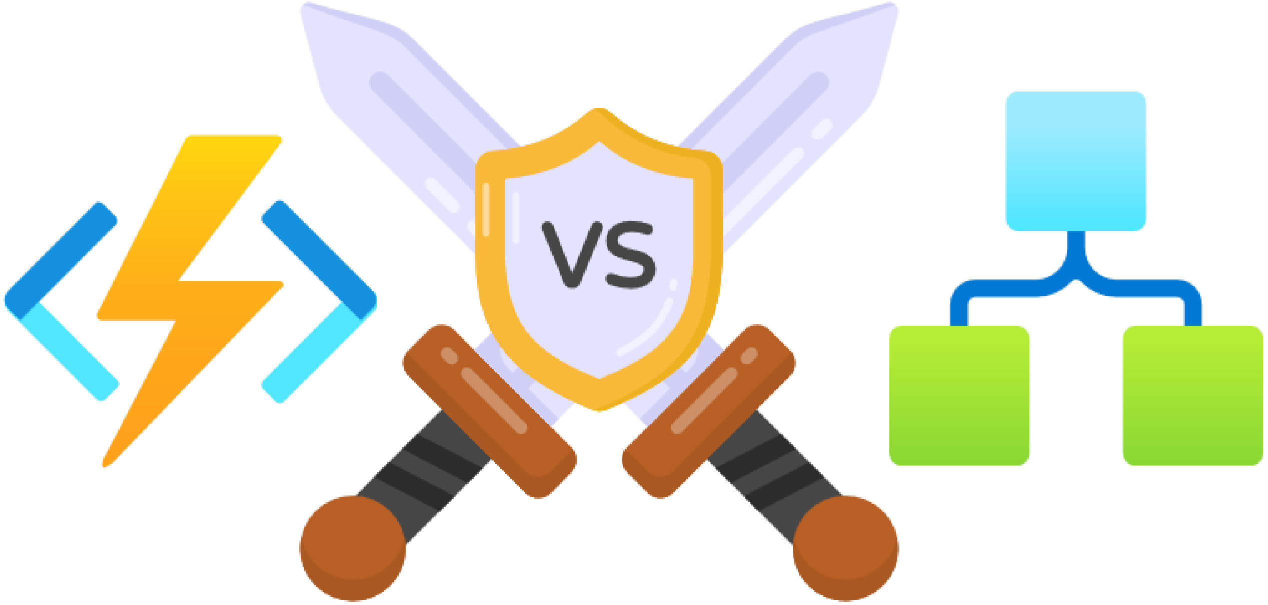
#PitchOnline presents:

Durable Functions vs Logic App
la guerra dei workflow!!



Guest Speaker

Massimo Bonanni
Technical Speaker & Azure Technical Trainer
Microsoft

#Coding

Scifoni Ivano

Fabio Mannis

Francesco Del Re

Matteo Riccardi

Valerio Benedetti

What is serverless?

Full abstraction of servers

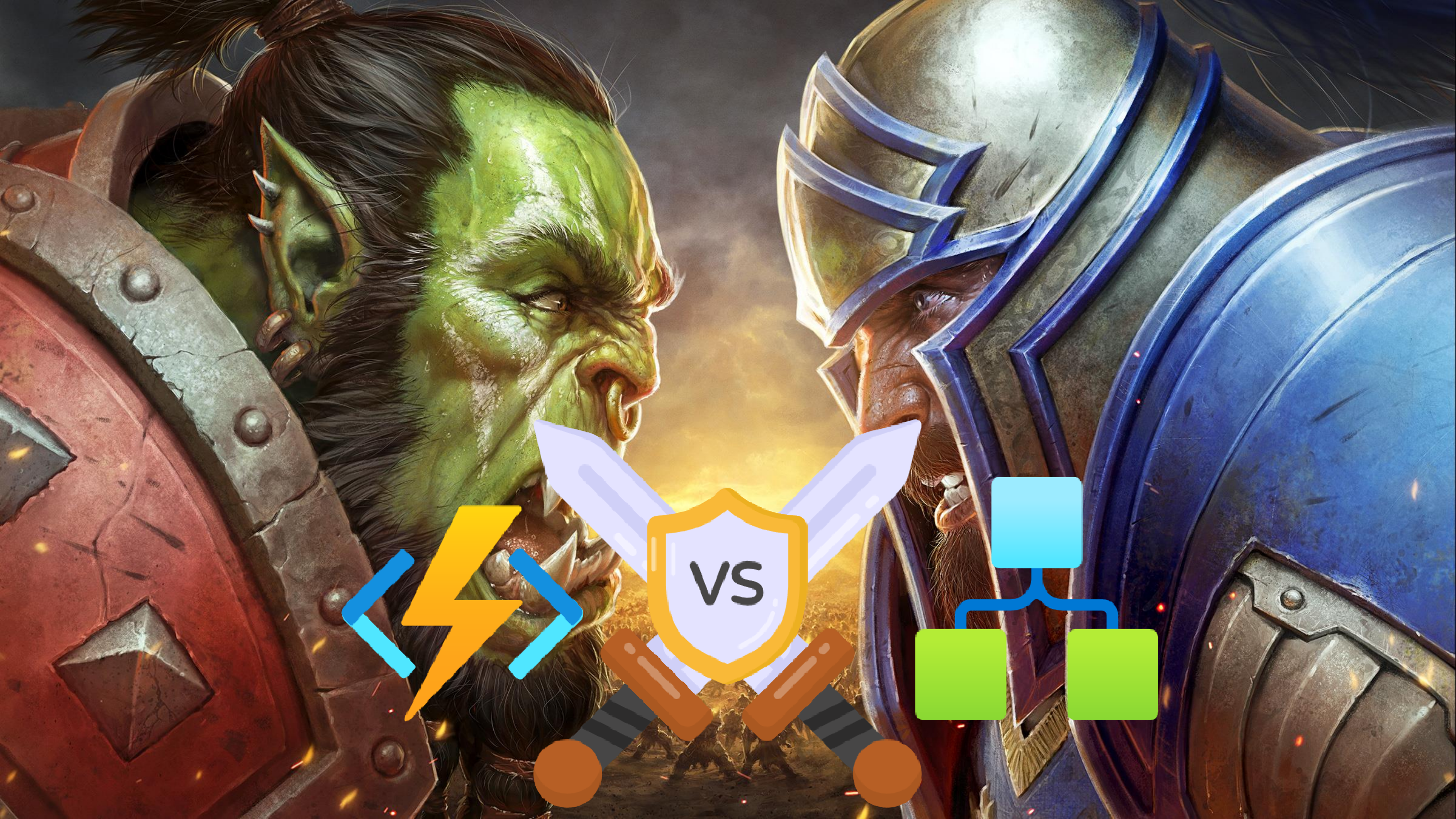
Developers can just focus on their code—there are no distractions around server management, capacity planning, or availability.

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.



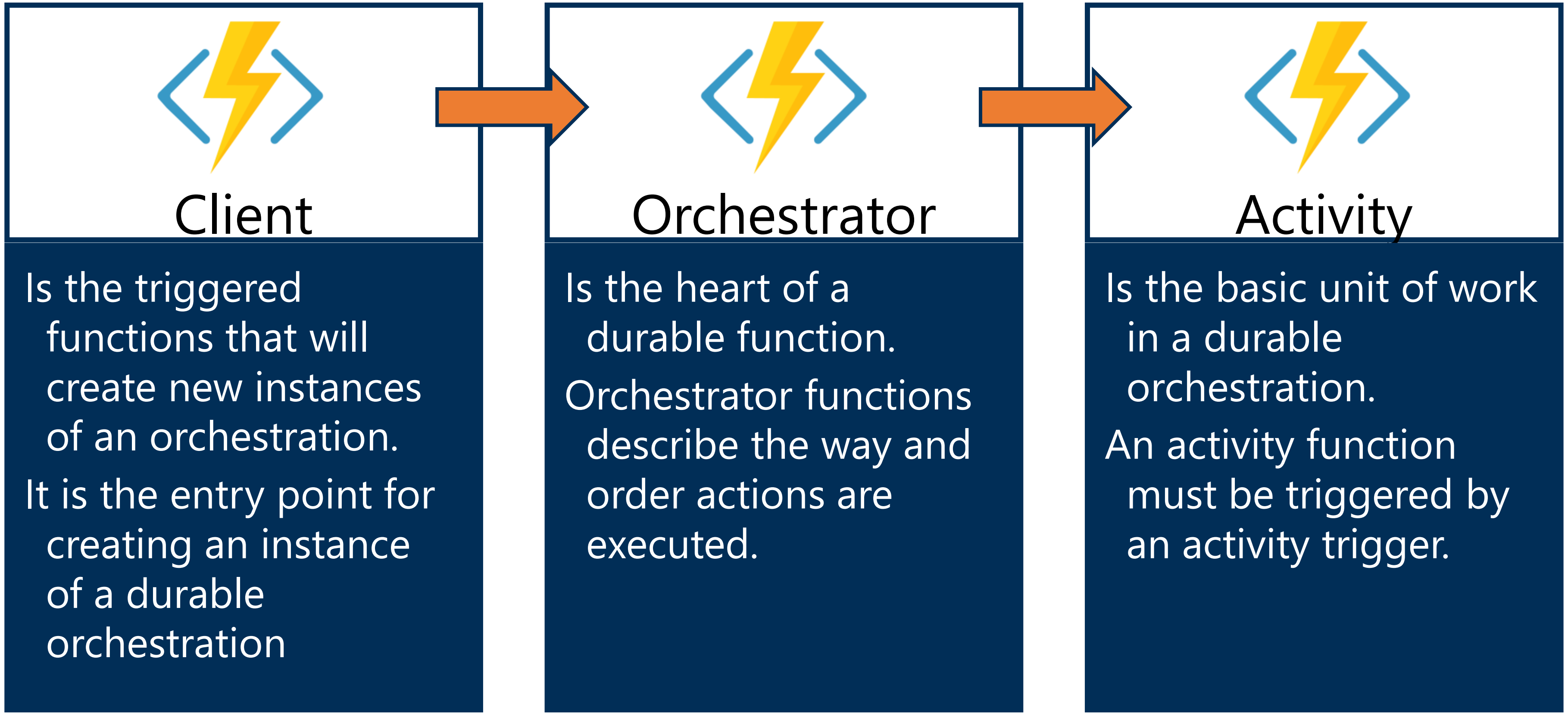
VS

Durable Functions

Durable Functions key points

- An extension of Azure Functions
- Are built on top of the Durable Task Framework
- Written in C#, Node.Js (Javascript), F#, Python and PowerShell
  No JSON, No Designer!!!
- Abstract persistence layer (Azure Storage, SQL Database or Netherite)

Types of functions



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.

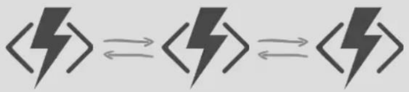
Orchestrator functions describe the way and order actions are executed.

