@bctechdays.com

Join me

Azure Functions Deep Dive for Dynamics 365 Business Central developers: present and future

Stefano Demiliani

BC/NAV TECH DAYS 2023

mibuso.com

Session goals

- Understand the importance of Azure Functions in a Business Central project
- Understand the future of Azure Functions platform
- Understand what you can do more than publishing code in the cloud
- Understand best practices and tricks for efficiently use them in production

Not a lot of AL code here but…

- **a lot of C#**
- **a lot of Azure stuffs**

Why Azure Functions in Business Central projects?

Executing .NET code in a SaaS environment (DotNet variables substitution)

Re-using existing libraries (DLLs) in a SaaS environment

Interacting with Azure Services

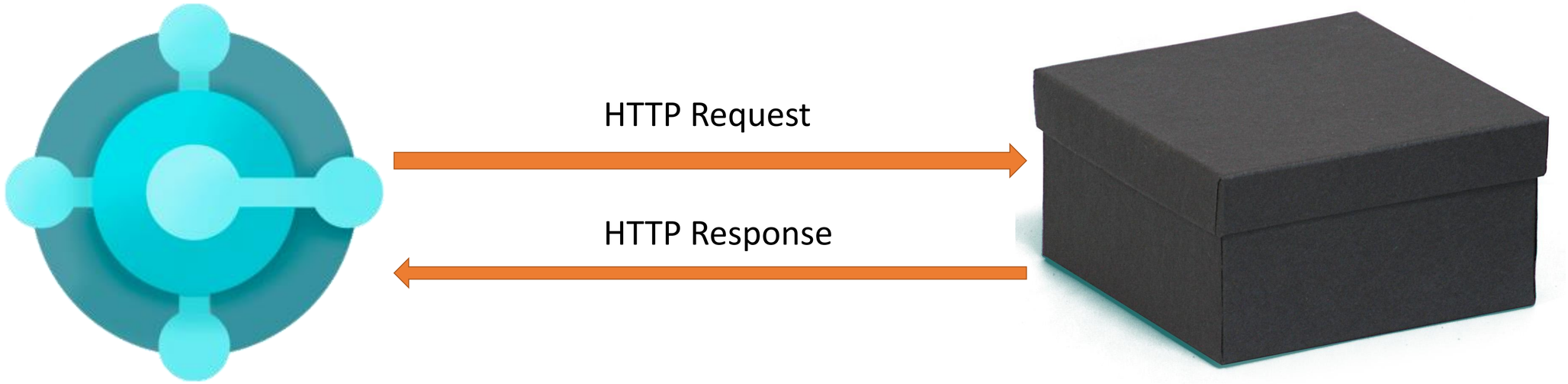
Integrations

Creating serverless processes (workflows, timer based, …)

Layer above Business Central APIs

Executing code in the cloud

Azure Functions is a serverless compute service that lets you run event-triggered code without having to explicitly provision or manage infrastructure.



Azure Function: what is it?

A fully serverless way to run your (non-BC) code: you write the code and easily deploy it, the service does the rest.

- Provide the runtime infrastructure
- Scale automatically (consumption, premium and dedicated plan)
- HTTP triggers, Azure services triggers, 3° party triggers, timer events
- Integrated monitoring with Azure Application Insights
- Automated CI/CD including staging

Azure Functions: anatomy

- **Azure Function App :** collection of one or more Azure Functions. A Function App is the deployment unit for Azure Functions. All functions inside a Function App share the same configuration, such as the runtime version, the application settings, and the storage account.
- **Runtime:** responsible for running your code. The runtime includes logic on how to trigger, log, and manage function executions.
- **Scale Controller:** responsible for managing the number of instances of a function that are running.
- **Application Settings:** configuration values that a function can use at runtime. Application settings can be stored securely in the Azure portal or in a *local.settings.json* file during development
- **Triggers:** events that trigger the execution of an Azure Function. Each function should have only one trigger. Triggers have associated data, which is often provided as the payload of the function. There are several types of triggers available in Azure Functions.

Azure Functions: anatomy

- **Function Code:** code that is executed when the function is triggered. The function code is written in any of the supported programming languages, such as C#, JavaScript, Python, and F#. Azure Functions are designed to be lightweight and short-lived, so the function code should be optimized for performance and efficiency.
- **Bindings:** provide a way to declaratively connect other resources to the function. They are provided to the function as parameters. By using bindings, developers can write less code to interact with external systems, as the binding takes care of the details of connecting to and interacting with the external system.
  - *Input* bindings are used to read data from other resources
  - *output* bindings are used to write data to other resources.

Azure Functions: plans

When you create a function app in Azure, you must choose a **hosting plan** for your app.

There are 3 basic hosting plans available for Azure Functions:

- Consumption plan
- Premium plan
- Dedicated (App Service) plan

Plan	Benefits
Consumption plan	Scale automatically and only pay for compute resources when your functions are running. On the Consumption plan, instances of the Functions host are dynamically added and removed based on the number of incoming events. ✓ Default hosting plan. ✓ Pay only when your functions are running. ✓ Scales automatically, even during periods of high load.
Premium plan	Automatically scales based on demand using pre-warmed workers, which run applications with no delay after being idle, runs on more powerful instances, and connects to virtual networks. Consider the Azure Functions Premium plan in the following situations: ✓ Your function apps run continuously, or nearly continuously. ✓ You have a high number of small executions and a high execution bill, but low GB seconds in the Consumption plan. ✓ You need more CPU or memory options than what is provided by the Consumption plan. ✓ Your code needs to run longer than the maximum execution time allowed on the Consumption plan. ✓ You require features that aren't available on the Consumption plan, such as virtual network connectivity. ✓ You want to provide a custom Linux image on which to run your functions.
Dedicated plan	Run your functions within an App Service plan at regular App Service plan rates. Best for long-running scenarios where Durable Functions can't be used. Consider an App Service plan in the following situations: ✓ You have existing, underutilized VMs that are already running other App Service instances. ✓ Predictive scaling and costs are required.

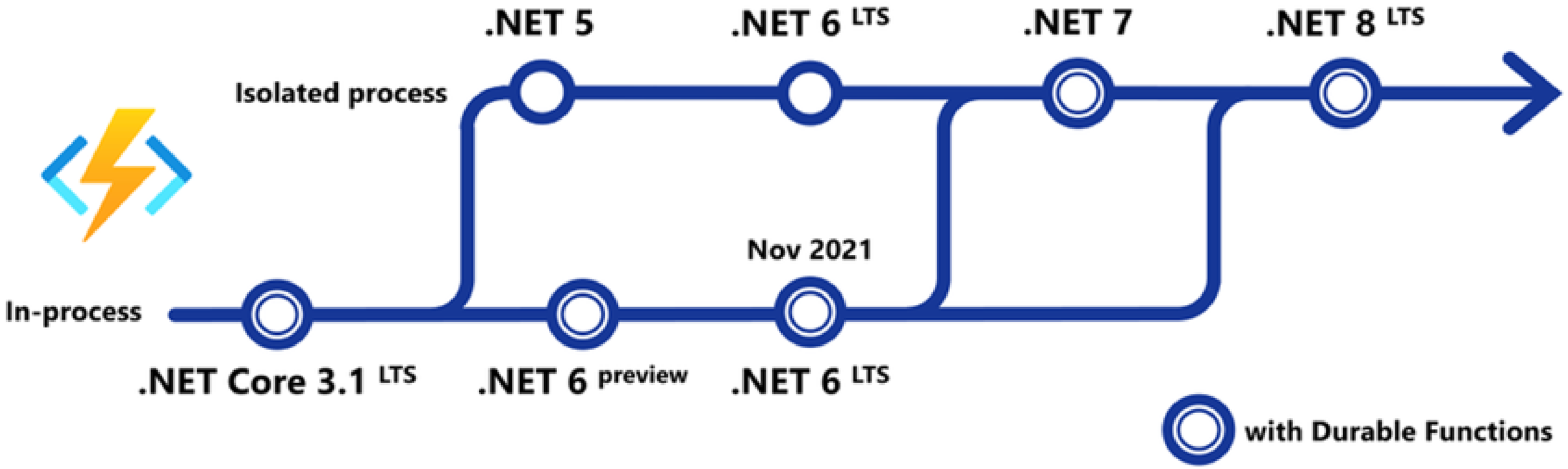
Azure Functions: execution modes

- **In-process mode**: your function code runs in the same .NET process as the host (Azure Functions runtime). In this mode, your code should run on the same framework version used by the runtime.
- **Isolated-process** (or **Out-of-process**) **mode:** it decouples your function code from the Azure Functions runtime, thus letting the users utilize any supported version of .NET, even if it's different from the runtime version.

Isolated mode removed the limitations of in-process execution mode, as it provided the user with the following:

- Full control over how you configure & run your code inside Azure Functions
- Ability to utilize features such as implementing custom Middleware, Logging, etc.
- Encountering fewer conflicts between the code assemblies & the assemblies used by the host process.

Azure Functions: execution modes



Azure Functions: execution modes

DEMO

Using Azure Functions from AL

New Azure Functions System Module

```
codeunit 50100 "SD Azure Functions Demo"
{
    0 references
    procedure SendGetRequest_Example()
    var
        AzureFunctionAuthentication: Codeunit "Azure Functions Authentication";
        AzureFunction: Codeunit "Azure Functions";
        AzureFunctionResponse: Codeunit "Azure Functions Response";
        IAzurefunctionAuthentication: Interface "Azure Functions Authentication";
        QueryDictionary: Dictionary of [Text, Text];
        ResponseTxt: text;
    begin
        // Code authentication example
        IAzurefunctionAuthentication := AzureFunctionAuthentication.CreateCodeAuth('<Function URL>', '<Function Code>');
        // OAuth authentication example
        // IAzurefunctionAuthentication := AzureFunctionAuthentication.CreateOAuth2('<Function URL>', '<Function Code>', '<Client ID>', 'Client Secret', '<OAuthAuthorityUrl>', '<ReturnURL>', 'ResourceURL');
        QueryDictionary.Add('name', 'value');
        AzureFunctionResponse := AzureFunction.SendGetRequest(IAzurefunctionAuthentication, QueryDictionary);
        if AzureFunctionResponse.IsSuccessful() then begin
            AzureFunctionResponse.GetResultAsText(ResponseTxt);
            // Display the response
            Message(ResponseTxt);
        end
        else
            // Display the error message
            Message(AzureFunctionResponse.GetError());
    end;

}
```

Supports *Function* or *OAuth2* authorization levels.

