THOMAS MORE

Stage infrastructuur: Platform Engineering

Realisatiedocument

Yorrit Belmans
Student Bachelor in de Elektronica-ICT – Cloud & Cyber Security

2026

Inhoudsopgave

FIGUUR LIJST

TABELLEN LIJST

1. Inleiding

Binnen IT-organisaties groeit de nood aan gestandaardiseerde, herhaalbare platformen waarop ontwikkelteams zelfstandig applicaties kunnen uitrollen zonder telkens opnieuw infrastructuur handmatig te configureren. Platform Engineering is de discipline die dit probleem aanpakt. Het primaire doel van deze visie is Cognitive Load Reduction: het verlagen of wegnemen van de operationele complexiteit voor ontwikkelaars, zodat zij zich volledig kunnen focussen op het schrijven van applicaties in plaats van infrastructuurbeheer.

Om dit te bereiken bouwt en onderhoudt een platformteam een Internal Developer Platform (IDP). Dit platform wordt intern behandeld volgens het Platform as a Product-principe, de ontwikkelaars zijn de 'klanten' en hun Developer Experience (DX) staat centraal. Het IDP biedt hen een gestroomlijnde Self-Service Infrastructure-ervaring, terwijl het platformteam onder de motorkap de controle houdt over best practices, beveiliging en schaalbaarheid.

Deze stage, uitgevoerd bij Axxes Consulting in de periode van 23 februari tot en met 22 mei 2026, heeft als doel een MVP (Minimum Viable Product) van een dergelijk IDP te ontwerpen, te documenteren en te realiseren op een on-premises VMware vSphere-omgeving. Het platform wordt volledig declaratief beheerd volgens GitOps-principes, de gewenste toestand van de infrastructuur en alle applicaties wordt beschreven in Git-repositories en geautomatiseerde tooling zorgt ervoor dat het cluster continu met die gewenste toestand in overeenstemming blijft.

De stageopdracht omvat het volledige traject van architectuurontwerp tot werkende implementatie. Concreet gaat het om het definiëren van een doelarchitectuur, het vastleggen van onderbouwde ontwerpkeuzes via Architecture Decision Records (ADR's), het opzetten van een Kubernetes-cluster met alle ondersteunende platformcomponenten en het beschrijven van een golden path waarmee ontwikkelteams op een gestandaardiseerde manier applicaties kunnen deployen.

Het beoogde eindresultaat is een werkend on-premises Kubernetes-platform met geautomatiseerde CI/CD-pipelines, observability, beveiligingsbeleid en een onboarding-ervaring voor ontwikkelaars. Daarnaast levert de stage een uitgebreid documentatiepakket op, die bestaat uit runbooks, ADR's, standaardisatierichtlijnen en een service-onboarding-gids, deze documentatie kan Axxes hergebruiken als referentiemateriaal in klantopdrachten en interne trajecten.

Dit realisatiedocument beschrijft de analyse, het ontwerp en de daadwerkelijke realisatie van het platform. Hoofdstuk 2 behandelt de analyse van de vereisten en de probleemstelling. Hoofdstuk 3 beschrijft het architectuurontwerp en de gemaakte ontwerpkeuzes. Hoofdstuk 4 documenteert de realisatie van alle platformcomponenten. Hoofdstuk 5 beschrijft de lessons learned en hoofdstuk 6 sluit af met een besluit.

2. Analyse

2.1. Probleemstelling en context

Bij Axxes worden consultants ingezet bij uiteenlopende klantprojecten. Daarbij worden Kubernetes-platformen regelmatig vanaf nul opgezet, vaak met wisselende toolkeuzes, uiteenlopende configuratiestandaarden en handmatige stappen. Dit leidt tot een aantal concrete knelpunten:

1. **Inconsistente onboarding en oplevering**: De manier waarop nieuwe services worden uitgerold verschilt per team en per opdracht. Handmatige configuratiestappen verhogen de kans op fouten en maken het moeilijk om opstellingen te reproduceren.
2. **Onvoldoende standaardisatie**: Infrastructure-as-Code-patronen, deploymentworkflows, beveiligingsbeleid en observability worden niet structureel vastgelegd en gebruikt. Dit veroorzaakt verschil in configuraties en bemoeilijkt troubleshooting.
3. **Ontbreken van herbruikbare patronen**: Er is geen centraal referentieplatform dat collega's kunnen raadplegen voor bewezen architectuurkeuzes, templates of operationele richtlijnen.

De primaire waarde van deze stage voor Axxes ligt daarom niet in het één-op-één overnemen van het gebouwde platform, maar in de leeropbrengst en de herbruikbare patronen. De expliciete ontwerpkeuzes vastgelegd in ADR's, een golden path met bijhorende templates en operationele richtlijnen die selectief toepasbaar zijn in toekomstige projecten.

Door het ontbreken van deze standaardisatie ontstaat er een onnodig hoge Cognitive Load bij de ontwikkelteams. Ontwikkelaars moeten immers naast hun applicatiecode ook diepgaande kennis opbouwen over Kubernetes, netwerkconfiguraties, pipelines of cloudproviders zoals Azure of AWS. Door het bouwen van dit platform plannen we de Developer Experience (DX) aanzienlijk te verbeteren en ontwikkelaars een simpele weg naar productie te bieden.

2.2. Functionele vereisten

Op basis van de kickoff en de afstemming met de stagebegeleider zijn de volgende functionele vereisten voor het IDP gedefinieerd:

VEREISTE	BESCHRIJVING
GEAUTOMATISEERDE CLUSTERPROVISIONING	Het volledige Kubernetes-cluster moet vanaf nul opgebouwd kunnen worden via één geautomatiseerde pipeline, zonder handmatige stappen op individuele nodes.
GITOPS-GEBASEERD DEPLOYMENTMODEL	Alle applicaties en platformconfiguraties worden beheerd vanuit Git volgens de Everything as Code-filosofie. Een GitOps-operator synchroniseert de gewenste toestand continu naar het cluster.
GECENTRALISEERD GEHEIMENBEHEER	Secrets worden opgeslagen in een dedicated secrets-engine en automatisch gesynchroniseerd naar Kubernetes, zonder dat secrets in Git-repositories terechtkomen.
OBSERVABILITY-STACK	Het platform biedt metrics, logging, dashboarding en alerting out-of-the-box.
BEVEILIGINGSBELEID EN TOEGANGSBEHEER	Gebruikersauthenticatie verloopt via Active Directory. Rollen en rechten worden afgedwongen via RBAC op Kubernetes- en applicatieniveau.
GOLDEN PATH VOOR ONTWIKKELAARS	Ontwikkelteams kunnen via een enkel declaratief bestand (platform.yaml) een applicatie deployen, inclusief ingress, database, secrets en monitoring.
PROJECTBEHEER VIA PIPELINE	Het aanmaken van een nieuw team/project: inclusief namespace, ArgoCD-project, Vault-secrets-engine, Harbor-project en netwerksegmentatie: gebeurt geautomatiseerd met pipelines/API/Frontend.
CONTAINER REGISTRY	Het platform beschikt over een eigen container-registry voor het opslaan van Docker-images en Helm-charts, inclusief het spiegelen van upstream-images.
PERSISTENT STORAGE	Stateful workloads beschikken over gedistribueerde, gerepliceerde opslag met geautomatiseerde snapshot- en backupmechanismen.
CI/CD-PIPELINE VOOR ONTWIKKELAARS	Een standaardpipeline verzorgt het bouwen, scannen (code-kwaliteit en vulnerabilities), verpakken en deployen van applicaties. Dit ondersteunt de Shift-Left Security-beweging door security-kwetsbaarheden (vulnerabilities) al in de pipeline te detecteren vóór de deployment.

