# Hoe ziet de architectuur van een monolieten applicatie eruit?

Een monolithische software applicatie is een traditionele architectuurstijl waarbij de hele applicatie wordt gebouwd als een enkel, op zichzelf staand programma. Deze aanpak wordt al jaren veel gebruikt en staat bekend om zijn relatief eenvoudige manier van ontwikkelen. Volgens Techopedia wordt een monolithische architectuur gekenmerkt door "een softwaretoepassing op één niveau waarin alle componenten in één codebase aanwezig zijn en als één pakket worden ingezet" (bron: Techopedia).

Een monolithische applicatie volgt in een gemiddelde applicatie een gelaagde architectuur. De verschillende lagen zijn verantwoordelijk voor specifieke aspecten van de functionaliteit van de applicatie. Er kan bijvoorbeeld een presentatie laag zijn, die verantwoordelijk is voor de gebruikersinterface, een Business Logic laag die de kernfunctionaliteit inkapselt, en een Data acces laag voor de interactie met de database.

Monolithische architecturen worden echter geconfronteerd met uitdagingen op het gebied van schaalbaarheid en onder houdbaarheid. Naarmate de applicatie groter en complexer wordt, wordt het moeilijker om veranderingen te onderhouden en uit te rollen. Om de applicatie te schalen moet de hele monoliet worden geschaald, wat inefficiënt kan zijn. Ondanks deze beperkingen worden monolithische architecturen nog steeds gebruikt in diverse bedrijfstakken, met name voor kleinere toepassingen met eenvoudiger eisen, waarbij de voordelen van snelle ontwikkeling en onderhoudsgemak opwegen tegen de schaalbaarheidsproblemen.

<https://www.techopedia.com/definition/27003/monolithic-kernel>

<https://www.techtarget.com/whatis/definition/monolithic-architecture>

# Hoe ziet de architectuur van een microservices applicatie eruit?

Een microservice is een architectuurstijl voor het bouwen van software-applicaties, waarbij de applicatie is opgedeeld in kleinere, onafhankelijke services die samenwerken om de functionaliteit van de applicatie te maken. In deze stijl van architectuur is elke microservice een zelfstandig onderdeel van de applicatie die kan worden ontwikkeld, geüpdatet en gedeployed zonder afhankelijk te zijn van de andere delen van de applicatie. Hierdoor wordt de applicatie flexibeler, schaalbaarder en eenvoudiger te onderhouden.

Elke microservice heeft zijn eigen interface en communicatieprotocol, waardoor het mogelijk is om verschillende technologieën en talen te gebruiken voor het bouwen van verschillende microservices in de applicatie. Zo kan de ene microservice bijvoorbeeld geschreven zijn in C#, terwijl een andere microservice bijvoorbeeld gebruik maakt van Java of Python. Door deze segregatie kunnen specifieke problemen binnen een applicatie opgelost worden die bijvoorbeeld beter geschikt zijn bij een bepaalde programmeertaal.

Een ander belangrijk kenmerk van microservices is dat ze goed geen directe afhankelijkheden hebben met andere microservices. Dit betekent dat een storing in één microservice niet de hele applicatie zal laten crashen, en dat elke microservice afzonderlijk kan worden getest en gedeployed kan worden naar bijvoorbeeld een cloud provider. Een microservice-applicatie wordt dan ook vaak gebruikt in combinatie met een kubernetes cluster. Een van de uitzonderingen hierop is de authentication service. Zodra deze een storing heeft kan de rest van de applicatie hier problemen mee ondervinden. Dit zal verder onderzocht gaan worden in dit onderzoek.

In het kort biedt de architectuur van microservices een manier om complexe softwareapplicaties te bouwen door deze op te splitsen in kleine, afzonderlijke diensten die kunnen worden ontwikkeld, getest, gedeployed en onderhouden met een hoge mate van autonomie en onafhankelijkheid.

Lewis, J., & Fowler, M. (2014). Microservices: a definition of this new architectural term. Retrieved from <https://martinfowler.com/articles/microservices.html>

<https://www.true.nl/blog/microservices-waarom-inzetten-voor-je-applicatie/>

<https://azure.microsoft.com/nl-nl/solutions/microservice-applications/>

# Wat zijn de voor- en nadelen van een microservices applicatie ten opzichte van een monolieten applicatie?

Zoals bij elk softwareproject wordt aan het begin van het ontwerp een keuze gemaakt over welke soort architectuur er gebruikt gaat worden. Terwijl veel applicaties nog gebruik maken van een monolieten architectuur, is een steeds voor de hand liggender ontwerp de microservice. In de volgende opsomming wordt kort toegelicht waarom je juist wel voor een microservice architectuur zou kunnen kiezen ten opzichte van een monolieten aanpak.

