#PitchOnline presents:

Implementare Stateful pattern con le Azure Functions

Massimo Bonanni

Paranormal Trainer, with the head in the Cloud and all the REST in microservices!

massimo.bonanni@microsoft.com

@massimobonanni

Scifoni Ivano

Fabio Mannis

Francesco Del Re

Matteo Riccardi

Valerio Benedetti

What is serverless?

Full abstraction of servers

Developers can just focus on their code—there are no distractions around server management, capacity planning, or availability.

Instant, event-driven scalability

Application components react to events and triggers in near real-time with virtually unlimited scalability; compute resources are used as needed.

Pay-per-use

Only pay for what you use: billing is typically calculated on the number of function calls,
code execution time, and memory used.*

*Supporting services, like storage and networking, may be charged separately.

What are Azure Functions?

An event-based, serverless compute experience that accelerates app development

Azure Functions = FaaS++

Integrated programming model

Use built-in triggers and bindings to define when a function is invoked and to what data it connects

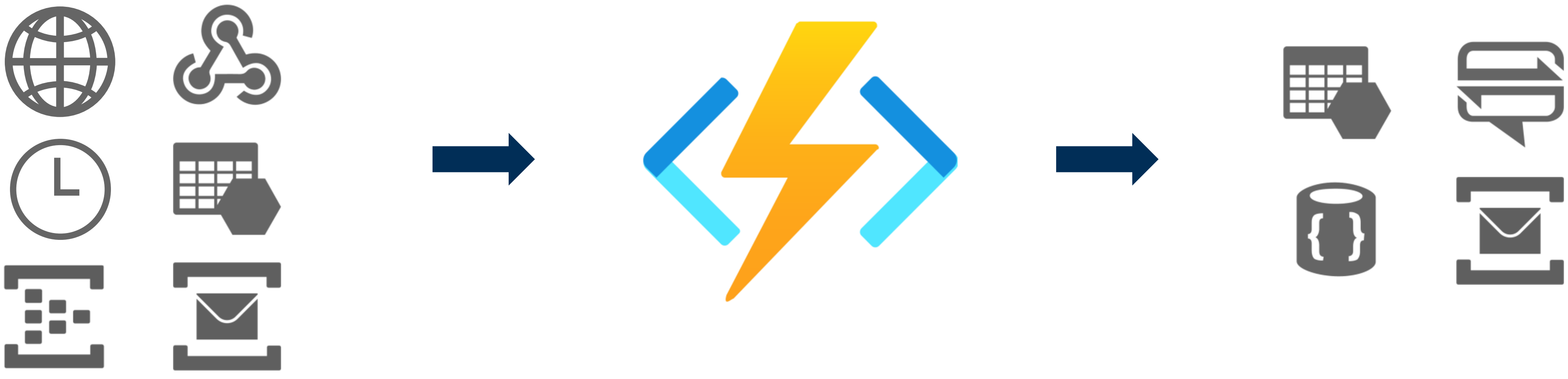
Enhanced development experience

Code, test and debug locally using your preferred editor or the easy-to-use web based interface including monitoring

Hosting options flexibility

Choose the deployment model that better fits your business needs without compromising development experience

What are Azure Functions?



Events

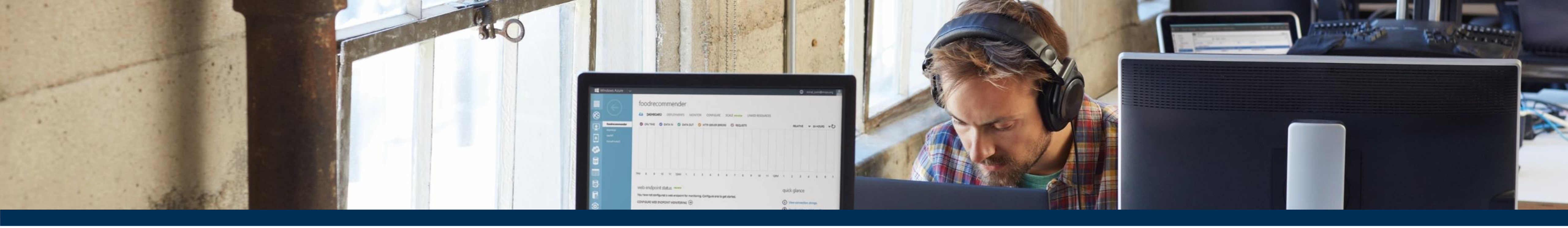
React to timers, HTTP, or events from your favorite Azure services, with more on the way