Tabel 1: Functionele vereisten van het Internal Developer Platform

2.3. Niet-functionele vereisten

Naast de functionele eisen gelden de volgende niet-functionele vereisten:

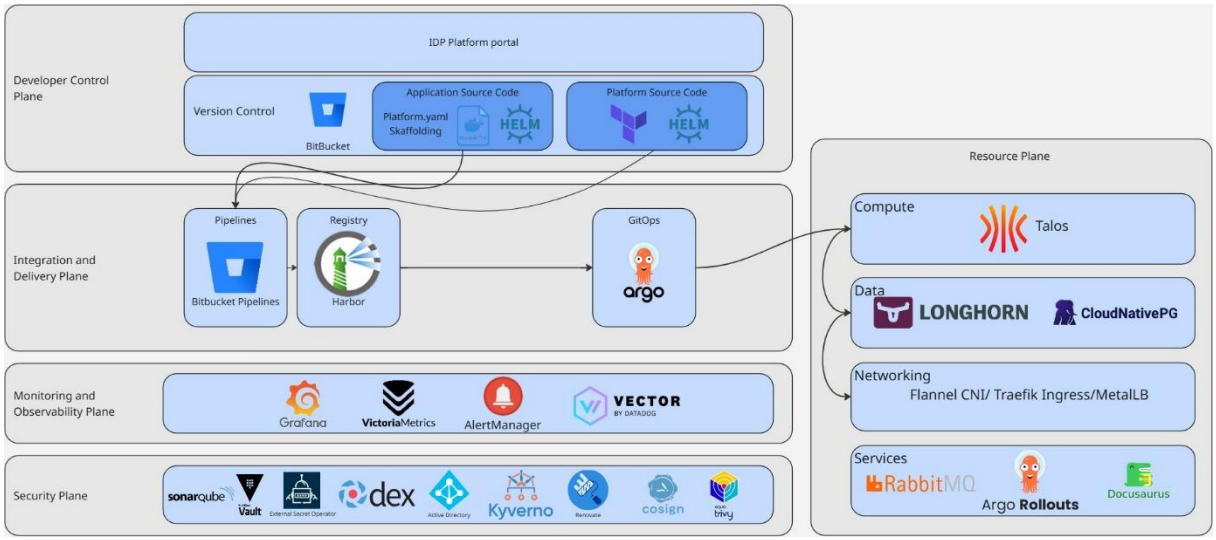
VEREISTE	BESCHRIJVING
HERHAALBAARHEID	Het volledige platform moet reproduceerbaar zijn vanuit versiebeheer.
DECLARATIEF BEHEER	Alle configuratie wordt vastgelegd als code (Terraform, Helm-values, YAML).
BEVEILIGING	Nodes draaien een hardened besturingssysteem. Houd zo goed mogelijk aan de security best practices.
DOCUMENTATIE	Architectuurkeuzes worden onderbouwd in een ADR. Elke operationele handeling wordt beschreven in een runbook. Standaarden voor naamgeving en repository-indeling worden vastgelegd.
SCHAALBAARHEID	Het platform ondersteunt het toevoegen van extra worker-nodes.
VENDOR-ONAFHANKELIJKHEID	Waar mogelijk worden open-source, CNCF-geaffilieerde tools gekozen om vendor-lock-in te minimaliseren.
MINIMAAL RESOURCEGEBRUIK	Toolkeuzes houden rekening met de beperkte resources van de on-premises-omgeving. Lichtgewicht alternatieven krijgen de voorkeur boven zware enterprise-suites.

Tabel 2: Niet-functionele vereisten van het Internal Developer Platform

2.4. Doelarchitectuur: high-level design

Het platform is onderverdeeld volgens vijf vlakken (five-plane model), een architectuurpatroon dat de verantwoordelijkheden van een IDP logisch scheidt in vijf lagen:

1. **Developer Control Plane**: De laag waarmee ontwikkelaars interageren: versiebeheer (Bitbucket), de platform.yaml-specificatie, Helm-charts en het IDP-portaal.
2. **Integration and Delivery Plane**: De laag die code omzet in draaiende applicaties: CI/CD-pipelines (Bitbucket Pipelines), de container-registry (Harbor) en de GitOps-operator (Argo CD).
3. **Monitoring and Observability Plane**: De laag die inzicht biedt in de gezondheid van het platform en de applicaties: metrics (VictoriaMetrics), dashboarding (Grafana), logging (VictoriaLogs + Vector) en alerting (Alertmanager).
4. **Security Plane**: De laag die beveiliging en naleving afdwingt: geheimenbeheer (Vault + External Secrets Operator), authenticatie (Active Directory via Dex/LDAP) en beleidsafdwinging (Kyverno).
5. **Resource Plane**: De onderliggende infrastructuur: compute (Talos Linux op vSphere), opslag (Longhorn), database-operator (CloudNativePG) en netwerking (Flannel CNI, Traefik Ingress, MetalLB).



Figuur 1: Het volledige architectuuroverzicht met alle componenten per vlak

Laag	Tool	Functie
Hypervisor	VMware vSphere	Virtualisatieplatform
OS / Kubernetes	Talos Linux	Immutable OS met ingebouwde Kubernetes
Infrastructure as Code	Terraform	Declaratieve infrastructuurprovisioning
GitOps	Argo CD	Continu afstemmen van cluster op Git
Container Registry	Harbor	Opslag van images en Helm-charts
CI/CD	Bitbucket Pipelines	Build-, scan- en deploypipelines
Secret Management	HashiCorp Vault	Gecentraliseerde geheimensopslag
Secret Sync	External Secrets Operator	Synchronisatie van Vault naar Kubernetes
Observability	VictoriaMetrics + Grafana + VictoriaLogs	Metrics, dashboards en logging
Storage	Longhorn	Gedistribueerde blokopslag
Policy Engine	Kyverno	Beleidsafdwinging via admission control
Database Operator	CloudNativePG	PostgreSQL-operator voor Kubernetes
Progressive Delivery	Argo Rollouts	Gecontroleerde canary- en blue-green-deployments
Secret Reload	Stakater Reloader	Automatische herstart van pods bij wijziging van Secrets of ConfigMaps
Message Queue	RabbitMQ (Cluster Operator)	Asynchrone berichtenwachtrij voor ontwikkelteams
Security Scanning	Trivy Operator	Continu CVE-scanning van alle draaiende workloads

Tabel 3: Overzicht van de platformstack

Voordat er in detail treden getreden wordt over de stack, is er een overzicht van alle tools, weergegeven in Figuur 1 en Tabel 3. Figuur 1 is het conceptuele en logische model, het illustreert de informatiestromen, de tools die gebruikt zijn en hoe de abstracte verantwoordelijkheden verdeeld zijn over de vijf vlakken van het platform. Tabel 3 slaat de brug naar de praktijk, het is de technische invulling (de "BOM" of Bill of Materials) van dit model. Waar de figuur de abstracte functie "Observability" toont in het Monitoring Plane, benoemt Tabel 3 specifiek de gekozen tools: VictoriaMetrics en Grafana. Met een korte uitleg over de functie en doel van deze tools. Samen bieden ze een compleet beeld: de figuur legt uit hoe en waarom het systeem in fasen is opgedeeld en de tabel geef aant waarmee deze fasen daadwerkelijk gebouwd zijn.

3. Ontwerp

Dit hoofdstuk beschrijft het architectuurontwerp van het Internal Developer Platform en onderbouwt de gemaakte technologiekeuzes. Elke keuze wordt voorafgegaan door een vergelijking van relevante alternatieven, beoordeeld aan de hand van de Weighted Ranking Methode (WRM) en afgesloten met een motivatie. Daarnaast worden de standaardisatierichtlijnen beschreven die het platform consistent en onderhoudbaar houden.

