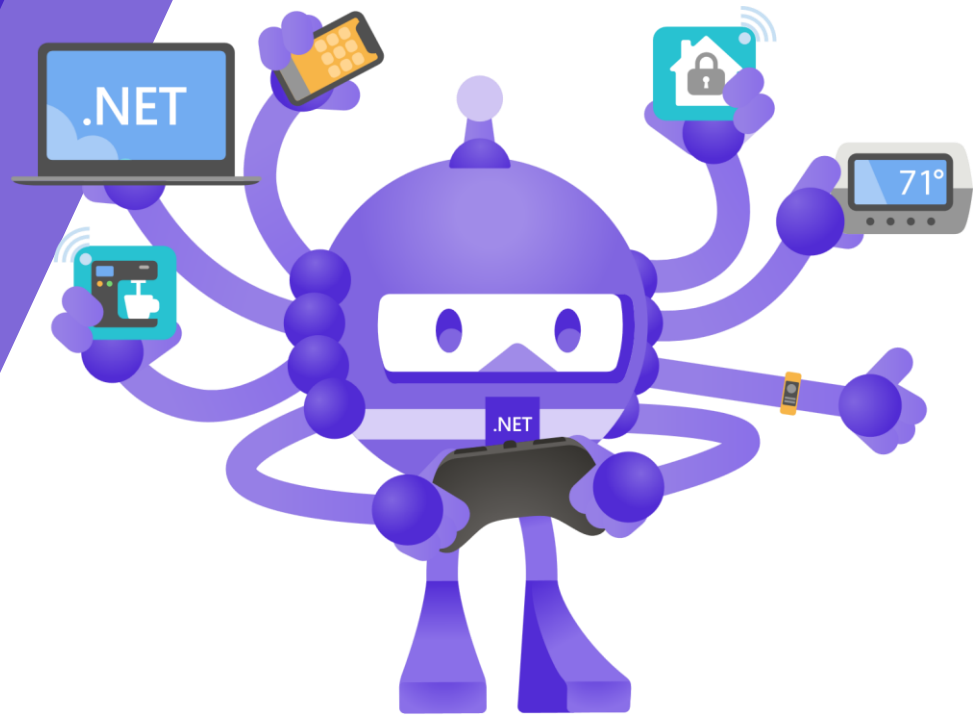
.NET Conference 2024

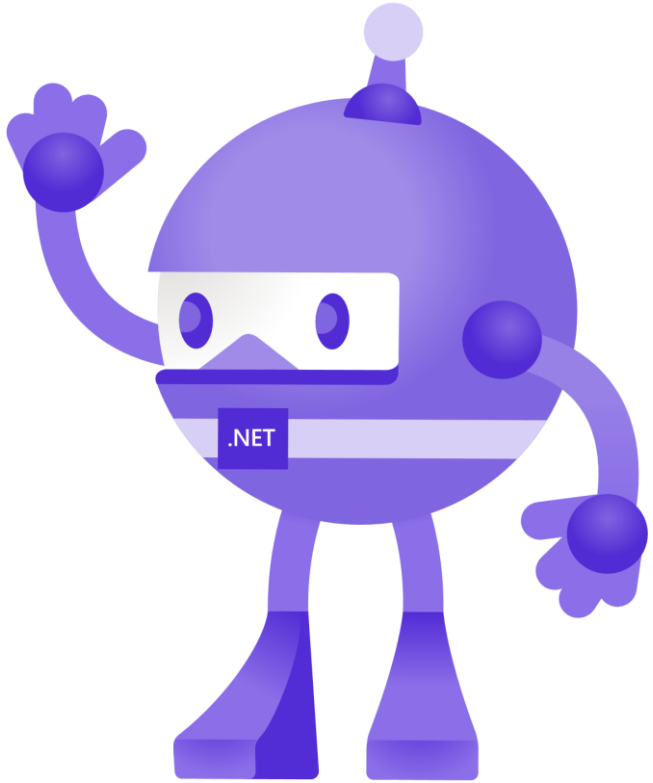


Microsoft avanade BC Soft consulenza sistemi informativi randstad UNIKEY Packt Reti Società Benefit JET BRAINS

Azure Functions e .NET 8.0: funzioni da paura!

