I.3 Implement Azure functions

jeudi 16 septembre 2021 14:23

Create and deploy Azure Functions apps

Consumption plan types :

Plan	Benefits
Consumption plan	This is the default hosting plan. It scales automatically and you only pay for compute resources when your functions are running. Instances of the Functions host are dynamically added and removed based on the number of incoming events.
Functions 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.
App service 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.

Hosting options :

- App Service Environment (ASE) : the classic one, fully isolated and dedicated environement. Always on option must always be activated in order to avoid "wake-up" latency effect.
- Kubernetes : fully isolated & dedicated environment on top of Kubernetes platform. Uses KEDA.

Storage account requirements :

- General SA supporting Blob, Queue, File & Table storage.

1. **Create**

Azure CLI

```
RESOURCEGROUP=learn-e7872128-f540-4a07-99bd-bf8f9bd95d20
STORAGEACCT=learnstorage$(openssl rand -hex 5)
FUNCTIONAPP=learnfunctions$(openssl rand -hex 5)

az storage account create \
  --resource-group "$RESOURCEGROUP" \
  --name "$STORAGEACCT" \
  --kind StorageV2 \
  --location centralus

az functionapp create \
  --resource-group "$RESOURCEGROUP" \
  --name "$FUNCTIONAPP" \
  --storage-account "$STORAGEACCT" \
  --runtime node \
  --consumption-plan-location centralus
```

2. **Deploy**

```
# publish the code
dotnet publish -c Release
$publishFolder = "FunctionsDemo/bin/Release/netcoreapp2.1/publish"


# create the zip
$publishZip = "publish.zip"
if(Test-path $publishZip) {Remove-item $publishZip}
Add-Type -assembly "system.io.compression.filesystem"
[io.compression.zipfile]::CreateFromDirectory($publishFolder, $publishZip)


# deploy the zipped package
az functionapp deployment source config-zip `
 -g $resourceGroup -n $functionAppName --src $publishZip
```

Implement input and output bindings for a function

Example with a blob trigger

Project structure :

```
<framework.version>
```

```
| - bin
| - MyFirstFunction
| | - function.json
| - MySecondFunction
| | - function.json
| - host.json
```

The following example is a C# function that uses a blob trigger and two output blob bindings.
The function is triggered by the creation of an image blob in the *sample-images* container. It creates small and medium size copies of the image blob.

```
using System.Collections.Generic;
using System.IO;
using Microsoft.Azure.WebJobs;
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.Formats;
using SixLabors.ImageSharp.PixelFormats;
using SixLabors.ImageSharp.Processing;

public class ResizeImages
{
    [FunctionName("ResizeImage")]
    public static void Run([BlobTrigger("sample-images/{name}")] Stream image,
        [Blob("sample-images-sm/{name}", FileAccess.Write)] Stream imageSmall,
        [Blob("sample-images-md/{name}", FileAccess.Write)] Stream imageMedium)
    {
        IImageFormat format;

        using (Image<Rgba32> input = Image.Load<Rgba32>(image, out format))
        {
            ResizeImage(input, imageSmall, ImageSize.Small, format);
        }

        image.Position = 0;
        using (Image<Rgba32> input = Image.Load<Rgba32>(image, out format))
        {
            ResizeImage(input, imageMedium, ImageSize.Medium, format);
        }
    }

    public static void ResizeImage(Image<Rgba32> input, Stream output, ImageSize size, IImageFormat format)
    {
        var dimensions = imageDimensionsTable[size];

        input.Mutate(x => x.Resize(dimensions.Item1, dimensions.Item2));
        input.Save(output, format);
    }

    public enum ImageSize { ExtraSmall, Small, Medium }

    private static Dictionary<ImageSize, (int, int)> imageDimensionsTable = new Dictionary<ImageSize, (int, int)>()
        { ImageSize.ExtraSmall, (320, 200) },
        { ImageSize.Small,      (640, 400) },
        { ImageSize.Medium,     (800, 600) }
    };

}
```

