THOMAS MORE

Azure reference environment - Terraform deployment automation on GitHub

Realization document

Sepp Eyckmans
Student Bachelor in de Elektronica-ICT – Cloud & Cyber Security

2025 - 2026

Inhoudsopgave

1. Inleiding

Binnen Arxus wordt gebruikgemaakt van steeds complexere cloudtechnologieën en automatiseringtools voor het implementeren en beheren van cloudinfrastructuur bij klanten. Voor veel klanten zijn deze technologieën echter relatief nieuw of onbekend, waardoor er onzekerheid kan bestaan over hun werking, meerwaarde en praktische toepasbaarheid. Hierdoor ontstond de behoefte aan een omgeving waarin deze technologieën niet alleen theoretisch kunnen worden toegelicht, maar ook op een concrete en praktijkgerichte manier kunnen worden gedemonstreerd.

Hoewel de conceptuele uitwerking van een dergelijke demonstratieomgeving reeds aanwezig was, bestond er nog geen technische implementatie waarmee deze omgeving op een efficiënte, consistente en reproduceerbare manier kan worden opgebouwd en beheerd.

Om dit probleem aan te pakken, werd tijdens mijn stage bij Arxus een Azure referentieomgeving ontwikkeld met behulp van Terraform voor de geautomatiseerde uitrol van infrastructuur en GitHub Actions voor de gestandaardiseerde uitvoering van de infrastructuur deployment. Dankzij deze aanpak kan de omgeving on-demand worden opgebouwd en afgebroken, waardoor ze geschikt is voor demonstraties.
(zie Glossary)

Dit document schetst de realisatie van mijn stageproject volgens de voorwaarden vastgelegd in het Project Charter (Eyckmans, 2026). Het doel is om de uitvoering van het project aan te tonen en te onderbouwen aan de hand van bewijsmateriaal en onderdelen van de deliverables (diagrammen, foto's, code snippets, …). Hierbij wordt geëvalueerd in welke mate, via alle succes criteria uit het Project Charter, het project is voldaan.

Daarnaast bevat dit document:

1. Een korte samenvatting van de projectstelling.
2. Het overzicht van de gebruikte tools.
3. De nodige analyse en research ter voorbereiding op de uitvoering van het project.
4. Een architectuur- en ontwerpoverzicht met bijhorende diagrammen.
5. Alle gedetailleerde implementatie aspecten volledig toegelicht.
6. Een beknopte conclusie over alles wat in dit project is gerealiseerd.

Indien er technische termen worden gebruikt die onduidelijk of onbekend zijn, kan men hiervoor terecht in de woordenlijst (Glossary) op het einde van dit document in bijlage. Bij de eerste vermelding van een term, wordt er steeds verwezen naar deze glossary.

2. Samenvatting van projectstelling

Voordat er verder wordt ingegaan op de technische realisatie van het project, wordt hier kort de oorspronkelijke projectstelling samengevat. Dit biedt de nodige context voor de verdere uiteenzetting van het stageproject. Voor een diepere omschrijving met zowel functionele als niet-functionele vereisten, bekijk het project charter document van dit project.
(Eyckmans, 2026)

2.1. Probleemomschrijving

Arxus is een Cloud Service Provider en een Microsoft Azure Expert MSP partner. Om klanten een consistente en betrouwbare service te bieden, zet Arxus sterk in op de standaardisatie van Azure Landing Zones en de automatisering van Azure implementaties. Deze implementaties worden uitgevoerd volgens de richtlijnen van het Microsoft Cloud Adoption Framework (CAF).
(zie Glossary)

Binnen deze context bestaat er voor de referentieklant “Demotronix” een standaard Azure Landing Zone architectuur. Momenteel is deze architectuur voornamelijk gedefinieerd op architectuurniveau en dient deze verder uitgewerkt te worden van concept naar een geautomatiseerde infrastructuur implementatie.

Het doel van dit project is om deze architectuur om te zetten naar Infrastructure as Code met behulp van Terraform. Hiervoor wordt gebruikgemaakt van de interne Terraform Module Library van Arxus. De infrastructuur zal automatisch uitgerold worden naar een bestaande Azure demo tenant via GitHub Actions pipelines.
(zie Glossary)

Het project heeft als doel een reproduceerbare en gestandaardiseerde implementatie van de Azure Landing Zone te realiseren volgens de principes en standaarden van het Microsoft CAF framework.

3. Analyse & Research

Onderstaande lijst geeft een overzicht van alle technologieën en tools die noodzakelijk waren om dit project te realiseren:

- **Microsoft Azure** (Cloud Provider)
- **Terraform** (Infrastructure as Code Tool)
- **GitHub** (Code Repository & Version Control)
- **GitHub Actions** (CI/CD)
- **Visual Studio Code** (Code Editor Omgeving)
- **Draw.io** (Diagram design program)
- **KeePass** (wachtwoordkluis)

Binnen deze lijst, zijn er een aantal belangrijke overkoepelende technologieën en tools waar ik de focus op wil leggen voor deze research & analyse, namelijk:

- **Microsoft Azure**
- **Terraform**
- **GitHub Actions**

Elk van deze technologieën en tools zullen afzonderlijk geanalyseerd en onderbouwd worden om het gebruik ervan te verantwoorden. Hoewel de keuze voor deze oplossingen vooraf al vastlagen, wordt in dit document elke optie kritisch geëvalueerd en vergeleken met relevante alternatieven. Op deze manier worden de gemaakte keuzes systematisch onderbouwd binnen een professioneel en analytisch kader. Deze evaluatie zal worden uitgevoerd aan de hand van de Weighted Ranking Method.
(zie Glossary)

3.1. Microsoft Azure

Microsoft Azure is het Cloud computing platform van Microsoft. Met meer dan 600 verschillende platform onderdelen & functionaliteiten in de aanbieding, is het een van de grootste en populairste Cloud platformen op de markt. De grootste concurrenten in dit veld zijn Amazon Web Services (AWS) en Google Cloud Platform (GCP).

3.1.1. Analyse

Microsoft Azure onderscheidt zich voornamelijk door de sterke integratie met het ecosysteem van Microsoft. Diensten zoals: Windows 11, Windows Server en Microsoft 365 zijn volledig geïntegreerd met Azure. Ontwikkeling omgevingen en gereedschappen zoals GitHub en Visual Studio Code hebben officiële Microsoft ondersteuning waardoor ontwikkeling aan Azure met deze diensten ideaal is. Programmeertalen van Microsoft zoals het .NET framework en C# sluiten naadloos aan op Azure. Deze integraties verlagen de implementatie drempel en verhogen de efficiëntie binnen organisaties die reeds gebruikmaken van Microsoft-technologieën. Bovendien biedt Azure uitgebreide ondersteuning voor automatisatie, continue integratie en levering via programma's zoals GitHub Actions en Terraform. Azure zet zich ook sterk in voor enterprise- en hybride cloud integratie via services zoals Azure Arc. Tot slot biedt Azure een Service-Level Agreement aan met een 99,9% gegarandeerde beschikbaarheid. Dit samen vormt een totaalpakket waarom Azure een populaire keuze is in de Cloudmarkt. (aws vs. azure, 2025)
(zie Glossary)

Amazon Web Services (AWS) staat bekend om zijn maturiteit en brede dienstaanbod. AWS biedt een zeer hoge mate van flexibiliteit en schaalbaarheid. Zowel op Cloud als op automatisatie en CI/CD vlak is dankzij de maturiteit en schaal van AWS een diepere integratie en compatibiliteit met automatisatie tools zoals Terraform. AWS ondersteunt exclusieve functionaliteiten binnen Terraform die op het moment van schrijven niet bestaan of ondersteund worden bij andere cloud providers. Amazon ondersteunt de open source gemeenschap dieper terwijl cloud providers zoals Azure meer hun focus legt op hun eigen ecosysteem. Integraties tussen AWS en diensten zoals Ansible, Terraform, GitHub en Linux hebben een native integratie met AWS. Hierdoor is AWS voor bedrijven met veel open source implementaties en diensten een meer tactische keuze als cloud provider. (aws vs. azure, 2025)

Google Cloud Platform (GCP) positioneert zich sterk op het vlak van data-analyse, big data en AI-applicaties. Omdat Google dezelfde infrastructuur gebruikt voor hun Cloud platform als voor hun eigen producten zoals Gmail en YouTube, staan zij sterk op voorsprong tegenover de competitie. Hoewel ze in het algemeen kleiner zijn en achterstaan op de concurrentie zoals AWS en Azure, zijn zij industrie leidend in big dataopslag, analyse en querying. Bovendien zijn ze gespecialiseerd in AI en machine learning, waardoor ze op dat vlak sterker zijn dan de concurrenten.
(DataCamp, 2025)

3.1.2. Weighted Ranking – Criteria en gewichten

Criteria	Uitleg	Gewicht
Dienstaanbod & flexibiliteit	Hoe breed en flexibel het platform is	20%
Integratie & ecosystemen	Integratie met bestaande tools en technologieën	20%
Wereldwijde dekking & datacenters	Wereldwijde dekking, datacenters en beschikbaarheid	20%
Automatisatie & CI/CD	Ondersteuning voor automatisatie en CI/CD	15%
Enterprise & hybride Cloud	Geschikt voor grote organisaties en hybride setup	15%
Data, AI & ML	Sterkte in data-analyse, AI en machine learning	10%
Totaal		**100%**

Elk criteria krijgt een score op 5, waarna een berekening op de gewogen scores wordt gecalculeerd om tot de totaalscore te komen.

Binnen de scope van dit project wordt veel belang gehecht aan flexibiliteit, integratie en beschikbaarheid. Daarom krijgen deze criteria een zwaardere weging.

3.1.3. Weighted Ranking – Totaalscore

Criteria	Gewicht	Azure	AWS	GCP
Dienstaanbod & flexibiliteit	0.20	5	5	3
Integratie & ecosystemen	0.20	5	3	3
Global presence & datacenters	0.20	4	5	3
Automatisatie & CI/CD	0.15	4	5	4
Enterprise & hybride Cloud	0.15	5	4	3
Data, AI & ML	0.10	4	4	5
Totaalscore	**1.00**	**4.55 / 5**	**4.35 / 5**	**3.35 / 5**

Omwille van het brede aanbod aan diensten, de wereldwijde beschikbaarheid, de automatisatie mogelijkheden en de ondersteuning van Enterprise- en hybride cloudoplossingen was de keuze voor Azure snel gemaakt. De doorslaggevende factor was echter de naadloze integratie en compatibiliteit met het Microsoft ecosysteem dat bij Arxus al sterk aanwezig is en uiteindelijk het verschil maakte.

3.2. Terraform

Terraform is een Infrastructure as Code (IaC) tool waarmee je infrastructuur via code kan opzetten, aanpassen en beheren. Je kan verschillende resources automatisch aanmaken, wijzigen, uitlezen en weer verwijderen, allemaal zonder manuele input. Dit alles wordt vastgelegd in code die onder versiebeheer staat. Hierdoor kan de configuratie op een gestructureerde manier opgeslagen en beheerd worden. Zo kan deze code op GitHub gezamenlijk ontwikkeld worden met ondersteuning voor integratie met CI/CD tools zoals GitHub Actions. (GeeksForGeeks, 2026)

De grootste concurrent in het IaC veld zijn OpenTofu, Pulumi en Bicep. OpenTofu is een afsplitsing van Terraform, Pulumi is een IaC alternatief dat algemene programmeertalen ondersteunt zoals Python en C#, tot slot is Bicep de IaC tool van Microsoft voor Azure.

3.2.1. Analyse

HashiCorp Terraform is de meest populaire en breed geadopteerde IaC tool op de markt. Het biedt IaC aan met diepe en robuuste complexiteit, stabiliteit, features en ondersteuning. Terraform gebruikt declaratieve configuratie code structuur i.p.v. imperatief. Dit betekent dat in de code gedefinieerd staat wat de definitieve gewenste staat moet zijn (bv. "ik wil 2 virtual machines"), niet hoe dit moet of welke stappen nodig zijn om er te geraken. Een van de grote voordelen aan Terraform is het cruciale state file component. Terraform bewaart de huidige staat in een state file, die zowel lokaal als remote op Azure kan bewaard worden zodat meerdere developers kunnen samenwerken aan dezelfde projecten. Hierdoor kan Terraform onthouden wat er allemaal bestaat, de verbanden tussen onderdelen in Azure en alles dat het al tot nu toe heeft opgebouwd. Zowel binnen als buiten Azure. Bovendien ondersteunt Terraform verschillende Cloud infrastructure providers, waaronder Microsoft Azure. Terraform is in staat dezelfde acties uit te voeren als die je in Azure portal kan uitvoeren. Dit kan gaan van Virtual Machines tot het aanmaken van netwerk route tables. Een mogelijk nadeel met Terraform is het gebrek aan lokale staat encryptie. Enkel als de state file bewaard wordt in de cloud (Azure, AWS, Terraform Cloud,…), is dit standaard geëncrypteerd. Terwijl OpenTofu wel lokale staat encryptie ondersteunt. (Terraform vs OpenTofu, 2025)

Bicep is het IaC tool gemaakt door Microsoft met naadloze integratie met Azure en andere Microsoft producten. Tegenover IaC alternatieve, is Bicep naadloos geïntegreerd met Azure en het Microsoft ecosysteem. Hierdoor is Bicep ideaal binnen een Azure omgeving. Een belangrijk voordeel aan Bicep, is hun dag 0 ondersteuning voor nieuwe functionaliteiten binnen Azure. Terwijl alternatieve IaC tools zoals Terraform deze nieuwe functionaliteit eerst moet implementeren in hun Azure module vooraleer dit kan gebruikt worden, kan dit onmiddellijk gebruikt worden binnen Bicep. Enkele nadelen van Bicep zijn het gebrek aan state management, gelimiteerd zijn aan Azure Cloud en het gebruik van enkel officiële Microsoft modules. (Wat is Bicep?, 2026)

OpenTofu is een fork van Terraform die beheerd wordt door de Linux Foundation. OpenTofu is ontstaan in 2023 na de Terraform licentie verandering van open source naar het Business Source Licence (Terraform Licence Change, 2025). Het grote voordeel van OpenTofu is de comptabiliteit en integratie met Terraform. Hoewel Terraform beschikt over een uitgebreidere en meer volwassen library van modules en providers dan het jongere tool OpenTofu, is het volledig in staat om dezelfde configuratie, modules en providers te gebruiken doordat het bijna identiek dezelfde tools zijn. Bovendien implementeert OpenTofu additionele features die niet in Terraform bestaan, zoals state encryptie. Wat de inhoud van de state file versleuteld, niet alleen tijdens gebruik, maar ook tijdens rust. De open source licentie is ook een stevig pluspunt tegenover Terraform. Maar OpenTofu heeft tragere adoptie van nieuwe functionaliteiten. Omdat alle nieuwe functionaliteiten eerst uitkomt op Terraform, waarna OpenTofu dit kan overerven, loopt OpenTofu altijd achter op Terraform. Bovendien beschikt OpenTofu over een minder optimale ondersteuning dan Terraform. Vooral omdat OpenTofu geen on-demand ondersteuning plan heeft dat Terraform wel bezit. (Terraform vs OpenTofu, 2025)

Pulumi is een open source IaC tool dat zichzelf onderscheidt van de competitie door hun configuratie taal te laten ondersteunen met populaire programmeertalen. Dit betekent dat je infrastructuur code kan schrijven in populaire programmeertalen zoals Java, Typescript, Python, Go, C# of zelfs YAML. Dit maakt het flexibeler dan de competitie. Het nadeel hieraan is dat hun provider library niet zo groot is als Terraform en OpenTofu, wat je opties voor Cloud providers of infrastructuur verkleint. (Pulumi vs. Terraform, 2026)

3.2.2. Weighted Ranking – Criteria en gewichten

Criteria	Uitleg	Gewicht
Adoptie en populariteit	Hoe breed gebruikt en vertrouwd de tool is.	25%
Features en maturiteit	Stabiliteit, complexiteit en aanwezige functies.	25%
Cloud & provider support	Hoeveel Cloud providers het ondersteunt en de robuustheid van de module library	20%
Automatisatie & CI/CD	Ondersteuning voor automatisatie en CI/CD	10%
Samenwerking & state management	Ondersteuning voor samenwerking en state management	10%
Licentiemodel	Impact van licentie op zakelijk gebruik	5%
Gebruiksgemak & flexibiliteit	Hoe toegankelijk de tool is voor nieuwe gebruikers.	5%
Totaal		**100%**

Elk criteria krijgt een score op 5, waarna een berekening op de gewogen scores wordt gecalculeerd om tot de totaalscore te komen.

Binnen de scope van dit project wordt veel belang gehecht aan adoptie, maturiteit, ondersteuning en integratie met de gekozen CI/CD technologie. Daarom krijgen deze criteria een zwaardere weging.

3.2.3. Weighted Ranking – Totaalscore

Criteria	Gewicht	Terraform	OpenTofu	Pulumi	Bicep
Adoptie en populariteit	0.25	5	4	3	3
Features en maturiteit	0.25	5	4	3	4
Cloud & provider support	0.20	5	4	3	2
Automatisatie & CI/CD	0.10	5	5	2	4
Samenwerking & state management	0.10	5	4	3	3
Licentiemodel	0.05	3	5	4	4
Gebruiksgemak & flexibiliteit	0.05	4	4	5	4
Totaalscore	**1.00**	**4.85 / 5**	**4.15 / 5**	**3.05 / 5**	**3.25 / 5**

Dankzij de flexibiliteit en stabiliteit van Terraform, blijft het de voorkeurskeuze voor bedrijven met grote, complexe en production-ready deployments op enterprise niveau. Dit is een combinatie van de brede adoptie en het veelvuldig gebruik van Terraform binnen bedrijven. Hun uitgebreide set aan functionaliteiten voor infrastructure as code is een groot pluspunt. Daarnaast is er een sterke integratie met Azure en GitHub Actions voor CI/CD binnen projecten. Terraform staat ook bekend om zijn betrouwbaar state management in complexe omgevingen. Hoewel het Microsoft ecosysteem sterk van toepassing is bij Arxus, is Terraform zeker voor dit project, waarbij collaboratie en integratie met GitHub Actions zeer belangrijks is, toch de betere keuze tegenover Microsoft Bicep of andere IaC concurrenten.

3.3. GitHub Actions

GitHub Actions is het ingebouwde CI/CD platform binnen GitHub. CI/CD maakt het mogelijk om nieuwe code automatisch te testen, te valideren en uiteindelijk uit te rollen met minimale manuele tussenkomst. Naast het testen en uitrollen van code, kan CI/CD ook ingezet worden voor verschillende andere taken. CI/CD wordt veel gebruikt voor het uitvoeren van unit tests of het compileren van applicaties. Voor het beheren van infrastructuur kan CI/CD gebruikt worden om infrastructuur uit te rollen of container images te bouwen. Ook algemene automatisering zoals: code formatering, code validatie en het hernieuwen van certificaten zijn allemaal mogelijk via CI/CD.

Populaire CI/CD alternatieve op de markt buiten GitHub Actions zijn Jenkins en Azure Pipelines. Jenkins is open source CI/CD oplossing waarbij het volledige CI/CD platform zelf kan gehost en beheerd worden. Terwijl Azure Pipelines de CI/CD oplossing is van Microsoft voor CI/CD integratie met Azure en andere Microsoft producten. Dit was hun originele oplossing van voor de Microsoft overname van GitHub in 2018.
(zie Glossary)

3.3.1. Analyse

GitHub Actions is het CI/CD oplossing van GitHub. Hun grote onderscheiding van alternatieve CI/CD oplossingen, is hun directe integratie met GitHub. Bijvoorbeeld kan workflows automatisch geactiveerd worden nadat er in een GitHub repository een pull request gemaakt wordt. Dit samen met alternatieve activatie zoals manuele en via cron job, maakt dit een flexibele automatisatie oplossing. Daarnaast biedt de GitHub Marketplace een uitgebreide bibliotheek van herbruikbare workflows en acties. Deze zijn door de community en/of door grote bedrijven zoals Microsoft gemaakt. Zo kunnen bestaande oplossingen gedeeld en gebruikt worden zonder dat deze telkens van nul moet uitgevonden worden. Bovendien biedt GitHub Actions sterke functionaliteiten zoals matrix strategieën, dispatch events, reusable workflows en actions. Deze versterken de schaalbaarheid, flexibiliteit en modulariteit van iedere CI/CD workflow. Tot slot wordt het meeste beheer van GitHub Actions achter de schermen gebeurt. Dit zorgt voor minder complexiteit. Aspecten zoals de GitHub Runners, de virtuele machines waarop alle CI/CD acties op worden uitgevoerd, zijn allemaal automatisch beheert door GitHub. Dit allemaal tezamen maakt GitHub Actions een zeer aantrekkelijk alternatief. Het enige mogelijke struikelblok, is het feit dat deze oplossing beperkingen heeft wanneer je code repository niet binnen GitHub staat. (Azure DevOps vs. GitHub Actions, 2025)

Azure Pipelines is de originele CI/CD oplossing van Microsoft. Het is een van de verschillende producten in het Azure DevOps pakket. Azure Pipelines distinctie van de concurrentie is door hun naadloze integratie met alle producten in het Microsoft ecosysteem, waaronder Azure. Hierdoor is het mogelijk om nieuwe infrastructuur code te pushen van Azure Repos naar Azure pipelines. Zodat er onmiddellijk nieuwe infrastructuur kan gebouwd worden in Azure. Bovendien komt Azure Pipelines ook met zijn eigen Marketplace. Hiermee kunnen pipelines gedeeld worden met community of externe technologieën geïntegreerd worden met Azure Pipelines. Tot slot biedt het ook sterke integratie aan met externe technologieën zoals GitHub en Terraform. Waardoor Azure Pipelines een sterke keuze is als CI/CD oplossing. Mogelijks zijn er wel limitaties wanneer je buiten het Microsoft ecosysteem gaat. Bijvoorbeeld als je code in GitHub staat, je Terraform gebruikt i.p.v. Bicep of je Linux wilt implementeren kunnen er mogelijks struikelblokken zijn die bij alternatieve CI/CD oplossingen niet bestaan. (Azure DevOps vs. GitHub Actions, 2025) (Khan, 2024)

Jenkins is een open-source CI/CD oplossing ontwikkeld door het Jenkins Project. Jenkins is populair vanwege de sterke focus op zelf hosten en de privacy voordelen die daarbij komen kijken. Bedrijven behouden hierdoor volledige controle over hun CI/CD-omgeving, aangezien deze in een private infrastructuur draait in plaats van in de cloud. Daarnaast staat Jenkins bekend om zijn uitgebreide integratiemogelijkheden met technologieën zoals GitHub, Microsoft Azure, Amazon Web Services en GitLab. Hierdoor kunnen bedrijven flexibel blijven en vermijden ze afhankelijkheid van één specifiek ecosysteem, zoals dat van Microsoft of GitHub. De keerzijde van Jenkins is echter de hoge complexiteit. Omdat de omgeving volledig zelf beheerd moet worden, ligt ook het onderhoud volledig bij het bedrijf zelf. Dit brengt extra beheer, tijdsinvestering en technische complexiteit met zich mee. (What is Jenkins?, 2026)

3.3.2. Weighted Ranking – Criteria en gewichten

Criteria	Uitleg	Gewicht
Integratie (ecosysteem)	Hoe goed de tool samenwerkt met andere systemen zoals GitHub of Azure.	20%
Onderhoud / complexiteit	Hoeveel tijd en moeite nodig is om de tool draaiende te houden.	20%
Gebruiksgemak / beheer	Hoe eenvoudig de tool te gebruiken is en hoeveel beheer nodig is.	15%
Flexibiliteit / features	Mogelijkheden zoals workflows, automatisatie en uitbreidingen.	15%
Schaalbaarheid	Hoe goed de tool kan meegroeien met grotere projecten.	10%
Community & Marketplace	Beschikbaarheid van kant-en-klare acties en ondersteuning.	10%
Ecosysteem afhankelijkheid	Hoe afhankelijk je wordt van één ecosysteem.	10%
Totaal		**100%**

Elk criteria krijgt een score op 5, waarna een berekening op de gewogen scores wordt gecalculeerd om tot de totaalscore te komen.

Binnen de scope van dit project wordt er veel belang gehecht aan integratie, complexiteit, gebruiksgemak en flexibiliteit. Daarom krijgen deze criteria een zwaardere weging.

3.3.3. Weighted Ranking – Totaalscore

Criteria	Gewicht	GitHub Actions	Azure Pipelines	Jenkins
Integratie (ecosysteem)	0.20	5	4.5	4
Onderhoud / complexiteit	0.20	4.5	4.5	1.5
Gebruiksgemak / beheer	0.15	4.5	4.5	2
Flexibiliteit / features	0.15	4.5	4	3
Schaalbaarheid	0.10	4.5	4.5	3.5
Community & Marketplace	0.10	4.5	4.5	4.5
Ecosysteem afhankelijkheid	0.10	2	2	5
Totaalscore	**1.00**	**4.35 / 5**	**4.18 / 5**	**3.15 / 5**

Voor dit project is er gekozen voor het gebruik van GitHub Actions. De naadloze integratie met GitHub en Azure maakt GitHub Actions een sterke keuze als CI/CD oplossing. Het biedt ook geavanceerde en flexibele functionaliteiten zoals Matrix Strategieën en Dispatch events. Dit gecombineerd met het delen van workflows via GitHub Marketplace en de schaalbaarheid met GitHub Runners. Maakte GitHub Actions de ideale CI/CD oplossing voor dit project. Hoewel Azure Pipelines, binnen een Microsoft ecosysteem, op eerste zicht de ideale keuze lijkt. Maar binnen de context van dit project, is GitHub Actions toch een betere oplossing. Omdat binnen dit project met GitHub als code repository wordt gewerkt, zijn er extra voordelen & functionaliteiten om zowel het code repository als CI/CD oplossing in hetzelfde ecosysteem te hebben. Met Azure Pipelines zijn deze functionaliteiten gelimiteerd omdat er niet volledig in Azure DevOps wordt gewerkt. Hierdoor is GitHub Actions toch gekozen in plaats van Azure Pipelines.

Dit betekent niet dat Azure Pipelines slechter is, bij Arxus wordt er nog veel gewerkt met Azure DevOps en Azure Pipelines bij klanten. Het biedt veel functionaliteit en voordelen aan. Maar als er gekeken wordt naar de scope van dit project, is GitHub Actions een betere aansluiting met de andere gebruikte technologieën.

