

**SkillSeeker: in de cloud**

Samuel Wouters

Promotoren:

|  |  |
| --- | --- |
| Zoë Viola | Mediaan |
| Carine Derkoningen | Hogeschool PXL Hasselt |



**SkillSeeker: in de cloud**

Samuel Wouters

Promotoren:

|  |  |
| --- | --- |
| Zoë Viola | Mediaan |
| Carine Derkoningen | Hogeschool PXL Hasselt |

Dankwoord

Ik heb dit eindwerk geschreven als afstuderende aan de opleiding Toegepaste Informatica aan de Hogeschool PXL. Ik wil Mediaan, mijn stagebedrijf, bedanken om mij deze kans te geven om het werkveld te ervaren. Mijn bedrijfspromotor, N. Darcis, en begeleidster binnen Hogeschool PXL, C. Derkoningen, hebben mij ondersteund bij elk stap van dit proces. Wanneer ik vragen had of nood had aan steun stonden zij mij altijd bij.

Vervolgens wil ik mijn teamgenoten bedanken voor de leuke momenten tijdens het project. R. Lux, S. Ekici en C. Belmans hebben ervoor gezorgd dat elke dag van de stage een nieuw avontuur was vol met lachen en uitdagingen. Samen hebben wij vele nieuwe dingen geleerd en ervaring opgedaan op de werkvloer.

Ten slotte wil ik mijn familie en vrienden bedanken voor de steun tijdens stressvolle tijden. Wanneer de projectstress even te veel werd kon ik altijd bij hen terecht om even te ontspannen.

Abstract

Mediaan Conclusion is een IT-consultancybedrijf met vestigingen die gelegen zijn in België, Nederland en Duitsland. Mediaan heeft zich recent aangesloten bij de Conclusion-groep en wordt nu Mediaan Conclusion genoemd.

Er wordt een product voor Mediaan ontwikkeld dat het beheren van medewerkers vereenvoudigt voor het salesteam. Dit houdt in dat wanneer er een nieuw project wordt aangeboden aan Mediaan, dit ingevoerd kan worden in de applicatie en medewerkers eraan kunnen worden toegewezen. De uitdaging is dat wanneer er een project is waarvoor iemand met Java-ervaring nodig is, ook C#-ontwikkelaars moeten worden getoond, aangezien Java vergelijkbaar is met C#. Om dit te realiseren, wordt er gebruikgemaakt van een relationele databank waarin vaardigheden aan elkaar zijn gerelateerd op basis van hun overeenkomsten.

De applicatie stelt een medewerker in staat zijn vaardigheden in te vullen met een score van 1 tot 10. Vervolgens zal de applicatie relaties creëren tussen de vaardigheid en de medewerker, waarbij de score als koppeling fungeert.

De applicatie wordt ontwikkeld met behulp van een Agile-methodologie, waarbij gebruik wordt gemaakt van 5 sprints van elk 2 weken. In elke sprint worden user story’s met de bijbehorende taken voltooid. Na elke sprint wordt een evaluatie en reflectie uitgevoerd om de verbeterpunten te identificeren, gevolgd door een planning voor de volgende sprint.

Aangezien de verkoopafdeling geen technologische kennis heeft, is het belangrijk dat de applicatie online gehost kan worden en niet alleen lokaal draait. Er zijn verschillende mogelijkheden voor het hosten van een grafische databank, variërend van databank-as-a-service tot het zelf online plaatsen van een container. Er wordt een onderzoek uitgevoerd waarbij de verschillende opties worden vergeleken op het gebied van kosten en gebruiksvriendelijkheid, evenals hoe de applicatie in de cloud communiceert tussen de componenten.

De SkillSeeker-applicatie wordt ontwikkeld met een Java Spring Boot-applicatie als backend en een Angular-applicatie als frontend. Neo4j wordt ingezet als grafische databank om relaties tussen entiteiten te onderhouden. Voor hosting in de Azure-cloud worden serverloze container-apps gebruikt voor zowel de backend als de databank, samen met een web-app voor de frontend.

Na uitgebreid onderzoek is besloten dat een kleine Neo4j databank best in de cloud kan draaien op Azure Container Apps wanneer de focus ligt op lage kosten.

Inhoudsopgave

Contents

[Dankwoord ii](#_Toc168673333)

[Abstract iii](#_Toc168673334)

[Inhoudsopgave iv](#_Toc168673335)

[Lijst van gebruikte figuren vi](#_Toc168673336)

[Lijst van gebruikte tabellen vii](#_Toc168673337)

[Lijst van gebruikte afkortingen viii](#_Toc168673338)

[Inleiding 1](#_Toc168673339)

[I. Stageverslag 2](#_Toc168673340)

[1 Bedrijfsvoorstelling 2](#_Toc168673341)

[2 SkillSeeker 3](#_Toc168673342)

[2.1 Probleemstelling 3](#_Toc168673343)

[2.2 Doelstelling 3](#_Toc168673344)

[2.3 Omgeving uitvoering stageopdracht 4](#_Toc168673345)

[3 Uitwerking stageopdracht 5](#_Toc168673346)

[3.1 Planning fase 5](#_Toc168673347)

[3.2 Cloud 6](#_Toc168673348)

[3.2.1 Continuous Integration and Deployment (CI/CD) Pipeline voor Backend 8](#_Toc168673349)

[3.2.2 Werking binnen de Cluster 9](#_Toc168673350)

[3.2.3 Frontend binnen de Azure App Services 10](#_Toc168673351)

[3.2.4 Dataopslag 11](#_Toc168673352)

[3.3 Ontwikkeling en Structurering van de Applicatie 12](#_Toc168673353)

[3.4 Pipelines 14](#_Toc168673354)

[3.4.1 Frontend 14](#_Toc168673355)

[3.4.2 Backend 16](#_Toc168673356)

[3.5 Authenticatie 17](#_Toc168673357)

[3.6 Downloaden CV 19](#_Toc168673358)

[3.6.1 Azure Functions 20](#_Toc168673359)

[4 Besluit 21](#_Toc168673360)

[4.1 Conclusie stageopdracht 21](#_Toc168673361)

[4.2 Reflectie stageopdracht 23](#_Toc168673362)

[II. Verschillende hosting platformen en hun services 24](#_Toc168673363)

[1 Probleemstelling 24](#_Toc168673364)

[1.1 Onderzoekvraag 24](#_Toc168673365)

[2 Onderzoeksmethodiek 25](#_Toc168673366)

[3 Resultaten 26](#_Toc168673367)

[3.1 Literatuurstudie 26](#_Toc168673368)

[3.1.1 *AWS vs Azure:* *Which Cloud Platform* *Should You Choose in 2024?* 26](#_Toc168673369)

[3.1.2 *How to choose a* *graph databank: we compare 6 favorites* 27](#_Toc168673370)

[3.1.3 *Containerization on Azure: a comparison of services* 28](#_Toc168673371)

[3.1.4 Conclusie Literatuurstudie 28](#_Toc168673372)

[3.2 Hardware gebruik meten 29](#_Toc168673373)

[3.3 DBaaS 30](#_Toc168673374)

[3.3.1 *Serverless* DBaaS 30](#_Toc168673375)

[3.3.2 AWS Neptune Cluster instantie 33](#_Toc168673376)

[3.3.3 AuraDB 33](#_Toc168673377)

[3.4 Cloud Containers 34](#_Toc168673378)

[3.4.1 Azure 34](#_Toc168673379)

[3.4.2 AWS 35](#_Toc168673380)

[3.5 Virtual machine hosting 37](#_Toc168673381)

[3.5.1 Azure Virtual Machines 37](#_Toc168673382)

[3.5.2 AWS EC2 37](#_Toc168673383)

[3.5.3 AWS Lightsail 38](#_Toc168673384)

[3.6 Virtuele machines en containers 38](#_Toc168673385)

[3.6.1 Azure App Services 38](#_Toc168673386)

[3.6.2 AWS Elastic Beanstalk 39](#_Toc168673387)

[3.7 Praktijk 39](#_Toc168673388)

[3.7.1 AWS 40](#_Toc168673389)

[3.7.2 Azure Container Apps voor backend en databank 40](#_Toc168673390)

[3.7.3 App Services voor frontend en backend + directe integratie met databank 43](#_Toc168673391)

[Conclusie 43](#_Toc168673392)

[Hypothese: geen backend? 44](#_Toc168673393)

[Reflectie 45](#_Toc168673394)

[Bronnenlijst 46](#_Toc168673395)

Lijst van gebruikte figuren

[Figuur 1 Cloud infrastructuurplan 1 overzicht 5](#_Toc167791008)

[Figuur 2 Cloud infrastructuurplan 2 overzicht 6](#_Toc167791009)

[Figuur 3 Cloud infrastructuurplan 2 deel 1 7](#_Toc167791010)

[Figuur 4 Cloud infrastructuurplan 2 deel 2 8](#_Toc167791011)

[Figuur 5 Cloud infrastructuurplan 2 deel 3 9](#_Toc167791012)

[Figuur 6 Cloud infrastructuurplan 2 deel 4 10](#_Toc167791013)

[Figuur 7 Visualisatie berekening probleem zonder Dijkstra’s algoritme 12](#_Toc167791014)

[Figuur 8 Frontend pipeline 13](#_Toc167791015)

[Figuur 9 Frontend pipeline trigger. 14](#_Toc167791016)

[Figuur 10 Backend pipeline 15](#_Toc167791017)

[Figuur 11 Dockerfile Keycloak in production 17](#_Toc167791018)

[Figuur 12 Visuele ondersteuning optimale SkillSeeker infrastructuur 20](#_Toc167791019)

[Figuur 13 Visualisatie databank met Neo4j dashboard. 27](#_Toc167791020)

Lijst van gebruikte tabellen

[Tabel 1 Metingen van de reactietijd van SkillSeeker in de cloud 37](#_Toc167790996)

[Tabel 2 Genoteerde waarden vanuit de container logs van de backend container 38](#_Toc167790997)

[Tabel 3 Meting starttijd van Spring Boot applicatie binnen de container 38](#_Toc167790998)

[Tabel 4 Logs van de container die de databank bevat 39](#_Toc167790999)

Lijst van gebruikte afkortingen

|  |  |
| --- | --- |
| ACA | Azure Container Apps |
| ACI | Azure Container Instances |
| AKS | Azure Kubernetes Service |
| API | Application Programming Interface |
| ARO | Azure RedHat OpenShift |
| AWS | Amazon Web Services |
| CI/CD | Continuous Integration and Continuous Delivery |
| CPU | Central Processing Unit |
| DBaaS | Databank as a Service |
| ECS | Elastic Container Service |
| MSAL | Microsoft Authentication Library |
| NCU | Neptune Capacity Unit |
| OS | Operating System |
| PKCE | Proof Key for Code Exchange |
| POC | Proof Of Concept |
| RAM | Random Access Memory |
| SaaS | Software as a Service |
| TCP | Transmission Control Protocol |
| VPC | Virtual Private Cloud |

Inleiding

Mediaan Conclusion heeft nood aan een applicatie voor hun salesteam genaamd SkillSeeker. Deze applicatie maakt het mogelijk om de werknemers te filteren gebaseerd op hun vaardigheden. SkillSeeker maakt gebruik van een *graph* databank zodat vaardigheden die hard lijken op elkaar invloed hebben op een werknemer hun positie in de rangschikking zelfs al wordt er niet rechtstreeks gefilterd op deze vaardigheid.

2024 is een tijdperk waar er talloze opties zijn voor het hosten van een service. Momenteel zijn er zelfs zoveel opties dat men vaak de foute optie kiest voor de service en daardoor veel meer betaalt dan nodig.

In deze paper wordt er onderzoek gedaan naar wat de beste hosting oplossing is voor een simpele applicatie. Er wordt gekeken naar hoe krachtig de host moet zijn in verband met opslag, RAM en *computing* power terwijl er rekening gehouden wordt met het verkeer naar de applicatie.

Verschillende platformen worden onderzocht waaronder Azure cloud hosting en Amazon Web Services. Er wordt ook dieper gekeken naar de opties die op deze platformen aangeboden worden zoals betalen-per-gebruik en virtuele machines.

Mediaan Conclusion is een bedrijf dat vooral werkt met Azure en dus ook hier hun services host. Binnen Mediaan Conclusion is er vaak verwarring over wanneer het beter is om over te schakelen van betalen-per-gebruik naar een virtuele machine of zelfs om te wisselen van cloud provider. Om hier verduidelijking in te brengen is er onderzoek verricht naar verschillende cloud oplossingen om een *graph* databank te draaien.

1. Stageverslag

# Bedrijfsvoorstelling

Mediaan Conclusion is een internationaal bedrijf voor digitale transformatie en IT-oplossingen. Als toonaangevend IT-bedrijf zijn zij gespecialiseerd in het ontwikkelen van software, het leveren van advies op het gebied van digitale strategieën en het optimaliseren van bedrijfsprocessen. Met een focus op innovatie en technologie helpen zij hun klanten in diverse sectoren om te groeien en te excelleren in het digitale tijdperk. Mediaan Conclusion gelooft in het creëren van oplossingen die naadloos aansluiten op de behoeften van hun klanten, met een sterke nadruk op kwaliteit, betrouwbaarheid en klanttevredenheid.

“Het Mediaan Conclusion team combineert digitale, business en organisatorische expertise om je te brengen van waar je nu bent naar waar je wilt zijn. Medianers geloven dat technologie het verschil kan maken voor iedere organisatie. Ongeacht welke industrie, je loopt altijd een stap vooruit op de concurrentie. Maak kennis met ons! Een IT en consultancy bedrijf dat opgericht is in 1969. We werken voor onze klanten vanuit onze kantoren in Heerlen, Düsseldorf, München, Hasselt en Brussel.” [1]

De stageopdracht en het onderzoek vindt plaats binnen de technologiesector van Mediaan Conclusion en het product dat ontwikkeld wordt zal voornamelijk benut worden door de verkoopafdeling.

Binnen Mediaan Conclusion is er geen voorkeur naar 1 enkele programmeertaal of besturingssysteem. Mediaan Conclusion is een consultancy bedrijf dat hun developers verhuurt aan bedrijven met een nood aan een IT-oplossing. Mediaan Conclusion heeft dus een breed spectrum van developers met ieder zijn of haar eigen voorkeur en specialiteiten. Er is 1 methodologie die bij Mediaan Conclusion overheerst net zoals bij een groot deel van moderne IT-bedrijven en dit is de agile werkwijze. Agile betekent dat er in sprints (korte periode van tijd) gewerkt wordt en er aan het einde van elke sprint een oplevering is voor de klant zodat er tijdig ingespeeld kan worden op de opmerkingen van de klant.