3.1. Architectuuroverzicht

Het platform volgt het vijf-vlakken-model (five-plane model) zoals te zien in Figsuur 1, een architectuurpatroon dat de verantwoordelijkheden van een Internal Developer Platform logisch scheidt. Figuur 1 toont het volledige overzicht. De vijf vlakken zijn:

1. **Developer Control Plane**: de laag waarmee ontwikkelaars rechtstreeks interageren. Hier bevinden zich het versiebeheersysteem (Bitbucket), de platform.yaml-specificatie waarmee teams hun applicatie declaratief beschrijven, Helm-charts voor het verpakken van applicaties en op termijn een IDP-portaal voor self-service.
2. **Integration and Delivery Plane**: de laag die broncode omzet in draaiende applicaties. Bitbucket Pipelines verzorgen het bouwen, scannen en verpakken van container-images. Harbor slaat deze images en Helm-charts op. Argo CD synchroniseert de gewenste toestand vanuit Git naar het cluster via een GitOps.
3. **Monitoring and Observability Plane**: de laag die inzicht biedt in het gedrag en de gezondheid van zowel het platform als de applicaties die erop draaien. VictoriaMetrics verzamelt metrics, Grafana visualiseert deze in dashboards, VictoriaLogs in combinatie met Vector verzorgt gecentraliseerde logging en Alertmanager stuurt meldingen naar Microsoft Teams wanneer drempelwaarden overschreden worden.
4. **Security Plane**: de laag die beveiliging en naleving afdwingt over het gehele platform. HashiCorp Vault fungeert als gecentraliseerde secret engine met per-project isolatie. External Secrets Operator (ESO) synchroniseert Vault-secrets naar Kubernetes Secrets zodat applicaties er transparant gebruik van kunnen maken. Dex koppelt Argo CD, kubectl en andere tools aan Active Directory via OIDC/LDAP. Kyverno dwingt beleid af als admission controller, waaronder het automatisch genereren van standaard-NetworkPolicies en het valideren van verplichte labels.
5. **Resource Plane**: de onderliggende infrastructuurlaag. Talos Linux draait als immutable besturingssysteem op VMware vSphere en levert upstream Kubernetes. Longhorn voorziet in gedistribueerde blokopslag met replicatie en backups. CloudNativePG beheert PostgreSQL-databases als Kubernetes-native operator. Flannel verzorgt het pod-netwerk (CNI), Traefik fungeert als ingress-controller en MetalLB biedt load balancing via Layer 2 in de on-premises-omgeving.

Elk vlak kan onafhankelijk evolueren: een wijziging in de observability-stack heeft geen impact op het security-vlak en een aanpassing aan de storage-laag vereist geen wijziging in de developer-facing tooling. Dit maakt het platform modulair en onderhoudbaar.

3.2. Ontwerpkeuzes

In deze sectie worden de belangrijkste technologiekeuzes beschreven. Per component wordt eerst de context geschetst, vervolgens worden de overwogen alternatieven vergeleken en ten slotte wordt de keuze gemotiveerd.

3.2.1. Compute: Talos Linux

Context en rol:
Het besturingssysteem en de Kubernetes-distributie vormen het fundament van het platform. De keuze heeft directe impact op beveiliging, operationeel beheer en reproduceerbaarheid.

Vergelijking van alternatieven

Dimensie	RKE2	K3s	Talos Linux	VMware Tanzu	OpenShift
Type	Geharde upstream K8s-distributie	Ultralichte CNCF-gecertificeerde K8s (één binair bestand)	Immutable, API-gestuurd Linux OS met upstream K8s	Enterprise K8s geïntegreerd in vSphere	Volledig enterprise K8s-platform (Red Hat)
Node-OS	Zelf te kiezen Linux	Zelf te kiezen Linux	Talos Linux (immutable, read-only)	Zelf te kiezen (RHEL/Ubuntu/Photon)	RHCOS (immutable)
SSH-toegang	Ja	Ja	Nee - alleen via talosctl API	Ja	Beperkt (debug-pods)
Container-runtime	containerd	containerd	containerd	containerd	CRI-O
Standaard-CNI	Canal (Calico/Flannel)	Flannel	Geen - vrij te kiezen	Antrea	OVN-Kubernetes
GitOps-vriendelijkheid	Matig - configuratie deels imperatief	Matig	Hoog - machineconfiguratie is YAML in versiebeheer	Matig	Matig
Operationele complexiteit	Rancher vereenvoudigt beheer	Laagste complexiteit	API-first, geen SSH, steile leercurve	Hoog - vereist vSphere-expertise	Hoog - eigen operator-ecosysteem
Productiegeschiktheid	Hoog	Beperkt (SQLite standaard, geen multi-tenancy)	Hoog - immutable ontwerp	Hoog	Hoog
Schaalbaarheid	Goed voor grote clusters	Beperkt voor grote clusters	Goed - schaalt via Cluster API	Goed - gebruikt vSphere-schaal	Goed
Licentie	Open source	Open source	Open source	Commercieel (Broadcom)	Commercieel (Red Hat) / OKD open source

Tabel 4: Vergelijking van Kubernetes-distributies

Beoordeling

Criterium	Toelichting
Beveiliging	Talos en OpenShift bieden een immutable OS zonder SSH, wat de aanvalsvector drastisch verkleint. RKE2, K3s en Tanzu draaien op een conventioneel Linux-OS met volledige shell-toegang.
Reproduceerbaarheid	Talos-machineconfiguraties zijn volledig declaratief in YAML en leven in Git. Dit sluit naadloos aan bij de GitOps-werkwijze van het platform. Bij RKE2 en K3s is de node-configuratie deels imperatief.
Operationele complexiteit	K3s is het eenvoudigst voor dag-2-beheer, maar mist productiefeatures. Talos vereist een aanpassing in werkwijze (geen SSH, alles via API), maar levert in ruil volledige reproduceerbaarheid. Tanzu en OpenShift brengen een groot ecosysteem mee dat buiten scope valt voor een enkel cluster.
Feature-fit	Voor één productie-achtig cluster is Talos doelgericht en lean. RKE2 blinkt uit in multi-clusterbeheer via Rancher, wat hier niet nodig is. Tanzu en OpenShift bevatten platformsuites die buiten scope vallen.
Licentie	Talos, RKE2 en K3s zijn volledig open source. Tanzu valt onder Broadcom-licenties en OpenShift onder Red Hat-commerciële voorwaarden, wat niet past bij de voorkeur voor vendor-onafhankelijkheid.

Keuze: Talos Linux

Talos Linux is gekozen vanwege de combinatie van een immutable, read-only besturingssysteem zonder SSH-toegang, een volledig API-gestuurd levenscyclusbeheer via talosctl en de nauwe aansluiting bij GitOps-principes doordat alle machineconfiguratie in YAML-bestanden in versiebeheer leeft. Talos draait ongewijzigd upstream Kubernetes, waardoor er geen distributiespecifieke abstracties of lock-in ontstaan. Het compacte OS laat maximale resources over voor workloads.

3.2.2. Container Registry: Harbor

Context en rol
Het platform heeft een interne container-registry nodig voor het opslaan van Docker-images en Helm-charts. Daarnaast moet de registry upstream-images kunnen spiegelen zodat ontwikkelaars uitsluitend vanuit de interne registry pullen, wat zowel de beveiliging als de onafhankelijkheid van externe bronnen ten goede komt.