Activity

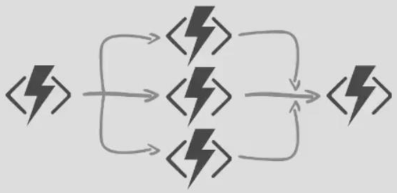
Is the basic unit of work in a durable orchestration.

An activity function must be triggered by an activity trigger.

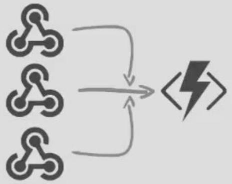
What can you do with Durable Functions?



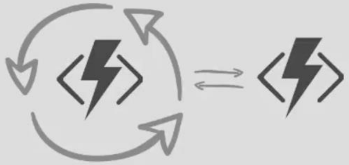
Manageable Sequencing
+ Error Handling / Compensation



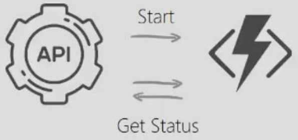
Fanning-out & Fanning-in



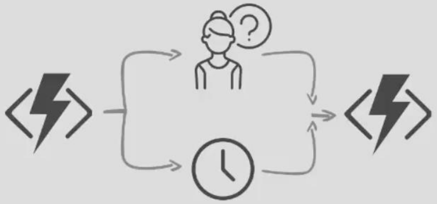
External Events Correlation



Flexible Automated Long-running
Process Monitoring

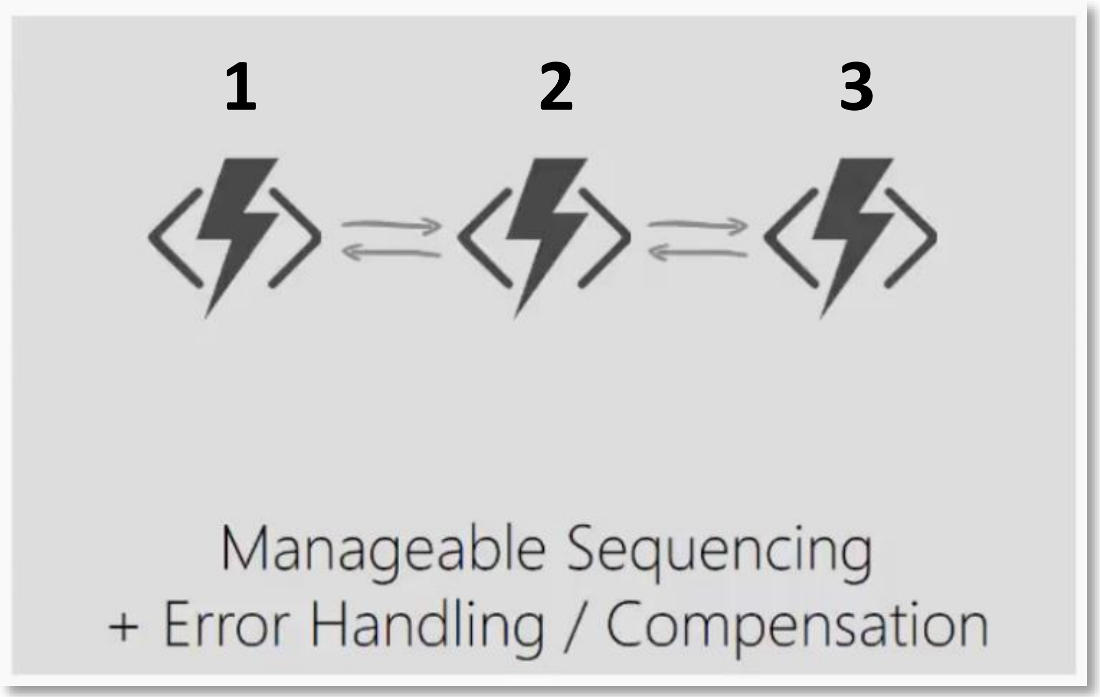


Http-based
Async Long-running APIs



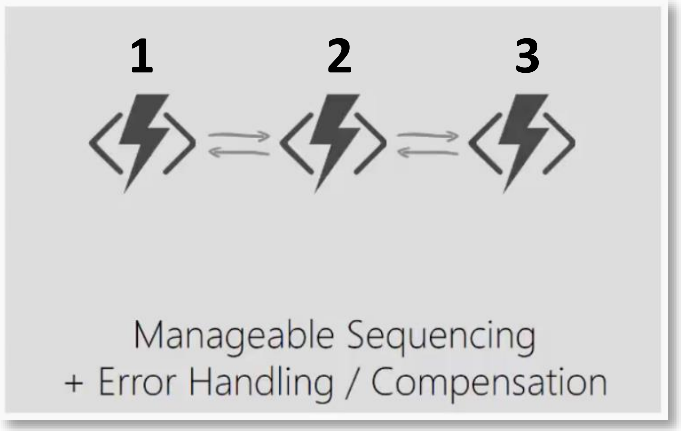
Human Interaction

Manageable Sequencing

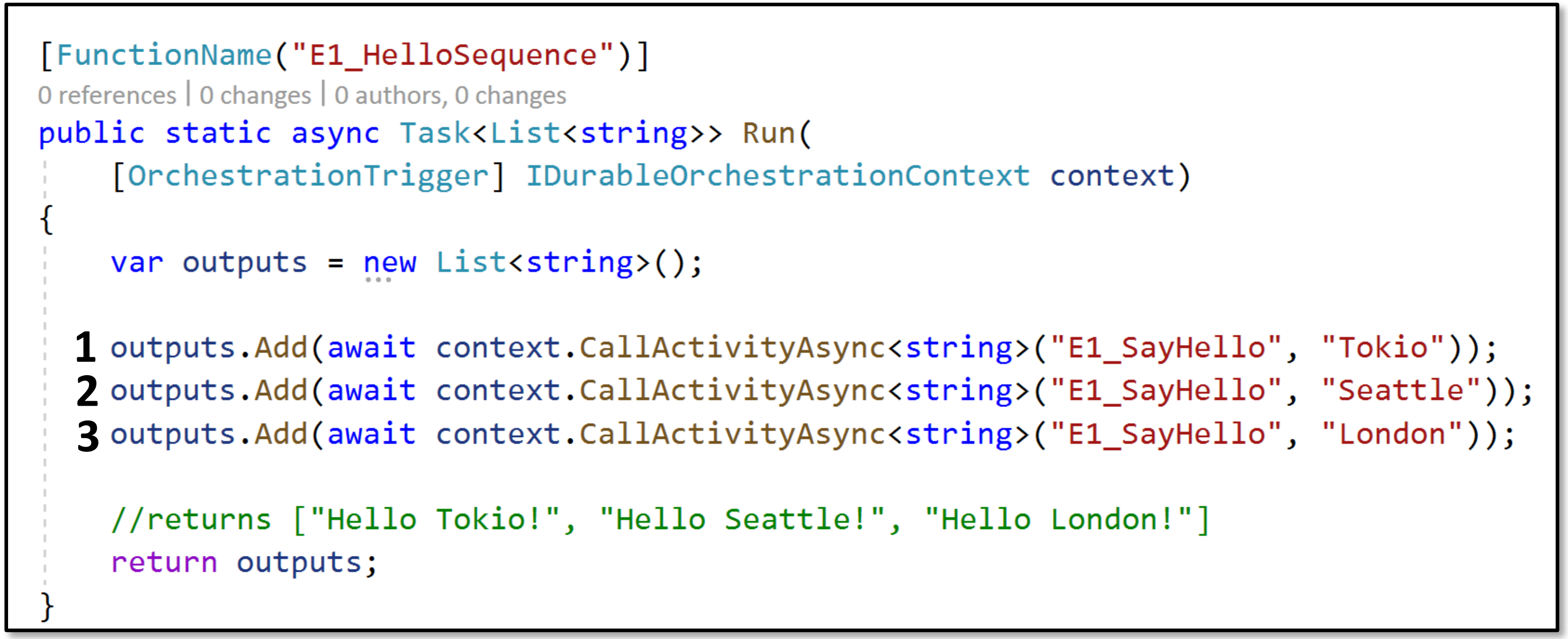


Manageable Sequencing
+ Error Handling / Compensation

Manageable Sequencing

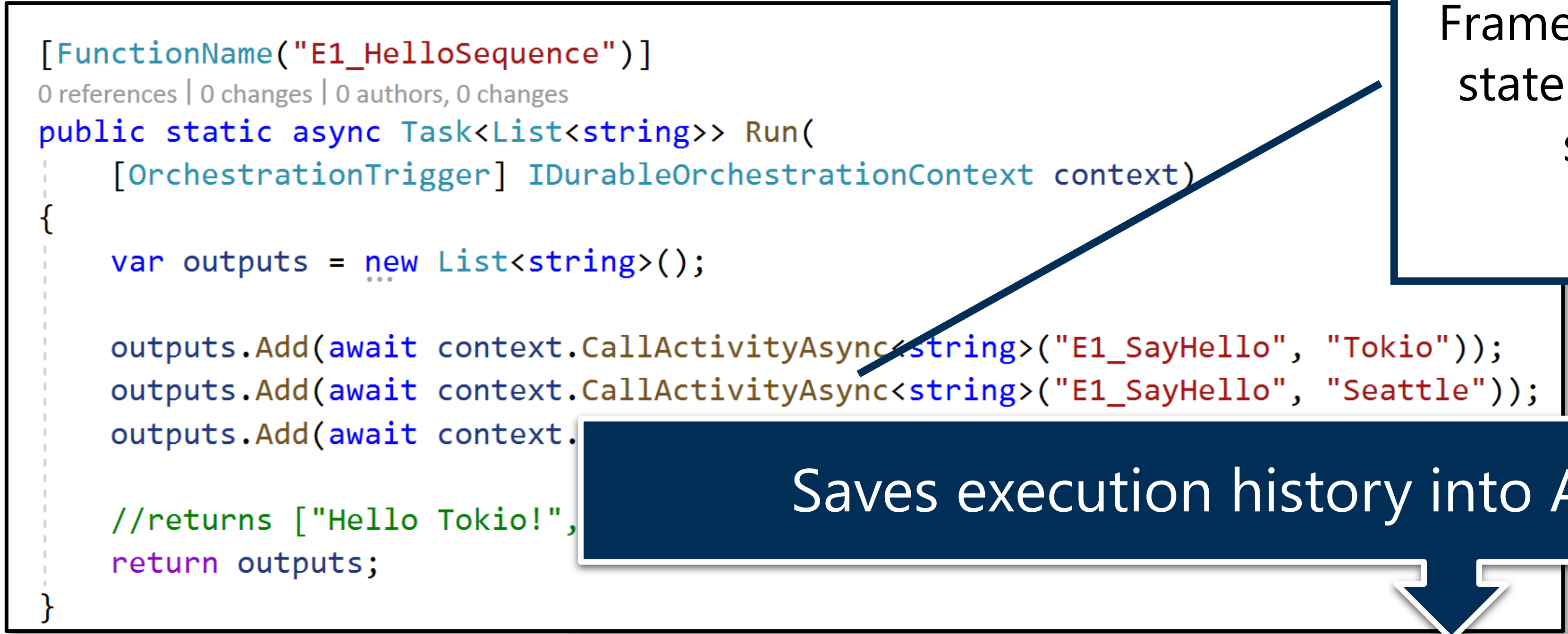


Manageable Sequencing
+ Error Handling / Compensation



```
[FunctionName("E1_HelloSequence")]
0 references | 0 changes | 0 authors, 0 changes
public static async Task<List<string>> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var outputs = new List<string>();

  1 outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Tokio"));
  2 outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Seattle"));
  3 outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "London"));

    //returns ["Hello Tokio!", "Hello Seattle!", "Hello London!"]
    return outputs;
}
```