Code

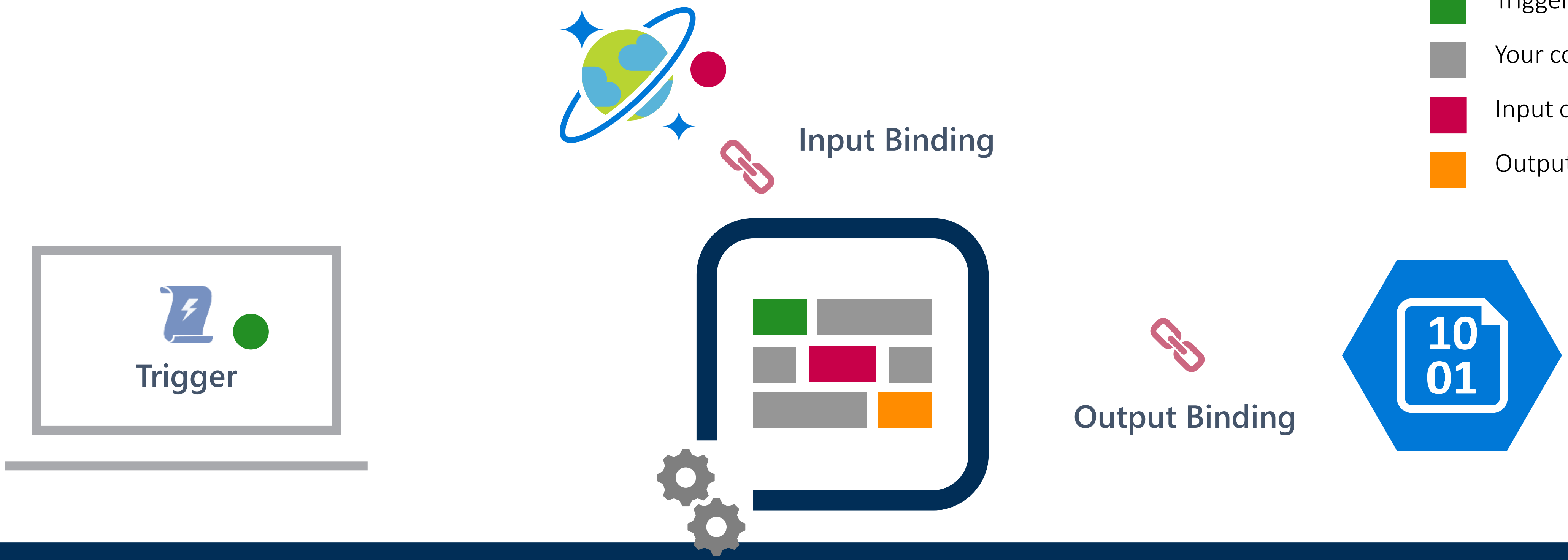
Author functions in C#, F#, Node.JS, Java, and more

Outputs

Send results to an ever-growing collection of services



Boost development efficiency



Input Binding

Trigger

Output Binding

🟩 Trigger object

⬛ Your code

🟥 Input object

🟧 Output object

FaaS principles and best practices

Functions must be stateless

Functions must not call other functions

Functions should do only one thing

… and workflows!?!?!

Workflows manage state

Workflow is interactions between components

Workflows must do more than one thing

The magic is Durable Functions!!

What are Durable Functions?

Azure Functions Extension

- Based on Azure Functions
- Adds new Triggers and Binding
- Manages state, checkpoints, and restarts

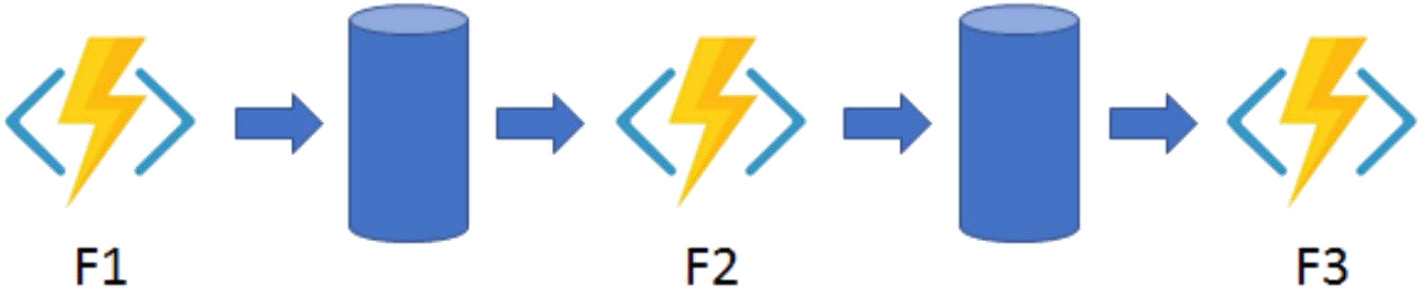
Durable Task Framework

- Long running persistent workflows in C#
- Used within various teams at Microsoft to reliably orchestrate long running operations

Languages

- C#
- JavaScript
- F#

Function chaining



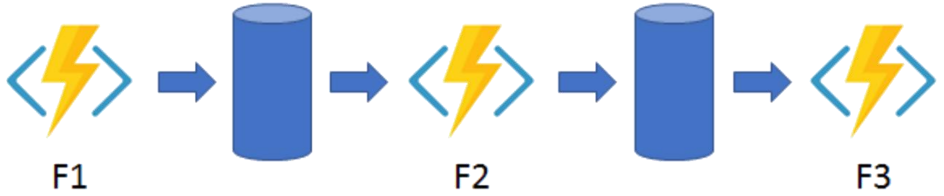
- ⚠ Relations between functions and queues aren’t clearly identifying
- ⚠ Queues are an implementation detail
- ⚠ Operation context management is difficult
- ⚠ Error handling is difficult

