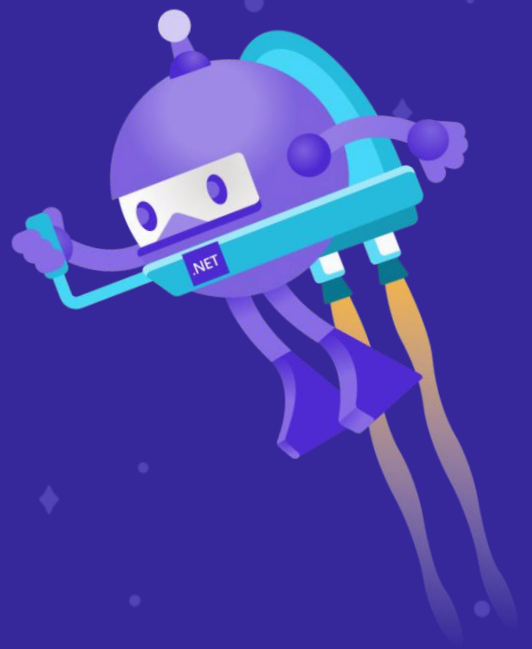
Writing stateful patterns with .NET Core

Massimo Bonanni

Azure Technical Trainer

@massimobonanni



SPONSOR


Microsoft


avanade


ENGINEERING


blexin.
CON NOI È SEMPLICE

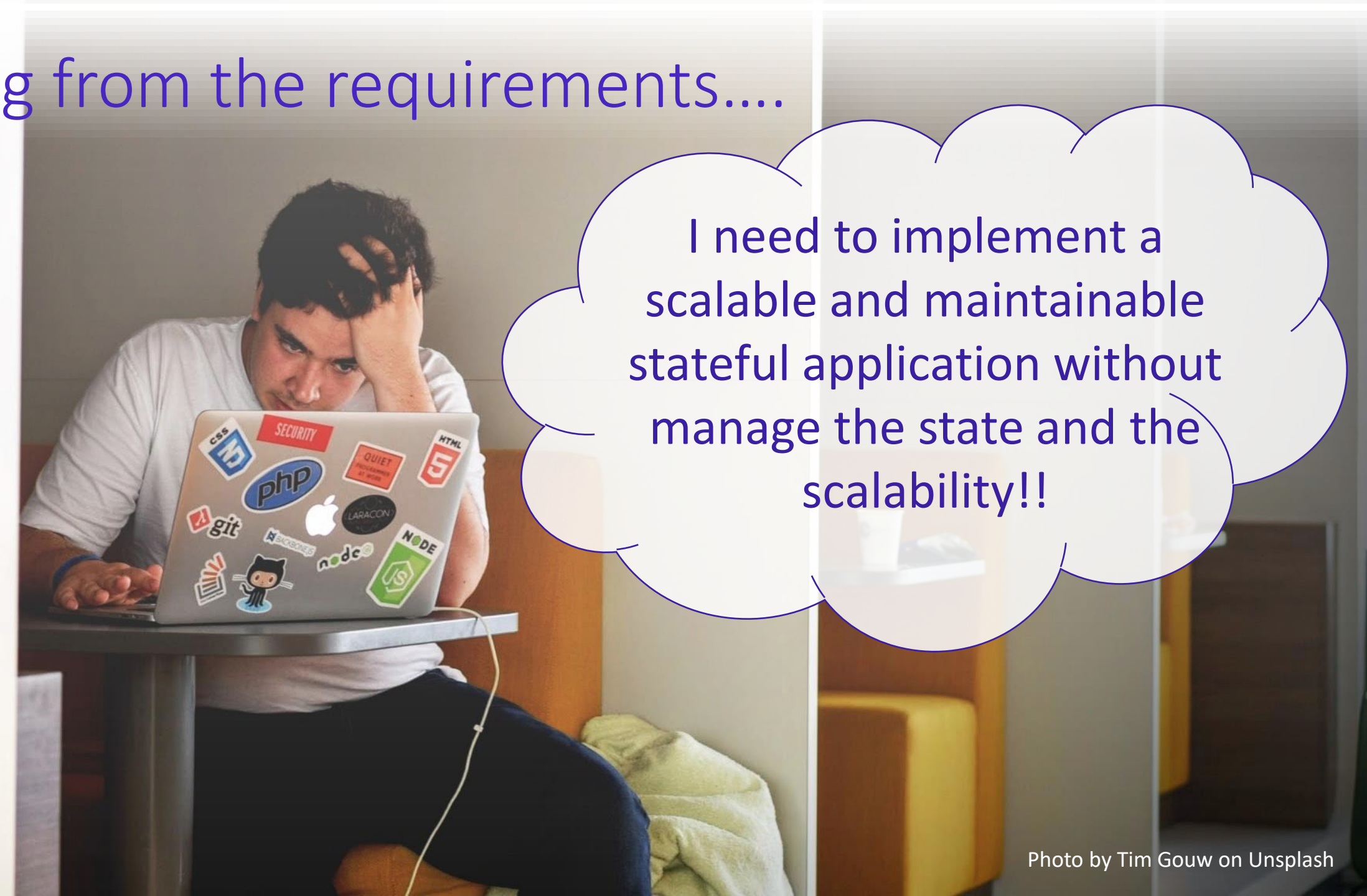

Packt>


PLURALSIGHT


JET
BRAINS


NET
CODE

Starting from the requirements….



I need to implement a scalable and maintainable stateful application without manage the state and the scalability!!

Photo by Tim Gouw on Unsplash

What you are looking for….

I want:

- ✓ A runtime to run my .NET Core code
- ✓ A state management layer
- ✓ A platform that gives me the scalability by design

Photo by Tim Gouw on Unsplash

Foto Robin Higgins da Pixabay

What are Azure Functions?