Mediaan Conclusion hanteert een drievoudige aanpak die centraal staat in hun werkwijze: transparantie, competentie en flexibiliteit. [2]

Ze streven ernaar om alle prestaties traceerbaar en inzichtelijk te maken. Dit doen ze al jarenlang door gebruik te maken van objectieve meetinstrumenten. Hierdoor kunnen ze hun klanten waarborgen dat projecten binnen de afgesproken tijd en het budget worden afgerond.

Mediaan Conclusion focust zich op het aantrekken en behouden van de beste talenten. Hun team bestaat dan ook uit een breed scala aan ervaren professionals, van software engineers en data *scientists* tot data engineers, management, consultants en cybersecurityspecialisten.

Dankzij hun flexibiliteit is Mediaan Conclusion, ondanks hun kleinere omvang, in staat om complexe taken aan te pakken. Ze tonen een grote mate van inzet en durven ook in uitdagende tijden een stapje extra te zetten.

Sinds 2021 is Mediaan deel van het Nederlandse Conclusion ecosysteem. Elk bedrijf is autoriteit in zijn vakgebied. Vanuit diversiteit inspireren de bedrijven elkaar en vullen ze elkaar aan.

# SkillSeeker

## Probleemstelling

Momenteel wanneer er door een klant nood is aan een IT-oplossing zal deze een vacature opstellen waarin beschreven staat wat voor developer zij nodig hebben. Deze vacature wordt dan ontvangen door de verkoopafdeling van Mediaan Conclusion en zij gaan op zoek naar een of meerdere geschikte developers om het probleem vermeld in de vacature op te lossen. De verkoopafdeling doorzoekt een lijst van al hun ontwikkelaars, waarbij geen rekening wordt gehouden met het potentieel van een ontwikkelaar. In plaats daarvan worden alleen hun huidige vaardigheden getoond, wat resulteert in beperktere mogelijkheden. Vervolgens is het aan de verkoopafdeling om te puzzelen wanneer hun meest geschikte developer al aan een ander project bezig is.

## Doelstelling

SkillSeeker biedt een filterfunctie die developers sorteert gebaseerd op hoe goed ze voldoen aan de ingegeven parameters. Er wordt gebruikgemaakt van een *graph* databank om zo relaties te leggen tussen skills en meer accurate resultaten te geven. SkillSeeker toont ook welke developers op dat moment al in een project zitten en hoeveel uren per week deze nog vrij hebben om aan andere projecten te besteden.

SkillSeeker zal voornamelijk gebruikt worden door de verkoopafdeling, medewerkers van deze afdeling krijgen toegang tot de filterfunctie en ook om bepaalde informatie van developers zoals het loon per uur te zien. Medewerkers van het technologiedepartement die getoond worden in de applicatie, krijgen enkel toegang om hun gegevens in te voeren en hun projecten met details te bekijken. De administratorrol die gebruikt wordt door sales krijgt ook toegang om nieuwe skills toe te voegen en de relaties tussen de skills aan te passen, deze functie dient enkel gebruikt te worden door een persoon met kennis van deze skills.

## Omgeving uitvoering stageopdracht

De stageopdracht wordt ontwikkeld op de Corda Campus Hasselt in het Corda A gebouw in samenwerking met de medewerkers van het technologiedepartement van Mediaan Conclusion in hun kantoorruimte. Azure DevOps wordt gebruikt als versiebeheersysteem, om de pipelines te maken en planning op te stellen. Azure Boards op het Azure DevOps portaal wordt gebruikt om de sprint planning op een agile manier op te stellen. Er wordt gebruikgemaakt van *epics*, *features*, *user stories* en *tasks* om de voortgang van de applicatie en de resterende werkzaamheden te volgen. Azure Pipelines helpt met het op een geautomatiseerde manier implementeren van de applicatie naar een Azure Cloud omgeving op regelmatige basis, om zo snel productieproblemen te kunnen oplossen.

Na grondig onderzoek is besloten om een Neo4j databank te gebruiken, omdat deze het meest gebruiksvriendelijk is voor SkillSeeker. De '4j' in Neo4j staat voor "for Java", waardoor het logisch is dat er gekozen is voor een Java-backend om de *Application Programming Interface (API)* te ontwikkelen. Het framework dat gebruikt wordt om de API-ontwikkeling te vereenvoudigen is Spring Boot, vanwege de aanzienlijke hoeveelheid voorkennis van Spring Boot binnen het SkillSeeker-team. Voor de frontend wordt er gebruikgemaakt van Angular 17, deze keuze is gebaseerd op de expertise binnen de groep.

Wat betreft de cloudinfrastructuur op Azure, wordt er gebruikgemaakt van Azure Container Apps voor de backend en databank, en een Web App voor de frontend. Er wordt een Azure Container Registry aangemaakt om de containerimage van de databank en backend op te slaan.

# Uitwerking stageopdracht

## Planning fase

Tijdens de planningsfase begint het SkillSeeker team met de bespreking van de applicatie met de klant en een diepgaand onderzoek. De primaire focus ligt op onderzoeken hoe medewerkers die gespecialiseerd zijn in skill A ook getoond worden als er gefilterd wordt op skill B wanneer skill A en B op elkaar lijken. Toen ontstond het idee om een *graph* databank te gebruiken, aangezien deze gespecialiseerd is in databanken met veel relaties. Vervolgens is er vastgesteld wat de meest geschikte databank is om het kortste pad tussen *nodes* te identificeren door een onderzoek. Als eerste proefdatabank is Memgraph onderzocht, gezien haar reputatie als een potentieel lichtgewicht en sneller alternatief voor Neo4j. Na een aantal dagen van pogingen om een query te construeren die het kortste pad tussen talen met een link en de bijbehorende percentages kon retourneren, besloot één van de teamleden om Neo4j te evalueren. Neo4j wordt beschouwd als de oudste en meest uitgebreide *graph* databank in het domein. Binnen een korte tijdspanne leverde Neo4j de gewenste resultaten op, waardoor de focus verschoof naar het gebruik van Neo4j.

Daaropvolgend begonnen twee teamleden met het formuleren van query’s die noodzakelijk zijn voor de applicatie, om de geschiktheid van Neo4j te beoordelen. Een ander teamlid nam de verantwoordelijkheid op zich om een *Proof of Concept* (POC) op te zetten voor de klant, met als doel een duidelijke presentatie van de beoogde functionaliteiten van de applicatie.

Nadien is er onderzoek gedaan naar de integratiemogelijkheden tussen de Neo4j databank en een Java Spring Boot backend. Parallel daaraan is de frontend voorbereid met alle vereiste pakketten. Vervolgens zijn de mock-ups ontwikkelt voor de applicatie zodat de klant in de tweede meeting een visuele steun heeft. De resterende tijd van die sprint werd gewijd aan het opstellen van user story’s en het plannen van toekomstige sprints.

Aangezien de geselecteerde databank niet alleen lokaal bruikbaar moet zijn, maar ook geschikt moet zijn voor gebruik in de cloud, moet hier ook rekening mee gehouden worden. Een onderzoek naar deze cloudopties kwam al snel tot de conclusie dat AuraDB, de volledig onderhouden cloudoplossing voor Neo4j, financieel niet haalbaar is. Hierop is gestart met onderzoek naar databanken die serverless kunnen draaien, aangezien dit optimaal is voor SkillSeeker. Uit dit onderzoek kwamen twee serverless databanken naar voren die ook volledig beheerd worden, namelijk Cosmos DB van Azure en Neptune van Amazon Web Services (AWS).

Vervolgens is er tijd besteed aan het opstellen van query’s in Cosmos DB en het voorbereiden van een serverless Neptune-instantie, evenals het aanbrengen van verbeteringen aan het Proof of Concept (POC). Na enkele experimenten is besloten dat Cosmos DB en Neptune niet zo gebruiksvriendelijk zijn als Neo4j, waardoor de noodzaak ontstond om naar alternatieve oplossingen te zoeken.

## Cloud

Om deze alternatieven te zoeken is er in dit schrijven uitgebreid onderzoek naar cloudoplossingen verricht, om te bepalen of het mogelijk is om Neo4j in de cloud te gebruiken en wat de meest geschikte optie zou zijn.

Na een uitgebreid onderzoek kwam de conclusie dat Neo4j kan worden ingezet als een serverloze Container App in Azure.

Naar aanleiding van bovenstaand besluit is er een infrastructuurplan opgesteld, welke hieronder nader wordt beschreven.

A diagram of a cloud storage

Description automatically generated

Figuur 1 Cloud infrastructuurplan 1 overzicht

Het voorliggende plan neemt aan dat zowel de backend als frontend kunnen opereren binnen de gratis tier van Azure app services. Ook wordt verondersteld dat de backend kan communiceren met de databank via Transmission Control Protocol (TCP), gezien het Bolt-protocol dat door Neo4j wordt gebruikt over TCP verloopt. Deze oplossing wordt als kostenefficiënt beschouwd, gezien de backend en frontend kosteloos zijn en de container serverloos draait, wat impliceert dat er slechts wordt betaald voor daadwerkelijk verbruikte resources.

Azure Container Apps maakt gebruik van Kubernetes op de achtergrond om automatisch applicaties te schalen naar 0 wanneer ze niet worden gebruikt, en om meer instanties van een applicatie te creëren wanneer het verkeer toeneemt, zonder dat de gebruiker rechtstreeks toegang heeft tot de Kubernetes API om deze instellingen te beheren.

Uiteraard kan een container geen data opslaan die persistent is bij herstart, en daarom wordt een Azure File Share gecreëerd om de databank te *mounten*, aangezien dit eenvoudige integratie met Azure Container Apps biedt. In hetzelfde opslagaccount wordt tevens een blobopslag aangemaakt voor de foto's van de werknemers die de applicatie gebruiken. De backend verzendt een foto naar de opslag, waarna de opslag de URL van de foto terugstuurt, die vervolgens in de databank wordt opgeslagen. Het is onwenselijk om grote bestanden zoals foto's direct in een grafische databank op te slaan vanwege de impact op de databankprestaties, vandaar de aparte opslag; op deze wijze hoeft de Neo4j databank alleen de URL bij te houden.

De opslagfunctie voor foto's is helaas nog niet beschikbaar in de app vanwege tijdgebrek en probleem met permissies. Het plan zelf is echter wel degelijk.

Vervolgens is het plan goedgekeurd door een Azure expert binnen mediaan en is de infrastructuur opgezet, waarbij wordt vastgesteld dat Azure Container Apps geen externe TCP-communicatie buiten de cluster toelaat.

Om communicatie mogelijk te maken, gebruikt de cluster Ingress om poorten naar buiten of binnen de cluster bloot te leggen. De enige mogelijkheid om TCP te gebruiken voor communicatie in Azure Container Apps is binnen de cluster zelf. Er is dus geen andere optie dan de backend ook binnen dezelfde cluster in een container te laten draaien. Op basis van deze bevindingen is een nieuw infrastructuurplan opgesteld, die hieronder te zien is. Dit plan bevat ook meer details zoals de pipelines en de Azure Container Registry.

A screen shot of a computer

Description automatically generated

Figuur 2 Cloud infrastructuurplan 2 overzicht

Om een overzichtelijke uitleg te bieden, wordt bovenstaande infrastructuur besproken in vier afzonderlijke delen.

### Continuous Integration and Deployment (CI/CD) Pipeline voor Backend

Hieronder is het eerste deel van de infrastructuur te zien.

A diagram of a computer

Description automatically generated

Figuur 3 Cloud infrastructuurplan 2 deel 1

In bovenstaand deel wordt de workflow beschreven waarbij een CI/CD-pipeline wordt gebruikt om een nieuwe image van de backend te genereren aan het einde van elke sprint. Deze image wordt vervolgens gepusht naar de container registry en gebruikt om een nieuwe container te creëren en in de cluster te implementeren.

Een basis Neo4j image wordt als start genomen, waarbij slechts enkele aanpassingen worden gemaakt om de APOC plugin toe te voegen. Dit resulteert in een aangepaste image die ook wordt opgeslagen in de container registry. Aangezien er geen reguliere updates nodig zijn voor de databank, wordt handmatig een instantie gemaakt met behulp van de image uit de registry.

Als er significante updates plaatsvinden aan de Neo4j base image of aan de plugin, dient er handmatig een nieuwe container te worden gemaakt. Dit is een cruciaal aspect van het proces dat aandacht vereist om ervoor te zorgen dat de databank en plug-in correct worden bijgewerkt wanneer nodig.

### Werking binnen de Cluster

In de foto die hieronder afgebeeld is is het tweede deel van de cloud opstelling te vinden.

A screenshot of a computer

Description automatically generated

Figuur 4 Cloud infrastructuurplan 2 deel 2

Bovenstaande afbeelding belicht de werking binnen de cluster, waarbij de backend container wordt gecreëerd met behulp van de image in de registry via de pipeline, terwijl de databank container handmatig wordt gemaakt.

De databankcontainer exposeert zijn TCP-poort 7687 aan de rest van de cluster, waardoor de backend TCP-query’s kan uitvoeren op de databank om gegevens op te halen en weg te schrijven. Deze poort is gekoppeld aan de standaardpoort 7687 waarop Neo4j Bolt-verzoeken ontvangt binnen de container.

Vervolgens opent de backend zijn HTTPS-poort 8080 naar buiten toe via Ingress, zodat de frontend API-verzoeken kan sturen. De uitgaande pijl vanuit de container verwijst naar de specifieke opslag, die in foto 4 wordt besproken. Dit zorgt voor een gestroomlijnde communicatie tussen de verschillende onderdelen binnen de cluster, waardoor een soepele uitwisseling van gegevens mogelijk wordt gemaakt.

### Frontend binnen de Azure App Services

Volgende deel van de infrastructuur is afgebeeld op onderstaande foto.

A diagram of a diagram

Description automatically generated

Figuur 5 Cloud infrastructuurplan 2 deel 3

Hierboven op de foto wordt geïllustreerd hoe de frontend draait op de gratis tier van Azure App Services en communiceert met de backend via API-verzoeken. Een pipeline wordt ingezet die geactiveerd wordt telkens wanneer de backend-pipeline succesvol is voltooid. Hierdoor wordt de nieuwe versie van de frontend gebouwd en gedeployed naar de App Service.

De frontend wordt gehost op Azure App Services, waardoor het mogelijk is om deze kostenefficiënt te laten draaien op de gratis tier. Via API-verzoeken communiceert de frontend met de backend, waarbij de data-uitwisseling plaatsvindt volgens vooraf gedefinieerde endpoints. Het gebruik van een pipeline zorgt voor een geautomatiseerd proces voor het bouwen en implementeren van de frontend, wat bijdraagt aan een efficiënte ontwikkel- en releasecyclus.

### Dataopslag

Op onderstaande afbeelding is het laatste deel van de infrastructuur afgebeeld.