1. Betere Performance

Door een applicatie op te delen in kleine stukjes kan men beter inzien welke delen van het systeem juist wel of niet goed performen. In een grote applicatie kan het dus zijn dat een context van het systeem extreem veel gebruikt wordt, terwijl andere delen van de applicatie juist niet veel worden gebruikt. Door deze contexten te splitsen van elkaar krijgt men de mogelijkheid om beter horizontaal te schalen voor de meest gebruikte services. Zo blijft het systeem snel en draait het op een uitstekend performance level.

1. Beter onderhoudbare software

Dit betekent concreet dat software sneller doorontwikkeld kan worden, en dat aanpassingen en nieuwe features dus sneller bij de eindgebruiker terecht komen. De gedachtegang hier is dat kleine applicaties makkelijker te begrijpen zijn, en daarmee makkelijker aanpasbaar zijn en er minder bugs geïntroduceerd worden.

1. Efficiëntere werkverdeling

Naarmate een softwareproject groter wordt, en er meer mensen aan werken, groeit de benodigde communicatie tussen mensen in het project. Dit kan zo ernstig worden dat developers en productmanagers alleen nog maar aan het praten zijn, maar daadwerkelijk niets meer opleveren. Bij een microservices-architectuur kun je teams rond een microservice bouwen. Omdat de services per definitie klein zijn vermijd je dit probleem van de verlammende communicatie-overhead, en kan de werkverdeling optimaal ingedeeld worden.

1. Ontwikkeling in meerdere programmeertalen

Doordat alle applicaties hun eigen codebase hebben, en alleen via standaardprotocollen met elkaar praten is het mogelijk dat elke microservice in een andere taal geschreven wordt. Dit biedt de mogelijkheid om een taal uit te kiezen die precies past bij het presente probleem, en biedt je ook de mogelijkheid om uit een grotere pool developers te vissen.

<https://www.sqli.com/nl-nl/insights/blog/monoliet-vs-microservices-de-juiste-architectuur-kiezen-voor-uw-organisatie>

<https://www.emerce.nl/achtergrond/microservices-wel-of-niet-gebruiken>

Fowler, M., & Lewis, J. (2014). Microservices: a definition of this new architectural term. Retrieved from https://martinfowler.com/articles/microservices.html

Na het benoemen van een aantal voordelen zijn er natuurlijk ook nadelen aan het gebruiken van een microservices architectuur

1. Grotere complexiteit

Het Gebruik van microservices vereist meer coördinatie en management binnen een (of meerdere) development teams. Dit kan leiden tot extra complexiteit, waardoor de ontwikkeling en het onderhoud van een applicatie ingewikkelder kan worden. Daarnaast moeten de developers die werken aan een microservice meer kennis hebben dan een gemiddelde monolieten developers, sinds een groot onderdeel van een microservice-applicatie te maken heeft met gedistribueerde systemen en containerization zoals bijvoorbeeld Docker of Kubernetes.

1. Hogere kosten

Het gebruik van microservices kan leiden tot hogere kosten, bijvoorbeeld doordat er meer infrastructuur en resources nodig zijn voor het beheer en onderhoud van de applicatie. Vaak moet een microservice applicatie gedeployed worden naar bijvoorbeeld een Cloud provider, wat zeker niet goedkoop is. Ook kan de ontwikkeling van microservices duurder zijn, omdat er meer coördinatie en planning nodig is om de diensten te ontwikkelen en te integreren.

1. Hogere latency en foutgevoeligheid

Doordat microservices communiceren met elkaar door bijvoorbeeld HTTP of een message broker, kan de latency (vertraging) tussen de microservices hoger zijn dan bijvoorbeeld een monolieten applicatie. Daarnaast zijn er ook meer resources en werknemers nodig om deze microservices te onderhouden en beheren.

# Welke build in authenticatie mogelijkheden zijn er voor microservices?

Voor deze deelvraag is er gekeken naar de mogelijkheden voor een build in autorisatie voor microservices. Ondanks dat deze opties ook beschikbaar zijn als externe providers, is het doel van deze deelvraag dat deze technieken binnen de microservices worden geimplementeerd en niet via een externe Identity provider.