Implement function triggers by using data operations, timers, and webhooks

1. Function trigger by using data operations

Type	1.x	2.x and higher[1]	Trigger	Input	Output
Blob storage	✓	✓	✓	✓	✓
Azure Cosmos DB	✓	✓	✓	✓	✓
Azure SQL (preview)		✓		✓	✓
Dapr[3]		✓	✓	✓	✓
Event Grid	✓	✓	✓		✓
Event Hubs	✓	✓	✓		✓
HTTP & webhooks	✓	✓	✓		✓
IoT Hub	✓	✓	✓		✓
Kafka[2]		✓	✓		✓
Mobile Apps	✓			✓	✓
Notification Hubs	✓				✓
Queue storage	✓	✓	✓		✓

Queue storage	✓	✓	✓		✓
RabbitMQ[2]		✓	✓		✓
SendGrid	✓	✓			✓
Service Bus	✓	✓	✓		✓
SignalR		✓		✓	✓
Table storage	✓	✓		✓	✓
Timer	✓	✓	✓		
Twilio	✓	✓			✓

2. Function trigger by using timer

The following example shows a C# function that is executed each time the minutes have a value divisible by five (eg if the function starts at 18:57:00, the next performance will be at 19:00:00). The TimerInfo object is passed into the function.

C# Copy

```
[FunctionName("TimerTriggerCSharp")]
public static void Run([TimerTrigger("0 */5 * * * *")]TimerInfo myTimer, ILogger log)
{
    if (myTimer.IsPastDue)
    {
        log.LogInformation("Timer is running late!");
    }
    log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
}
```

Azure functions : **CRON** Expressions (Timer Trigger)
- **Format : {seconde} {minute} {heure} {jour} {mois} {jour de la semaine}**
  "*" = tout
  "/" = "chaque
  "-" = plage
  "," = et

* * * ? * *	Every second
0 * * ? * *	Every minute
0 */2 * ? * *	Every even minute
0 1/2 * ? * *	Every uneven minute

3. Function trigger by using webhook

Webhook : Request that is made on a certain url (configured at the time of the creation of the webhook) when an event occurs. Example: When the wiki page of a github repository is modified, a webhook configured in github sends the information of the modification (who, when, etc...) to an azure function that displays them.

Possible to **secure the webhook with a secret** to be sure that the requests come from an approved source. **x-hub-signature** header.

As part of an azure function, **webhook is a behavior of the HTTPTrigger**, only usable for an **azure function v1. 3** possible webhooks:

- **genericJson**—A general-purpose webhook endpoint without logic for a specific provider. This setting restricts requests to only those using HTTP POST and with the application/json content type.
- **Github**
- **Slack**

Implement Azure Durable Functions

Long and stateful operations (conserve les informations d'état entre les appels de fonctions)
-> Event-based code
-> chainage de fonctions qui peuvent être appelé sync ou async

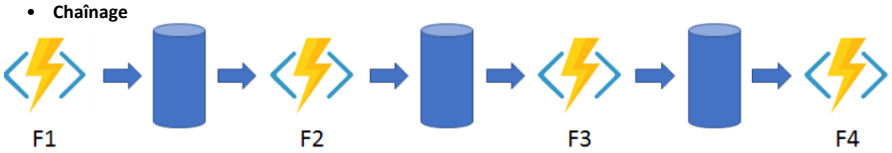
4 types of durable functions :
- **Client** : entry point, instantiates orchestration, so uses durable client output binding
- **Orchestration** : to describe how actions (activity) are performed and the order ([OrchestrationTrigger])
- **Activité** : basic work unit,
- **Entity** : read and update small parts of state

Example :

Workflow function	Durable Function Type
Submitting a project design proposal for approval	*Client* Function
Assign an Approval task to relevant member of staff	*Orchestration* Function
Approval task	*Activity* Function
Escalation task	*Activity* Function