A diagram of a computer

Description automatically generated

Figuur 6 Cloud infrastructuurplan 2 deel 4

Dit laatste stuk van het plan, wat op bovenstaande foto afgebeeld is, belicht de dataopslag, die onveranderd is gebleven ten opzichte van het vorige infrastructuurplan.

De opslag van data omvat het gebruik van een Azure file share voor het *mounten* van de databank en een blobopslag voor het opslaan van foto's van werknemers.

## Ontwikkeling en Structurering van de Applicatie

De backend heeft een algemene structuur met een domein-, controller-, repository- en servicelaag. Deze structurering stelt het team in staat om op een georganiseerde manier aan de backendfunctionaliteit te werken.

Aan de frontend is onmiddellijk routing toegevoegd en een algemene structuur gecreëerd, waardoor ontwikkelaars kunnen werken met behulp van featurebranches. Deze aanpak bevordert een gestructureerde ontwikkeling van de frontendfunctionaliteit en maakt het mogelijk om wijzigingen geïsoleerd te ontwikkelen en te testen.

Er is een main branch, voor zowel de frontend als de backend, waarop deze veranderingen, zoals de folderstructuur en routing, onmiddellijk aangebracht zijn. Door deze veranderingen onmiddellijk aan te brengen zal elke branch die gemaakt wordt vanuit de main branch ook deze structuur aanhouden wat samenwerking bevordert. Vervolgens is er een development branch aangemaakt vanuit deze main branch die bijgevolg ook deze veranderingen bevat.

Voor elke user story die tijdens de planningsfase is gecreëerd, wordt een nieuwe branch aangemaakt met de naam "feat-{id user-story}-{korte beschrijving}". Deze benoeming maakt het gemakkelijk om de branches te identificeren en te koppelen aan specifieke user stories. Nadat een user story is ontwikkeld, wordt de branch met behulp van een *pull* *request* samengevoegd in de development branch.

Na een succesvolle sprint van 2 weken wordt de development branch samengevoegd in de main branch. Vervolgens wordt met behulp van een pipeline deze code in de cloud geplaatst om online beschikbaar te zijn. Het implementeren van deze code in de cloud stelt het team in staat om de nieuwste functionaliteit direct in de cloud uit te testen.

Het team was opgesplitst in twee frontendontwikkelaars en twee backendontwikkelaars, waarvan één ook onderzoek deed naar de cloud. Dit zorgt ervoor dat elk teamlid werkt in het veld waar hij het best in is om zo op een efficiënte manier een kwaliteitsproduct op te leveren.

Nadat het grootste deel van de backend ontwikkeld is kwam de conclusie dat onze huidige query niet het optimale pad teruggeeft en dat we Dijkstra’s algoritme nodig hebben voor SkillSeeker. Zonder dit algoritme zou bijvoorbeeld een werknemer die gezocht wordt op taal 2 een score krijgen van {zijn score \* 20 procent}, terwijl hij eigenlijk een betere score zou moeten krijgen als hij een langer pad zou nemen langs taal 3. Als het pad langs taal 3 wordt genomen, heeft hij een score van {zijn score \* 80 procent \* 70 procent}. De afbeelding hieronder biedt een visuele steun voor het voorbeeld.

A diagram of a diagram

Description automatically generated

Figuur 7 Visualisatie berekening probleem zonder Dijkstra’s algoritme

Het resultaat van een onderzoek naar dit algoritme kwam met de oplossing dat een extensie genaamd "APOC" aan de databank moet toegevoegd worden. Awesome Procedures on Cypher (APOC) is een uitgebreide bibliotheek met procedures en functies die extra functionaliteit bieden voor Neo4j databanken. Het biedt onder andere ondersteuning voor geavanceerde algoritmen, waaronder Dijkstra's algoritme, dat nodig is voor SkillSeeker.

## Pipelines

### Frontend

Voor het implementeren van *Continuous Integration and Continuous Delivery (*CI/CD) zijn pipelines gebruikt. Een pipeline is een stuk code dat wordt uitgevoerd wanneer aan bepaalde vereisten wordt voldaan. Figuur 1 toont de code van de pipeline als visuele leidraad bij onderstaande uitleg.

A screenshot of a computer program

Description automatically generated

Figuur 8 Frontend pipeline

Zoals blijkt uit Figuur 8 start de pipeline met het specificeren dat de code wordt uitgevoerd op een Ubuntu-systeem met de laatste versie.

Vervolgens wordt Node.js geïnstalleerd als *package manager* op het systeem. Daarna navigeert de pipeline naar de applicatie en gebruikt de *package manager* om de Angular *command-line interface* te installeren. Nadat deze *command-line* is geïnstalleerd, kunnen de resterende packages worden geïnstalleerd.

Ten slotte bouwt dit deel van de pipeline de applicatie met het “ng build” commando en specificeert dat het voor productie moet worden gebouwd, zodat de applicatie andere instellingen gebruikt. De frontend applicatie bevat 2 bestanden met settings, één voor productie en één voor ontwikkeling. In de pipeline moet dus het juiste bestand worden gekozen. Dit commando resulteert in een folder genaamd “browser” die alle benodigde bestanden bevat om de applicatie te draaien.

De tweede taak van de pipeline is om de inhoud van deze folder naar de App Service in de Azure Portal te sturen. De authenticatie wordt uitgevoerd aan de hand van een serviceconnectie die wordt aangemaakt voor de pipeline.

A screenshot of a computer

Description automatically generated

Figuur 9 Frontend pipeline trigger.

Bovenstaande screenshot illustreert de trigger die de pipeline activeert. De pipeline start telkens wanneer de backend pipeline succesvol is voltooid. De logica hierachter is dat als zowel de frontend- als de backend-pipeline tegelijkertijd worden gestart, maar de backend-pipeline faalt, gebruikers mogelijk nieuwe frontend-functies zien die vervolgens een foutmelding veroorzaken omdat het backend-eindpunt niet up-to-date is. Daarom moet de backend altijd up-to-date zijn voordat de frontend wordt geüpdatet.

### Backend

Voor correcte CI/CD moet er een pipeline zijn voor zowel de backend als frontend. Figuur 2 toont de pipeline voor de backend.

A screenshot of a computer program

Description automatically generated

Figuur 10 Backend pipeline

De backend pipeline, zoals weergegeven in Figuur 10, wordt geactiveerd telkens wanneer er een verandering plaatsvindt op de *main branch*. Bovendien is de *main branch* beveiligd, zodat er geen directe *commits* op kunnen worden uitgevoerd. Aanpassingen kunnen alleen worden gedaan via een *pull request* die door het hele team wordt beoordeeld.

De pipeline begint met het gebruik van een up-to-date Ubuntu-image als besturingssysteem. Vervolgens worden enkele variabelen gedefinieerd die door de hele pipeline kunnen worden gebruikt.

In Java is het niet mogelijk om expliciet aan te geven of het om een productie- of ontwikkelomgeving gaat. Daarom wordt er een andere methode toegepast.

De Spring Boot-applicatie bevat een bestand genaamd “application.properties” dat de URL van de databank specificeert. In ontwikkeling moet dit naar “localhost” verwijzen, maar in productie moet dit naar het adres van de container binnen de cluster refereren. Daarom is er een tweede bestand genaamd “application-production.properties” dat de productie-URL van de databank bevat. De eerste stap in de pipeline wisselt de inhoud van deze bestanden om, zodat de juiste URL in het “application.properties”-bestand wordt ingesteld.

Vervolgens wordt de Maven-taak gebruikt om de benodigde bibliotheken die zijn gedefinieerd in het pom.xml-bestand te downloaden. Nadat deze zijn gedownload, kan het Docker-commando worden gebruikt om een image van de code te maken en deze vervolgens naar de Azure Container Registry (ACR) te pushen. De ACR wordt gedefinieerd door een Azure Container Registry-verbinding die ook meteen toegang verleent.

Ten slotte wordt de AzureContainerApp-taak gebruikt om het signaal te geven dat er een nieuwe container moet worden gemaakt met het zojuist naar de ACR gestuurde image. Hiermee wordt de implementatie van de backend naar de productieomgeving afgerond.

## Authenticatie

Authenticatie is vanzelfsprekend een belangrijk onderdeel van een applicatie die gevoelige data van medewerkers verwerkt. In SkillSeeker zijn er 2 lagen voor authenticatie. De eerste laag maakt gebruik van Azure Entra ID, waarbij is ingesteld dat alleen personen binnen de Mediaan tenant toegang hebben tot de frontend. Uiteraard is dit op zichzelf niet voldoende beveiliging.

De tweede laag van beveiliging omvat het onderscheid tussen gebruikers. Natuurlijk moeten alle gebruikers een eigen account hebben om hun gegevens in te voeren, maar er zijn ook verschillende rollen toegekend. Een standaardgebruiker mag bijvoorbeeld alleen zijn eigen gegevens invoeren en projecten bekijken, terwijl een gebruiker die deel uitmaakt van het sales team toegang krijgt tot de filterfunctie van de applicatie.

Om dit te realiseren is origineel Keycloak gebruikt. Keycloak biedt een veilige manier om gebruikers te authenticeren en te autoriseren door gebruik te maken van *Proof Key for Code Exchange* (PKCE). Omdat er geen directe toegang is tot de Mediaan Entra ID voor SkillSeeker, waardoor niet kan worden vastgesteld tot welk departement een gebruiker behoort, wordt Keycloak ingezet. In Keycloak is een administrator aangemaakt die toegang heeft tot de filterfunctie, en elke medewerker moet zich registreren en inloggen via een registratiepagina en een inlogpagina om zo zijn gegevens aan de applicatie toe te voegen.

Om Keycloak toe te voegen aan SkillSeeker wordt hiervan een container gemaakt en toegevoegd aan de Azure Container Apps Environment.

De eerste stap is om een image van Keycloak lokaal van de officiële pagina te downloaden. Vervolgens wordt deze een passende tag gegeven en gepusht naar de Azure Container Registry waar ook de image van de databank en backend staan. Vervolgens wordt er een nieuwe Container App aangemaakt in de omgeving.

Vanuit een meting is er de conclusie getrokken dat keycloak toekomt met slechts 0.25 vCPU en 2 GB RAM. Azure laat enkel toe om vCPU:RAM in een 1:2 ratio te selecteren. Om 2 GB RAM te krijgen is er dus 1 vCPU geselecteerd.

De Keycloak website bevat een Dockerfile gemaakt voor Keycloak in productie die weergegeven wordt op de afbeelding hieronder.

A screen shot of a computer program

Description automatically generated

Figuur 11 Dockerfile Keycloak in production

Hieruit kan afgeleid worden dat er een databank nodig is en niet slechts een opslagplaats om op een gepaste manier Keycloak in productie te gebruiken.

Eerst wordt er dus een PostgreSQL databank in de cloud aangemaakt en vervolgens wordt het passwoord, username en url hiervan ingevuld en een image gemaakt met bovenstaande Dockerfile. Dit image wordt dan gepusht naar de repository en gebruikt om een Keycloak container te maken.

Helaas heeft de klant laten weten dat aangezien de applicatie hoogstwaarschijnlijk niet in productie gebruikt zal worden, het geen meerwaarde heeft om bezig te zijn met authenticatie, en is dit in zijn geheel achtergelaten.

## Downloaden CV

Mediaan wil graag dat SkillSeeker over de functionaliteit beschikt om een CV van een werknemer te downloaden die dan doorgegeven kan worden aan de klant. Momenteel maken zij gebruik van een template die dan iedere keer handmatig ingevuld wordt door de werknemer met informatie die interessant is voor de klant.

Door gebruik te maken van Word maillings is het mogelijk om de velden in een template in te vullen met behulp van bijvoorbeeld data in een excel spreadsheet. Er is uitbundig onderzoek verricht naar hoe deze maillings gebruikt kunnen worden om de template in te vullen met data van de gebruiker in de applicatie waarvoor een CV gegenereerd wordt.

De voornaamste optie was om dit te doen in de backend met behulp van Javacode maar na een uitgebreid onderzoek werd de conclusie gemaakt dat meeste Java libraries die gebruikt kunnen worden om Word documenten aan te passen heel outdated zijn en vaak niet meer ondersteund worden, kwetsbaarheden bevatten, geen merge tussen Word documenten en gebruikerdata kunnen doen of simpelweg te complex omdat er veel kennis van de onderliggende technologie nodig is.

Om dit probleem op te lossen is er meer buiten de grenzen van de Java backend gekeken en is er ontdekt dat deze merge tussen data uitvoeren heel simpel is in Python. Natuurlijk is het niet de bedoeling om een volledig extra container toe te voegen aan de SkillSeeker infrastructuur simpelweg voor een Python API die een document invult en terugstuurt dus is er onderzoek begonnen naar Azure Functions.

### Azure Functions

Volgens het onderzoek naar cloudoplossingen blijkt Azure Functions een geschikte tool te zijn om een kleine, onafhankelijke *codebase* in de cloud te implementeren. Functions zijn met name nuttig wanneer deze kleine codebase een ander framework gebruikt dan de rest van de applicatie. In het geval van SkillSeeker is de primaire applicatie een Java Spring Boot-applicatie, maar is er nood aan een kleine hoeveelheid Python-code.

In het kader van SkillSeeker is er, met het oog op kostenbeheersing, gekozen voor een Consumption-abonnement in plaats van een Premium-abonnement. Dit betekent dat de infrastructuur minder performant is, maar wel goedkoper. Het belangrijkste verschil tussen de twee abonnementen is de manier waarop de servers omgaan met piekmomenten in het aantal inkomende *requests*.

Bij een Consumption-abonnement worden servers bij piekmomenten opnieuw opgestart (cold start). Dit kost extra tijd, omdat de benodigde software eerst moet worden gedownload en geïnstalleerd. Bij een Premium-abonnement daarentegen zijn de servers altijd warm en klaar voor gebruik, wat resulteert in een snellere responstijd, maar ook in hogere kosten.

De keuze voor een Consumption-abonnement is in dit geval gerechtvaardigd, omdat de extra kosten voor een Premium-abonnement niet opwegen tegen de verwachte toename in gebruiksgemak. De impact van de cold start op de gebruikerservaring is acceptabel binnen de context van de SkillSeeker-applicatie.

Naast de Consumption- en Premium-abonnementen biedt Azure ook een Container Apps-omgeving. Hierbij draait de functie in een container en maakt gebruik van de Kubernetes API. Dit geeft de gebruiker meer flexibiliteit en maakt het mogelijk om complexere functies te ontwikkelen die het cluster kunnen aansturen.

Een voorbeeld hiervan is het uitvoeren van query’s op een Neo4j container via TCP. Echter, deze optie bevindt zich momenteel nog in de testfase en is nog niet volledig uitgewerkt door Azure. Deze updates brengen enkele risico's met zich mee en kan nog veranderen.