Massimo Bonanni

Technical Trainer @ Microsoft



SPONSOR


Microsoft


avanade


BC Soft
consulenza sistemi informativi


randstad


UNIKEY
Bringing IT knowledge to the people


Reti
Società Benefit

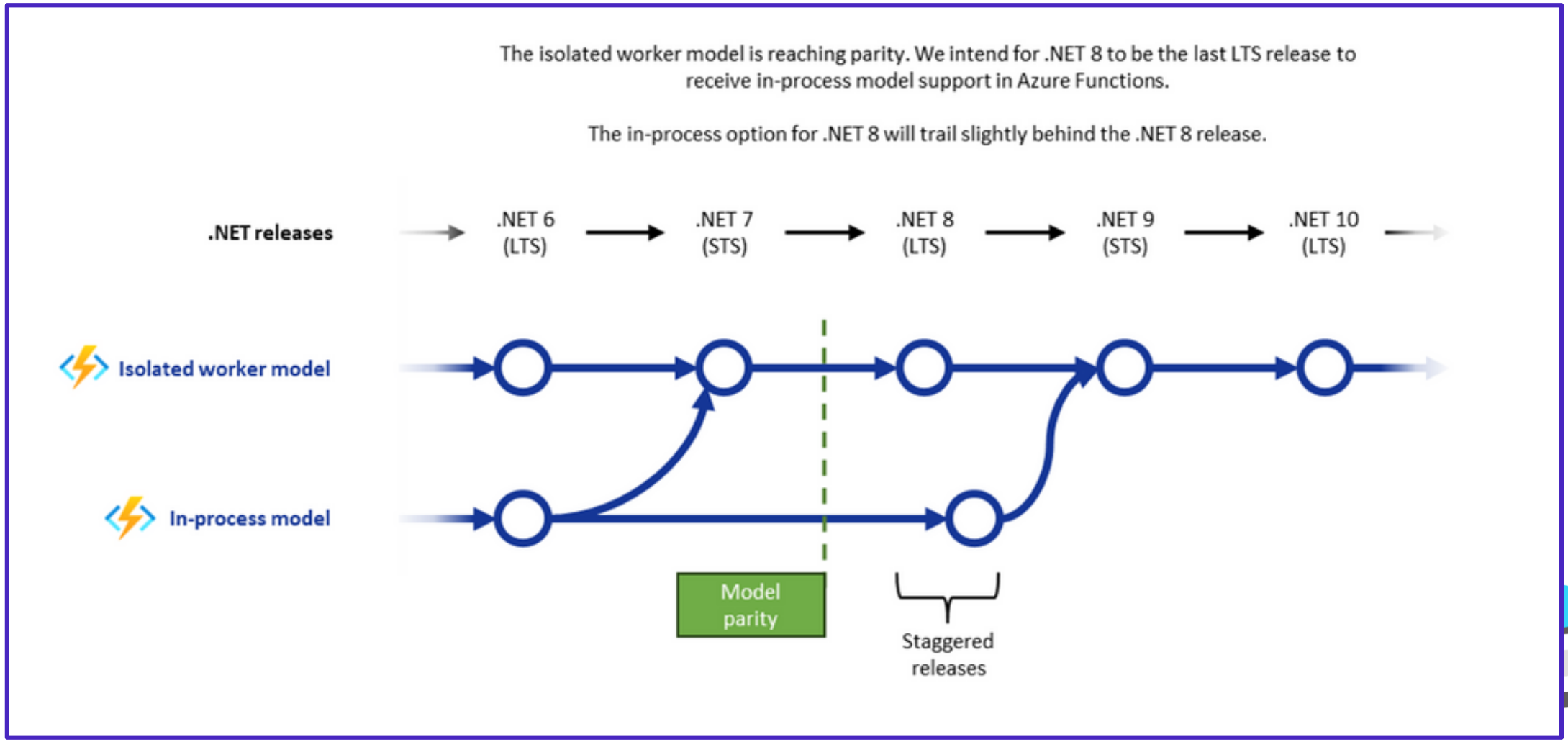

JET BRAINS


Packt


NET CODE


NET CODE

Roadmap


The isolated worker model is reaching parity. We intend for .NET 8 to be the last LTS release to receive in-process model support in Azure Functions.
The in-process option for .NET 8 will trail slightly behind the .NET 8 release.
.NET releases
.NET 6 (LTS)
.NET 7 (STS)
.NET 8 (LTS)
.NET 9 (STS)
.NET 10 (LTS)
Isolated worker model
In-process model
Model parity
Staggered releases

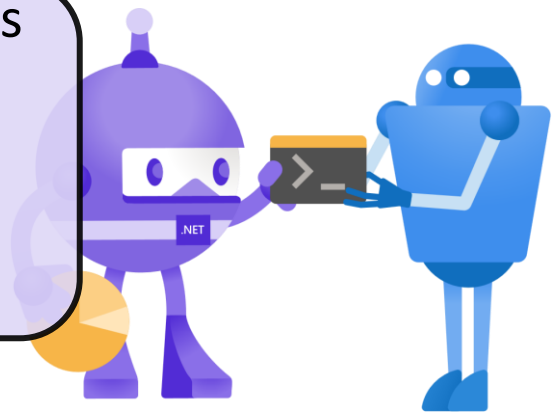
Isolated vs In-process worker mode

Isolated

- ✓ Your function code runs in a separate .NET worker process
- ✓ .NET Framework 4.8, .NET 6.0, 7.0, 8.0

In-process

- ✓ Your function code runs in the same process as the Functions host process