3.4. Conclusie

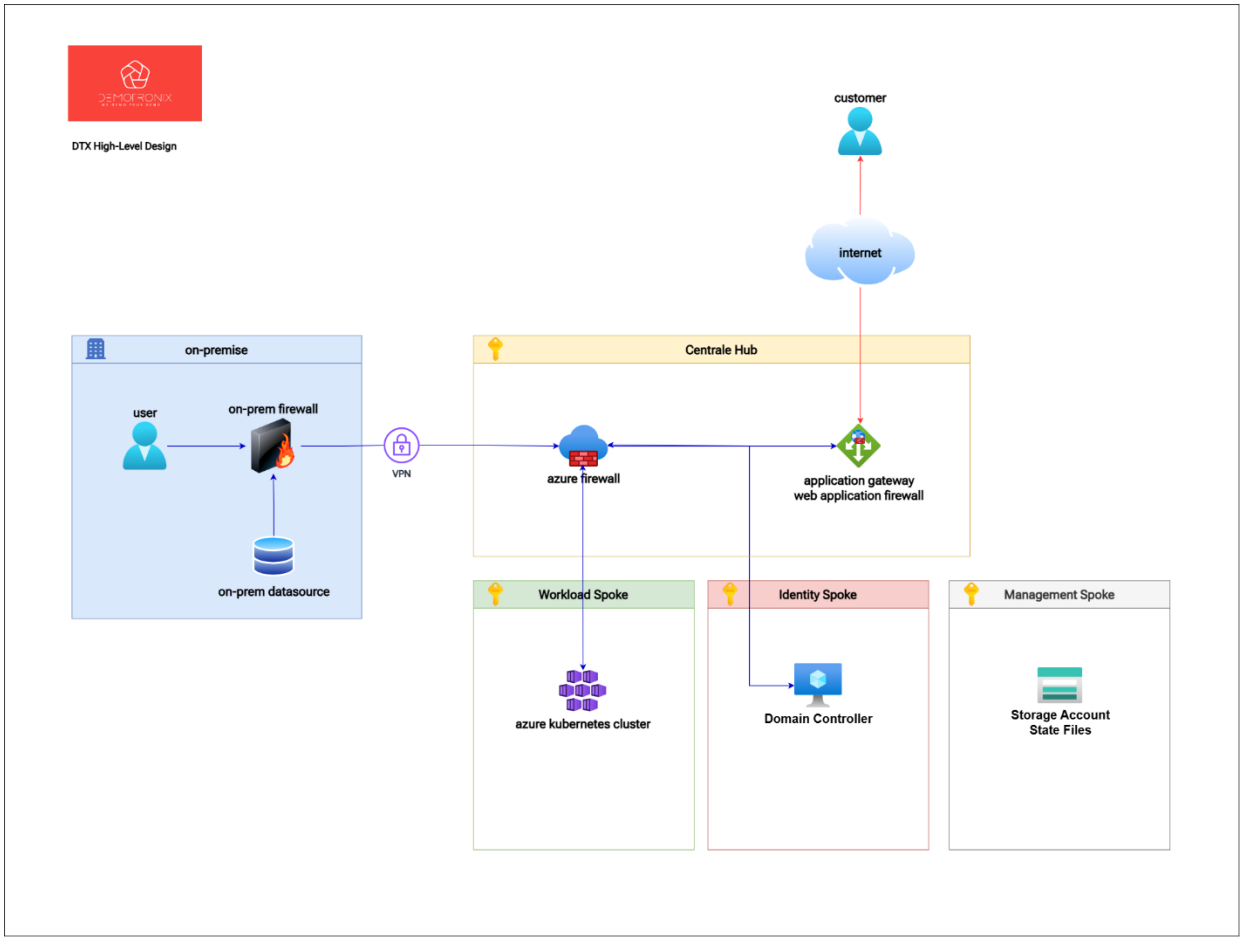
Hoewel de keuze voor alle technologieën en tools vooraf al vastlag, zijn ieder van deze keuzes marktbewuste en strategische beslissingen. Een belangrijke factor om naar te kijken is de integratie met Arxus zelf. Als Microsoft Expert MSP bedrijf, is het een grote toegevoegde waarde om Microsoft ecosysteem producten te kunnen gebruiken waardoor klanten erop kunnen vertrouwen dat deze oplossingen beheerd worden door gespecialiseerde experten. Bovendien speelt toekomstgericht denken ook een belangrijke factor. Hoewel het mogelijk is om uitsluitend voor Microsoft producten te kiezen, zijn er vandaag technologieën op de markt, zoals Terraform, die voor bedrijven in de komende jaren essentieel zullen worden om mee te blijven in het moderne IT landschap. Voor Arxus is het strategisch belangrijk om expertise op te bouwen en services aan klanten te verlenen daarin. Daarom is iedere technologie en tool in dit project niet zomaar gekozen, maar een bewuste en doordachte keuze, waarbij zowel huidige noden als toekomstige evoluties in rekening zijn gebracht.

4. Architectuur- en Ontwerpoverzicht

Voor we dieper ingaan op de implementatiedetails van het project, volgt eerst een macro en microniveau overzicht van de architectuur en het ontwerp. Dit biedt een helder beeld van wat er ontwikkeld zal worden en hoe de fictieve klanten omgeving eruitziet.

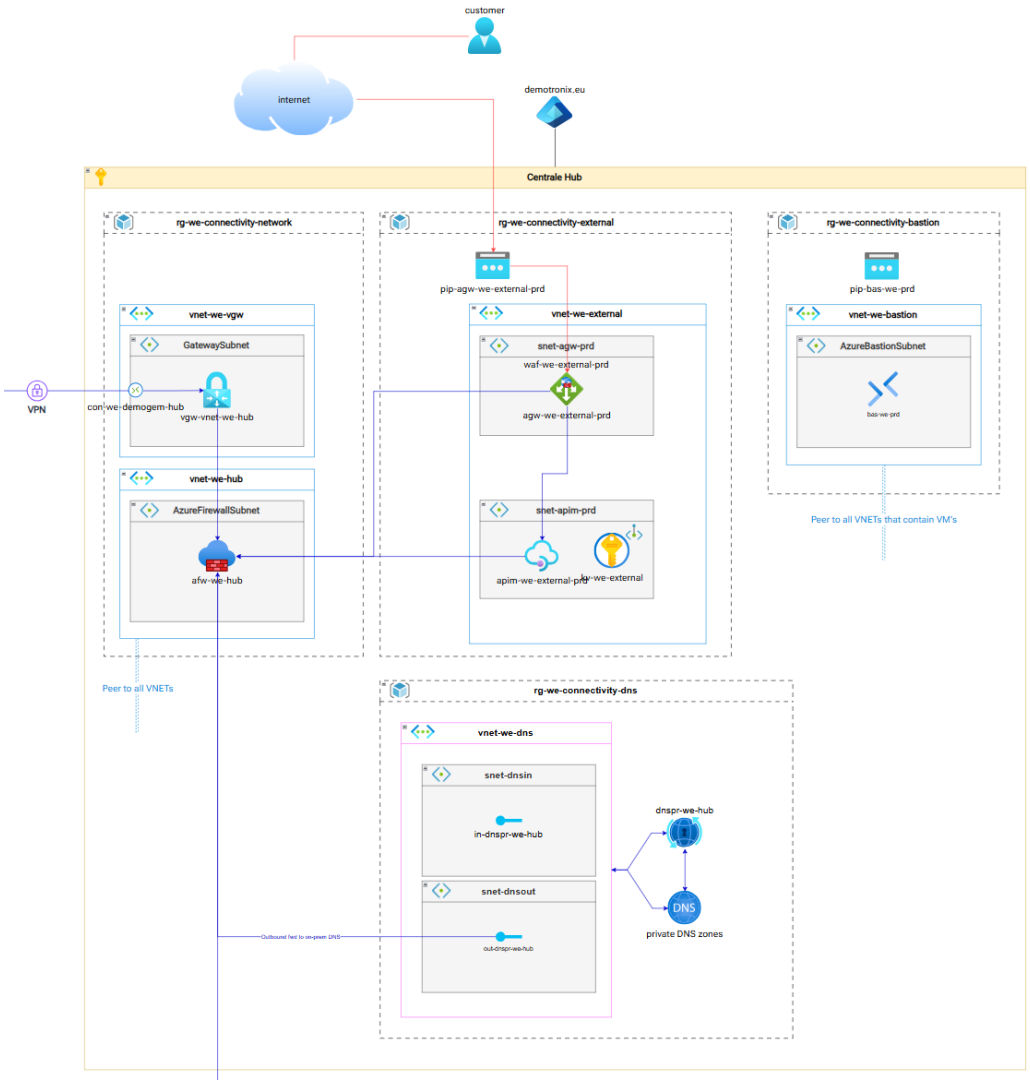
Een standaard Azure Landing Zone volgt het Hub-en-Spoke topologie model voor zijn architectuurdesign. Dit zorgt voor een flexibel en modulaire structuur van de omgeving. Via deze structuur kan eenvoudig nieuwe onderdelen toegevoegd worden zonder grote architectuur veranderingen. Ieder stuk van de omgeving is opgesplitst in zijn eigen onderdeel, genaamd een spoke. Iedere spoke heeft zijn eigen individuele subscriptie waarin alle netwerk en infrastructuur onderdelen onder vallen.

Zoals te zien in bijlage 8, zal voor Demotronix de standaard Azure Landing Zone bestaan uit een centrale hub en drie spokes: Identity, Management en de Workload spoke. In dit hoofdstuk gaan we dieper in op elk van deze onderdelen.



Bijlage 8: High-Level Design
(Zie *Bijlage 8* op pagina 62 voor een uitvergrote versie van "High-Level Design")

4.1. Centrale Hub



Bijlage 9: Hub Design
(Zie *Bijlage 9* op pagina 63 voor een uitvergrote versie van "Hub Design")

Het is mij gelukt om een centrale hub volledig op te zetten in de Demotronix omgeving volgens het design in bijlage 9. In dit hoofdstuk bespreek ik alle verschillende infrastructuur onderdelen, ik leg uit wat deze onderdelen zijn en hoe ze functioneren binnen Demotronix.

De centrale hub bevat de volgende onderdelen:

- **Azure Firewall**
- **VPN Gateway**
- **Bastion**
- **DNS Private Resolver**
- **DNS Zones**
- **Application Gateway & WAF**
- **API Management**
- **Key Vault**
- **Public & Private Endpoints**
- **Virtual Network Peering**

4.1.1. Azure Firewall

Azure Firewall is een managed netwerk firewall service die alle netwerkverkeer in Azure controleert, beschermt en filtert. Verkeer tussen Virtual Networks (VNets), van on-premises omgevingen naar Azure en van het internet naar Azure wordt gecontroleerd aan de hand van ingestelde regels, zoals weergegeven in bijlage 11. Deze firewall rules beslissen welk verkeer is toegestaan en welk verkeer wordt geblokkeerd. Bovendien kan het beslissen welk verkeer naar welk vnet mag gaan en of resources zoals API Management toegang krijgt tot externe Azure Storage Account voor additionele resources op te halen nodig voor deployment.

Ook ondersteunt het DNS Proxy functionaliteit, zodat er verkeer regels kunnen gemaakt worden gebaseerd op fully qualified domain names (FQDN), zoals te zien hieronder.

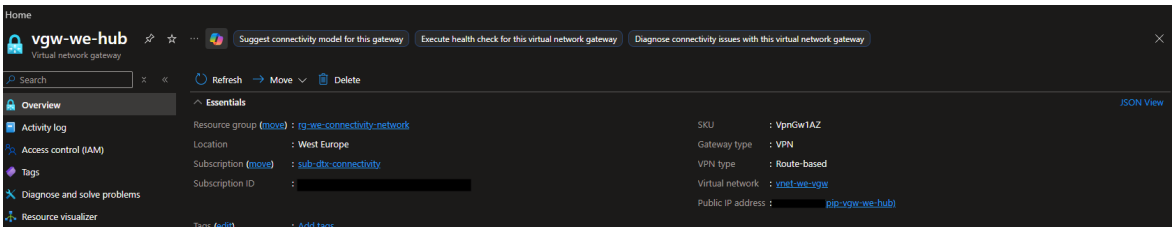
	column 1	column 2	column 3	column 4	column 5	column 6	column 7
1	name	source_addresses	source_ip_groups	protocols	fqdn_tags	destination_fqdns	web_categories
2	APIM-To-KV	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	kv-dtx-we-external.vault.azure.net	null
3	APIM-To-AAD	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	login.windows.net	null
4	APIM-To-Blob	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.blob.core.windows.net	null
5	APIM-To-File	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.file.core.windows.net	null
6	APIM-To-PackageStorage	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.blob.core.windows.net	null
7	APIM-To-SmtpQueue	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.queue.core.windows.net	null
8	APIM-To-Queue	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.queue.core.windows.net	null
9	APIM-To-Table	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.table.core.windows.net	null
10	APIM-To-RegionalAD	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	westeurope.login.microsoft.com	null

Bijlage 11: Firewall Rules
(Zie *Bijlage 11* op pagina 65 voor een uitvergrote versie van "Firewall Rules")

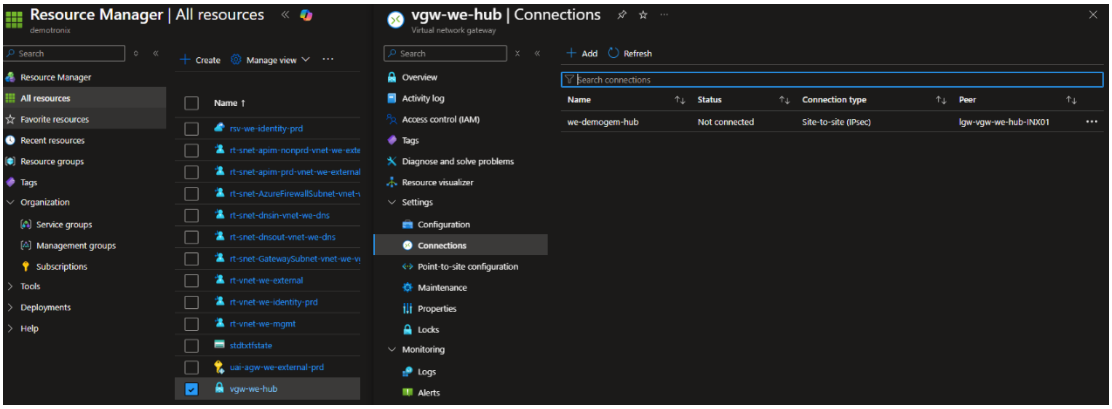
4.1.2. VPN Gateway

VPN Gateway is een Virtual Private Network (VPN) apparaat dat connectiviteit voorziet tussen een on-prem netwerk en het Azure centrale hub netwerk. Deze geëncrypteerde tunnel over internet legt een encrypted Site-to-Site VPN connectie tussen de 2 sites. Alle verkeer over de tunnel is geëncrypteerd en secure.

Binnen Demotronix, is het een Route-based Site-to-Site VPN Gateway die tussen het centrale hub netwerk in Azure en de fictieve on-prem omgeving Demogem een tunnel legt. Hoewel de connectie resource bestaat, is er in de praktijk geen echte connectie gemaakt, want Demotronix is een fictieve klant en hun on-premises is ook fictief, dus er is geen omgeving om naar te verbinden. Dit is te zien in bijlage 12. De tunnel connectie resource en VPN Gateway worden voor dit project toch opgezet voor demonstratie doeleinden, zoals te zien in bijlage 10 & 12.



Bijlage 10: VGW Overzicht
(Zie *Bijlage 10* op pagina 64 voor een uitvergrote versie van "VGW Overzicht")



Bijlage 12: VGW Connectie
(Zie Bijlage 12 op pagina 66 voor een uitvergrote versie van "VGW Connectie")

```
virtual_network_gateways = {
  vgw-we-hub = {
    prefix                      = "vgw" # Optional: Defaults to virtual_network_gateway value in var.prefixes
    local_network_gateway_prefix = "lgw" # Optional: Defaults to local_network_gateway value in var.prefixes
    public_ip_prefix            = "pip" # Optional: Defaults to public_ip value in var.prefixes
    connection_prefix           = "con" # Optional: Defaults to connection value in var.prefixes
    delimiter                   = "-"   # Optional: Change the delimiter. Defaults to -
    …
  }
}
```

Bijlage 6: VPN Gateway code snippet
(Om deze code snippet in groter formaat & beter kwaliteit te zien, ga naar Bijlage 6 op pagina 60)

4.1.3. Bastion

Azure Bastion is een beheerde Platform-as-a-Service (PaaS) jumpbox die op een veilige manier toegang biedt tot virtuele machines binnen een Azure tenant. Verbindingen verlopen zonder dat virtual machines een publiek IP adres nodig hebben of dat RDP/SSH poorten aan het internet worden blootgesteld.

Alleen de Bastion host beschikt over een publiek IP adres. Alle virtuele machines blijven hierdoor volledig bereikbaar via private IP, wat de beveiliging sterk verhoogt. Azure Bastion functioneert als centraal toegangspunt tot virtuele machines en laat gebruikers toe om verbinding te maken met elke VM binnen hetzelfde vnet of een peered vnet.

Bij Demotronix is Azure Bastion uitgerold in één vnet en via vnet peering verbonden met alle vnets waarin virtuele machines draaien, zoals te zien in de full code snippet versie van bijlage 7. Dankzij security rules en firewall rules, kunnen enkel Arxus consultants via de Azure Portal en Demotronix werknemers over VPN-connectie verbinden met de Bastion host.

```
bastion_hosts = {
  bas-we-prd = {
    prefix                      = "bas" # Optional: Defaults to var.prefixes.bastion
    delimiter                   = "-"   # Optional: Defaults to var.delimiter
    suffix                      = "we-bastion"
    custom_name                 = null # Optional: takes precedence over prefix, delimiter & suffix
    resource_group_name         = "rg-we-connectivity-bastion"
    bastion_sku                 = "Basic"             # Optional: Accepted values are Developer, Basic, Standard
and Premium. Defaults to Standard.
    zones                       = [1]                 # Optional: Defaults to [1, 2, 3]
    …
  }
}
```

Bijlage 7: Bastion Host code snippet

(Om deze code snippet in groter formaat & beter kwaliteit te zien, ga naar Bijlage 7 op pagina 61)

4.1.4. DNS Private Resolver

DNS private resolver is een Azure service die toelaat om naar Azure Private DNS zones te queryen van binnenin Azure of van on-prem naar Azure en vice versa. Dit vervangt het deployen van virtuele machines DNS servers in Azure of on-prem.

Wanneer dit wordt opgezet in een vnet, wordt er een Inbound Endpoint en een Outbound Endpoint gemaakt en in aparte subnets geplaats binnenin het vnet. Het Inbound Endpoint krijgt een dedicated IP waarmee alle vnets en resources die een DNS server nodig hebben via dit IP adres toegang krijgen tot DNS functionaliteit via de private resolver en de DNS zones die eraan verbonden zijn.

Via deze methode is er DNS functionaliteit binnen Demotronix. Bij Demotronix wordt er ook een Outbound Endpoint gemaakt om al het verkeer dat richting on-premises moet, via conditional forwarding door te sturen.

4.1.5. Private DNS Zones

Azure Private DNS Zones is een dienst in Azure die zorgt voor DNS resolutie binnen een vnet. In een Private DNS Zone kan je zelf DNS records toevoegen, of DNS records automatisch laten aanmaken via Azure Private Link.

Bij Demotronix worden de Private DNS Zones centraal gemaakt. Ze worden ook gekoppeld in de Terraform code aan Azure Private Link. Wanneer een nieuw Private Endpoint wordt gedeployed en gekoppeld is aan een Private DNS Zone, wordt automatisch een DNS record gemaakt.

Op die manier krijgt elke nieuwe service met een Private Endpoint automatisch een DNS record. Alle services die gebruikmaken van de Private DNS Resolver kunnen zo correct verbinding maken met het Private Endpoint, zonder manuele configuratie. Bovendien is de levenscyclus van deze DNS record gekoppeld aan de Terraform resource. Wanneer Terraform de resource verwijdert, wordt ook het bijhorende DNS record automatisch verwijderd.

Bij Demotronix worden Private Endpoint momenteel uitsluitend gebruikt voor de koppeling met Key Vaults om secrets en certificaten op te halen. Door ze binnen dit project modulair op te zetten, wordt het mogelijk om in de toekomst eenvoudig ook Private Endpoints voor andere use cases te implementeren.

4.1.6. Application Gateway & WAF

Een Application Gateway is een web traffic (OSI Layer 7) load balancer die verkeer beheert en vervoert naar de juiste applicatie. Gebaseerd op de gekozen SKU bij het deployen, heeft het ook een built-in Web Application Firewall (WAF) zodat het ook deze web verkeer slim kan blokkeren indien malicieus.
(zie Glossary)

Bij Demotronix wordt de Application Gateway met built-in WAF deployed als public-facing Internet gateway tot de Azure omgeving. Alle web verkeer komt hier binnen en routeert zich rechtstreeks naar de Azure Firewall om zo uiteindelijk in de AKS cluster te geraken in de workload spoke. Via de WAF wordt er ook een eerste vorm van web filtering en security geïmplementeerd. Door het activeren van de “Microsoft Default Rule Set 2.1” security rules op de WAF, beschermt het Azure tegen verschillende security threats vooraleer er verkeer zelfs binnen in Azure komt.

Ten slotte wordt het HTTPS protocol gebruikt met signed certificaten uit de Key Vault zodat publiekelijk toegang van het internet tot Azure veilig gebeurt.

4.1.7. API Management

API Management is een Platform-as-a-Service (PaaS) dienst binnen Azure die het maken, publiceren, monitoren en beheren van API's centraliseert op een gestructureerde en schaalbare manier met automatische schaalbaarheid en failover. Deze service wordt public-facing gepositioneerd, vanwege de ingebouwde developer portal.

De developer portal is een website die volledig gehost wordt binnen API Management en de volledige documentatie van alle API's bevat. Programmeurs gebruiken deze portal om API's te verkennen, te leren gebruiken en te testen. Omdat deze omgeving toegankelijk moet zijn voor externe developers, is een publiekgerichte opstelling noodzakelijk.

Binnen Demotronix wordt een Internal API Management (APIM) uitgerold in de Azure omgeving. Deze omvat een Developer Portal die, hoewel de APIM volledig private staat, toch toegankelijk wordt gemaakt voor externe gebruikers. Dit gebeurt door de portal via de AGW te exposen aan het internet. Hoewel APIM in dit project verder niet actief gebruikt wordt, moet de configuratie zodanig opgezet worden dat collega's in de toekomst eenvoudig en modulair nieuwe API's kunnen toevoegen bij het deployen van APIM. Daarnaast wordt APIM zo ingericht dat developers via de portal API's rechtstreeks kunnen raadplegen en testen, op voorwaarde dat zij over de juiste authenticatie beschikken.

Het doel van het inzetten van APIM is om collega's binnen Arxus in staat te stellen snel API's op te zetten voor demonstratiedoeleinden. Deze API's kunnen vervolgens eenvoudig gekoppeld worden aan een webapplicatie binnen een workload spoke, zodat snel een werkende demo-omgeving kan worden opgezet.

4.1.8. Key Vault

Key Vault is een service binnen Azure die ontworpen is om sensitieve informatie op een veilige en centrale manier op te slaan, zoals API sleutels, wachtwoorden en certificaten. Door deze gegevens te centraliseren in één beveiligde omgeving, helpt Key Vault om toegang tot secrets beter te beheren en te controleren.

Binnen de connectivity subscriptie bij Demotronix wordt de Key Vault specifiek ingezet voor het opslaan en beheren van het Demotronix certificaat. Dit certificaat wordt gebruikt voor zowel de AGW als het developer portal van API Management. Door het certificaat centraal te bewaren in een Key Vault, kan op een veilige manier en consistente manier deze gedeeld en gebruikt worden tussen verschillende services zoals APIM, AGW of meer. Bovenop vermijdt dit het hard-coderen van secrets in applicaties of certificaten te bewaren op onveilige plaatsen en/of zonder encryptie.

4.1.9. Public & Private Endpoints

Binnen Demotronix worden publieke endpoints gemaakt voor specifieke services waar toegang vanaf het internet naar nodig is zoals AGW, APIM en Bastion. Deze endpoints worden automatisch bij het deployen van deze services in Terraform aangemaakt. Een publiek endpoint is een publiek IP adres dat van zowel binnen als buiten Azure routeerbaar is.

Een Private Endpoint (ook wel Private Link genoemd) is een netwerkinterface die een service op een privé en beveiligde manier beschikbaar maakt binnen een vnet. In tegenstelling tot een publiek endpoint krijgt een Private Endpoint een privé IP adres uit het IP range van het vnet, waardoor de service niet rechtstreeks via het publieke internet bereikbaar is. Toegang enkel mogelijk binnenin Azure of vanaf on-prem via de VPN Gateway.

Voor dit project krijgen enkel de Key Vaults een Private Endpoint. Dit is omwille van het feit dat, voor de basisfunctionaliteit te garanderen, bepaalde services toegang moeten hebben tot een Key Vault. Door gebruik te maken van een Private Endpoint wordt deze toegang beperkt tot het interne netwerk, zonder blootstelling aan het publieke internet.

Bovendien zorgt Private Endpoint voor automatische registratie van DNS records. Hierdoor worden de benodigde private DNS-records automatisch aangemaakt en beheerd bij het maken van een nieuwe Private Endpoint, zodat services binnen het Virtual Network de Key Vault via de juiste interne naamresolutie kunnen bereiken zonder extra configuratie.

4.1.10. Virtual Network Peering

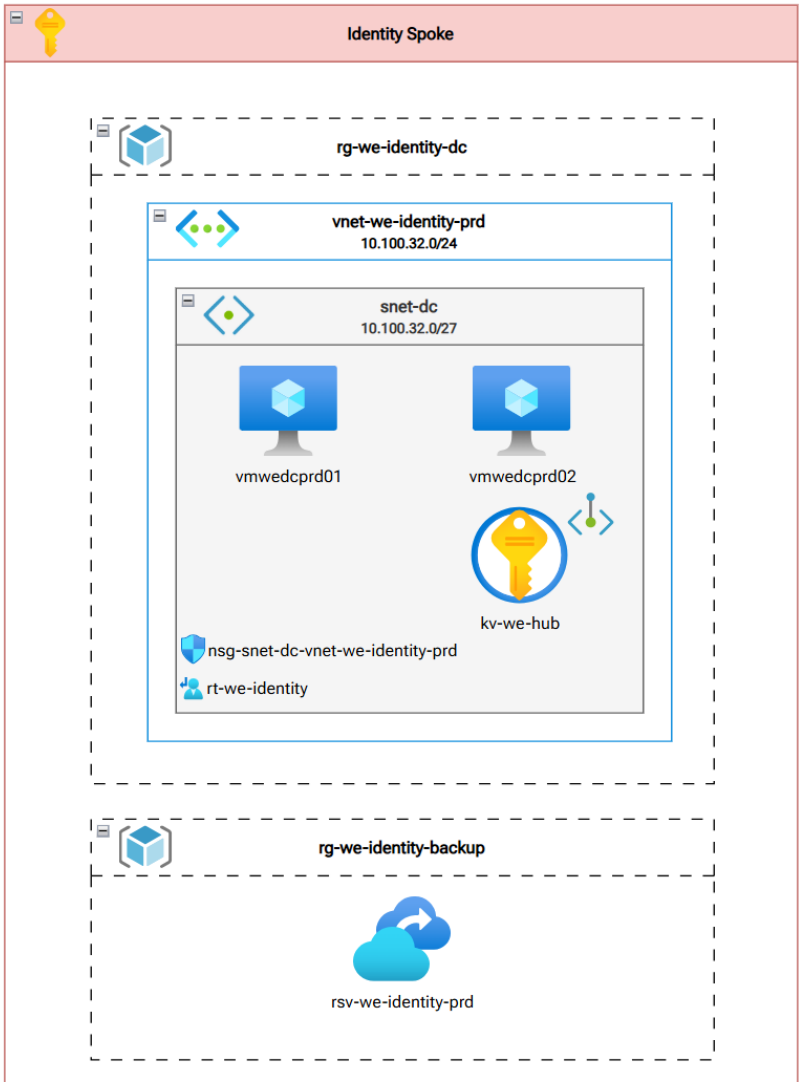
Virtual Network Peering is een cruciaal onderdeel van het centrale hub onderdeel binnen Demotronix. Het is een netwerk functionaliteit waarmee twee virtuele netwerken rechtstreeks met elkaar verbonden worden. Deze peering is de primaire manier waarop vnets onderling kunnen communiceren binnen Azure.