Azure Functions: calling from AL

DEMO

Azure Functions should be stateless

An Azure Function should be stateless as there's no control over where and when function instances are provisioned and de-provisioned.

Managing and storing data/state between requests can lead to inconsistencies.

If, for any reason, you need to have a stateful function, consider using the *Durable Functions* extension of Azure Functions (not covered in this session).

```
0 references
public static class Function1
{
    private static int RequestNumber = 0;

    [FunctionName("Function1")]
    0 references
    public static async Task<HttpResponseMessage> Run([HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)]HttpRequestMessage req, TraceWriter log)
    {
        log.Info("C# HTTP trigger function processed a request.");

        RequestNumber++;
        log.Info($"Number of POST requests is {RequestNumber}.");

        // parse query parameter
        string name = req.GetQueryNameValuePairs()
            .FirstOrDefault(q => string.Compare(q.Key, "name", true) == 0)
            .Value;

        if (name == null)
        {
            // Get request body
            dynamic data = await req.Content.ReadAsAsync<object>();
            name = data?.name;
        }

        return name == null
            ? req.CreateResponse(HttpStatusCode.BadRequest, "Please pass a name on the query string or in the request body")
            : req.CreateResponse(HttpStatusCode.OK, "Hello " + name);
    }
}
```

Using HttpClient in Azure Functions

Simple usage of **HttpClient** to make HTTP requests presents several issues, including vulnerability to socket exhaustion.

In a Function app, calling the *HttpClient* constructor in the body of a function method will create a new instance with every function invocation, amplifying these issues.

For apps running on a *Consumption* hosting plan, inefficient *HttpClient* usage can exhaust the plan's outbound connection limits.

The recommended best practice is to use an **[IHttpClientFactory]** with dependency injection or a single static *HttpClient* instance, depending on the nature of your application.

Using HttpClient in Azure Functions

BC/NAV TECH DAYS 2023
mibuso.com

DEMO

Azure Functions: timeouts

The timeout duration for functions in a function app is defined by the **functionTimeout** property in the *host.json* project file.

Plan	Default	Maximum[1]
Consumption plan	5	10
Premium plan	30	Unlimited
Dedicated plan	30	Unlimited

Regardless of the function app timeout setting, 230 seconds is the maximum amount of time that an HTTP triggered function can take to respond to a request. This is because of the default idle timeout of Azure Load Balancer.

Azure Functions: cold start

Consumption plan	Apps may scale to zero when idle, meaning some requests may have additional latency at startup. The consumption plan does have some optimizations to help decrease cold start time, including pulling from pre-warmed placeholder functions that already have the function host and language processes running.
Premium plan	Perpetually warm instances to avoid any cold start.
Dedicated plan	When running in a Dedicated plan, the Functions host can run continuously, which means that cold start isn't really an issue.

Timer Triggered AF and CRON

Timer Triggered functions have CRON expression embedded in code:

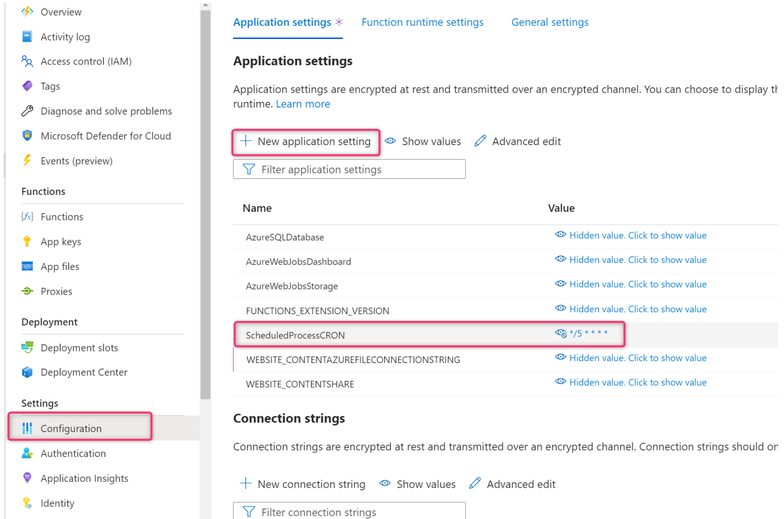
```
[FunctionName("ScheduledProcess")]
0 references
public static void Run([TimerTrigger("*/5 * * * *")] TimerInfo myTimer, ILogger log)
{
    log.LogInformation($"ScheduledProcess Timer trigger function executed at UTC: {DateTime.UtcNow}");
    // Your code here...
}
```

Make them dynamic:

```
[FunctionName("ScheduledProcess")]
0 references
public static void Run([TimerTrigger("%ScheduledProcessCRON%")] TimerInfo myTimer, ILogger log)
{
    log.LogInformation($"ScheduledProcess Timer trigger function executed at UTC: {DateTime.UtcNow}");
    // Your code here...
}
```

