Building stateful serverless orchestrations with Azure Durable Functions

Callon Campbell
Microsoft MVP | Azure

@flying_maverick

About me

Callon Campbell

Azure Architect | Developer
Adastra
Microsoft MVP | Azure

- 20 years enterprise development with Microsoft technologies – .NET (C#), Azure, ASP.NET, Desktop, SQL, and Mobile
- Passionate about serverless and cloud-native application development, with focus on app migration and modernization, app integration and data analytics
- Blogging at https://TheFlyingMaverick.com
- Speaker at community events and meetups
- Organizer of “Canada’s Technology Triangle .NET User Group” in Kitchener, Ontario

Agenda

- Serverless refresher
- Serverless challenges
- Solutions through Durable Functions
- Storage provider options
- Demos
- Wrap up

Azure Functions

Run code when events occur

Event-driven scaling

No infrastructure management

Consumption pricing with free grant

What is Azure Functions?

Events

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

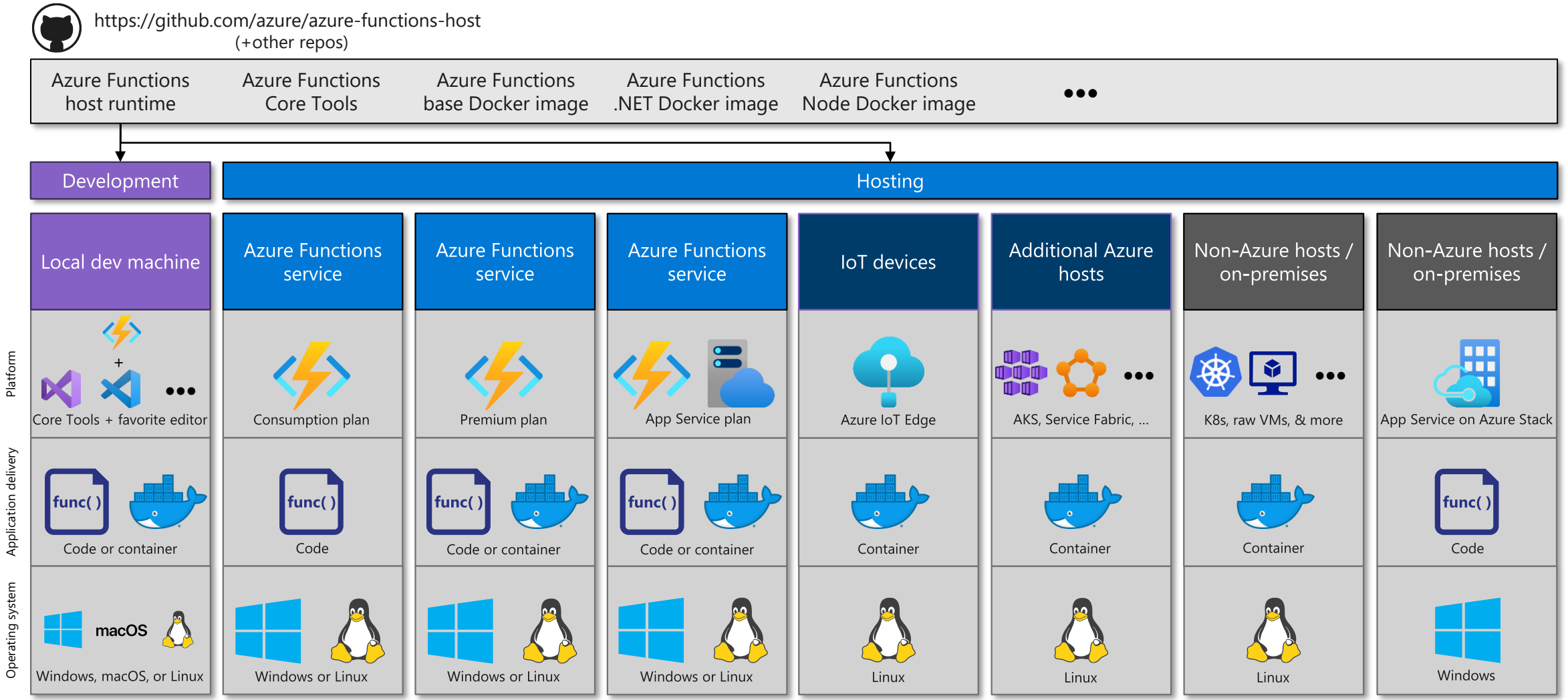
Code

Author functions in C#, F#, Node.js (JavaScript, TypeScript), Java, Python, and PowerShell