Binnen een gestandaardiseerde Azure Landing Zone worden standaard de volgende peerings gelegd:

- Peering tussen elk vnet en het Firewall vnet in de hub
  - Zorgt ervoor dat al het netwerkverkeer van workloads via de centrale firewall loopt.
  - Doordat het DNS vnet gekoppeld is aan het Firewall vnet, kunnen alle vnets met een peering naar de firewall gebruikmaken van de DNS Private Resolver.
- Bastion vnet is de enige uitzondering die geen peering moet hebben met Firewall vnet
  - Peering tussen het Bastion vnet en elk vnet met Virtual Machines
  - Maakt het mogelijk dat de Bastion Host veilig verbinding maakt met Virtual Machines in andere vnets, zonder publieke IP-adressen.

Hoewel het voor mij een moeilijk concept was om te begrijpen, is het van groot belang om de basic functionaliteit te garanderen binnen de Demotronix Azure omgeving.

4.2. Identity Spoke



Bijlage 16: Identity Spoke
(Zie Bijlage 16 op pagina 70 voor een uitvergrote versie van "Identity Spoke")

Deze subscriptie bevat de resources nodig voor het beheren van de Domain Controllers en back-ups van deze Domain Controller Virtual Machines, zoals te zien in bijlage 16. Ten slotte worden de login gegevens van deze VM bewaard in een Key Vault.

Deze subscriptie bevat de volgende resources:

- **Virtual Machines (DC 1 & DC 2)**
- **Key Vault + Private Endpoint**
- **Recovery Service Vault (RSV)**

4.2.1. Virtual Machines (DC 1 & DC 2)

Deze virtuele machines zijn standaard Windows Server Datacenter 2025 machines. Na deployment starten ze in een out-of-the-box staat en vereisen nog configuratie. Ze dienen geconfigureerd te worden als domain controllers voor het domein "demotronix.eu", zodat zowel Azure als on-premises omgevingen gebruikt kunnen maken van deze domain controllers voor aanmelden en authenticatie.

De configuratie van deze out-of-the-box VMs tot domain controllers valt buiten de scope van dit project.

4.2.2. Key Vault + Private Endpoint

Een Key Vault instantie met een Private Endpoint werd aangemaakt en gelinkt aan DNS Private Zone en Azure Private Link om automatisch een DNS record te maken bij het creëren van de Key Vault.

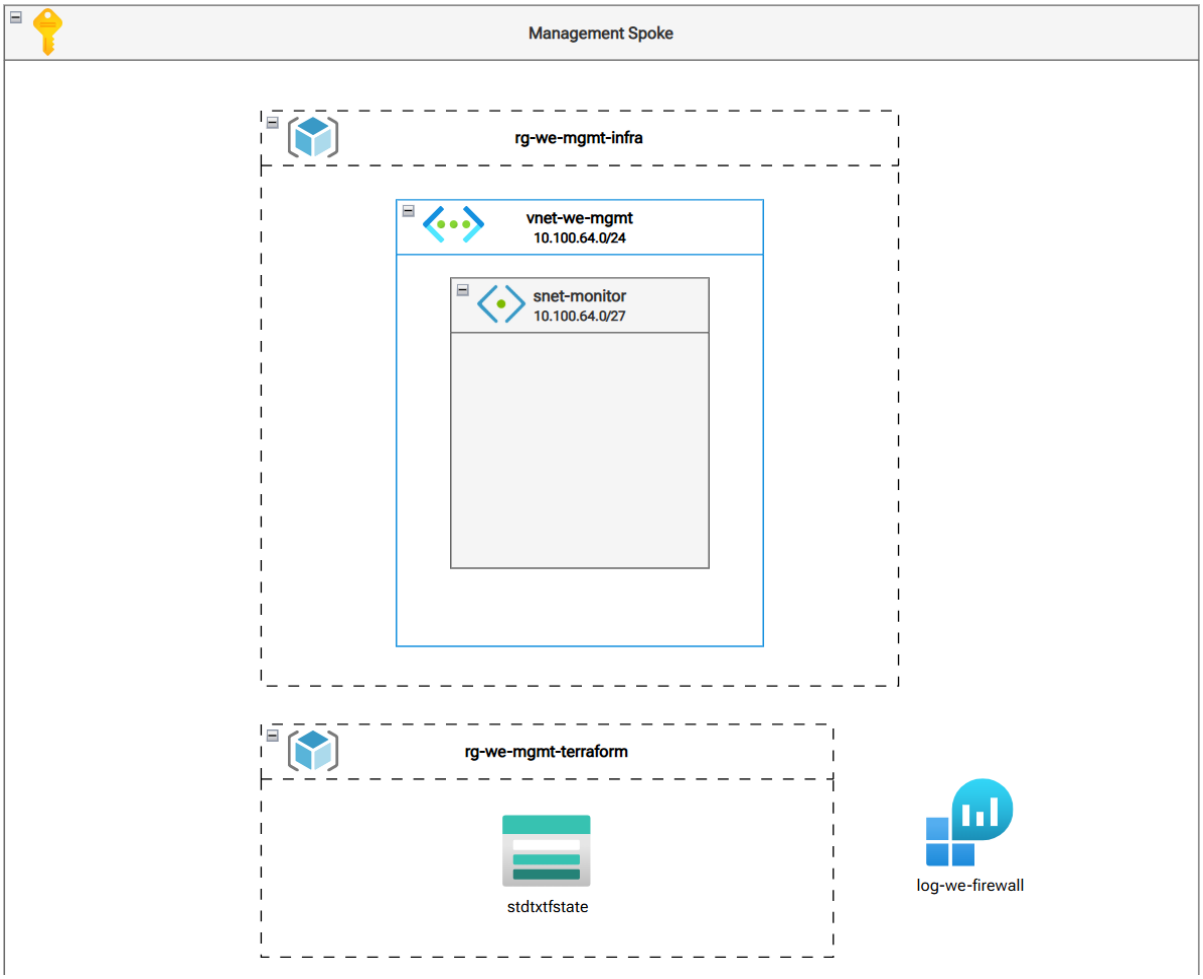
Binnen Demotronix dient deze Key Vault voor het bewaren van de login gegevens van de Domain Controller Virtual Machines.

4.2.3. Recovery Service Vault

Recovery Service Vault (RSV) is een service binnen Azure die gebruikt wordt om back-ups en herstelpunten te maken en te beheren voor zowel Virtual Machines als SQL databases. Het is een centrale opslagplaats voor back-ups, back-up policies en herstelpunten. Een back-up policy definieert de retentie range en schema om automatische back-ups te implementeren voor maximale back-up bescherming.

Binnen Demotronix, wordt er een RSV deployed om de Domain Controller Virtuele Machines te back-uppen zodat in het geval van een ramp, de Virtuele Machines kunnen hersteld worden. Bovendien wordt er een zelfgemaakte back-up policy gebruikt zodat er een maal iedere dag automatisch een back-up gemaakt wordt van de Virtuele Machines zonder manuele input.

4.3. Management Spoke



Bijlage 17: Management Spoke
(Zie *Bijlage 17* op pagina 71 voor een uitvergrote versie van "Management Spoke")

In bijlage 17 kan je zien dat deze subscriptie de nodige resources bevat voor het beheren van de Firewall logs via een Log Analytics Workspace, plaats voor toekomstige uitbreiding van de Landing Zone voor bijvoorbeeld een monitoring setup en ten slotte een storage account met alle Terraform state files.

Deze subscriptie bevat de volgende resources:

- **Log Analytics Workspace**
- **State file Storage Account**

4.3.1. Log Analytics Workspace

Een Log Analytics Workspace is een centrale opslagplaats voor logs en analyse omgeving voor performantie metingen & telemetrie. Een Log Analytics Workspace resource kan gemaakt worden en gekoppeld worden aan welke service dan ook en zal hier logs voor verzamelen, performantie metingen opslaan en telemetrie bewaren indien deze bestaan.

In dit project binnen Demotronix wordt dit enkel gebruikt om logs en andere nuttige metingen te bewaren en te verzamelen voor de Azure Firewall.

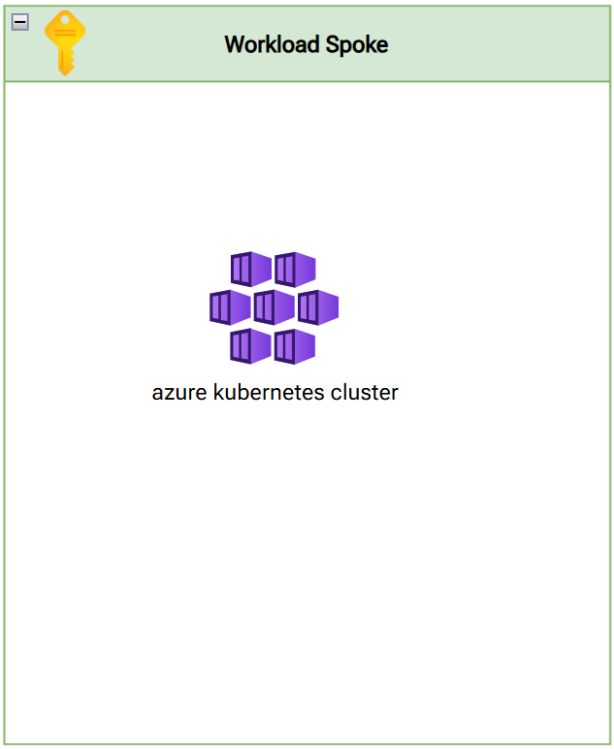
4.3.2. State File Storage Account

Dit is een speciaal aangemaakte Azure Storage Account die dient als backend voor Terraform. Het was één van de noodzakelijke vereisten om een geautomatiseerde deployment via Terraform mogelijk te maken.

De state file speelt een grote rol bij Terraform. Deze file functioneert als het "geheugen" van Terraform en houdt exact bij welke infrastructuur er bestaat en welke wijzigingen er in de loop van de tijd zijn doorgevoerd. Zonder deze state kan Terraform niet correct bepalen wat er moet worden aangemaakt, aangepast of verwijderd bij een nieuwe deployment.

Door de state file op te slaan in een Azure Storage Account, wordt deze centraal, betrouwbaar en veilig beheerd. Dit zorgt er bovendien voor dat de state toegankelijk is voor de volledige CI/CD pipeline, zodat alle pipeline runs met dezelfde actuele infrastructuurstatus werken. Daarnaast biedt Azure Storage ingebouwde beveiligings- en toegangscontrolemechanismen, waardoor de state file niet lokaal hoeft te worden opgeslagen en het risico op verlies of inconsistentie wordt vermeden.

4.4. Workload Spoke



Bijlage 18: Workload Spoke
*(*Zie *Bijlage 18* op pagina 72 voor een uitvergrote versie van "Workload Spoke")

Deze subscriptie bevat de Azure Kubernetes Service (AKS) waarin alle container workloads draaien in de Demotronix Landing Zone, te zien in bijlage 18.

Deze subscriptie bevat de volgende resource:

- **AKS**

4.4.1. AKS

Dit is de Kubernetes cluster waarin alle workloads worden uitgevoerd. De cluster is rechtstreeks verbonden met de AGW via een Kubernetes ingress connectie met de AGW backend. Via deze verbinding kunnen externe gebruikers, dus ook buiten Azure, toegang krijgen tot webapplicaties die als pods binnen de cluster draaien.

Door deze architectuur kunnen requests vanuit het internet via de AGW veilig worden doorgestuurd naar de juiste services binnen de Kubernetes cluster, wat zorgt voor een gecontroleerde, schaalbare en veilige toegang tot de applicaties.
(zie Glossary)

Door tijdsbeperkingen binnen het project is het niet gelukt om een aparte AKS spoke te implementeren. Toch is het relevant om deze spoke te vermelden aangezien dit de start kan vormen van een mogelijk toekomstig project en een basis kan bieden voor verdere uitbreiding van de architectuur.

4.5. Summary

Om het Architectuur en ontwerp deel af te sluiten, even een overzicht van alle verschillende Azure services die er moeten opgezet worden via Terraform:

- **Resource Groups**
- **Virtual Networks**
- **Azure Firewall**
- **VPN Gateway**
- **Bastion**
- **DNS Private Resolver**
- **DNS Zones**
- **Application Gateway & WAF**
- **API Management**
- **2 Key Vaults + Private Endpoint**
- **Virtual Network Peering**
- **Virtual Machines (DC 1 & DC 2)**
- **Recovery Service Vault (RSV)**
- **Log Analytics Workspace**

Tot slot ook nog een lijst van alle publieke endpoints die binnen Azure worden gemaakt:

- **Application Gateway (AGW) Publieke Endpoint**
- **Bastion Host Publieke Endpoint**

Eerst moeten alle GitHub deployment repositories aangemaakt worden. Dit gebeurt via **GitHub-manager** voor de effectieve GitHub repositories en **IAM** voor alle rollen en permissies die moeten gezet worden. Hierdoor zijn alle permissies ingeregeld om alle nodige functionaliteiten effectief te kunnen uitvoeren. Zoals de toegang tot Azure en het beheren van alle Azure diensten.

Alle infrastructuur hier besproken, wordt volledig uitgerold en beheerd in Terraform code. Deze realisatie is enkel mogelijk dankzij de Terraform Module Library. Meer hierover in het volgende hoofdstuk.

5. Terraform & Terraform Module Library

De Terraform Module Library is de interne en zelfgemaakte Terraform module collectie van Arxus. Iedere Azure service (Resource Group, Virtual Network, Firewall, DNS Private Resolver,…) heeft zijn eigen Terraform module om die service te beheren en uit te rollen. Het maakt niet uit in welke environment van een Azure Tenant of in welke Azure Tenant de resources moeten worden uitgerold. Alle modules van alle Azure services zijn te vinden in deze uitgebreide library.

Mijn taak is om deze bestaande module library te gebruiken om de demo omgeving op te zetten. Het is niet te bedoeling om de module library van nul te bouwen.

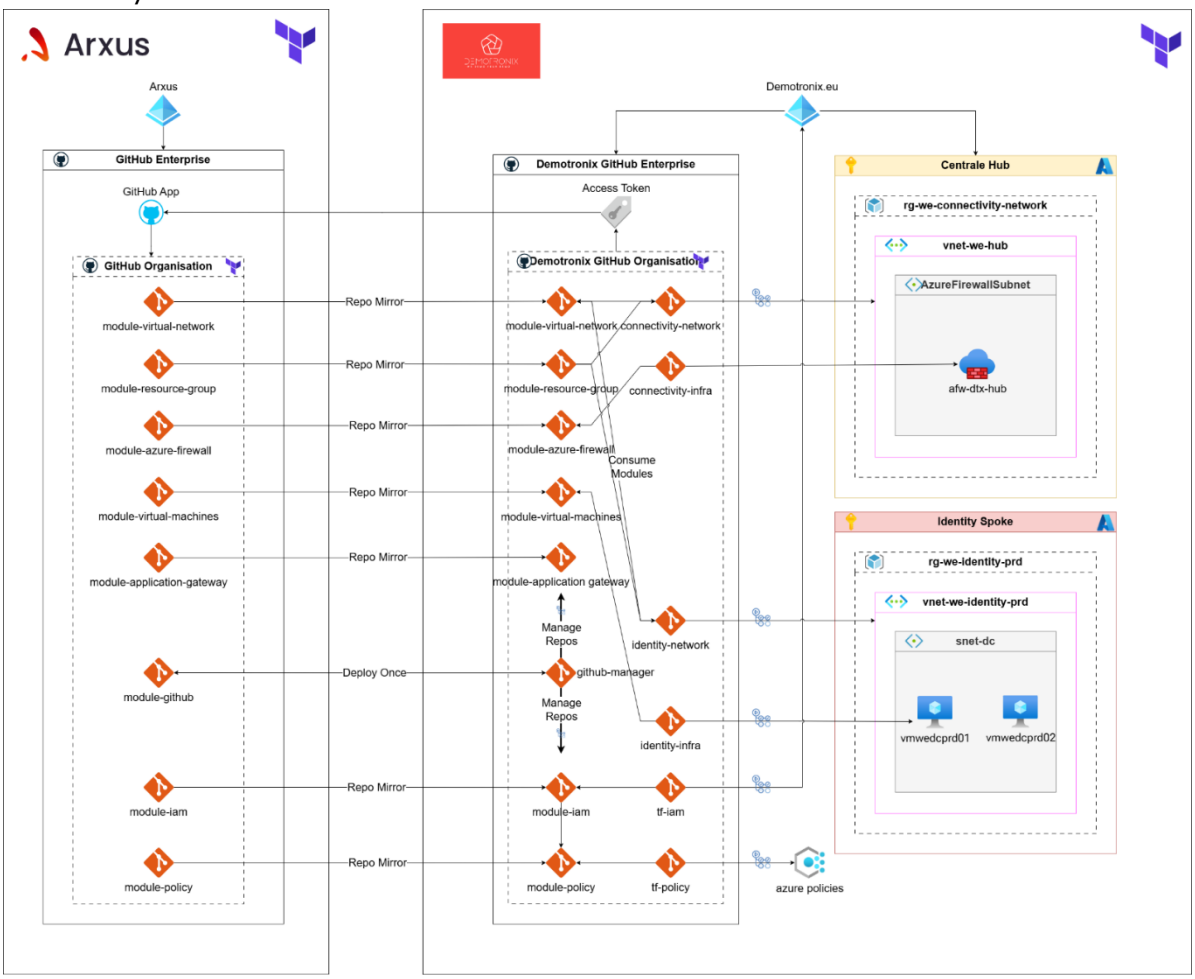
5.1. Module Sync & Module Deployment

De originele versie van deze library bevindt zich in de private GitHub-repository van Arxus. De GitHub-beheermodule, die verantwoordelijk is voor het beheren van de klantcode (Demotronix), wordt handmatig toegevoegd aan de GitHub-organisatie van de klant. Deze module initialiseert de nodige lege repositories en implementeert het beheer van GitHub-structuren, zoals approval rules, approval groups en rollen.

Vervolgens kan je in bijlage 2 zien dat de modules van Arxus worden gemirrored naar de klantomgeving, waarbij repositories met dezelfde naam worden gesynchroniseerd. Op die manier wordt de inhoud van de modules bij Arxus consistent gespiegeld in de klantomgeving.

Daarna kunnen deployment-repositories worden aangemaakt. De eerder gesynchroniseerde modules kunnen vervolgens in deze deployment-repositories worden geïmporteerd, zodat services kunnen worden uitgerold, zoals firewalls, domain controllers en virtual networks.

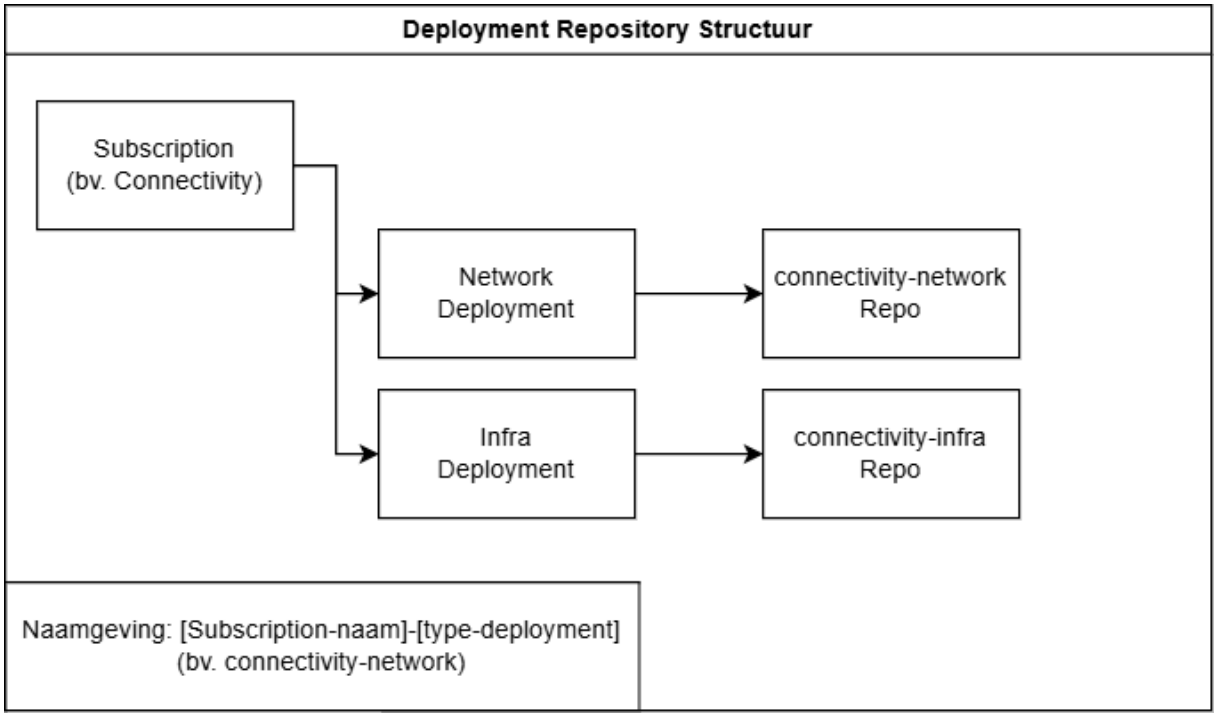
Er wordt wekelijks een module-synchronisatie uitgevoerd. Wanneer Arxus een Terraform-module wijzigt, worden deze aanpassingen automatisch doorgevoerd in de klantomgeving via deze synchronisatie.



Bijlage 2: Terraform Module Library & Module Sync Design Diagram

(Zie Bijlage 2 op pagina 57 voor een uitvergrote versie van "Terraform Module Library & Module Sync Design Diagram")

Alle interacties met GitHub en Azure verlopen via GitHub Actions workflows. Zowel repository-mirroring, repositorybeheer als Azure-deployments worden volledig geautomatiseerd uitgevoerd en nooit handmatig.



Bijlage 3: Deployment Repository Structuur diagram

(Zie *Bijlage 3* op pagina 58 voor een uitvergrote versie van "Deployment Repository Structuur Diagram")

Het maken van deze repositories met de GitHub Manager module pipeline gebeurt via een repository template. Voor network en infra deployment repositories zijn er aparte templates. Gebaseerd op de template zijn de nodige standaard modules alvast geïmporteerd in de repository (bv. netwerk repo heeft Virtual Network module). Wanneer deze repository bestaat, heeft het al de juiste mappenstructuur en GitHub Actions Calling Workflow files.

5.2. Import modules van Library

Wanneer een deployment repository kant en klaar staat, moet eerste de nodige modules geïmporteerd worden in de repo om deze te kunnen gebruiken (als alleen de standaard modules niet voldoende zijn). Dit is door in de repo een Terraform file te maken met de naam van de module en import module code te pasten hierin, die te vinden is in de README.md van elke module in de library.

```
# Create Azure Firewall
module "azure_firewall" {
  for_each = { for k, v in var.azure_firewalls : k => v }
  source   = "git::https://github.com/[org]/[module].git"

  prefix                              = try(each.value.prefix, var.prefixes.firewall)
  public_ip_prefix                    = try(each.value.public_ip_prefix, var.prefixes.public_ip_address)
  firewall_policy_prefix              = try(each.value.firewall_policy_prefix, var.prefixes.firewall_policy)
  delimiter                           = try(each.value.delimiter, var.delimiter)
  …
}
```

Bijlage 4: Terraform import module into deployment repo code snippet
(Code snippet van module import in klant repo, voor beter kwaliteit, ga naar Bijlage 4 op pagina 59)

Hierna staat alles klaar om de resources te maken en configureren. Alle configuratie in een deployment repository bij een klant gebeurt in de Terraform Variables (.tfvars) file. Voor iedere omgeving (dev, tst, acc, stg, prd), is er een aparte tfvars file. Hierin staat beschreven in code wat er allemaal wordt deployed, zoals bijvoorbeeld in bijlage 5. De GitHub Actions workflows en Terraform modules zijn zo gemaakt dat aan de module zelf niks moet aangepast worden om resources te deployen in Azure. Alleen maar de Terraform Variables file zou moeten geconfigureerd worden. De modules zijn zo gemaakt dat je van hetzelfde type resource, meerdere resources kunt maken in een variable file. Hierdoor is het mogelijk om in een repo en een variable file volledige deployment te maken. Zoals 4 Resource Groups, 5 Virtual Networks, 5 DNS Zones, etc.

```
subscription_id     = "[full-subscription-id]"
hub_subscription_id = "[hub-subscription-id]"
usecase_name        = "connectivity-network"
environment         = "prd"
location            = "westeurope"
network_state_file  = null

resource_groups = {
  rg-we-connectivity-network = {
    prefix               = "rg" # Optional: Defaults to var.prefixes.resource_group
    delimiter            = "-"  # Optional: Change the delimiter. Defaults to var.prefixes
    …
    }
  }

  rg-we-connectivity-bastion = {
    ...
    }
  }
  ...
}
```

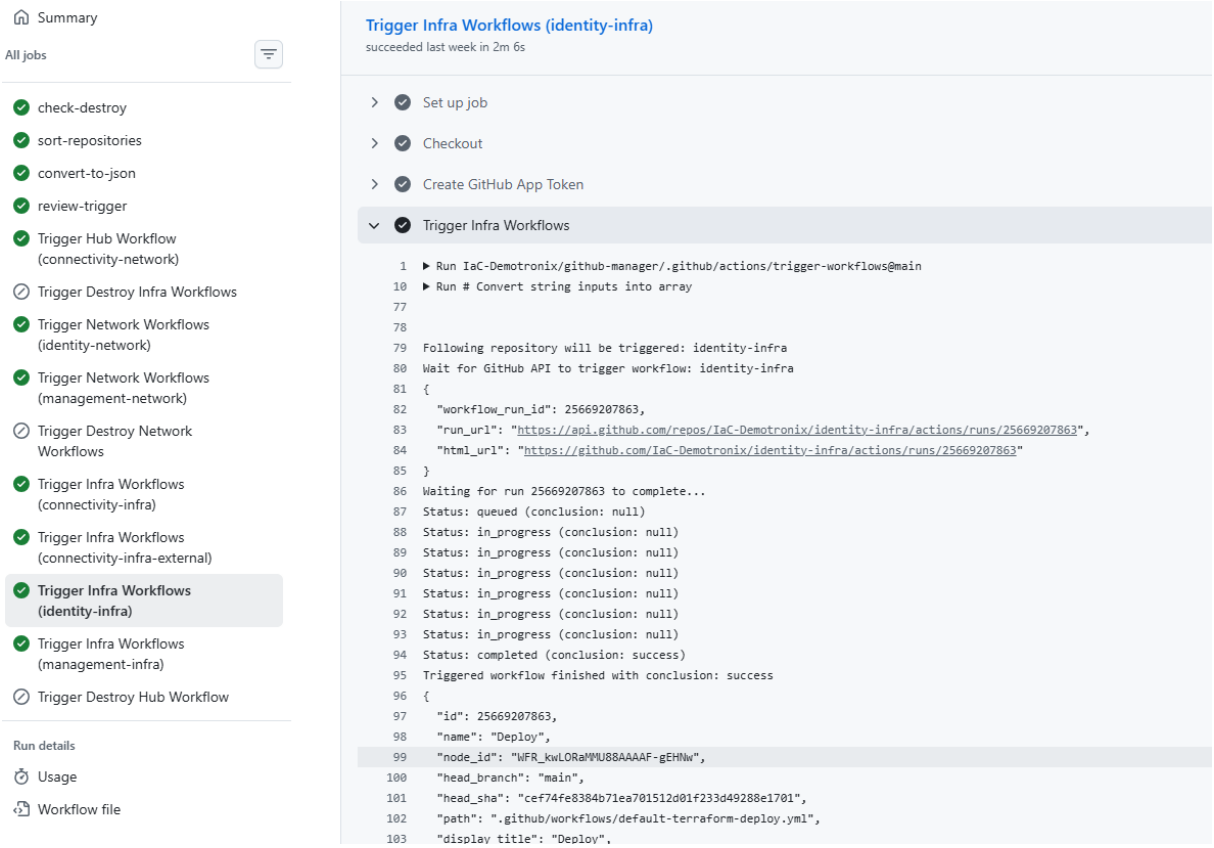
Bijlage 5: Terraform import module into deployment repo code snippet
(Code snippet van een resource creation in variables file, voor beter kwaliteit, ga naar Bijlage 5 op pagina 60)

5.3. Integratie Terraform met GitHub Actions

Elke keer dat er een wijziging wordt gemaakt aan een Terraform file, zoals de variable file, en deze wordt gepushed naar GitHub, wordt er bij het aanmaken van een pull-request (PR) automatisch een Terraform Plan uitgevoerd. Zo kan er gecontroleerd worden of de geplande wijzigingen correct zijn, werken zoals verwacht en geen conflicten veroorzaken voordat deze changes effectief worden uitgerold naar de hoofdbranch. Bijlage 19 is een voorbeeld van een Terraform Plan resultaat. Zodra changes mergen met de hoofdbranch, dan pas wordt de werkelijke infrastructuur opgezet.