Vergelijking van alternatieven

Dimensie	Harbor	Zot	AWS ECR	Azure ACR	Bitbucket Packages
Type	Zelfgehost, volledige registry met UI	Zelfgehost, minimale OCI-registry	Beheerde AWS-cloudservice	Beheerde Azure-cloudservice	Atlassian SaaS
Vulnerability scanning	Ingebouwd (Trivy/Clair), blokkering van kwetsbare images	Geen ingebouwde scanner	Via AWS Inspector (apart in te schakelen)	Via Microsoft Defender (apart)	Geen
Helm-chart-opslag	Ja - OCI-artefacten en ChartMuseum	Ja - OCI-compliant	Ja - OCI	Ja - OCI	Nee
Proxy cache / spiegeling	Ja - replicatieregels en proxy-cache	Beperkt	Nee	Nee	Nee
RBAC en multi-tenancy	Projectgebaseerde RBAC, LDAP/OIDC-integratie	Basis (htpasswd, LDAP)	IAM-gebaseerd	AAD-gebaseerd	Workspace-tokens
Image signing	Cosign / Notary v2	Cosign	Niet standaard	Notary v2	Nee
Retentiebeleid	Ja - tag-retentieregels per project	Basis	Ja - levenscyclusregels	Ja	Nee
Air-gap-geschiktheid	Volledig - zelfgehost, geen externe afhankelijkheden	Volledig - één binair bestand	Nee - vereist internetverbinding	Nee - vereist internetverbinding	Nee - SaaS
Resourcegebruik	Matig - vereist PostgreSQL, Redis en opslagbackend	Zeer laag - één binair bestand, lokaal bestandssysteem	Geen on-prem-resources	Geen on-prem-resources	Geen on-prem-resources
Kubernetes-integratie	Pull secrets, Argo CD Image Updater, imagebeleidautomatisering	Standaard Kubernetes-tooling	Vereist cloudspecifieke credential-helpers	Vereist workload identity	Handmatig secretbeheer

Tabel 5: Vergelijking van container-registries

Beoordeling

Criterium	Toelichting
Feature-volledigheid	Harbor is de meest complete optie: vulnerability scanning, proxy cache, Helm-charts, OCI-artefacten, replicatie, retentiebeleid en uitgebreide RBAC in één tool. Zot is licht en spec-compliant maar mist een volwaardige UI en ingebouwde scanning. Bitbucket Packages is het meest beperkt.
Beveiliging	Harbor biedt ingebouwde Trivy-scanning met de mogelijkheid om het pullen van kwetsbare images te blokkeren, een directe bijdrage aan de security-by-default-doelstelling.
Setup en beheer	Zot is het eenvoudigst. Harbor vereist meer componenten maar heeft een volwassen Helm-chart en goede documentatie. Cloudregistries vereisen weinig setup maar passen niet in het on-premises-model.
Kubernetes- en Argo CD-integratie	Harbor integreert naadloos met Kubernetes pull secret. Cloudregistries vereisen cloudspecifieke credential-helpers die complexiteit toevoegen in een zelfgehoste omgeving.

Keuze: Harbor
Harbor is gekozen als de meest complete zelfgehoste registry die vulnerability scanning, proxy cache, Helm-chart-opslag, projectgebaseerde RBAC met LDAP-integratie combineert in één tool. De ingebouwde Trivy-scanner en de mogelijkheid om kwetsbare images te blokkeren sluiten direct aan bij de security-by-default-doelstelling van het platform.

3.2.3. GitOps: Argo CD

Context en rol
GitOps is het kernprincipe van het platform: de gewenste toestand van alle applicaties en configuraties wordt beschreven in Git en een operator zorgt ervoor dat het cluster continu met die toestand in overeenstemming blijft. De keuze van de GitOps-operator bepaalt hoe applicaties worden uitgerold, hoe multi-tenancy wordt ingericht en welke zichtbaarheid het platformteam en ontwikkelteams hebben.

Vergelijking van alternatieven

Dimensie	Flux	Argo CD	Rancher Fleet
Type	CNCF graduated GitOps-toolkit (modulaire controllers)	CNCF graduated declaratieve GitOps-operator met ingebouwde web-UI	Multi-cluster GitOps-agent geïntegreerd in Rancher
Architectuur	Set van modulaire controllers, geen enkel binair bestand	Eén server + Redis + application controller + repo server	Fleet manager + clusteragenten
Web-UI	Geen ingebouwde UI (Weave GitOps is een betaalde add-on)	Eersteklas ingebouwde UI - applicatiediffs, gezondheid, synchronisatiegeschiedenis	Basis Rancher-UI-integratie
Multi-app-beheer	Kustomization en HelmRelease-resources; geen ApplicationSet-equivalent	ApplicationSet-controller - templated multi-app vanuit één bron	Fleet Bundle en GitRepo-resources
RBAC en multi-tenancy	Kubernetes RBAC op Flux CRD's; beperkte ingebouwde multi-tenancy	ArgoCD Projects met rolgebaseerd beleid, OIDC/LDAP-integratie	Rancher workspace-RBAC
OIDC/LDAP-authenticatie	Nee (leunt op K8s RBAC, geen login-UI)	Ja - Dex OIDC-integratie voor UI-login met groepgebaseerde RBAC	Via Rancher SAML/OIDC
Helm-ondersteuning	Native HelmRelease-CRD	Native Helm-rendering; Argo CD beheert release-levenscyclus	Helm via Fleet Bundles
Secretbeheer	Native SOPS-integratie met age/GPG	Vereist Argo CD Vault Plugin (AVP) of ESO voor Vault-secrets	Rancher Secrets-integratie
Multi-cluster	Ja - Kustomization kan remote clusters targeten	Ja - Argo CD beheert meerdere clustercontexten	Best-in-class - gebouwd voor grote vlootbeheer

Tabel 6: Vergelijking van GitOps-operators

Beoordeling

Criterium	Toelichting
Gebruiksgemak en UI	De ingebouwde UI van Argo CD is een doorslaggevend voordeel voor platformzichtbaarheid: live diffs, gezondheidsstatus en synchronisatiegeschiedenis zonder aanvullende tooling. Flux heeft geen ingebouwde UI; Rancher Fleet's UI is beperkt.
Multi-tenancy en RBAC	Argo CD Projects met OIDC/LDAP-integratie mappen rechtstreeks op AD-groepen en teamnamespaces. Flux leunt uitsluitend op Kubernetes RBAC, wat meer configuratie vereist voor hetzelfde resultaat.
Multi-app-beheer	Argo CD's ApplicationSet-controller maakt het mogelijk om vanuit één ApplicationSet-resource meerdere applicaties templated uit te rollen. Dit patroon wordt intensief gebruikt in het platform (cluster-apps en bootstrap-apps ApplicationSets).
Secretbeheer-integratie	Flux heeft native SOPS/age-integratie, wat eenvoudiger is dan Argo CD's afhankelijkheid van AVP of ESO. Aangezien dit platform echter ESO met Vault gebruikt, is het SOPS-voordeel van Flux niet relevant.

Keuze: Argo CD
Argo CD is gekozen vanwege de combinatie van een ingebouwde web-UI met live synchronisatiestatus, de ApplicationSet-controller voor templated multi-app-beheer en de native OIDC/LDAP-integratie via Dex die directe koppeling met Active Directory mogelijk maakt. ArgoCD Projects bieden projectgebaseerde RBAC die naadloos aansluit bij het multi-tenancy-model van het platform: elke team krijgt een eigen project met afgebakende rechten.

3.2.4. Observability: VictoriaMetrics, Grafana en Alertmanager

Context en rol
Het platform moet out-of-the-box inzicht bieden in de gezondheid van zowel de infrastructuur als de applicaties die erop draaien. Dit omvat het verzamelen van metrics, het visualiseren in dashboards, het centraliseren van logs en het versturen van waarschuwingen wanneer drempelwaarden worden overschreden.