Orchestration history



```
[FunctionName("E1_HelloSequence")]
0 references | 0 changes | 0 authors, 0 changes
public static async Task<List<string>> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var outputs = new List<string>();

    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Tokio"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Seattle"));
    outputs.Add(await context.

    //returns ["Hello Tokio!",
    return outputs;
}
```

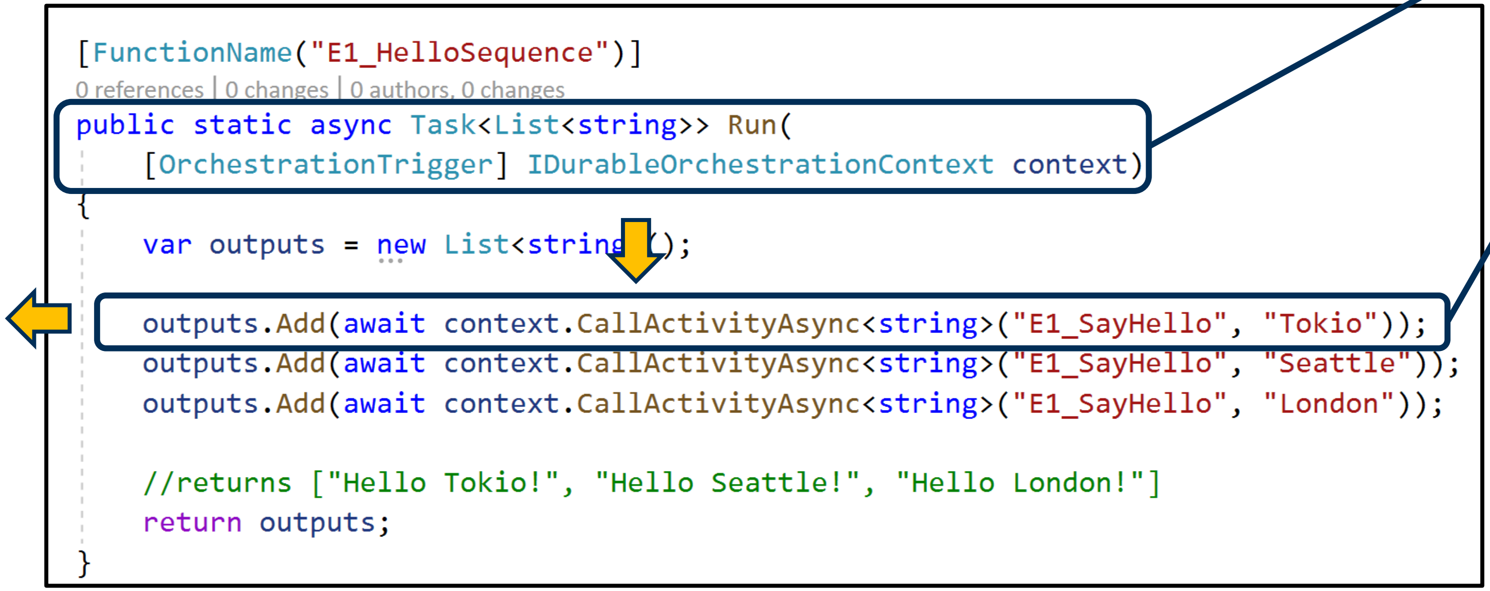
At each activity call, the Durable Task Framework checkpoints the execution state of the function into underlying storage. This state is called “orchestration history”.

Saves execution history into Azure Storage tables.

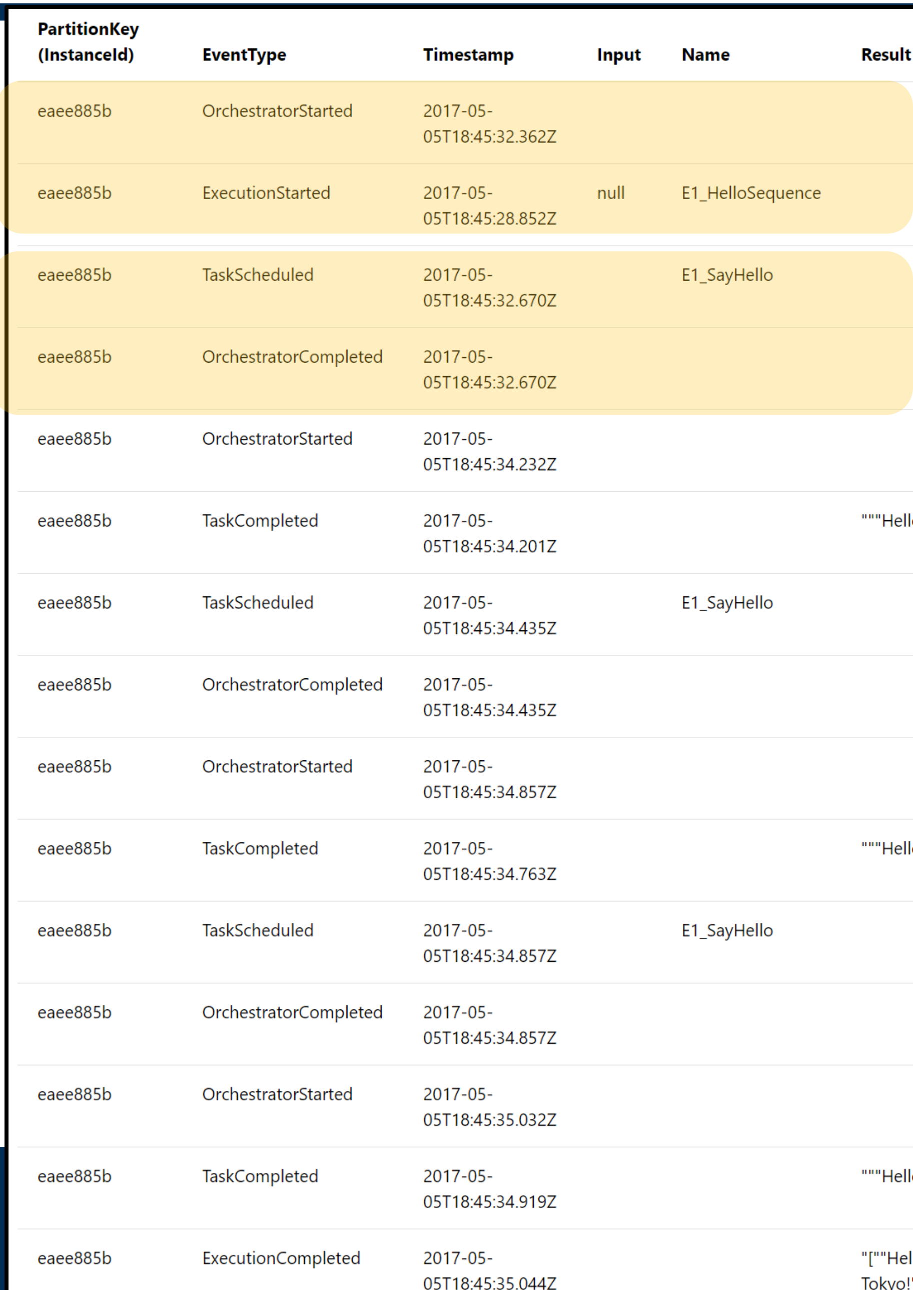
Enqueues messages to run the functions the orchestrator wants to invoke.

Enqueues messages for the orchestrator itself — for example, durable timer messages.

Orchestration History



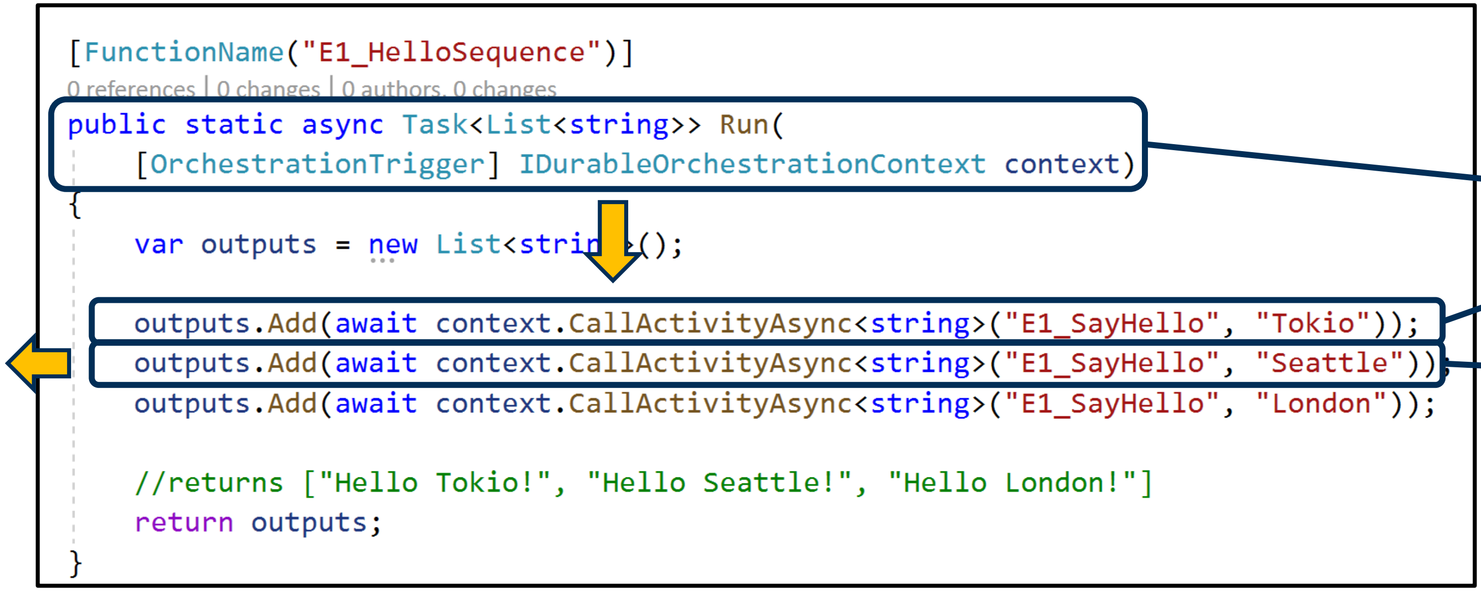
```
[FunctionName("E1_HelloSequence")]
0 references | 0 changes | 0 authors, 0 changes
public static async Task<List<string>> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var outputs = new List<string>();

    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Tokio"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Seattle"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "London"));

    //returns ["Hello Tokio!", "Hello Seattle!", "Hello London!"]
    return outputs;
}
```



