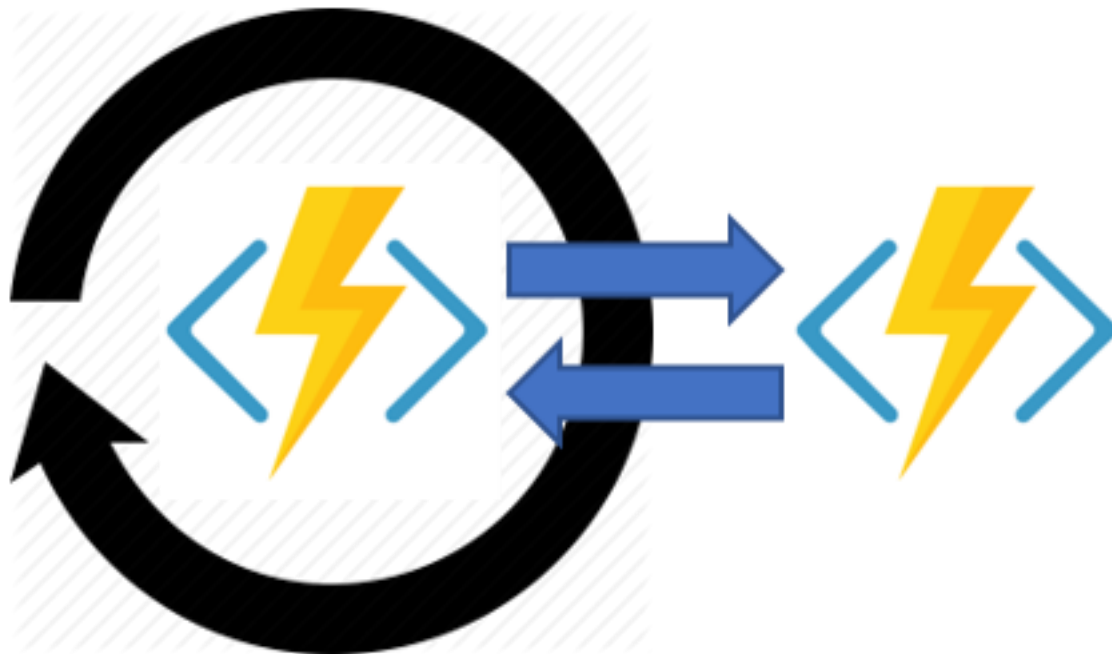
```
> curl https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/instances/b79baf67f717453ca9e86c5da21e03
HTTP/1.1 200 OK
Content-Length: 175
Content-Type: application/json

{"runtimeStatus":"Completed","lastUpdatedTime":"2019-03-16T21:20:57Z", ...}
```

https://vaibhavgujral.com/

https://vaibhavgujral.com/

```csharp
[FunctionName("MonitorJobStatus")]
public static async Task Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    int jobId = context.GetInput<int>();
    int pollingInterval = GetPollingInterval();
    DateTime expiryTime = GetExpiryTime();

    while (context.CurrentUtcDateTime < expiryTime)
    {
        var jobStatus = await context.CallActivityAsync<string>("GetJobStatus", jobId);
        if (jobStatus == "Completed")
        {
            // Perform an action when a condition is met.
            await context.CallActivityAsync("SendAlert", machineId);
            break;
        }

        // Orchestration sleeps until this time.
        var nextCheck = context.CurrentUtcDateTime.AddSeconds(pollingInterval);
        await context.CreateTimer(nextCheck, CancellationToken.None);
    }

    // Perform more work here, or let the orchestration end.
}
```

https://vaibhavgujral.com/

RequestApproval → Person/Clock → ProcessApproval / Escalate

```csharp
[FunctionName("BudgetApproval")]
public static async Task Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    bool approved = await context.WaitForExternalEvent<bool>("Approval");
    if (approved)
    {
        // approval granted – do the approved action
    }
    else
    {
        // approval denied – send a notification
    }
}
```

You can use the RaiseEventAsync method to send an external event to an orchestration.

https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-phone-verification

https://vaibhavgujral.com/

- Read and update small pieces of state, known as *durable entities*

- Comes with a special trigger type, the *entity trigger*

- Unlike orchestrator functions, entity functions manage the state of an entity explicitly