Function chaining in Durable Functions

Orchestrator Function

Activity Functions

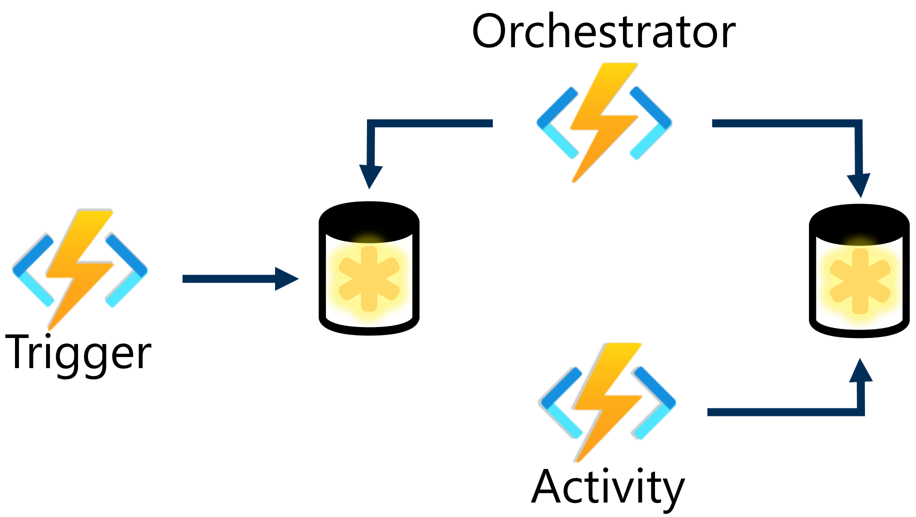
```
[FunctionName("FunctionsChainingOrchestrator")]
public static async Task<int> Orchestrator([OrchestrationTrigger] IDurableOrchestrationContext context)
{
    try
    {
        var x = await context.CallActivityAsync<int>("F1", null);
        var y = await context.CallActivityAsync<int>("F2", x);
        return await context.CallActivityAsync<int>("F3", y);
    }
    catch (Exception)
    {
        // Error handling ...
    }
    return 0;
}
```



The magic is Event Sourcing!!

Orchestrator Function

```
var x = await context.CallActivityAsync<int>("F1", null);
var y = await context.CallActivityAsync<int>("F2", x);
return  await context.CallActivityAsync<int>("F3", y);
```



```
F1 => return 42;
F2 => return value + 1;
F3 => return value + 2;
```

Event History

Orchestrator Started
Task Scheduled, F1
Task Completed, F1 => 42
Task Scheduled, F2
Task Completed, F2 => 43
Task Scheduled, F3
Task Completed, F3 => 45
Orchestrator Completed => 45

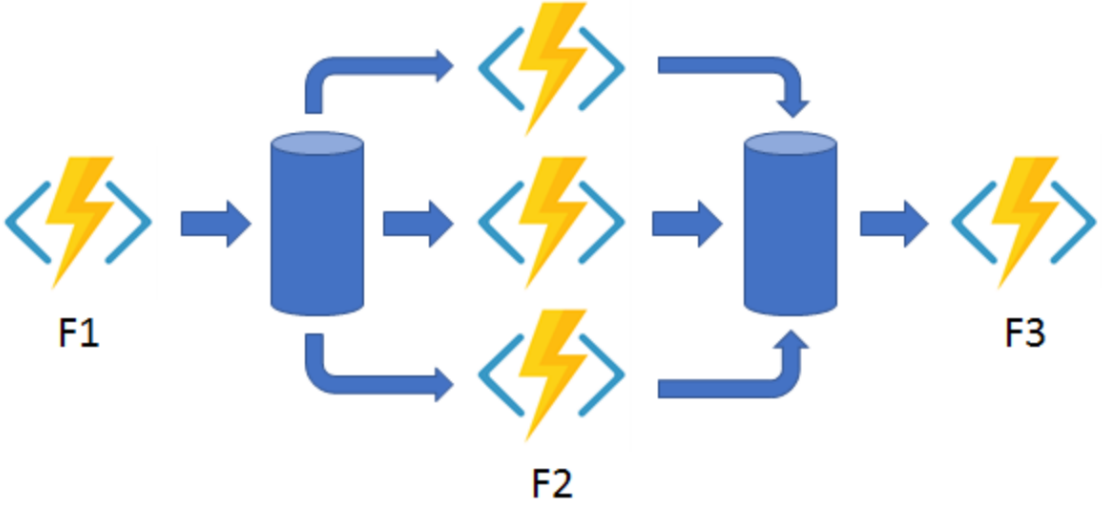
DEMO

Event History

Orchestrator MUST be deterministic

- ⚠ Never write logic that depends on random numbers, current date/time, delay, etc.
- ⚠ Never do I/O in the orchestrator function
- ⚠ Never start custom thread in the orchestrator function
- ⚠ Do not write infinite loops