Bovendien zijn complexe functies zoals deze niet relevant voor het downloaden van een CV. Daarom wordt er in dit project geen gebruik gemaakt van de Container Apps-omgeving.

Voor de Python-functie is de docxtpl-bibliotheek gebruikt. Deze bibliotheek maakt het mogelijk om een Word-sjabloon te laden en daarin codeachtige stukjes tekst te plaatsen, zoals {{ name }}. Wanneer er vervolgens een variabele met de naam name aan de geladen sjabloon wordt toegevoegd, wordt deze op de plaats van de code ingevuld.

Op basis van dit mechanisme is een complete template aangemaakt en toegevoegd aan de functie. De functiecode verwacht een JSON-object met alle benodigde gegevens van de gebruiker. Dit object wordt vervolgens toegevoegd aan de template en het ingevulde template wordt teruggestuurd naar de frontend van de applicatie waar het werd opgeroepen. De gebruiker kan vervolgens het document downloaden.

# Besluit

## Conclusie stageopdracht

De stageopdracht is beëindigd met een functioneel product volgens de vereisten van de klant. SkillSeeker kan zowel lokaal als in de cloud gebruikt worden. Voor het lokaal gebruiken van de applicatie is er documentatie aanwezig, voor het gebruik in de cloud is dit momenteel slechts zeer minimaal aangezien er vanuit het bedrijf gevraagd is om de authenticatie en authorisatie niet verder te ontwikkelen. De cloud versie van de applicatie toont wel interessante functionaliteiten zoals het serverloos, tegen lage kost, gebruikmaken van de filter functionaliteit. Ook is er in de cloud de Azure Function die gebruikt wordt om de CV te downloaden, deze Function staat los van de rest van de applicatie en wordt zowel benut wanneer de applicatie lokaal draait als wanneer de applicatie in de cloud draait. Deze functie bevat ook uitgebreide documentatie over hoe deze mogelijks gebruikt kan worden in andere projecten.

Na verdere verkenning van Azure Functions voor het downloaden van CV's, blijkt dat dit ook geschikt is voor de volledige backend. Spring Boot kan via de juiste Spring Cloud-plugin samenwerken met Azure Functions. Echter, gezien de integratie van Functions met .NET en de uitgebreidere ondersteuning en documentatie van Microsoft voor dit platform, is het aantrekkelijker om de backend in .NET te ontwikkelen.

Bij een volledige herontwikkeling van de applicatie is het aantrekkelijker om de cloud als uitgangspunt te nemen en te werken vanuit Azure Functions, waarbij elk endpoint van de applicatie een aparte functie is. Dit kan complexer zijn wanneer integratie met een Neo4j databank in een Kubernetes-cluster vereist is. In dat scenario is het gunstiger om te focussen op het leren van Cosmos DB als de SkillSeeker-graphdatabank, vanwege de eenvoudigere integratie met andere Azure-applicaties.

Tijdens de ontwikkeling van de CV-functionaliteit is er geconcludeerd dat er aanzienlijk meer data aan objecten kan worden toegevoegd om gebruikers meer details te tonen. Echter, grafische databanken zoals Neo4j zijn niet ontworpen om grote hoeveelheden data in elke node op te slaan, hoewel dit mogelijk is, maar niet optimaal. Daarom is het aantrekkelijker om twee instanties van Cosmos DB te gebruiken: één die fungeert als een *graph* databank voor filterfunctionaliteit en één die opereert als een eenvoudige documentdatabank voor de details van gebruikers, projecten, etc. Zoals hieronder afgebeeld.

A diagram of a network

Description automatically generated

Figuur 12 Visuele ondersteuning optimale SkillSeeker infrastructuur

De bovenstaande figuur toont een vereenvoudigde voorstelling van de optimale infrastructuur, zonder pipelines, authenticatie, storage accounts, etc.

In dit scenario blijft de frontend ongewijzigd. De backend wordt volledig opgesplitst in Azure Functions, waarbij elk endpoint een eigen functie krijgt. Dit maakt het mogelijk om endpoints afzonderlijk aan te roepen zonder de volledige applicatie op te starten. De domeinklassen worden gedefinieerd in aparte bestanden binnen de Function App en worden geïmporteerd door de Functions die ze nodig hebben, wat consistentie in de code garandeert.

De databank wordt opgesplitst in twee instanties van Cosmos DB: één voor documenten en één voor grafieken. Hierdoor wordt de grafiekdatabank niet belast met het opslaan van grote hoeveelheden data, maar kan deze zich focussen op waarvoor hij ontworpen is.

## Reflectie stageopdracht

De ontwikkeling van de applicatie verliep overwegend soepel, met weinig obstakels. De samenwerking binnen het team was prettig en kenmerkte zich door een goede balans tussen werkervaring en informele sfeer.

Echter, tegen het einde van de projectperiode bleek dat de planning niet optimaal was. Te vaak werden er vlak voor een deadline nog aanpassingen doorgevoerd, wat tot problemen leidde. Daarnaast werd er soms gestart met functionaliteiten die niet in de planning van de sprint pasten, als gevolg van een onjuiste inschatting van de benodigde ontwikkeltijd.

Het grootste knelpunt dat voortkwam uit de gebrekkige planning was de overhaaste start van de ontwikkeling zonder voldoende onderzoek. Zo werd er spoedig gekozen voor Spring Boot Java als backend, zonder rekening te houden met de implicaties voor cloud deployment. Achteraf bleek dat er, in het kader van serverloze container-apps, betere Java-alternatieven beschikbaar zijn, zoals het Quarkus-framework, dat specifiek ontworpen is voor snelle opstarttijden in containers. Bovendien is een .NET-backend niet eens overwogen omdat binnen het team een voorkeur is voor Java, het resultaat hiervan is dat wij gekozen hebben voor Java zonder eerst te overwegen of .NET voordeliger kan zijn met onze cloud situatie.

1. Verschillende hosting platformen en hun services

# Probleemstelling

## Onderzoekvraag

**Welke hostingmogelijkheid is het meest geschikt om een kleinschalige graph databank met weinig interactie (of gebruik) te hosten?**

Binnen Mediaan Conclusion is er nood aan de SkillSeeker-applicatie om meer overzicht te krijgen van de beschikbare developers. De SkillSeeker-applicatie zal gebruikt worden door de verkoopafdeling. De medewerkers van de verkoopafdeling weten weinig van technologie af en deze applicatie moet dus via een simpele browserlink toegankelijk zijn. Hiervoor zal de SkillSeeker-applicatie gehost worden maar omwille van het grote assortiment aan opties is er binnen Mediaan Conclusion verwarring over wat de beste oplossing is. SkillSeeker is een applicatie die vrij weinig gebruikt zal worden maar omdat deze gebruikmaakt van een graph databank is de kostenberekening wat complexer.

Om deze onderzoeksvraag te beantwoorden zijn er verschillende deelvragen opgesteld.

Welke belangrijke aspecten onderscheiden AWS en Azure van elkaar als cloud-providers? Dit omvat een analyse van hun sterke en zwakke punten, prijsstructuren, beschikbare services en geschiktheid voor specifieke *use-cases*.

Welke hoeveelheid CPU- en RAM zijn nodig om een Neo4j-databank met ongeveer 400 nodes en 1500 relaties soepel te laten draaien? Dit onderzoek omvat het analyseren van prestatiebenchmarks en het bepalen van optimale configuraties op basis van de complexiteit van de databank en de verwachte workload.

Welke databanktaal biedt de beste balans tussen gebruiksgemak en een lage leercurve? Dit onderzoek omvat het evalueren van verschillende *graph* databanken, rekening houdend met factoren zoals *syntax*, documentatie en beschikbare leermiddelen.

# Onderzoeksmethodiek

De eerste stap die ondernomen wordt om een marktanalyse te doen is onderzoeken hoeveel resources er nodig zijn voor de databank zodat metingen accuraat kunnen gebeuren. Er wordt een nepdatabank gevuld met 200 test gebruikers met nep skills, een *mock* databank die ongeveer dezelfde grootte zal hebben als de productiedatabank. Deze databank draait in een Docker-container zodat we de gebruikte hoeveelheid CPU en RAM kunnen meten.

Vervolgens wordt er een extensieve analyse gedaan van de vele opties. Een van deze opties is het hosten door bedrijven die specialiseren in hosten en onderhouden van een databank zoals GrapheneDB, AuraDB en DeltaBlue. Bedrijven die *Databank as a Service* (DBaaS) aanbieden maken nog steeds gebruik van de infrastructuur van grote cloud providers zoals AWS, GCP en Azure. Er wordt dus onderzoek gedaan naar welke extra voordelen er zijn om een middelman in te schakelen.

Er wordt eerst onderzoek gedaan naar welke databank bruikbaar en gebruiksvriendelijk is voor onze noden. Vervolgens wordt er onderzocht wat de opties zijn om deze meest geschikte databank te hosten. Deze opties worden eerst gefilterd op prijs. Een kleine applicatie zoals SkillSeeker mag nooit meer dan 10 euro per maand kosten. Ten slotte wordt de uitgekozen optie opgesteld in de cloud om de gebruiksvriendelijkheid en leercurven te testen.

Zelf de infrastructuur aankopen is geen optie omdat dit simpelweg niet meer gedaan wordt in de tijd van cloud *computing* waarin wij leven en elk bedrijf dat “zelf de databank host” maakt bijna altijd gebruik van de infrastructuur van AWS, GCP of Azure.

# Resultaten

## Literatuurstudie

### *AWS vs Azure:* *Which Cloud Platform* *Should You Choose in 2024?*

In volgend artikel door Shyamli Jha worden AWS en Azure vergeleken. Dit is een belangrijk onderwerp voor een onderzoek over cloud opties. Voordat er een service gekozen kan worden moet er eerst besloten worden op welk platform.

Eerst wordt de geschiedenis van AWS besproken. AWS is opgericht om Amazon’s e-commerce activiteiten te onderhouden. Na enige tijd zijn ze begonnen met hun services te bieden aan externen omdat zij de ervaring hadden en de computer grondstoffen om gemakkelijk services voor anderen te onderhouden. Met de dure investering die serverinfrastructuur is zijn velen deze services beginnen gebruiken en is AWS snel een van de grootste cloud providers ter wereld geworden. AWS biedt zijn services aan `s werelds bekendste merken zoals Netflix, Spotify en Ubisoft. [3]

Vervolgens gaat het over Azure die is opgericht als Windows Azure door Microsoft maar is al snel hernoemd naar Microsoft Azure. Azure is een cloudplatform die gebruikers instaat stelt om hun applicaties online beschikbaar te maken zonder hun eigen infrastructuur op te moeten zetten. [3]

In het artikel wordt er gesproken over dat de AWS-documentatie overzichtelijker is en gemakkelijker om te gebruiken dan die van Azure. Dit komt niet overeen met de ondervindingen, op basis van documentatie is er het besluit geformuleerd dat Azure en AWS gelijk zijn. [3]

AWS biedt meer flexibele licenties dan Azure dit betekent dat wanneer het neerkomt op een cloudexpert worden het interessanter is om AWS te beheersen. Aan de andere kant zorgt Azure er vaak voor dat zijn partners volledig afhankelijk zijn van Microsoft-producten. Hierdoor heeft iemand met een Azure-certificaat een grotere kans om bij deze partners binnen te komen. [3]

AWS stelt gebruikers in staat om *Virtual Private Cloud* of VPC-verbindingen te maken. Dit betekent dat gebruikers privé netwerken hebben en deze gemakkelijker kunnen beveiligen. Azure maakt gebruik van *virtual networks* wat conceptueel hetzelfde is maar de implementatie achterliggend gebeurt anders. [3]

Voor *logging* en monitoren gebruiken AWS en Azure allebei hun eigen tool die functioneel gelijk zijn aan elkaar. [3]

Een van de grootste verschillen is dat AWS veel meer ondersteuning heeft voor *open-source software* dan Azure. Het artikel heeft hier gelijk dat Microsoft het moeilijker maakt om producten te integreren die niet door hen ontwikkeld zijn. [3]

De conclusie van het artikel dat parallel loopt met dit onderzoek is dat zowel AWS als Azure zeer populair en uitgebreid zijn, en een breed scala aan diensten bieden met hun respectieve voor- en nadelen. Voor een klein project zoals SkillSeeker zal het verschil tussen AWS en Azure verwaarloosbaar zijn.

Het uitgevoerde onderzoek toont echter aan dat het Azure-dashboard meer beginnersvriendelijk is dan dat van AWS. Mediaan Conclusion, het bedrijf dat de opdracht heeft gegeven voor de ontwikkeling van SkillSeeker, maakt al geruime tijd gebruik van Azure. Daarom wenst Mediaan Conclusion dat als de verschillen verwaarloosbaar zijn, Azure wordt gebruikt voor de infrastructuur van SkillSeeker.

### *How to choose a* *graph databank: we compare 6 favorites*

In dit artikel worden 6 *graph* databanken vergeleken, waaronder AWS Neptune, Azure Cosmos DB en Neo4j. Na eerdere onderzoeken naar hostingopties is besloten dat deze drie de beste opties zijn voor SkillSeeker, daarom worden de andere 3 niet besproken.

De eerste optie die in het artikel wordt besproken, is AWS Neptune. Het grootste voordeel is dat Neptune volledig wordt onderhouden door AWS zelf, waardoor de gebruiker zich geen zorgen hoeft te maken over updates, onderhoud en andere administratieve taken. Omdat Neptune een AWS-product is, biedt het ook eenvoudige integratie met andere AWS-producten, wat het gebruiksvriendelijker maakt dan de meeste databanken. Een van de sterkste punten van Neptune is dat het ondersteuning biedt voor meerdere querytalen, waaronder TinkerPop Gremlin en openCypher. [4]

De mening van Cambridge Intelligence dat Neptune goed wordt onderhouden door AWS en dus ook gebruiksvriendelijke integratie biedt gaat gelijk met de bevinding binnen dit onderzoek. Waar deze studie niet strookt met de bevindingen binnen dit onderzoek, is de ondersteuning van meerdere talen. Neptune ondersteunt deze talen theoretisch wel, maar de documentatie over hoe deze querytalen binnen Neptune te gebruiken is bijna volledig afwezig, wat Neptune verre van gebruiksvriendelijk maakt. Bovendien is er vrijwel geen documentatie over hoe Neptune te laten communiceren met een Spring Boot Java-backend. Als Neptune wordt gebruikt, is de kans groot dat er veel tijd en dus kosten nodig zijn om de applicatie uit te breiden en te onderhouden, omdat nieuwe ontwikkelaars moeten leren hoe ze ermee moeten werken.