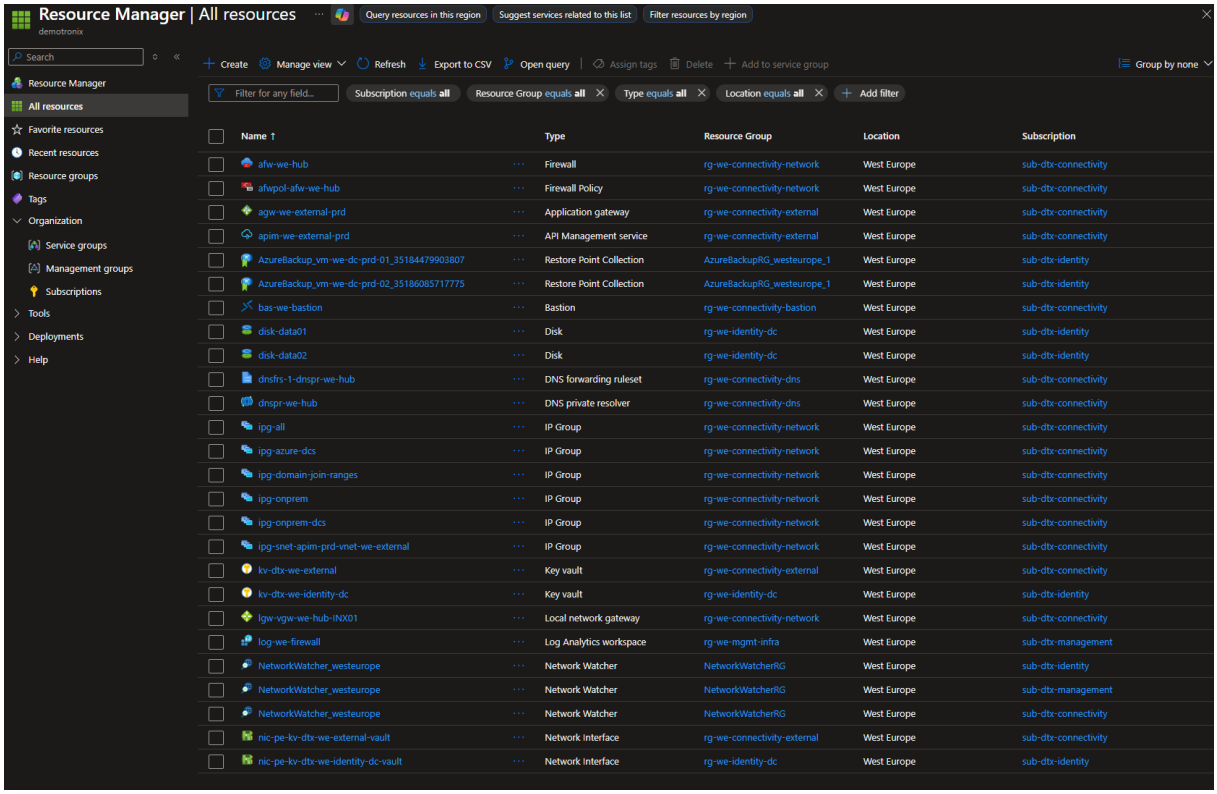
Bijlage 19: Terraform Build Plan
(Zie *Bijlage 19* op pagina 74 voor een uitvergrote versie van "Terraform Build Plan")

Om te vermijden dat voor iedere repository er een workflow moet gestart worden om alles te kunnen deployen, wordt de Orchestrator Workflow gebruikt dat alle repositories in de juiste vollegorde opstart. Bijlage 20 is een voorbeeld van een Orchestrator workflow deployment (meer hierover in hoofdstuk 7).



Bijlage 20: Orchestrator Workflow Deployment
(Zie *Bijlage 20* op pagina 75 voor een uitvergrote versie van "Orchestrator Workflow Deployment")

Op deze manier heb ik voor iedere repository in elke subscriptie de variables file zo aangepast om uiteindelijk de volledige omgeving te kunnen opzetten, zoals te zien in bijlage 21.



Bijlage 21: Full Azure Deployment
(Zie *Bijlage 21* op pagina 76 voor een uitvergrote versie van "Full Azure Deployment")

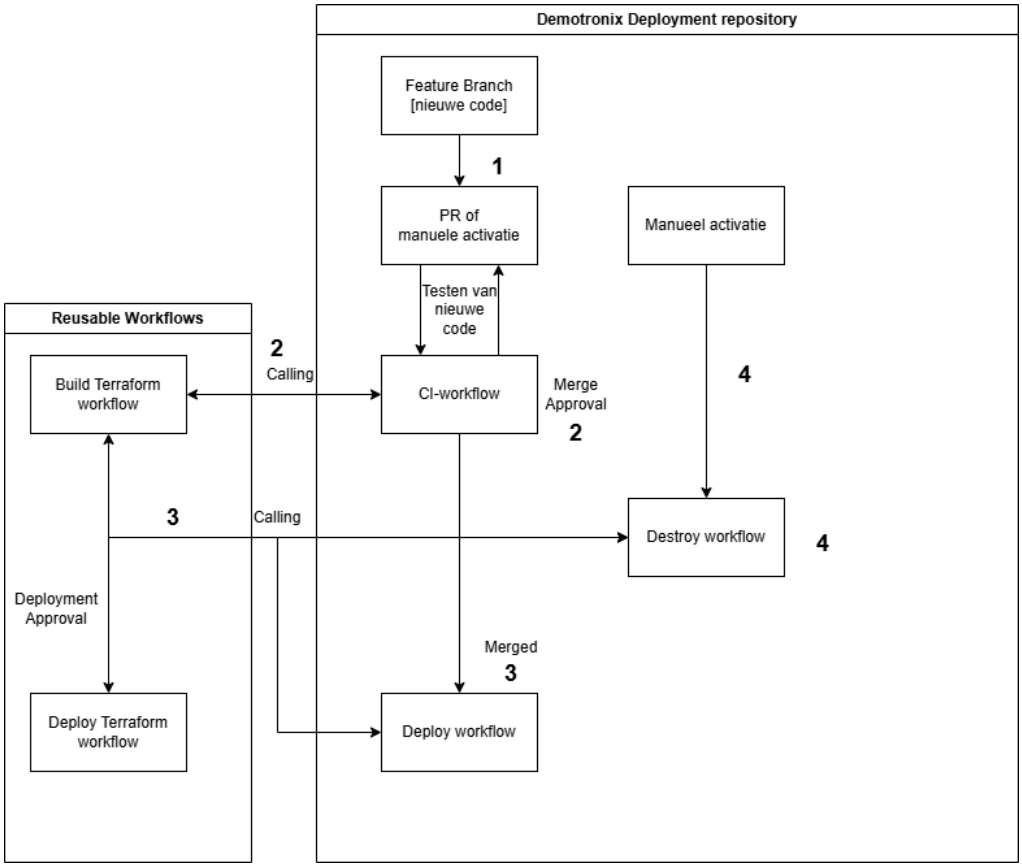
Dus concreet samengevat, De taak van Terraform is om alle manuele taken en inputs te automatiseren om human error te vermijden en terwijl optimalisatie en standaardisatie te garanderen. Terraform bouwt, breekt af en verandert infrastructuur en GitHub configuratie. Natuurlijk start Terraform niet vanzelf. Om ervoor te zorgen dat deze commando's ook niet manueel ingevoerd moeten worden en ook automatisch kunnen uitgevoerd worden bij iedere verandering aan de Terraform code in de code repositories, wordt er gebruikgemaakt van GitHub Actions Workflows.

6. GitHub Actions Workflows

De GitHub Actions workflows zijn de CI/CD pipelines die in elke deployment repository zit om automatisch een infrastructuurplan te maken via Terraform en na de nodige goedkeuring automatisch alle resources uit dit plan te deployen in een Azure omgeving.

Het doel hiervan is niet om deze workflows van nul te maken, maar deze correct te gebruiken om het Demotronix omgeving op te zetten.

Deze workflows sluiten aan bij de CI/CD werkwijzen die reeds binnen Arxus worden toegepast. Daarom zijn ze modulair opgebouwd met behulp van reusable en calling workflows. Deze aanpak zorgt voor een duidelijke en consistente structuur, verhoogt de herbruikbaarheid van bestaande workflows en optimaliseert de bestaande code voor redundantie te vermijden. Daarnaast vereenvoudigt dit het beheer en onderhoud van de workflows, wat de efficiëntie en schaalbaarheid ten goede komt.
(zie Glossary)



Bijlage 13: Deployment Workflow Structuur
(Zie Bijlage 13 op pagina 68 voor een uitvergrote versie van "Deployment Workflow Structuur")

Binnen Demotronix, heeft iedere deployment repo de volgende calling workflows:

- **CI-workflow**
- **Deploy workflow**
- **Destroy workflow**

Op hun beurt, spreken ze een van of beide reusable workflows aan:

- **Build Terraform workflow**
- **Deploy Terraform workflow**

Het geautomatiseerd infrastructuur deployment proces start altijd bij een **calling workflow**.

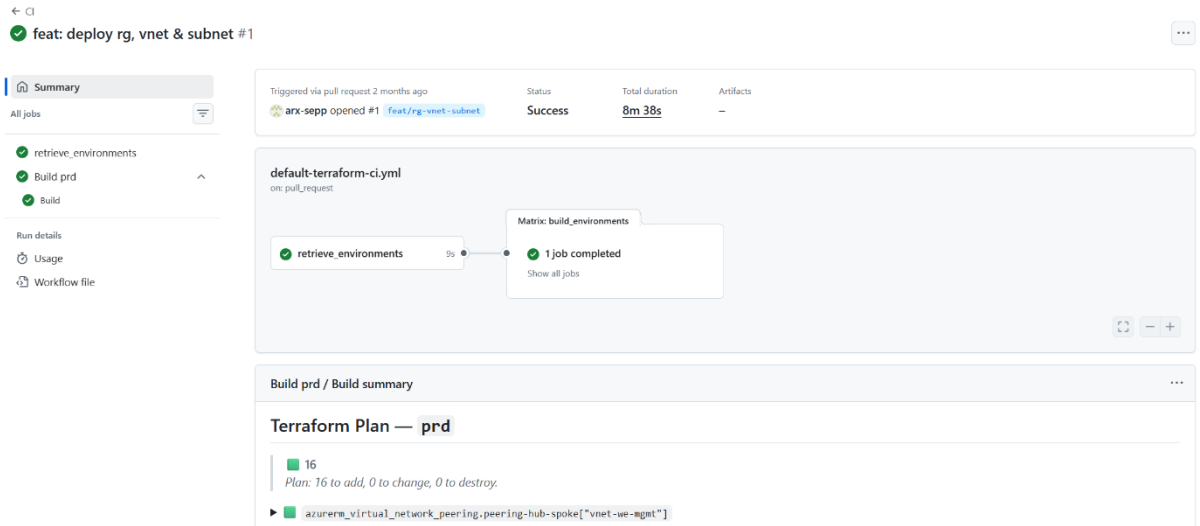
6.1. Calling Workflows

Calling workflows zijn normale workflows die tijdens een bepaalde stap in hun workflow een reusable workflow kunnen oproepen. Deze workflows zijn meestal het startpunt waaraan een serie van reusable workflows aan gekoppeld worden om zo grote en lange taken uit te voeren. Bij Demotronix, zijn er 3 standaard calling workflows:

6.1.1. CI-workflow

Deze workflow wordt gestart met elke PR dat er wordt gemaakt (stap 2 in bijlage 13). De taak van deze workflow is om een Terraform Plan te maken me de nieuwe code die naar main wilt mergen. Met dit plan wordt er ook getest op syntax en configuratie fouten, als ze er zijn. Bovenop doe het ook auto-formatting van de code en checkt in welke environment het de code in moet testen. De workflow dat echt het plan maakt, is de build reusable workflow dat de CI-workflow oproept.

Een branch kan alleen mergen met main als de CI-pipeline de code heeft getest en een succesvol plan heeft kunnen maken. Waarna er een andere developer dit plan moet nakijken en approven. Alleen dan pas kan de nieuwe code mergen met main en de nieuwe infrastructuur gebouwd worden. Bijlage 24 toont een CI-workflow dat een reusable workflow gebruikt heeft bij het opmaken van een plan, namelijk de Build workflow.



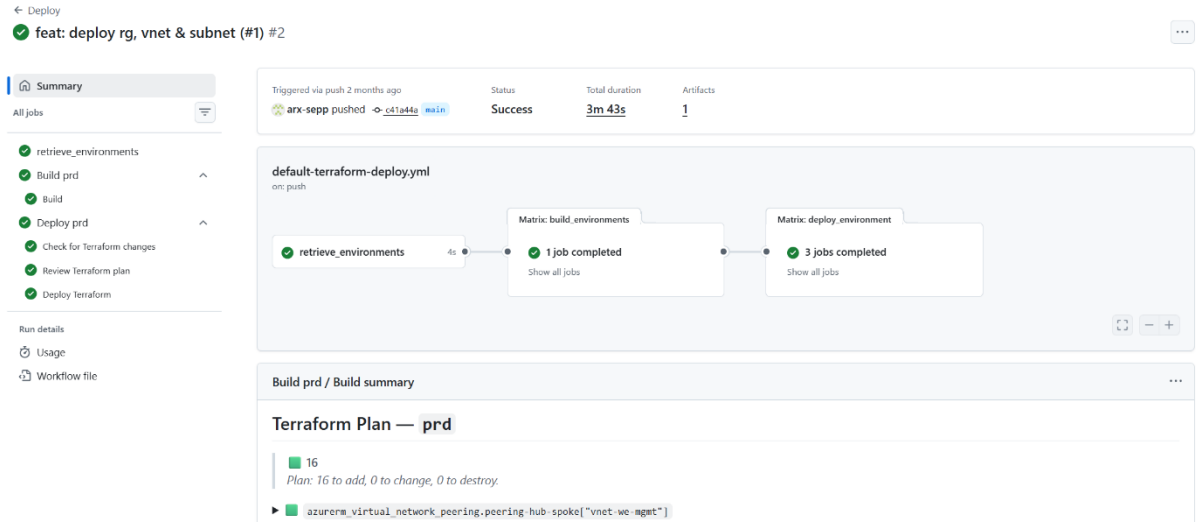
Bijlage 22: CI-Workflow
(Zie *Bijlage 22* op pagina 78 voor een uitvergrote versie van "CI-Workflow")

6.1.2. Deploy workflow

Nadat een PR approved is en nieuwe code kan merged met main, start deze workflow (stap 3 in bijlage 13). Het is de taak van deze workflow om de nieuwe infrastructuur te deployen volgens het infrastructuurplan. De workflow dat de werkelijke deployment uitvoert, is het deploy reusable workflow dat de deze workflow oproept.

Een groot deel van de structuur van deze workflow is hetzelfde, van auto-formatting tot het approven van de deployment door een collega. Het is na de approval, dat het plan van de vorige stap gebruikt om de nieuwe infrastructuur te deployen.

Dit kan nieuwe resources maken, oude afzetten en vervangen met een nieuwe, kapot maken, updaten of aanpassen. Bijlage 23 toont een Deploy workflow dat 2 reusable workflows gebruikt bij het opbouwen van infrastructuur. Namelijk de Build workflow en de Deploy Terraform workflow.



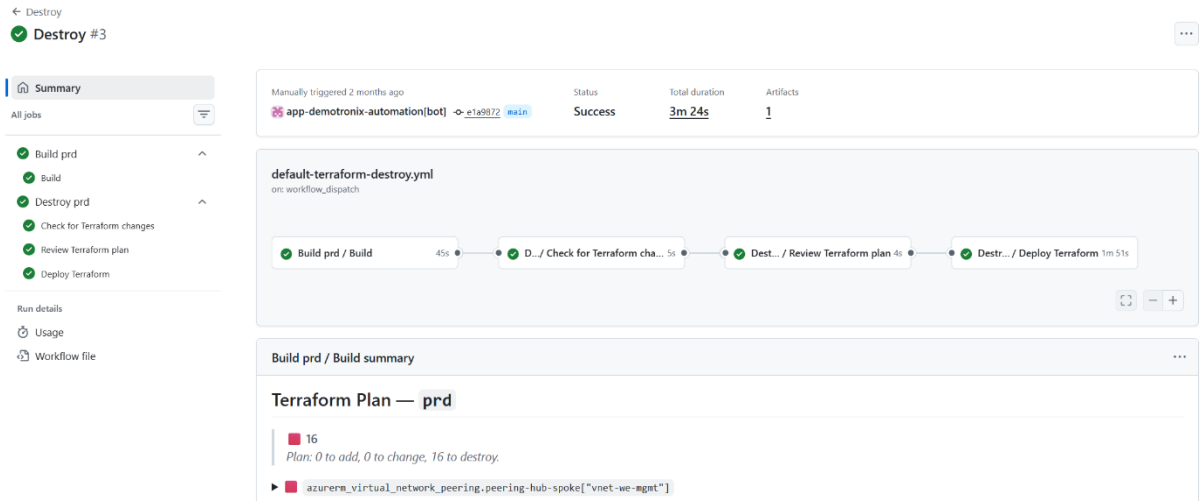
Bijlage 23: Deploy Workflow
(Zie *Bijlage 23* op pagina 78 voor een uitvergrote versie van "Deploy Workflow")

6.1.3. Destroy workflow

Het afzetten van infrastructuur kan alleen manueel gestart worden in de Actions tab van de deployment repo naar keuze of via de Orchestrator workflow (stap 4 in bijlage 13). De taak van deze workflow is om de bestaande infrastructuur terug af te zetten. De infra dat wordt afgezet is gebaseerd op het infrastructuurplan dat Terraform maakt.

Deze workflow is bijna identiek aan de deploy workflow. Het grote verschil ligt aan hoe deze start. Dit kan alleen manueel gestart worden, waarbij je de omgeving moet ingeven en in een aparte input vak terug de omgeving als veiligheidsvoorziening. Boven op deze veiligheidsvoorziening, is er ook een approval dat moet goedgekeurd worden voordat er effectief infrastructuur wordt afgezet.

Bijlage 24 toont een Destroy workflow dat 2 reusable workflows gebruikt. Namelijk de Build workflow en de Deploy Terraform workflow. Dit is geen fout, Deploy Terraform workflow wordt gebruikt bij zowel het opzetten als het afzetten. Deze workflow volbrengt beide taken.



Bijlage 24: Destroy Workflow
(Zie *Bijlage 24* op pagina 79 voor een uitvergrote versie van "Destroy Workflow")

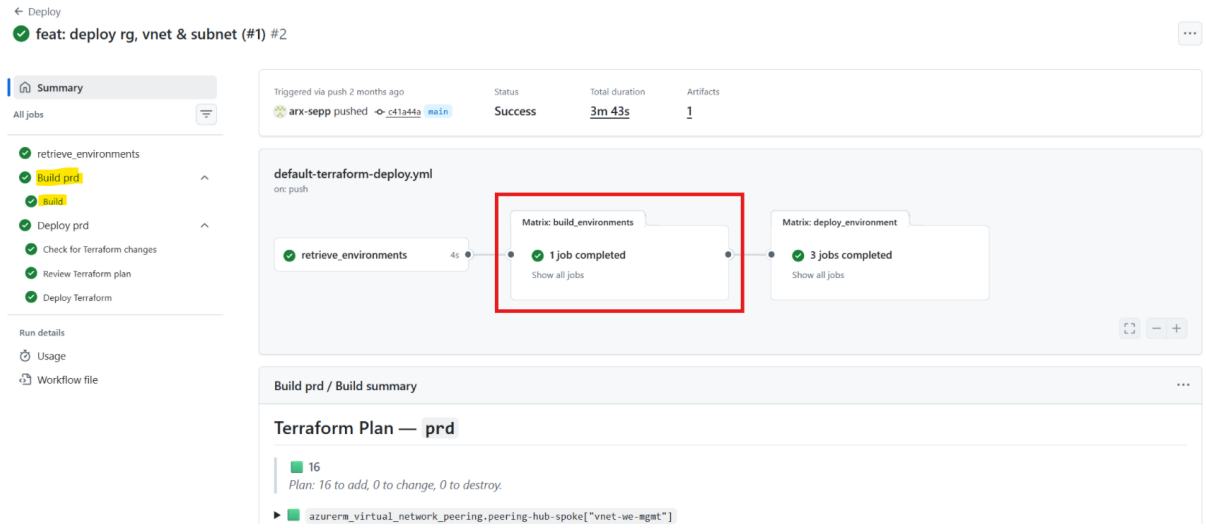
6.2. Reusable Workflows

Een belangrijk onderdeel van de CI/CD workflow structuur bij Arxus, zijn de reusable workflows. Dit zijn herbruikbare workflows die kunnen aangeroepen worden in elke calling workflows zonder dat je code moet copy & pasten van bestaande workflows. Dit vermijdt redundante code en maakt de workflows meer modulair. Alle reusable workflows zitten in hun eigen repository, die manueel bij de klant gezet wordt, waarna deze worden opgeroepen van elke deployment repository bij de klant. Bij Demotronix zijn er 2 reusable workflows:

6.2.1. Build Terraform Workflow

Dit is de workflow verantwoordelijk voor het maken van een infrastructuur deploy of destroy plan en het formatteren van de Terraform code.

In bijlage 25 zie je een Deploy Workflow dat een Build reusable workflow gebruikt bij het maken van een infrastructuurplan voor een netwerk uitrol.

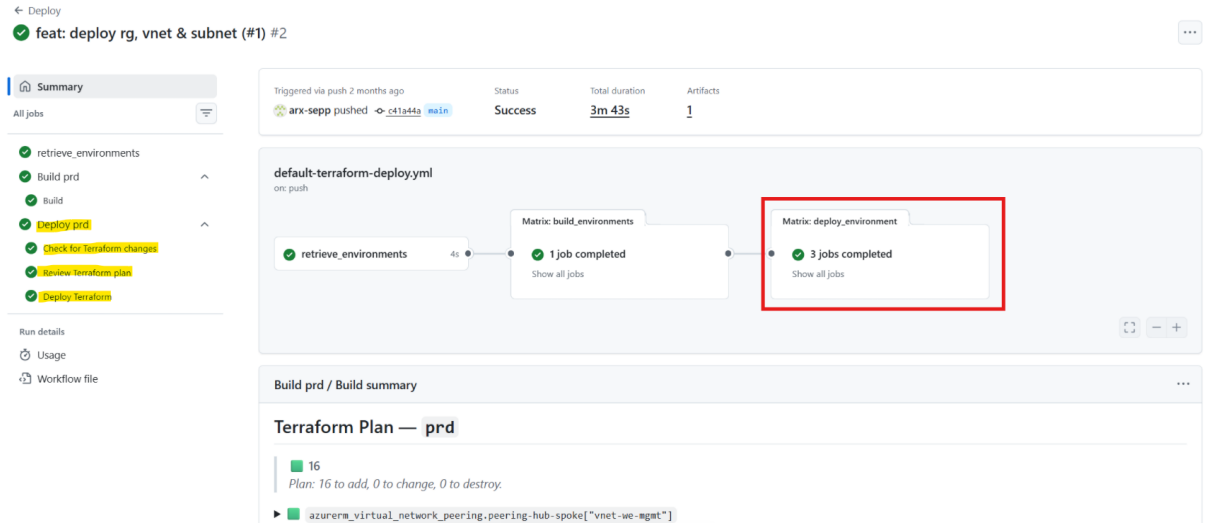


Bijlage 25: Reusable Workflow - Build
(Zie *Bijlage 25* op pagina 80 voor een uitvergrote versie van "Reusable Workflow - Build")

6.2.2. Deploy Terraform Workflow

Dit is de workflow verantwoordelijk voor het opzetten of afzetten van infrastructuur. Hoewel iedere workflow deze kan oproepen, is het een dependency van de Build workflow. Enkel als het een geldig plan heeft gemaakt en dit approved is door een developer, kan deze pas starten.

Bijlage 26 toont een Deploy Workflow dat na het gebruiken van een Build reusable workflow voor het maken van een infrastructuurplan, de Deploy Terraform reusable workflow wordt gebruikt bij het opzetten van het netwerkinfrastructuur. Indien de Destroy Workflow opgestart werd, zou deze reusable workflow de nodige parameters gekregen hebben om deze netwerkinfrastructuur juist af te breken.



Bijlage 26: Reusable Workflow – Deploy/Destroy
(Zie *Bijlage 26* op pagina 81 voor een uitvergrote versie van "Reusable Workflow – Deploy/Destroy")

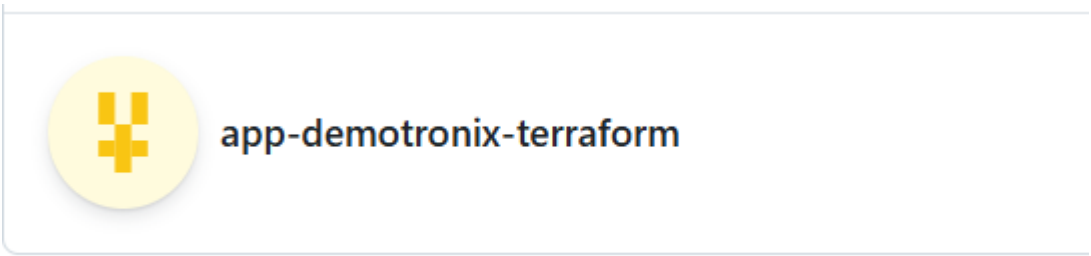
6.3. Authenticatie

Natuurlijk is authenticatie een zeer belangrijk onderdeel van de workflow. Want niet alles en iedereen mag zomaar verbinden met de Azure tenant. Er zijn 2 kritieke onderdelen die moeten bestaan en verbonden zijn aan de juiste deployment repository vooraleer connectie met Azure mogelijk is:

- GitHub App
- Azure Service Principle

6.3.1. GitHub App

Dit is een functionaliteit binnen GitHub dat verschillende taken automatisch uitvoeren onder de rol van een GitHub gebruiker. Tegenover gelijkaardige functionaliteiten, hebben GitHub Apps granulaire permissies, zijn instaleerbaar op repository of organisation niveau en authentiseren als zichzelf wanneer ze willen authentiseren met Azure. Bijlage 27 toont een voorbeeld GitHub App binnen Demotronix.