PartitionKey (InstanceId)	EventType	Timestamp	Input	Name	Result
eaee885b	OrchestratorStarted	2017-05-05T18:45:32.362Z			
eaee885b	ExecutionStarted	2017-05-05T18:45:28.852Z	null	E1_HelloSequence	
eaee885b	TaskScheduled	2017-05-05T18:45:32.670Z		E1_SayHello	
eaee885b	OrchestratorCompleted	2017-05-05T18:45:32.670Z			
eaee885b	OrchestratorStarted	2017-05-05T18:45:34.232Z			
eaee885b	TaskCompleted	2017-05-05T18:45:34.201Z			"""Hell
eaee885b	TaskScheduled	2017-05-05T18:45:34.435Z		E1_SayHello	
eaee885b	OrchestratorCompleted	2017-05-05T18:45:34.435Z			
eaee885b	OrchestratorStarted	2017-05-05T18:45:34.857Z			
eaee885b	TaskCompleted	2017-05-05T18:45:34.763Z			"""Hell
eaee885b	TaskScheduled	2017-05-05T18:45:34.857Z		E1_SayHello	
eaee885b	OrchestratorCompleted	2017-05-05T18:45:34.857Z			
eaee885b	OrchestratorStarted	2017-05-05T18:45:35.032Z			
eaee885b	TaskCompleted	2017-05-05T18:45:34.919Z			"""Hell
eaee885b	ExecutionCompleted	2017-05-05T18:45:35.044Z			"[""Hel Tokyo!

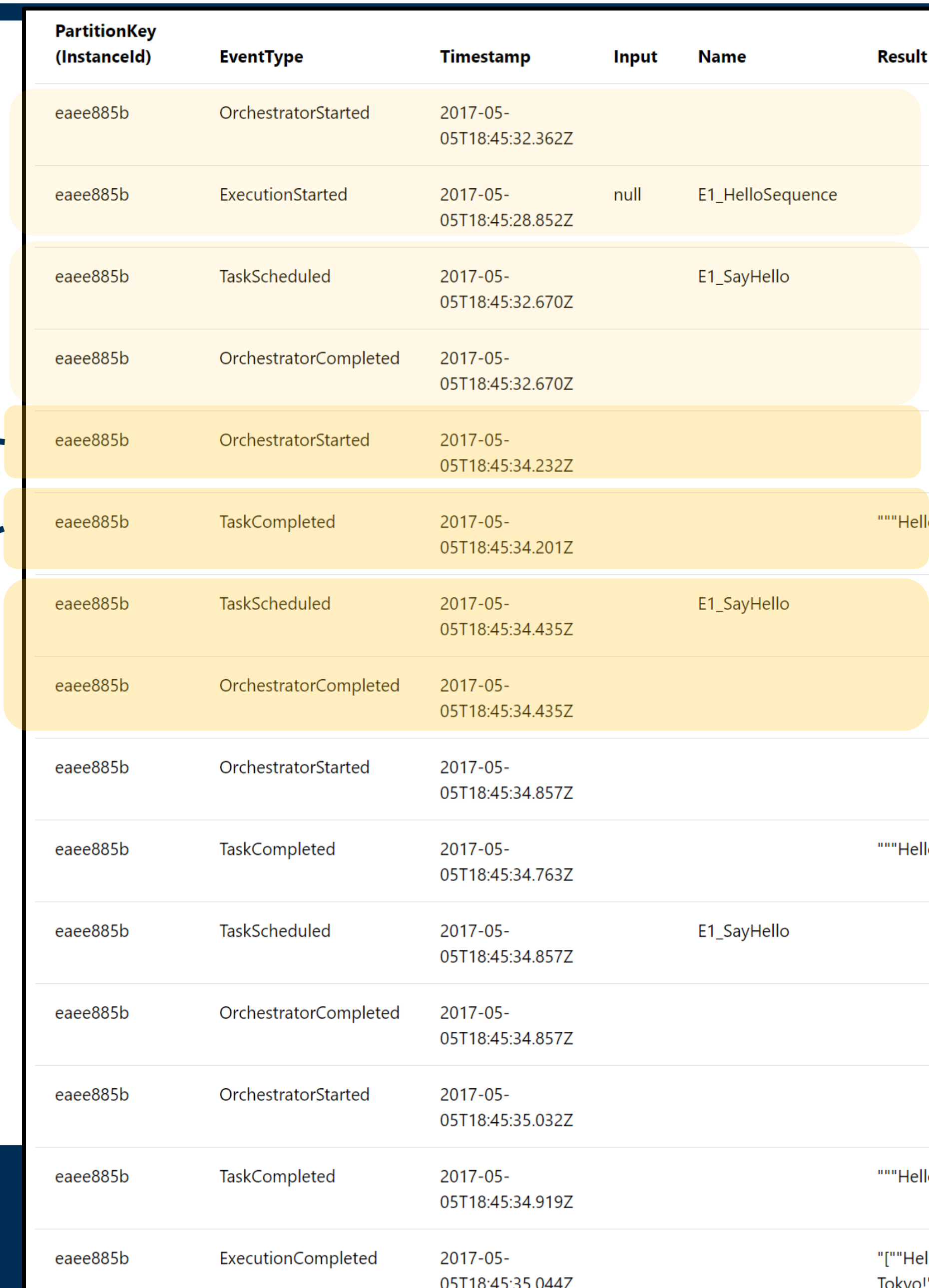
The orchestrator function starts the activity and wait for its completion.

Orchestration History



```
[FunctionName("E1_HelloSequence")]
0 references | 0 changes | 0 authors, 0 changes
public static async Task<List<string>> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var outputs = new List<string>();

    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Tokio"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Seattle"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "London"));

    //returns ["Hello Tokio!", "Hello Seattle!", "Hello London!"]
    return outputs;
}
```

When the activity completes its job, the orchestrator starts again from the begin and rebuilds its status using event sourcing table.

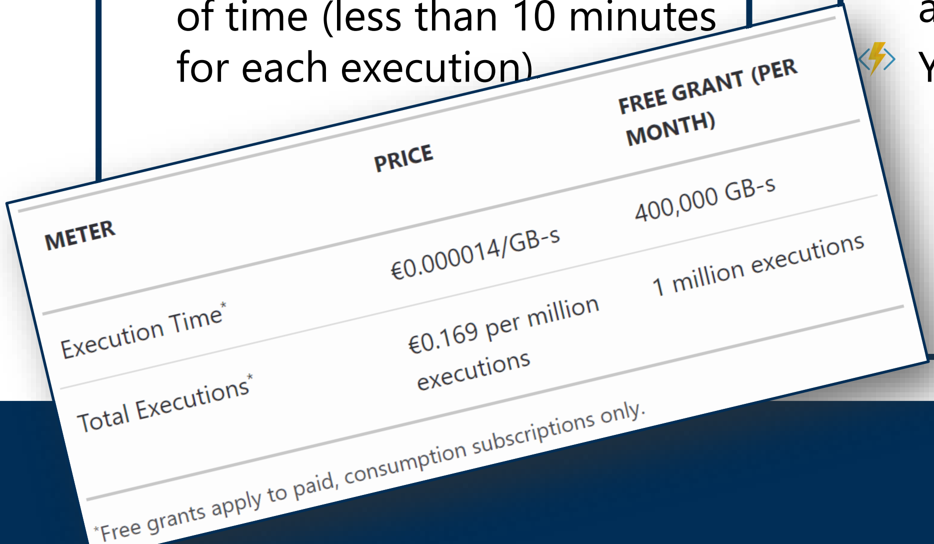


PartitionKey (InstanceId)	EventType	Timestamp	Input	Name	Result
eaee885b	OrchestratorStarted	2017-05-05T18:45:32.362Z			
eaee885b	ExecutionStarted	2017-05-05T18:45:28.852Z	null	E1_HelloSequence	
eaee885b	TaskScheduled	2017-05-05T18:45:32.670Z		E1_SayHello	
eaee885b	OrchestratorCompleted	2017-05-05T18:45:32.670Z			
eaee885b	OrchestratorStarted	2017-05-05T18:45:34.232Z			
eaee885b	TaskCompleted	2017-05-05T18:45:34.201Z			"""Hell
eaee885b	TaskScheduled	2017-05-05T18:45:34.435Z		E1_SayHello	
eaee885b	OrchestratorCompleted	2017-05-05T18:45:34.435Z			
eaee885b	OrchestratorStarted	2017-05-05T18:45:34.857Z			
eaee885b	TaskCompleted	2017-05-05T18:45:34.763Z			"""Hell
eaee885b	TaskScheduled	2017-05-05T18:45:34.857Z		E1_SayHello	
eaee885b	OrchestratorCompleted	2017-05-05T18:45:34.857Z			
eaee885b	OrchestratorStarted	2017-05-05T18:45:35.032Z			
eaee885b	TaskCompleted	2017-05-05T18:45:34.919Z			"""Hell
eaee885b	ExecutionCompleted	2017-05-05T18:45:35.044Z			"[""Hel Tokyo!"

Pricing

Consumption Plan

- Pay only when your functions run.
- Scale out automatically, even during periods of high load.
- Function execution times out after a configurable period of time (less than 10 minutes for each execution)



METER	PRICE	FREE GRANT (PER MONTH)
Execution Time*	€0.000014/GB-s	400,000 GB-s
Total Executions*	€0.169 per million executions	1 million executions

*Free grants apply to paid, consumption subscriptions only.

App Sevice Plan

- You won't pay more than the cost of the VM instance that you allocate.
- You can manually scale out by adding more VM instances, or you can enable autoscale.
- You must enable AlwaysOn.