1. JWT (Json Web Tokens)

JWT is een populaire manier van authenticatie binnen microservices. Een JWT is een compact, zelf-inhoudelijk en cryptografisch beveiligd token dat informatie kan bevatten over de gebruiker en zijn/haar rol binnen de applicatie. De JWT kan worden gebruikt om de identiteit van de gebruiker te verifiëren bij elke microservice die de gebruiker aanroept. Voor verschillende programmeertalen zijn packages en extensies beschikbaar waardoor men in een microservice een token kan generen en valideren.

1. OAuth (2.0)

OAuth is een autorisatieprotocol dat veel wordt gebruikt voor authenticatie binnen microservices. Met OAuth kan een gebruiker een token krijgen waarmee hij/zij toegang heeft tot bepaalde resources binnen de applicatie. De verschillende microservices kunnen dit token verifiëren om te bepalen of de gebruiker toegang heeft tot de gewenste resources. Deze token wordt gegeven zonder dat een gebruiker zijn/haar gebruikersnaam en wachtwoord uit handen hoeven te geven.

Dit wordt bijvoorbeeld gedaan door in te loggen via google of facebook. Daar wordt gecontroleerd of dit de juiste gebruiker is en zo hoeft een gebruiker zijn/haar inloggegevens dus niet te delen met een bepaalde website.

1. OpenID

OpenID is een open standaard voor authenticatie, waarmee gebruikers één set aanmeldingsgegevens kunnen gebruiken om toegang te krijgen tot verschillende websites en applicaties. In plaats van dat gebruikers voor elke website of applicatie een nieuw account en wachtwoord moeten aanmaken, kunnen ze inloggen met hun bestaande OpenID.

OpenID is ontwikkeld om het proces van inloggen op verschillende websites te vereenvoudigen en te beveiligen. Het maakt gebruik van een open protocol om de authenticatie van gebruikers te verifiëren en biedt een veiligere methode om toegang te krijgen tot meerdere websites en applicaties. Ondanks dat deze manier vooral gebruikt wordt als externe authenticatie, zijn er ook build in mogelijkheden om dit lokaal op te lossen. Bij de deelvraag welke externe tools zijn er zal deze techniek ook nog voorbij komen.

<https://jwt.io/introduction/>

<https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>

<https://www.digitalocean.com/community/tutorials/an-introduction-to-openid>

<https://openid.net/>

# Welke externe authenticatie mogelijkheden worden het meest gebruikt binnen microservices applicatie?

Voor deze deelvraag is er gekeken naar 3 populaire externe tools om een microservice te authenticaten/autoriseren. Hieronder staan deze verder uitgelegd en benoemd:

## Keycloak

Keycloak is een open-source identity and access management (IAM) oplossing ontwikkeld door Red Hat. Het biedt authenticatie, autorisatie en single sign-on (SSO) mogelijkheden voor webapplicaties, microservices en API's. Keycloak volgt de OpenID Connect en OAuth 2.0 standaarden, waardoor het compatibel is met een breed scala aan applicaties.

In de kern fungeert Keycloak als een gecentraliseerde authenticatieserver, die de authenticatie en autorisatie van gebruikers afhandelt. Het ondersteunt verschillende authenticatiemechanismen, waaronder gebruikersnaam/wachtwoord, sociaals login (bijv. Google, Facebook) en Multi-factor authenticatie. Keycloak biedt ook functies voor gebruikersbeheer, zoals user registratie, wachtwoord reset en user profielbeheer. Daarnaast heeft keycloak ook functionaliteiten voor rollenbeheer.

Keycloak is een open source tool, maar heeft ook betaalde functionaliteiten zoals bijvoorbeeld het hosten van de authenticatie server en ondersteuning. Het is alleen ook heel goed mogelijk om keycloak zelf te hosten en te onderhouden, sinds er genoeg documentatie beschikbaar is

Keycloak werkt via een combinatie van client applicaties en de Keycloak server. Client applicaties communiceren met Keycloak om gebruikers te authentiseren en toegangstokens te verkrijgen. De server is verantwoordelijk voor het valideren van gebruikersgegevens, het beheren van sessies en het uitgeven van acces tokens. acces tokens bevatten informatie over de identiteit en machtigingen van de gebruiker en kunnen door server side Endpoint worden gebruikt om toegang te krijgen tot beschermde bronnen.

Official Keycloak Documentation: <https://www.keycloak.org/documentation>

## Okta

Okta is ook een IAM-platform (identity and access management) dat organisaties helpt bij het beheren van gebruikersidentiteiten, verificatie en autorisatie voor verschillende toepassingen en diensten. Het biedt een gecentraliseerd systeem voor het veilig beheren van gebruikerstoegang, het afdwingen van beveiligingsbeleid en het mogelijk maken van single sign-on (SSO) voor meerdere applicaties.