FanIn-FanOut



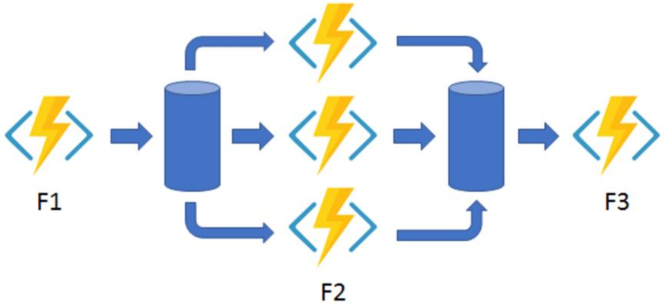
- FanIn is simple, but FanOut is more complicated
- The platform must track progress of all work
- All the same issues of Function Chain

FanIn-FanOut in Durable Functions

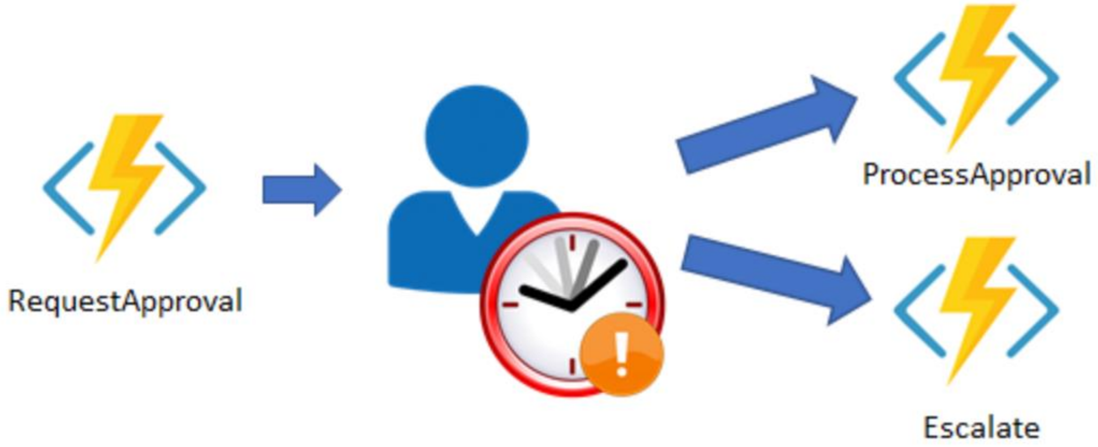
```
[FunctionName("FanOutFanInOrchestrator")]
public static async Task<int> Run([OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var parallelTasks = new List<Task<int>>();

    var workBatch = await context.CallActivityAsync<in

    for (var i = 0; i < workBatch.Length; i++)
    {
        Task<int> task = context.CallActivityAsync<int>("F2", workBatch[i]);
        parallelTasks.Add(task);
    }
    await Task.WhenAll(parallelTasks);
    var sum = parallelTasks.Sum(t => t.Result);

    return await context.CallActivityAsync<int>("F3", sum);
}
```