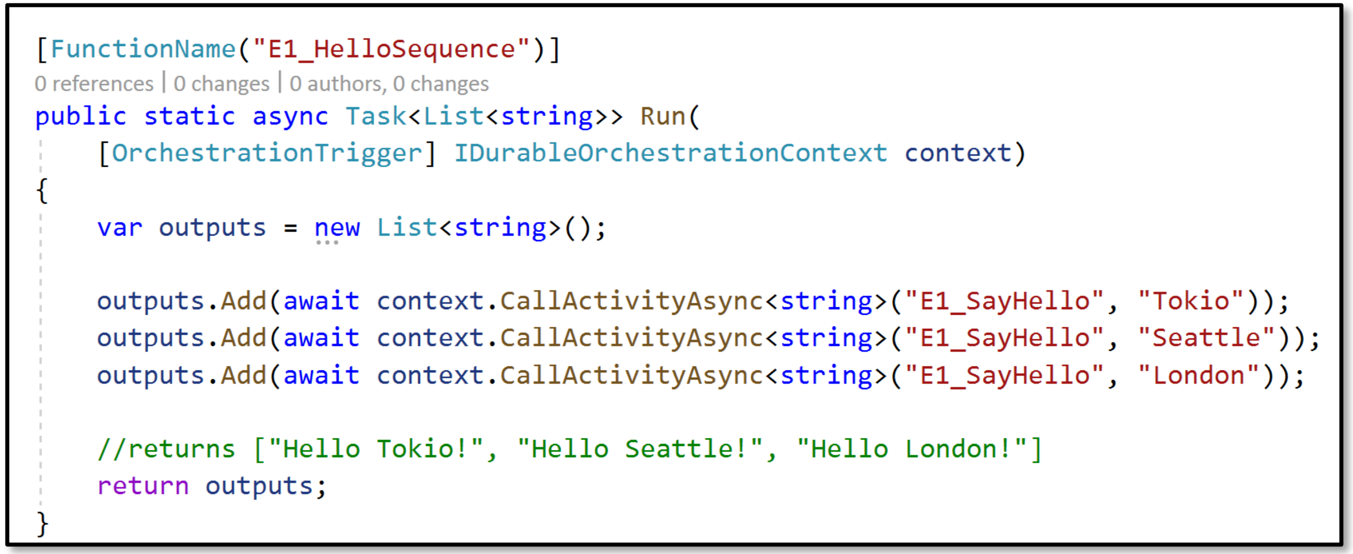
Premium Plan

- Perpetually warm instances to avoid any cold start.
- VNet connectivity.
- Unlimited execution duration.
- Premium instance sizes (one core, two core, and four core instances).
- More predictable pricing
- High-density app allocation for plans with multiple function apps

Pricing sample – Manageable Sequencing

- Client Function:
  - 512 Mb, 100 msec
  - 1 execution for each workflow
- Orchestrator Function:
  - 512 Mb, 100 msec
  - 4 executions for each workflow
- Activity Function:
  - 512 Mb, 100 msec
  - 3 executions for each workflow

- Monthly Requests : 5.000.000



```
[FunctionName("E1_HelloSequence")]
0 references | 0 changes | 0 authors, 0 changes
public static async Task<List<string>> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var outputs = new List<string>();

    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Tokio"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Seattle"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "London"));

    //returns ["Hello Tokio!", "Hello Seattle!", "Hello London!"]
    return outputs;
}
```

Monthly executions : 40.000.000

Pricing sample – Manageable Sequencing

Resource cost

- Seconds:

  40.000.000 exec * 0,1 sec = 4.000.000 secs
- GB*Seconds:

  512 GB/1024 GB * 4.000.000 secs = 2.000.000 GB*secs
- GB*secs to pay:

  2.000.000 GB*secs – 400.000 GB*secs = 1.600.000 GB*secs
- GB*secs cost:

  1.600.000 GB*secs * 0,000014 € = 22,400 €

Resource Cost: 22,400 €

Pricing sample – Manageable Sequencing

Execution cost

- Executions to pay:

  40.000.000 exec – 1.000.000 exec = 39.000.000 exec

- Executions cost:

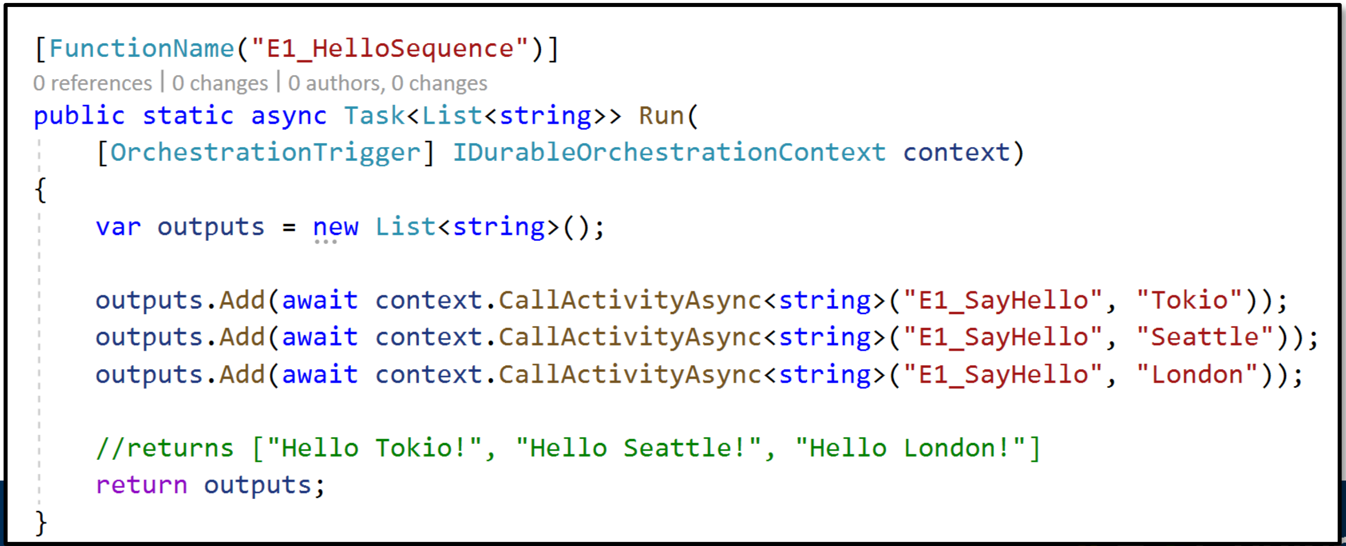
  39.000.000/1.000.000 * 0,169 € = 6,591 €

Executions Cost: 6,591 €

Pricing sample – Manageable Sequencing

Monthly cost

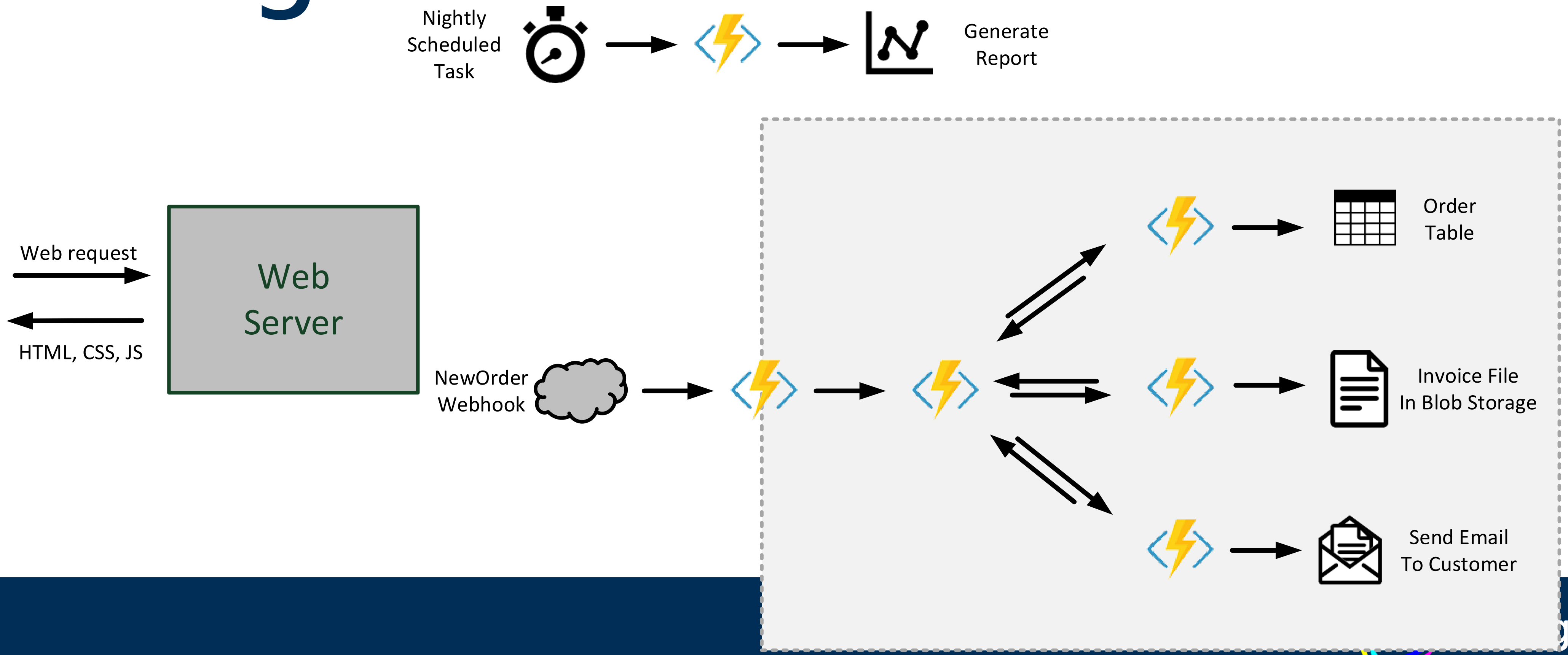
22,400 + 6, 591 = 28,991 €



```
[FunctionName("E1_HelloSequence")]
0 references | 0 changes | 0 authors, 0 changes
public static async Task<List<string>> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var outputs = new List<string>();

    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Tokio"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Seattle"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "London"));

    //returns ["Hello Tokio!", "Hello Seattle!", "Hello London!"]
    return outputs;
}
```