In de kern werkt Okta door het integreren van applicaties en services om op deze manier veilige user authenticatie en acces control te bieden. Wanneer een gebruiker toegang probeert te krijgen tot een applicatie, treedt Okta op als “Identity provider” en verifieert de identiteit van de gebruiker via verschillende inlogmogelijkheden, zoals wachtwoorden, biometrie of Multi-factor authenticatie (MFA). Na authenticatie geeft Okta een “Acces Token” uit waarmee de gebruiker toegang krijgt tot de applicatie. Dankzij deze token-based aanpak hoeven gebruikers niet meer meerdere inloggegevens te onthouden, wat beter is voor de gebruikerservaring en de security van de verschillende applicaties.

Okta website: https://www.okta.com/

Okta Developer documentation: https://developer.okta.com/

<https://support.okta.com/help/s/article/what-is-okta?language=en_US>

<https://hkrtrainings.com/what-is-okta>

## Auth0

Auth0 is een cloud-based identiteitsplatform dat authenticatie- en autorisatiediensten biedt voor applicaties, websites en API's. Het vereenvoudigt het proces van implementatie van veilige gebruikersverificatie door het aanbieden van vooraf gebouwde oplossingen voor gangbare verificatiemethoden zoals gebruikersnaam/wachtwoord, sociale login en Multi-factor authenticatie (MFA). Met Auth0 kunnen ontwikkelaars robuuste authenticatie- en autorisatiemogelijkheden toevoegen aan hun applicaties zonder complexe identiteitssystemen vanaf nul te hoeven bouwen.

Auth0 werkt als een authenticatie- en autorisatie laag tussen applicaties en identity providers. Wanneer een gebruiker toegang probeert te krijgen tot een applicatie of dienst, handelt Auth0 het authenticatieproces af. Het ondersteunt verschillende identity providers, waaronder sociaal media platforms zoals Google en Facebook, maar ook Enterprise identity providers zoals Active Directory en LDAP. Auth0 verifieert de identiteit van de gebruiker, handelt de registratie en aanmelding van gebruikers af, en genereert tokens die kunnen worden gebruikt om volgende verzoeken te authentiseren.

<https://auth0.com/docs/get-started/auth0-overview>

<https://www.mindbowser.com/what-is-auth0/>

<https://traefik-forward-auth0.readthedocs.io/en/latest/auth0/auth0.html>

# Wat zijn de meest gebruikelijke manieren van authenticatie binnen een microservices applicatie?)

De meest gebruikte methoden om microservices applicaties te authentiseren zijn token-based authenticatie, OAuth 2.0, en OpenID Connect. In de onderstaande kopjes worden deze 3 methoden verder belicht:

## Token-based authenticatie:

token-based authenticatie omvat het uitgeven en valideren van tokens om verzoeken te authentiseren en autoriseren. JWT's (JSON Web Tokens) worden veel gebruikt voor dit doel. Microservices ontvangen het token in elk verzoek en valideren de “Signature” ervan om de authenticiteit te waarborgen en relevante gebruikersinformatie te extraheren. In deze “Signature” staat vaak een unieke string aan random tokens die alleen bekend is aan de server kant van de microservice.

## OAuth 2.0:

OAuth 2.0 is een industriestandaard autorisatiekader dat vaak wordt gebruikt voor gedelegeerde autorisatiescenario's in microservices. Hiermee kunnen gebruikers beperkte toegang tot hun bronnen verlenen zonder hun referenties te delen. Volgens een artikel van Red Hat maakt OAuth 2.0 veilige interacties tussen microservices mogelijk door toegangstokens uit te geven die machtigingen verlenen. Deze tokens kunnen door microservices worden gevalideerd voordat het verzoek wordt verwerkt.

## OpenID Connect:

OpenID Connect bouwt bovenop OAuth 2.0 en voegt een authenticatie laag toe. Het biedt identiteitsverificatie en single sign-on (SSO) mogelijkheden voor microservices toepassingen. Volgens de OpenID Foundation stelt OpenID Connect microservices in staat gebruikers te authentiseren door ID-tokens te verkrijgen van een identity provider. Microservices kunnen de tokens valideren en gebruikersinformatie ophalen voor authenticatie- en autorisatiedoeleinden.

Bron: OpenID Foundation - "OpenID Connect Core 1.0 incorporating errata set 1" - <https://openid.net/specs/openid-connect-core-1_0.html>

<https://oauth.net/2/>

# Welke authenticatie mogelijkheid is het meest geschikt voor de kwetter applicatie?

Na het onderzoek naar verschillende mogelijkheden om microservices te authentiseren, zal er gekozen kunnen worden tussen build-in mogelijkheden en externe tools.