Schema: https://json.schemastore.org/local.settings.json

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "
    "FUNCTIONS_WORKER_RUNTIME": "dotnet",
    "ScheduledProcessCRON": "*/2 * * * *"
  }
}
```

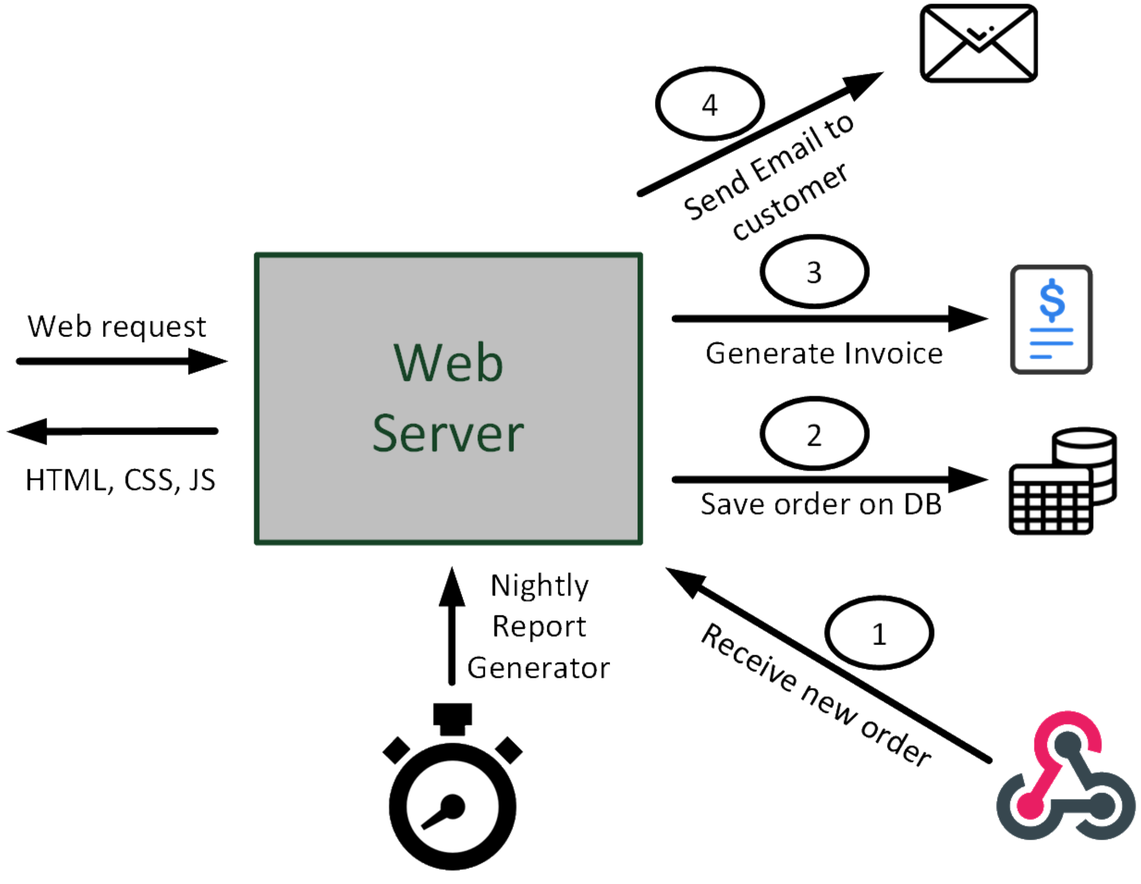


Azure Functions Bindings

Bindings: provide a way to declaratively connect other resources to the function (you declare the data sources to read and write, and let Azure Functions take care of the rest.)

Azure Functions Bindings: an example

- The customer makes a POST call to endpoint to place an order
- The solution checks if the order is valid and then responds immediately to the customers.
- The solution saves the order in a database (in the demo we use a storage table).
- The solution creates the invoice in a blob object.
- The solution sends a mail to the customer with the invoice attached
- The solution implements a timer-triggered process that retrieves the orders received during tha day and creates a daily report.

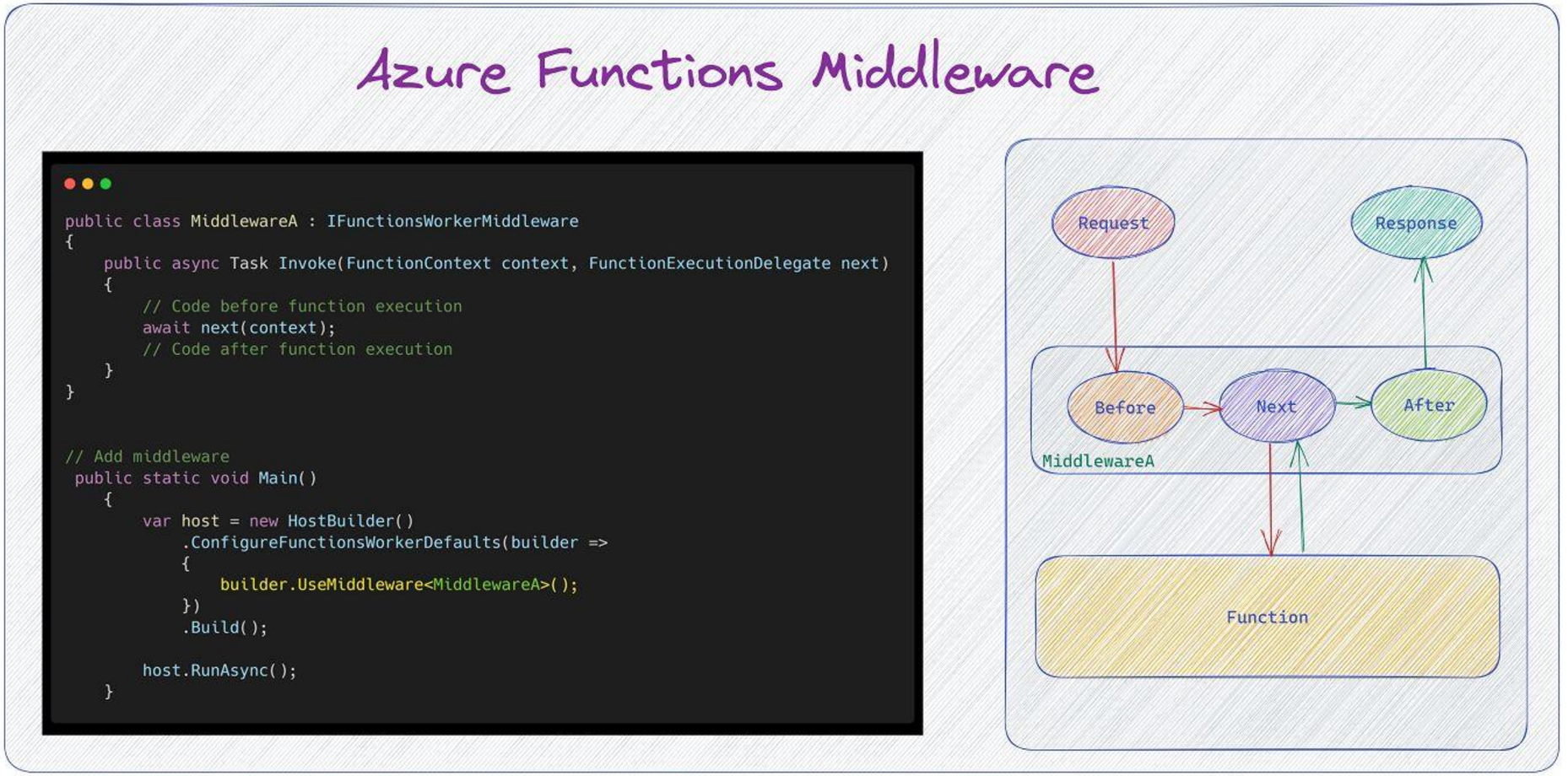


Azure Functions: bindings

DEMO

Azure Functions: middlewares

Azure Functions in the isolated model supports **middlewares**.
A *middleware* acts as a pipeline through which each incoming HTTP request passes, and it can perform various tasks such as processing requests, handling responses, and modifying the behavior of the request/response pipeline.

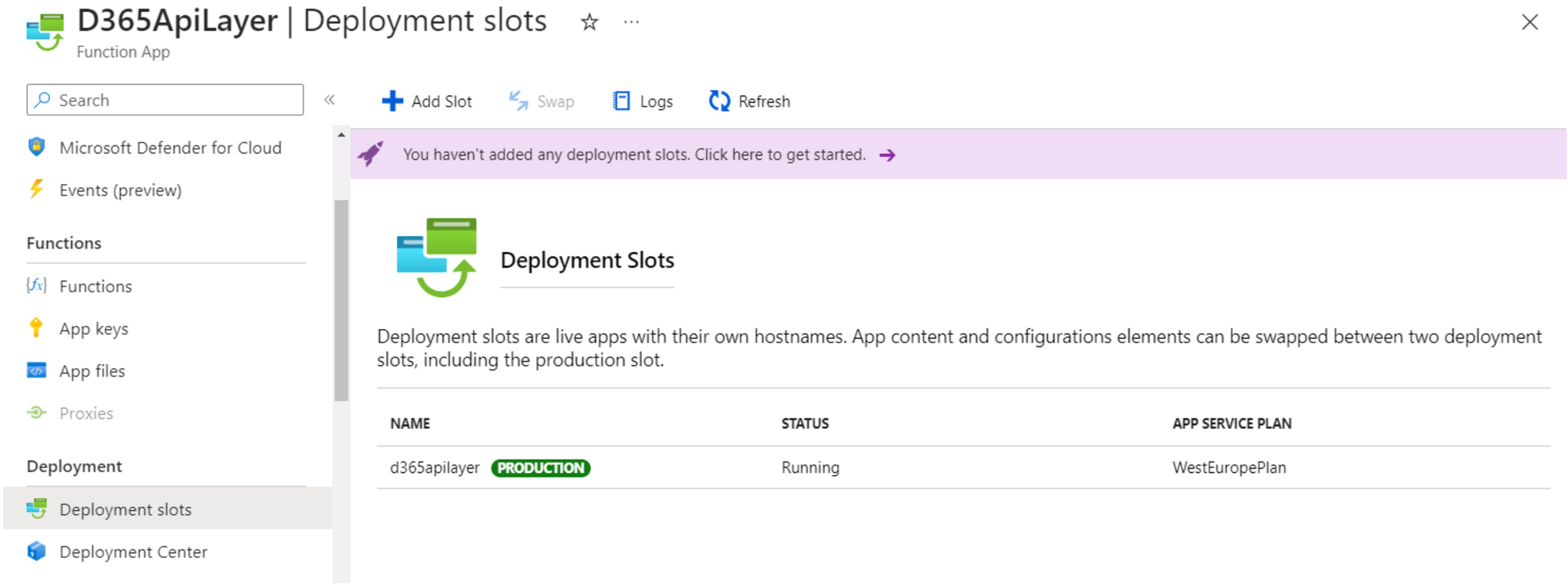


Azure Functions: middlewares

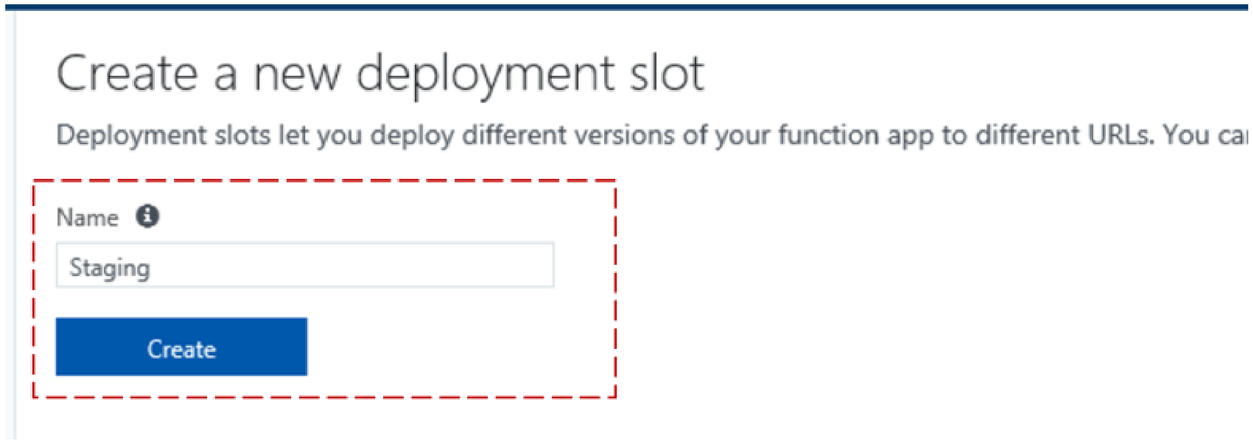
DEMO

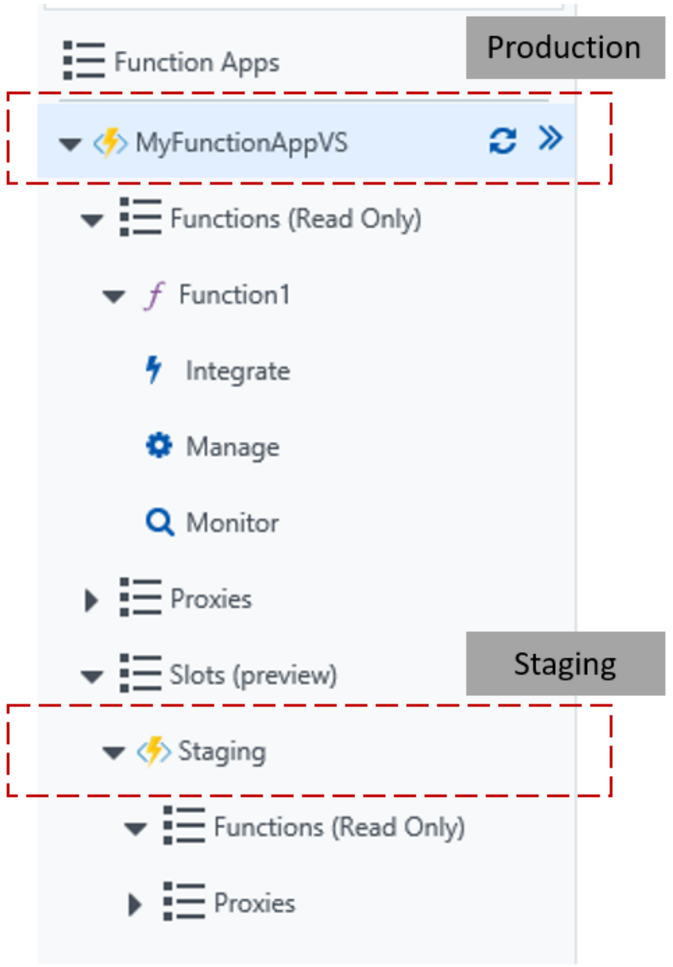
Azure Functions: deployment slots