- ✓ Supports only Long Term Support (LTS) versions of .NET
- ✓ .NET 6.0



Isolated vs In-process worker mode

In-Process

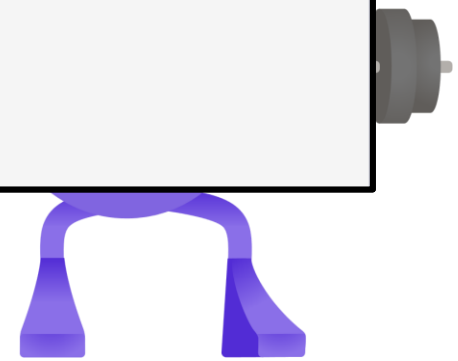
- InProcessFunc
  - Connected Services
  - Dependencies
    - Analyzers
    - Frameworks
    - Packages
      - Microsoft.NET.Sdk.Functions (4.2.0)
  - Properties
  - .gitignore
  - Function1.cs
  - host.json
  - local.settings.json

Isolated

- IsolatedFunc
  - Connected Services
  - Dependencies
    - Analyzers
    - Frameworks
    - Packages
      - Microsoft.ApplicationInsights.WorkerService (2.21.0)
      - Microsoft.Azure.Functions.Worker (1.20.1)
      - Microsoft.Azure.Functions.Worker.ApplicationInsights (1.1.0)
      - Microsoft.Azure.Functions.Worker.Extensions.Http (3.1.0)
      - Microsoft.Azure.Functions.Worker.Extensions.Http.AspNetCore (1.2.0)
      - Microsoft.Azure.Functions.Worker.Sdk (1.16.4)
  - Properties
  - .gitignore
  - Function1.cs
  - host.json
  - local.settings.json
  - Program.cs