FanOut

FanIn

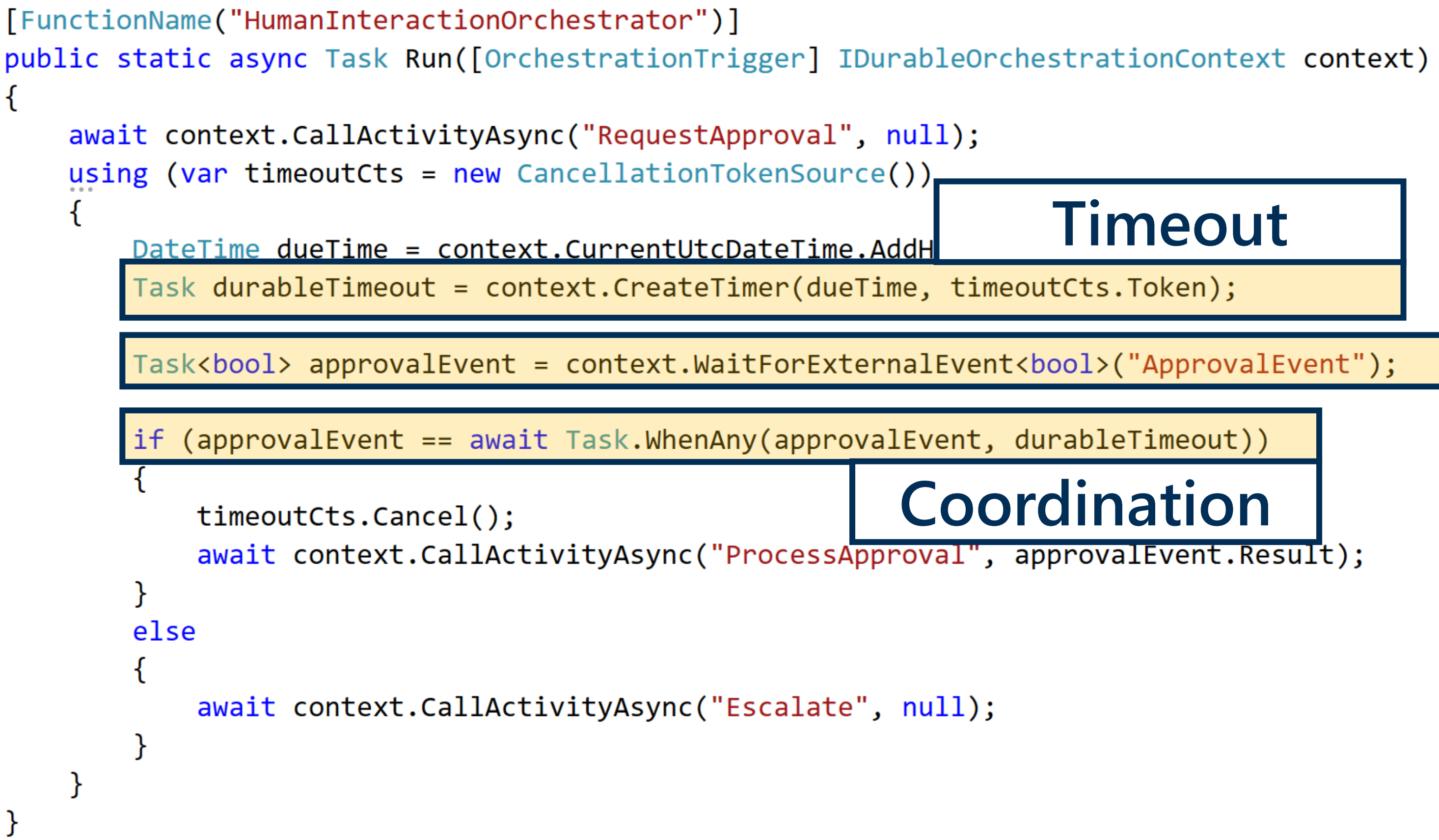


Human Interaction



- Handling race conditions between timeouts and approval
- Need mechanism for implementing and cancelling timeout events
- Same issues as the other pattern

Human Interaction in Durable Functions

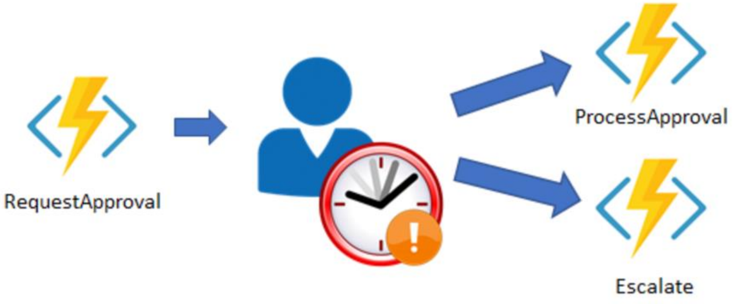


```
[FunctionName("HumanInteractionOrchestrator")]
public static async Task Run([OrchestrationTrigger] IDurableOrchestrationContext context)
{
    await context.CallActivityAsync("RequestApproval", null);
    using (var timeoutCts = new CancellationTokenSource())
    {
        DateTime dueTime = context.CurrentUtcDateTime.AddH
        Task durableTimeout = context.CreateTimer(dueTime, timeoutCts.Token);

        Task<bool> approvalEvent = context.WaitForExternalEvent<bool>("ApprovalEvent");

        if (approvalEvent == await Task.WhenAny(approvalEvent, durableTimeout))
        {
            timeoutCts.Cancel();
            await context.CallActivityAsync("ProcessApproval", approvalEvent.Result);
        }
        else
        {
            await context.CallActivityAsync("Escalate", null);
        }
    }
}
```

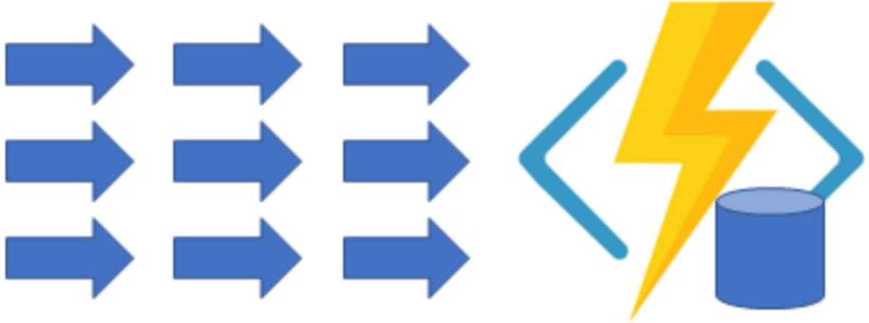
Timeout

Human

Coordination



Aggregator



- ⚠ Storing the state
- ⚠ Correlation of event for a particular state
- ⚠ Syncronization of access to the state