Vervolgens wordt Azure Cosmos DB besproken. Net als Neptune is het grootste voordeel van Cosmos DB dat het rechtstreeks deel uitmaakt van een groot ecosysteem zoals Azure, waardoor 99,999% *uptime* gegarandeerd is en de integratie met andere Azure-services soepel verloopt. Wanneer er gebruik wordt gemaakt van Cosmos DB, hoeft de ontwikkelaar niet te denken aan schalen wanneer het verkeer naar de website toeneemt, dit wordt allemaal geregeld door Microsoft. Cosmos DB maakt gebruik van Gremlin als querytaal, wat volgens Cambridge Intelligence één van de meest volwassen querytalen is. [4]

Dit paper heeft gelijk dat Azure Cosmos DB veel voordelen heeft omdat het deel uitmaakt van het Microsoft-ecosysteem. Waar het artikel verschilt van dit onderzoek, is de gebruiksvriendelijkheid van de Gremlin-taal. Na enkele experimenten is de conclusie getrokken dat de Gremlin-taal veel complexer is en minder beginner-vriendelijk dan andere talen. Bovendien is het dashboard van Cosmos DB helemaal niet overzichtelijk.

Ten slotte wordt Neo4j besproken, de oudste *graph* databank van de 3. Neo4j heeft niet het voordeel om deel uit te maken van een groot ecosysteem zoals Neptune en Cosmos DB. Wat Neo4j wel heeft, is het voordeel om de oudste *graph* databank te zijn. Cambridge Intelligence vertelt hier dat Neo4j een zeer gebruiksvriendelijke databank is voor beginners, met veel documentatie en tutorials. Omdat Neo4j zo oud is, heeft het ook een groot aantal functies, zoals meerdere visualisatieopties. [4]

Het onderzoek loopt hier gelijk met dit paper. Na het uitproberen van Neo4j werd al snel duidelijk dat de querytaal veel meer uitgewerkt en gebruiksvriendelijk is. De documentatie van Neo4j is, zoals het artikel vermeldt, overal te vinden en zeer goed uitgewerkt en up-to-date gehouden.

### *Containerization on Azure: a comparison of services*

In deze bron bespreekt Jasper Baljeu de verschillende opties om containers te hosten binnen Azure. In dit document wordt niet alleen gekeken naar het beste platform om de Neo4j databank te hosten, maar ook naar de specifieke opties binnen dit platform. Er kan dus gebruikgemaakt worden van het onderzoek van meneer Baljeu om een vergelijking uit te voeren. [5]

De eerste service die wordt besproken, is Azure App Service. Deze kan gebruikt worden om op een gebruiksvriendelijke manier een container in de cloud te implementeren. Volgens meneer Baljeu is deze optie het meest geschikt wanneer een applicatie een continue stroom van verkeer heeft. Aangezien de SkillSeeker-applicatie niet consistent wordt gebruikt, is dit niet de beste optie voor SkillSeeker. [5]

Vervolgens wordt Azure Container Instances (ACI) besproken. ACI is eenvoudigweg het hosten van een enkele container, waardoor het snel en eenvoudig is, maar zonder extra functies. ACI biedt geen schaalbaarheid of load *balancing* over meerdere instanties, omdat het slechts één enkele container ondersteunt. Deze optie is het meest geschikt wanneer de databank slechts af en toe wordt gebruikt en is daarom beter geschikt dan Azure App Services voor het hosten van de SkillSeeker-databank. [5]

Dan wordt gesproken over Azure Kubernetes Service of AKS. AKS stelt de gebruiker in staat om zijn eigen Kubernetes-cluster in de cloud te beheren op de infrastructuur van Azure. Hiervoor is specifieke kennis vereist en is het alleen nuttig wanneer er meerdere containers worden gebruikt. [5]

Meneer Baljeu gaat vervolgens verder over Azure RedHat OpenShift. Deze service is bedoeld om gebruiksvriendelijker te zijn dan Kubernetes, met meer ondersteuning, maar tegen een hogere prijs. Net als Kubernetes is dit niet relevant voor SkillSeeker. [5]

Tot slot wordt gesproken over Azure Container Apps (ACA). Dit is de nieuwste containerservice aangeboden door Microsoft. ACA biedt de gebruiker een omgeving die gebruikmaakt van Kubernetes zonder de gebruiker directe toegang te geven tot de Kubernetes API. ACA is de perfecte tool voor de SkillSeeker-applicatie, omdat het een *serverless* databank mogelijk maakt die alleen resources gebruikt tijdens gebruik en niet constant actief is. [5]

### Conclusie Literatuurstudie

Uit het eerste onderzoek blijkt dat voor een kleine applicatie zoals SkillSeeker er geen significant verschil is tussen AWS en Azure. Als beginner wordt de conclusie getrokken dat het Azure-dashboard meer beginner-vriendelijk aanvoelde dan dat van AWS, wat van belang is bij de keuze. Uiteindelijk is Mediaan Conclusion van mening dat, tenzij er een specifiek en groot voordeel was om AWS te gebruiken, er gebruikgemaakt wordt van Azure vanwege hun ervaring hiermee.

Het tweede onderzoek onthult dat Neo4j verreweg de meest gebruiksvriendelijke en overzichtelijke *graph* databank is. Door de leeftijd van Neo4j zijn er ook tal van extensies die van pas komen voor SkillSeeker.

Uit het derde onderzoek blijkt dat de beste Azure-service voor een kleine applicatie de Azure Container Apps zijn. Deze apps kunnen *serverless* worden gebruikt om zo de kosten voor de applicatie te minimaliseren.

Op basis van deze drie onderzoeken blijkt dus dat Azure, met een *serverless* container waarop Neo4j draait, de beste optie is voor SkillSeeker. Dit is ook de keuze die is gemaakt voor de ontwikkeling van de applicatie.

## Hardware gebruik meten

Door middel van een Docker-container kunnen de resources die de databank zal gebruiken eenvoudig worden gemeten. Het gebruik van het commando "docker container stats {containerId}" levert een continue stroom van gegevens op over het gebruik van resources. [6]

De eerste stap in dit proces is het creëren van een container waarin een instantie van een Neo4j databank draait. Voor dit doel is er gebruik gemaakt van een docker-compose.yml-bestand in de backend repository. Een docker-compose.yml vereenvoudigt het proces van het aanmaken van Docker-containers en biedt tevens de mogelijkheid om extra eigenschappen toe te kennen.

Na het activeren van het docker-compose.yml-bestand kan er op het Docker-desktopdashboard waargenomen worden dat er een nieuwe container is gecreëerd. Deze container wordt gegroepeerd onder skillseeker-backend, omdat dit de naam is van de map waarin de Java-applicatie en het docker-compose.yml-bestand zich bevinden.

Vervolgens wordt er gebruik gemaakt van een StartupRunner-klasse in de Java Spring Boot-applicatie, die wordt geactiveerd bij het opstarten van de applicatie. De methode controleert eerst of de databank leeg is en creëert vervolgens alle vaardigheden, drie beheerdersgebruikers, dertig projecten, tweehonderd medewerkers en al hun relaties voor de applicatie in de databank met behulp van Neo4j-query's.

A screenshot of a computer screen

Description automatically generated

Figuur 13 Visualisatie databank met Neo4j dashboard.

Nu is er een Docker-container met een databank die naar schatting even groot zal zijn als de databank die in productie zal worden gebruikt. Het commando “docker container stats {containerId}” maakt een stream in de console die elke 0.5 seconden nieuwe output genereerd over de gebruikte resources.

De meest veeleisende query die de applicatie zal uitvoeren, is het ophalen van alle medewerkers met hun vaardigheden en hun kortste pad naar de gespecificeerde vaardigheid. Daarom wordt om de geschatte resources te bepalen die de databank in productie zal gebruiken, deze query uitgevoerd op de databank met behulp van het Neo4j dashboard.

Na het uitvoeren van de query vindt er een toename in het gebruik van hardware resources, wat waargenomen kan worden aan de datatransmissie via het eerder uitgevoerde commando.

Hieruit wordt afgeleid dat de kleine databank op het moment dat zijn zwaarste query wordt uitgevoerd, minder dan de helft van één CPU-kern gebruikt. Het uitvoeren van de query heeft geen invloed op de hoeveelheid gebruikte RAM; dit hangt af van de grootte van de databank, en we zien dat de mock-databank slechts 620 MB RAM nodig heeft.

Uit dit experiment wordt de conclusie gemaakt dat de databank kan draaien met slechts 1 CPU en 1 GB RAM. De prijsschatting wordt gedaan op basis van deze resources. Indien nodig wordt er ook rekening gehouden met het feit dat er ongeveer 174 werkuren in een maand zijn en dat de applicatie buiten deze uren niet wordt gebruikt.

Achteraf is besloten dat we de APOC plugin voor Neo4j nodig hebben om het Dijkstra's Shortest Path Algorithm toe te passen, wat resulteert in een verhoogd verbruik van 1 CPU-core en 1 GB RAM tijdens de berekening.

## DBaaS

DBaaS, ofwel Databank as a Service, is een vorm van *Software as a Service* (SaaS) waarbij een externe partij de databank implementeert en onderhoudt via cloud *computing*. DBaaS wordt doorgaans geleverd met ingebouwd onderhoud en beveiliging, waardoor gebruikers zich niet langer bezig hoeven te houden met het bijwerken van de databank, het maken van back-ups of het zich zorgen maken over beveiligingslekken. Al deze taken worden afgehandeld door de externe partij.

DBaaS biedt voordelen voor alle gebruikers. Voor kleinere entiteiten bieden DBaaS-services de mogelijkheid om tegen betaalbare prijzen toegang te krijgen tot databanken, in plaats van te investeren in dure hardware die mogelijk niet volledig wordt benut. Voor grotere bedrijven biedt DBaaS het voordeel dat al het intensieve werk voor hen wordt uitgevoerd. Het onderhouden van een grote databank vereist speciale kennis en fysieke ruimte voor serverinfrastructuur, maar met DBaaS wordt dit alles uitbesteed aan de externe partij. [7]

“The market for DBaaS and [cloud databanken](https://www.ibm.com/topics/cloud-database) is among the fastest-growing [Software-as-a-Service (SaaS)](https://www.ibm.com/topics/iaas-paas-saas) markets, expected to grow to [USD 320 billion by 2025](https://www.businesswire.com/news/home/20190809005257/en/Global-Cloud-Database-DBaaS-Market-Report-2019)”. [7]

Aangezien de applicatie slechts sporadisch verkeer zal genereren en slechts enkele keren per week zal worden gebruikt, zou het inefficiënt zijn om de databank continu actief te laten zijn. Dit zou zowel een verspilling van resources als geld betekenen. Dit is waar *serverless* DBaaS voordeliger is.

### *Serverless* DBaaS

Ondanks wat de term *serverless* suggereert, worden er nog steeds servers gebruikt bij *serverless* *computing*. *Serverless* houdt echter in dat de ontwikkelaar zich niet hoeft bezig te houden met het opzetten van servers en het onderhouden van infrastructuur. Deze verantwoordelijkheden worden overgenomen door hostingbedrijven, en vallen onder de "onderhoudskosten", waarbij een medewerker van het hostingbedrijf deze taken voor de ontwikkelaar afhandelt. [8]

Wat *serverless* onderscheidt van eenvoudige cloudhosting is dat een *serverless* applicatie automatisch, zonder tussenkomst van het hostingbedrijf of de ontwikkelaar, schaalt in het gebruik van resources naarmate het verkeer naar de website toeneemt of afneemt. *Serverless* houdt dus ook in dat wanneer de applicatie niet wordt gebruikt, deze automatisch wordt geschaald naar 0 en er geen kosten in rekening worden gebracht. De mogelijkheid om te schalen naar 0 is waarom *serverless* zo interessant is voor de SkillSeeker-applicatie. [8]

Omdat *serverless* een geavanceerde infrastructuur vereist en de fysieke infrastructuur in deze tijd bijna volledig wordt gemonopoliseerd door de grote cloudaanbieders, zijn zij ook de enigen die *serverless* hosting aanbieden. Dit resulteert in zeer beperkte opties voor *serverless* hosting, aangezien de grote *cloud providers* de markt domineren. [8]

Hierdoor zijn de enige opties voor *serverless* databanken die extern worden beheerd, en dus niet zelf-onderhouden *serverless* containers, AWS Neptune en Azure Cosmos DB.

#### Containers

Containers zijn één van de belangrijkste uitvindingen wanneer het over de cloud gaat. Deze technologie ligt aan het centerpunt van op een gebruiksvriendelijke manier een applicatie rap op te zetten zonder lokaal te installeren. Zonder containers kan *serverless* niet bestaan.

Een container is volledig geïsoleerd van de gebruiker zijn bestuurssysteem en weet dus niets af van de gebruiker zijn data. Een container bevat alles dat het nodig heeft om de code die aan de container meegegeven is uit te voeren zelfs zijn eigen bestuurssysteem. [9]

Een containerimage is een softwarepakket dat alles bevat om de code uit te voeren van *libraries* tot systeem tools. Wanneer een image gebouwt wordt is dit een container die dus enkel het essentiële bevat om te werken. Aangezien een container geen kennis heeft van de gebruiker zijn apparaat en wat daarop staat betekent dit dat een container altijd hetzelfde zal reageren, waar deze ook gebruikt wordt. [9]

#### Kubernetes

Om *serverless* te realiseren wordt meestal gebruikgemaakt van het Kubernetes-containerorkestratieplatform. Alle containers binnen een Kubernetes instantie worden een cluster genoemd.

Kubernetes is een platform om containers te onderhouden. Met Kubernetes kan het voorkomen worden dat een van de containers in de cluster uitvalt en moet er niet handmatig een nieuwe gestart worden. Kubernetes biedt vele voordelen, zoals het automatisch herstarten of verwijderen van niet-reagerende services en het automatisch starten van nieuwe instanties. Met Kubernetes is er ook direct load *balancing,* zodat wanneer er plotseling veel verkeer naar een van de draaiende microservices gaat, er meerdere instanties worden gestart om een deel van deze belasting op te vangen. [10]

#### AWS Neptune

In de eerste plaats maakt AWS Neptune gebruik van Neptune Capacity Unit (NCU) om zijn schaalbaarheid te meten. 1 NCU komt overeen met 2 GB RAM en een proportionele toewijzing van CPU- en netwerkbandbreedte. Bij het gebruik van de *serverless* Neptune-service kan een gebruiker een minimale en maximale NCU instellen om de kosten van de service te beheersen. [11]

Ten tweede is Neptune niet strikt *serverless*. Op het moment van het schrijven van dit document staat Neptune niet toe dat een gebruiker zijn minimale NCU instelt op 0, wat betekent dat de instantie altijd actief is en er altijd een minimumprijs aangerekend wordt, wat het concept van *serverless* ondermijnt. Desondanks classificeert AWS het als *serverless*, dus wordt het in dit hoofdstuk besproken. [12]

De schatting voor de maandelijkse kosten van een *serverless* Neptune-instantie met minimale vereisten bedraagt $136. Deze hoge kosten zijn te wijten aan twee factoren: ten eerste wordt een Neptune-instantie nooit volledig uitgeschakeld wanneer deze niet wordt gebruikt, en ten tweede is er een aanzienlijke initiële kost verbonden aan het aanmaken van de instantie. Gezien deze kosten is een *serverless* Neptune-instantie enkel geschikt voor grotere projecten waar de service voortdurend wordt gebruikt en waar flexibiliteit om de resources te schalen naar behoefte vereist is.