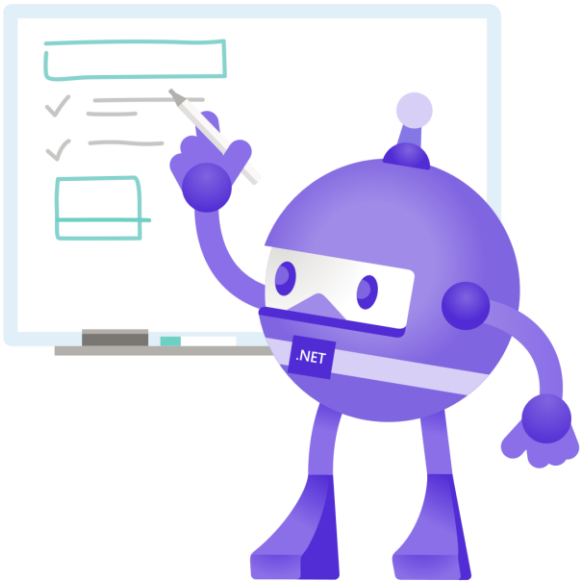
```
var host = new HostBuilder()
    .ConfigureFunctionsWebApplication()
    .ConfigureServices(services =>
    {
        services.AddApplicationInsightsTelemetryWorkerService();
        services.ConfigureFunctionsApplicationInsights();
    })
    .Build();

host.Run();
```



Benefit of Isolated worker model

- **Fewer conflicts:** Because your functions run in a separate process, assemblies used in your app don't conflict with different versions of the same assemblies used by the host process.
- **Full control of the process**: You control the start-up of the app, which means that you can manage the configurations used and the middleware started.
- **Standard dependency injection:** Because you have full control of the process, you can use current .NET behaviors for dependency injection and incorporating middleware into your function app.
- **.NET version flexibility:** Running outside of the host process means that your functions can run on versions of .NET not natively supported by the Functions runtime, including the .NET Framework.

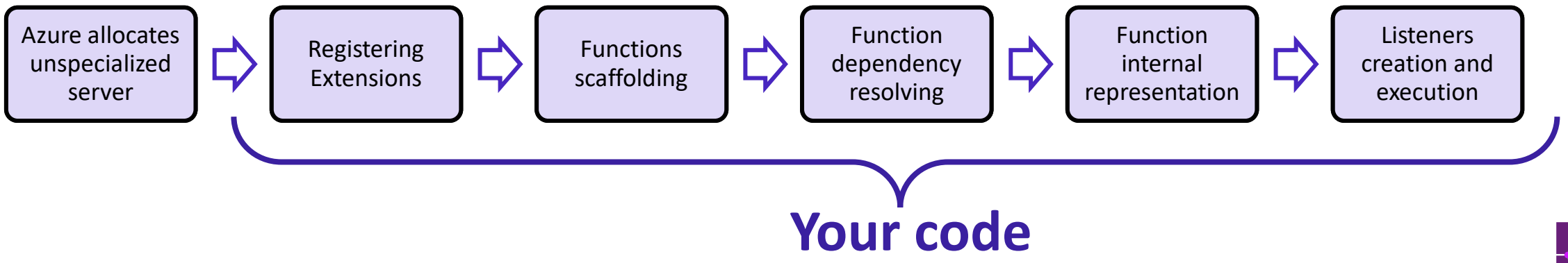


Cold Start

Cold Start is the time Azure Functions spends getting ready to run your code.

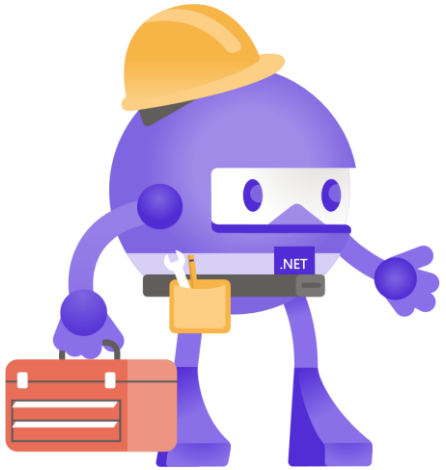
In the **Consumption** plan, if your code is not executed for 20/30 minutes, the Function App is "turned off" (it helps you spend less).

At the next request, the Function App must start from scratch.