Actor model

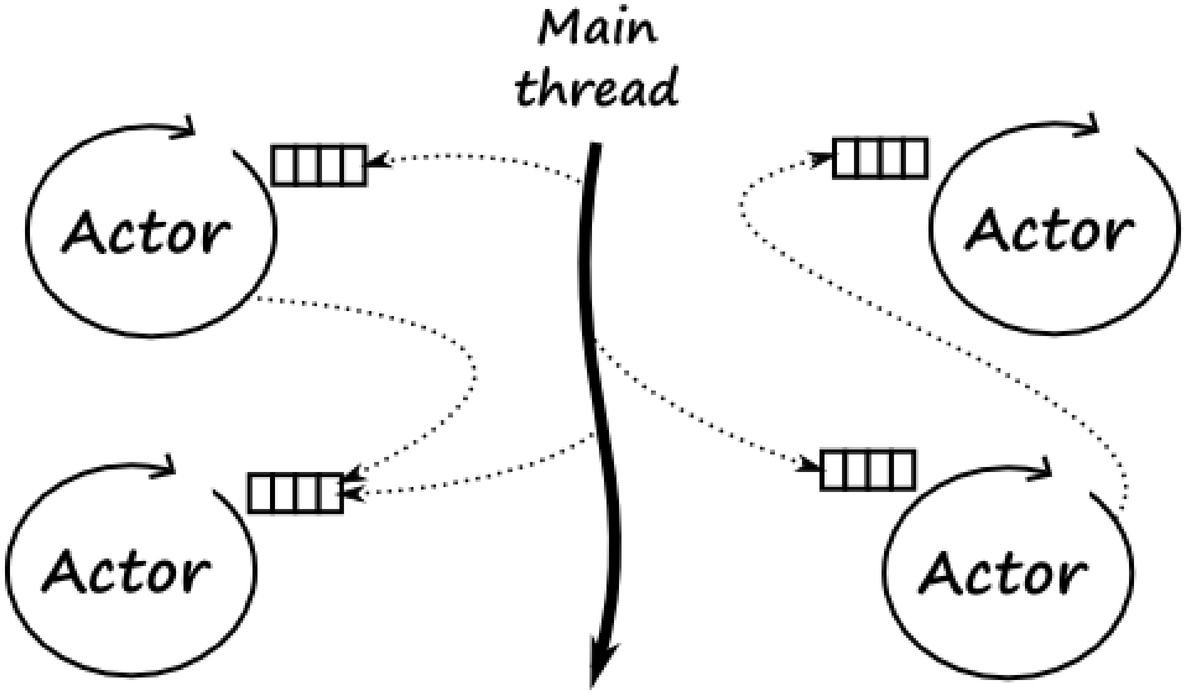
The actor model in computer science is a mathematical model of concurrent computation (originated in 1973).

In response to a message it receives, an actor can:

- make local decisions,
- create more actors,
- send more messages,
- determine how to respond to the next message received.

Actors are identified by ids and have their own private state.

Actors can process only one message at time.



The magic is Durable Entities!!

Durable Entities aka Entity Functions

Entity Functions define operations for reading and updating small piece of state

Entity Functions are functions with special trigger

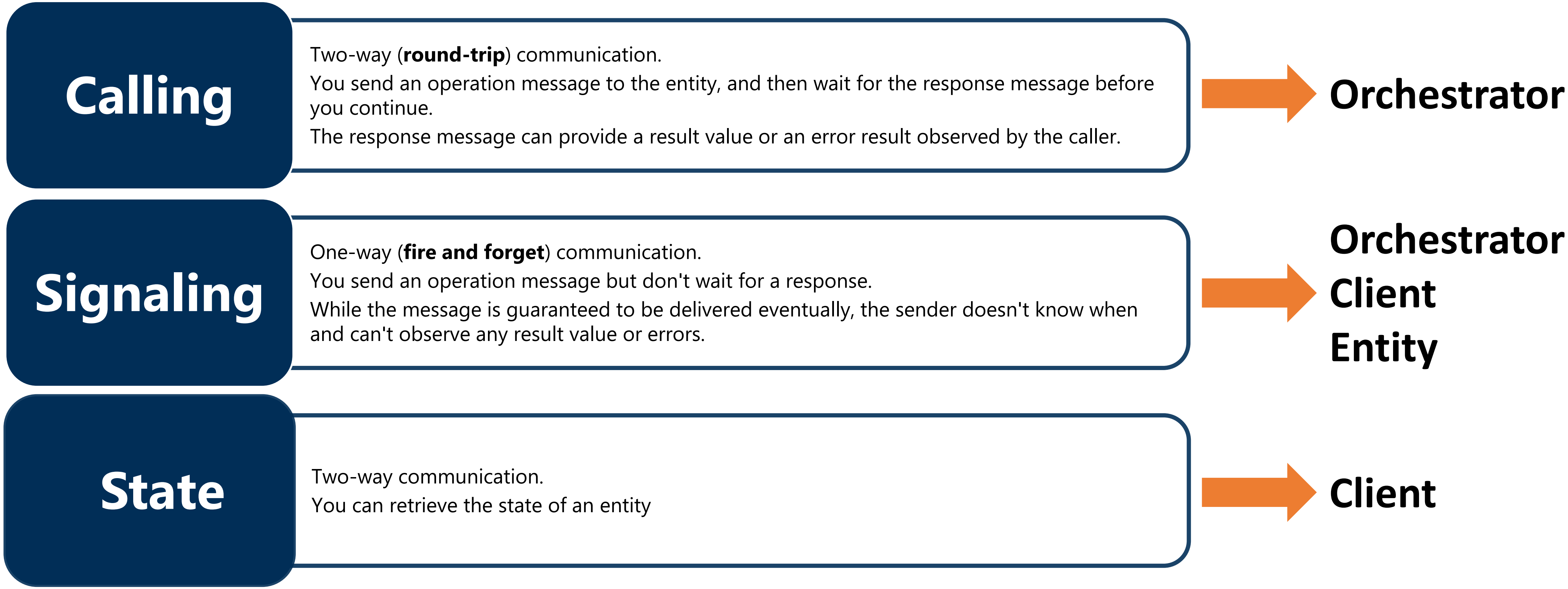
Entity Functions are accessed using:

- Entity Name
- Entity key

Entity Functions expose operations that can be accessed using:

- Entity Key
- Operation Name
- Operation Input
- Scheduled time

Access Entities



Calling

Two-way (**round-trip**) communication.