Aangezien de geschatte prijs zo hoog uitvalt voor minimale vereisten, werd overwogen dat de calculator mogelijk een fout bevatte. Daarom is er een Neptune-instantie aangemaakt met de benodigde instellingen om te zien wat de werkelijke kosten zijn op het AWS-dashboard.

Na deze instantie één dag te hebben laten draaien, is de geschatte kost bijgewerkt en bedraagt deze $81.

Vervolgens is er, ondanks de hoge prijs, onderzoek gedaan naar de bruikbaarheid van Neptune voor de SkillSeeker-applicatie. Neptune kan worden gebruikt met verschillende querytalen, waaronder Gremlin en een variant van OpenCypher. Uit het onderzoek naar Cosmos DB is gebleken dat de Gremlin-taal veel minder goed gedocumenteerd is en niet zo gebruiksvriendelijk en duidelijk is als OpenCypher.

Een moeilijke taal mag geen reden zijn om een optie uit te sluiten, maar er moet ook rekening worden gehouden met het onderhoud van de applicatie. De slechte documentatie en moeilijkheidsgraad van Gremlin leiden tot de conclusie dat het onderhoud van de applicatie en mogelijke uitbreidingen langer duurt bij het gebruik van deze querytaal, en dus ook veel duurder is. Dit moet worden meegenomen in de kostenbesprekingen.

De documentatie voor OpenCypher en Neptune is uitermate beperkt, waardoor de voorkennis van het Neo4j onderzoek hier niet helpt. De documentatie om een verbinding te maken tussen Neptune en een Java-backend is nog beperkter, laat staan een Spring Boot backend. Hieruit is besloten om geen gebruik te maken van Neptune.

#### Cosmos DB for Apache Gremlin

Azure Cosmos DB is de door Microsoft zelf ontwikkelde *graph* databank. Microsoft besefte dat veel teams een combinatie van databanken gebruikten door de opkomst van AI, wat niet optimaal is. Er was behoefte aan een databank die aan alle eisen voldeed. Cosmos DB kan worden gebruikt voor relationele data, document data, vector data, key-value data, *graph* data en tabel data. Het is een *fully managed* service, wat betekent dat Azure de databank administratie uit handen van de developer neemt, zodat deze zich geen zorgen hoeft te maken over updates en onderhoud. [13]

Het team heeft vastgesteld dat Cosmos DB, een alles-in-één databank die zich over meerdere gebieden uitstrekt zonder zich op een specifiek terrein te specialiseren, een verwarrende gebruikersinterface biedt voor zijn *graph* databehoeften. De *graph* component van Cosmos DB wordt volledig genoemd als Azure Cosmos DB for Apache Gremlin. Gremlin is de querytaal ontwikkeld voor het Apache Tinkerpop *graph computing framework*. De Cosmos DB *graph* engine is ontworpen om sterk te lijken op Apache Tinkerpop, en ondersteunt daarom ook de Gremlin-taal. [14]

Na enkele experimenten is de conclusie gemaakt dat Gremlin niet de gewenste gebruiksvriendelijke ondersteuning biedt, waardoor het nodig was om verder te zoeken naar een andere databank-oplossing. Door een gebrek aan succes is het besluit gemaakt dat CosmosDB niet geschikt is voor SkillSeeker.

Een serverloze instantie van Cosmos DB voor Apache Gremlin kost naar schatting $0.50, aangezien deze serverloos draait. De lage kosten van de databank vormen een van de grootste voordelen ervan. Echter, zoals eerder vermeld, is de gebruiksvriendelijkheid van de databank niet optimaal, wat kan resulteren in hogere kosten voor het onderhoud of mogelijke uitbreiding van code. [15]

### AWS Neptune Cluster instantie

Naast de *serverless* optie biedt Neptune ook de mogelijkheid om een eenvoudige Neptune-instantie aan te maken die niet automatisch schaalt zoals bij *serverless*, maar in plaats daarvan vaste resources heeft. In dit geval moet de gebruiker zelf de schaling beheren wanneer dat nodig is. Wat ook een kost is als werkgever.

Neptune biedt niet de mogelijkheid om te kiezen voor minder dan 2 vCPU's, wat een nadeel is voor een kleine applicatie. Wanneer een instantie aangemaakt wordt met minimale resources die alleen tijdens de werkuren actief is, komt de prijsschatting uit op $19.47. Deze prijs ligt een pak hoger dan andere schattingen en wordt dus niet verder besproken.

### AuraDB

AuraDB is de cloudoplossing ontwikkeld door de makers van Neo4j, wat inhoudt dat het onderhoud van de databank, zoals back-ups en updates, volledig wordt verzorgd door het AuraDB-team. Gebruikers hoeven zich hier dus geen zorgen over te maken. Echter, bij een volledig beheerde cloudoplossing is de prijs voornamelijk afhankelijk van het externe beheer, en minder van de onderliggende infrastructuur zelf. [16]

AuraDB biedt een gratis optie aan, wat een aantrekkelijk aanbod lijkt voor SkillSeeker. Het onderzoek naar AuraDB is uitgevoerd zonder de kennis over de APOC-plugin. Zelfs zonder deze plugin is er ondervonden dat er aanzienlijke prestatieproblemen met AuraDB waren.

Hoewel AuraDB ook betaalde versies aanbiedt, zijn deze al snel onbetaalbaar. De startprijs van minstens $130 per maand voor SkillSeeker is simpelweg niet haalbaar. [17]

## Cloud Containers

### Azure

#### Azure Container Apps

Azure Container Apps (ACA), is een volledig onderhouden platform gebaseerd op Kubernetes. Normaal gesproken moet een gebruiker, om ervoor te zorgen dat zijn applicatie geen resources gebruikt wanneer er geen verkeer is en automatisch meer instanties aanmaakt of meer resources benut bij hoog verkeer, gebruikmaken van Kubernetes. Echter, Kubernetes vereist een zeer steile leercurve en op platforms zoals Azure gaat het inzetten van een Kubernetes-instantie gepaard met hoge startkosten. ACA maakt gebruik van Kubernetes op de achtergrond, waardoor een gebruiker geen uitgebreide kennis nodig heeft van de werking van de Kubernetes API, maar nog steeds kan profiteren van de voordelen van Kubernetes, zoals het automatisch schalen van zijn applicatie. [18]

Azure biedt gratis Container Apps aan voor de eerste 2 miljoen *requests* per maand. Aangezien SkillSeeker niet meer dan twee miljoen aanvragen per maand heeft, kan Azure Container Apps gratis gebruikt worden om een container te hosten. Dat betekent dat er een zelfgemaakte databank-image opgezet kan worden of zelfs een Spring Boot Java-backend zonder extra kosten. Het kost ongeveer hetzelfde als Azure Cosmos DB, maar met de flexibiliteit om zelf een databank te kiezen en aan te passen. Het enige nadeel is dat er wat meer technische kennis nodig is om dit op te zetten. [15]

#### Azure Container Instances

Azure Container Instances (ACI), zoals de naam al aangeeft, biedt gebruikers de mogelijkheid om eenvoudig containers in de cloud te hosten. Azure Container Instance is 1 alleenstaande container die vanzelf naar 0 schaalt wanneer deze niet gebruikt wordt. [19]

Het grote verschil tussen ACI en ACA is dat ACA achterliggend met Kubernetes werkt en dus complexere opstellingen voor bijvoorbeeld een microservice architectuur toelaat door middel van *service discovery* en andere tools.

Een containerinstantie die 8 uur per dag draait met 1 vCPU en 2 GB RAM kost ongeveer $10 per maand. Deze container blijft gedurende die 8 uur actief, wat betekent dat er geen opstarttijd is zoals bij Container Apps, maar het wel geld kost wanneer hij niet wordt gebruikt.

#### Azure Functions

Azure Functions is een serverloze compute-omgeving ontwikkeld door Microsoft Azure, die het mogelijk maakt om applicaties te hosten in de cloud zonder noodzaak voor beheer van servers of infrastructuur. Het platform biedt ondersteuning voor diverse programmeertalen, waaronder Java, Python, JavaScript en C#, waardoor ontwikkelaars hun code kunnen implementeren zonder zich zorgen te maken over de onderliggende infrastructuur.

Een belangrijk voordeel van Azure Functions ten opzichte van traditionele containeroplossingen is de vereenvoudigde implementatie. In plaats van een containerimage te creëren en te configureren, hoeven ontwikkelaars enkel hun code te uploaden en de gewenste programmeertaal te selecteren. Azure Functions kan dan vervolgens deze containerimage veel sneller ophalen waardoor er geen langdurige periode gespendeerd wordt aan het laden van de containerimage bij het opstarten van de Function.

Echter, ondanks de voordelen van Azure Functions, is het niet geschikt voor alle applicaties. De functionaliteit is beperkt tot het uitvoeren van event-gedreven code en zorgt ervoor dat het niet geschikt is voor de SkillSeeker Neo4j databank.

### AWS

Net zoals Azure de optie biedt om een containerinstantie rechtstreeks te draaien of om de instantie *serverless* te maken, waardoor de gebruiker minder controle maar ook minder zorgen heeft en kan genieten van een *serverless* omgeving waarbij hij alleen betaalt voor wat hij gebruikt, biedt AWS dezelfde mogelijkheden.

#### AWS Elastic Container Service

Terwijl Azure Kubernetes gebruikt als zijn containerorkestratieplatform, heeft AWS zijn eigen platform ontwikkeld: Elastic Container Service (ECS). Dit platform is door Amazon ontwikkeld en wordt regelmatig bijgewerkt, wat betekent dat ontwikkelaars op Amazon kunnen vertrouwen dat hun omgeving beveiligd is met de nieuwste technologieën. Net als Kubernetes is de taak van ECS het onderhouden van containers, waarbij het aantal instanties van een container wordt verminderd of de toewijzing van resources wordt verminderd wanneer er minder verkeer is, en vice versa tijdens periodes van zwaar verkeer. [20]

AWS biedt ook de optie van EKS, Elastic Kubernetes Service, wat in principe hetzelfde is als ECS maar voor gebruikers die liever Kubernetes gebruiken in plaats van AWS's eigen orkestratieplatform.

#### AWS Fargate

AWS Fargate gebruikt AWS ECS als containerorchestratieplatform om het schalen van container simpeler te maken voor de gebruiker.

AWS Fargate is vergelijkbaar met Azure Container Apps, waarbij de gebruiker zich geen zorgen hoeft te maken over de infrastructuur en schaling. Het verschil met ACA is dat er bij ACA werkelijk geen kennis nodig is van Kubernetes terwijl er bij Fargate wel lichte kennis nodig is over ECS om deze service juist in te stelling. [21]

Waar Fargate overeenkomt met ACA is dat er enkel betaald wordt in de hoeveelheid gebruikte resources en dus werkelijk serverloos is omdat het ook naar 0 kan schalen.

Uit metingen blijkt dat een Neo4j container met de APOC-afhankelijkheid 1 virtuele CPU's aan rekenkracht vereist. Volgens de AWS-prijscalculator moet een instantie van 1 virtuele CPU’s minstens 2 GB aan RAM toegewezen hebben. [22]

Op basis van deze instellingen in te geven in de AWS-prijscalculator kan geconcludeerd worden dat AWS Fargate, net als Azure Container Apps, gratis is voor onze databank. Dit komt doordat er een bepaalde hoeveelheid gratis *requests* per maand zijn en SkillSeeker niet over dit limiet gaat.

#### AWS App Runner

AWS App Runner is een service die de voordelen van Fargate en Lightsail combineert. Het is vergelijkbaar met Fargate doordat het containers beheert voor de gebruiker, maar net als Lightsail vereenvoudigt App Runner het hosten van containers aanzienlijk door de complexe details weg te nemen die nodig zijn voor Fargate. [23]

App Runner biedt een meer gestroomlijnde aanpak voor het hosten van containers, vergelijkbaar met hoe Elastic Beanstalk EC2 vereenvoudigt. Hierdoor kunnen gebruikers snel en eenvoudig containers implementeren zonder zich te hoeven verdiepen in de complexiteit van de onderliggende infrastructuur. [23]

Het grote nadeel van App Runner tegenover bijvoorbeeld Fargate is dat App Runner momenteel nog niet de optie biedt om volledig naar 0 te schalen.

De geschatte maandelijkse kosten voor een instantie van AWS App Runner, die voldoet aan de noden van SkillSeeker, bedraagt $29.70. Zoals bij de meeste AWS-services het geval is, ligt deze prijs te hoog voor de applicatie. Azure biedt daarentegen goedkopere opties. [22]

## Virtual machine hosting

Toen er eerst begonnen werd met het ontwikkelen van servers, was het proces heel eenvoudig: een fysieke server behandelde één proces. Dit kwam doordat de besturingssystemen (OS) van die tijd niet de mogelijkheden hadden om meerdere applicaties binnen één OS op een veilige manier uit te voeren. Het gevolg hiervan was dat voor elke applicatie die een bedrijf ontwikkelde, een nieuwe fysieke server moest worden aangeschaft. Aangezien een applicatie vaak niet de volledige capaciteit van een server benutte, resulteerde dit in aanzienlijke verspilling van middelen. [24]

Door gebruik te maken van virtuele machines wordt een hypervisor geïnstalleerd op de hardware. Deze hypervisor controleert welke virtuele machines toegang hebben tot welke resources. Boven op de hypervisor worden virtuele machines geïnstalleerd, elk met zijn eigen OS. [24]

Het grootste nadeel van virtuele machines is dat elk zijn eigen OS heeft, wat veel opslagruimte in beslag neemt op de harde schijf. Bovendien verbruikt een OS veel RAM en CPU-kracht om te kunnen draaien. [24]

Het belangrijkste verschil tussen virtuele machines en containers ligt in het type isolatie dat ze bieden. Virtuele machines bieden systeemisolatie, wat betekent dat ze een gecompleteerd besturingssysteem simuleren en strikt gescheiden zijn van andere VM's op dezelfde host. Containers daarentegen bieden processenisolatie, waarbij applicaties worden gegroepeerd in afzonderlijke omgevingen met hun eigen resources en *dependencies*. [24]

Dit verschil in isolatietype heeft verschillende implicaties. Containers starten doorgaans sneller op, verbruiken minder systeembronnen en hebben een kleinere bestandsgrootte dan virtuele machines. Dit maakt ze een goede keuze voor microservices-architecturen en *workloads* die dynamisch kunnen worden geschaald. Virtuele machines daarentegen bieden een hogere mate van isolatie en beveiliging, waardoor ze geschikter zijn voor kritische *workloads* en legacy-applicaties.