Cold Start – Best Practices

- ✓ Avoid unused dependencies
- ✓ Remove functions no longer used
- ✓ Distribute the Functions appropriately among different Function Apps
- ✓ Create a trigger timer function that keeps the Function App awake
- ✓ Linux plans generally have a shorter time than Cold Start
- ✓ Cold Start also depends on language
- ✓ Use a different plan such as Premium or Dedicated (or other ☺)

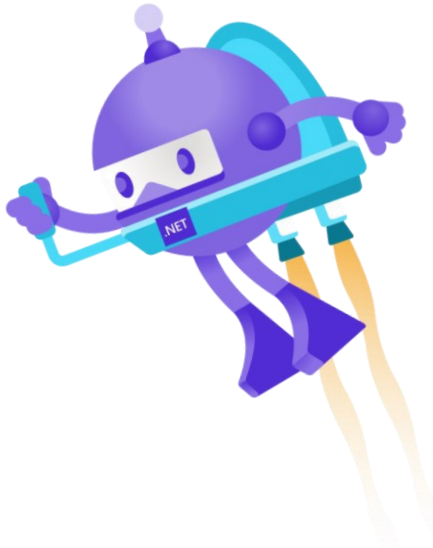


Cold Start optimization

- ✓ Enable Placeholders setting `WEBSITE_USE_PLACEHOLDER_DOTNETISOLATED` to 1
- ✓ Make sure that the `netFrameworkVersion` property of the function app matches your project's target framework
- ✓ Make sure that your function app is configured to use a 64-bit process
- ✓ An optimized version of this component is enabled by default starting with version 1.16.2 of the SDK
- ✓ You can compile your function app as `ReadyToRun` binaries

Function Executor Optimization

- ✓ When you invoke a function (triggered by an event or HTTP request), the Function Executor comes into play.
- ✓ It handles the execution of your function code, ensuring that it runs in response to the specified trigger.
- ✓ The Function Executor manages the entire lifecycle of a function execution, from initialization to cleanup.
- ✓ It uses metadata to direct call the function method.
- ✓ A newer version is available in preview which optimize the function run.
- ✓ To use this improvement set `FunctionEnableExecutorSourceGen` to true in project file.



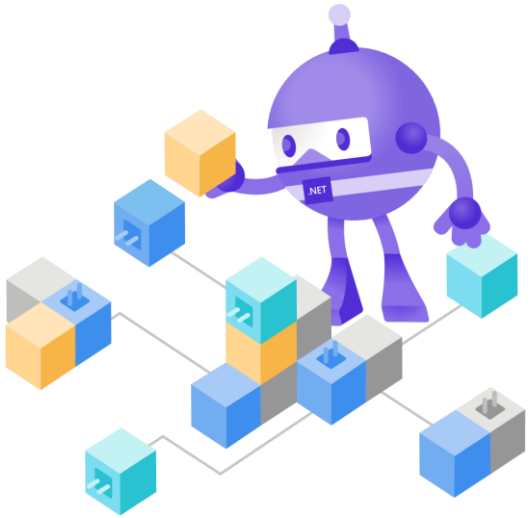
ASP.NET Core Integration

- ✓ Add package:

  `Microsoft.Azure.Functions.Worker.Extensions.Http.AspNetCore`

- ✓ Use:
  - `HttpRequest`
  - `HttpResponse`
  - `IActionResult`

- ✓ Full control of the emission of telemetry using Application Insight SDK.
- ✓ Support for Middlewares.

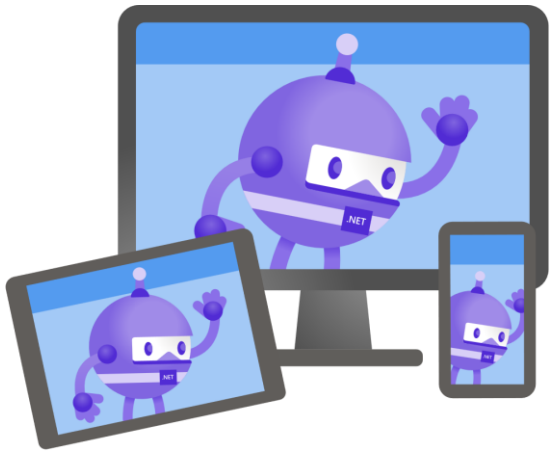


Azure Functions binding for OpenAI

- ✓ Available Features:
  - Text completions
  - Chat completion
  - Assistants
  - Embeddings generators
  - Semantic search
- ✓ It depends on the Azure AI OpenAI SDK
- ✓ It is in alpha