You send an operation message to the entity, and then wait for the response message before you continue.

The response message can provide a result value or an error result observed by the caller.

→ **Orchestrator**

Signaling

One-way (**fire and forget**) communication.

You send an operation message but don't wait for a response.

While the message is guaranteed to be delivered eventually, the sender doesn't know when and can't observe any result value or errors.

→ **Orchestrator**
Client
Entity

State

Two-way communication.

You can retrieve the state of an entity

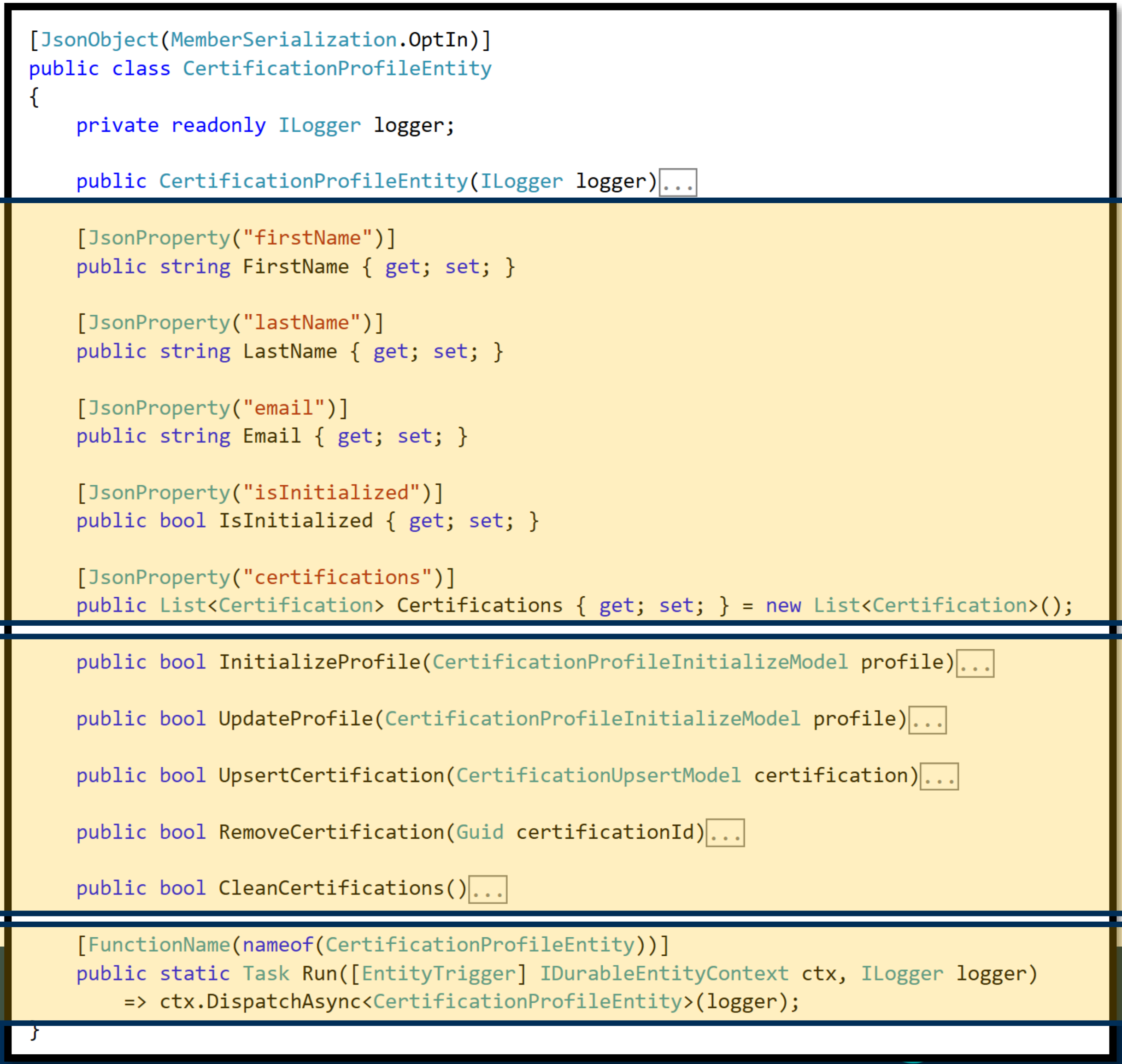
→ **Client**

Anatomy of an Entity

Properties (state)