De conclusie hier is dat een virtuele machine niet geschikt is voor de SkillSeeker-applicatie, gezien deze applicatie meer belang hecht aan snelle opstarttijden, schaalbaarheid en kostenefficiëntie. Desondanks worden de opties nog steeds in beperkte mate onderzocht.

### Azure Virtual Machines

Op basis van de instellingen benodigd voor de SkillSeeker databank wordt de geschatte kostenindicatie van het draaien van deze virtuele machines ongeveer $4.70. Hoewel dit nog steeds als een lage kost wordt beschouwd, blijft een container nog steeds goedkoper. Bovendien maakt het gebruik van moderne tools het eenvoudiger om containers te onderhouden in vergelijking met virtuele machines. Daarnaast bieden containers de mogelijkheid om bij hoge verkeersvolumes zeer snel te schalen, iets wat virtuele machines niet kunnen.

### AWS EC2

AWS EC2 is de equivalent van Azure Virtual Machines waar dat er betaald wordt om virtuele machines in de cloud te hosten. [25]

Een AWS EC2-instantie die 5 dagen per week 8 uur per dag draait met 1 vCPU en 2 GB RAM kost ongeveer $10 per maand. Dit tarief ligt dus al hoger dan dat van virtuele machines van Azure. Samen met de steilere leercurve van AWS wordt het gebruik van AWS niet aangeraden, tenzij er al een bestaande infrastructuur op AWS is en daarom de voorkeur heeft vanwege deze bestaande setup. [25]

### AWS Lightsail

De makkelijkste manier om AWS Lightsail te beschrijven is: simpel. Lightsail is een van de nieuwere AWS-producten die als doel heeft om AWS gebruiksvriendelijker te maken. Bij het aanmaken van een EC2-instantie zijn er vaak overweldigend veel opties en instellingen, wat de leercurve voor nieuwe gebruikers afschrikkend kan maken. Lightsail stelt de gebruiker in staat om enkel de belangrijkste configuratie-opties te kiezen, terwijl alle technische details achter de schermen worden afgehandeld. Kortom Lightsail is een gebruiksvriendelijke manier om dezelfde service als EC2 te krijgen zonder dat de gebruiker alles tot in detail moet begrijpen. [26]

Het is echter belangrijk om op te merken dat het hosten van de SkillSeeker-databank in AWS Lightsail gepaard gaat met hogere kosten vanwege de gebruiksvriendelijke installatie en onderhoudsafhandeling.

Voor een container met 1 vCPU en 2 GB RAM wordt er $39.25 per maand betaald. Dit tarief ligt dus aanzienlijk hoger dan andere schattingen voor het hosten van de SkillSeeker databank, waardoor het een ongeschikte optie lijkt voor dit specifieke gebruik.

Hoewel Lightsail voornamelijk gericht is op het hosten van virtuele servers en niet primair op het ondersteunen van containers - hiervoor zijn andere services zoals Fargate en ECS beschikbaar - biedt het nog steeds een aantrekkelijke optie voor het hosten van een Java-backend. Met een maandtarief van slechts $4 is de prijs aanvaardbaar. Ondanks dit feit blijft het hosten van de SkillSeeker-backend op een Azure Container App nog steeds goedkoper.

## Virtuele machines en containers

Zowel Azure App Services als AWS Elastic Beanstalk kunnen gebruikmaken van virtuele machines of containers, afhankelijk van de configuratie. Beide services hebben als doel het vereenvoudigen van het implementatieproces door de gebruiker minder directe controle te geven, wat resulteert in meer gebruiksvriendelijkheid.

### Azure App Services

Azure App Services is een handige service voor het hosten van verschillende soorten applicaties, en de gratis tier maakt het aantrekkelijk voor hobbyprojecten en kleinere toepassingen. Het is jammer dat het niet geschikt is voor het hosten van een Neo4j-databank vanwege de vereiste TCP-communicatie. Voor dergelijke toepassingen zou het nodig zijn om naar andere Azure-services te kijken die wel TCP-communicatie ondersteunen en geschikt zijn voor het hosten van databanken, zoals Azure Virtual Machines, Azure Kubernetes Service, of Azure Databank Services, afhankelijk van de behoeften van het project. [27]

### AWS Elastic Beanstalk

AWS Elastic Beanstalk is een platform bovenop EC2-containers, welke containers inzet om applicaties schaalbaar en onderhoudbaar te maken, waarbij gebruikers minimale betrokkenheid vereisen. Elastic Beanstalk biedt ondersteuning voor diverse omgevingen, maar is niet ontworpen voor het gebruik als databank, aangezien AWS alternatieve services zoals Neptune aanbiedt voor dergelijke doeleinden. Dit maakt het een minder geschikte optie voor de SkillSeeker-databank. Elastic Beanstalk biedt de meest eenvoudige methode om containers te hosten op AWS. Het platform maakt gebruik van de EC2 API op de achtergrond, waarbij alle instellingen automatisch worden beheerd voor de gebruiker. De gebruiker hoeft slechts te kiezen voor welke omgeving, bijvoorbeeld Java, .Net, Node, enzovoort, hij zijn code wil implementeren, waarna de rest op de achtergrond wordt afgehandeld. [28]

Elastic Beanstalk kan worden gebruikt om containers te hosten. In dit geval maakt het platform gebruik van ECS op de achtergrond, wat het proces vereenvoudigt voor de gebruiker. Echter, Elastic Beanstalk kan ook worden gebruikt om virtuele machines te hosten, waarbij het platform EC2-instanties gebruikt en deze automatisch configureert.

## Praktijk

Door AWS te onderzoeken is de conclusie getrokken dat AWS zich meer op grote ondernemingen richt en ver verwijderd is van gebruiksvriendelijkheid. De steile leercurve zorgt voor de beslissing om AWS alleen theoretisch te bespreken in dit rapport. Het project biedt niet voldoende tijd om AWS volledig te beheersen en meerdere infrastructuren op te zetten.

Mediaan Conclusion heeft nooit overwogen om AWS te gebruiken voor SkillSeeker. Daarom is het efficiënter om te focussen op onderzoek naar Azure. Dit komt enerzijds door een gebrek aan tijd, en anderzijds door het ontbreken van de benodigde middelen voor een uitgebreide analyse van AWS.

Aangezien Azure en AWS beide wereldleiders zijn op het gebied van cloudinfrastructuur, is het veilig om aan te nemen dat de kwaliteit van hun services niet veel verschilt. Het belangrijkste verschil, dat ook uit de proef bleek, is dat Azure veel gebruiksvriendelijker is voor beginners, terwijl AWS meer geschikt is voor experts vanwege de overweldigende hoeveelheid opties.

Een tweede conclusie uit de proef is dat beginners op AWS uiterst voorzichtig moeten zijn, omdat het niet altijd duidelijk is of een service per ongeluk niet is verwijderd. Dit kan maanden later resulteren in een zware rekening.

### AWS

Als Mediaan besluit om SkillSeeker volledig in AWS te hosten, zijn er verschillende AWS-services die kunnen worden gebruikt om de optimale infrastructuur op te zetten.

Voor de *graph* databank is Neptune een uitstekende keuze vanwege zijn ondersteuning voor grafiekgegevens en de integratie met AWS-services.

Voor de backend is Elastic Beanstalk een optie, aangezien het de implementatie en het beheer van applicaties vereenvoudigt. Het biedt automatische schaalbaarheid, load balancing en andere handige functies.

Voor de Angular frontend is Lightsail perfect, vooral als er behoefte is aan een eenvoudige, gebruiksvriendelijke manier om de frontend te hosten. Lightsail biedt vooraf geconfigureerde virtuele servers met eenvoudige beheeropties.

Als er ook een Neo4j databank nodig is, kan deze worden opgezet met behulp van AWS App Runner. App Runner biedt een eenvoudige manier om containerized applicaties te implementeren en te schalen, wat ideaal is voor het uitvoeren van een Neo4j databank in AWS.

Samengevat, met Neptune voor de grafische databank, Elastic Beanstalk voor de backend, Lightsail voor de frontend en mogelijks AWS App Runner voor de Neo4j databank, kan SkillSeeker effectief worden gehost in AWS met een optimale infrastructuur.

### Azure Container Apps voor backend en databank

ACA en ACI zijn allebei even geschikt voor SkillSeeker naast dat ACA in theorie iets complexer is dan ACI omdat ACI slechts alleenstaande containers zijn. Toch na experimenteren met ACI zijn er problemen ondervonden met het werkend krijgen van *persistent data* door een Azure File Share te *mounten* waardoor er overgeschakeld is naar ACA.

Samenvatting van het stageverloop: De frontend draait op de gratis tier van een app-service, terwijl de backend wordt gehost in een containerapplicatie met 2 vCPU's en 4 GB RAM. De databank wordt ook in een container uitgevoerd, met 1 vCPU's en 2 GB RAM.

Een van de grootste voordelen van serverless containers is dat ze niet constant actief zijn, waardoor alleen wordt betaald voor daadwerkelijk gebruik. Dit voordeel wordt echter ook een nadeel, omdat bij inactiviteit van 5 minuten de applicatie opnieuw moet opstarten.

Om de prestaties te meten, wordt een stopwatch gebruikt in combinatie met de frontend in de cloud om te bepalen hoelang het gemiddeld duurt voordat de gegevens worden weergegeven na een periode van 15 minuten inactiviteit.

Tabel 1 Metingen van de reactietijd van SkillSeeker in de cloud

|  |  |
| --- | --- |
|  | Tijd in seconden |
| Eerste meting | 26.28 |
| Tweede meting | 26 |
| Derde meting | 18 |
| Vierde meting | 35.03 |
| Vijfde meting | 23.05 |
| Gemiddelde tijd | 25.672 |

Uit deze metingen, te zien in tabel 1, is geconcludeerd dat er fluctuaties zijn in de opstarttijden, maar dat deze gemiddeld rond de 26 seconden liggen. Dit wordt als een acceptabele tijd beschouwd, gezien het voordeel dat er geen resources worden gebruikt wanneer de applicatie niet actief is.

De opstarttijd van de backendcontainer wordt gemeten door middel van containerlogs. De totale opstarttijd en de tussenliggende stappen worden geanalyseerd. Dit wordt uitgevoerd door een verzoek naar de applicatie te sturen, zodat de container opstart, en de systeemlogs te inspecteren op relevante details. Vervolgens wordt gewacht tot de applicatie automatisch terugkeert naar schaal 0. Deze procedure wordt vijfmaal herhaald.

Deze gedetailleerde analyse biedt waardevolle inzichten in de opstartprestaties van de backendcontainer. De identificatie van de "ImagePullFailure"-waarschuwing als knelpunt kan leiden tot gerichte optimalisaties om de opstarttijd te verkorten.

Tabel 2 Genoteerde waarden vanuit de container logs van de backend container

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Tijdstempel begin opstarten | Tijdstempel “ImagePullFailure” | Tijdstempel image tweede keer ophalen | Tijdstempel image opgehaald | Tijdstempel container opgestart | Totale opstarttijd in seconden |
| Eerste meting | 12:09:52 | 12:10:10 | 12:10:10 | 12:10:13 | 12:10:13 | 21 |
| Tweede meting | 12:18:51 | 12:19:08 | 12:19:08 | 12:19:11 | 12:19:11 | 20 |
| Derde meting | 12:27:59 | / | / | 10:28:12 | 10:28:13 | 14 |
| Vierde meting | 03:15:08 | 03:15:25 | 03:15:25 | 03:15:29 | 03:15:29 | 21 |
| Vijfde meting | 03:34:51 | / | / | 03:35:00 | 03:35:01 | 10 |

Tabel 2 presenteert een significant gevarieerd beeld van de gemeten opstarttijden. Merkwaardig is de substantiële tijdsduur die wordt besteed aan het ophalen van de container image, waarbij dit proces frequent resulteert in falen, wat de totale opstarttijd verdubbelt. Om de efficiëntie van het containeropstartproces te verbeteren, is een diepgaande analyse noodzakelijk om de oorzaken van de vertraagde en falende image retrieval te achterhalen.

Gezien het beperkte belang is er geen grondige meting uitgevoerd van de benodigde tijd om de applicatie terug te brengen naar nul actieve instanties. Het blijkt uit de gemiddelde gegevens dat de achterliggende container na een periode van inactiviteit van vijf minuten naar nul schaaleenheden wordt teruggebracht.

Verder wordt aandacht besteed aan de opstarttijd van de Spring Boot applicatie binnen de container. Deze parameter kan worden afgelezen via de console van de container. Het opstartproces wordt vijf keer herhaald, waarbij de betreffende waarden consequent vijfmaal worden vastgelegd.

Tabel 3 Meting starttijd van Spring Boot applicatie binnen de container

|  |  |
| --- | --- |
|  | Tijd in seconden |
| Eerste meting | 4.108 |
| Tweede meting | 4.014 |
| Derde meting | 3.96 |
| Vierde meting | 4.095 |
| Vijfde meting | 4.084 |
| Gemiddelde tijd | 4.0522 |

De gegevens in Tabel 3 tonen aan dat de gemiddelde opstarttijd van de Spring Boot-applicatie in de container slechts 4 seconden bedraagt. De keuze voor Spring Boot is gebaseerd op expertise met dit framework. Indien efficiënte schaalbaarheid een hogere prioriteit had, zou een framework als Quarkus, dat specifiek ontworpen is voor gebruik in Kubernetes-clusters, wellicht interessanter zijn geweest. Een grondige verkenning van deze alternatieve optie valt echter buiten de grenzen van dit onderzoek.

Ten slotte wordt een meting uitgevoerd van de opstarttijd van de container die de databank bevat. Deze meting wordt verricht door de logs te observeren. Er wordt een verzoek naar de backend gestuurd, aangezien de backendcontainer onmiddellijk de databankcontainer activeert. Vervolgens worden de logs geïnspecteerd voor een meer gedetailleerd overzicht. De eerste kolom bevat de begintijd vanaf het moment dat de container het commando ontving om van 0 naar 1 te schalen en dus op te starten, in het formaat uur:minuten:seconden. De tweede kolom geeft de tijdstempel weer waarop de container image wordt opgehaald vanuit het register. De derde kolom geeft aan wanneer de container image is opgehaald, en de vierde kolom bevat het tijdstip waarop de container is opgestart. In de laatste kolom is de totale opstarttijd genoteerd.