Vergelijking van alternatieven

Dimensie	Prometheus + Grafana (kube-prometheus-stack)	VictoriaMetrics + Grafana	Datadog	OpenTelemetry Collector
Type	OSS metrics + dashboarding suite	OSS metrics-engine met Prometheus-compatibiliteit + Grafana	Commercieel SaaS observability-platform	OSS telemetrie-pipeline (vendor-neutraal)
Metrics-verzameling	Prometheus scraping	VictoriaMetrics scraping (Prometheus-compatibel)	Datadog Agent per node	OTLP, Prometheus en andere receivers
Opslag	Prometheus TSDB (lokaal)	VictoriaMetrics TSDB (efficiënter)	Volledig beheerd (cloud)	Geen - pipeline, geen opslag
Dashboarding	Grafana (ingebouwd in stack)	Grafana (apart of ingebouwd)	Ingebouwde dashboards	Geen - vereist backend
Alerting	Alertmanager	Alertmanager (compatibel)	Ingebouwde monitors	Geen eigen alerting
Schaalbaarheid	Moeizaam bij hoge kardinaliteit zonder Thanos/Mimir	Uitstekend - verwerkt hoge kardinaliteit met een fractie van de resources	Volledig beheerd, schaalt transparant	Zeer schaalbaar als stateless pipeline
Resourcegebruik	Matig tot hoog - Prometheus, Alertmanager, Grafana, exporters	Laag tot matig - 5–10× minder RAM en schijf dan Prometheus voor equivalente data	Agent-gebaseerd; opslag en verwerking off-cluster	Zeer licht - stateless
Kosten	Gratis (OSS) - alleen infrastructuurkosten	Gratis (OSS) - lagere infrastructuurkosten door efficiëntie	Abonnement per host + feature-add-ons (APM, Log Management)	Gratis (OSS)
HA-configuratie	Complex - vereist Thanos of Mimir	Eenvoudig - ingebouwde replicatie	Volledig beheerd	N.v.t. (stateless)

Tabel 7: Vergelijking van observability

Beoordeling

Criterium	Toelichting
Kosten en licentie	Prometheus, VictoriaMetrics en de OTel Collector zijn alle drie open source zonder licentiekosten. VictoriaMetrics bespaart daarnaast op infrastructuurkosten dankzij zijn efficiëntie. Datadog is een SaaS-abonnement dat snel kan groeien met clustergrootte.
Operationele complexiteit	Datadog scoort het hoogst op gemak: de backend is volledig beheerd. VictoriaMetrics heeft een eenvoudiger HA-verhaal dan Prometheus (geen Thanos/Mimir nodig). Prometheus vereist meer aandacht voor retentie en schaling.
Setup-snelheid	Datadog is het snelst naar productie. VictoriaMetrics + Grafana omvat minder componenten dan de volledige kube-prometheus-stack. De OTel Collector vereist het plannen van een volledige pipeline-integratie voordat er iets zichtbaar is.
Schaalbaarheid en prestatie	VictoriaMetrics en Datadog verwerken beide hoge ingestieratio's. Prometheus vereist Thanos of Mimir om voorbij één instantie te schalen.

Criterion	Weight	Prometheus + Grafana	VictoriaMetrics + Grafana	Datadog	OTel Collector
Cost & Licensing	25	5	5	1	5
Operational Complexity	20	3	4	5	2
Kubernetes Integration	15	5	4	5	3
Feature Completeness	15	4	3	5	1
Ease of Setup	10	3	4	5	2
Scalability & Performance	10	3	5	5	3
Community & Ecosystem	5	5	3	4	4
Weighted Total	100%	4,05	4,15	3,95	2,95

Keuze: VictoriaMetrics + Grafana + Alertmanager
VictoriaMetrics scoorde het hoogst in de WRM (4,15/5,00), boven Prometheus + Grafana (4,05). De belangrijkste voordelen zijn het significant lagere resourcegebruik, de eenvoudigere HA-configuratie zonder externe componenten als Thanos of Mimir en de volledige Prometheus-compatibiliteit waardoor bestaande dashboards en alertregels ongewijzigd kunnen worden overgenomen.
Voor logging wordt VictoriaLogs ingezet als opslagbackend in combinatie met Vector als log-shipping agent. Alertmanager stuurt meldingen naar Microsoft Teams via een webhook-integratie.

3.2.5. Secret Management: HashiCorp Vault + External Secrets Operator

Context en rol
Secrets, zoals databasewachtwoorden, API-sleutels en TLS-certificaten mogen nooit in Git-repositories terechtkomen. Het platform heeft een gecentraliseerde secrets engine nodig die secrets per project isoleert en secrets automatisch synchroniseert naar Kubernetes.

Vergelijking van alternatieven

Dimensie	HashiCorp Vault	External Secrets Operator (ESO)	Sealed Secrets
Type	Gecentraliseerde secrets-engine met beleidsafdwinging	Kubernetes-operator die secrets synchroniseert vanuit externe backends	Kubernetes-controller die versleutelde secrets decrypteert in het cluster
Encryptie	Secrets versleuteld at-rest in Vault's opslagbackend; TLS in transit	Geen eigen encryptie - afhankelijk van de backend	Asymmetrische encryptie (RSA-OAEP); publieke sleutel versleutelt, clusterprivésleutel ontsleutelt
Toegangsbeheer	Rijk beleidsysteem; ACL-policies per pad/methode; identity-based auth (K8s, LDAP, OIDC, AppRole)	K8s RBAC op ExternalSecret-CRD's; backend controleert secrettoegang	K8s RBAC op SealedSecret- en Secret-CRD's
Auditlogging	Volledige auditlog van elke secret-lees/schrijfactie; configureerbare auditbackends	Kubernetes-events en controllerlogboeken; geen dedicated audittrail	Alleen Kubernetes-events
Secret-injectie	Env vars / volumebestanden via Vault Agent; VSO synchroniseert naar K8s Secrets	K8s Secrets (env vars / volumemounts) - standaard Pod-spec	K8s Secrets (env vars / volumemounts) - standaard Pod-spec
Leercurve	Hoog - Vault-concepten (authmethoden, secret-engines, leases, policies, tokens) vereisen tijd	Laag tot matig - K8s-native CRD's, bekend voor platformteams	Laag - kubeseal-CLI + enkele CRD's
Per-project-isolatie	Ja - aparte secret-engines per project met eigen policies	Ja - via aparte SecretStores per namespace	Beperkt - geen native projectisolatie
Dynamische secrets	Ja - databasecredentials, cloud-credentials on-demand	Nee - synchroniseert statische secrets	Nee

Tabel 8: Vergelijking van secretbeheeroplossingen

Criterion	Weight	HashiCorp Vault	ESO	Sealed Secrets
Cost & Licensing	15%	3	5	5
Operational Complexity	20%	2	4	5
GitOps / Argo CD Integration	20%	2	4	5
Security & Compliance	15%	5	4	3
Ease of Setup	10%	1	4	5
Secret Rotation & Dynamic Secrets	10%	5	3	1
Developer Experience	10%	3	4	4
Weighted Total	100%	2,9	4,05	4,2

Beoordeling en evolutie
Sealed Secrets was aanvankelijk de winnaar uit de initiële WRM-beoordeling (Bijlage 2) en werd kort gebruikt tijdens de eerste platformopbouw. Na verdere evaluatie is **HashiCorp Vault** gekozen als runtime secrets-engine, gecombineerd met **External Secrets Operator (ESO)** om Vault-secrets te synchroniseren naar standaard Kubernetes Secrets.
De redenen voor deze verschuiving zijn:

1. **Per-project-isolatie**: Vault biedt aparte KV v2 secret-engines per project met eigen RBAC-policies, wat aansluit bij het multi-tenancy-model van het platform. Sealed Secrets biedt deze granulariteit niet.
2. **Auditlogging**: Vault registreert elke lees- en schrijfactie op secrets, wat essentieel is voor compliance en troubleshooting. Sealed Secrets levert alleen Kubernetes-events.
3. **Active Directory-integratie**: Vault's LDAP-authmethode maakt het mogelijk om AD-groepen rechtstreeks te mappen op Vault-policies, consistent met de AD-integratie in Argo CD, Grafana en Harbor.
4. **Kubernetes-authenticatie**: Applicatiepods en ESO kunnen via een Kubernetes service account secrets lezen zonder wachtwoorden, met een TTL van één uur per token.