Externe tools:

Door gebruik te maken van externe tools neem je een hoop ontwikkelwerk bij jezelf weg. Doordat een hoop benodigde implementaties al bestaat, kun je hier gebruik van maken. Een nadeel hiervan is dan weer dat, op het moment dat je kiest hiervan gebruik te maken, je afhankelijk bent van de implementaties van die tool. Daarnaast is het zo dat er extra complexiteit komt kijken bij het verifiëren van gebruikers. Sinds de connectie tussen authenticatie server en microservices applicatie compleet ingericht moet worden.

Build-in:

Build-in mogelijkheden zijn een perfecte manier om de authenticatie binnen een microservice te houden en niet afhankelijk te zijn van externe tools. Door het downloaden van NuGet packages, houd je de benodigde functionaliteit binnen een service. Voor dit project is er gebruik gemaakt van de build in mogelijkheden van asp. net Core autorisatie en van Microsoft identitymodel. Door verschillende claims toe te voegen aan een JWT token, kun je zelf bepalen welke claims er in een token moeten, zoals bijvoorbeeld een Signature, expiratie date of ID.

Voor dit project is er gekozen om een build in mogelijkheid toe te passen. Dit heeft te maken dat er minder afhankelijkheden zijn dan met een externe tool. Daarnaast is het voor dit project uitstekend geschikt om gebruik te maken van een build in tool, wat ook minder tijd zal kosten om te implementeren.

Nuget Packages:

Microsoft aspnetcore authorization

System security claims

Microsoft.IdentityModel.Tokens;

# Hoe valideer ik of de authenticatie mogelijkheid werkt voor de kwetter applicatie?

In de laatste deelvraag zal een validatie test plaatsvinden om te testen of de geïmplementeerde authenticatie oplossing naar behoren werkt. Allereerst zal de context van de test worden uitgelicht, daarna zal er getest worden of er toegang verleend kan worden bij API Endpoint die afgeschermd zijn.

A screen shot of a computer code

Description automatically generated with low confidenceVoor deze test zal een token gegenereerd worden. Deze JWT token zal worden gebruikt om te valideren of de gebruiker recht heeft om de data te ontvangen van het desbetreffende Endpoint:

Figuur 2 Valideren van user en roles

A picture containing text, screenshot, software

Description automatically generated

Figuur 1 Genereren van het JWT Token

A screenshot of a computer

Description automatically generated with medium confidenceDe eerste test zal zijn op het Endpoint “[Controller])/validate”. Er zal geen JWT token zijn gespecificeerd. De expected outcome van deze test zal een : “401 – Unauthorized zijn”

A screenshot of a computer

Description automatically generated with medium confidenceZoals te zien krijg je een 401 – unauthorized terug. Laten we nu inloggen om te checken of we dan wel een 200 terugkrijgen.

Er wordt ingelogd en een JWT token wordt teruggegeven. Nu kunnen we deze Token kopiëren en meegegeven als Authorization Header.

A screenshot of a computer

Description automatically generated with medium confidenceZoals te zien is op de bovenstaande afbeelding is de zojuist verkregen token meegegeven als bearer token in de headers. Na een get request te plaatsen krijgen we inderdaad een 200 OK terug en een stukje tekst met User validated!

A screenshot of a computer

Description automatically generated with medium confidenceNu gaan we proberen om een admin request proberen op te halen, terwijl deze gebruiker daar geen authorizatie voor heeft.

Zoals te zien heeft deze gebruiker geen authorizatie voor deze Endpoint, en de server geeft hier dan ook een 403 -Forbidden terug.

A screenshot of a chat

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with low confidenceLaten we inloggen met een ander account die wel admin acces heeft. We voeren dan dezelfde stappen uit als met de vorige gebruiker, alleen horen we nu geen 403 terug te krijgen, maar een 200!

Zoals te zien geeft deze request netjes een 200 terug met de tekst user validated!

A picture containing text, line, font, screenshot

Description automatically generatedWaar zit nu precies het verschil? Zodra we beide jwt tokens decrypten komen we het antwoord tegen.

Zoals je kunt zien in de bovenstaande afbeelding, wordt de role meegegeven in de Token. Op het moment dat er een request wordt gemaakt met die token, wordt er gekeken of de role matched met wat er benodigd is om het Endpoint te mogen bereiken. Zodra dit klopt wordt de benodigde data teruggegeven.