Events

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

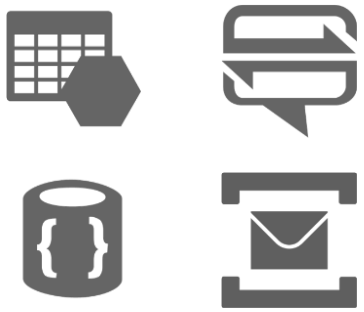
Trigger

Code

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

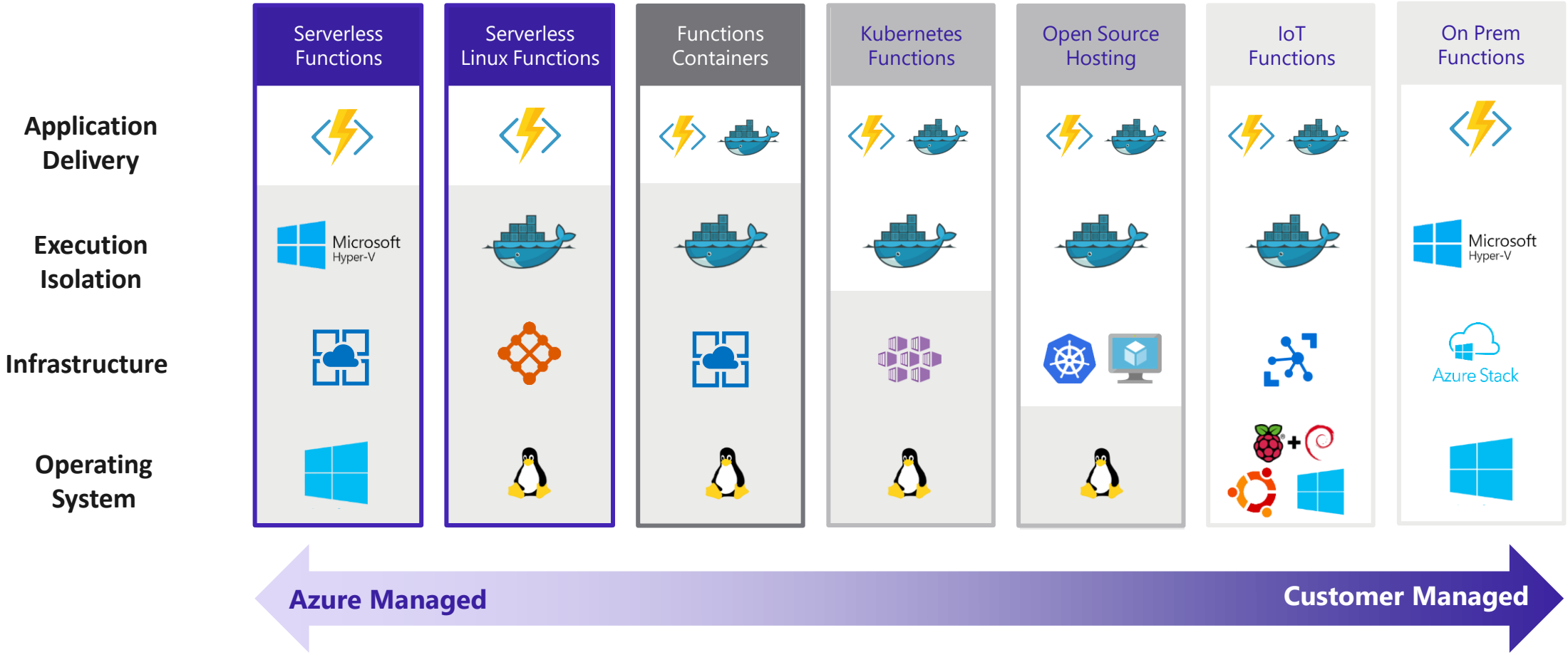
Bindings

Outputs



Send results to an ever-growing collection of services

Functions Hosting Models



What are Durable Functions?

Azure Functions Extension

Based on Azure Functions

Adds new Triggers and Bindings

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# (.NET Core)