Bijlage 27: GitHub App
(Zie *Bijlage 27* op pagina 82 voor een uitvergrote versie van "GitHub App")

Het grote pluspunt aan GitHub Apps, is dat dankzij granulaire permissies, deze Apps toegang kan geconfigureerd worden zodat ze alleen maar toegang krijgen tot wat ze nodig hebben om hun taak uit te voeren en niet meer. Dit allemaal volgens de Least Privilege methode. Bovendien kan iedereen dat een pipeline runt, deze user kunnen vragen voor taken uit te voeren.

Hoewel deze GitHub Apps vooraf al gemaakt zijn, wordt er bij elke workflow run een token aangevraagd om deze App even te gebruiken. Nadat deze run voltooid is, wordt de token weggegooid.

Kort gezegd, GitHub bots kunnen doen alsof ze een GitHub gebruiker zijn op repository of organisatie niveau binnen GitHub om geautomatiseerde taken uit te voeren.

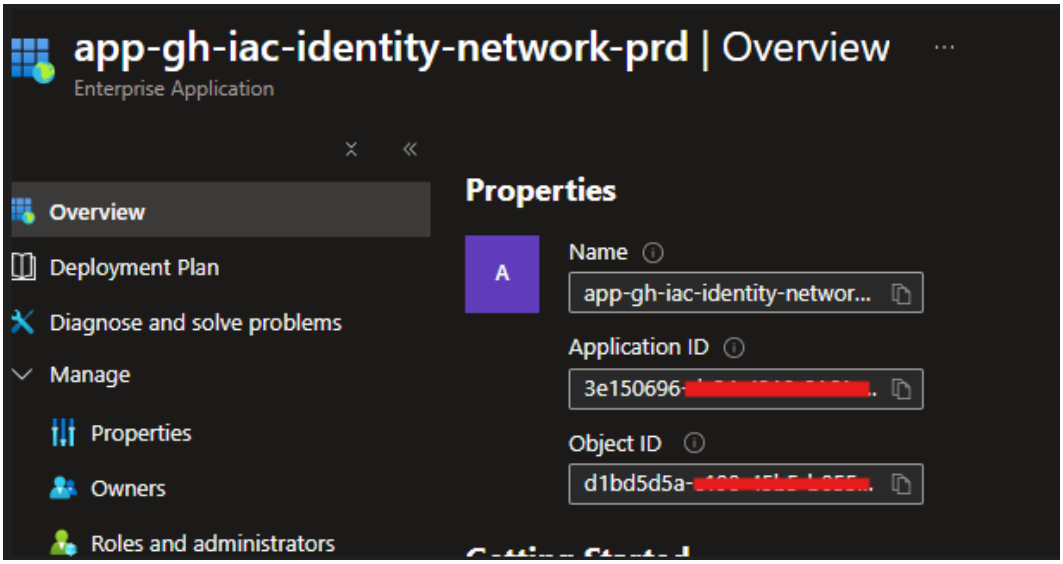
Binnen Demotronix zijn er 2 GitHub Apps:

- **Automation**: Dit is voor aanpassingen te maken aan de GitHub Organisation zelf
  - Nieuwe repositories maken
  - Nieuwe GitHub rollen
  - Module Sync
  - …
- **Terraform**: Dit is voor aanpassingen te maken aan Azure tenants met Terraform
  - Nieuwe infrastructuur deployen in Azure
  - Azure policies aanpassen in Azure
  - …

Maar authenticatie moet van twee kanten komen, dus hoe werkt dit in Azure?

6.3.2. App Registrations & Service Principles

App Registrations is een connectie tussen external OpenID Connect (OIDC) identity providers, zoals GitHub, en Azure Entra. Binnen een App Registration kunnen alle login gegevens opgeslagen worden zodat externe apps zich kunnen voordoen als een gebruiker in de Azure tenant. Via Federated Credential tokens, kan bijvoorbeeld een GitHub App toegang krijgen tot Azure en veranderingen maken aan de Azure tenant waarvoor het toegang heeft.



Bijlage 28: App Registration
(Zie *Bijlage 28* op pagina 83 voor een uitvergrote versie van "App Registration")

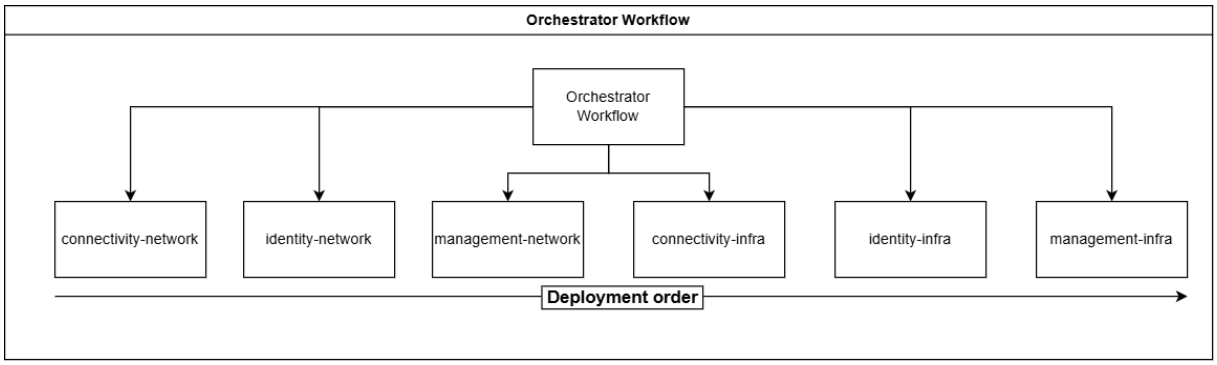
Terwijl de App Registration beschrijft wat het is en hoe externe apps zich moeten aanmelden bij Azure, is een Service Principle de identiteit van de externe app binnen Azure. Dit is het identiteit waar de GitHub App tot inlogt om acties uit te voeren op de Azure Tenant. Permissies en rollen worden gezet op deze Service Principle zodat het alleen maar acties mag doen dat zijn job is om te doen.

Binnenin Demotronix, zijn er voor iedere deployment repository een App Registration met een bijbehorende Service Principle. Bijlage 28 toont een App Registration voor de deployment repository "identity-network". De login gegevens van de App Registration worden in de GitHub Secrets Environment gezet van deze deployment repo, zodat de GitHub App kan authentiseren tot Azure via OIDC. Iedere App Registration is ingesteld en heeft alleen de rollen en bevoegdheid tot de subscriptie en taken die ze daarin moeten uitvoeren.

Een van de enige nadelen aan deze workflows en manier van werken, is dat bij regelmatige deployment en destroy van een omgeving, je constant alle verschillende workflows moet aanspreken en draaien. Een grote oplossing hiervoor zou zijn een centraal overheersende pipeline dat alle nodige deployment repository workflows start in de juiste vollegorde, de oplossing hiervoor is de Orchestrator workflow.

7. Orchestrator Workflow

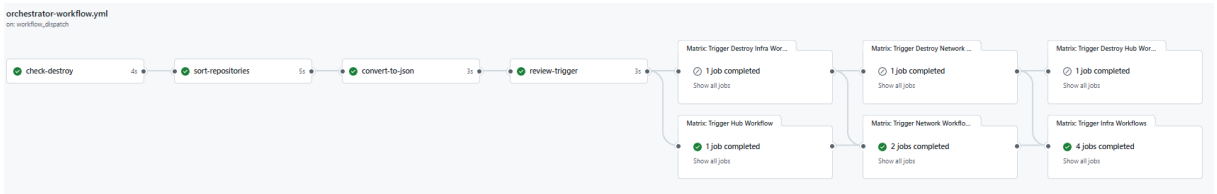
De Orchestrator workflow is een GitHub Actions workflow dat enkele onderdelen naar keuze of alle Demotronix infrastructuur kan opzetten oftewel kan afzetten. Deze workflow zou alle manueel werk moeten vervangen dat neemt om in elke deployment repository een deploy workflow manueel te starten. Werkend voor zowel deployments als destroys kan dit alle workflows orkestreren.



Bijlage 29: Orchestrator Workflow Diagram
(Zie *Bijlage 29* op pagina 84 voor een uitvergrote versie van "Orchestrator Workflow Diagram")

Deze workflow functioneert door API-calls te maken naar de gewenste repositories via de GitHub API om de deploy of destroy workflows te activeren. Hoewel dit misschien simpel lijkt op eerste zicht, is het allesbehalve.

Het grote probleem is de repository volgorde. Want een deployment kan niet zomaar in een willekeurige volgorde plaatsvinden. Zoals te zien is in bijlage 29, zijn sommige repositories afhankelijk van andere repositories. Bovendien moet de volgorde bij een destroy actie in de omgekeerde volgorde zijn van hoe deze in bijlage 29 staat. In combinatie met het feit dat al deze repositories handmatig moesten worden opgestart, wilde ik een gestroomlijnde oplossing ontwikkelen. Mijn oplossing hiervoor was de 'sort-repositories'-action



Bijlage 20: Orchestrator Workflow - Proces
*(*Zie *Bijlage 30* op pagina 85 voor een uitvergrote versie van "Orchestrator Workflow - Proces")

7.1. Sort Repositories Action

Een Action workflow binnen GitHub is een type workflow dat herbruikbaar, modulair en specifiek gemaakt is voor kleine specifieke taken uit te voeren binnenin een normale workflow. Voor individuele taken die herbruikbaar kunnen zijn in toekomstige workflows, worden vaak Actions van gemaakt.

De sort-repositories Action doet exact wat de naam zegt, het sorteer alle gekozen repositories die zijn ingevoerd in de Orchestrator Workflow. Het sorteren is gebaseerd op:

1. Prioriteit
2. Alfabetisch op naam
3. Deploy of Destroy actie

7.1.1. Prioriteit

```
        ...
            priority() {
                local name="$1"

                if [[ "$name" == "connectivity-network" ]]; then
                    echo 0
                elif [[ "$name" == *network* ]]; then
                    echo 1
                elif [[ "$name" == *infra* ]]; then
                    echo 2
                else echo 3
                fi
            }

            # sort repos based on priority, then alphabetical order.
Connectivity-network is always first!
            sorted=$(printf "%s\n" "${repos[@]}" \
                | while IFS= read -r repo; do
                    printf "%s\t%s\n" "$(priority "$repo")" "$repo"
                done \
                | sort -n -k1,1 -k2,2 \
                | cut -f2
            )
        ...
```

Bijlage 14: Prioriteiten toewijzen en sorteren op alfabet
*(*Zie *Bijlage 14* op pagina 69 voor een uitvergrote versie van "Prioriteiten toewijzen en sorteren op alfabet")

In bijlage 14 ziet u hoe de Action eerst elke repository doorloopt en op basis van de naam van de repository bepaalt welke prioriteit deze krijgt. Hoe lager het cijfer, des te hoger de prioriteit. Een hoge prioriteit betekend dat die repo eerst in de lijst staat om te deployen en als laatste om te destroyen. Een prioriteit wordt gegeven gebaseerd op het type repository dat het is. Hier is een tabel om het duidelijker te visualiseren:

Naam	Prioriteit
Gelijk is aan "connectivity-network"	0
Network repositories	1
Infra repositories	2
[Geen Demotronix repo]	3

Afhankelijk van de naam, krijgt het een hogere of lagere prioriteit. Bijvoorbeeld "connectivity-network" wordt altijd als eerste deployed. Dit komt omdat elke andere network deployment repo deze als dependency heeft. Hierna is het elke network deployment repo voor Demotronix. Dit is omdat elke infra deployment repo deze repos als dependency heeft, want er kan geen infra deployed worden zonder netwerk. Waarna infra volgt. Indien er repositories zijn opgegeven dat geen Demotronix deployment repositories zijn, worden deze pas op het einde deployed.

7.1.2. Alfabetisch op naam

Nadat een prioriteit is gegeven, sorteert het verder alfabetisch op naam. Bijvoorbeeld tussen "identity-network" en "management-network", hoewel ze dezelfde prioriteit hebben (1), wordt "identity-network" eerder deployed dan "management-network". Omdat deze eerder voorkomt in het alfabet. Deze logica is te zien in bijlage 14.

7.1.3. Deploy of Destroy actie

```
...
        # Reverse order when performing destroy
        if [[ "${{ inputs.destroy }}" == "true" ]]; then
            sorted=$(printf "%s\n" "$sorted" | tac)

            echo "Repo Count: ${#sorted[@]}"
            echo "Repository List:"
            echo "$sorted"
        else
            echo "Repo Count: ${#sorted[@]}"
            echo "Repository List:"
            echo "$sorted"
        fi
...
```

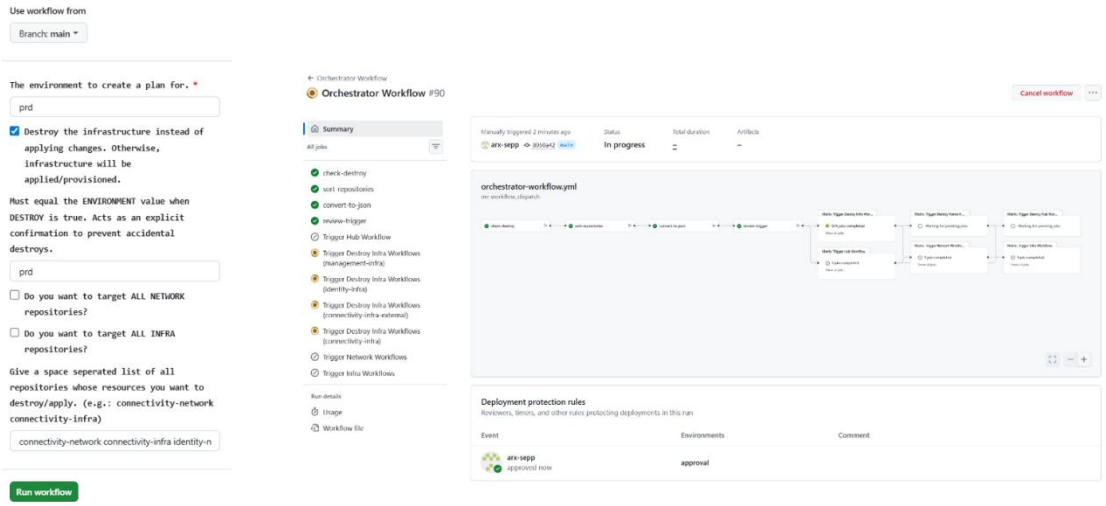
Bijlage 15: Deploy/Destroy Logica
(Zie *Bijlage 15* op pagina 70 voor een uitvergrote versie van "Deploy/Destroy Logica")

Ten slotte, nadat het gesorteerd heeft op naam en alfabet, wordt er gecheckt of de Orchestrator workflow als doel heeft om infrastructuur te deployen of destroyen. In bijlage 15 ziet u dat, wanneer het deployen is, zal er niks veranderen en zijn de repositories klaar met sorteren.

Indien het destroyen is, wordt de definitieve reeks in de omgekeerde volgorde gezet. Net zoals Terraform bij een destroy, resources afgezet worden in de omgekeerde versie van een Terraform apply, moet de vollegorde dat we de destroy workflows starten bij alle repositories dan ook in de omgekeerde vollegorde moeten zijn dat ze applied zijn.

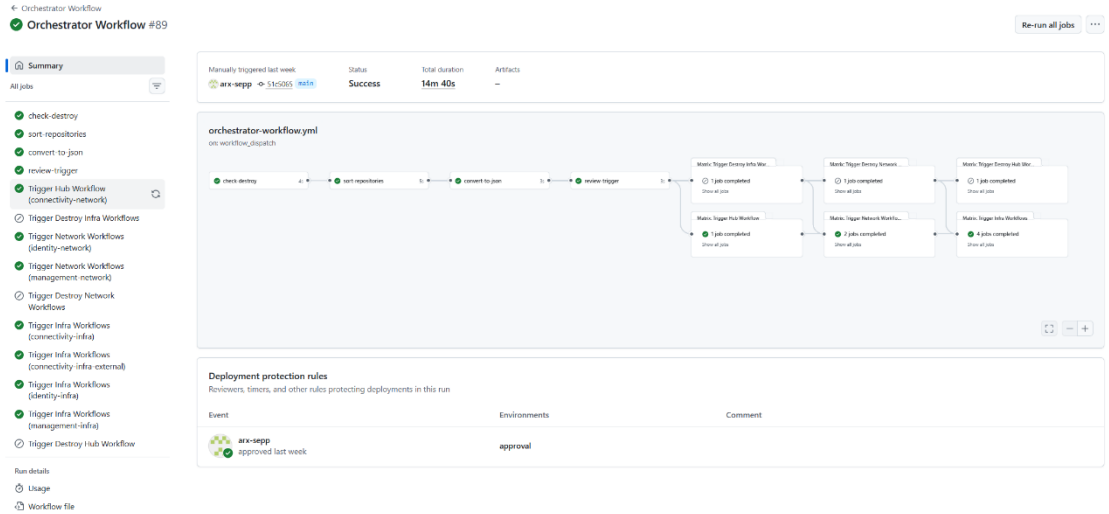
7.2. Demonstratie & Validatie

In dit hoofdstuk wordt kort de werking van de Orchestrator workflow aangetoond aan de hand van bewijsmateriaal uit GitHub Actions. De uitvoering start in de GitHub-Manager repository, waar de Orchestrator Workflow staat. Deze workflow wordt manueel geactiveerd, zoals weergegeven in de linkerhelft van bijlage 31. Tijdens deze activatie kan worden gekozen of de Azure-omgeving opgebouwd of afgebroken moet worden, evenals welke specifieke onderdelen hierbij betrokken zijn. Op basis van deze selectie zal de Orchestrator workflow enkel de relevante onderliggende workflows triggeren, zoals te zien in de rechterhelft van bijlage 31.



Bijlage 31: Orchestrator Workflow – Manuele Dispatch + Destroy Workflow
(Zie *Bijlage 31* op pagina 86 voor een uitvergrote versie van "Orchestrator Workflow – Manuele Dispatch + Destroy Workflow")

Aan de hand van bijlage 31 en bijlage 1 wordt aangetoond dat deze aanpak zowel het geautomatiseerd opzetten als het gecontroleerd afbreken van de volledige omgeving ondersteunt.



Bijlage 1: Orchestrator Workflow – Deploy Workflow
(Zie Bijlage 1 op pagina 56 voor een uitvergrote versie van "Orchestrator Workflow – Deploy Workflow")

8. Project Review

Om het project goed af te sluiten, wil ik een duidelijk terugblik geven op de gerealiseerde resultaten en de vooropgestelde doelstellingen. Het hoofddoel van dit project was het opzetten van een herbruikbare, geautomatiseerde en gestandaardiseerde Azure Landing Zone die kan gebruikt worden voor demo's en interne doeleinden. Aan de hand van de succes criteria die zijn vastgelegd in het Project Charter, kan er worden geëvalueerd in welke mate dit hoofddoel is behaald. Deze criteria zijn het volgende:

- **Geautomatiseerde deployment van de omgeving**
- **On-demand opzetten en verwijderen van de omgeving**
- **Gebruik van gestandaardiseerde infrastructuurcode**
- **Werkende demo omgeving in de Azure demo-tenant**
- **Goedkeuring door de technische begeleiding**
- **Project oplevering binnen de deadline**

Allereerst is aangetoond dat de **Azure demo-tenant omgeving volledig geautomatiseerd kan worden opgezet**. Door het gebruik van Terraform en GitHub Actions kan de volledige Azure Landing Zone opgezet worden zonder manuele configuratiestappen of interventie. Dit zorgt voor een betrouwbare en efficiëntere deployment van de demo omgeving. Daarnaast is er voorzien van de **mogelijkheid om zowel on-demand de omgeving op te zetten als opnieuw af te breken.** Hierdoor kan werknemers snel demo omgevingen voorzien voor klanten en deze nadien weer verwijderen om kosten te besparen.

Verder is er altijd gewerkt met de Terraform module library van Arxus, waardoor **de oplossing gebruik maakt van gestandaardiseerde infrastructuur code**. Door gebruik te maken van deze module library is de code en de infrastructuur dat het opzet altijd **volgens de standaarden en richtlijn van het Microsoft CAF en de werking binnen Arxus.** Dit resulteert in een overdraagbare oplossing dat onmiddellijk in praktijk kan gebruikt worden. Bovendien werd de **oplossing succesvol uitgerold in de Azure demo-tenant, waar alle basis functionaliteit correct functioneert**, zoals netwerking, DNS, back-ups, Private Endpoints en de connectie met de Kubernetes cluster.

Op basis van dit document kan geconcludeerd worden dat de vooropgestelde doelstellingen van het project behaald zijn. De oplossing werd niet alleen technisch correct geïmplementeerd, maar heeft ook **alle goedkeuring gekregen van de betrokken stakeholders**, **zowel op technische als op architecturaal vlak.** Daarnaast werd het volledig project, inclusief documentatie en nodige goedkeuringen, **opgeleverd binnen de vastgelegde deadline van 22 Mei.** Wat bevestigt dat het project is afgerond volgens de afgesproken scope en planning.

9. Besluit

Om te starten, kan er dankzij dit document verondersteld worden dat dit project niet alleen heeft aangetoond hoe een hub-spoke Landing Zone succesvol kan worden opgezet, maar ook een duurzame basis legt voor toekomstige uitbreiding. Dankzij het modulair ontwerp van de hub-spoke Landing Zone blijft de oplossing flexibel en schaalbaar. Hierdoor kan de demo omgeving eenvoudig worden uitgebreid door Arxus collega's of toekomstige stagairs met bijkomende workloads, uitgebreidere monitoring installatie en aanvullende security componenten, zonder ingrijpende architecturale wijzigingen.

Daarnaast kan de ontwikkelde oplossing binnen de organisatie dienen als een gestandaardiseerd template en startpunt voor het opzetten van nieuwe Landing Zones bij klanten. Dit draagt bij aan een meer consistente, veilige en efficiënt uitrol van Azure omgevingen. Op die manier overstijgt dit project zijn oorspronkelijke doel en vormt het niet alleen een praktische demonstratie, maar ook een waardevolle fundering voor toekomstige Cloud projecten binnen de organisatie.

Tot slot zou ik graag mijn technische coaches bij Arxus, Brent Boutmans en Jan de Munck, willen bedanken voor alle hulp en steun doorheen mijn stageperiode bij Arxus. Zonder hen zou ik nooit zo sterk gegroeid zijn als een Cloud & Cybersecurity professional en zou dit project niet zo soepel verloopt zijn. Ook zou ik mijn stage supervisor Cas Magnus willen bedanken voor zijn steun doorheen mijn stage traject. Zijn goede feedback en inzicht hielp mij klaarstomen voor de professionele werkwereld.

Glossary

Term	Definitie
Azure demo-tenant	Het afgebakend stuk van Microsoft Azure waarin alle Clouddiensten, gebruikers en instellingen van een organisatie beheerd worden.
Azure Kubernetes Service (AKS)	Een managed service binnen Azure voor het implementeren, beheren en schalen van Kubernetes clusters. AKS vereenvoudigt het beheren van gecontaineriseerde applicaties in Azure zonder de onderliggende controle plane te onderhouden.
Azure Landing Zone	Gestructureerde en geoptimaliseerde Azure omgeving. Een omgeving ontworpen om zo veilig, flexibel en optimaal mogelijk te zijn. Kan modulair aangepast worden aan de behoeften van een organisatie dankzij het hub-spoke design.
Continuous Integration & Continuous Delivery (CI/CD)	Verschillende werkwijzen om het software ontwikkeling proces te automatiseren en te versnellen. Bij nieuwe code veranderingen in de code repository zou dit automatisch deze code testen, integreren met de bestaande code en het uitrollen van de infrastructuur. Dit laat toe om frequente veranderingen, reparaties en nieuwe implementaties snel door te voeren. Deze automatie implementatie wordt soms ook een deployment pipeline of gewoon een pipeline genoemd.
GitHub Actions pipeline	Het Continuous Integration & Delivery dienst van GitHub. Het volledige proces van code migreren, testen en uit te rollen gebeurt rechtstreeks vanuit de code repository in GitHub en gebeurt automatisch met minimale manuele interactie zodra er veranderingen zijn in de code.
GitHub Organisation	Een privé stuk in GitHub waarin alle code repositories van een organisatie kan bewaart en beheerd worden. Allemaal privé en geïsoleerd.
GitOps	Een werkwijze waarbij je infrastructuur en applicatie configuratie beheert via Git repositories, die als enige bron van waarheid dienen. Deployments gebeurt automatisch op basis van wijzigingen in Git.
Infrastructure as Code (IaC)	Het automatisch opzetten en beheren van IT-infrastructuur via code in plaats van handmatige interacties. Hierdoor krijg je consistente, herhaalbare en snel uitrolbare omgevingen.
Ingress	Een unieke Kubernetes onderdeel die inkomende HTTP/HTTPS verkeer van buiten de cluster naar interne services routeert via regels (zoals host- of pad gebaseerde routing). Meestal wordt dit beheert door een ingress controller.
Microsoft Azure Expert MSP partner	Het officiële Microsoft certificaat om als Cloud Solution Provider bedrijf samen te werken met Microsoft als partners. Dit certificaat bewijst dat dit bedrijf uitblinkt in het verkopen en beheren van Azure services bij klanten. Met deze partnership krijgt dit bedrijf verschillende voordelen zoals kortingen op Azure licenties & subscripties en prioriteit in Microsofts "referral engine", dat het bedrijf helpt om nieuwe klanten te kunnen krijgen. Maar een partner worden is een complex proces. Dit omvat verschillende audits en allerlei hoge verwachtingen. Dit certificaat moet ook jaarlijks hernieuwd worden.
Microsoft Cloud Adoption Framework (CAF)	Richtlijnen en best practices om bedrijven te helpen Azure Cloud te implementeren in hun huidige IT-omgeving.
Pods	De kleinste uitvoerbare eenheid in Kubernetes. Een Pod bevat een of verschillende containers die samen draaien, dezelfde netwerkconfiguratie delen en toegang hebben tot gedeelde opslag. Pods worden gebruikt om applicaties of onderdelen van applicaties te draaien in een Kubernetes cluster.
SKU (Stock Keeping Unit)	Een identificatiecode voor specifieke productvarianten binnen Azure te onderscheiden. Een SKU is een specifieke configuratie of versie van een Azure service die bepaalt welke functies, prestaties en prijs van toepassing zijn. Elke SKU komt overeen met een bepaalde capaciteit of service niveau binnen een Azure product. Zoals VM size, opslagtype of database tier.
Terraform	Terraform is een Infrastructure as Code tool waarmee infrastructuur automatisch wordt opgezet en beheert via code, grotendeels onafhankelijk van de Cloud provider.
Terraform module library van Arxus	De interne en zelfgemaakte Terraform module collectie. Deze collectie wordt gebruikt om de volledige infrastructuur omgeving in code te definiëren en te bouwen. Het gebruik van deze collectie standaardiseert het proces van Azure resource deployment via Terraform.
Weighted Ranking Method	De Weighted Ranking Method is een gestructureerde beslissingsmethode waarbij verschillende alternatieve systematisch met elkaar vergeleken wordt op basis van vooraf gelegde criteria. Elk criteria krijgt een bepaald gewicht gebaseerd op het belang ervan. Elk alternatief wordt per criteria gescoord, om zo uiteindelijk een totaalscore te berekenen. Hierdoor kan er constructief en systematisch keuzes onderbouwd worden.