Outputs

Send results to an ever-growing collection of services

Functions everywhere



Azure Functions Hosting Models

Which one do I use?

In-process

- Migrate from .NET Core 3.1 in-process
- Need Durable Functions or “rich” SDK type bindings

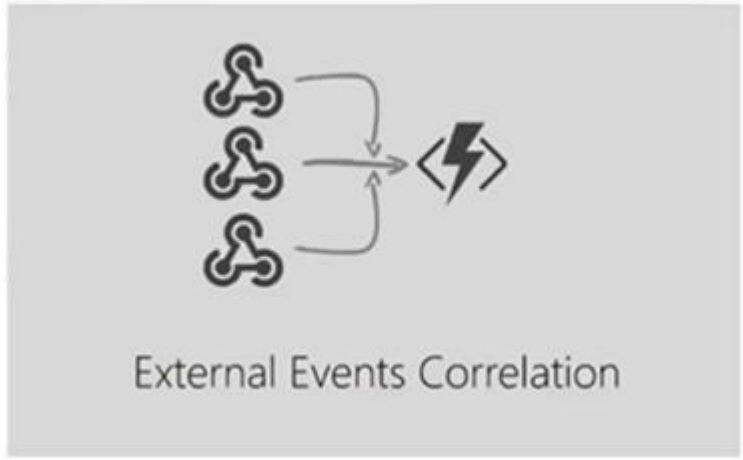
Isolated process

- Migrate from .NET 5 or .NET 6 isolated process
- Durable Functions now supported in .NET 7*

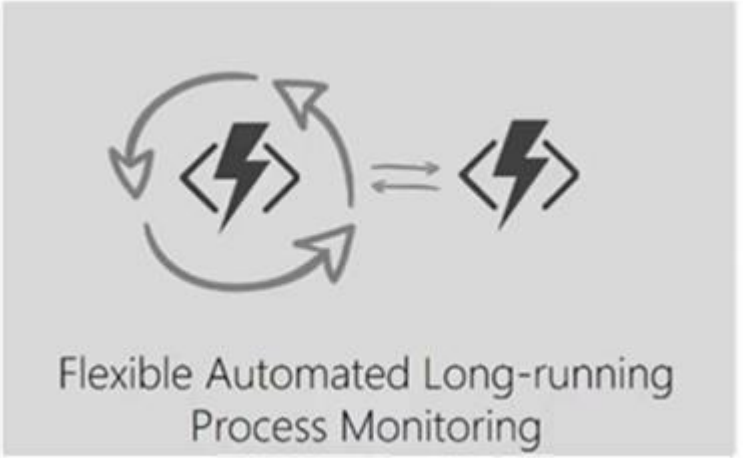
What are the challenges in serverless?