DEMO

Orders Manager



Nightly Scheduled Task

Generate Report

Web request

HTML, CSS, JS

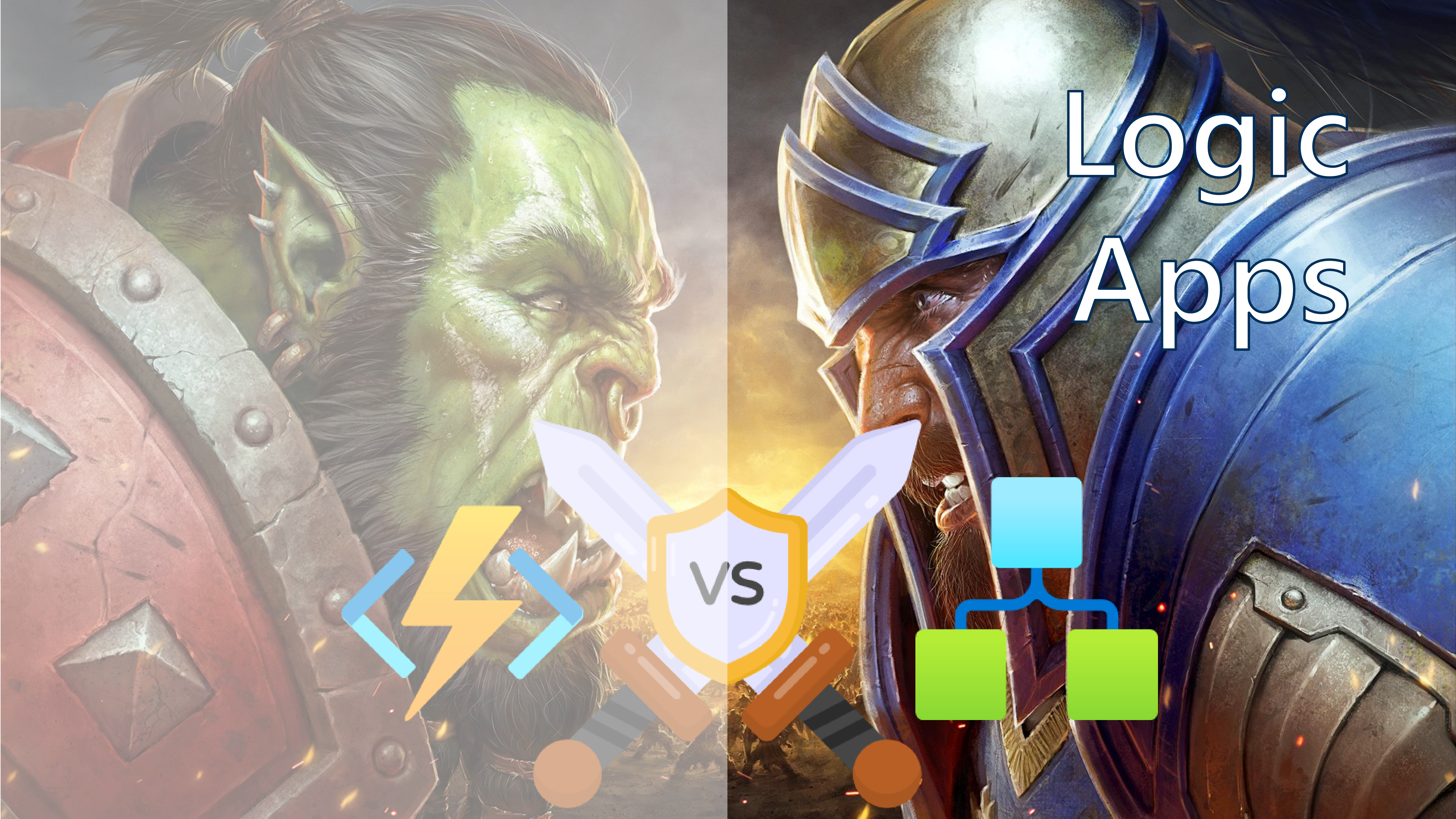
Web Server

NewOrder Webhook

Order Table

Invoice File In Blob Storage

Send Email To Customer


Logic Apps
VS

What are Logic Apps?

Cloud service that helps you automate and orchestrate tasks, business processes, and workflows

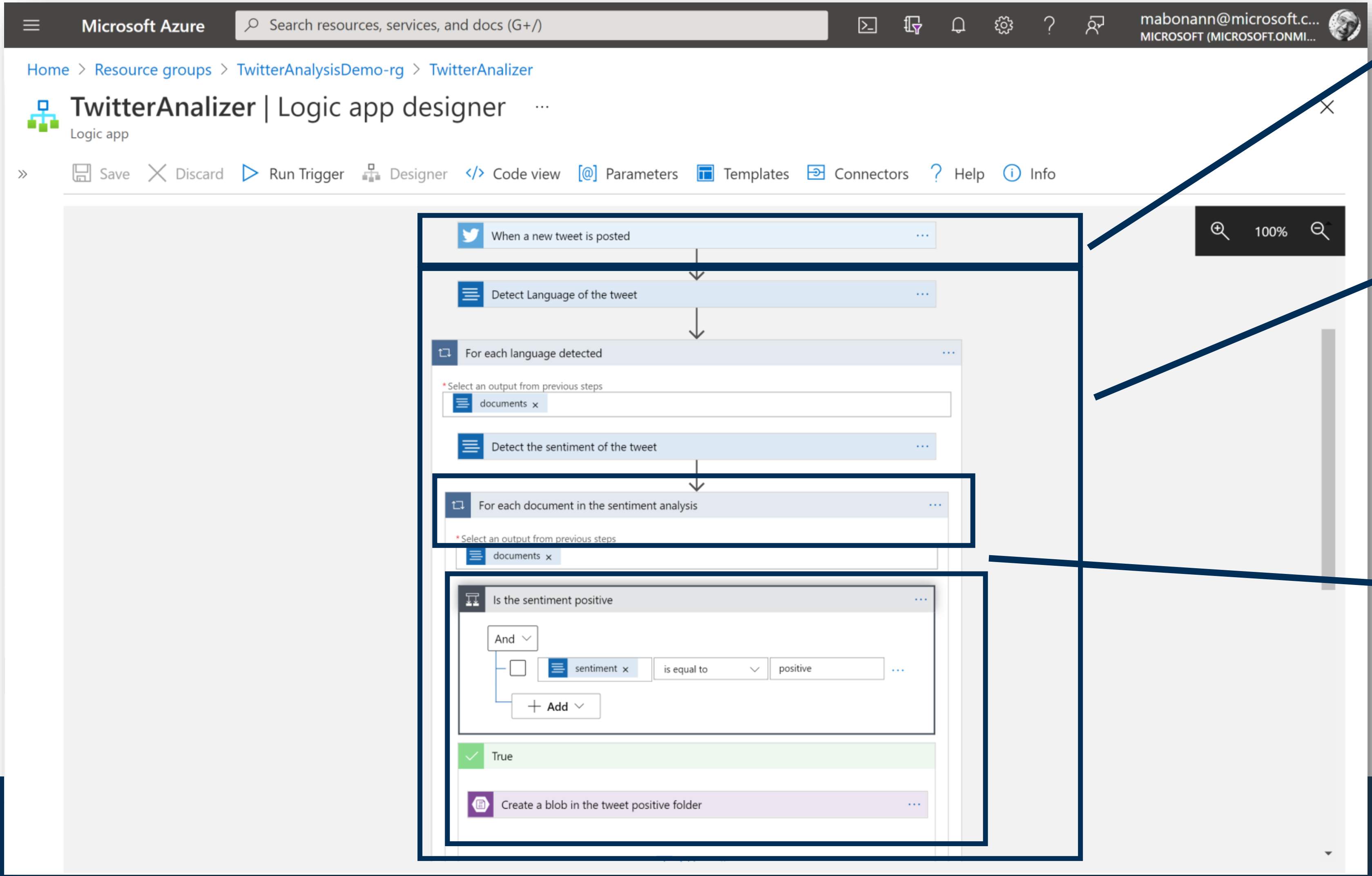
Integrate apps, data, systems, and services across enterprises or organizations

Simplifies how you design and build scalable solutions for integration whether in the cloud, on premises, or both

Sample scenarios

- Process and route orders across on-premises systems and cloud services.
- Send email notifications with Office 365 when events happen in various systems, apps, and services.
- Move uploaded files from an SFTP or FTP server to Azure Storage.
- Monitor tweets for a specific subject, analyze the sentiment, and create alerts or tasks for items that need review.

How does Logic Apps work?



Microsoft Azure
Search resources, services, and docs (G+/)
mabonann@microsoft.c...
MICROSOFT (MICROSOFT.ONMI...

Home > Resource groups > TwitterAnalysisDemo-rg > TwitterAnalizer

TwitterAnalizer | Logic app designer ...

Logic app

When a new tweet is posted

Detect Language of the tweet

For each language detected

*Select an output from previous steps: documents

Detect the sentiment of the tweet

For each document in the sentiment analysis

*Select an output from previous steps: documents

Is the sentiment positive

And: sentiment — is equal to — positive

+ Add

True

Create a blob in the tweet positive folder

Each time that the trigger fires, the Logic Apps engine creates a logic app instance

The Logic Apps instance runs the actions in the workflow

The actions can also include data conversions and flow controls, such as conditional statements, switch statements, loops, and branching.

Why use Logic Apps?