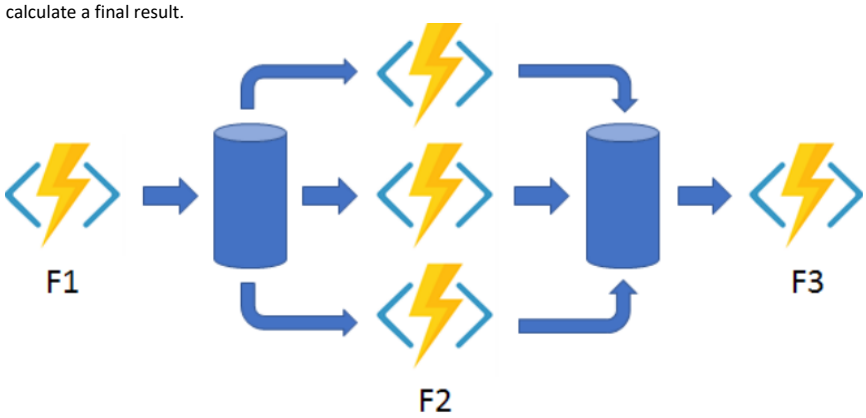
Patterns :

- **Chaînage**




```
[FunctionName("Chaining")]
public static async Task<object> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    try
    {
        var x = await context.CallActivityAsync<object>("F1", null);
        var y = await context.CallActivityAsync<object>("F2", x);
        var z = await context.CallActivityAsync<object>("F3", y);
        return  await context.CallActivityAsync<object>("F4", z);
    }
    catch (Exception)
    {
        // Error handling or compensation goes here.
    }
}
```

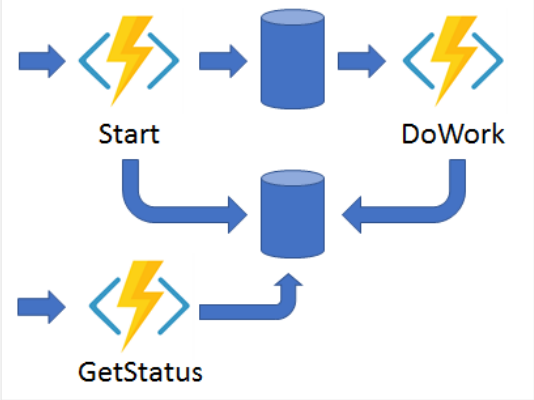
- **Fan-out/Fan-in** :
  This model runs multiple functions in parallel and then waits for all the functions to finish executing. The results of parallel runs can be aggregated or used to calculate a final result.




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

    // Get a list of N work items to process in parallel.
    object[] workBatch = await context.CallActivityAsync<object[]>("F1", null);
    for (int i = 0; i < workBatch.Length; i++)
    {
        Task<int> task = context.CallActivityAsync<int>("F2", workBatch[i]);
        parallelTasks.Add(task);
    }

    await Task.WhenAll(parallelTasks);

    // Aggregate all N outputs and send the result to F3
```

```
    // Aggregate all N outputs and send the result to F3.
    int sum = parallelTasks.Sum(t => t.Result);
    await context.CallActivityAsync("F3", sum);
}
```

- **API HTTP Async** :to coordinate the status of time-consuming operations with external customers.



Durable Functions provides **built-in support** for this pattern, simplifying or even removing the code you need to write to interact with long-running function executions. After an instance starts, the extension exposes webhook HTTP APIs that query the orchestrator function status.

The following example shows REST commands that start an orchestrator and query its status. For clarity, some protocol details are omitted from the example.

Copy

```
> curl -X POST https://myfunc.azurewebsites.net/orchestrators/DoWork -H "Content-Length: 0" -i
HTTP/1.1 202 Accepted
Content-Type: application/json
Location: https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/b79baf67f717453ca9e86c5da21e03ec