Keuze: HashiCorp Vault + ESO
Vault draait in HA-modus met drie replica's en geïntegreerde Raft-opslag. Elke replica heeft een Longhorn-PVC van 10 GiB. ESO synchroniseert secrets vanuit Vault naar Kubernetes Secrets per projectnamespace met een verversingsinterval van één minuut.

3.2.6. Storage: Longhorn

Context en rol
Stateful workloads op het platform: waaronder Vault, Harbor, CloudNativePG-databases en VictoriaMetrics hebben persistent storage nodig die gedistribueerd, gerepliceerd en geautomatiseerd beheerd wordt. De storage-oplossing moet naadloos integreren met Kubernetes via de Container Storage Interface (CSI) en snapshot- en backupmechanismen bieden.

Vergelijking van alternatieven

Dimensie	Rook/Ceph	Longhorn	OpenEBS	NFS
Type	K8s-orchestrator die een Ceph-cluster beheert - blok, bestand en objectopslag	CNCF-incubating gedistribueerde blokopslag, native voor K8s	CNCF graduated K8s-native opslag met meerdere engines	Netwerkbestandssysteem - gedeeld bestandssysteem via centraal server
Opslagtypen	Blok (RBD), bestand (CephFS), object (RGW/S3)	Blok (CSI)	Blok (Mayastor, Jiva, cStor, LocalPV)	Bestand (RWX)
Replicatie	Ceph-replicatie over OSD's	Cross-node-replicatie (configureerbaar per volume)	Afhankelijk van engine (Mayastor: NVMe-oF, Jiva: iSCSI)	Geen - single point of failure
Snapshots en backups	Ceph RBD-snapshots; S3-backup via Velero	Native snapshots + S3-backup (RustFS)	Engine-afhankelijk	Geen native K8s-snapshots
RWX (ReadWriteMany)	Ja - CephFS	Experimenteel	Beperkt	Ja - native
UI	Rook-dashboard (Ceph Manager)	Ingebouwde management-UI	Beperkt	Geen
Setup-complexiteit	Hoog - vereist dedicated OSD-nodes, schijflabeling, Ceph-bootstrapping	Laag - één Helm-chart	Varieert per engine	Laag (als server al bestaat)
Prestatie	Uitstekend bij juiste tuning (RBD)	Solide voor de meeste workloads	Mayastor excellent (NVMe-oF); Jiva/cStor matig	Afhankelijk van netwerk; ongeschikt voor latentiegevoelig werk
Resourcegebruik	Hoog - Ceph-daemons op elke OSD-node	Matig	Varieert	Laag (off-cluster)

Tabel 9: Vergelijking van storage-oplossingen

Beoordeling

Criterium	Toelichting
Feature-volledigheid	Rook/Ceph is als enige optie in staat blok, bestandssysteem (RWX) en S3-compatibele objectopslag vanuit één systeem te leveren. Longhorn dekt blokopslag goed af maar heeft alleen experimentele RWX. OpenEBS focust op blokopslag. NFS biedt alleen RWX-bestandsopslag.
Setup-gemak	Longhorn is het snelste pad naar productiewaardige gedistribueerde opslag: één Helm-chart en het draait. NFS is snel als er al een server beschikbaar is. Rook/Ceph vereist dedicated OSD-nodes, zorgvuldige schijfconfiguratie en Ceph-clusterbootstrapping.
Kubernetes-integratie	Longhorn is doelgericht gebouwd voor Kubernetes met volledige CSI-compliance, native snapshot/backup-CRD's en een management-UI die diep geïntegreerd is met de Kubernetes API.
Prestatie	OpenEBS Mayastor leidt voor raw block I/O via NVMe-oF. Rook/Ceph RBD presteert uitstekend bij correcte tuning. Longhorn is solide voor de meeste workloads en voldoende voor de verwachte belasting op dit platform.
Resourcegebruik	Longhorn is matig in resourcegebruik. Rook/Ceph is zwaar door de Ceph-daemons. Dit is relevant gezien de beperkte resources van de on-premises-omgeving.

Keuze: Longhorn
Longhorn is gekozen vanwege de eenvoudige setup (één Helm-chart), de diepe Kubernetes-integratie met native CSI, snapshots en backups en het beheersbare resourcegebruik dat past bij de beperkte on-premises-omgeving. Longhorn biedt cross-node-replicatie, een ingebouwde management-UI en ondersteuning voor S3-compatibele backups.

3.2.7. Database Operator: CloudNativePG

Context en rol
Ontwikkelteams die via de golden path een applicatie deployen, moeten optioneel een PostgreSQL-database kunnen aanvragen door simpelweg een spec.database-blok toe te voegen aan hun platform.yaml. De database-operator beheert vervolgens het volledige levenscyclusbeheer: provisioning, replicatie, failover en backups.

Keuze: CloudNativePG
CloudNativePG is gekozen als de PostgreSQL-operator voor het platform. De belangrijkste redenen zijn:

- **CNCF-project**: doelgericht gebouwd voor Kubernetes met actieve community en duidelijke roadmap.
- **Declaratieve Cluster-CRD**: instances, storage class, PostgreSQL-versie, backuptarget en monitoring worden in één manifest gedefinieerd. Geen handmatige psql-commando's of shellscripts.
- **Native streaming replicatie**: de operator beheert de primary/replica-topologie automatisch, inclusief failover, zonder externe coordinator.
- **Longhorn-PVC-integratie**: elke PostgreSQL-instance krijgt een Longhorn-backed PVC voor gedistribueerde, gerepliceerde blokopslag. Longhorn-snapshots vangen de databasestatus op volumeniveau.
- **Ingebouwde S3-backup**: CloudNativePG kan WAL-archieven en base-backups pushen naar S3-compatibele opslag, beschikbaar voor toekomstig gebruik naast Longhorn-volumebackups.

3.2.8. Networking: Flannel, Traefik en MetalLB

Context en rol
Het netwerkvlak van het platform bestaat uit drie componenten die elk een specifieke verantwoordelijkheid hebben:

1. **CNI (Container Network Interface)**: verzorgt de pod-to-pod-communicatie binnen het cluster.
2. **Ingress Controller**: routeert extern HTTP/HTTPS-verkeer naar de juiste services.
3. **Load Balancer**: wijst externe IP-adressen toe aan Kubernetes Services van het type LoadBalancer in een on-premises-omgeving zonder cloudprovider.

Keuzes

Component	Gekozen tool	Motivatie
CNI	Flannel	Lichtgewicht, stabiel en voldoende voor het platform. Talos Linux bevat geen standaard-CNI, waardoor een expliciete keuze nodig is. Flannel is eenvoudig te configureren en voldoet aan de netwerkbehoeften van het platform.
Ingress	Traefik	Kubernetes-native ingress-controller met ondersteuning voor IngressRoute-CRD's, automatische TLS via cert-manager, middleware-chains en een ingebouwd dashboard.
Load Balancer	MetalLB	Biedt Layer 2 load balancing voor on-premises-omgevingen waar geen cloudprovider-loadbalancer beschikbaar is. Vereist strictARP op kube-proxy, wat geconfigureerd wordt via een Talos-machineconfiguratiepatch.