Manageable Sequencing
+ Error Handling / Compensation

Fanning-out & Fanning-in

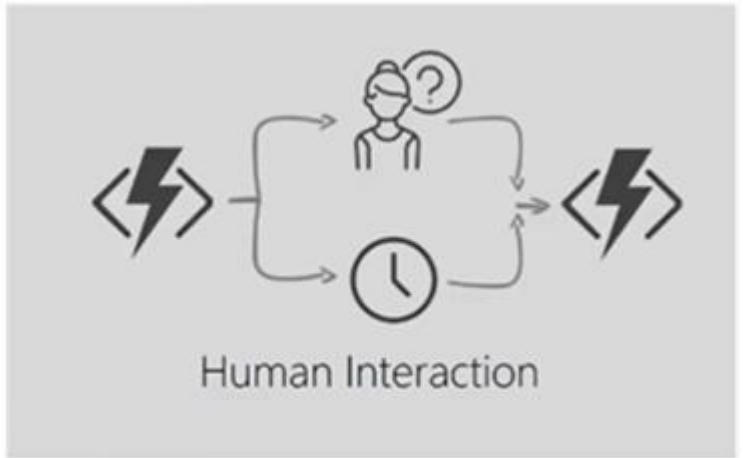


External Events Correlation



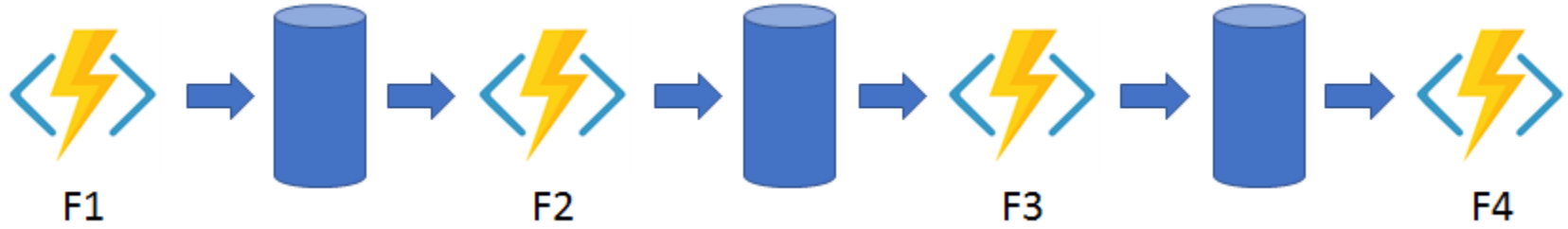
Flexible Automated Long-running
Process Monitoring

Http-based
Async Long-running APIs



Human Interaction

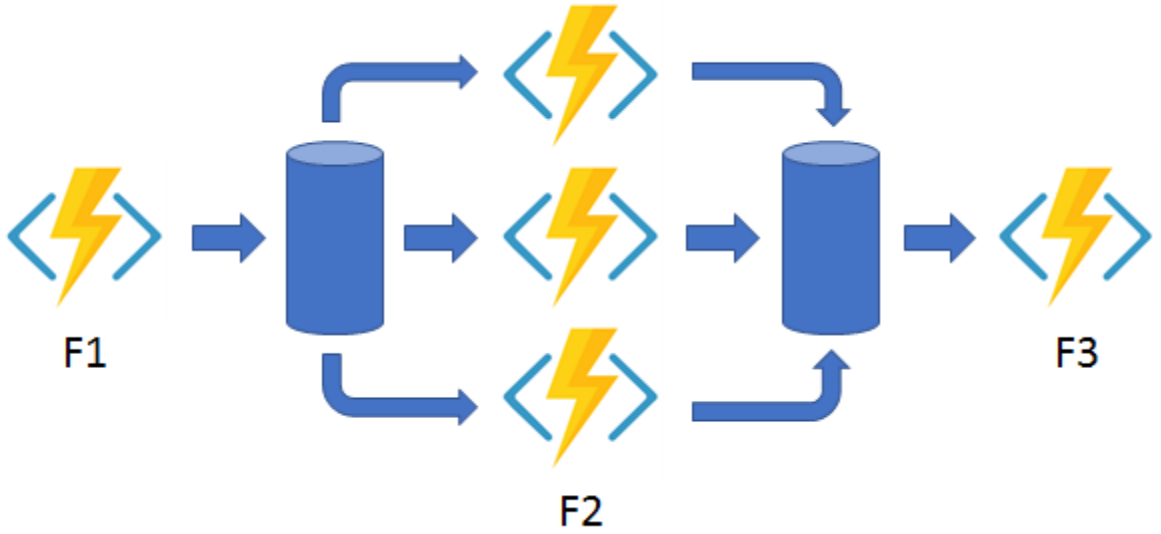
Pattern #1: Function chaining



Problems:

- No visualization to show relationships between functions and queues
- Middle queues are an implementation detail – conceptual overhead
- Error handling adds a lot more complexity

Pattern #2: Fan-out/fan-in



Problems:

- Fanning-out is easy, but fanning-in is significantly more complicated
- Functions offers to help with this scenario today
- All the same problems as the previous pattern

Introducing Durable Functions

- Durable Functions is an extension of Azure Functions and is built on top of the Durable Task Framework.
- Enables you to write long-running orchestration as a single function, that is reliable, event-driven, and is stateful.
- Simplifies complex, stateful coordination requirements in serverless applications
- Execution state is saved in an Azure Storage account, ensuring that functions could recover automatically from any infrastructure failure
- Supports .NET (C#/F#), Java, JavaScript, TypeScript, Python, and PowerShell.

Application Patterns

The primary use case for Durable Functions is simplifying complex, stateful coordination requirements in serverless applications.

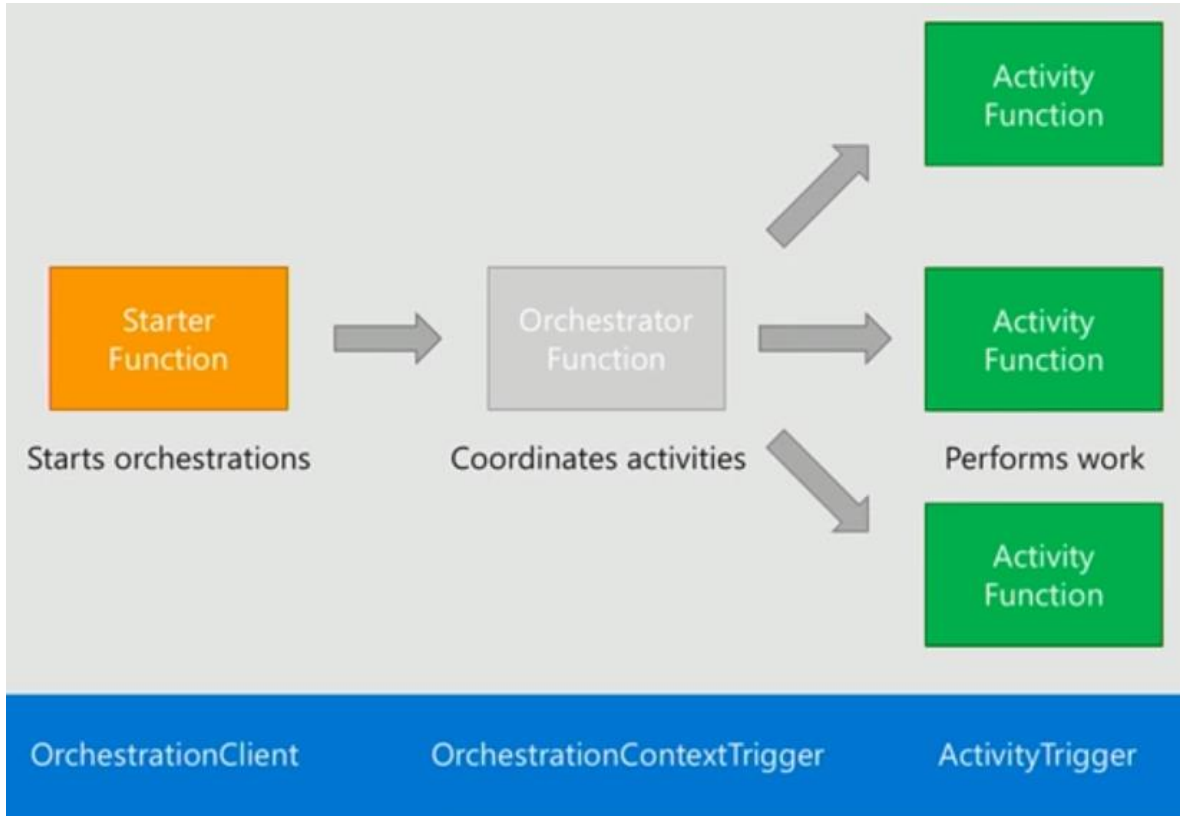
Typical application patterns that can benefit from Durable Functions:

- Function chaining
- Fan-out/fan-in
- Async HTTP APIs
- Monitoring
- Human interaction
- Aggregartor (stateful entities)

Durable Functions Components

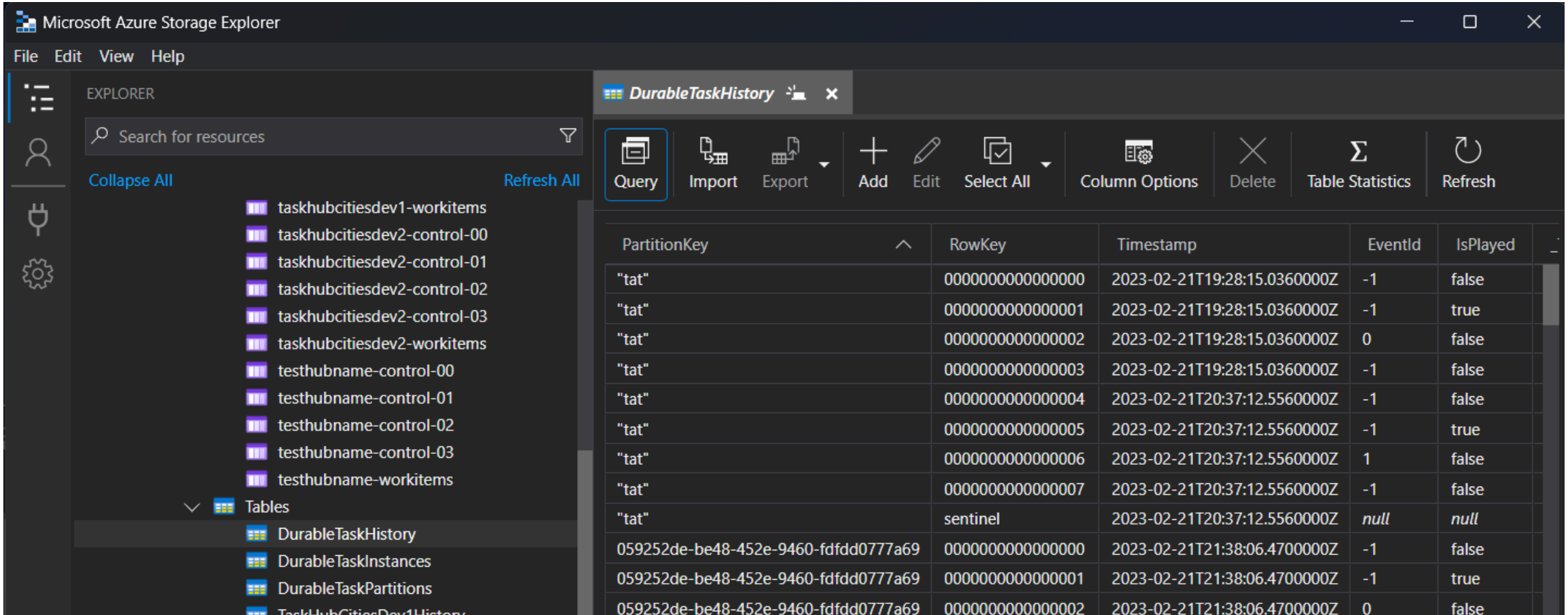
What makes up Durable Functions:

- Starter Function
- Orchestrator Function
- Activity Function



Event Sourcing Log

- 111

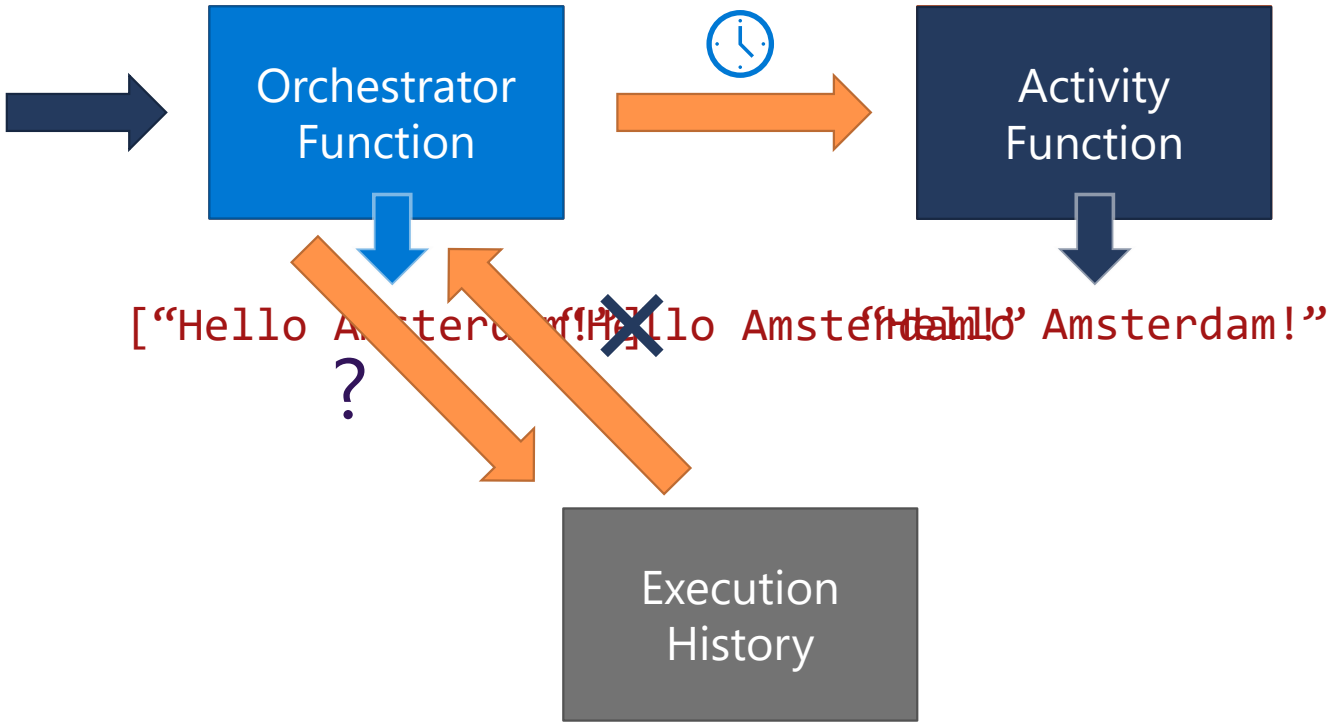


How does it work?

```
var outputs = new List<string>();

outputs.Add(await context.CallActivityAsync<string>("SayHello", “Amsterdam"));

return outputs;
```



Orchestrator Started
Execution Started
Task Scheduled, SayHello, “Amsterdam”
Orchestrator Completed
Task Completed, “Hello Amsterdam!”
Orchestrator Started
Execution Completed, ["Hello Amsterdam!"]
Orchestrator Completed

Orchestration Constraints

- Orchestrator code must be deterministic
- Never use random numbers, DateTime.UtcNow, Guid.NewGuid(), etc.
  - Use DurableOrchestrationContext.CurrentUtcDateTime
- Orchestrator code should be **non-blocking**. Never do I/O directly in the orchestrator
  - Do I/O in activity functions
- Don't write infinite loops
  - Use DurableOrchestrationContext.ContinueAsNew()

Demo 1 – Hello Durable!

Function Chaining

Demo 2 – Configuration!

Demo 3 – Star Wars API!