Azure Functions **deployment slots** allow your function app to run different instances called *slots*.
Slots are different environments exposed via a publicly available endpoint.
One app instance is always mapped to the production slot, and you can swap instances assigned to a slot on demand.
Function apps running under the *App Service* plan may have multiple slots, while under the *Consumption* plan only one slot is allowed.

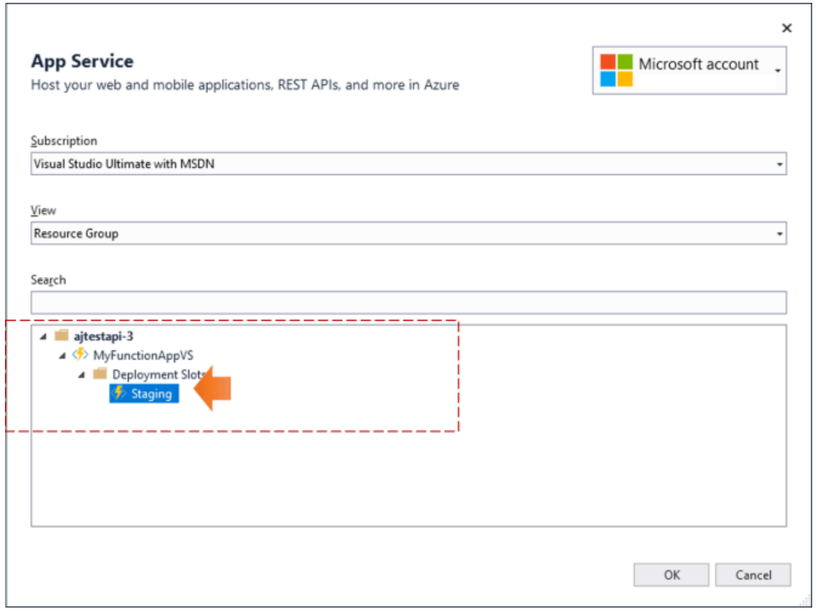


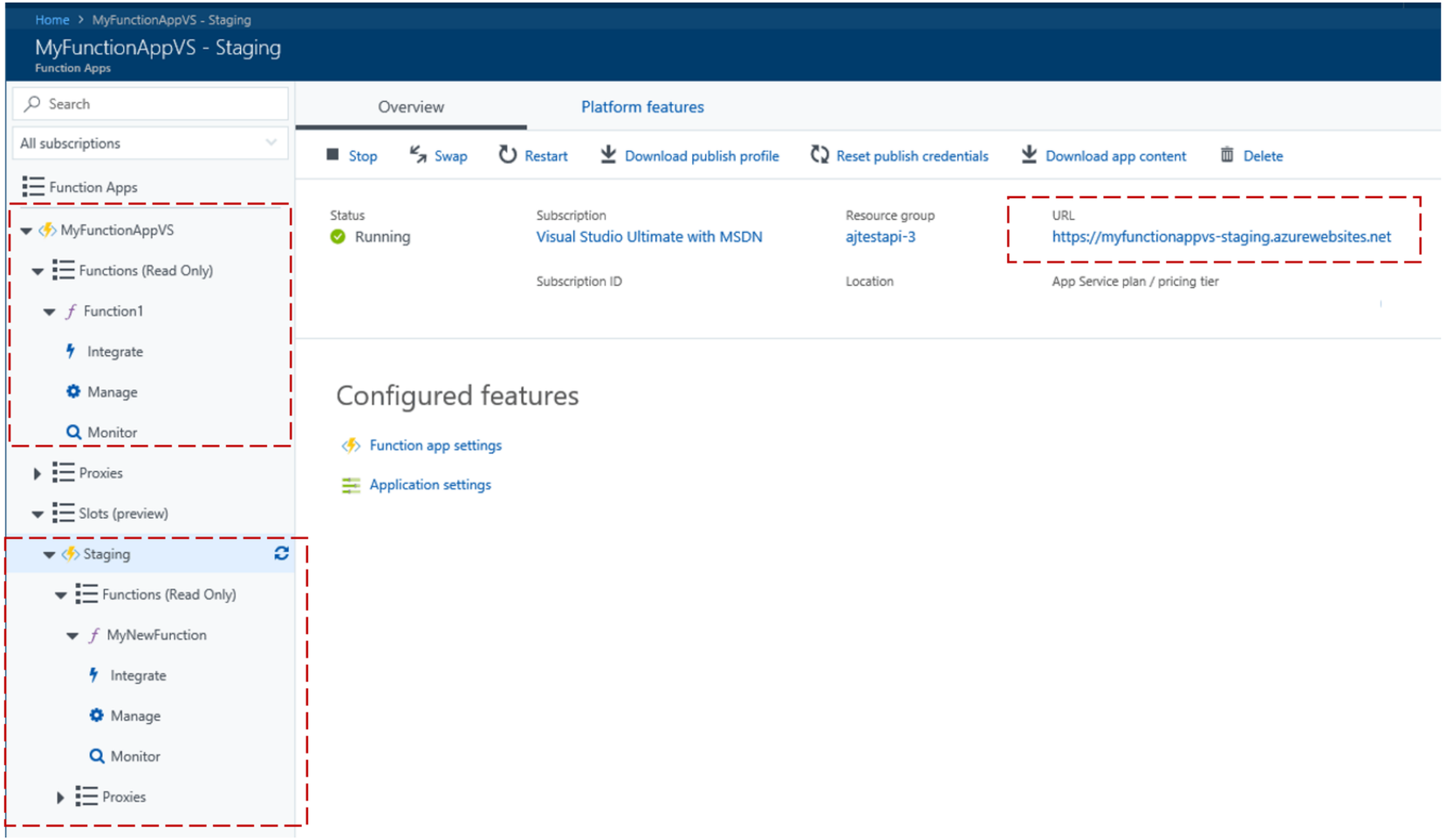
Azure Functions: deployment slots





Azure Functions: deployment slots





Continuous Deployment

You can use Azure Functions to deploy your code continuously by using source control integration. Source control integration activates a workflow in which a code update triggers deployment to Azure.

NOTE:

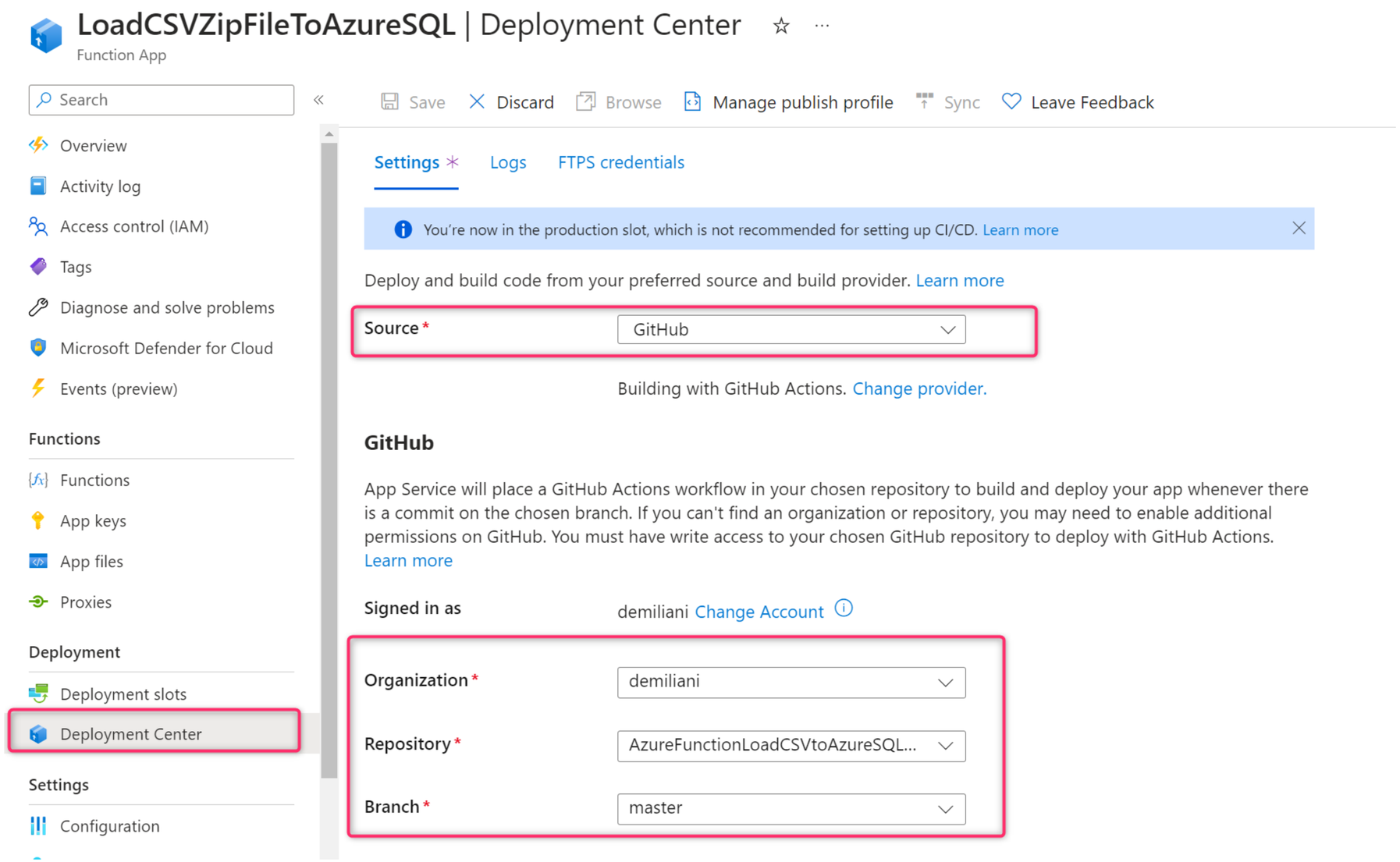
- The unit of deployment for functions in Azure is the function app. All functions in a function app are deployed at the same time.
- After you enable continuous deployment, access to function code in the Azure portal is configured as *read-only* because the source of truth is set to be elsewhere.

Continuous deployment should never be enabled for your production slot:

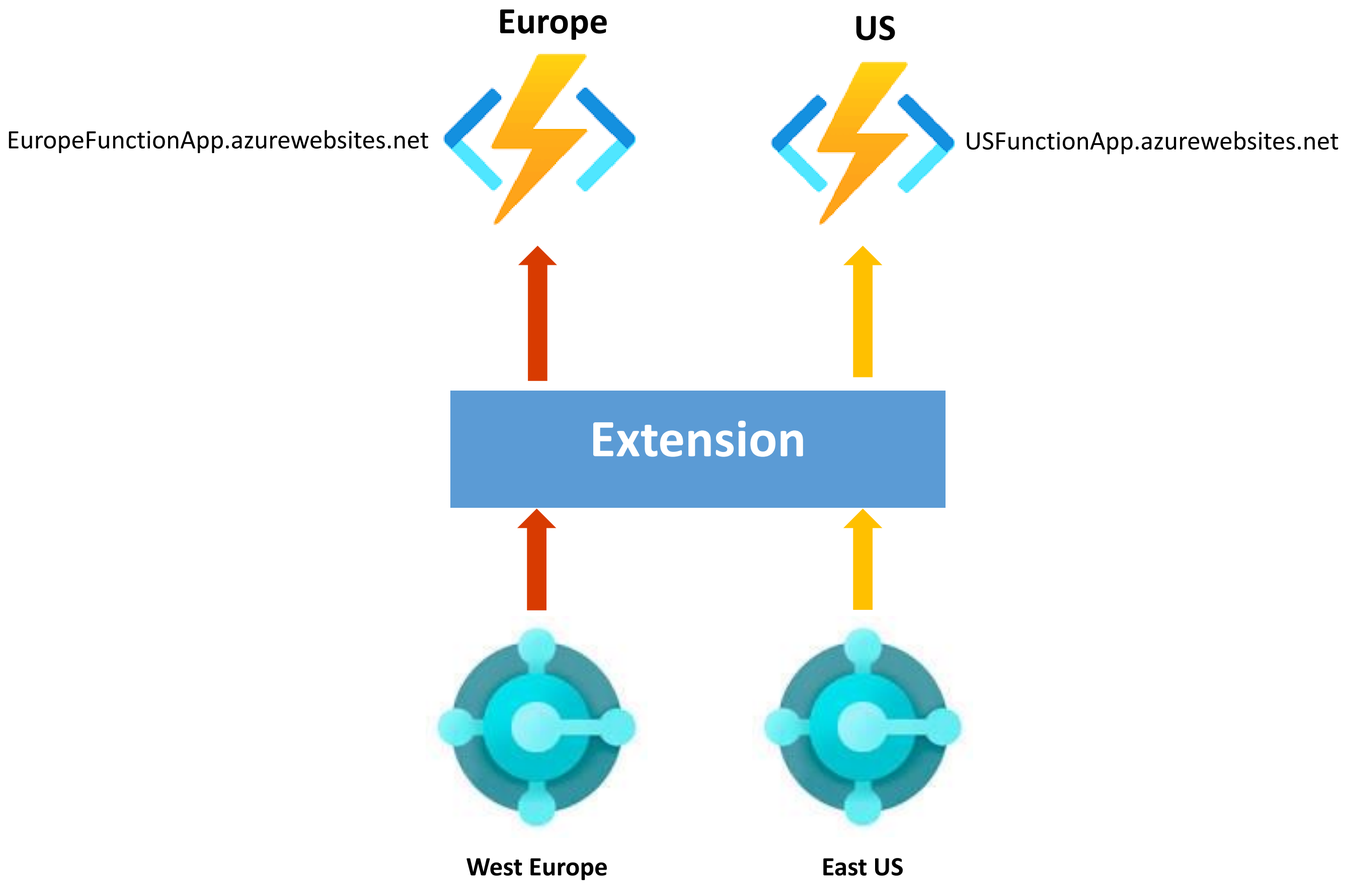
- your production branch (*main*) should be deployed onto a non-production slot.
- when you are ready to release the base branch, swap it into the production slot.

Swapping into production (instead of deploying to production) prevents downtime and allows you to roll back the changes by swapping again.

Continuous Deployment