JavaScript

F#

Durable Function components

Client

Is the triggered functions that will create new instances of an orchestration.

It is the entry point for creating an instance of a durable orchestration

Orchestrator

Is the heart of a durable function.

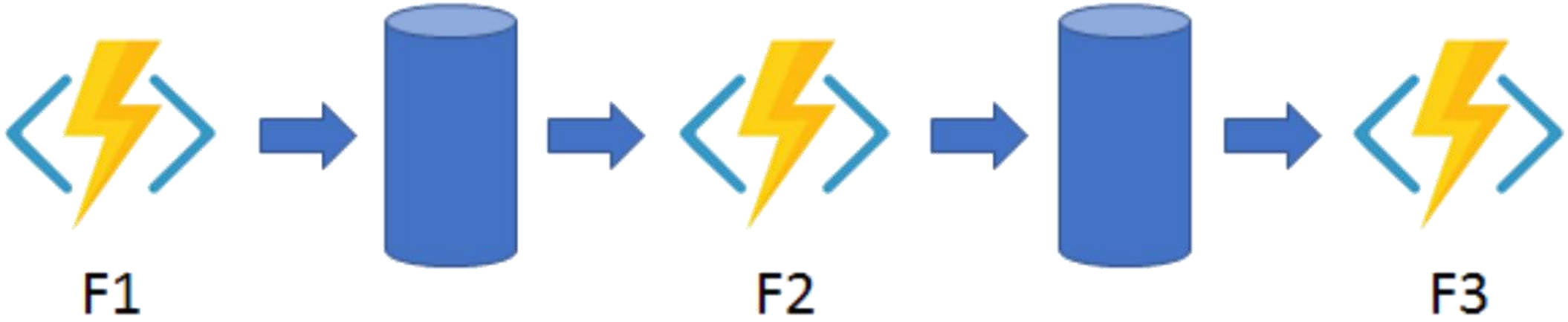
Orchestrator functions describe the way and order actions are executed.

Activity

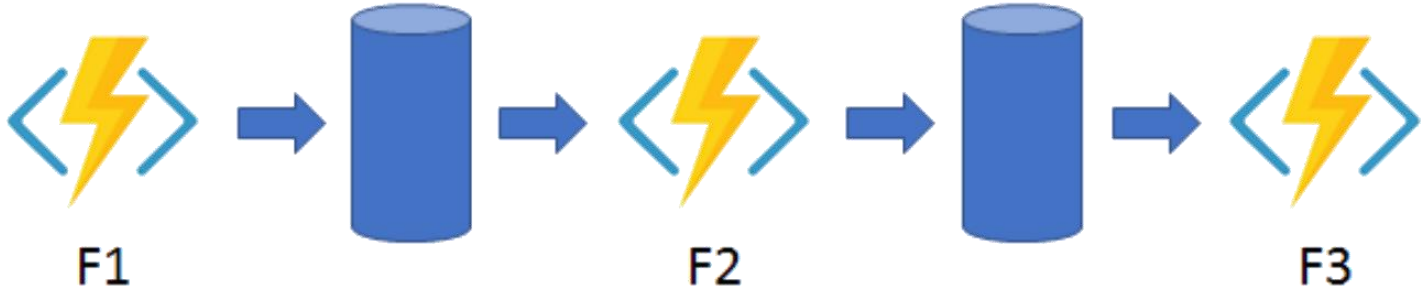
Is the basic unit of work in a durable orchestration.