{"id":"b79baf67f717453ca9e86c5da21e03ec", ...}

> curl https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/b79baf67f717453ca9e86c5da21e03ec -i
HTTP/1.1 202 Accepted
Content-Type: application/json
Location: https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/b79baf67f717453ca9e86c5da21e03ec

{"runtimeStatus":"Running","lastUpdatedTime":"2019-03-16T21:20:47Z", ...}

> curl https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/b79baf67f717453ca9e86c5da21e03ec -i
HTTP/1.1 200 OK
Content-Length: 175
Content-Type: application/json

{"runtimeStatus":"Completed","lastUpdatedTime":"2019-03-16T21:20:57Z", ...}
```

```
public static class HttpStart
{
    [FunctionName("HttpStart")]
    public static async Task<HttpResponseMessage> Run(
        [HttpTrigger(AuthorizationLevel.Function, methods: "post", Route = "orchestrators/{functionName}")] HttpRequ
        [DurableClient] IDurableClient starter,
        string functionName,
        ILogger log)
    {
        // Function input comes from the request content.
        object eventData = await req.Content.ReadAsAsync<object>();
        string instanceId = await starter.StartNewAsync(functionName, eventData);

        log.LogInformation($"Started orchestration with ID = '{instanceId}'.");

        return starter.CreateCheckStatusResponse(req, instanceId);
    }
}
```

- **Monitor**: recurring process: Ex: poll until a specific condition is met.

```
[FunctionName("MonitorJobStatus")]
public static async Task Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    int jobId = context.GetInput<int>();
    int pollingInterval = GetPollingInterval();
    DateTime expiryTime = GetExpiryTime();

    while (context.CurrentUtcDateTime < expiryTime)
    {
        var jobStatus = await context.CallActivityAsync<string>("GetJobStatus", jobId);
        if (jobStatus == "Completed")
        {
            // Perform an action when a condition is met.
            await context.CallActivityAsync("SendAlert", machineId);
            break;
        }

        // Orchestration sleeps until this time.
        var nextCheck = context.CurrentUtcDateTime.AddSeconds(pollingInterval);
        await context.CreateTimer(nextCheck, CancellationToken.None);
    }

    // Perform more work here, or let the orchestration end.
}
```

- **Human interaction**