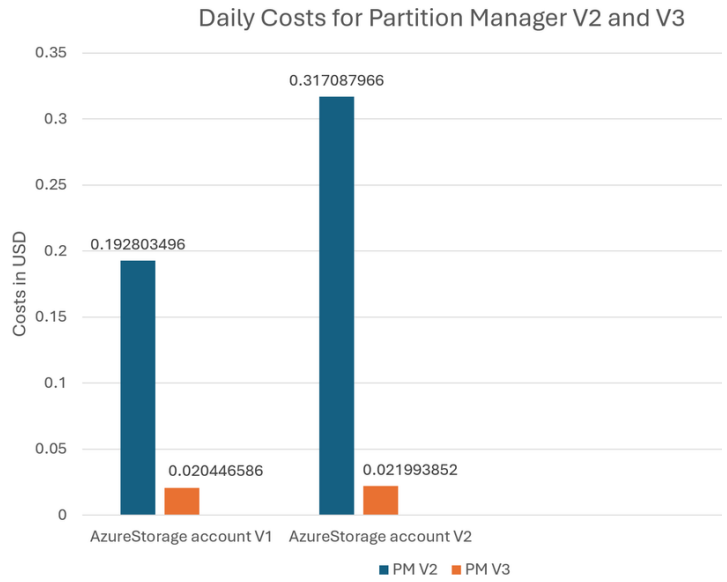
News

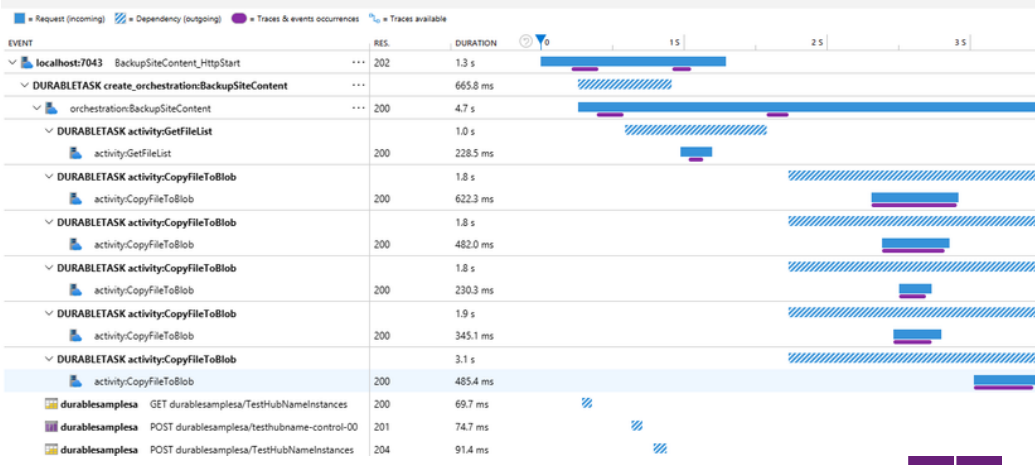
- ✓ Azure Functions in Container Apps
- ✓ Register Azure Clients, leverage DI to inject Azure service clients like Blobs, Service Bus or others
- ✓ SQL Binding for Azure Functions (GA)
- ✓ Azure Data Explorer bindings (Public Preview)
- ✓ Azure Cache for Redis triggers/bindings (Public Preview)
- ✓ Azure Functions .NET 8 support in Linux plans (Public Preview)
  - Azure Functions now supports .NET 8 preview 7 for applications using the isolated worker model and running on Linux Elastic Premium and Dedicated plans.
  - These projects can be deployed to newly created apps on Linux Elastic Premium and Dedicated plans which are configured to support .NET 8