An activity function must be triggered by an activity trigger.

Function chaining



Function chaining



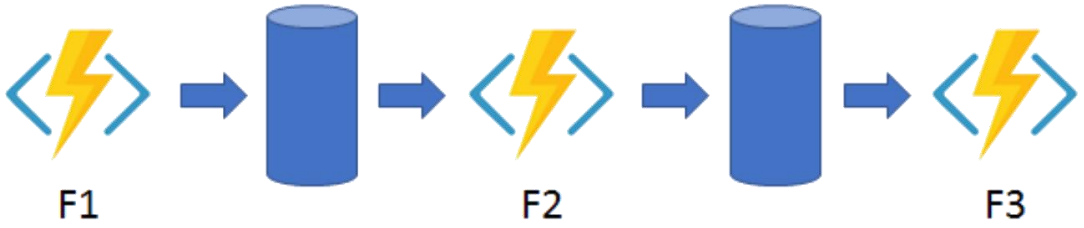
- Relations between functions and queues aren’t clearly identifying
- Queues are an implementation detail
- Operation context management is difficult
- Error handling is difficult

Function chaining in Durable Functions

Orchestrator Function

```
[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;
}
```

Activity Functions



Function chaining in Durable Functions

```
[FunctionName("FunctionsChainingOrchestrator")]
public static async Task<int> Orchestrator([OrchestrationTrigger] IDurableOrchestrationC
{
    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;
}
```

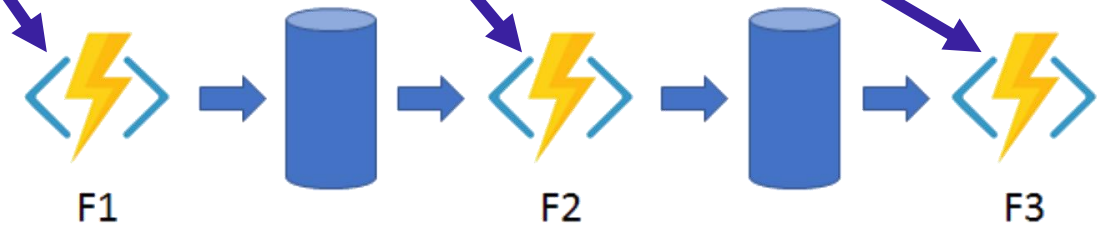
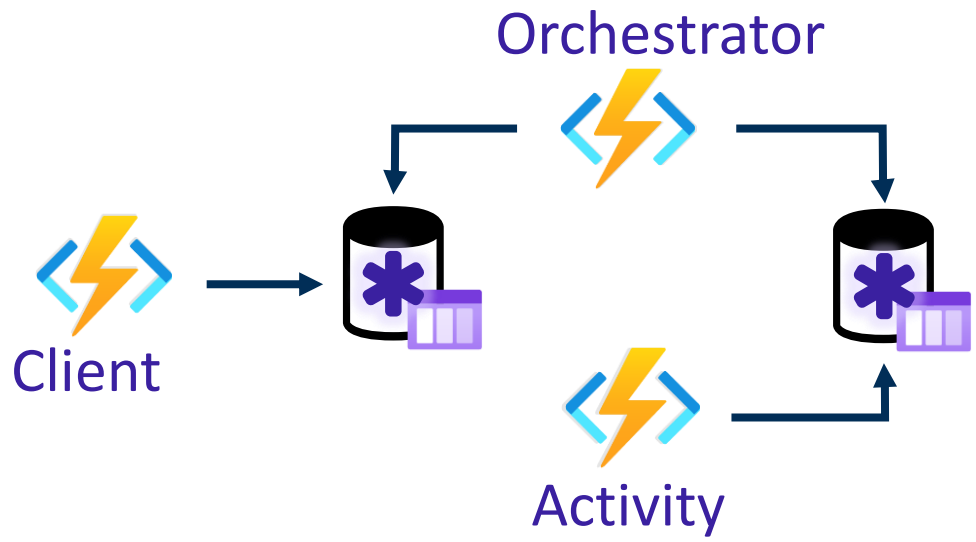