Literatuurlijst

Arxus NV. (2026). *Demotronix-Arxus-SolutionDesign_v1.0.* Kontich: Arxus.

aws vs. azure. (2025). Opgehaald van https://www.geeksforgeeks.org: https://www.geeksforgeeks.org/blogs/aws-vs-azure/

Azure DevOps vs. GitHub Actions. (2025). Opgehaald van Wildnetedge.com: https://www.wildnetedge.com/blogs/azure-devops-vs-github-actions-which-wins-for-ci-cd

Azure Expert MSP. (2026). Opgehaald van https://partner.microsoft.com: https://partner.microsoft.com/en-lb/partnership/azure-expert-msp

DataCamp. (2025, February 19). *AWS vs. Azure vs. Google Cloud*. Opgehaald van www.datacamp.com: https://www.datacamp.com/blog/aws-vs-azure-vs-gcp

Eyckmans, S. (2026). Project Charter.

GeeksForGeeks. (2026, Februari 10). *Introduction to Terraform*. Opgehaald van www.geeksforgeeks.org: https://www.geeksforgeeks.org/devops/what-is-terraform/

Hub-spoke network topology in Azure. (2026). Opgehaald van learn.microsoft.com: https://learn.microsoft.com/en-us/azure/architecture/networking/architecture/hub-spoke

Khan, H. (2024). *What is Azure DevOps?* Opgehaald van Dev.to/hamzakhan: https://dev.to/hamzakhan/azure-devops-vs-github-actions-a-comprehensive-guide-with-examples-performance-metrics-46oc

Microsoft. (2026). *Cloud-adoption-framework*. Opgehaald van learn.microsoft.com: https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/

Microsoft. (2026). *Wat is Azure?* Opgehaald van azure.microsoft.com: https://azure.microsoft.com/nl-nl/resources/cloud-computing-dictionary/what-is-azure/?ef_id=_k_9b4bf9c1c374145706f79023fdb97f0b_k_&OCID=AIDcmmbnk3rt9z_SEM__k_9b4bf9c1c374145706f79023fdb97f0b_k_&msclkid=9b4bf9c1c374145706f79023fdb97f0b

Pulumi vs. Terraform. (2026). Opgehaald van Pulumi.com: https://www.pulumi.com/docs/iac/comparisons/terraform/

Terraform Licence Change. (2025). Opgehaald van https://controlmonkey.io: https://controlmonkey.io/resource/terraform-license-change-impact-2025/

Terraform vs OpenTofu. (2025). Opgehaald van Medium.com: https://medium.com/@averageguymedianow/terraform-vs-opentofu-the-complete-guide-to-infrastructure-as-code-tools-in-2025-7f1b9dccd9e7

Wat is Bicep? (2026). Opgehaald van learn.microsoft.com: https://learn.microsoft.com/nl-nl/azure/azure-resource-manager/bicep/overview?tabs=bicep

What is Jenkins? (2026). Opgehaald van GeeksForGeeks: https://www.geeksforgeeks.org/devops/what-is-jenkins/

Generatieve AI Policy

Tijdens het schrijven van deze thesis, is er generatieve AI gebruikt om het geschreven werk te verbeteren. Specifiek de Copilot tool van Arxus is gebruikt om mijn geschreven werk te verbeteren. Zoals controleren op grammatica, spelling, formaliteit en algemene zinsbouw. Alle prompts die gebruikt zijn bij het schrijven van alle stagedocumenten, deze thesis inbegrepen, zijn allemaal in de aard van "Verbeter deze tekst op grammatica, spelling, formaliteit en algemene zinsbouw." geformuleerd. Waarna ik deze output zelf nog gecontroleerd heb op kwaliteit, accuraatheid en aangepast aan mijn persoonlijke voorkeur. Dit allemaal om tot de definitieve uitkomst te geraken.

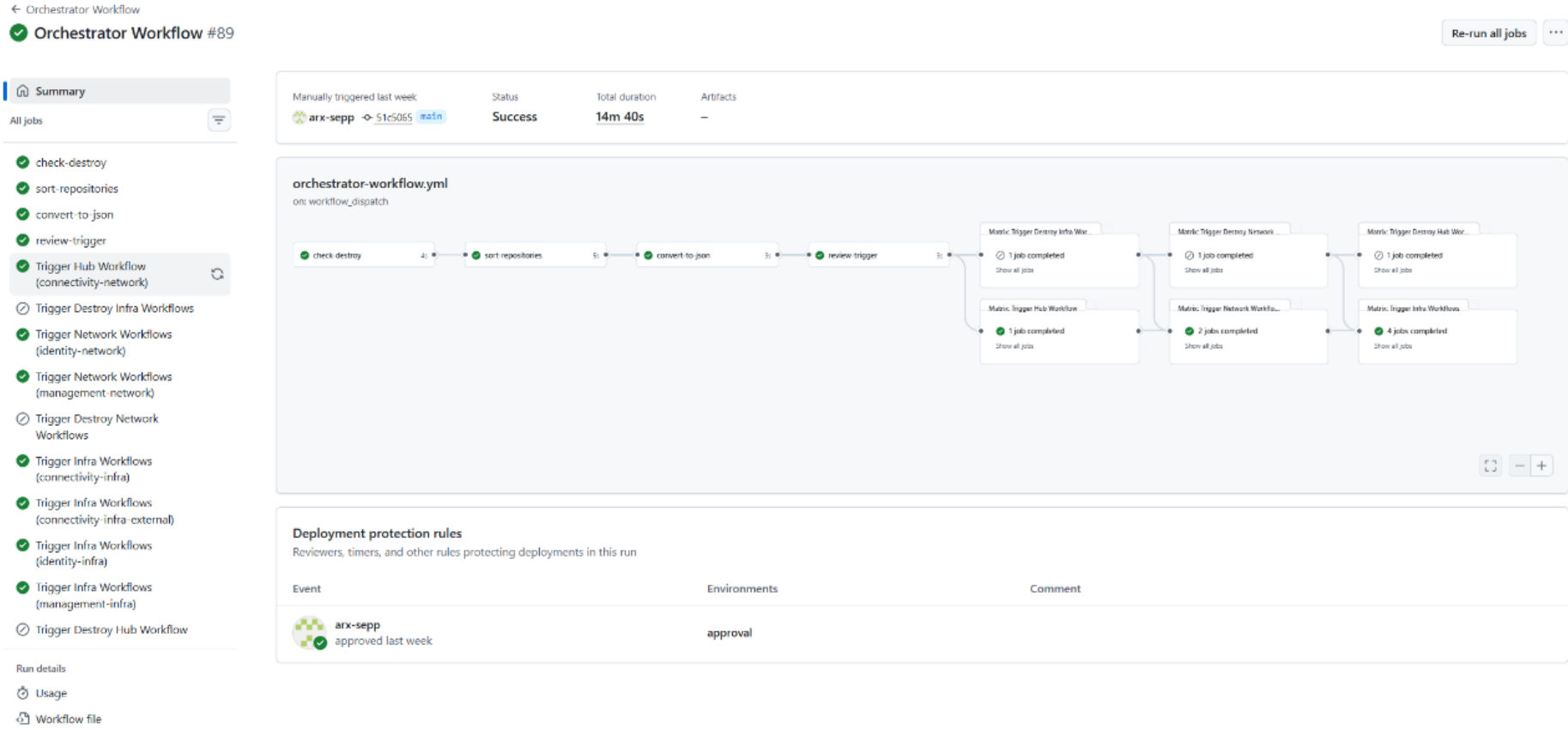
AI is altijd voorzichtig gebruikt en met een korrel zout opgenomen, het is nooit blindelings geïmplementeerd in het document. Want AI is niet perfect, het maakt regelmatig fouten of geeft een zelfverzonnen antwoord zonder bron. Bovendien is het partijdig (Copilot heeft voorkeur naar Microsoft producten bijvoorbeeld) en beperkt tot bronnen die dateren tot en met 2025. Hierdoor is AI niet een definitieve bron van waarheid.

Bijlagen

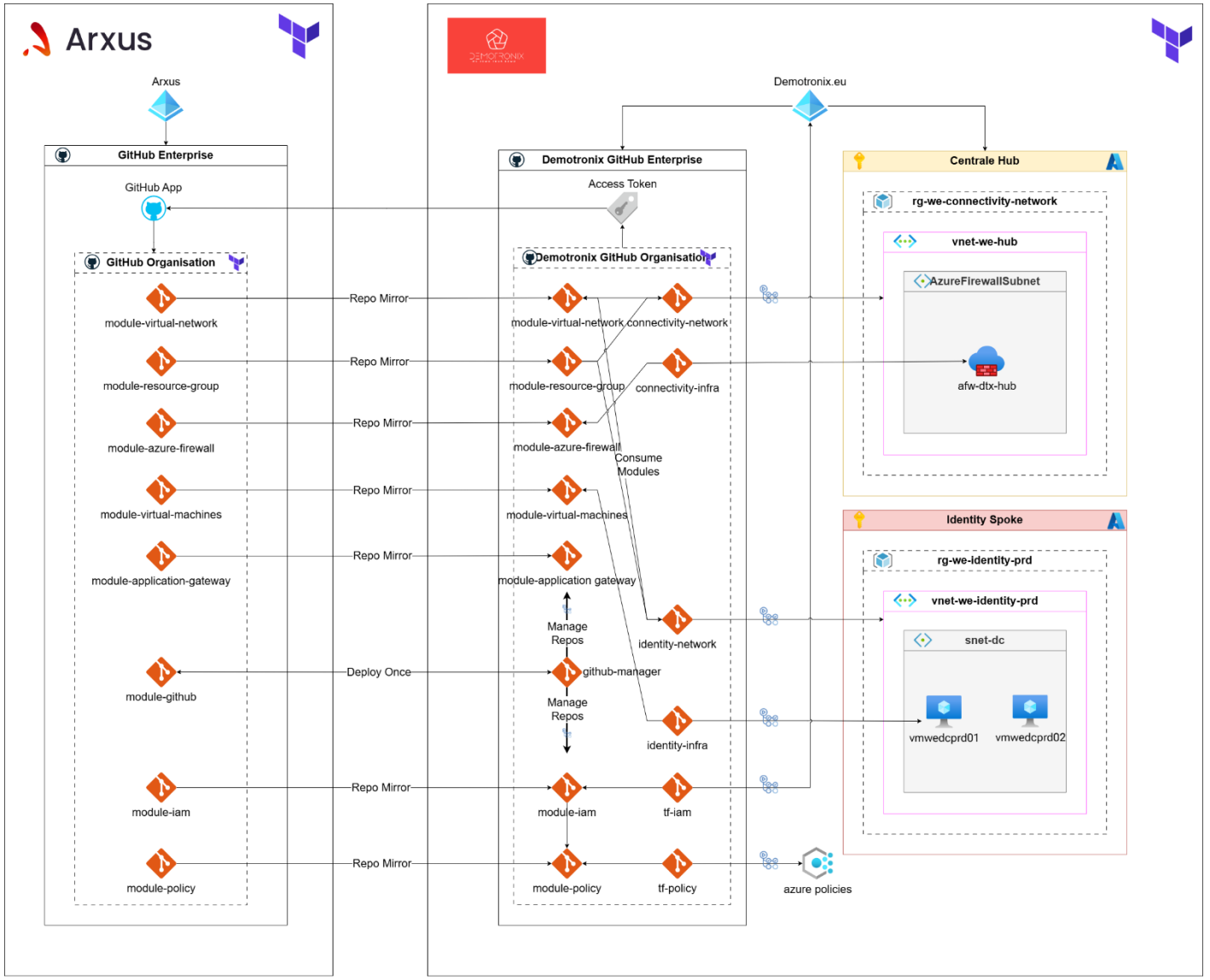
Dit hoofdstuk geeft een overzicht van alle bijlage die bij dit document horen en groepeert deze op één centrale locatie. Verwijzingen naar bijlagen in de hoofdtekst van deze thesis verwijzen steeds naar de bijlagen in dit hoofdstuk. Indien bepaalde bijlagen in de hoofdtekst niet duidelijk zijn zichtbaar of moeilijk toegankelijk zijn, kunnen ze hier opnieuw geraadpleegd worden voor volledigheid en verduidelijking.

De eerste bijlage bevindt zich op de volgende pagina van dit document.

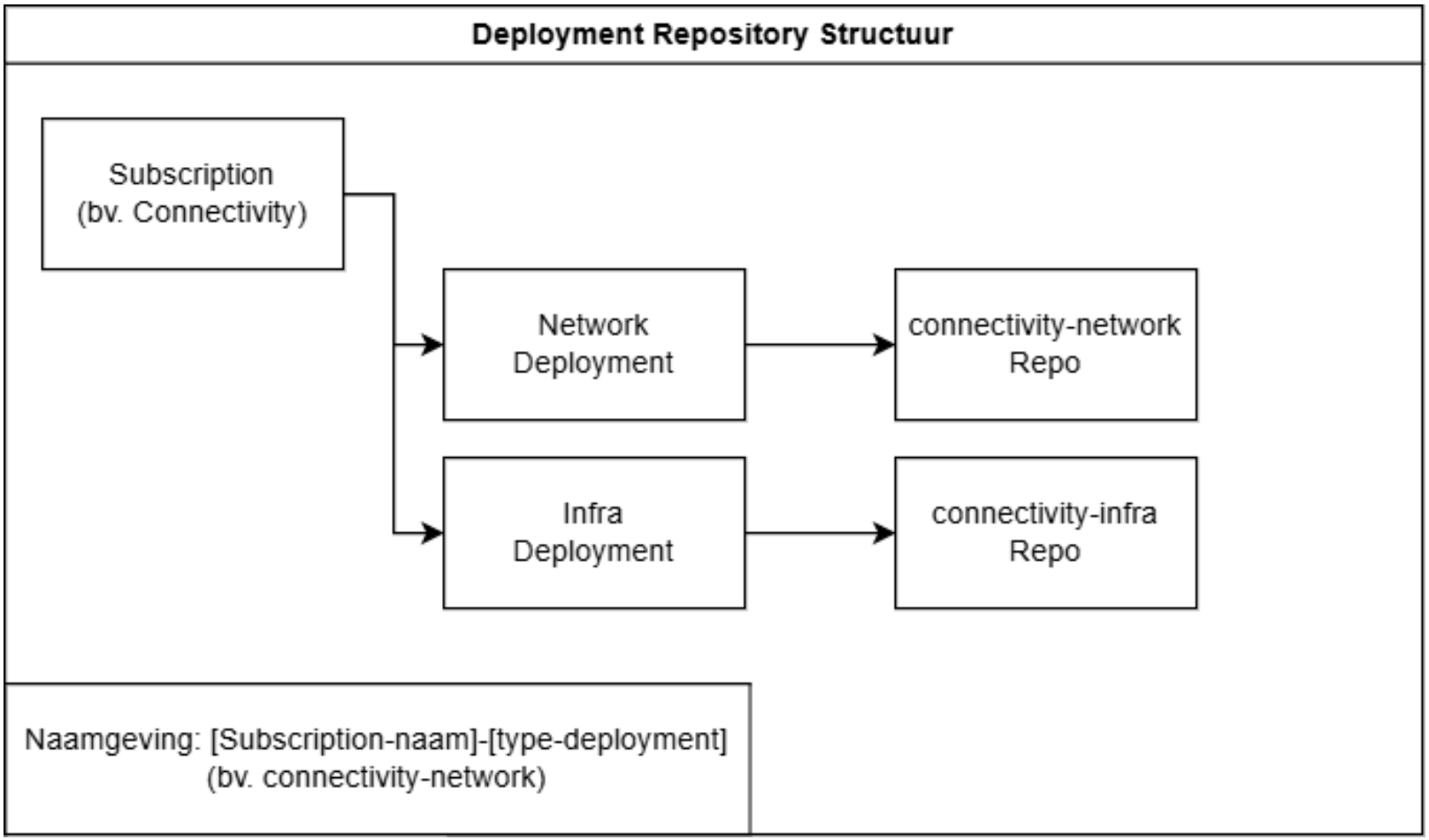
Bijlage 1: Orchestrator Workflow - Deploy Workflow



Bijlage 2: Terraform Module Library & Module Sync Design Diagram



Bijlage 3: Deployment Repository Structuur diagram



```
# Create Azure Firewall
module "azure_firewall" {
  for_each = { for k, v in var.azure_firewalls : k => v }
  source   = "git::https://github.com/[org]/[module].git"

  prefix                          = try(each.value.prefix, var.prefixes.firewall)
  public_ip_prefix                = try(each.value.public_ip_prefix, var.prefixes.public_ip_address)
  firewall_policy_prefix          = try(each.value.firewall_policy_prefix, var.prefixes.firewall_policy)
  delimiter                       = try(each.value.delimiter, var.delimiter)
  suffix                          = try(each.value.custom_name, null) == null ? each.value.suffix : null
  custom_name                     = try(each.value.custom_name, null)
  custom_public_ip_name           = try(each.value.custom_public_ip_name, null)
  custom_firewall_policy_name     = try(each.value.custom_firewall_policy_name, null)
  resource_group_name             = try(module.resource_groups[each.value.resource_group_name].data.name, each.value.resource_group_name)
  location                        = try(each.value.location, var.location)
  zones                           = try(each.value.zones, [1, 2, 3])
  ...
}
```

Bijlage 4: Terraform import module into deployment repo code snippet

```
subscription_id     = "[full-subscription-id]"
hub_subscription_id = "[hub-subscription-id]"
usecase_name        = "connectivity-network"
environment         = "prd"
location            = "westeurope"
network_state_file  = null

resource_groups = {
  rg-we-connectivity-network = {
    prefix                = "rg" # Optional: Defaults to var.prefixes.resource_group
    delimiter             = "-"  # Optional: Change the delimiter. Defaults to var.prefixes
    suffix                = "we-connectivity-network"
    custom_name           = null # Optional: takes precedence over prefix, delimiter, suffix
    lock_levels           = []   # Optional: CanNotDelete or ReadOnly
    deploy_default_budget = false

    tags = { # Optional
      environment = "prd"
      usecase     = "connectivity-network"
    }
  }

  rg-we-connectivity-bastion = {
    ...
    }
  }
  ...
}
```

Bijlage 5: Terraform resource creation code snippet

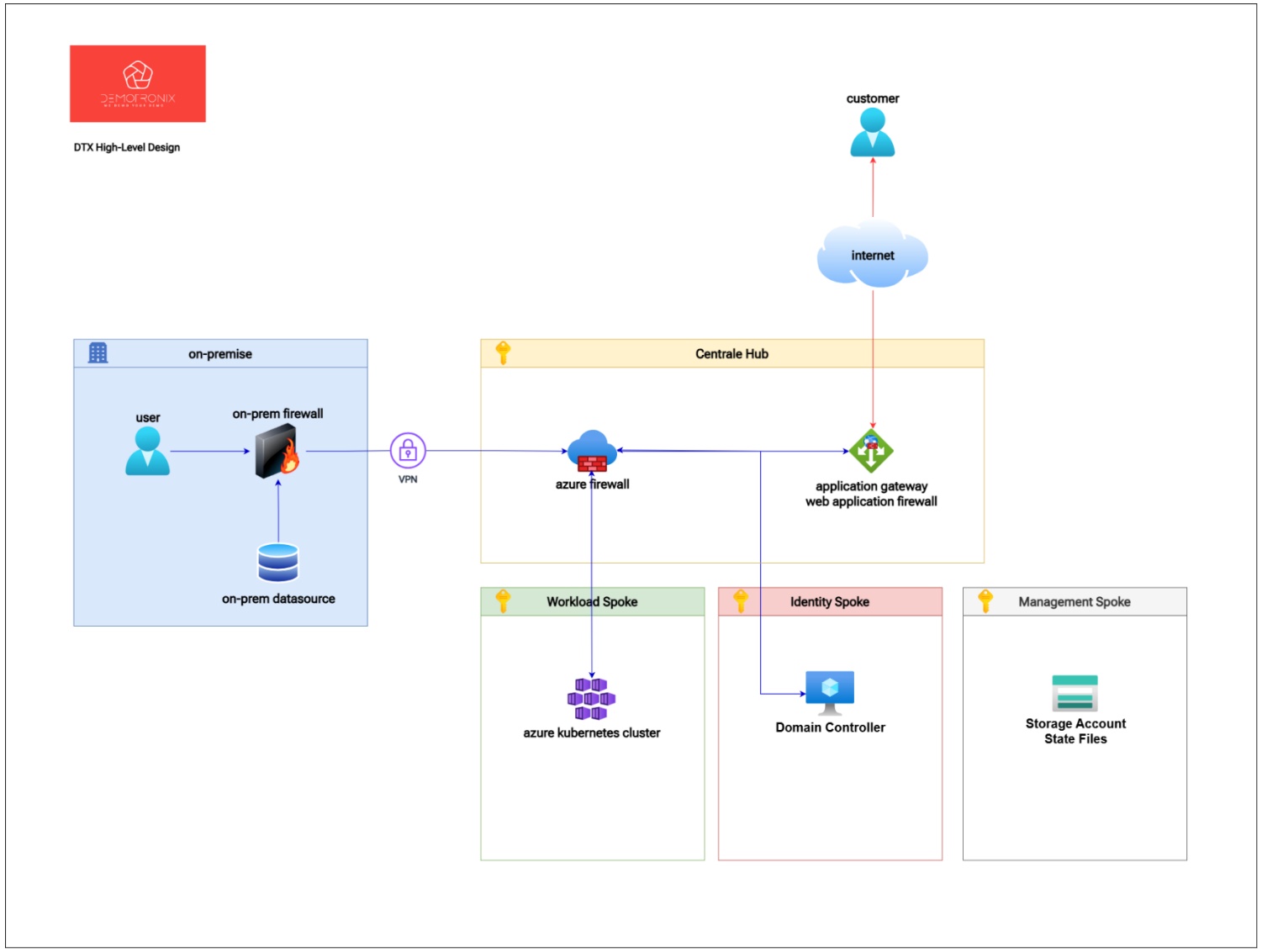
```
virtual_network_gateways = {
  vgw-we-hub = {
    prefix                        = "vgw" # Optional: Defaults to virtual_network_gateway value in var.prefixes
    local_network_gateway_prefix = "lgw" # Optional: Defaults to local_network_gateway value in var.prefixes
    public_ip_prefix              = "pip" # Optional: Defaults to public_ip value in var.prefixes
    connection_prefix             = "con" # Optional: Defaults to connection value in var.prefixes
    delimiter                     = "-"   # Optional: Change the delimiter. Defaults to -
    suffix                        = "we-hub"
    custom_name                   = null # Optional: takes precedence over prefix, delimiter, suffix
    resource_group_name           = "rg-we-connectivity-network"
    sku                           = "VpnGw1AZ"
    type                          = "Vpn"           # Vpn or Expressroute
    active_active                 = false           # Optional: requires a HighPerformance or an UltraPerformance SKU
    edge_zone                     = null            # Optional
    enable_bgp                    = false           # Optional
    bgp_asn                       = null            # Optional
    bgp_peering_addresses         = []              # Optional
    bgp_peer_weight               = null            # Optional
    gateway_subnet_id             = "GatewaySubnet" # Full resource ID or module key
    virtual_network_key           = "vnet-we-vgw"   # Only when using module key for gateway_subnet_id
    onprem_connections = [
      {
        name                              = "INX01"
        custom_local_network_gateway_name  = null            # Optional: takes precedence over prefix, delimiter, suffix
        custom_connection_name             = "we-demogem-hub" # Optional: takes precedence over prefix, delimiter, suffix
        type                              = "IPsec"          # IPsec or ExpressRoute
        ...
      }
    ]
  }
}
```

Bijlage 6: VPN Gateway full code snippet

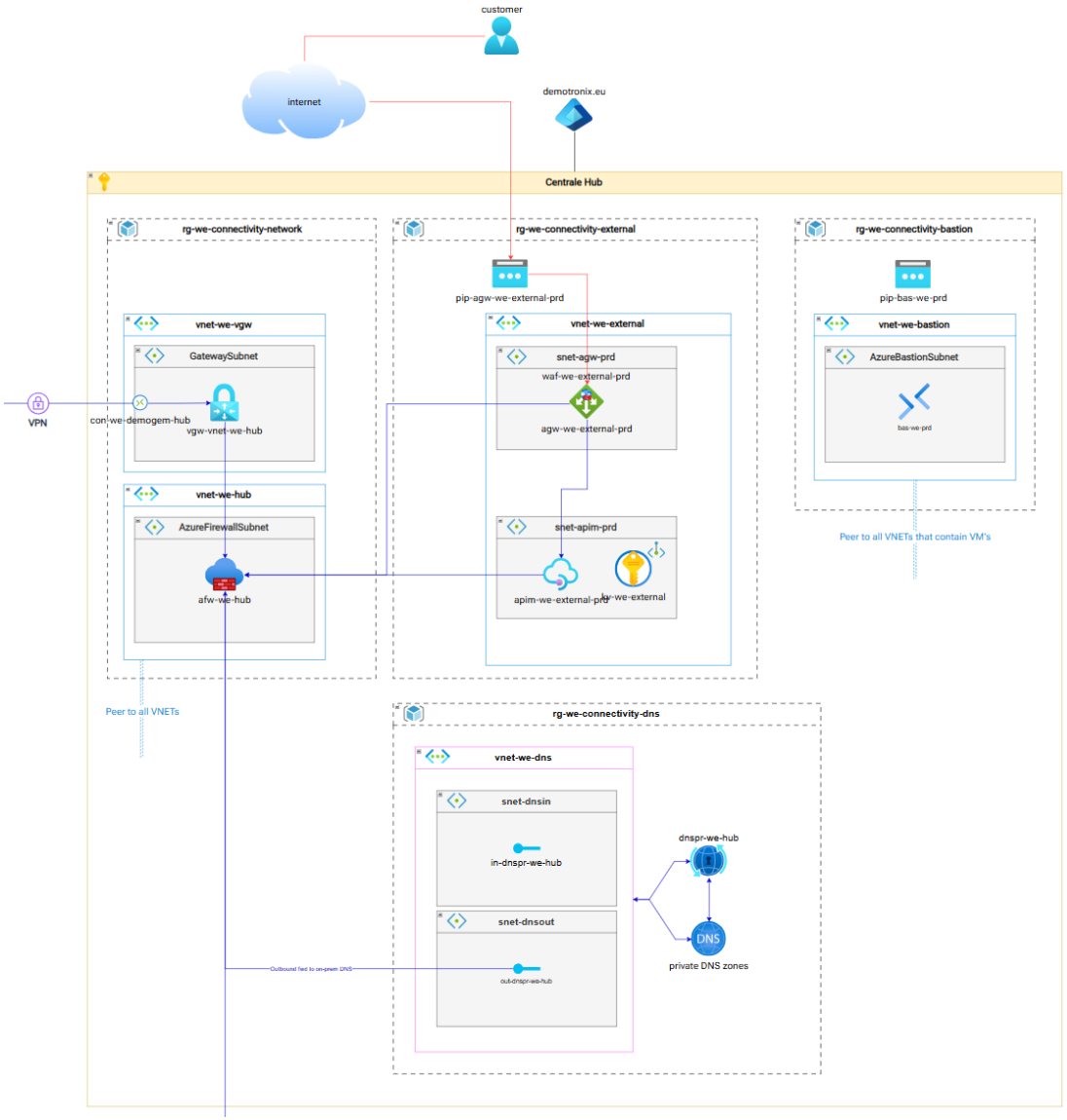
```
bastion_hosts = {
  bas-we-prd = {
    prefix                      = "bas" # Optional: Defaults to var.prefixes.bastion
    delimiter                   = "-"   # Optional: Defaults to var.delimiter
    suffix                      = "we-bastion"
    custom_name                 = null # Optional: takes precedence over prefix, delimiter & suffix
    resource_group_name         = "rg-we-connectivity-bastion"
    bastion_sku                 = "Basic"             # Optional: Accepted values are Developer, Basic, Standard and Premium. Defaults to Standard.
    zones                       = [1]                 # Optional: Defaults to [1, 2, 3]
    copy_paste_enabled          = true                # Optional: Defaults to true.
    file_copy_enabled           = false               # Optional: Defaults to false. Only supported in Standard or Premium SKU
    ip_connect_enabled          = false               # Optional: Defaults to false. Only supported in Standard or Premium SKU
    kerberos_enabled            = false               # Optional: Defaults to false. Only supported in Standard or Premium SKU
    scale_units                 = 2                   # Optional: Must be between 2 and 50. scale_units is always 2 when sku is Basic.
    shareable_link_enabled      = false               # Optional: Defaults to false. Only supported in Standard or Premium SKU
    tunneling_enabled           = false               # Optional: Defaults to false. Only supported in Standard or Premium SKU
    session_recording_enabled   = false               # Optional: Defaults to false. Only supported in Premium SKU
    ip_configuration_prefix     = "ipc"               # Optional: Defaults to var.prefixes.ip_configuration
    custom_ip_configuration_name = null               # Optional: takes precedence over ip_configuration_prefix, delimiter & suffix
    subnet_id                   = "AzureBastionSubnet" # Required: naming must be AzureBastionSubnet -> full resource ID or module key
    virtual_network_key         = "vnet-we-bastion"    # Only required when using module key within subnet_id
    public_ip_address_id        = null                # Optional: add an existing public IP to the bastion, otherwise a new public IP will be created
    public_ip_prefix            = "pip"               # Optional: Defaults to var.prefixes.public_ip_address
    custom_public_ip_name       = "pip-bas-we-prd"    # Optional: takes precedence over public_ip_prefix, delimiter & suffix
    tags                        = {}                  # Optional: Defaults to {}
  }
}
```