Visually build workflows with easy-to-use tools

Get started faster with logic app templates

Connect disparate systems across different environments

First-class support for enterprise integration and B2B scenarios

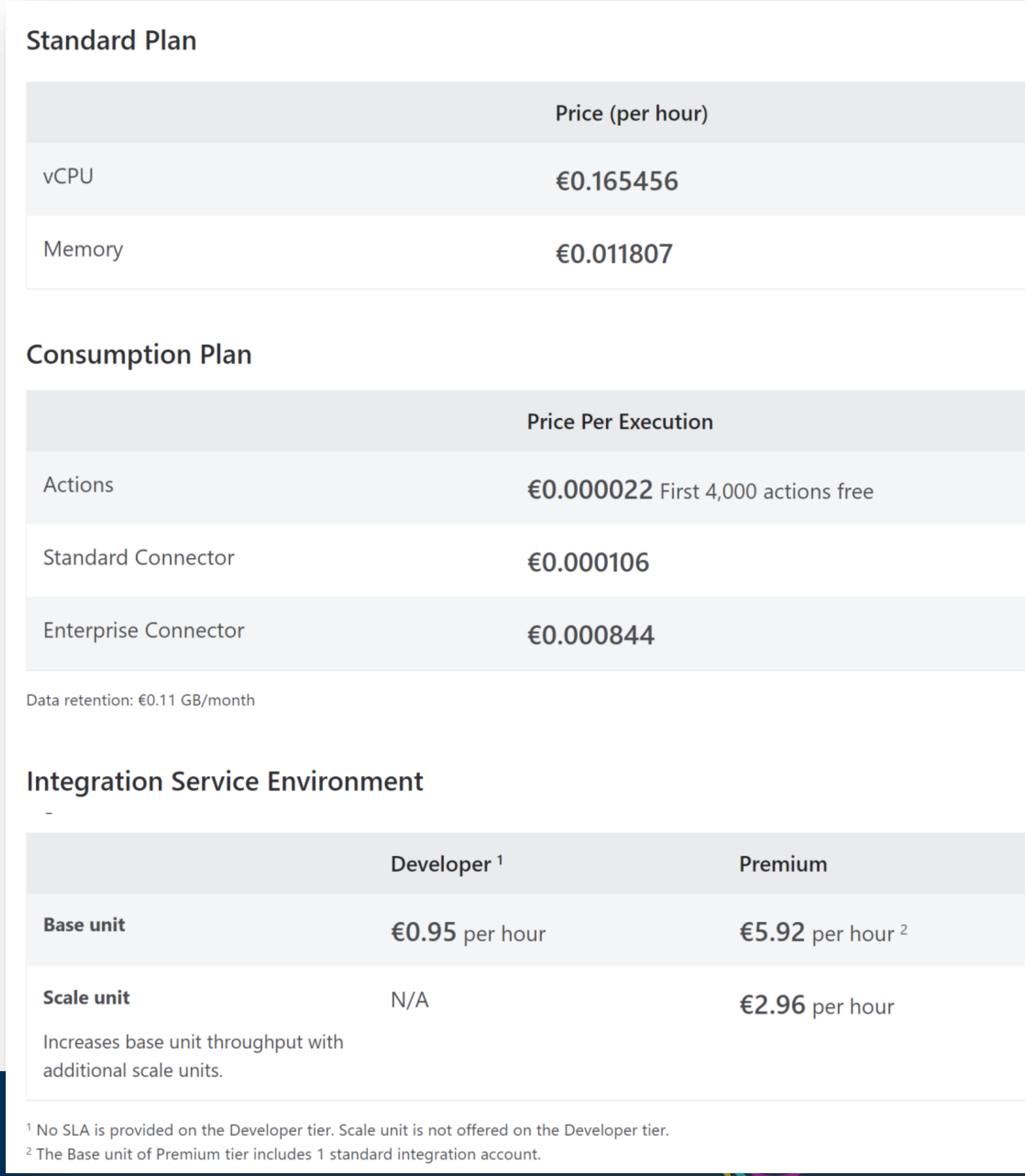
Built-in extensibility

Pay only for what you use

Pricing

Azure Logic Apps meters all the actions that run in your logic app:

- Triggers, which are special actions. All logic apps require a trigger as the first step.
- "Built-in" or native actions such as HTTP, calls to Azure Functions and API Management, and so on
- Calls to connectors such as Outlook 365, Dropbox, and so on
- Control flow steps, such as loops, conditional statements, and so on



Standard Plan

	Price (per hour)
vCPU	€0.165456
Memory	€0.011807

Consumption Plan

	Price Per Execution
Actions	€0.000022 First 4,000 actions free
Standard Connector	€0.000106
Enterprise Connector	€0.000844

Data retention: €0.11 GB/month

Integration Service Environment

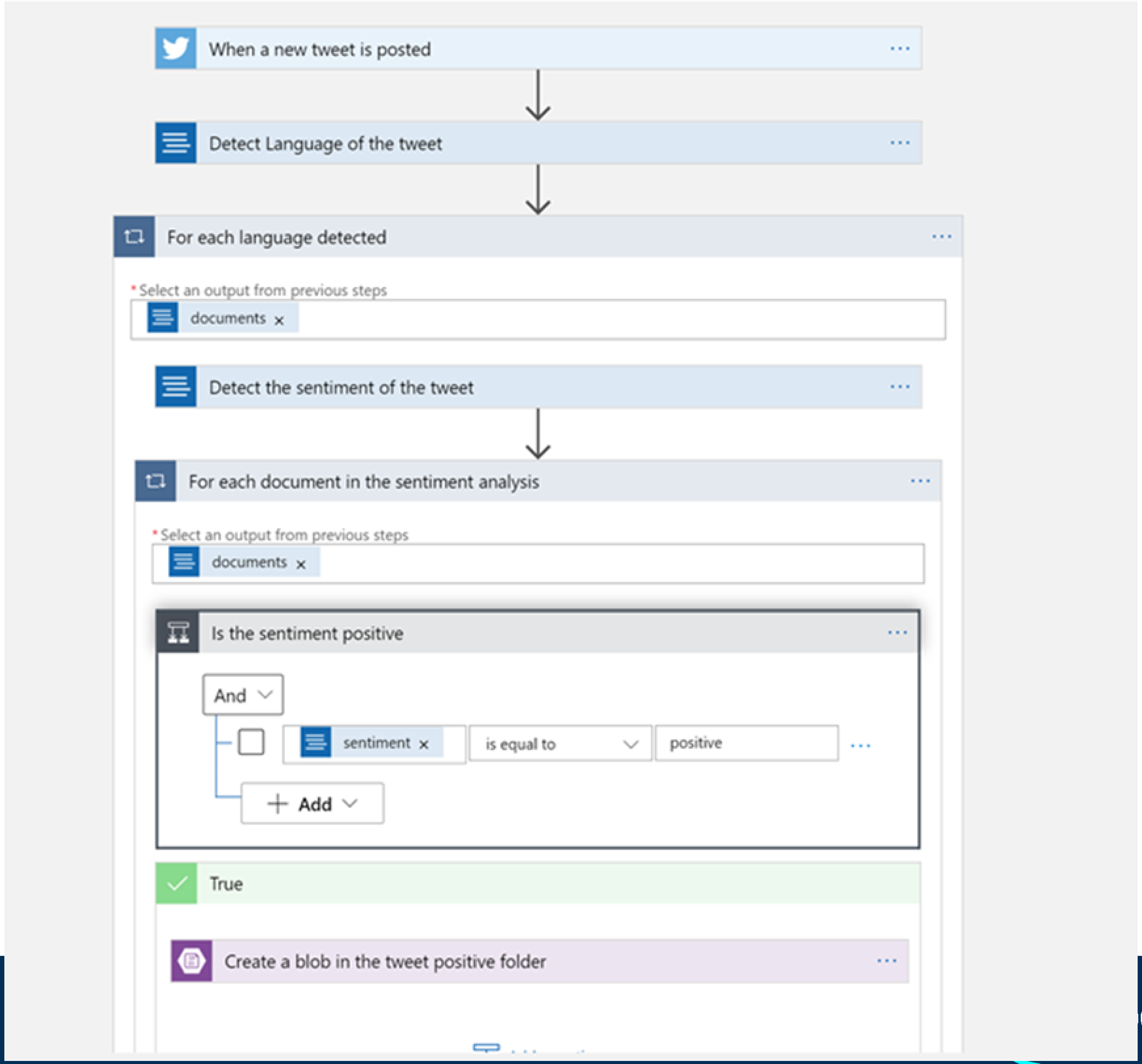
	Developer [1]	Premium
Base unit	€0.95 per hour	€5.92 per hour [2]
Scale unit Increases base unit throughput with additional scale units.	N/A	€2.96 per hour

[1] No SLA is provided on the Developer tier. Scale unit is not offered on the Developer tier.

[2] The Base unit of Premium tier includes 1 standard integration account.

DEMO

Twitter Sentiment Analysis



When a new tweet is posted ...

Detect Language of the tweet ...

For each language detected ...

* Select an output from previous steps

documents ×

Detect the sentiment of the tweet ...

For each document in the sentiment analysis ...

* Select an output from previous steps

documents ×

Is the sentiment positive ...

And

□ sentiment × | is equal to | positive ...

+ Add

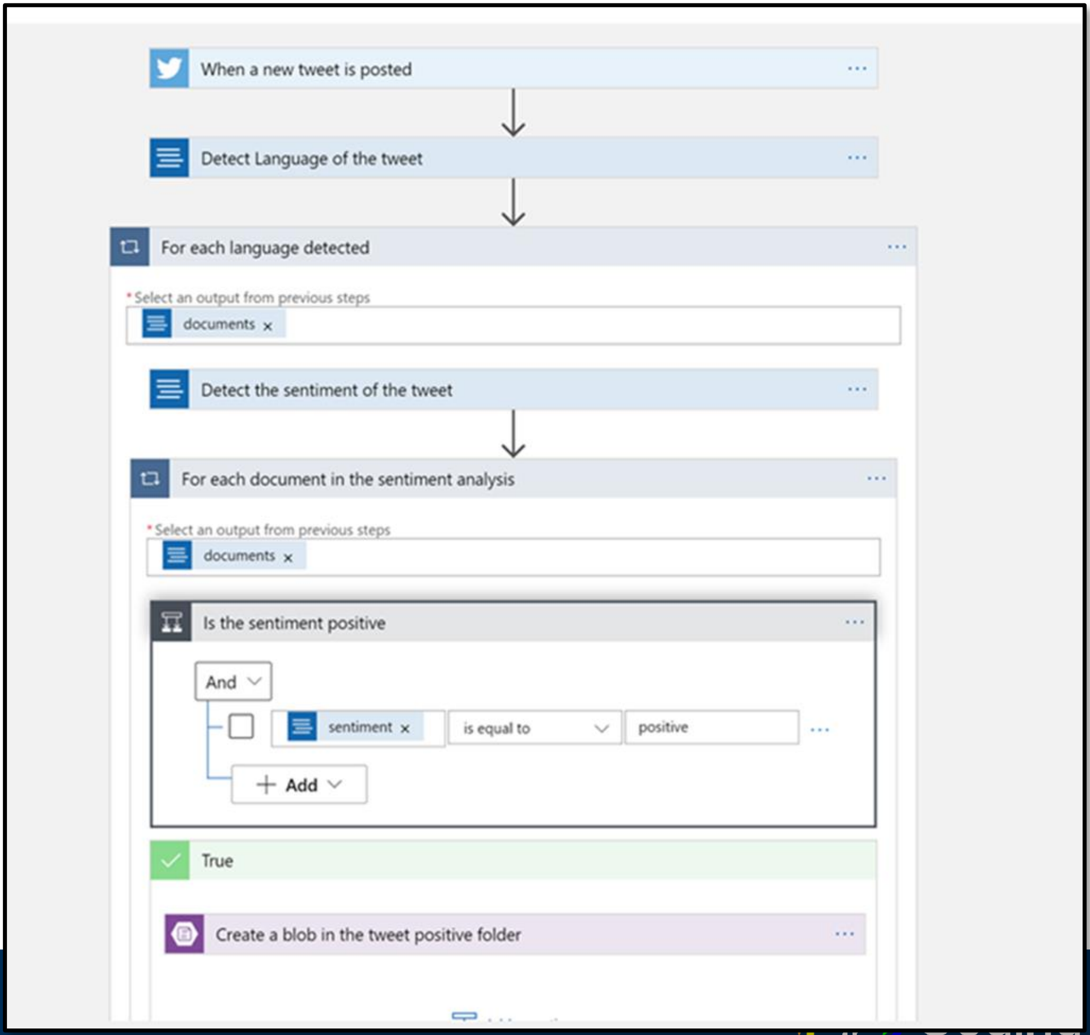
✓ True

Create a blob in the tweet positive folder ...

Pricing sample – Twitter Sentiment Analysis

Suppose that a tweet has only one sentence (best case for the loops) and a neutral sentiment (worst case for the if).