Azure Functions: high availability and traffic control



Azure Functions: high availability and traffic control

BC/NAV TECH DAYS 2023
mibuso.com

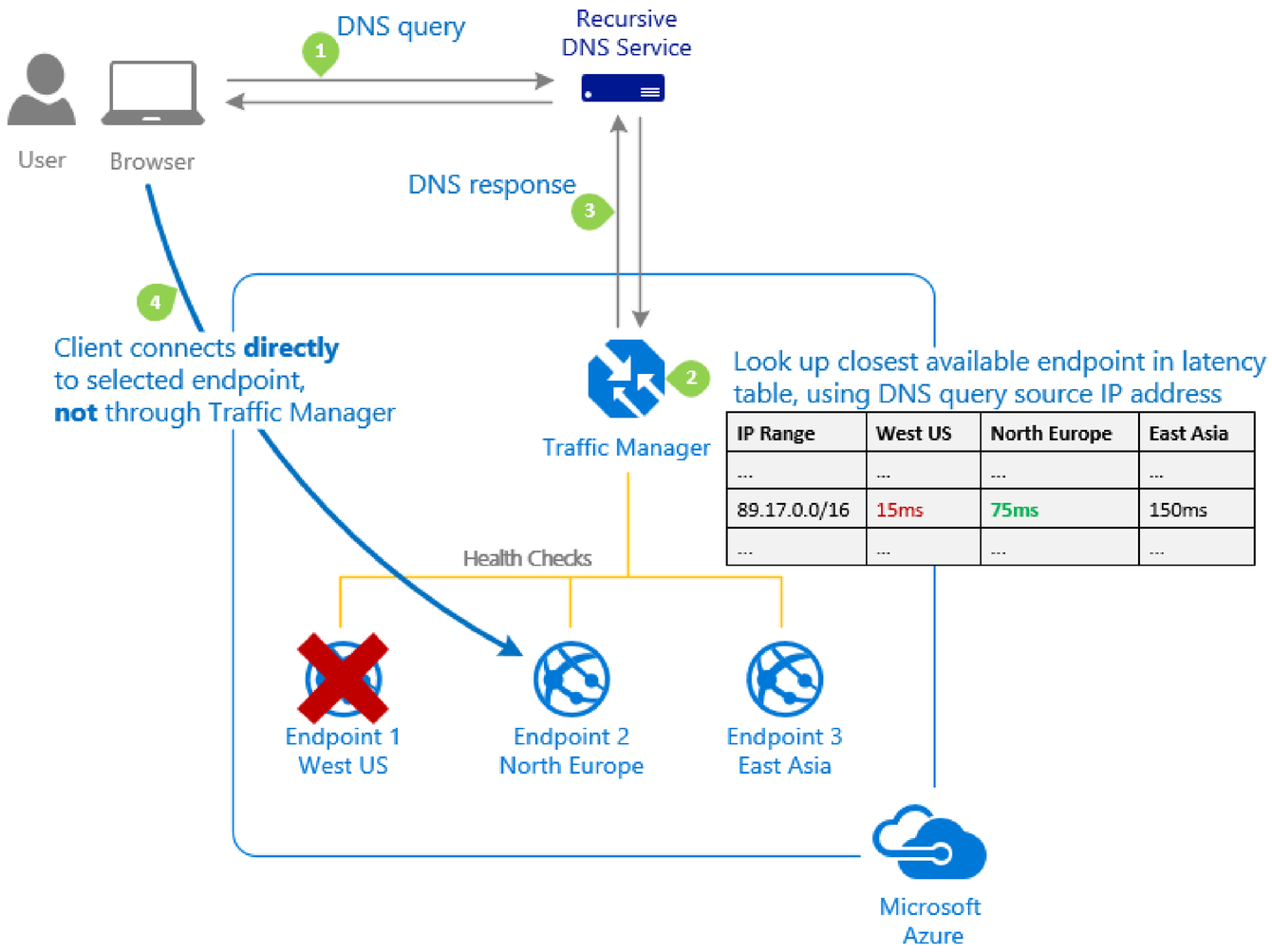
Azure Traffic Manager is a DNS-based traffic load balancer. This service allows you to distribute traffic to your public facing applications across the global Azure regions. Traffic Manager also provides your public endpoints with high availability and quick responsiveness.

The following traffic routing methods are available in Traffic Manager:

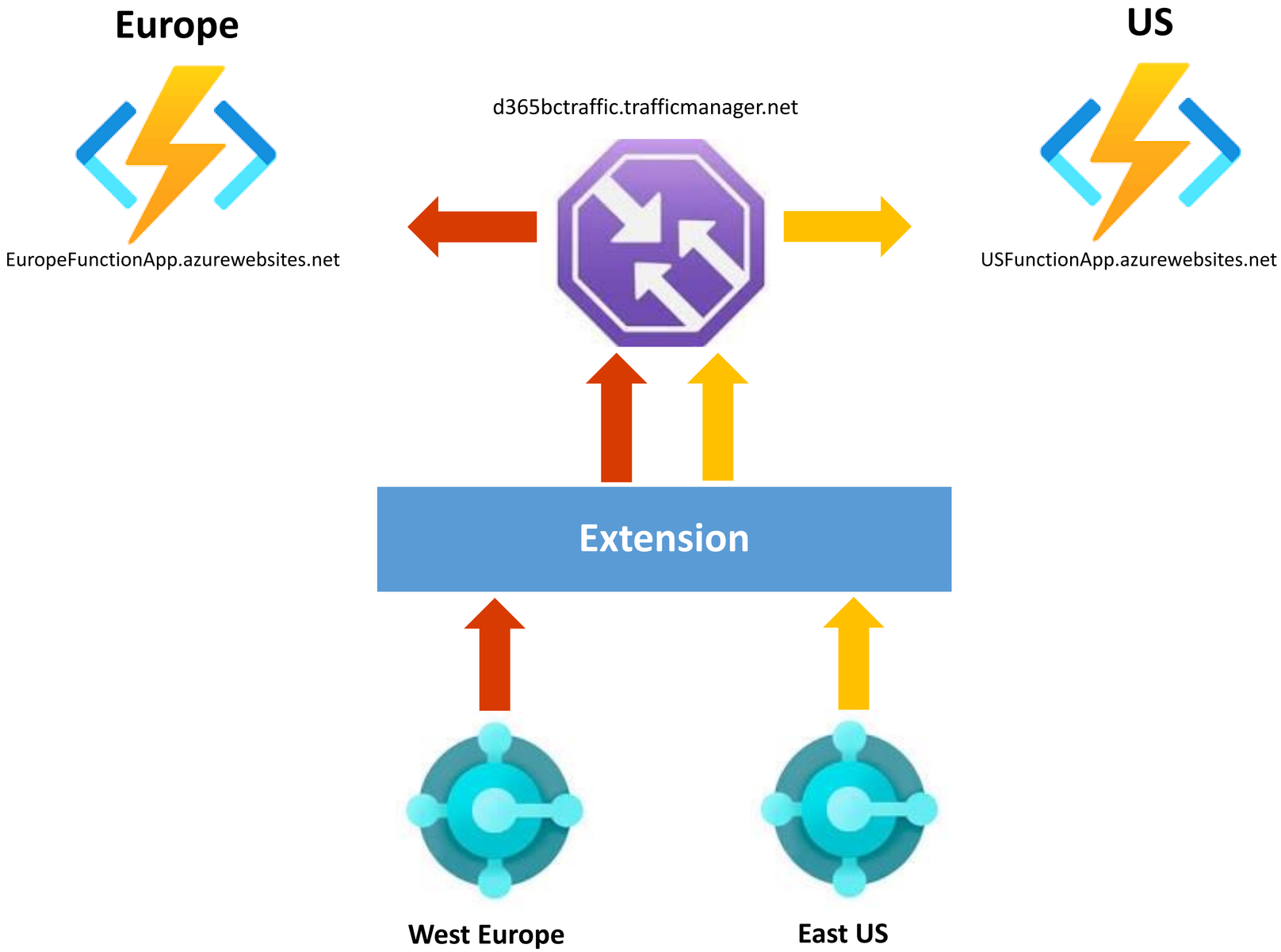
- **Priority:** Select **Priority** routing when you want to have a primary service endpoint for all traffic. You can provide multiple backup endpoints in case the primary or one of the backup endpoints is unavailable.
- **Weighted:** Select **Weighted** routing when you want to distribute traffic across a set of endpoints based on their weight. Set the weight the same to distribute evenly across all endpoints.
- **Performance:** Select **Performance** routing when you have endpoints in different geographic locations and you want end users to use the "closest" endpoint for the lowest network latency.
- **Geographic:** Select **Geographic** routing to direct users to specific endpoints (Azure, External, or Nested) based on where their DNS queries originate from geographically. With this routing method, it enables you to be in compliance with scenarios such as data sovereignty mandates, localization of content & user experience and measuring traffic from different regions.
