



# SERVERLESS FUNCTIONS

Semester 6 – Enterprise Software

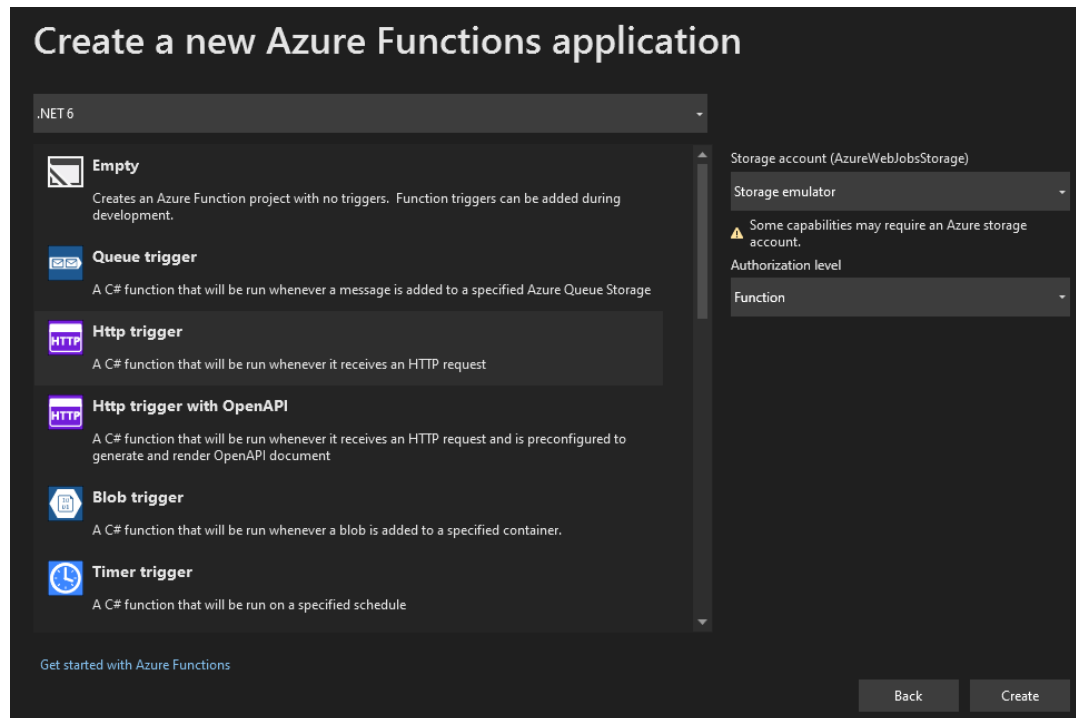
Rick Verhaegh; Stan Rutten  
Version 0.1

## Introductie

Om een outstanding te halen voor het semester hebben de docenten ons uitgedaagd om te kijken naar een serverless architectuur. Met behulp van Azure Functions hebben wij volledig serverless architectuur gemaakt die communiceert met een MongoDB database en afbeeldingen opslaat in een AWS S3 bucket. In dit document leggen wij vast hoe wij het project hebben opgezet en wat

## Hoe zetten we een Azure Functions project op?

Azure heeft meerdere ingebouwde triggers voor het uitvoeren van code. Zo kan code worden uitgevoerd op een tijdschema of wanneer een document in de CosmosDB wordt aangepast. Voor ons prototype hebben we gekozen voor een HTTP-trigger die afgaat wanneer er een verzoek wordt verstuurd op een specifieke URL.



## Hoe voegen we functies toe aan de applicatie?

Omdat we bij serverless applicaties de server zelf niet hoeven te beheren, wordt er geen gebruik gemaakt van het standaard MVC-patroon zoals we die gewend zijn bij voorgaande projecten. In plaats daarvan maakt Azure functions gebruik van een event-driven patroon.