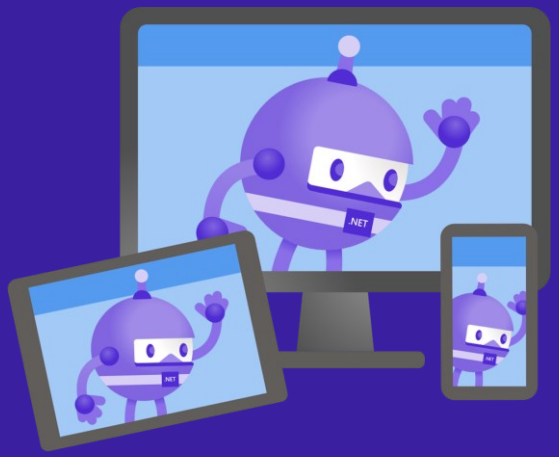
Durable Functions Extension v3.0.0

- ✓ Support for Isolated Model;
- ✓ Upgraded Azure Storage SDK and new partition manager introduction.
- ✓ Partition Manager V3:
  - More cost-efficient, uses Azure Tables for partition assignment (instead blob lease), and improves debuggability.
  - Also supported in Durable Functions V2 starting from version V2.10.0. Enabled using host.json file.
- ✓ No Code Changes: Updates integrate in the background; no code adjustments needed for the preview package.
- ✓ Distributed Tracing V2 supports all Durable Functions language SDKs, including .NET Isolated, and all Durable Functions backends.



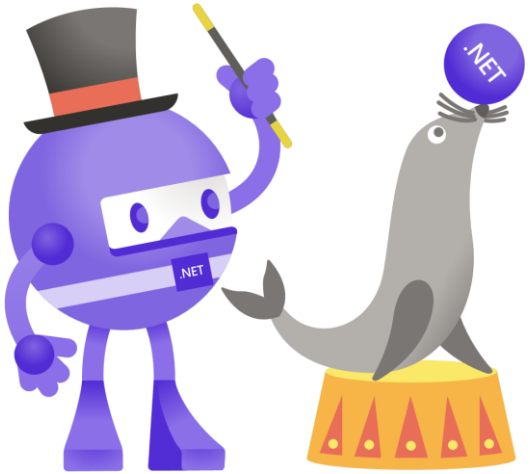


DEMOS



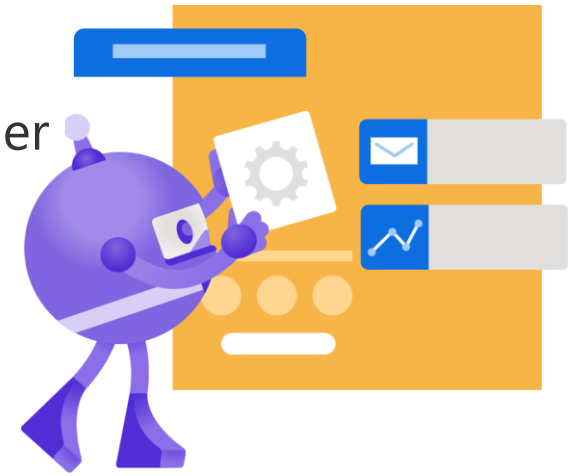
Ladies and Gentlemen, please welcome to …

Flex Consumption Plan



Flex Consumption Plan

- ✓ Flex Consumption is a new Azure Functions hosting option
- ✓ Flex Consumption will be **the best** choice for event-driven serverless functions
- ✓ Features:
  - VNet integration (same of Premium an App Service plans)
  - More control over per-instance concurrency.
  - The platform will elastically scale your functions faster and further to handle your high throughput needs
  - Allows you to optionally configure always ready instances
  - Allows long function execution times
  - Supports zone redundancy



Thank you for your attention!!!

Massimo Bonanni

Microsoft Technical Trainer

massimo.bonanni@microsoft.com
@massimobonanni

aka.ms/maxlinkedin