- **Multivalue:** Select **MultiValue** for Traffic Manager profiles that can only have IPv4/IPv6 addresses as endpoints. When a query is received for this profile, all healthy endpoints are returned.
- **Subnet:** Select **Subnet** traffic-routing method to map sets of end-user IP address ranges to a specific endpoint. When a request is received, the endpoint returned will be the one mapped for that request's source IP address.

Azure Functions: high availability and traffic control

EXAMPLE: Performance traffic routing method



Azure Functions: high availability and traffic control



Azure Functions: high availability

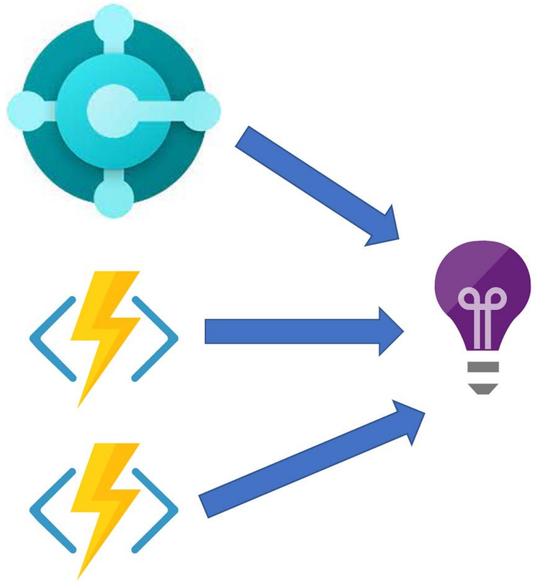
BC/NAV TECH DAYS 2023
mibuso.com

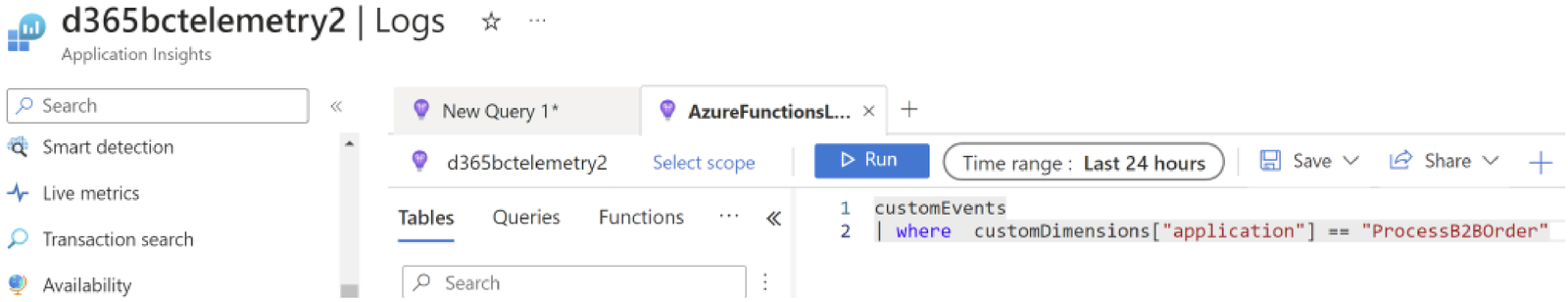
DEMO

Azure Functions monitoring

Azure Functions offers built-in integration with **Azure Application Insights** to monitor functions executions.

If you have AF tightly-coupled with BC processes:



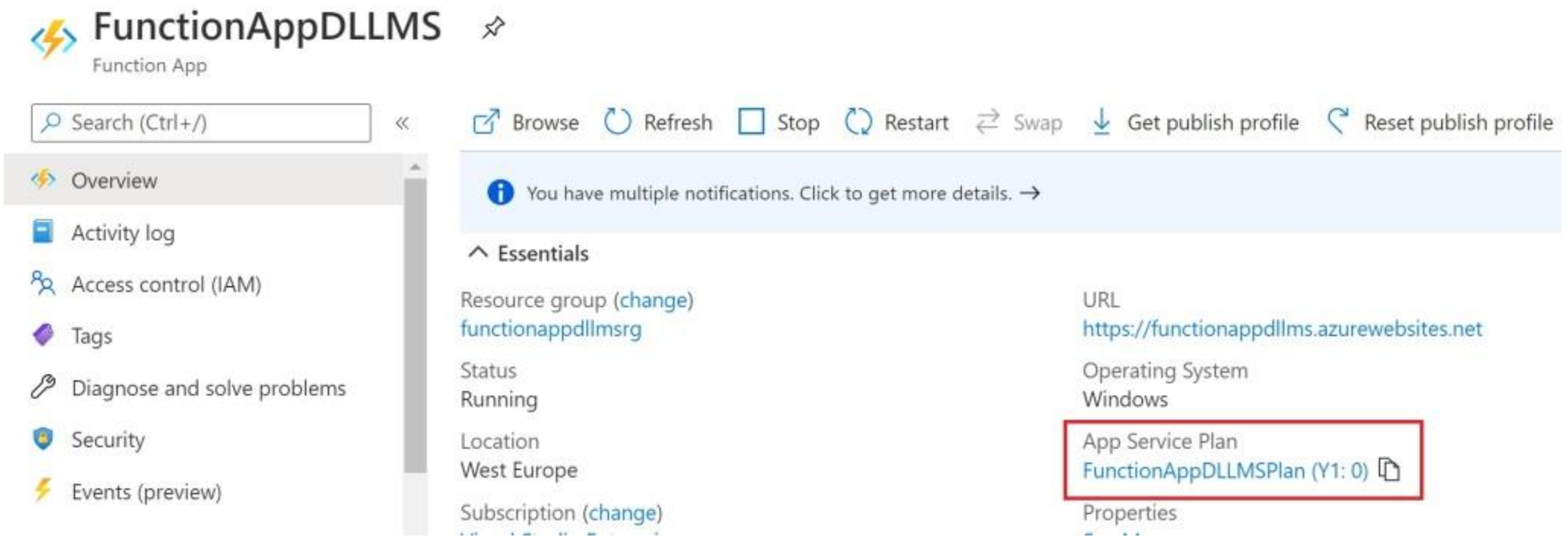


Azure Functions: monitoring

DEMO

Azure Functions: plan upgrade

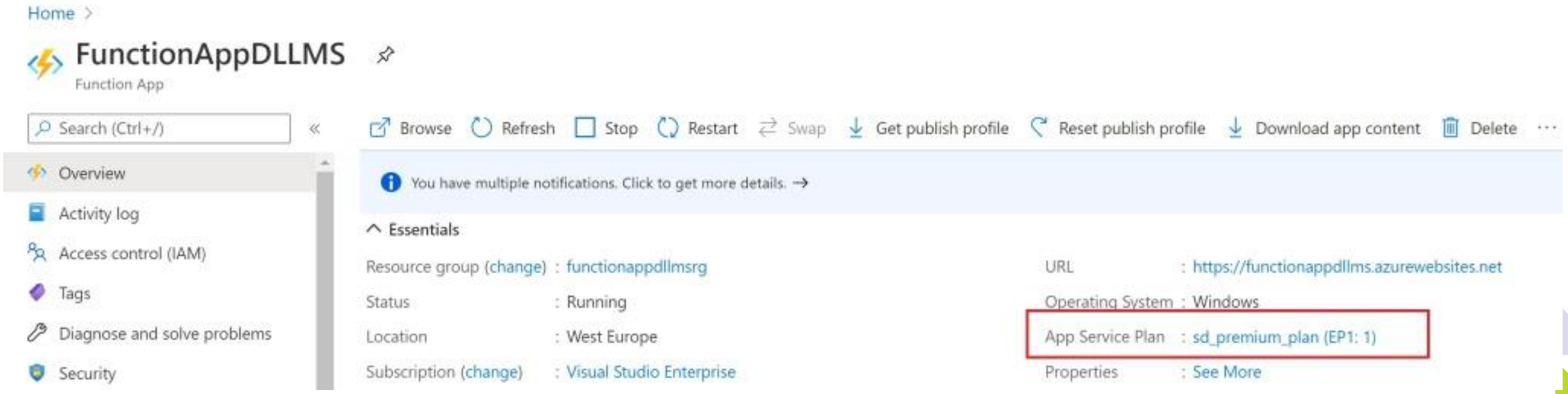
Function app running in *Consumption* plan:



1. Create a *Premium* plan with the type and resources you want
2. Move the Function app to the newly created *Premium* plan
3. Scale back down the Function app to the *Consumption* plan at the end of the period of work you need
4. Delete the *Premium* plan (don't forget to do this!)

Azure Functions: plan upgrade

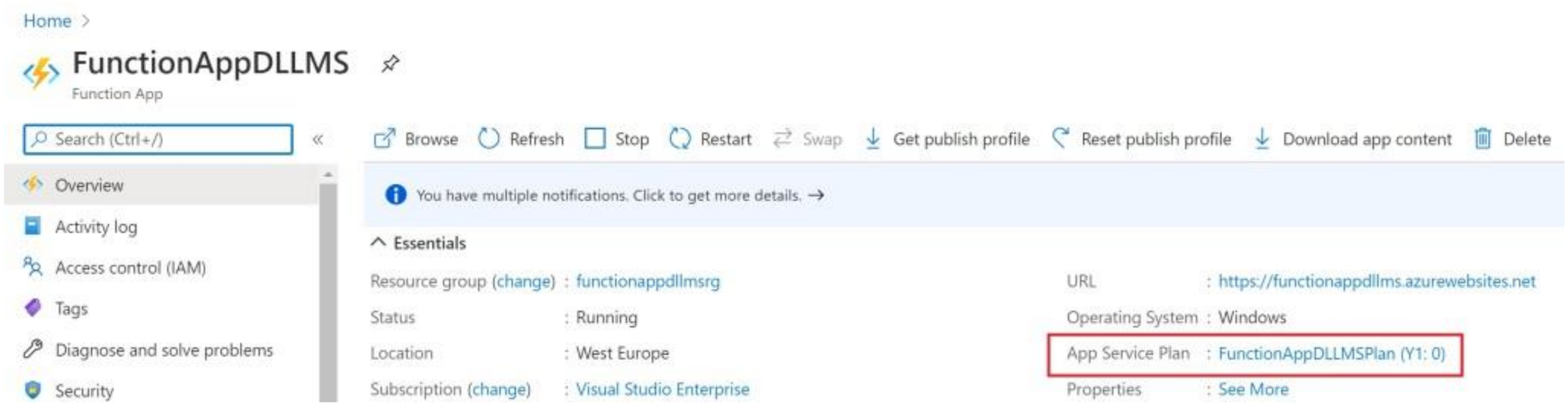
```
$resourceGroup = 'functionappdllmsrg'
$functionAppName = 'FunctionAppDLLMS'
$consumptionPlanName = 'FunctionAppDLLMSPlan'
$premiumPlanName = 'sd_premium_plan'

