Function app

A Required knowledge

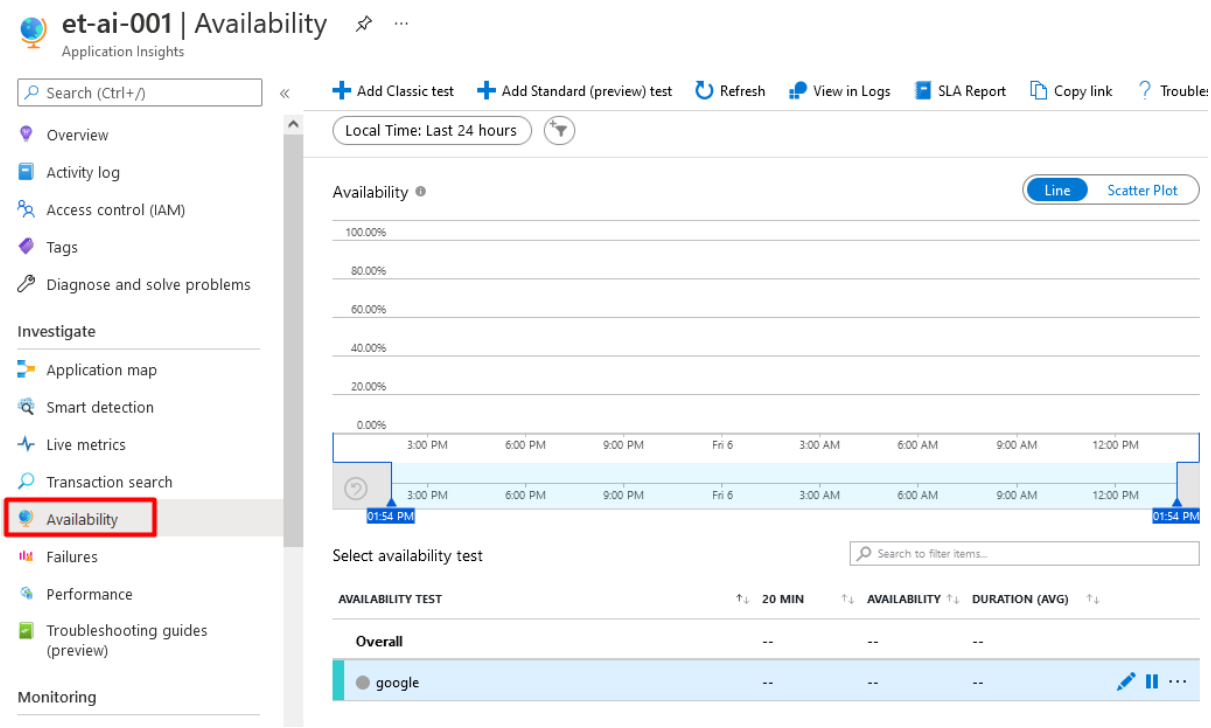
It is assumed that the following technologies are known

- Bicep / ARM
- Azure Pipelines
- Powershell

B Introduction

This task is based on a real solution used by one of our customers. Developers want to monitor the availability of their APIs. Microsoft has an out-of-the-box solution for this using availability tests which is a subcomponent of application insights. However, this will only work if the API is publicly accessible, which is not the case for some of our APIs. Luckily, a function app can help us in this case. The function app can reside behind the firewall and execute the necessary web requests, hereafter, the function app will forward the results to the application insights. As a side note, to not make the exercise overly complex, we are not actually going to monitor private api's but just pretend that www.realdolmen.com is a private API. In other words, the function app can be public.

So in short: Create a function app that monitors a certain URL endpoint and logs the results in application insights.