Tabel 4 Logs van de container die de databank bevat

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Tijdstempel begin opstarten | Tijdstempel image ophalen | Tijdstempel image opgehaald | Tijdstempel container opgestart | Totale opstarttijd in seconden |
| Eerste meting | 11:31:9 | 11:31:11 | 11:31:24 | 11:31:25 | 16 |
| Tweede meting | 12:1:02 | 12:01:03 | 12:01:14 | 12:01:15 | 13 |
| Derde meting | 03:15:34 | 03:15:52 | 03:16:01 | 03:16:01 | 27 |
| Vierde meting | 03:28:22 | 03:28:23 | 03:28:40 | 03:28:41 | 19 |
| Vijfde meting | 04:07:30 | 04:07:32 | 04:07:43 | 04:07:44 | 14 |

Net als bij de logs van de backendcontainer is het hier duidelijk dat het grootste struikelblok het ophalen van de containerafbeelding is, waarbij het ook regelmatig kan falen, zoals in de derde meting is voorgekomen. Een opvallend verschil is echter dat het falen bij het ophalen van deze containerafbeelding minder frequent voorkomt dan bij die van de backendcontainer. Om de opstarttijd te verbeteren, dient hier ook gezocht te worden naar alternatieven om het proces van het ophalen van de containerafbeelding te optimaliseren.

Vanwege een gebrek aan urgentie is hier geen uitgebreide meting van verricht, maar het kost gemiddeld 10 minuten voor de databankcontainer om van 1 naar 0 te schalen wanneer er geen activiteit is. Dit is twee keer zo lang als de container van de backend. Er zijn geen specifieke redenen voor dit verschil gevonden.

### App Services voor frontend en backend + directe integratie met databank

Een interessante optie, hoewel het implementeren ervan tijd en moeite vereist omdat het herontwerp van de applicatie nodig is, is het gebruik van Azure Web Apps met directe integratie van Azure databank services zoals Cosmos DB. Als er bijvoorbeeld Cosmos DB gebruikt wordt voor SkillSeeker, kan de Angular frontend gehost worden in een Node.js-omgeving op een Azure App Service. Vervolgens kan de Java backend ook als een App Service gedraaid worden, met directe integratie met Cosmos DB. Op deze manier kan de volledige SkillSeeker-applicatie gehost worden op twee App Services op de gratis tier, terwijl Cosmos DB volledig wordt afgeschermd van de buitenwereld voor maximale beveiliging.

Conclusie

Na uitgebreid onderzoek zijn verschillende conclusies vastgelegd. Ten eerste zijn er aanzienlijke verschillen tussen Azure en AWS. Het grootste verschil ligt in de focus van elk platform: AWS richt zich duidelijk meer op grote bedrijven en experts, terwijl Azure gemakkelijk toegankelijk is voor individuen met beperkte ervaring. Deze conclusie wordt ondersteund door zowel ervaringen als het feit dat Azure veel gratis opties biedt voor zijn services, terwijl de goedkoopste services op AWS vaak ook de meest complexe zijn. Bovendien blijkt uit de vergelijking van meer gebruiksvriendelijke services op AWS, zoals App Runner of Lightsail, dat extra kosten worden gerekend voor deze gebruiksvriendelijkheid, terwijl Azure gebruiksvriendelijke services biedt die goedkoper zijn dan de alternatieven op AWS.

Ten tweede is geconcludeerd dat de AWS-calculator niet altijd even nauwkeurig is, zoals bleek uit een incident met een Neptune-rekening. Hierbij werd na een experiment een voorspelde maandelijkse kost van 80 euro uiteindelijk 300 euro. Dit benadrukt het belang van het kritisch evalueren van kostenprognoses bij het plannen van AWS-resources.

Daarna blijkt dat Neo4j de meest gebruiksvriendelijke databank is door gebruik te maken van Cypher syntax. Uit het experiment naar verschillende databanken heeft het team geen één andere databank zo rap werkend gekregen als Neo4j. Dit duidt op de gebruiksvriendelijkheid en qualitatieve documentatie van Neo4j.

Vervolgens blijkt vanuit metingen dat een Neo4j-container met slechts 400 *nodes* en 1500 relaties kan opereren met 1 vCPU en 2 GB RAM zonder problemen. Het is dus goedkoop om deze databank als een container te gebruiken in de cloud.

Ten slotte is besloten vanuit het onderzoek om de Neo4j-databank te hosten op Azure Container Apps. Deze kunnen serverloos worden gebruikt en worden alleen betaald voor daadwerkelijk gebruik. Gebaseerd op deze conclusie wordt de SkillSeeker databank ook op Azure Container Apps gedraaid. Het nadeel hiervan is echter een gemiddelde opstarttijd van 26 seconden nadat de applicatie 5 minuten niet is gebruikt. Het voordeel van Azure Container Apps is echter dat Azure gratis *requests* biedt elke maand, waardoor deze service gratis kan worden gebruikt voor kleine en zelden gebruikte applicaties zoals SkillSeeker.

Hypothese: geen backend?

Momenteel fungeert de backend als tussenstation voor het ontvangen van verzoeken, het raadplegen van de databank en het terugsturen van gegevens. Er vindt slechts één complexe berekening plaats. Het is dus zeker het overwegen waard om de mogelijkheid te onderzoeken om de backend volledig te elimineren.

In theorie is het mogelijk om van de frontend rechtstreeks query’s naar de databank te sturen zonder gebruik te maken van een backend. Natuurlijk zijn hier een hele hoop beveiliging risico’s aan verbonden maar als de applicatie op Azure runt kan er rechtstreeks gebruik gemaakt worden van Entra ID als identity provider waardoor enkel mensen met een Mediaan account toegang hebben tot de applicatie. Er is binnen het project geen tijd om al het werk dat in de backend gestoken is weg te smijten en om de logica van de frontend volledig te veranderen. Desondanks is dit nog steeds een interessante theorie. Om dit te verwezenlijken moet er wel een manier zijn om van de frontend rechtstreeks TCP-*requests* te sturen naar de Neo4j databank waar momenteel nog geen optie voor gevonden is.

Als Cosmos DB wordt gebruikt voor SkillSeeker, is het mogelijk om de Angular frontend in een Node.js-omgeving op een Azure App Service te draaien, met directe integratie met Cosmos DB. Dit is zelfs een zeer kosteneffectieve oplossing die geen uitgebreide opstarttijd vergt, in tegenstelling tot het gebruik van containers.

Reflectie

Het onderzoek naar diverse cloudoplossingen voor de SkillSeeker-database was een cruciaal leerpunt in mijn schoolcarrière. Tijdens dit onderzoek heb ik mijn kennis van de afgelopen drie jaar kunnen toepassen om kwalitatief hoogwaardige resultaten te leveren.

Door dit project heb ik geleerd hoe belangrijk het is om bronnen kritisch te evalueren, vooral wanneer er sprake is van mogelijke bias. Ik heb ook mijn vaardigheden in het analyseren van loggegevens en het testen van infrastructuur op het Azure-platform verbeterd. Hoewel ik meer voorkennis had willen hebben van AWS, heeft het leerproces me geholpen om beter om te gaan met onbekende technologieën.

Ik heb tijdens het onderzoek geleerd dat het belangrijk is om meerdere bronnen te raadplegen omdat binnen een complex onderwerp zoals cloud er vaak veel informatie is. Zo heb ik vaak aanpassingen moeten doen omdat de ene pagina binnen bijvoorbeeld AWS info ontbrak dat op een andere pagina wel stond waardoor ik een foute conclusie maakte.

Het onderzoek was soms uitdagend, aangezien niet elke bron betrouwbaar bleek te zijn. Aanvankelijk dacht ik bijvoorbeeld dat een Azure Container Instance niet automatisch schaalde, wat onjuist bleek te zijn. Ook kostte het me enige tijd om het verschil tussen AWS App Runner en AWS Fargate te begrijpen, vanwege de overvloed aan beschikbare bronnen.

Dankzij een grondig theoretisch onderzoek verliep het praktische deel echter zeer soepel. Helaas hebben we door een gebrek aan tijd en middelen alternatieven alleen theoretisch kunnen bespreken.

# Bronnenlijst

|  |  |  |
| --- | --- | --- |
| [1] |  | M. Conclusion, „Mediaan Conclusion,” Mediaan Conclusion, 1 3 2024. [Online]. Available: https://mediaan.com/nl. [Geopend 1 3 2024]. |
| [2] |  | M. Conclusion, „Mediaan Conclusion,” Mediaan Conclusion, 1 3 2024. [Online]. Available: https://mediaan.com/nl/over-ons. [Geopend 1 3 2024]. |
| [3] |  | S. Jha, „AWS vs Azure: Which Cloud Platform Should You Choose in 2024?,” simplilearn, 15 2 2024. [Online]. Available: https://www.simplilearn.com/tutorials/cloud-computing-tutorial/aws-vs-azure. [Geopend 1 4 2024]. |
| [4] |  | A. Disney, „How to choose a graph database: we compare 6 favorites,” Cambridge Intelligence, 19 10 2023. [Online]. Available: https://cambridge-intelligence.com/choosing-graph-database/. [Geopend 22 3 2024]. |
| [5] |  | J. Baljeu, „Containerization on Azure: a comparison of services,” devoteam M Cloud, x x x. [Online]. Available: https://mcloud.devoteam.com/expert-view/containerization-on-azure-a-comparison-of-services/. [Geopend 22 3 2024]. |
| [6] |  | I. Curkovic, „How to Monitor Container Memory and CPU Usage in Docker Desktop,” Docker, 2 12 2022. [Online]. Available: https://www.docker.com/blog/how-to-monitor-container-memory-and-cpu-usage-in-docker-desktop/. [Geopend 14 3 2024]. |
| [7] |  | IBM, „What is database as a service (DBaaS)?,” IBM, x x x. [Online]. Available: https://www.ibm.com/topics/dbaas. [Geopend 7 3 2024]. |
| [8] |  | Red Hat, „What is serverless,” Red Hat, 11 5 2022. [Online]. Available: https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless. [Geopend 8 3 2024]. |
| [9] |  | Docker, „Use containers to Build, Share and Run your applications,” Docker, x x 2024. [Online]. Available: https://www.docker.com/resources/what-container/. [Geopend 21 3 2024]. |
| [10] |  | kubernetes, „kubernetes documentation overview,” kubernetes, 19 9 2023. [Online]. Available: https://kubernetes.io/docs/concepts/overview/. [Geopend 8 3 2024]. |
| [11] |  | B. Bebee, „Gettings started with neptune serverless,” AWS, x x 2023. [Online]. Available: https://pages.awscloud.com/rs/112-TZM-766/images/2023\_0206-SN-DAT\_Slide-Deck.pdf. [Geopend 7 3 2024]. |
| [12] |  | G. Swaminathan, „Neptune Serverless - "is it really Serverless?",” DEV, 30 10 2022. [Online]. Available: https://dev.to/aws-builders/neptune-serverless-is-it-really-serverless-24mo. [Geopend 14 3 2024]. |
| [13] |  | Microsoft, „Azure Cosmos DB – Unified AI Database,” Microsoft, 17 3 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/cosmos-db/introduction. [Geopend 22 3 2024]. |
| [14] |  | Microsoft, „What is Azure Cosmos DB for Apache Gremlin?,” Microsoft, 4 4 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/cosmos-db/gremlin/introduction. [Geopend 3 4 2024]. |
| [15] |  | Microsoft Azure, „Pricing calculator,” Microsoft, x x x. [Online]. Available: https://azure.microsoft.com/en-us/pricing/calculator/?ef\_id=\_k\_CjwKCAjwl4yyBhAgEiwADSEjeOwyZwxR6HRlatdV0RhQJgs2HiFHnZRNjlgwNTylvwYi9B3iF94rVhoCDawQAvD\_BwE\_k\_&OCID=AIDcmmbnk3rt9z\_SEM\_\_k\_CjwKCAjwl4yyBhAgEiwADSEjeOwyZwxR6HRlatdV0RhQJgs2HiFHnZRNjlgwNTylvwYi9B. [Geopend 14 5 2024]. |
| [16] |  | Neo4j, „Neo4j AuraDB overview,” Neo4j, x x 2024. [Online]. Available: https://neo4j.com/docs/aura/auradb/. [Geopend 16 4 2024]. |
| [17] |  | Neo4j, „Neo4j Pricing,” Neo4j, x x 2024. [Online]. Available: https://neo4j.com/pricing/. [Geopend 16 4 2024]. |
| [18] |  | Microsoft Azure, „Azure Container Apps,” Microsoft, x x x. [Online]. Available: https://azure.microsoft.com/en-us/products/container-apps. [Geopend 4 4 2024]. |
| [19] |  | Microsoft Azure, „Container Instances,” Microsoft, x x x. [Online]. Available: https://azure.microsoft.com/en-us/products/container-instances. [Geopend 10 4 2024]. |
| [20] |  | AWS, „Amazon Elastic Container Service,” AWS, x x x. [Online]. Available: https://aws.amazon.com/ecs/?pg=ln&sec=hiw. [Geopend 4 4 2024]. |
| [21] |  | AWS, „AWS Fargate,” Amazon Web Services, x x x. [Online]. Available: https://aws.amazon.com/fargate/. [Geopend 10 4 2024]. |
| [22] |  | AWS, „Pricing calculator,” AWS, x x x. [Online]. Available: https://calculator.aws/#/addService. [Geopend 14 5 2024]. |
| [23] |  | AWS, „AWS App Runner,” Amazon Web Services, x x x. [Online]. Available: https://aws.amazon.com/apprunner/. [Geopend 10 4 2024]. |
| [24] |  | Google Cloud, „Containers vs VMs (virtual machines): What are the differences?,” Google Cloud, x x x. [Online]. Available: https://cloud.google.com/discover/containers-vs-vms#:~:text=Virtual%20machines%20provide%20an%20abstracted,a%20physical%20or%20virtual%20machine.. [Geopend 7 4 2024]. |
| [25] |  | AWS, „Amazon EC2,” Amazon Web Services, x x x. [Online]. Available: https://aws.amazon.com/ec2/. [Geopend 10 4 2024]. |
| [26] |  | AWS, „Amazon Lightsail,” Amazon Web Services, x x x. [Online]. Available: https://aws.amazon.com/lightsail/. [Geopend 10 4 2024]. |
| [27] |  | Microsoft Azure, „App Service,” Microsoft, x x x. [Online]. Available: https://azure.microsoft.com/en-us/products/app-service. [Geopend 10 4 2024]. |
| [28] |  | AWS, „AWS Elastic Beanstalk,” Amazon Web Services, x x x. [Online]. Available: https://aws.amazon.com/elasticbeanstalk/. [Geopend 10 4 2024]. |
| [29] |  | M. Conclusion, „Mediaan Conclusion,” Mediaan Conclusion, 1 3 2024. [Online]. Available: https://mediaan.com/nl/cases/migratie. [Geopend 1 3 2024]. |