Tabel 10: Netwerkcomponenten en motivatie

3.2.9. CI/CD: Bitbucket Pipelines

Context en rol
Axxes gebruikt Bitbucket Data Center als versiebeheersysteem Project. Bitbucket Pipelines is de ingebouwde CI/CD-oplossing die rechtstreeks geïntegreerd is met het versiebeheersysteem. Door Bitbucket Pipelines te gebruiken, ontstaat er een koppeling tussen code-wijzigingen en geautomatiseerde build-, scan- en deploystappen zonder een apart CI/CD-systeem te hoeven beheren.
De pipelines draaien op zelfgehoste runners die in twee VLAN's zijn opgesplitst, een noodzaak die voortkomt uit de netwerksegmentatie van de on-premises-omgeving.

3.3. Standaardisatie

Een consistent platform vereist duidelijke afspraken over naamgeving, repository-indeling en configuratiepatronen. Deze standaarden zijn vastgelegd in drie richtlijndocumenten die als referentie dienen voor zowel het platformteam als ontwikkelteams. Dit was een vereiste van Axxes, van begin uit zoveel mogelijk standaardiseren. Dit geeft een duidelijk overzicht en maakt het project sneller te begrijpen voor anderen.

3.3.1. Git- en Bitbucket-conventies

De Git-standaard beschrijft de afspraken voor repository-indeling, branching, commitberichten en versiebeheer:

- **Commitberichten** volgen het Conventional Commits-formaat: <type>(<scope>): <beschrijving>, gevolgd door een optionele body en footer. Types zijn onder andere feat, fix, docs, refactor en chore.
- **Branch-bescherming** op de main-branch is strikt geconfigureerd: geen directe pushes, verplichte pull requests met goedkeuring, verplicht geslaagde builds, geen force-push en geen branchverwijdering.
- **Versiebeheer** volgt Semantic Versioning (SemVer): MAJOR.MINOR.PATCH. Tags worden aangemaakt bij releases en triggeren waar nodig geautomatiseerde acties.

3.3.2. Kubernetes-naamgeving en -labeling

De Kubernetes-standaard beschrijft de afspraken voor naamgeving van namespaces, resources en labels, evenals het netwerkbeveiligingsbeleid:

- **Elke namespace** start met een default-deny-policy voor zowel ingress als egress. Workloads moeten expliciet het verkeer openen dat ze nodig hebben via gerichte NetworkPolicies.
- **Ingress vanuit de ingress-controller** wordt toegestaan via een specifieke NetworkPolicy die verkeer toelaat vanuit de ingress-namespace naar pods.
- **Labels** volgen de Kubernetes recommended labels (app.kubernetes.io/name, app.kubernetes.io/instance, app.kubernetes.io/version, etc.) en worden afgedwongen door Kyverno.

3.3.3. Terraform-conventies

De Terraform-standaard beschrijft de bestandsstructuur, naamgevingsconventies en tagging-strategie:

- **Bestandsindeling**: elke root-module en child-module volgt een canonieke layout met main.tf, variables.tf, outputs.tf, locals.tf, versions.tf.
- **Tagging-strategie**: alle resources ontvangen verplichte tags via de common_tags-local: environment, project, managed_by.

3.4. Infrastructuurontwerp

Naast de Kubernetes-platformcomponenten omvat het ontwerp een aantal ondersteunende infrastructuurkeuzes die de werking van het platform mogelijk maken.

3.4.1. VLAN-segmentatie en netwerktopologie

De on-premises-omgeving is opgedeeld in meerdere VLAN's om verkeer te scheiden tussen het VMware vCenter-beheernetwerk en het Talos-clusternetwerk. Deze segmentatie heeft directe gevolgen voor de bootstrappipeline: de Terraform-stappen die VM's aanmaken vereisen toegang tot vCenter en draaien daarom op een Windows-runner in het vCenter-VLAN, terwijl de stappen die het cluster bootstrappen en applicaties installeren toegang tot de Talos API nodig hebben en draaien op een Linux-runner in het lab-VLAN.

3.4.2. Active Directory en DNS

Een eigen Active Directory-domein (stage.axxes.local) is opgezet als centrale identiteitsbron voor het platform. Alle platformcomponenten die gebruikersauthenticatie vereisen: Argo CD (via Dex), Grafana, Harbor en Vault, authenticeren tegen dit AD via LDAP.
Een eigen DNS-server zorgt ervoor dat alle interne domeinnamen (zoals argocd.stage.axxes.local, harbor.stage.axxes.local, grafana.stage.axxes.local) correct worden opgelost binnen de on-premises-omgeving.

3.4.3. Bitbucket Runners: Windows/Linux-splitsing

Door de VLAN-segmentatie moet de bootstrappipeline worden opgesplitst over twee runner:

- **Windows-runner** (vCenter-VLAN): deployt VM's vanuit OVA, injecteert Talos-machineconfiguraties en exporteert outputs als pipeline-artefact.
- **Linux-runner** (lab-VLAN): verbindt met de Talos API, bootstrapt etcd, installeert applicaties via Helm en configureert platformcomponenten.

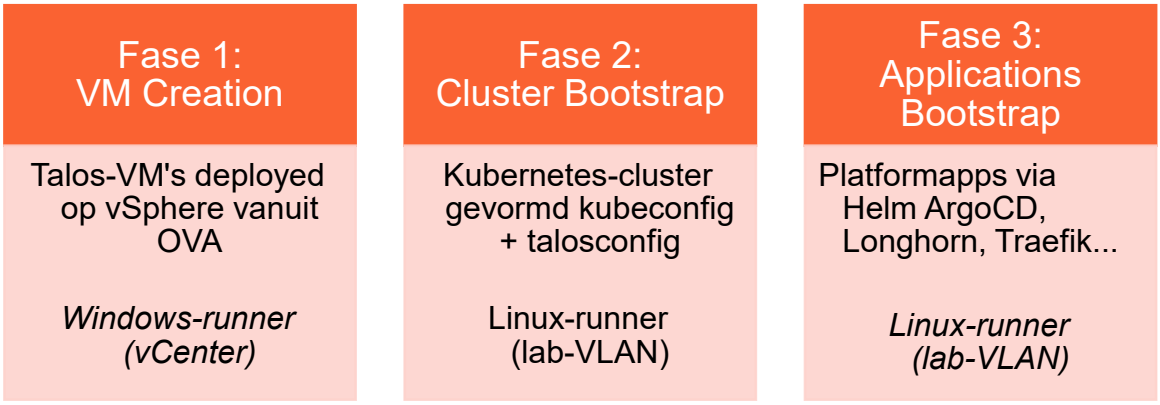
State wordt doorgegeven tussen de runners als een Bitbucket-pipeline-artefact (vm_creation.outputs.json). Als dit artefact ontbreekt of verouderd is, kan de tweede runner niet verdergaan.

4. Realisatie

Dit hoofdstuk beschrijft de effectieve implementatie van het Internal Developer Platform. De opbouw volgt de chronologische volgorde waarin het platform is opgebouwd: eerst de infrastructuurprovisioning en clusterbootstrap, vervolgens de platformdiensten die boven op de cluster draaien en ten slotte de developer-facing golden path en ondersteunende infrastructuur. Waar het relevant is worden kleine configuratiefragmenten opgenomen ter illustratie.

4.1. Platformbootstrap

Het volledige platform wordt opgebouwd in drie sequentiële fasen, aangestuurd door Terraform en georkestreerd via Bitbucket Pipelines



Figuur 2: De drie fasen van de platformbootstrap

Elke fase draait op een dedicated zelfgehoste Bitbucket-runner vanwege de netwerksegmentatie: de vSphere/vCenter-host en het Talos-labcluster bevinden zich op gescheiden VLAN's. Geen enkele runner heeft toegang tot beide netwerken, waardoor de pipeline state overdraagt tussen runners via Bitbucket-artefacten.