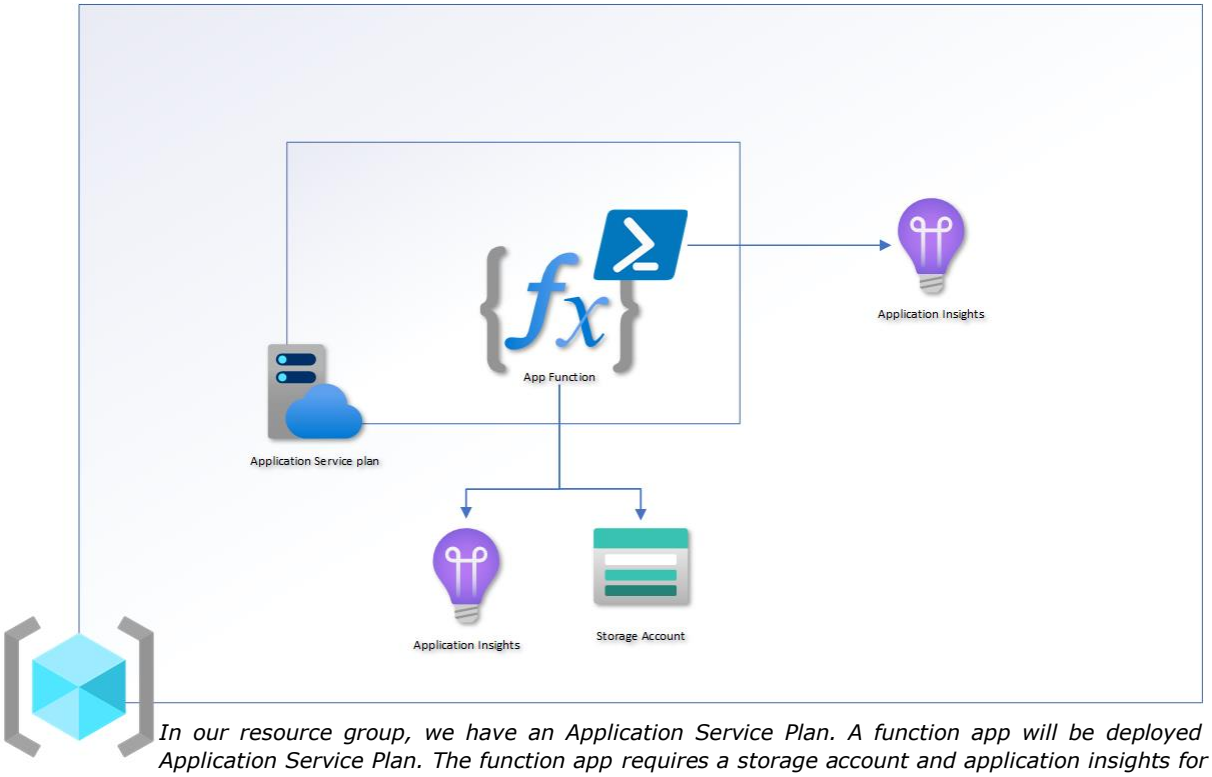
To get started, you should first enroll the function app and all its required components. Of course, we use an IaC-automated approach, so once the necessary bicep files have been created, a pipeline should be added. Hereafter, we will have to create a PowerShell script that processes the actual requests and finally, the pipeline needs to be updated so that this code is uploaded to the function app in an automated matter.

Thomas.detemmerman@inetum-realdolmen.world

A repository, service connection and a resource group have been created for you. You only have to replace the x with your sequence number.

Repository	et-00x
Service connection	et-svc-00x
Resource group	et-rg-00x

Architectural design



In our resource group, we have an Application Service Plan. A function app will be deployed on the Application Service Plan. The function app requires a storage account and application insights for proper functioning. On the function app, we use a PowerShell script that will communicate with another application insights for storing the availability tests (on the right).

C Function app requirements

Create a bicep file with the following resources.

- Function app
  - This will execute the custom written code.
- Storage account
  - The function app relies on a storage account for managing triggers, logging function executions…
  - SKU name: Standard_LRS
  - Name:
- Application insights
  - For storing execution logs.
- App service plan
  - For the function app to run on.
  - SKU: Free

Thomas.detemmerman@inetum-realdolmen.world

Use the following naming convention:

<project name>-<user sequence number>-<resource abbreviation>-<resource sequence number>

- Project name: et (expedition track)
- User sequence number : number of the resource group
- Resource abbreviation :
    - ai = application insights
    - asp = app service plan
    - storage account = sta
        - storage accounts do not allow hyphens in their name. So make sure to omit them.
    - fa = function app

examples:

- et-1-fa-1
- et2sta1

D Create a pipeline

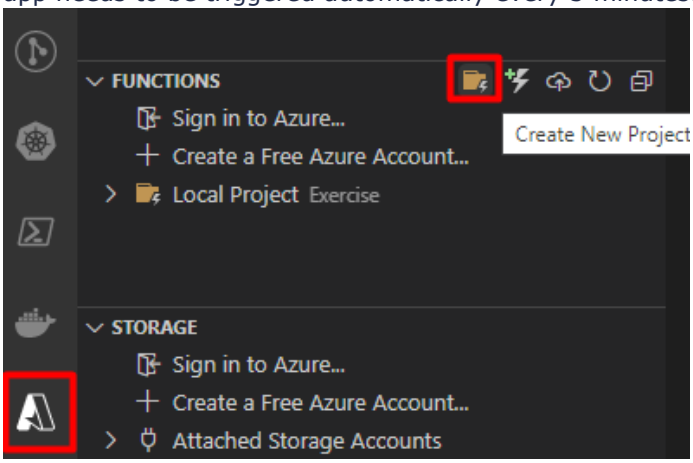
Create a pipeline that deploys your code to your resource group using the existing service connection. The service connection should already have the proper permissions to do so. At the time of writing, no predefined azure task exists for the deployment of bicep file. Use Azure CLI to accomplish this.

E Create a function

Let us now create a PowerShell script for the monitoring of API's. Of course, we want to keep logs from our function app separate from the results of our availability tests. Therefore, you should alter your bicep file and create second application insights.

It is now time to create the actual logic. You can generate most of the code as follows:

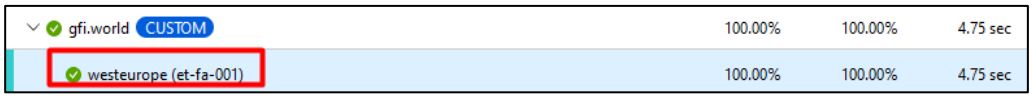
1. Install extension: Azure Functions
2. Create a folder PSFunction and generate the folder structure inside this map. Our function app needs to be triggered automatically every 5 minutes.



3. Include the Modules folder, which we provided you, in the PSFunction folder.
4. In Run.ps1, call this module with the proper parameters.

Thomas.detemmerman@inetum-realdolmen.world

a. Location must be fetched as an environment variable because it is dynamic. If somethings occurs, we easily want to know which service executed it and where it is located. To get you started, the following piece of code
-Testlocation "$($env:location) ($($env:functionappname))"
will result the following red box



b. instrumentation key must be accessed as an environment variable for security purposes. We don't want keys in our source code.