az functionapp plan create --name $premiumPlanName --sku EP1 --resource-group $resourceGroup --location 'West Europe'


az functionapp update --name $functionAppName --resource-group $resourceGroup --plan $premiumPlanName
```



Azure Functions: plan upgrade

BC/NAV TECH DAYS 2023
mibuso.com

```
az functionapp update --name $functionAppName --resource-group $resourceGroup --plan $consumptionPlanName
```

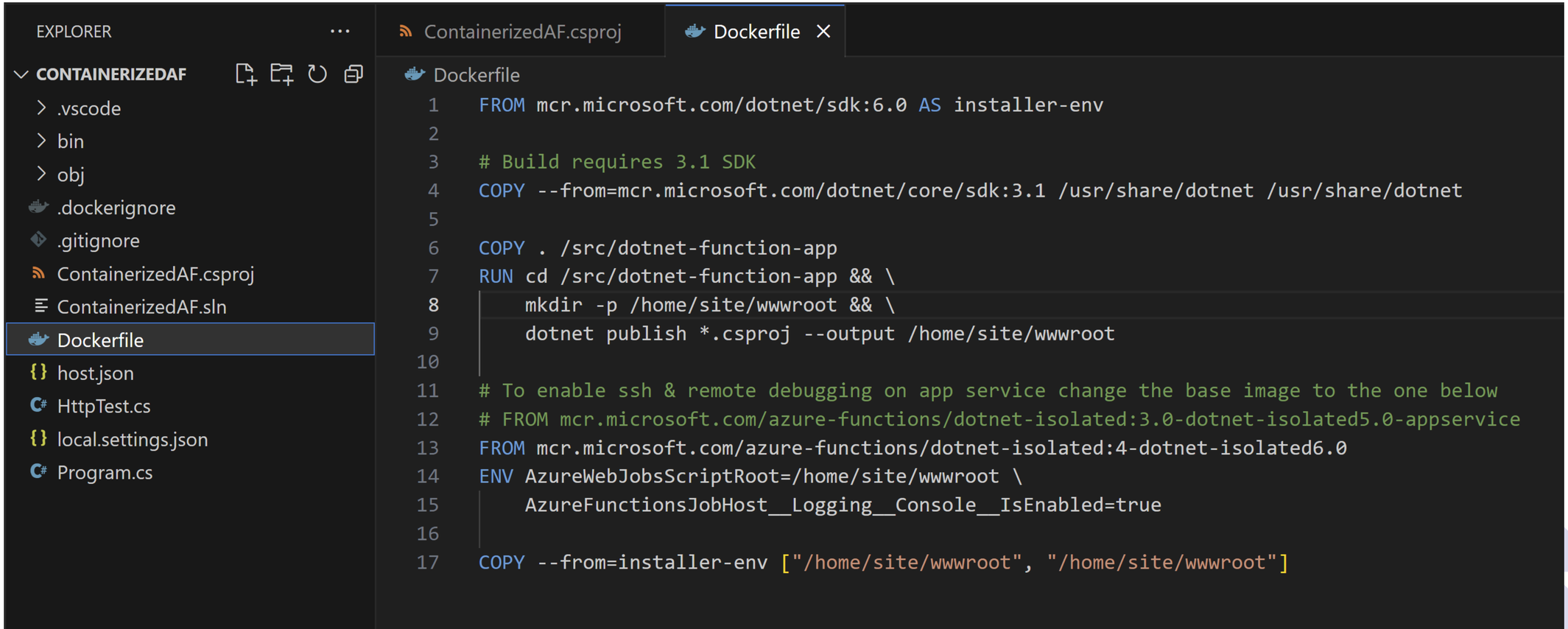



```
az functionapp plan delete --resource-group $resourceGroup --name $premiumPlanName
```

You can move a function app to another App Service plan as long as the source plan and the target plan are in the same resource group, region and OS type.

Running Azure Functions on Docker

- func init ContainerizedAF --worker-runtime dotnet-isolated --docker --target-framework net6.o
- func new --name HttpTest --template "HTTP trigger" --authlevel "anonymous"



Running Azure Functions on Docker

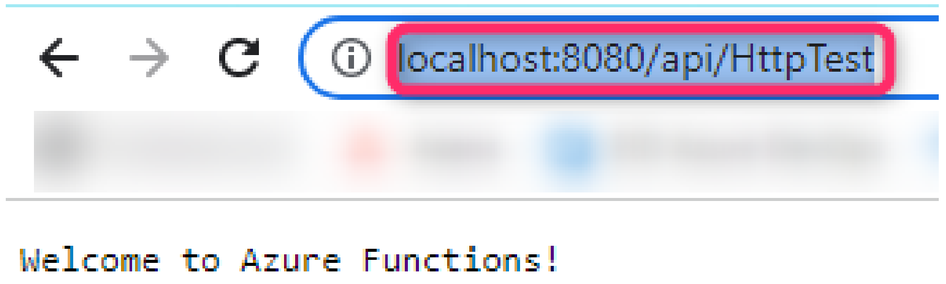
docker build --platform linux --tag <DOCKERID>/ContainerizedAF:v1.0.0 .