Iedere functie in Azure functions moet beschikken over een functie naam om de functie te identificeren. Daarnaast moet iedere functie in de parameters aangeven wat voor trigger bij de functie hoort. In de HTTP trigger kan bijvoorbeeld worden aangegeven welke methode bij de HTTP request hoort (get, post, put, delete), welke endpoint moet worden geroepen en welk authenticatie niveau we willen gebruiken.

```
[FunctionName("getAll")]
public static async Task<IActionResult> GetAsync(
    [HttpTrigger(AuthorizationLevel.Function, "get", Route = "post/getAll")] HttpRequest req,
    ILogger log)
{
    var findResult = await _mongoDbCollection.FindAsync(x => true);
    var listResult = await findResult.ToListAsync();

    return new JsonResult(listResult.OrderByDescending(x => x.CreatedAt));
}
```

In de functie kunnen we de code zetten zoals we die gewend zijn van de andere .NET projecten. In het voorbeeld hierboven communiceren we met de MongoDB database om alle posts uit de database op te halen.

## Hoe kunnen endpoints op een serverless project worden beveiligd?

Om endpoints te kunnen beveiligen kunnen sleutels worden gebruikt die mee moeten worden gegeven bij het aanvragen van de data. In Azure functions kennen we vier soorten authenticatie niveaus

- **Anonymous**

Bij de anonymous beveiliging zijn geen sleutels verplicht. Dit maakt de endpoint publiekelijk toegankelijk zonder versleuteling.

- **Function**

Bij de functie beveiliging moet er een unieke sleutel worden meegegeven die alleen geldt voor deze functie of hoofdsleutel. Zonder deze sleutel krijgt de user een foutmelding bij het aanroepen van de endpoint.

- **Admin**

De admin sleutel is de hoofdsleutel van de applicatie. Met de admin sleutel kunnen alle functions worden aangeroepen. Best practice is om de key niet client side mee te geven maar gebruik te maken van de overige authenticatie niveaus.

- **System**

De systeemsleutel is een speciale sleutel voor extensies die zijn geïnstalleerd op Azure functions.

## Hoe zorgen we ervoor dat we geen gevoelige gegevens vrijgeven?

In de Azure dashboard onder configuratie kunnen environments worden meegegeven aan de applicatie. Hierin hebben wij onze belangrijke keys bewaard die nodig zijn om een connectie te maken met AWS S3 Bucket en met MongoDB. Configuratie keys worden in de code opgehaald door middel van de functie `GetEnvironmentVariable()`.

Startpagina > rist-function-app

### rist-function-app | Configuratie

Zoeken (Cmd + /) Vernieuwen Opslaan Negeren Feedback geven

Nieuwe toepassingsinstelling Waarden weergeven Geavanceerd bewerken

Toepassingsinstellingen filteren



Naam	Waarde	Bron
APPINSIGHTS_INSTRUMENTATIONKEY	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
APPLICATIONINSIGHTS_CONNECTION_STRIN	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
AWS_ACCESS_KEY	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
AWS_BUCKET_NAME	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
AWS_SECRET_ACCESS_KEY	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
AzureWebJobsStorage	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
FUNCTIONS_EXTENSION_VERSION	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
FUNCTIONS_WORKER_RUNTIME	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
MONGODB_DATABASE_NAME	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
MONGODB_CONNECTION_STRING	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
WEBSITE_CONTENTAZUREFILECONNECTION	Verborgen waarde. Klik om de waarde w	Configuratie van App Service
WEBSITE_CONTENTSHARE	Verborgen waarde. Klik om de waarde w	Configuratie van App Service

```
static going_serverless()
{
    _mongoDbSettings = new MongoDBSettings
    {
        ConnectionString = Environment.GetEnvironmentVariable("MONGODB_CONNECTION_STRING"),
        DatabaseName = Environment.GetEnvironmentVariable("MONGODB_DATABASE_NAME")
    };


    _amazonS3Settings = new AmazonS3Settings
    {
        AccessKey = Environment.GetEnvironmentVariable("AWS_ACCESS_KEY"),
        SecretKey = Environment.GetEnvironmentVariable("AWS_SECRET_ACCESS_KEY"),
        BucketName = Environment.GetEnvironmentVariable("AWS_BUCKET_NAME")
    };
}
```

## Hoe koppelen we de frontend met de serverless backend?

Bij het testen van de frontend applicatie kwamen we erachter dat CORS niet goed was geconfigureerd. Omdat onze backend op een Azure server draait hebben wij in de Azure functions dashboard handmatig de toegestane URLs toegevoegd. Dit zijn localhost voor het lokaal testen van de applicatie en de Vercel app voor de deployments van de applicatie.