```
[FunctionName("ApprovalWorkflow")]
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);
        }
    }
```

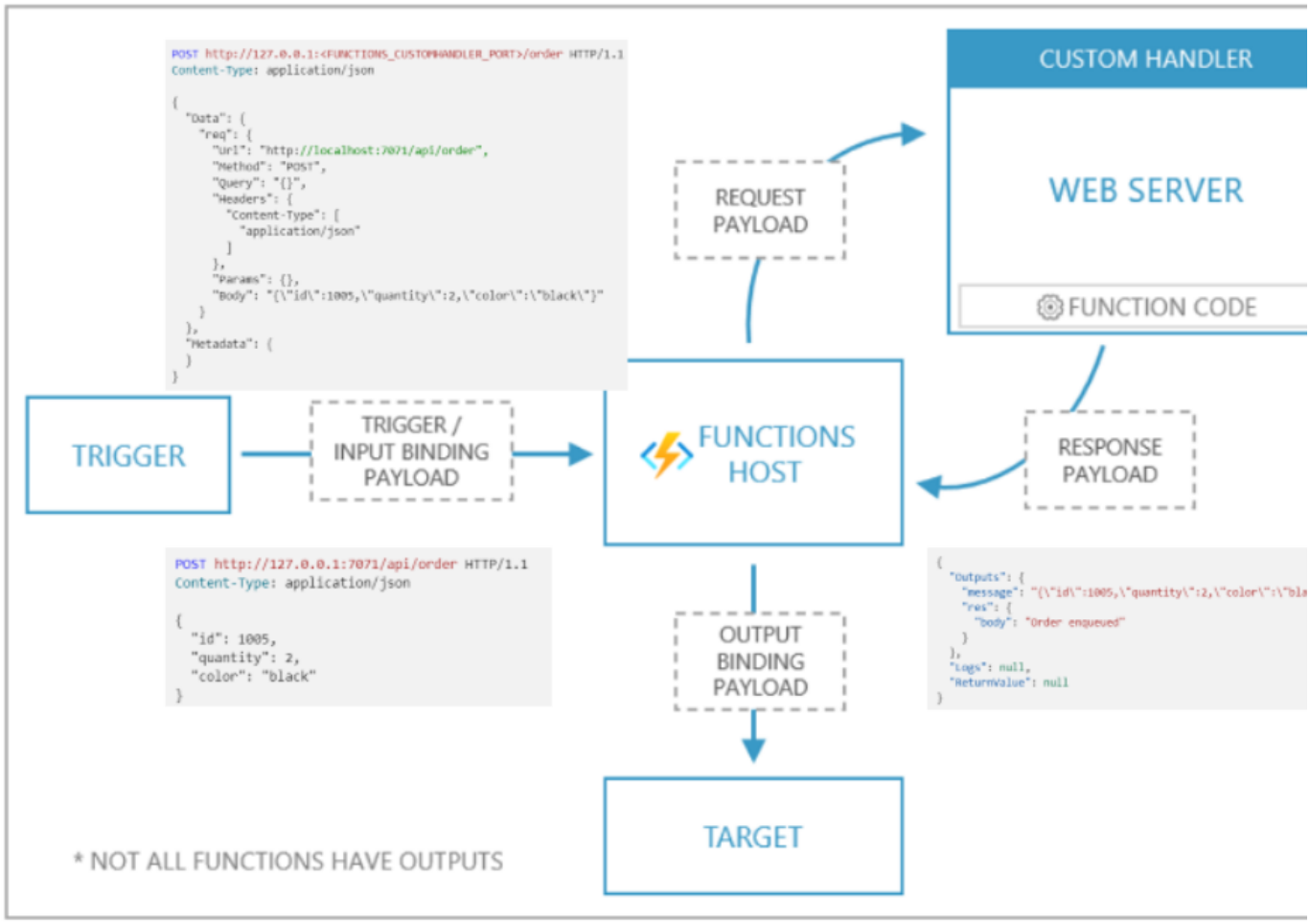
```
    }
}
```

Task hubs

A task hub in Durable Functions is a logical container for durable storage resources that are used for orchestrations and entities. Orchestrator, activity, and entity functions can only directly interact with each other when they belong to the same task hub.

Implement custom handlers

Définition : Small web servers that receive events from a "host" function. Any language that supports HTTP primitives can implement a custom handler. We can use triggers, input & output bindings via extension bundles



Structure :

- A *host.json* file always at the root of your app. Les extensions bundles are referenced in host.json.

```
{
  "version": "2.0",
  "customHandler": {
    "description": {
      "defaultExecutablePath": "handler.exe"
    }
  },
  "extensionBundle": {
    "id": "Microsoft.Azure.Functions.ExtensionBundle",
    "version": "[1.*, 2.0.0)"
  }
}
```

- A *local.settings.json* file at the root of your app.

```
{
  "IsEncrypted": false,
  "Values": {
    "FUNCTIONS_WORKER_RUNTIME": "Custom"
  }
}
```

- A *function.json* file for each function (inside a folder that matches the function name).
  This function is defined as an HTTP triggered function that returns an HTTP response and outputs a Queue storage message.

```
```

```
{
  "bindings": [
    {
      "type": "httpTrigger",
      "direction": "in",
      "name": "req",
      "methods": ["post"]
    },
    {
      "type": "http",
      "direction": "out",
      "name": "res"
    },
    {
      "type": "queue",
      "name": "message",
      "direction": "out",
      "queueName": "orders",
      "connection": "AzureWebJobsStorage"
    }
  ]
}
```

- A command, script, or executable, which runs a web server.