Foto Robin Higgins da Pixabay

Orchestrator Function

```
1. var x = await context.CallActivityAsync<int>("F1", null);
2. var y = await context.CallActivityAsync<int>("F2", x);
3. 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

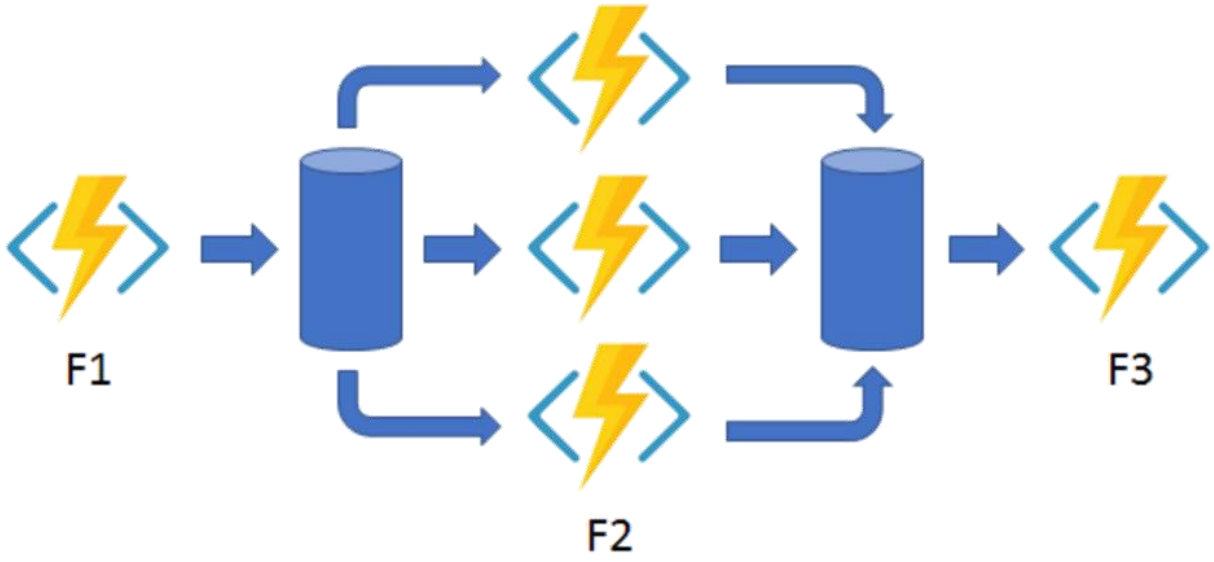
DEMO

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

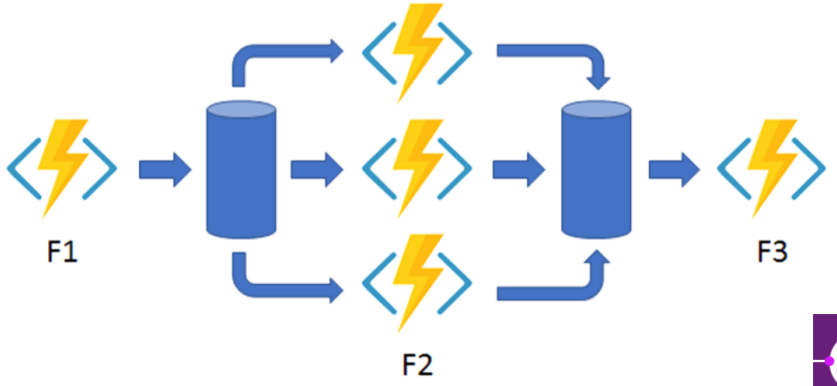
FanIn-FanOut



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

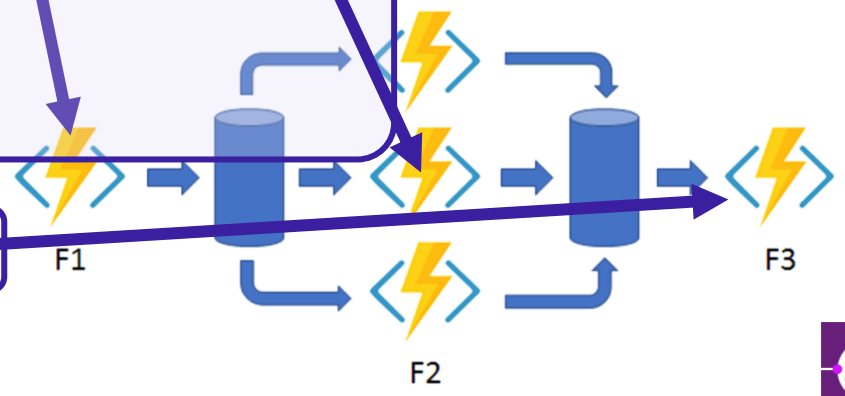
FanIn-FanOut in Durable Functions

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

    var workBatch = await context.CallActivityAsync<int[]>("F1", null);

    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);
}
```



