Development

Setting up the solution

We are starting right at the beginning. At first you'll need a solution to place the project in.

Navigate to a directory where you want the source to be in. For me it is `C:\Repos\azure\kaushK`. Open a console and type `dotnet new sln`, which will create a solution file. Open it. I am using Visual Studio 2022 Community Edition as the IDE of my choice.

Add a new Project.

Search for `Azure Functions` like so:

Hit Next and give a name. I'll choose `SoftDeleteDetection`. Hit Next.

Fill in the information like here. You want to use - .NET6.0 (because it's the current version) - Timer trigger (so that the function gets executed automatically) - Azurite (because it's default) - Docker if you want to (optional) - Schedule is every 5 Minutes by default. I set it to every Monday at 9 AM. Here's a cheat sheet

Hit Create - The project will be created.

Rename the function and use file scoped namespaces

By default the function will be named `Function1`, which is not very meaningful. I change it to `NotifyOnSoftDeletedResourcesFunctions`. Sounds like a lot - but now everyone knows what this function is supposed to do without having to take a look inside.

I like to change filenames through the solution explorer - because then a dialog appears asking if the class name should also be changed.

Your class now looks like this:

```
 using System;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Extensions.Logging;

namespace SoftDeleteDetection
{
    public class NotifyOnSoftDeletedResourcesFunctions
    {
        [FunctionName("Function1")]
        public void Run([TimerTrigger("0 0 9 * * MON")]TimerInfo myTimer, ILogger log)
        {
            log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
        }
    }
}
```

A class may contain multiple functions. That is why I pluralized the name - but this is just personal preference.

Change the name in the `FunctionName`-attribute and the method name `Run` as well to `ScanAndNotify` - again to have something meaningful.

```
public class NotifyOnSoftDeletedResourcesFunctions
    {
        [FunctionName("ScanAndNotify")]
        public void ScanAndNotify([TimerTrigger("0 0 9 * * MON")]TimerInfo myTimer, ILogger log)
        {
            log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
        }
    }
```

With dotnet6.0 comes file scoped namespaces. We can switch to that by typing a semicolon behind `namespace SoftDeleteDetection`.

Our code now looks like this:

```
 using System;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Extensions.Logging;

namespace SoftDeleteDetection;

public class NotifyOnSoftDeletedResourcesFunctions
{
    [FunctionName("ScanAndNotify")]
    public void ScanAndNotify([TimerTrigger("0 0 9 * * MON")]TimerInfo myTimer, ILogger log)
    {
        log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
    }
}
```

> The cron expression from the start is now insite the TimerTrigger(...). Change it as you like.

Settings

We will need to store some settings in our application. For that we have the file `local.settings.json`

Content looks like this:

```
{
    "IsEncrypted": false,
    "Values": {
        "AzureWebJobsStorage": "UseDevelopmentStorage=true",
        "FUNCTIONS_WORKER_RUNTIME": "dotnet"
    }
}
```

We want to send an email, so let's do an example here for report receipents.

```
{
  "IsEncrypted": false,
  "Values": {
      "AzureWebJobsStorage": "UseDevelopmentStorage=true",
      "FUNCTIONS_WORKER_RUNTIME": "dotnet",
      "ReportReceipents": "kontakt@fistelmann.de;ichhabjanixbessereszutun@web.de"
    }
}
```

We can access the value inside our code like this:

```
string reportReceipents = Environment.GetEnvironmentVariable("ReportReceipents");
```

Later when you deploy your application to azure you set things like this:

In order to have an overview of the settings I tend to create a class with properties of settings names.

```
namespace SoftDeleteDetection.Models;

public static class Settings
{
    public const string ReportReceipents = "ReportReceipents";
}
```

If you do this, you retrieve a setting like this:

```
string reportReceipents = Environment.GetEnvironmentVariable(Settings.ReportReceipents);
```

Setup Azure Monitor

We will use the Azure Monitor resource in order to get information about deletions of Resources and Azure Key Vault entries.

Azure Monitor Logs

For it to work properly we have to do some setup. 1. Create a log workspace 2. Set Diagnostic settings on resources

Create one if not existant.

Keep note of the workspace id as you will need it later.

https://portal.azure.com/#view/HubsExtension/BrowseResource/resourceType/Microsoft.OperationalInsights%2Fworkspaces

Now that we have a workspace we need to tell our resources to write logs there.

https://portal.azure.com/#view/Microsoft_Azure_Monitoring/AzureMonitoringBrowseBlade/~/diagnosticsLogs

We will do this for our Key Vault resource. Click on it and you will be redirected to a menu where you can set this up.

Now we will retrieve diagnostic log entries when something important happens to our key vault.

The next step is to get important log entries from our activity log. https://portal.azure.com/#view/Microsoft_Azure_Monitoring/AzureMonitoringBrowseBlade/~/activityLog

Click on `Export Activity Logs` and set up a Diagnostic setting just like you did for the key vault.

That's it. We don't need more than those two.

Access Azure Monitor

We are now able to perform Queries in the azure portal. But we want an azure function to query the resources for us. There are multiple ways to grant access. We will use client credentials for this, as the setup is pretty straight forward.

Go to the Active Directory resource: https://portal.azure.com/#view/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/~/Overview

And then to App Registrations

Click on `+ New registration` to create a new one.

Give a meaningful name. Supported account types: `Accounts in this organizational directory only (Standardverzeichnis only - Single tenant)` (default) Redirect URI (optional): ignore this one.

Hit `Register` and you will be redirected to the new app registration.

Take a note of the Application (client) ID and the Directory (tenant) ID as you will need them later.

On the left menu, go to Certificates & secrets. Create a client secret.

Enter a description and Expiracy.

> Remember that you need to refresh this from time to time. That is the nature of client secrets.

Keep note of the Value as you will need this later.

We now need to perform one final step. Giving this new app registration access to our Log Analytics Workspace.

Click Add and Add role assignment.

Choose the Contributor role and hit next. Click on select members.

Now comes a pretty weird part. On the right side there is a text box. You need to type in the name of your recently created app resource in order for it to be shown.

Click on the item to select it and then the blue select button on the bottom right corner.

Hit review + assign.

Done.

Write the code

We will use three Nuget Packages. - Azure.Identity (to use our client credentials and authorize) - Azure.Monitor.Query (to query the logs) - MailKit (to send emails)

The code is very straight forward, so I will not document it here again.

Deploy Azure function

There are multiple ways to deploy an azure function.

In this example I will deploy it from within Visual Studio.

From your Solution explorer, right click the project and hit publish.

Select Azure and Next. Select Azure Function App (Linux).

Click the green plus icon to create a new azure function.

Fill in everything as you need it and hit Create.

Hit Finish and Close.

Now we have a publish profile. Hit Publish and Visual Studio will deploy the app.

Configure Azure Function

The last thing we need to do is configuring the freshly published Azure Function. This will be done inside the azure portal.

Add the following settings:

Setting	Meaning
ReportReceipents	semicolon separated list of emails
SenderEmail	Email address from which the email should be sent
SenderName	Email Sender name
SmtpServer	smtp server host for email (outgoing)
SmtpPort	smtp server port (outgoing)
EnableSsl	whether to enable ssl or not
SmtpUsername	smtp server username
SmtpPassword	smtp server password
EmailSubject	email subject
LogWorkspaceId	log workspace id (noted before)
TenantId	app registration tenant id (noted before)
ClientID	app registration client id (noted before)
ClientSecret	app registration secret (noted before)