 Opslaan
  Negeren

---



### CORS




Met CORS (Cross-Origin Resource Sharing) kan JavaScript-code die wordt uitgevoerd in een browser op een externe host, interactie uitvoeren met uw back-end. Geef de oorsprongen op waarvoor u cross-origin-aanroepen wilt toestaan (bijvoorbeeld: <http://example.com:12345>). Gebruik \* als u dit voor alle oorsprongen wilt toestaan en verwijder alle andere oorsprongen uit de lijst. Slashes zijn niet toegestaan als deel van een domein of na een TLD. [Meer informatie](#)

---

Referenties aanvragen

☐ Access-Control-Allow-Credentials inschakelen ⓘ

Toegestane oorsprongen

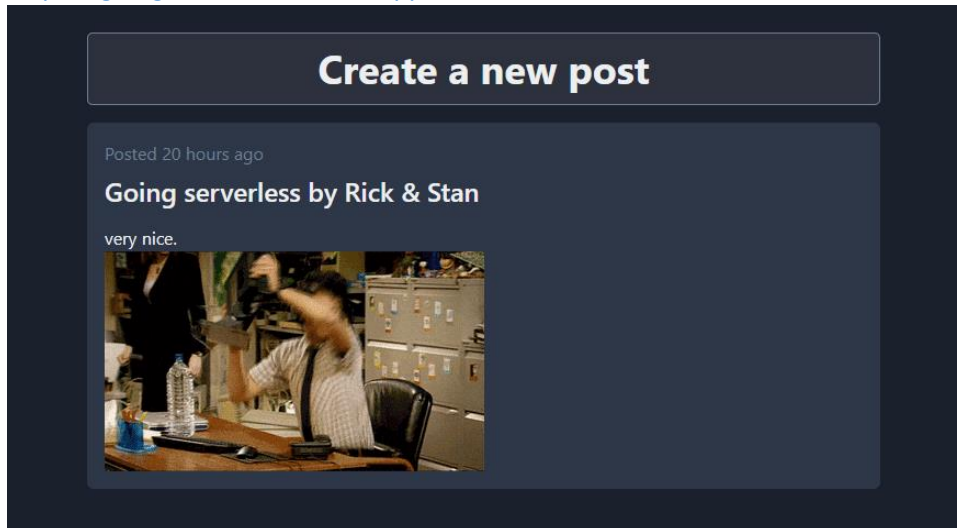
<a href="https://portal.azure.com">https://portal.azure.com</a>	 ...
<a href="http://localhost:3000">http://localhost:3000</a>	 ...
<a href="https://going-serverless.vercel.app">https://going-serverless.vercel.app</a>	 ...

## Conclusie

Wij hebben ons verdiept op het gebied van serverless functions. Hiervoor hebben wij gebruik gemaakt van Azure HTTP-trigger functions voor het ophalen en plaatsen van posts. Bij het maken van een post kan de gebruiker nog optioneel een afbeeldingen toevoegen die wordt weggeschreven in een AWS S3 Bucket.

De deployment van de applicatie kan je terugvinden op:

<https://going-serverless.vercel.app/>



The backend van de applicatie is gehost op Azure.

[GET] <https://rist-function-app.azurewebsites.net/api/post/getAll>

[POST] <https://rist-function-app.azurewebsites.net/api/post/create>

De broncode van de applicatie kan teruggevonden worden op GitHub

<https://github.com/Ruitjes/going-serverless>