- Actions :
  - 2 loop actions
  - 2 if actions
- Standard connectors:
  - 1 Twitter trigger
  - 2 Cognitive connectors
  - 1 StorageBlob connector
- Monthly Tweets : 100.000



When a new tweet is posted

Detect Language of the tweet

For each language detected

*Select an output from previous steps: documents

Detect the sentiment of the tweet

For each document in the sentiment analysis

*Select an output from previous steps: documents

Is the sentiment positive

And

sentiment is equal to positive

True

Create a blob in the tweet positive folder

Pricing sample – Twitter Sentiment Analysis

- Total Actions:

  100.000 exec * 4 actions = 400.000 actions

- Actions to pay:

  400.000 actions – 4.000 actions = 396.000 actions

- Actions cost:

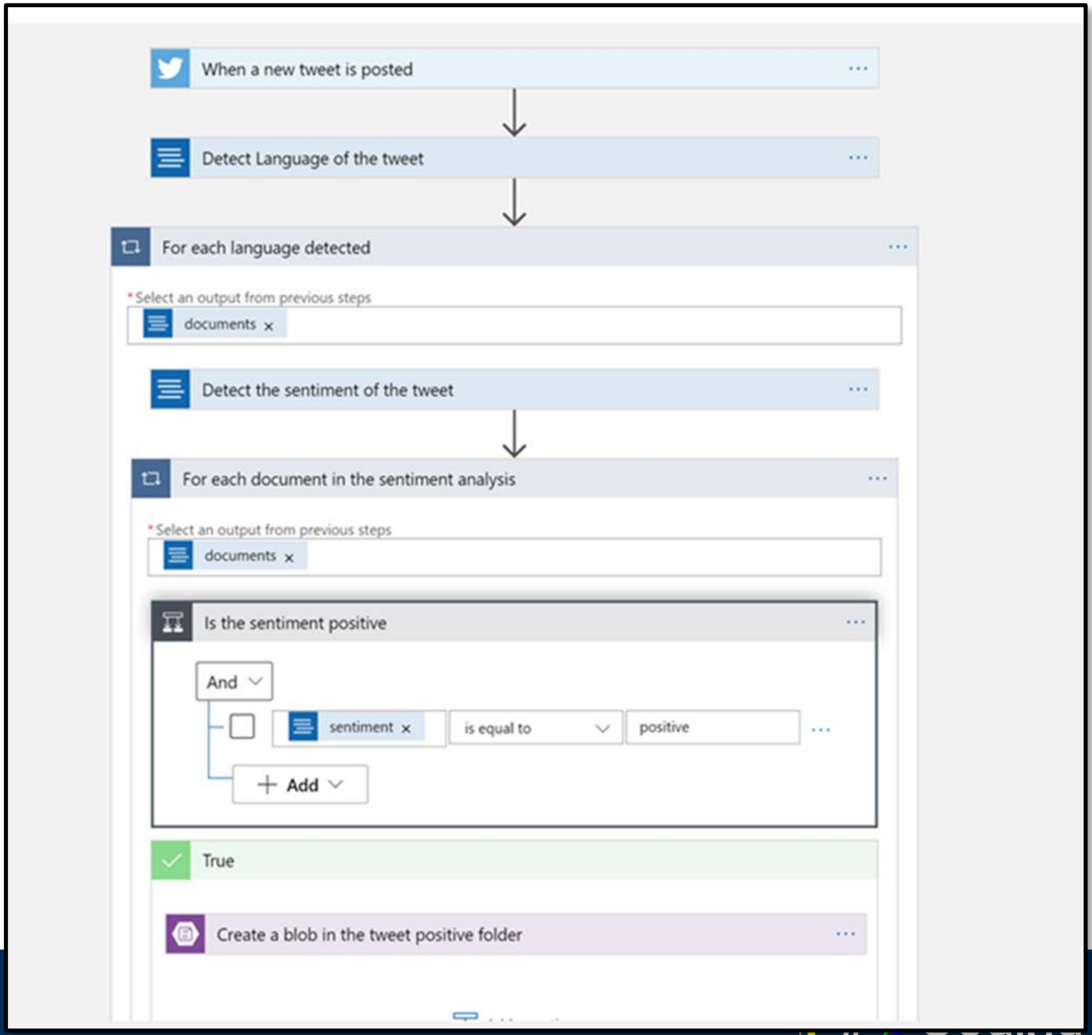
  396.000 actions * 0,000022 € = 8,712 €

- Total Standard Connectors:

  100.000 exec * 4 conn = 400.000 conn

- Standard Connectors cost:

  400.000 conn * 0,000106 € = 42,400 €



When a new tweet is posted

Detect Language of the tweet

For each language detected

* Select an output from previous steps: documents ×

Detect the sentiment of the tweet

For each document in the sentiment analysis

* Select an output from previous steps: documents ×

Is the sentiment positive

And: sentiment × | is equal to | positive

+ Add

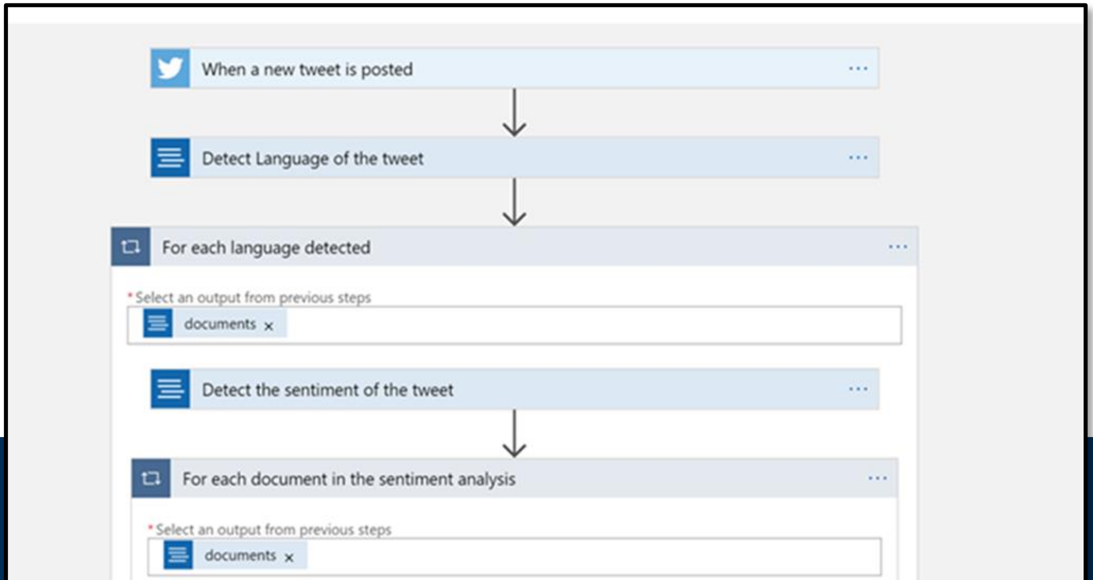
True

Create a blob in the tweet positive folder

Pricing sample – Twitter Sentiment Analysis

Monthly cost

8,712 + 42, 400 = 51,112 €



When a new tweet is posted

Detect Language of the tweet

For each language detected

* Select an output from previous steps

documents ×

Detect the sentiment of the tweet

For each document in the sentiment analysis

* Select an output from previous steps

documents ×

The final battle!!

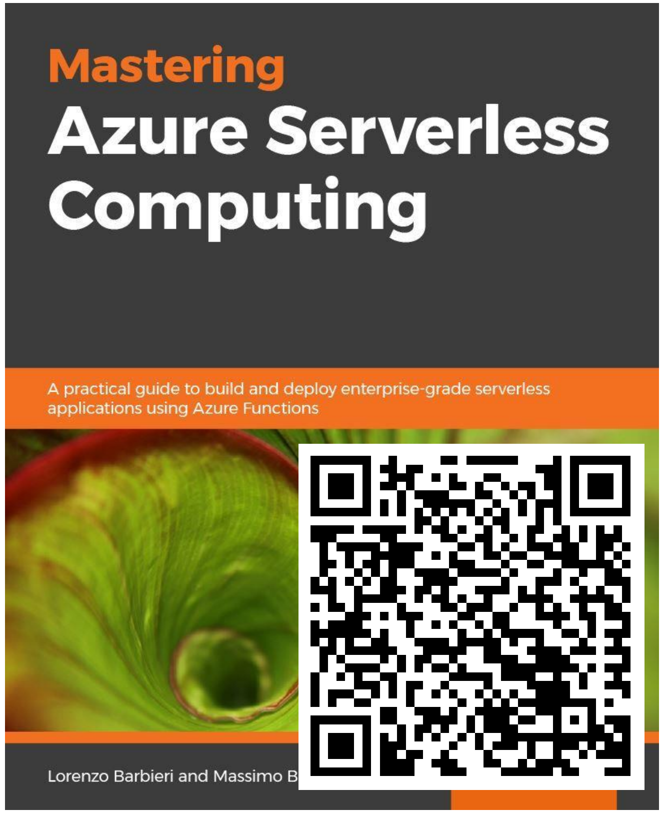
Durable Functions

- You have a team with development skill and the orchestration doesn’t involve complex systems
- You want to reuse code from other projects
- You require them to run not only on Azure, but on Azure Stack or Containers
- You prefer to have all the power and flexibility of a robust programming language
- You can implement stateful entities (similar to Virtual Actor)

Logic Apps

- Leveraging a huge list of connectors reducing the time-to-market and ease connectivity
- Visual tools to manage and troubleshoot workflows are required
- It’s ok to run only on Azure
- A visual designer and less coding are preferred
- Integrated versioning system for the orchestration

KEEP CALM AND USE TOGHETER



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 DATA FUNDAMENTALS ★
Microsoft CERTIFIED AZURE AI FUNDAMENTALS ★
Microsoft CERTIFIED SECURITY, COMPLIANCE, AND IDENTITY FUNDAMENTALS ★
Microsoft CERTIFIED AZURE IOT DEVELOPER SPECIALTY
Microsoft CERTIFIED AZURE DEVELOPER ASSOCIATE ★★
Microsoft CERTIFIED AZURE ADMINISTRATOR ASSOCIATE ★★
Microsoft CERTIFIED AZURE AI ENGINEER ASSOCIATE ★★
Microsoft CERTIFIED AZURE DEVOPS ENGINEER EXPERT ★★★
Microsoft CERTIFIED AZURE SOLUTIONS ARCHITECT EXPERT ★★★
MICROSOFT CERTIFIED TRAINER Microsoft MCT Trainer 2020 - 2021

Coding

References

- Durable Functions documentation
  https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview
- Logic App documentation
  https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-overview
- Netherite project
  https://github.com/microsoft/durabletask-netherite
- Demo GitHub repo
  https://github.com/massimobonanni/OrderManagerServerless
- Serverless learning path – Azure Functions
  https://docs.microsoft.com/en-us/learn/paths/create-serverless-applications/
- Serverless learning path – Logic App
  https://docs.microsoft.com/en-us/learn/paths/build-workflows-with-logic-apps/

Coding

Thank You!

Our Socials