Operations

Entry Function



```
[JsonObject(MemberSerialization.OptIn)]
public class CertificationProfileEntity
{
    private readonly ILogger logger;

    public CertificationProfileEntity(ILogger logger)...

    [JsonProperty("firstName")]
    public string FirstName { get; set; }

    [JsonProperty("lastName")]
    public string LastName { get; set; }

    [JsonProperty("email")]
    public string Email { get; set; }

    [JsonProperty("isInitialized")]
    public bool IsInitialized { get; set; }

    [JsonProperty("certifications")]
    public List<Certification> Certifications { get; set; } = new List<Certification>();

    public bool InitializeProfile(CertificationProfileInitializeModel profile)...

    public bool UpdateProfile(CertificationProfileInitializeModel profile)...

    public bool UpsertCertification(CertificationUpsertModel certification)...

    public bool RemoveCertification(Guid certificationId)...

    public bool CleanCertifications()...

    [FunctionName(nameof(CertificationProfileEntity))]
    public static Task Run([EntityTrigger] IDurableEntityContext ctx, ILogger logger)
        => ctx.DispatchAsync<CertificationProfileEntity>(logger);
}
```

Durable Entities vs Virtual Actor

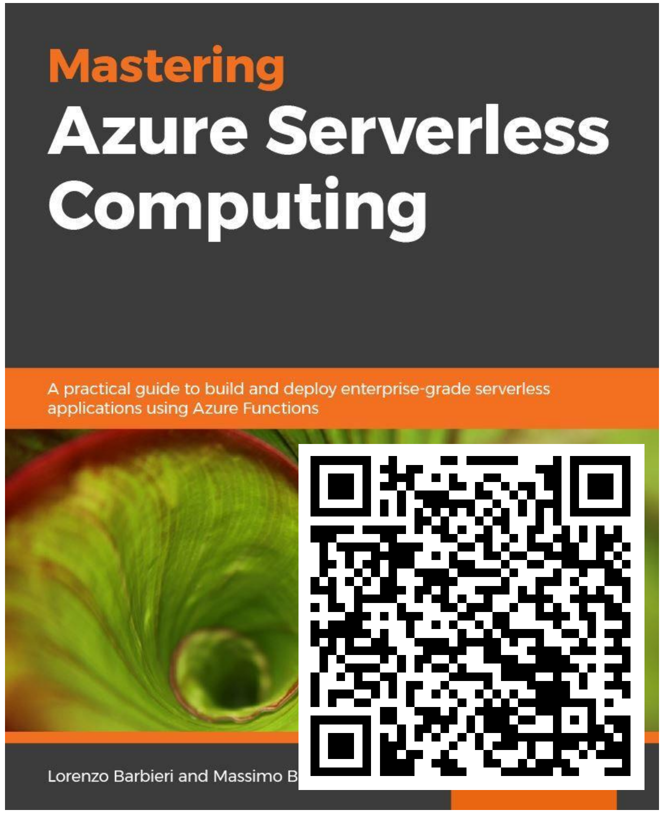
	Durable Entities	Virtual Actors (Orleans)
Addressable via Entity ID	✓	✓
Execute operations serially	✓	✓
Created implicit when are called	✓	✓
Garbaged when not used	✓	✓
Durability vs Latency	Durability	Latency
Timeout messaging	No timeout	Timeout
Message order	FIFO	FIFO not guaranteed
Message Deadlock	No deadlock	Deadlock

DEMO

Certification Profiles Management

Takeaways

- Designed for reliability, not for latency
- Workflow by code
- Similar to Virtual Actor but not the same
- Solve the concurrency, but think if is the right choice



Mastering
Azure Serverless Computing

A practical guide to build and deploy enterprise-grade serverless applications using Azure Functions

Lorenzo Barbieri and Massimo B

http://bit.ly/MasteringServerless

Connect with me on LinkedIn

linkedin.com/in/massimobonanni/

Massimo Bonanni

Azure Technical Trainer @ Microsoft

massimo.bonanni@microsoft.com
@massimobonanni

Microsoft CERTIFIED AZURE FUNDAMENTALS
Microsoft CERTIFIED AZURE DEVELOPER ASSOCIATE
Microsoft CERTIFIED AZURE SOLUTIONS ARCHITECT EXPERT
Microsoft CERTIFIED AZURE DEVOPS ENGINEER EXPERT
Microsoft CERTIFIED AZURE ADMINISTRATOR ASSOCIATE
Microsoft CERTIFIED AZURE AI ENGINEER ASSOCIATE
Microsoft CERTIFIED AZURE IOT DEVELOPER SPECIALTY
MICROSOFT CERTIFIED TRAINER Microsoft MCT Trainer 2020 - 2021

Coding

#Coding

Thank You!

Our Socials

f ● in

● ●

References

- **Durable Functions overview**
  https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp
- **Developer's guide to durable entities in .NET**
  https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-dotnet-entities
- **Entity Functions**
  https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-entities?tabs=csharp
- **Durable Task Framework**
  https://github.com/Azure/durabletask
- **GitHub Demo**
  https://github.com/massimobonanni/StatefulPatternFunctions