```
C:\Users\stefano\OneDrive\MS Workshop NAV D365BC\EVENTI\2023-06-22 BC Techdays 2023\ContainerizedAF>docker build --platform linux --tag demiliani/containerizedaf:v1.0.0 .
[+] Building 198.0s (13/13) FINISHED
 => [internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 821B                                                                                0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 34B                                                                                    0.0s
 => [internal] load metadata for mcr.microsoft.com/dotnet/sdk:6.0                                                   0.7s
 => [internal] load metadata for mcr.microsoft.com/azure-functions/dotnet-isolated:4-dotnet-isolated6.0             0.7s
 => CACHED FROM mcr.microsoft.com/dotnet/core/sdk:3.1                                                               0.0s
 => => resolve mcr.microsoft.com/dotnet/core/sdk:3.1                                                                0.1s
 => [stage-1 1/2] FROM mcr.microsoft.com/azure-functions/dotnet-isolated:4-dotnet-isolated6.0@sha256:45f2e3e58f7   81.7s
 => => resolve mcr.microsoft.com/azure-functions/dotnet-isolated:4-dotnet-isolated6.0@sha256:45f2e3e58f77061d821a   0.0s
 => => sha256:ec93b7813674e14a30106f78bdafc07955353b2855d34353a7befaa0b5e379ed 4.61kB / 4.61kB                      0.0s
 => => sha256:19c563ebbf97ae68f79f9462883c9c83ba924b52b67345e77c3164766e640153 31.63MB / 31.63MB                   12.5s
 => => sha256:45f2e3e58f77061d821adb1aa49c603e4b3d1a52a662689d0c3d4d9ecf6538f9 1.80kB / 1.80kB                      0.0s
 => => sha256:a71ed65a9b46e92cca539fe97f54a4e58ce5dfd97c492a24e2bbd11d8be5ce31 155B / 155B                          0.3s
 => => sha256:e9e927c4e8e4bd075d7f35ebc2be4a59db651b1f3ecf4c548aa5d93a73800ba5 24.76MB / 24.76MB                   10.6s
 => => sha256:fe3d1148fa14a01bc34640be0a5ed7860cce07eeaca47bbd1e639e4560728b3a 100.16MB / 100.16MB                 43.7s
 => => sha256:d162b0f460714059124501 6b7bfea4f2e9d313bdb99656d10640126f5407181f 9.46MB / 9.46MB                    15.8s
 => => extracting sha256:19c563ebbf97ae68f79f9462883c9c83ba924b52b67345e77c3164766e640153                           6.0s
 => => extracting sha256:a71ed65a9b46e92cca539fe97f54a4e58ce5dfd97c492a24e2bbd11d8be5ce31                           0.0s
 => => extracting sha256:e9e927c4e8e4bd075d7f35ebc2be4a59db651b1f3ecf4c548aa5d93a73800ba5                           5.7s
 => => extracting sha256:fe3d1148fa14a01bc34640be0a5ed7860cce07eeaca47bbd1e639e4560728b3a                          34.3s
 => => extracting sha256:d162b0f460714059124501 6b7bfea4f2e9d313bdb99656d10640126f5407181f                          2.4s
 => [installer-env 1/4] FROM mcr.microsoft.com/dotnet/sdk:6.0@sha256:a3bbff689a86ba7f3ddcee5089a729b20e20e3b4dbf  99.5s
 => => resolve mcr.microsoft.com/dotnet/sdk:6.0@sha256:a3bbff689a86ba7f3ddcee5089a729b20e20e3b4dbfb9d0a43bb3284d9  0.0s
 => => sha256:a3bbff689a86ba7f3ddcee5089a729b20e20e3b4dbfb9d0a43bb3284d9081023 1.82kB / 1.82kB                      0.0s
 => => sha256:c4631f0689929f4195743b9fa4bba3a4661b6f3ffd25513f5541d98bfdeb0ee2 2.01kB / 2.01kB                      0.0s
 => => sha256:d5ba51a50ad2b19d4f1299ef7a7347121079e2e36af873521c59ce57dba89d00 7.17kB / 7.17kB                      0.0s
 => => sha256:9e65f86790b6536830d80a8375d99d246155964368261d04852bb23fb2625d3b 31.65MB / 31.65MB                   26.4s
 => => sha256:217953d5b22071bd9537899b2fca68afd4ab9c47bec1b0dffde674e0476048f8 154B / 154B                         16.1s
 => => sha256:9abf5ceb3cbb36260f824678219b587f2fc8b56b540e284cd1046d090fde73f8 9.46MB / 9.46MB                     19.4s
 => => sha256:b14bfd45ff9c8a47bec3b136a1e7daf8245869e67f9790532a511273c72dcae4 25.37MB / 25.37MB                   30.6s
 => => sha256:c54b90e5d4b71517e37767e3d90f4b3012e81c786efbd133c8e99584b4145167 148.73MB / 148.73MB                 67.3s
 => => extracting sha256:9e65f86790b6536830d80a8375d99d246155964368261d04852bb23fb2625d3b                           5.9s
 => => sha256:21e7e09b3ced73c38640bc578950c6d931c6e8587177bc247f47527b7818723f 13.61MB / 13.61MB                   38.4s
 => => extracting sha256:217953d5b22071bd9537899b2fca68afd4ab9c47bec1b0dffde674e0476048f8                           0.0s
 => => extracting sha256:9abf5ceb3cbb36260f824678219b587f2fc8b56b540e284cd1046d090fde73f8                           1.9s
 => => extracting sha256:b14bfd45ff9c8a47bec3b136a1e7daf8245869e67f9790532a511273c72dcae4                           7.2s
 => => extracting sha256:c54b90e5d4b71517e37767e3d90f4b3012e81c786efbd133c8e99584b4145167                          25.9s
 => => extracting sha256:21e7e09b3ced73c38640bc578950c6d931c6e8587177bc247f47527b7818723f                           3.0s
 => [internal] load build context                                                                                   0.1s
 => => transferring context: 8.37kB                                                                                 0.1s
 => [installer-env 2/4] COPY --from=mcr.microsoft.com/dotnet/core/sdk:3.1 /usr/share/dotnet /usr/share/dotnet      13.6s
 => [installer-env 3/4] COPY . /src/dotnet-function-app                                                             0.3s
 => [installer-env 4/4] RUN cd /src/dotnet-function-app &&     mkdir -p /home/site/wwwroot &&     dotnet publish  83.4s
 => [stage-1 2/2] COPY --from=installer-env [/home/site/wwwroot, /home/site/wwwroot]                                0.1s
 => exporting to image                                                                                              0.2s
 => => exporting layers                                                                                             0.1s
 => => writing image sha256:be48b75f69eec46a89ae7d4f10e0f0b27acd27a746141e4e972b75823234a461                        0.0s
 => => naming to docker.io/demiliani/containerizedaf:v1.0.0                                                         0.0s
```

Running Azure Functions on Docker

docker run -p 8080:80 -it <DOCKERID>/ContainerizedAF:v1.0.0

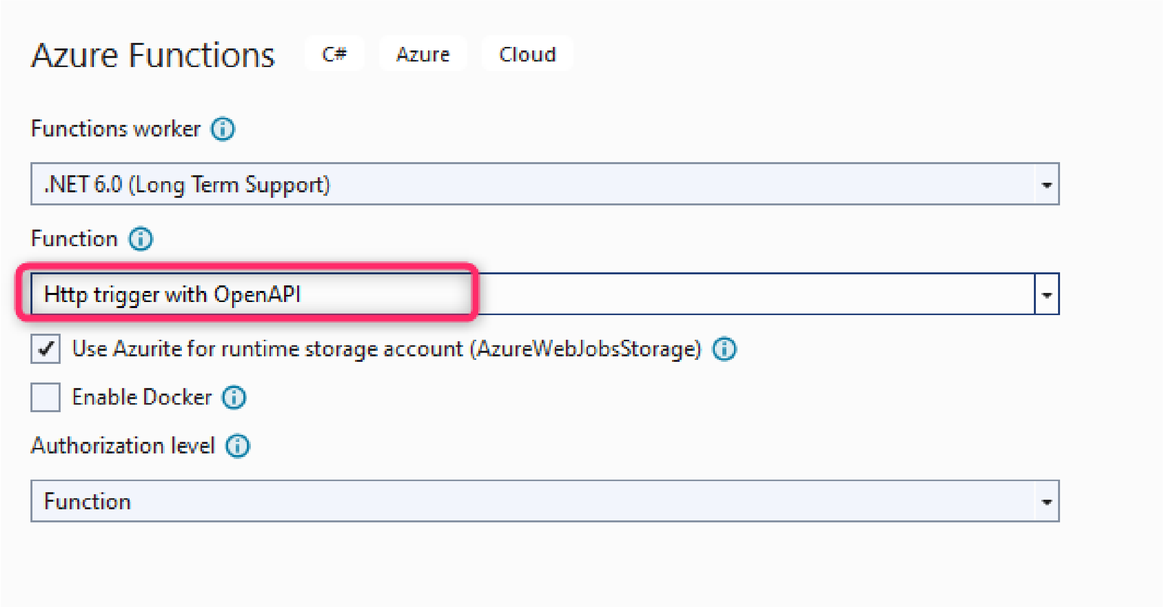
```
      Loading functions metadata
info: Host.Startup[326]
      Reading functions metadata
info: Host.Startup[327]
      2 functions found
info: Host.Startup[315]
      1 functions loaded
info: Host.Startup[0]
      Generating 1 job function(s)
info: Host.Startup[0]
      Found the following functions:
      Host.Functions.HttpTest

info: Microsoft.Azure.WebJobs.Script.WebHost.WebScriptHostHttpRoutesManager[0]
      Initializing function HTTP routes
      Mapped function route 'api/HttpTest' [get,post] to 'HttpTest'

info: Microsoft.Azure.WebJobs.Hosting.OptionsLoggingService[0]
      HttpOptions
      {
        "DynamicThrottlesEnabled": false,
        "EnableChunkedRequestBinding": false,
        "MaxConcurrentRequests": -1,
        "MaxOutstandingRequests": -1,
        "RoutePrefix": "api"
      }
info: Host.Startup[412]
      Host initialized (163ms)
info: Host.Startup[413]
      Host started (179ms)
info: Host.Startup[0]
      Job host started
Hosting environment: Production
Content root path: /azure-functions-host
Now listening on: http://[::]:80
Application started. Press Ctrl+C to shut down.
info: Microsoft.Azure.WebJobs.Script.Workers.Rpc.RpcFunctionInvocationDispatcher[0]
      Worker process started and initialized.
```



docker push <DOCKERID>/ ContainerizedAF :v1.0.0

Azure Functions and OpenAPI support

- You can now use the **Microsoft.Azure.Webjobs.Extensions.OpenApi** Nuget package to add OpenApi support to Azure Functions.



Useful for creating Dataverse custom connectors and extensions!

Conclusions

- Azure Functions are an important building block for a D365BC SaaS project.
- Azure Functions are a great low-cost (or often free) way to be SaaS-ready and *Universal-code* compliant.
- With Azure Functions code you can do all what you want (DLLs, NuGet packages, control performances, interact with any service etc.).
- Azure Functions it's not just deploying some code in the cloud.
- Start using the *isolated* model.
- In a multi-customer project, use tricks for high availability and performances.

BC/NAV TECH DAYS 2023
mibuso.com

Q&A

Any Questions?

BC/NAV TECH DAYS 2023
mibuso.com

Thank You!