Bijlage 7: Bastion Host full code snippet

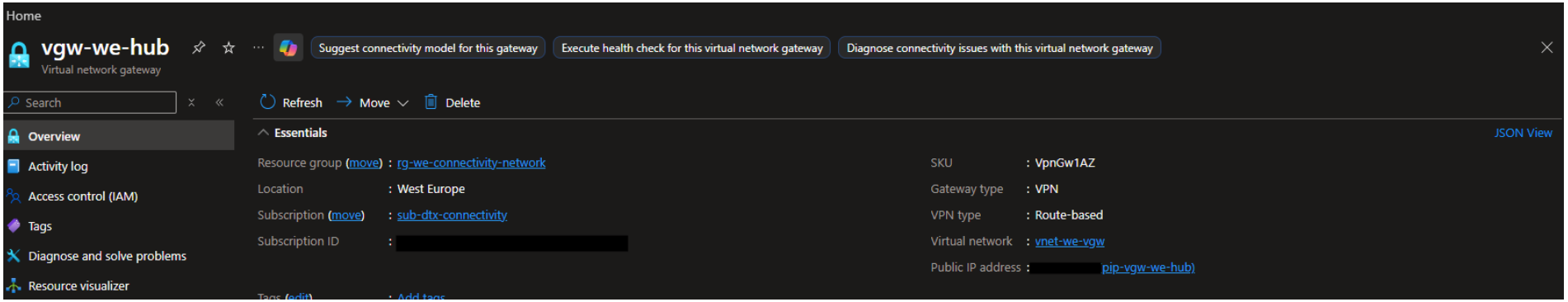
Bijlage 8: High-Level Design



Bijlage 9: Hub Design



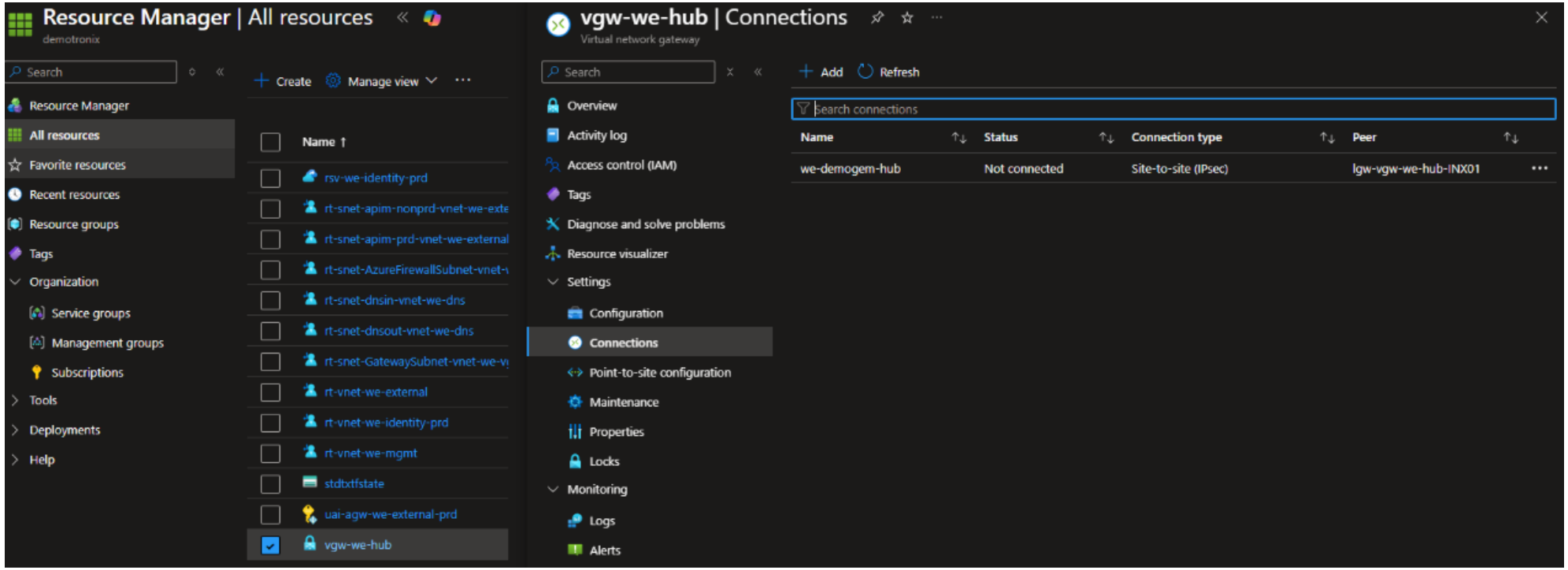
Bijlage 10: VGW Overzicht



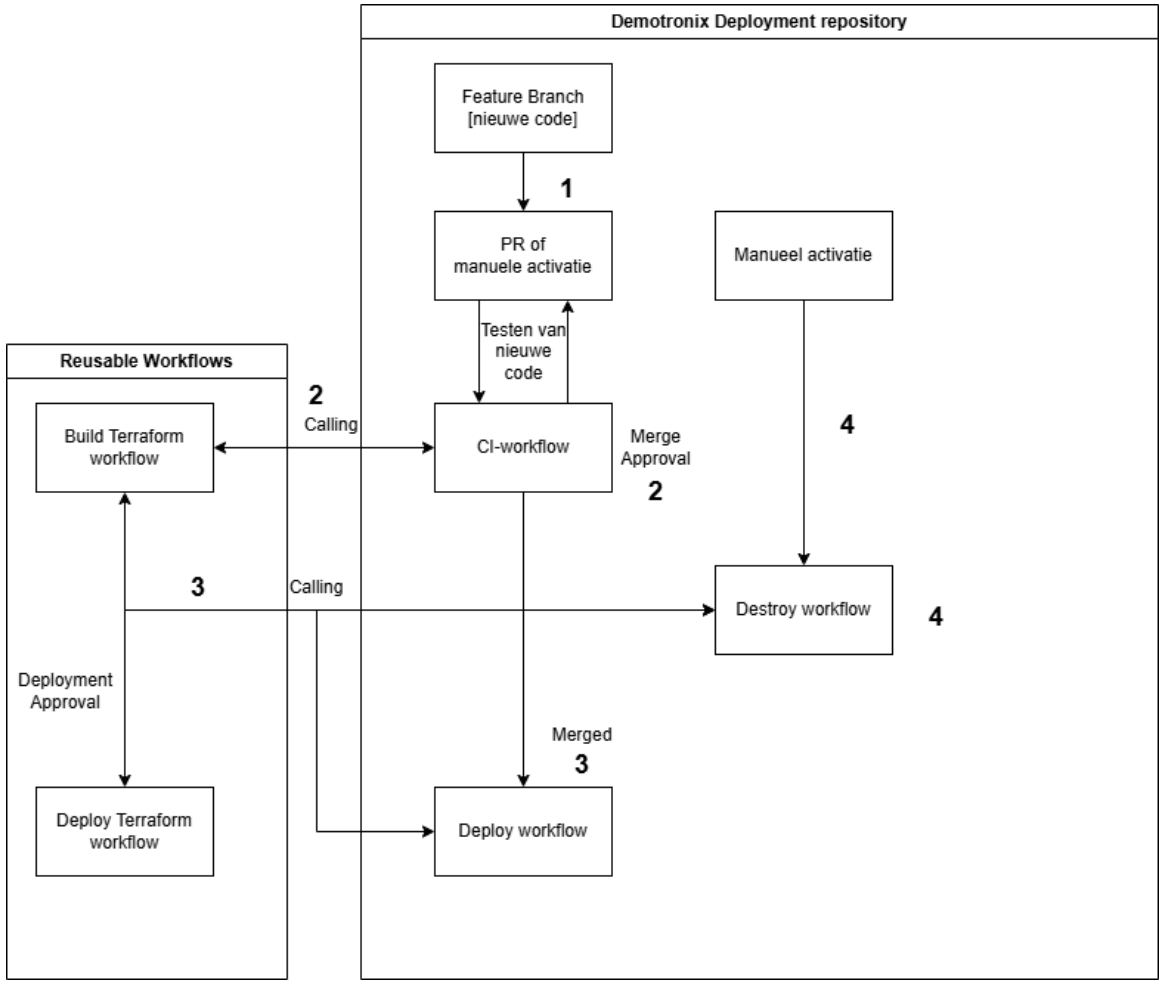
Bijlage 11: Firewall Rules

	column 1	column 2	column 3	column 4	column 5	column 6	column 7
1	name	source_addresses	source_ip_groups	protocols	fqdn_tags	destination_fqdns	web_categories
2	APIM-To-KV	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	kv-dtx-we-external.vault.azure.net	null
3	APIM-To-AAD	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	login.windows.net	null
4	APIM-To-Blob	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.blob.core.windows.net	null
5	APIM-To-File	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.file.core.windows.net	null
6	APIM-To-PackageStorage	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.blob.core.windows.net	null
7	APIM-To-SmtpQueue	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.queue.core.windows.net	null
8	APIM-To-Queue	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.queue.core.windows.net	null
9	APIM-To-Table	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	*.table.core.windows.net	null
10	APIM-To-RegionalAD	null	ipg-snet-apim-prd-vnet-we-external	Https:443	null	westeurope.login.microsoft.com	null

Bijlage 12: VGW Connectie



Bijlage 13: Deployment Workflow Structuur



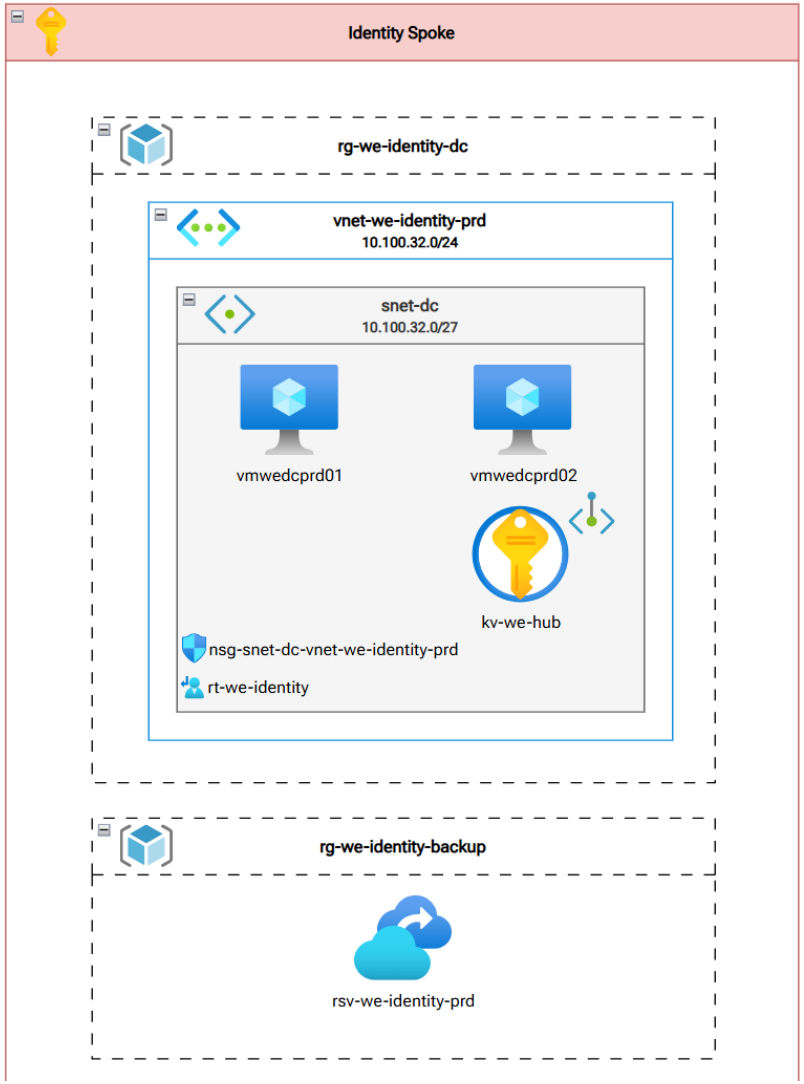
Bijlage 14: Prioriteit toewijzen en sorteren op alfabet

```
        ...
            priority() {
                local name="$1"

                if [[ "$name" == "connectivity-network" ]]; then
                    echo 0
                elif [[ "$name" == *network* ]]; then
                    echo 1
                elif [[ "$name" == *infra* ]]; then
                    echo 2
                else echo 3
                fi
            }

            # sort repos based on priority, then alphabetical order.
Connectivity-network is always first!
            sorted=$(printf "%s\n" "${repos[@]}" \
                | while IFS= read -r repo; do
                    printf "%s\t%s\n" "$(priority "$repo")" "$repo"
                done \
                | sort -n -k1,1 -k2,2 \
                | cut -f2
            )
      ...
```

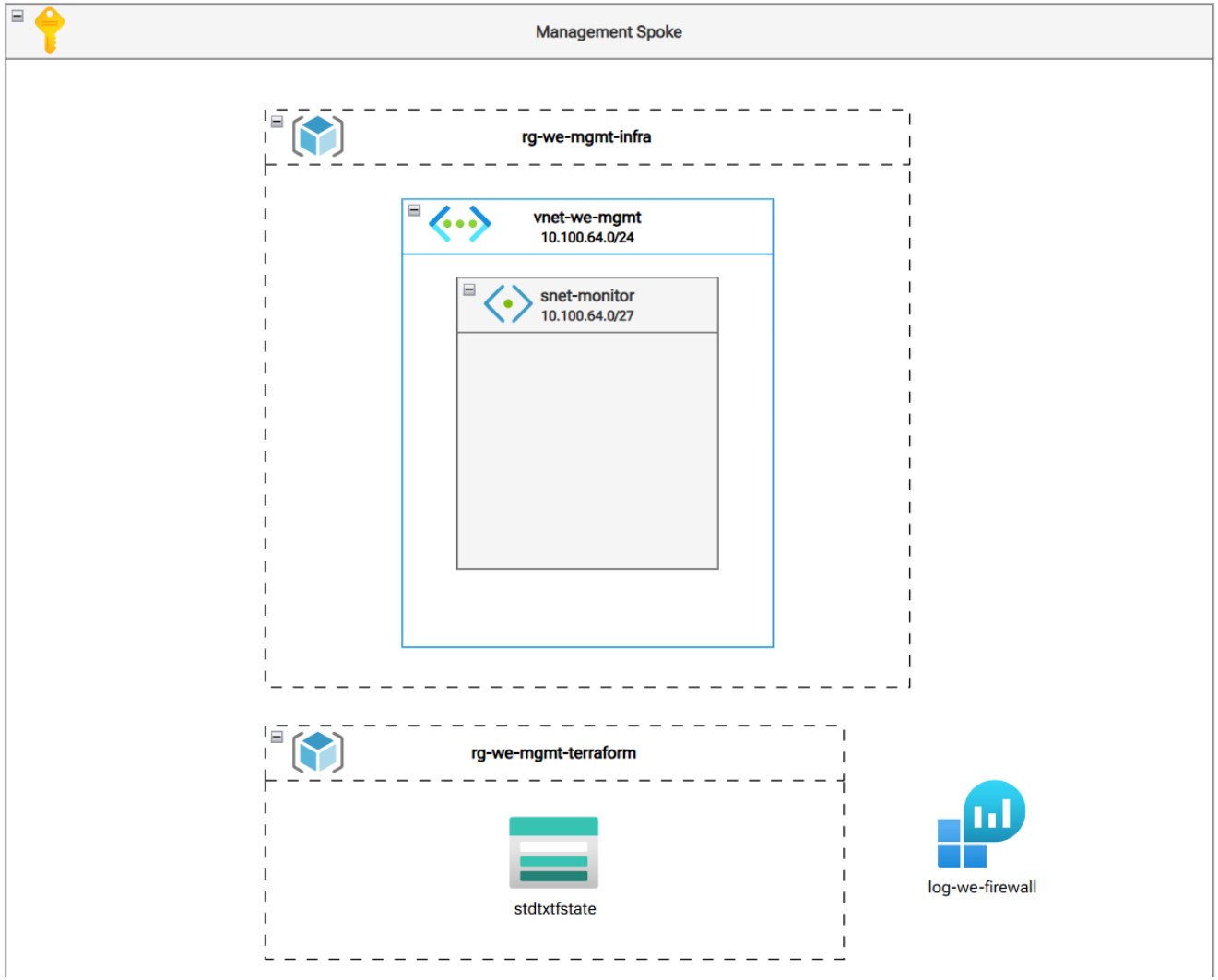
Bijlage 15: Deploy/Destroy Logica

```
    ...
        # Reverse order when performing destroy
        if [[ "${{ inputs.destroy }}" == "true" ]]; then
            sorted=$(printf "%s\n" "$sorted" | tac)

            echo "Repo Count: ${#sorted[@]}"
            echo "Repository List:"
            echo "$sorted"
        else
            echo "Repo Count: ${#sorted[@]}"
            echo "Repository List:"
            echo "$sorted"
        fi
    ...
```