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<int[]>("F1", null);

    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);
}
```



Human interaction

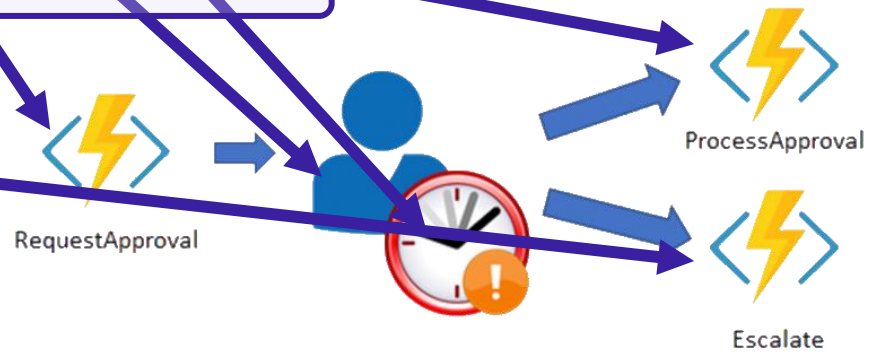
- Handling race conditions between timeouts and approval
- Need mechanism for implementing and cancelling timeout events
- Same issues as the other pattern

Human Interaction in Durable Functions

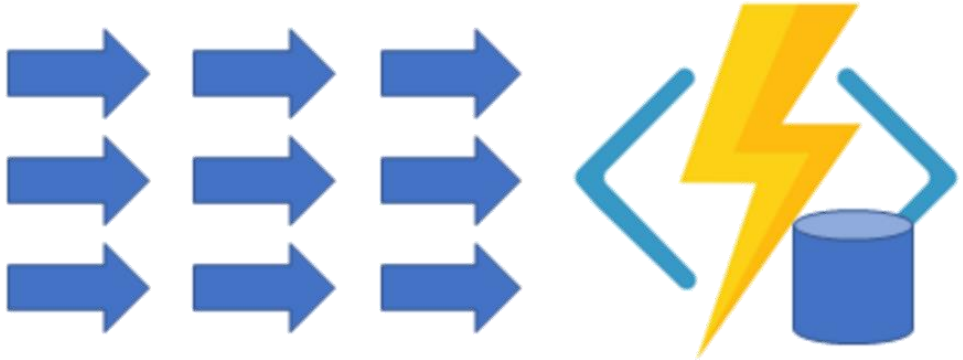
```
[FunctionName("HumanInteractionOrchestrator")]
public static async Task Run([OrchestrationTrigger] IDurableOrchestrationContext context)
{
    await context.CallActivityAsync("RequestApproval", null);
    using (var timeoutCts = new CancellationTokenSource())
    {
        DateTime dueTime = context.CurrentUtcDateTime.AddHours(72);
        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);
        }
    }
}
```

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.AddHours(72);
        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);
        }
    }
}
```



Aggregator



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

Foto Robin Higgins da Pixabay

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

Accessing the 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.

Orchestrator

Signaling

One-way (**fire and forget**) communication.
You send an operation message but don't wait for a response.

Orchestrator
Client
Entity

State

Two-way communication.
You can retrieve the state of an entity

Client

Anatomy of an Entity

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

Properties (state)

Operations

Entry Function

Takeaways

- State persistence abstraction
- Workflow by code
- Asynchronous by design
- Scalable by design

Thanks for your attention!!!!

Massimo Bonanni

Azure Technical Trainer

massimo.bonanni@microsoft.com
@massimobonanni
linkedin.com/in/massimobonanni/

bit.ly/MasteringServerless

References

- Azure Functions Documentation
  https://docs.microsoft.com/en-US/azure/azure-functions/
- 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

Photo by cottonbro from Pexels