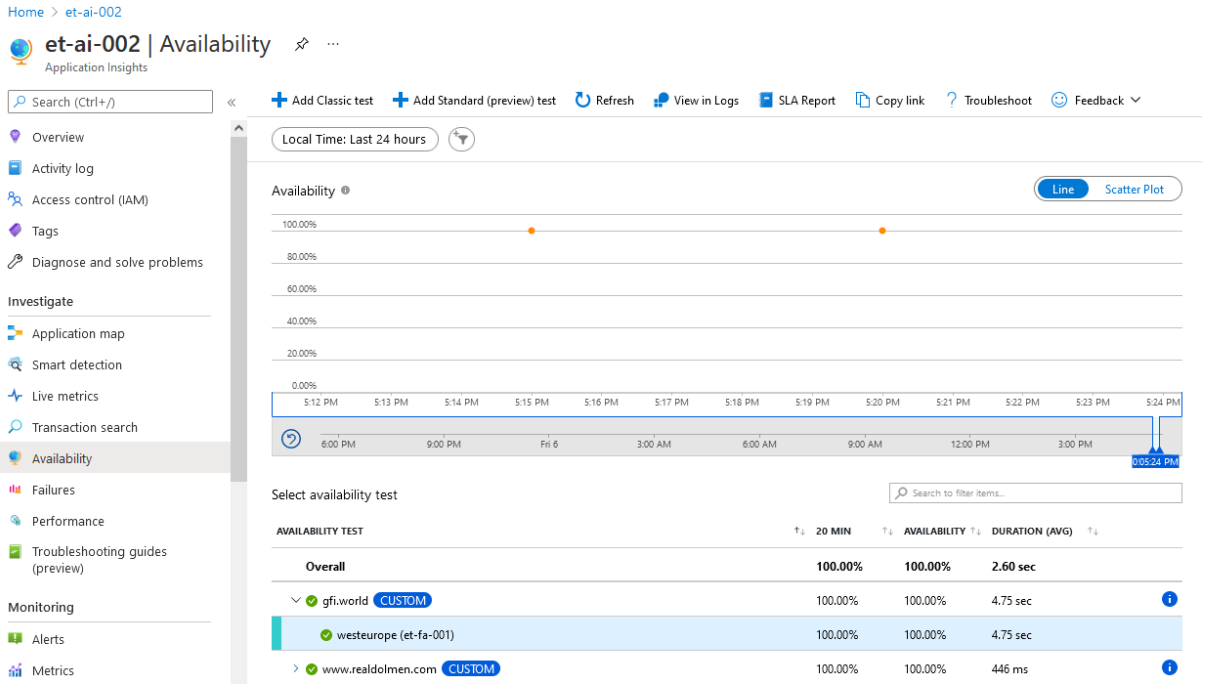
F Deploy the function

You can easily archive and deploy the function app using two predefined pipeline tasks. Check the documentation for more information: https://docs.microsoft.com/en-us/azure/devops/pipelines/targets/azure-functions?view=azure-devops&tabs=dotnet-core%2Cyaml

G Optional: improvements

- Use HTTP v2.0
- Disable FTP

Expected output



Appendix A

Thomas.detemmerman@inetum-realdolmen.world

```
# Input bindings are passed in via param block.
param($Timer)

$InstrumentationKey =  $ENV:AvailabilityResults_InstrumentationKey
$webtests = convertfrom-json $ENV:webtests

function convert-rawHTTPResponseToObject{
    param(
        [string] $rawHTTPResponse
        )

    if("$rawHTTPResponse" -match '(\d\d\d) \(((\w+|\s)+)\)')
    {

    $props = @{
    StatusCode = $Matches[1]
    StatusMessage = $Matches[2]

    }
    return new-object psobject -Property $props
    } else {
        write-error "Could not extract data from input"
    }



}
$OriginalErrorActionPreference = "Stop";
foreach ($webtest in $webtests) {
    $TestName = $webtest.name
    $Uri = $webtest.URL
    $Expected = $webtest.Expected

    $EndpointAddress = "https://dc.services.visualstudio.com/v2/track";
    $Channel = [Microsoft.ApplicationInsights.Channel.InMemoryChannel]::new();
    $Channel.EndpointAddress = $EndpointAddress;
    $TelemetryConfiguration = [Microsoft.ApplicationInsights.Extensibility.TelemetryConfiguration]
::new(
        $InstrumentationKey,
        $Channel
    );
    $TelemetryClient = [Microsoft.ApplicationInsights.TelemetryClient]::new($TelemetryConfigurati
on);


    $TestLocation = "$Env:GEOLOCATION ($Env:FunctionAppName)"; # you can use any string for
this
    $OperationId = (New-Guid).ToString("N");
```

```
    $Availability = [Microsoft.ApplicationInsights.DataContracts.AvailabilityTelemetry]::new();
    $Availability.Id = $OperationId;
    $Availability.Name = $TestName;
    $Availability.RunLocation = $TestLocation;
    $Availability.Success = $False;

    $Stopwatch =  [System.Diagnostics.Stopwatch]::New()
    $Stopwatch.Start();

    Try
    {
        # Run test
        $Response = Invoke-WebRequest -Uri $Uri  -SkipCertificateCheck;

    }
    Catch
    {
        $s = [string]$_.Exception.Message
        $Response = convert-rawHTTPResponseToObject -rawHTTPResponse  "$s"

    }
    Finally
    {
        $Success = $Response.StatusCode -eq $Expected;
        if($Success){
            Write-
host "Testing $TestName on $Uri (Successful: The server retuned the expected statuscode $Expec
ted)"
        } else {
            Write-
host "Testing $TestName on $Uri (Failed: expected $Expected. Got: "  $Response.StatusCode  ")"
        }
        $Availability.Success = $Success;

        $ExceptionTelemetry = [Microsoft.ApplicationInsights.DataContracts.ExceptionTelemetry]::ne
w($_.Exception);
        $ExceptionTelemetry.Context.Operation.Id = $OperationId;
        $ExceptionTelemetry.Properties["TestName"] = $TestName;
        $ExceptionTelemetry.Properties["TestLocation"] = $TestLocation;
        $TelemetryClient.TrackException($ExceptionTelemetry);

        $Stopwatch.Stop();
        $Availability.Duration = $Stopwatch.Elapsed;
        $Availability.Timestamp = [DateTimeOffset]::UtcNow;

        # Submit Availability details to Application Insights
        $TelemetryClient.TrackAvailability($Availability);
        # call flush to ensure telemetry is sent
```

```
        $TelemetryClient.Flush();
    }

}
```


Thomas.detemmerman@inetum-realdolmen.world