Function Fan-out / Fan-in

New Storage Providers

Support for two new backend storage providers for storing durable runtime state, “Netherite” and Microsoft SQL Server (including full support for Azure SQL Database).

These new storage options allow you to:

- Run at higher scale
- Greater price-performance efficiency
- Greater portability compared to the default Azure Storage configuration

Limitations of Azure Storage Provider

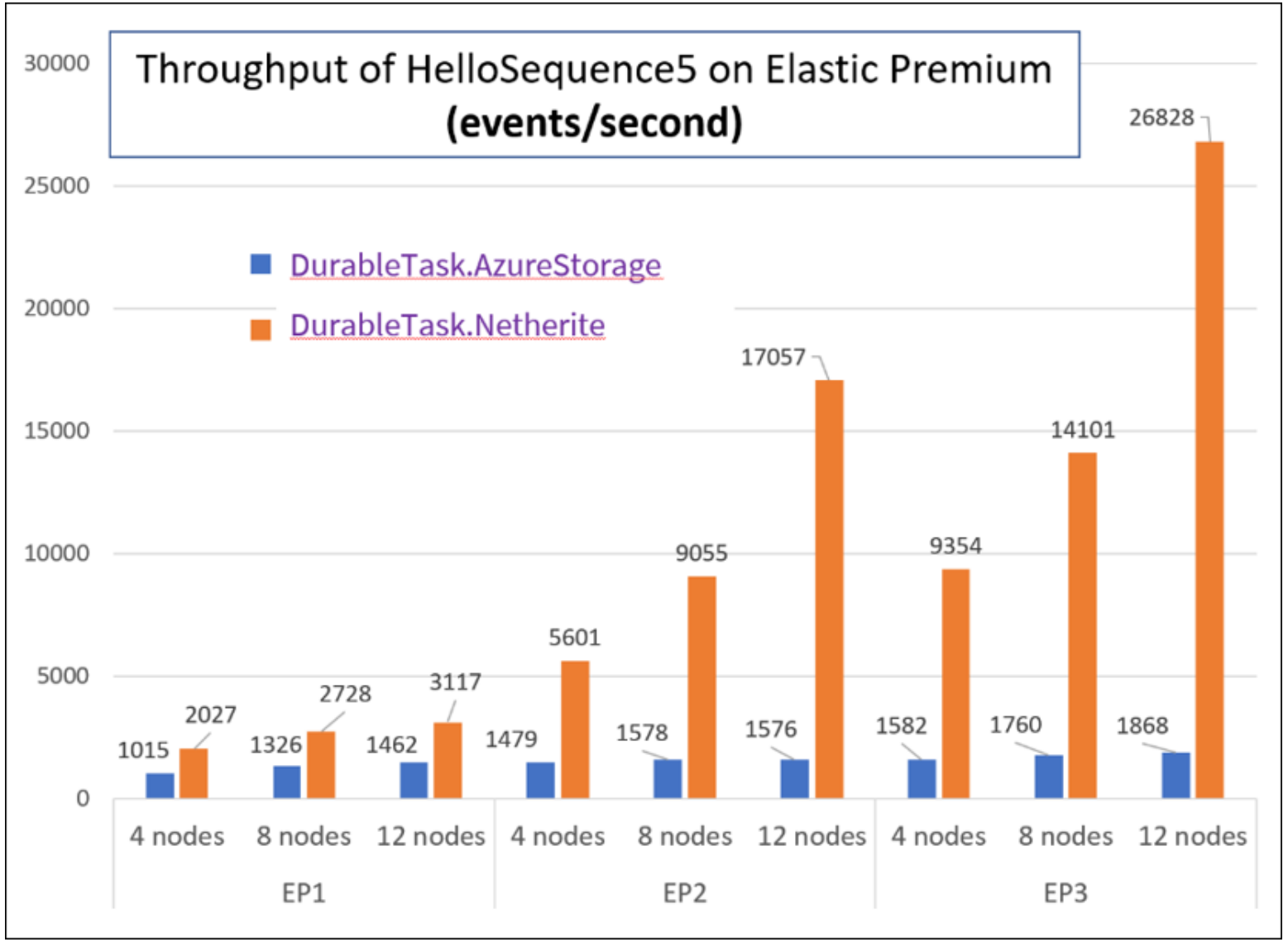
Azure Storage has the following limitations:

- Limits on the number of transactions per second
- Strict data size limits for queue messages and Azure Table entities
- Hard to predict costs
- Can’t easily support certain enterprise business continuity requirements, such as backup/restore and disaster recovery without data loss
- Azure only

Introducing “Netherite”

- The Netherite name originates from the world of Minecraft
- Developed by Microsoft Research, it combines Azure Event Hubs with the FASTER database technology on top of Azure Page Blobs
- Supports significantly higher-throughput of orchestrations and entities compared to other Durable storage providers
- More cost-effective for high-throughput loads

Performance Benchmark



Single Azure Event Hubs throughput unit (1 TU), costing approximately $22/month USD on the Standard plan, running a simple function-chaining sample with 5 activity calls running on the Azure Functions Elastic Premium plan.

Introducing SQL Server Provider

- SQL Server provider allows Durable Functions to run anywhere
  - Azure SQL Database
  - On-premises
  - Docker Containers / Kubernetes
  - IaaS
  - Multi-Cloud
- Leverage your existing SQL Server investments
  - Backup/restore
  - Failover
  - Encryption
  - Compliance
- Easily integrate with existing SQL-based applications

Demo 4 – Netherite storage!

Alternate Storage Provider

Choosing the right storage provider

Azure Storage

- Generally available
- Dependencies:
  - Azure Storage Account (general purpose v1)
- Minimal setup
- Lowest minimum cost
- Consumption plan: Yes
- Elastic Premium plan: Yes
- Disconnected support: No

Netherite

- Generally available
- Dependencies:
  - Azure Event Hubs
  - Azure Storage Account
- Max throughput
- Lower cost at scale
- Consumption plan: No
- Elastic Premium plan - requires runtime scale monitoring
- Disconnected support: No

SQL Server

- Generally available
- Dependencies:
  - SQL Server 2019 or Azure SQL Database
- Runs anywhere
- Enterprise friendly
- Consumption plan: No
- Elastic Premium plan- requires runtime scale monitoring
- Disconnected support: Yes

In closing

- Input/output of functions should be serializable.
- Orchestration Functions can only call Activity Functions in the same Function App.
- Don’t use an orchestration function to call a single activity function.
- Keep your orchestrations small.
- What changes together should be deployed together.
- Azure Durable Functions make managing state and complex workflows easy and familiar.

What's New

- Durable Function's support for .NET 7.0 running in the isolated worker process is now generally available (supports .NET 6.0 and .NET Framework too!)

- However, not all features from in-process Durable Functions have been migrated to the isolated worker. Known missing features are:
  - Durable Entities
  - CallHttpAsync

Let's Connect

Callon@CloudMavericks.ca

@flying_maverick

https://LinkedIn.com/in/CallonCampbell

https://GitHub.com/CallonCampbell

Thank You

ευχαριστώ Salamat Po متشكرم شكراً Grazie

благодаря ありがとうございます Kiitos Teşekkürler 谢谢

ขอบคุณครับ Obrigado شكريه Terima Kasih Dziękuję

Hvala Köszönöm Tak Dank u wel дякую Tack

Mulţumesc спасибо Danke Cám ơn Gracias

多謝晒 Ďakujem תודה நன்றி Děkuji 감사합니다

References

- Azure Durable Functions documentation | Microsoft Learn
- Durable orchestrator code constraints - Azure Functions | Microsoft Learn
- Handling errors in Durable Functions - Azure | Microsoft Learn
- Durable Functions storage providers - Azure | Microsoft Learn
- Overview of Durable Functions in the .NET isolated worker - Azure | Microsoft Learn
- Task hubs in Durable Functions - Azure | Microsoft Learn