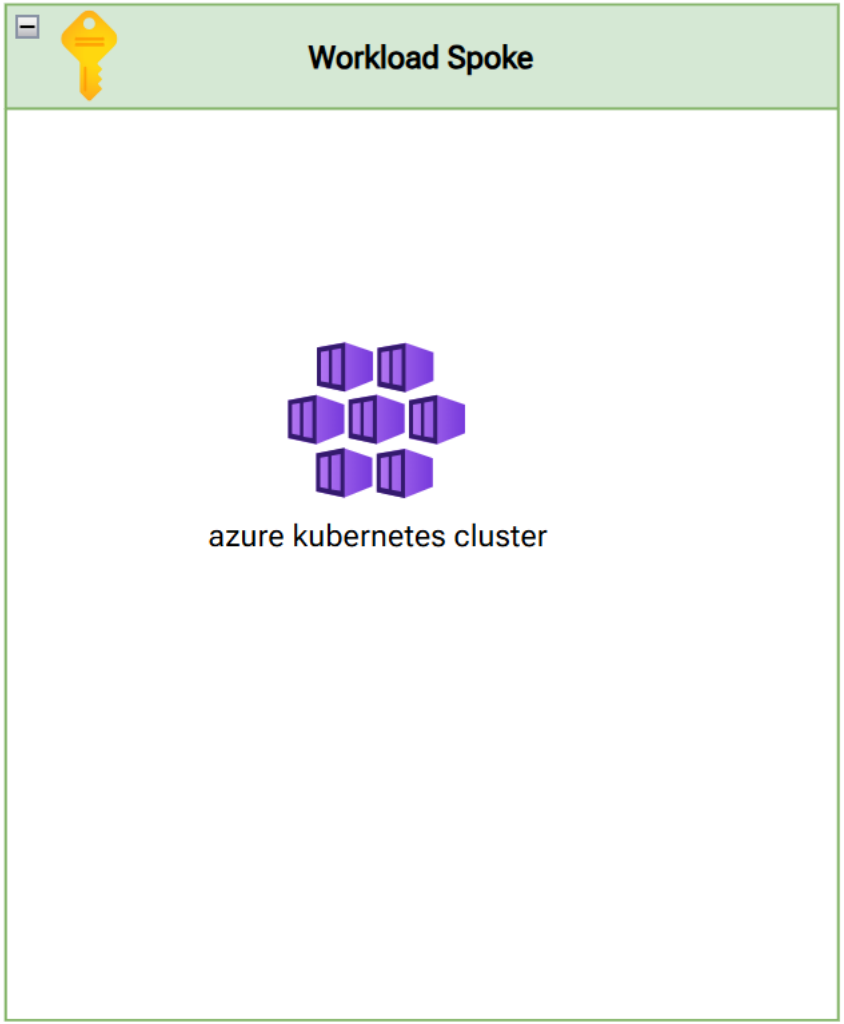
Bijlage 16: Identity Spoke



Bijlage 17: Management Spoke

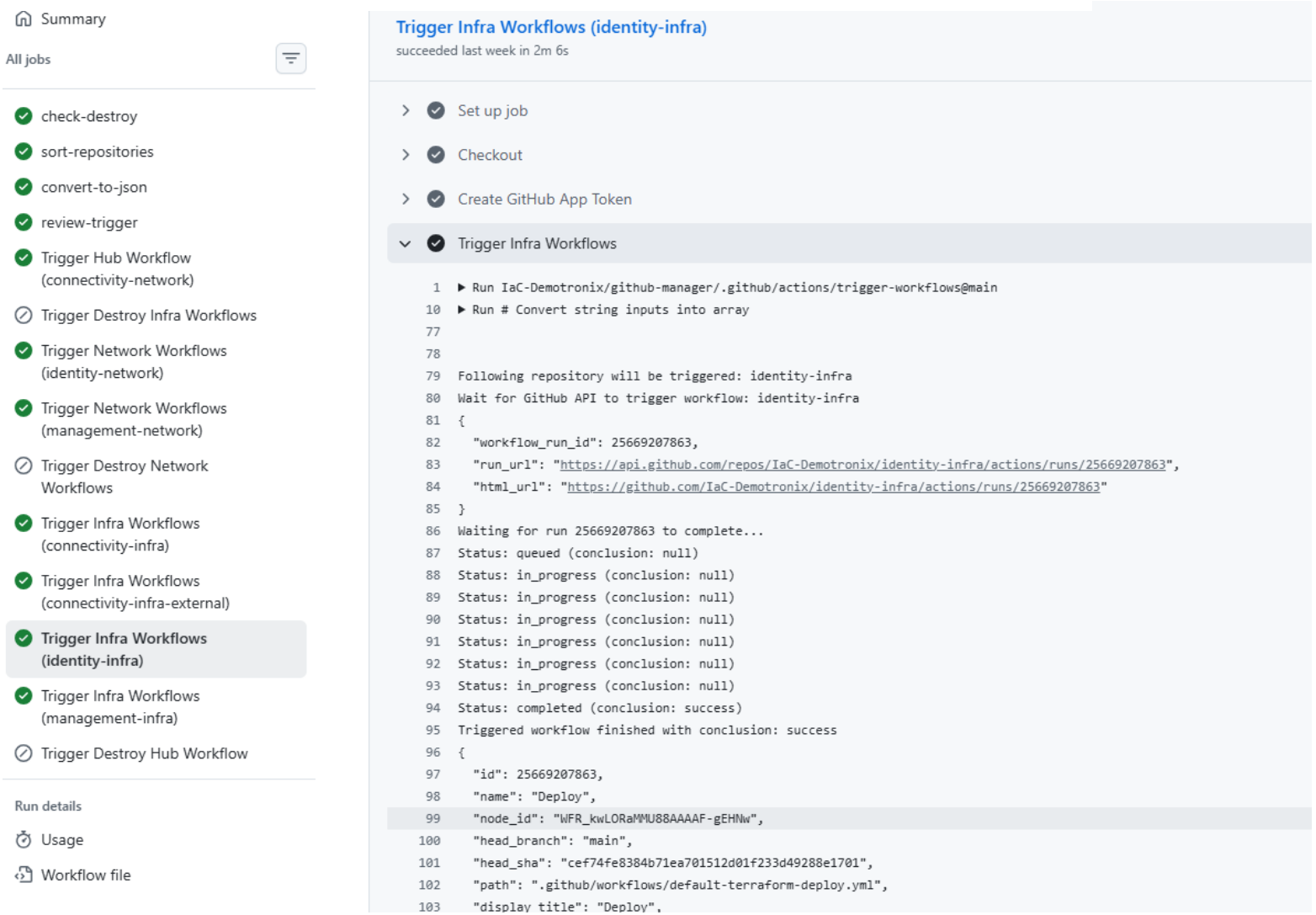


Bijlage 18: Workload Spoke

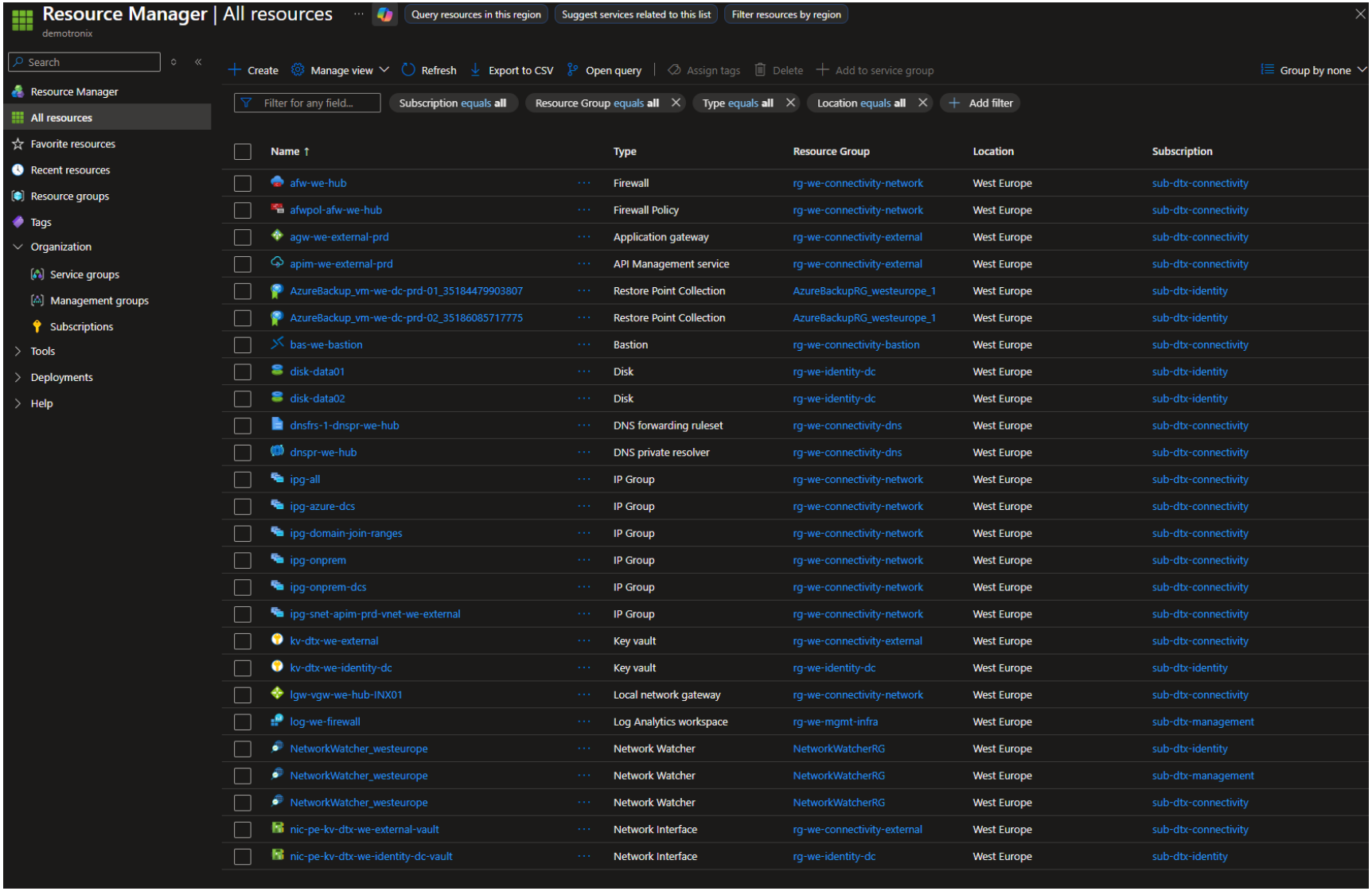


Bijlage 19: Terraform Build Plan

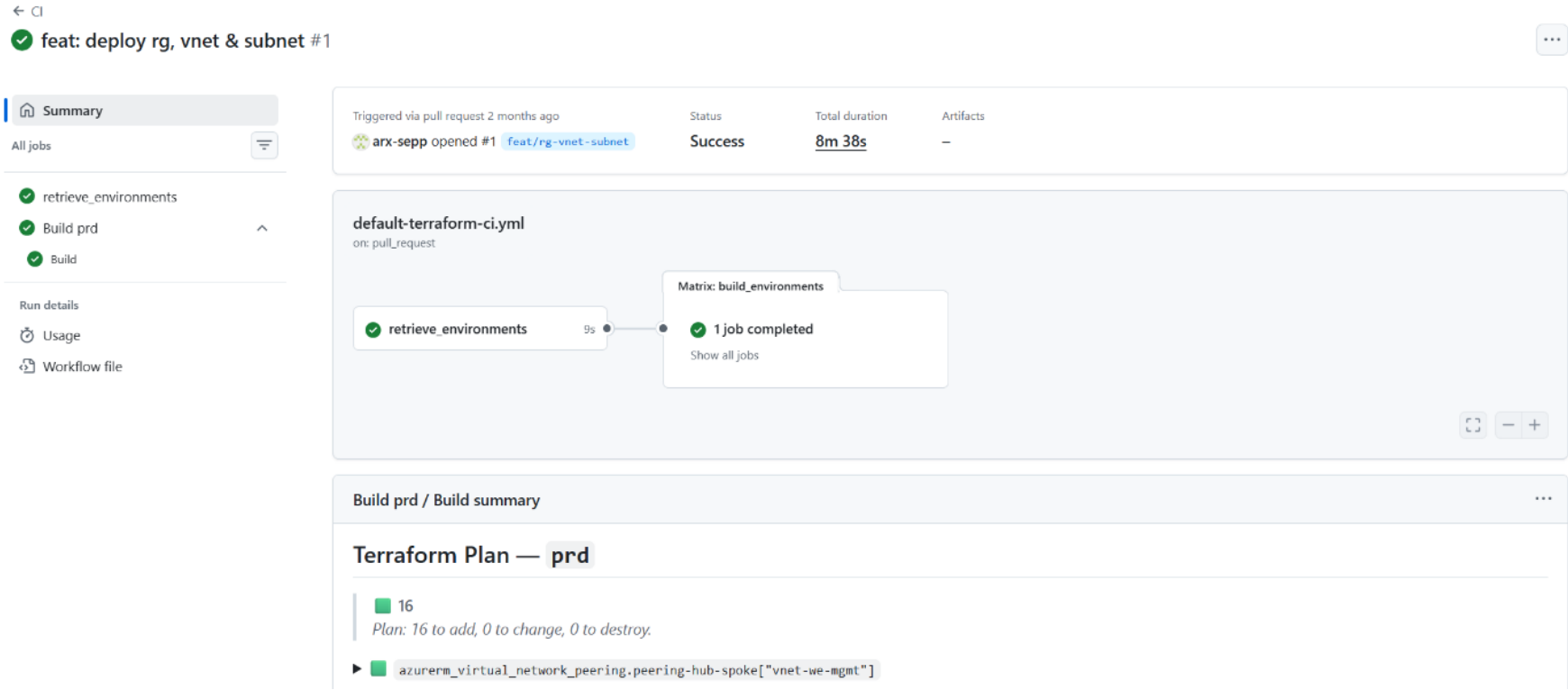
Bijlage 20: Orchestrator Workflow Deployment



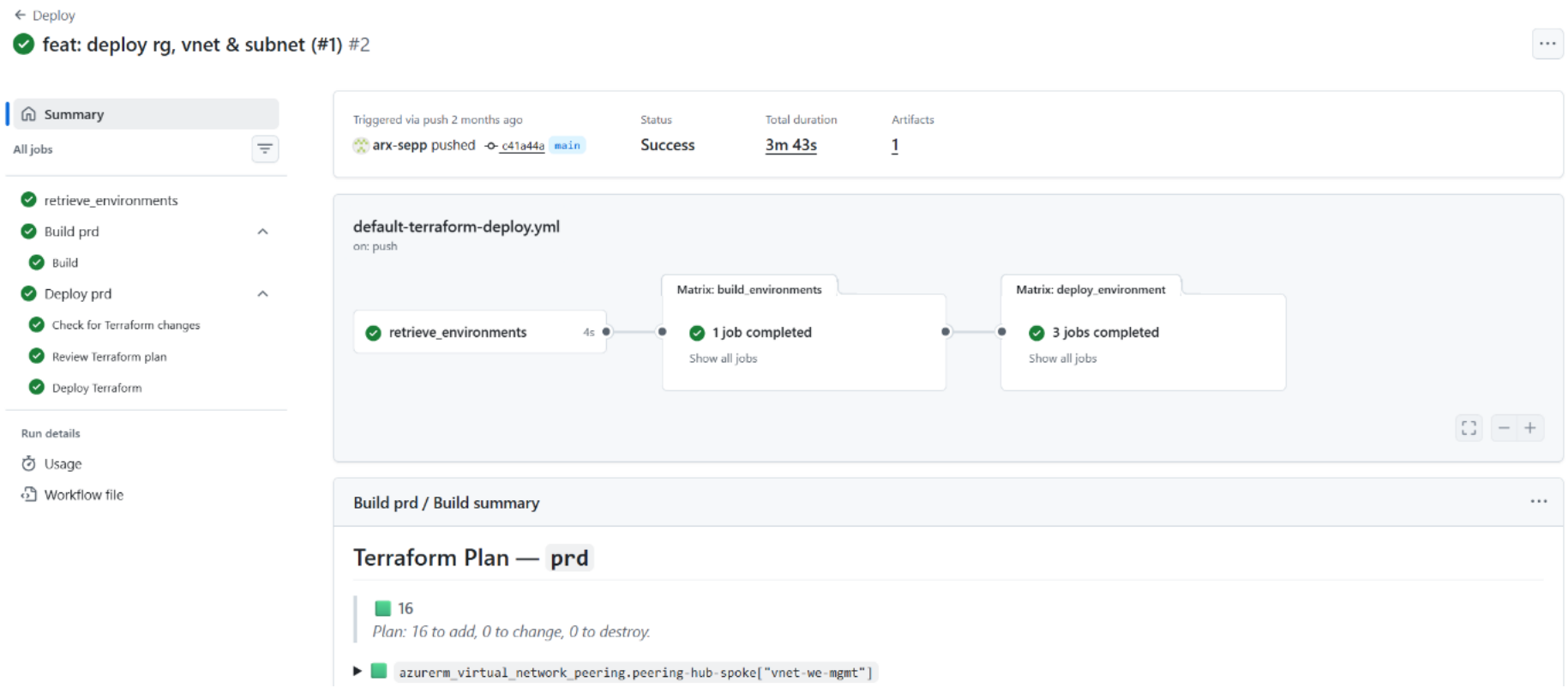
Bijlage 21: Full Azure Deployment



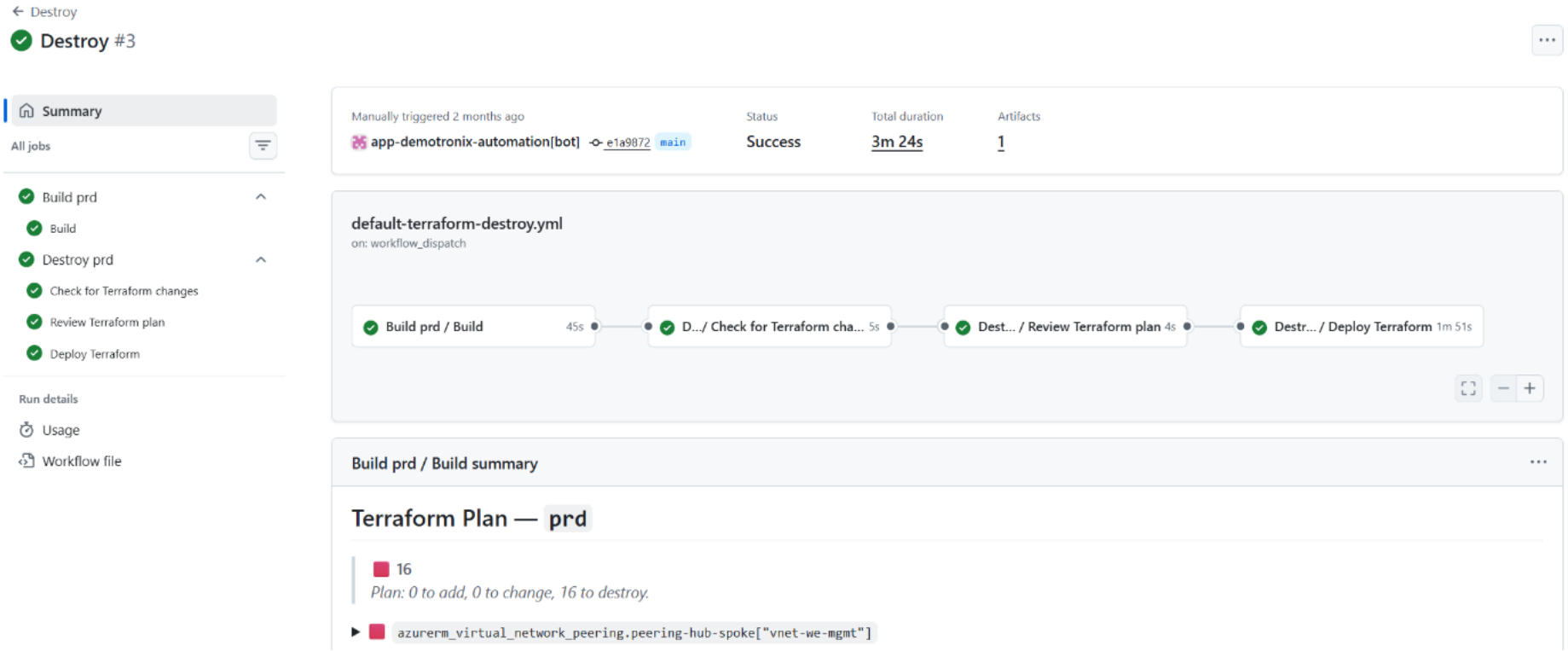
Bijlage 22: CI-Workflow



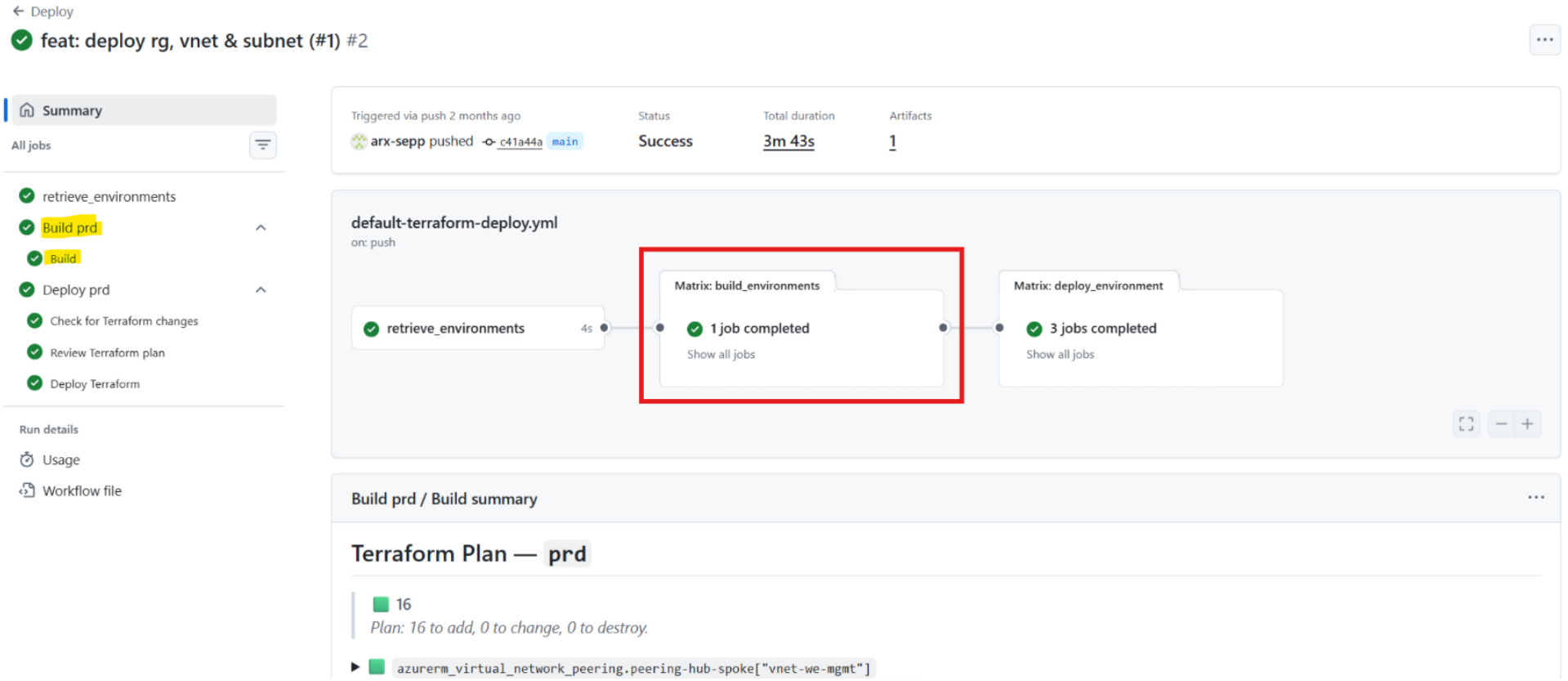
Bijlage 23: Deploy Workflow



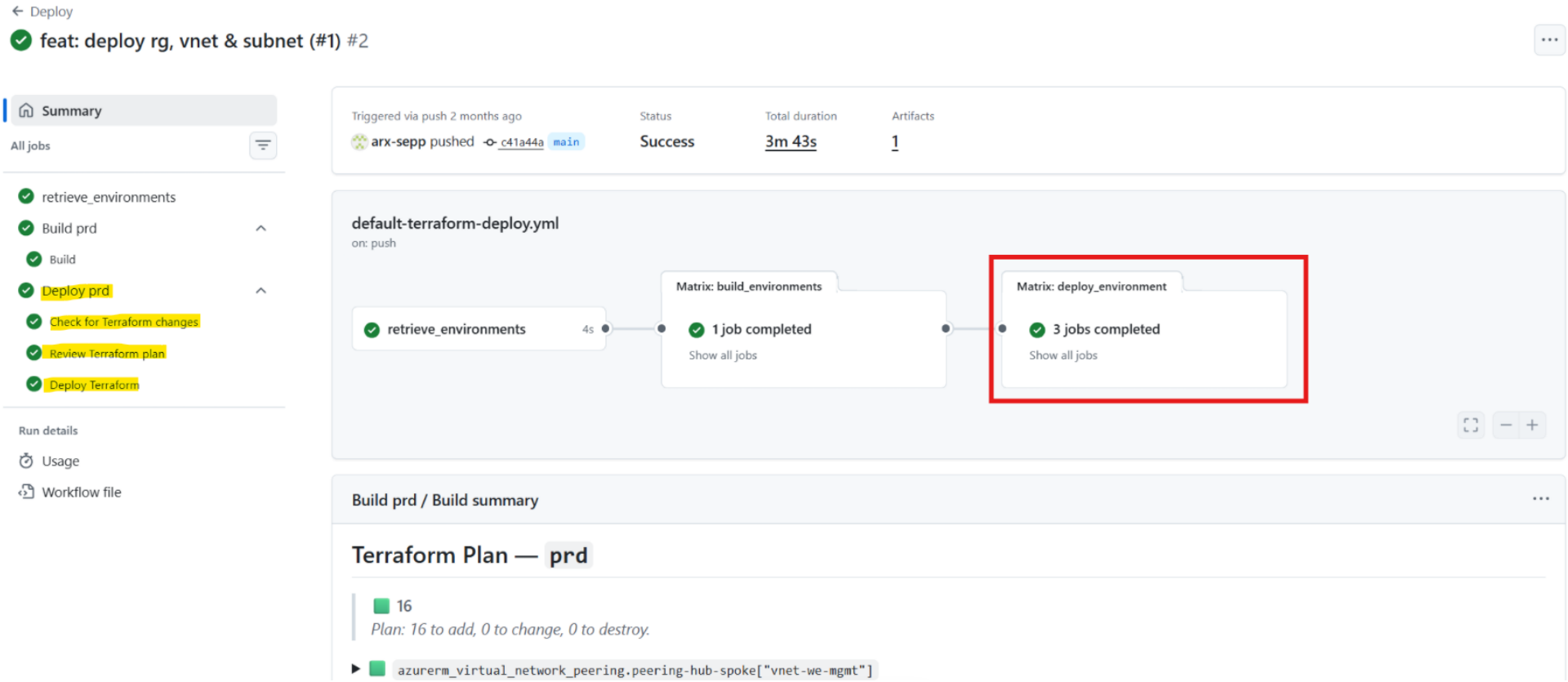
Bijlage 24: Destroy Workflow



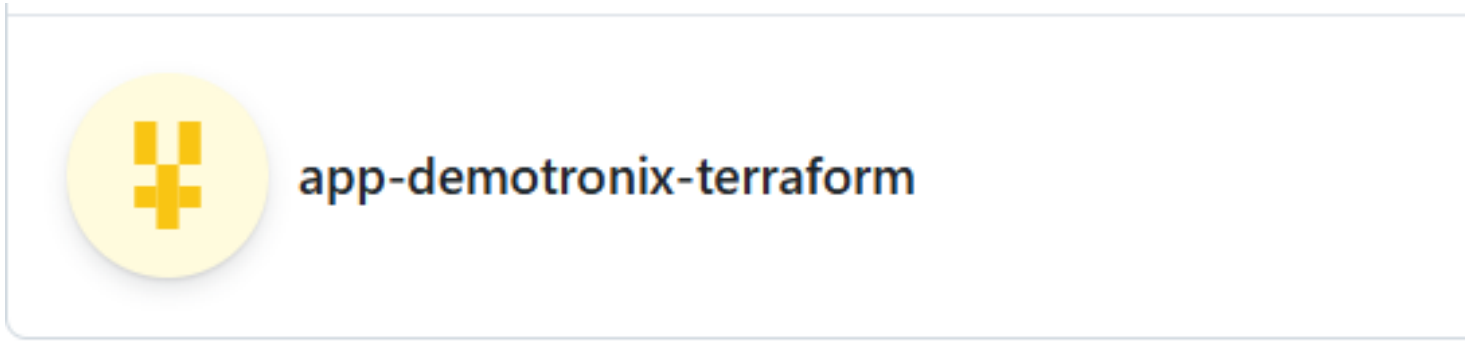
Bijlage 25: Reusable Workflow - Build



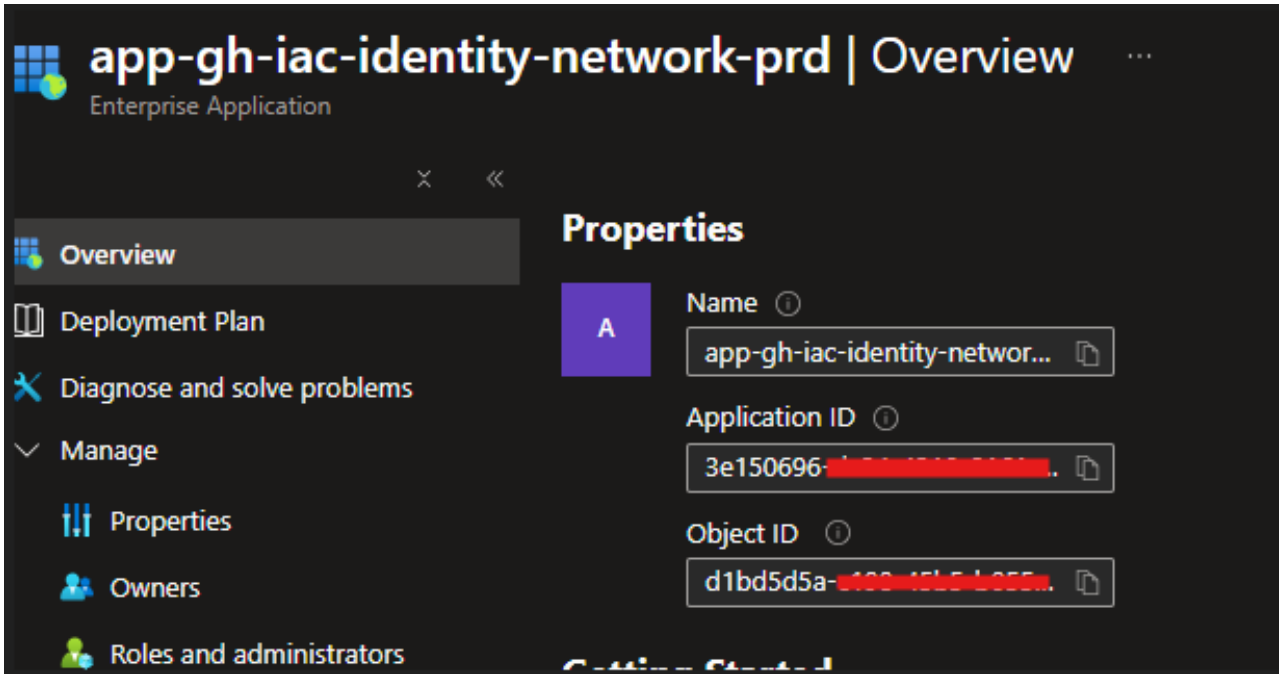
Bijlage 26: Reusable Workflow - Deploy/Destroy



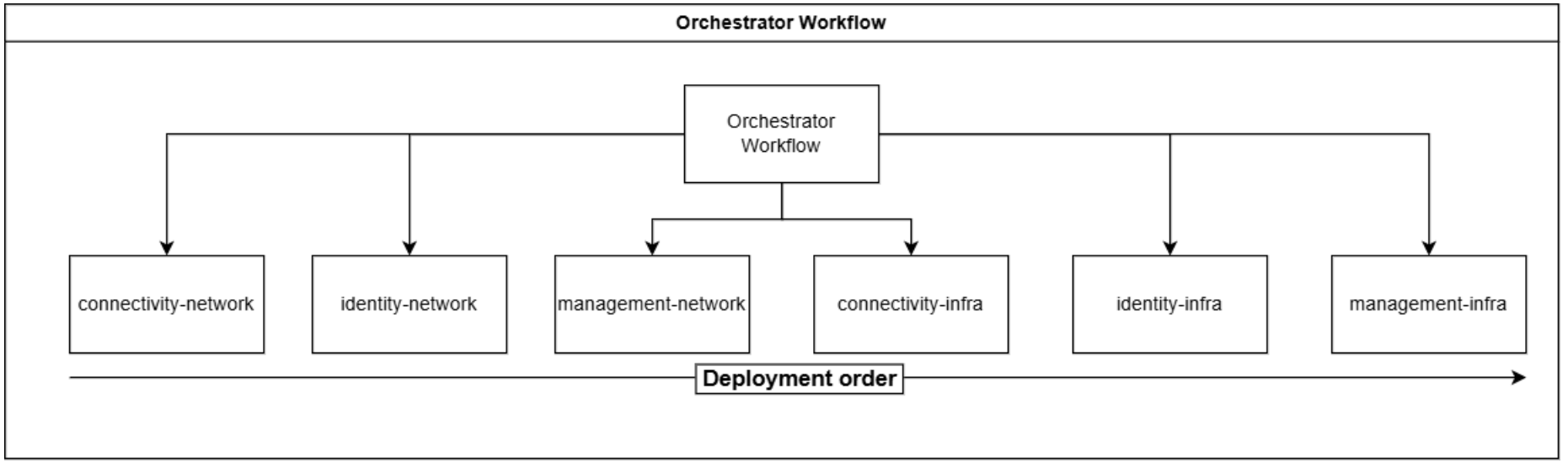
Bijlage 27: GitHub App



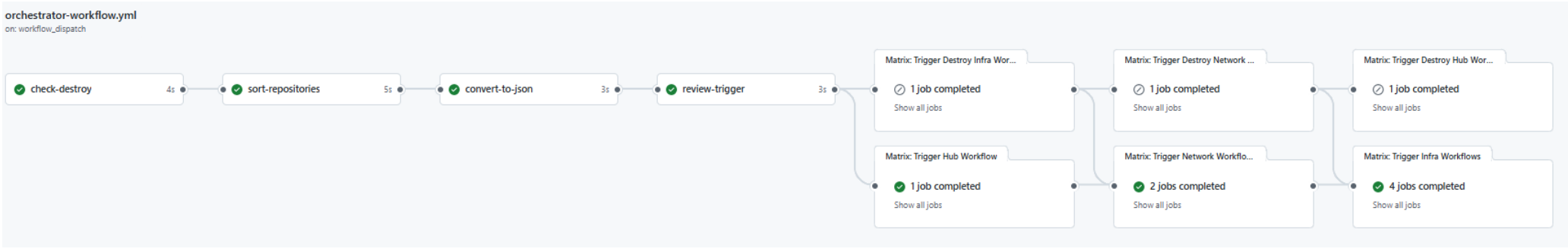
Bijlage 28: App Registration



Bijlage 29: Orchestrator Workflow Diagram



Bijlage 30: Orchestrator Workflow - Proces



Bijlage 31: Orchestrator Workflow - Manuel Dispatch + Destroy Workflow