- Entity functions are only available in Durable Functions 2.0
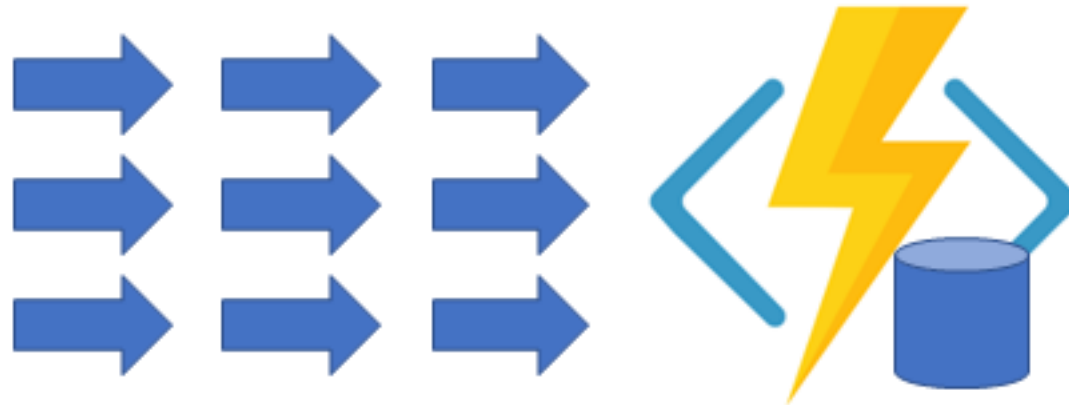
```csharp
[FunctionName("Counter")]
public static void Counter([EntityTrigger] IDurableEntityContext ctx)
{
    switch (ctx.OperationName.ToLowerInvariant())
    {
        case "add":
            ctx.SetState(ctx.GetState<int>() + ctx.GetInput<int>());
            break;
        case "reset":
            ctx.SetState(0);
            break;
        case "get":
            ctx.Return(ctx.GetState<int>());
            break;
    }
}
```

```csharp
[FunctionNa
0 references
public stat
    [Orches
{
    var ent

    // Two-
    int cur
    if (cur
    {
        //
        cor
    }
}
```

it a response

- Orchestrators can't do I/O directly. Instead, they call activities to do I/O operations.
- In Durable Functions v 2.x, you can call HTTP APIs directly from orchestrators.

```csharp
[FunctionName("CheckSiteAvailable")]
public static async Task CheckSiteAvailable(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    Uri url = context.GetInput<Uri>();

    // Makes an HTTP GET request to the specified endpoint
    DurableHttpResponse response =
        await context.CallHttpAsync(HttpMethod.Get, url);

    if (response.StatusCode >= 400)
    {
        // handling of error codes goes here
    }
}
```

https://vaibhavgujral.com/

Eternal Orchestrations

- Orchestrator functions that never end
- Used for aggregators and infinite loops

```csharp
[FunctionName("Trigger_Eternal_Orchestration")]
public static async Task<HttpResponseMessage> OrchestrationTrigger(
    [HttpTrigger(AuthorizationLevel.Function, "post", Route = null)] HttpRequestMessage request,
    [DurableClient] IDurableOrchestrationClient client)
{
    string instanceId = "StaticId";

    await client.StartNewAsync("Periodic_Cleanup_Loop", instanceId);
    return client.CreateCheckStatusResponse(request, instanceId);
}
        }
```

- Used for background jobs where you need only one instance of the orchestrator to run at a time.

```csharp
[FunctionName("HttpStartSingle")]
public static async Task<HttpResponseMessage> RunSingle(
    [HttpTrigger(AuthorizationLevel.Function, methods: "post", Route = "orchestrators/{functionName}/{instanceId}")] HttpRequestMessage req,
    [DurableClient] IDurableOrchestrationClient starter,
    string functionName,
    string instanceId,
    ILogger log)
{
    // Check if an instance with the specified ID already exists or an existing one stopped running(completed/failed/terminated).
    var existingInstance = await starter.GetStatusAsync(instanceId);
    if (existingInstance == null
    || existingInstance.RuntimeStatus == OrchestrationRuntimeStatus.Completed
    || existingInstance.RuntimeStatus == OrchestrationRuntimeStatus.Failed
    || existingInstance.RuntimeStatus == OrchestrationRuntimeStatus.Terminated)
    {
        // An instance with the specified ID doesn't exist or an existing one stopped running, create one.
        dynamic eventData = await req.Content.ReadAsAsync<object>();
        await starter.StartNewAsync(functionName, instanceId, eventData);
        log.LogInformation($"Started orchestration with ID = '{instanceId}'.");
        return starter.CreateCheckStatusResponse(req, instanceId);
    }
    else
    {
        // An instance with the specified ID exists or an existing one still running, don't create one.
        return new HttpResponseMessage(HttpStatusCode.Conflict)
        {
            Content = new StringContent($"An instance with ID '{instanceId}' already exists."),
        };
    }
}
```

- Versioning may be required when –
  - Changing activity or entity function signatures
  - Changing orchestrator logic

- Three strategies for dealing with versioning challenges:
  - Do nothing (not recommended)
  - Stop all in-flight instances
  - Side-by-side deployments

Durable Functions Limitations

- Orchestrator functions can call only deterministic APIs.
- Code updates can break replay behavior for unfinished orchestrations.
- Orchestrators can't do I/O directly. Instead, they call activities to do I/O operations.

https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-code-constraints

https://vaibhavgujral.com/

Resources for further reading

- Azure Functions Documentation – https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview

- Durable Functions – https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview

- Durable Functions Patterns - https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp#application-patterns

- Azure Functions YouTube Channel - https://www.youtube.com/watch?v=4n0QbG0wYEI

- Durable Task Framework - https://github.com/Azure/durabletask

https://vaibhavgujral.com/

https://vaibhavgujral.com/

Contact Information

AZURE DAY

LinkedIn	**Twitter**	**YouTube**	**Blog**	**Email**
vaibhavgujral	vaibhavgujral_	vaibhavgujral	https://vaibhavgujral.com	vaibhav@vaibhavgujral.com

https://vaibhavgujral.com/

https://vaibhavgujral.com/