4.1.1. Fase 1: VM Creation (Windows-runner, vCenter-VLAN)

In deze fase verbindt Terraform met de vSphere-API en voert de volgende stappen uit:

1. **vSphere-infrastructuur opvragen**: datacenter, cluster, datastore, netwerk en resource pool worden opgelost aan de hand van hun namen in de Terraform-variabelen.
2. **Talos-machinegeheimen genereren**: Terraform genereert een nieuwe set cryptografische geheimen die uniek zijn voor dit cluster: een cluster-CA, een etcd-CA en node-authenticatietokens. Deze geheimen worden nooit handmatig aangemaakt of beheerd.
3. **Machineconfiguratie bouwen voor elke node**: Talos Linux wordt volledig geconfigureerd via een YAML-machineconfiguratie. Boven op de standaardconfiguratie worden vijf patches toegepast die het platform correct laten functioneren:

Patch	Doel
Scheduling-patch	Staat workloads toe op control-plane-nodes, nuttig voor kleinere clusters waar de control-plane-nodes ook als worker fungeren
IPVS-proxy-patch	Schakelt strictARP in op kube-proxy, een vereiste van MetalLB voor Layer 2 load balancing
Kernelmodules-patch	Laadt de modules nbd, iscsi_tcp en configfs bij het opstarten, vereist door Longhorn voor blokopslag
Extensions-patch	Installeert iscsi-tools en util-linux-tools op Talos, aanvullende afhankelijkheden voor Longhorn
Kubelet-patch	Schakelt automatische kubelet-certificaatrotatie in voor verbeterde beveiliging
Kubelet GC-patch	Beperkt schijfgebruik door containerimagecache. Cleanup start bij 60% schijfgebruik, daalt terug naar 40%. Images jonger dan 2 minuten worden overgeslagen; images ouder dan 24 uur worden verwijderd.

Tabel 11: Talos-machineconfiguratiepatches

4. **VM's deployen**: voor elke node wordt een VM aangemaakt op vSphere vanuit een vooraf klaargezet Talos OVA-image. De gegenereerde machineconfiguratie wordt als metadata meegegeven aan de VM, zodat Talos Linux bij het eerste opstarten automatisch geconfigureerd is.
5. **Outputs exporteren**: de IP-adressen van alle nodes, de machineconfiguraties en de clustersecrets worden weggeschreven naar een JSON-bestand (vm_creation.outputs.json). Dit bestand wordt als Bitbucket-pipelineartefact beschikbaar gesteld aan fase 2.

De vSphere-credentials (vsphere_user en vsphere_password) worden nooit in de Terraform-variabelenbestanden opgeslagen. In plaats daarvan worden ze geïnjecteerd via TF_VAR_*-omgevingsvariabelen die als beveiligde variabelen in Bitbucket zijn geconfigureerd.

4.1.2. Fase 2: Cluster Bootstrap (Linux-runner, lab-VLAN)

Deze fase neemt het artefact uit fase 1 als invoer en vormt de losse VM's tot een werkend Kubernetes-cluster:

1. **Artefact inladen**: het vm_creation.outputs.json-bestand wordt gelezen om de node-IP's en de Talos-clientconfiguratie te verkrijgen. Zonder dit artefact kan deze fase niet starten.
2. **etcd bootstrappen**: Terraform roept de resource talos_machine_bootstrap exact één keer aan om het etcd-cluster te initialiseren. etcd is de gedistribueerde key-value store die Kubernetes gebruikt als achterliggende databank. Terraform houdt deze actie bij in zijn state: een tweede apply is een no-op en voert de bootstrap niet opnieuw uit.
3. **Kubeconfig en talosconfig genereren**: deze bestanden zijn nodig om respectievelijk met de Kubernetes-API en de Talos-API te communiceren. Ze worden als pipelineartefact en in de bitucket Downloads menu beschikbaar gesteld voor fase 3 en voor ontwikkelaars die kubectl-toegang nodig hebben.

4.1.3. Fase 3: Applications Bootstrap (Linux-runner, lab-VLAN)

In deze fase installeert Terraform alle platformapplicaties via Helm op het zojuist opgezette cluster. Helm is een pakketbeheerder voor Kubernetes die applicaties verpakt als herbruikbare charts met configureerbare waarden.

De configuratie voor elke applicatie is opgeslagen in afzonderlijke Helm-valuesbestanden:

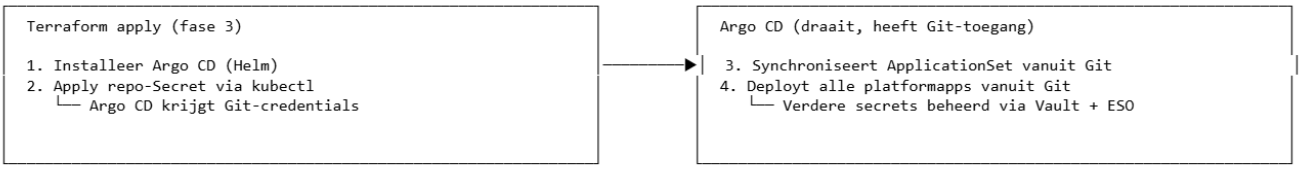
```
values/
├── argocd.yaml              # Argo CD + Dex LDAP-integratie
├── traefik.yaml             # Ingresscontroller
├── harbor.yaml              # Containerregistry
├── observability.yaml       # Grafana + VictoriaMetrics + Alertmanager
├── longhorn.yaml            # Gedistribueerde opslag
├── sealed-secrets.yaml      # Secretversleuteling (bootstrap)
├── cert-manager.yaml        # TLS-certificaatbeheer
└── metallb.yaml             # Bare-metal load balancer
```

Figuur 3: Structuur van de Helm-valuesbestanden

Deze valuesbestanden bevatten omgevingsspecifieke configuratie zoals ingress-hostnamen, Active Directory-endpoints, DNS-instellingen en webhook-URL's voor Microsoft Teams. Ze worden samengevoegd met de standaardwaarden van elke Helm-chart tijdens de installatie.

Het chicken-and-egg-probleem
Een zelfgehost platform kent circulaire afhankelijkheden: veel platformcomponenten hebben andere componenten nodig die zelf nog niet draaien. Argo CD heeft bijvoorbeeld Git-credentials nodig om repositories te klonen, maar die credentials moeten als Kubernetes Secret bestaan voordat Argo CD ze kan gebruiken. Tegelijkertijd beheert Argo CD normaal gesproken zélf alle Secrets via GitOps.

Terraform lost dit op door de bootstrapvolgorde buiten het GitOps-model af te handelen:



Figuur 4: Bootstrapvolgorde om circulaire afhankelijkheden op te lossen

Na fase 3 draait het cluster met alle platformcomponenten en neemt Argo CD het volledige applicatiebeheer over via GitOps. Vanaf dit punt worden wijzigingen aan het platform aangebracht door Git-commits, niet door handmatige terraform apply-commando's.

Configuratiepipeline
Na afloop van de drie bootstrapfasen draait automatisch een configuratiepipeline als child-pipeline.
Deze pipeline:

- Patcht LDAP-credentials en authenticatie-instellingen in Argo CD (via Dex), Grafana en Harbor
- Initialiseert HashiCorp Vault (unsealing, LDAP-authmethode, Kubernetes-authmethode)
- Kan ook onafhankelijk worden getriggerd om wachtwoorden te roteren zonder de volledige bootstrap opnieuw uit te voeren

Elke platformcomponent die gebruikersauthenticatie vereist, gebruikt een dedicated LDAP-serviceaccount:

Service	Authenticatiemechanisme	Beschrijving
Argo CD (Dex)	LDAP: argocd-svc	Serviceaccount waarmee Dex bind-queries uitvoert tegen Active Directory
Grafana	LDAP: Grafana-bind-account	Serviceaccount waarmee Grafana gebruikers authenticeert
Harbor	LDAP: harbor-svc	Serviceaccount waarmee Harbor LDAP-groepen ophaalt voor projecttoegang

Tabel 12 :LDAP-serviceaccounts per platformcomponent

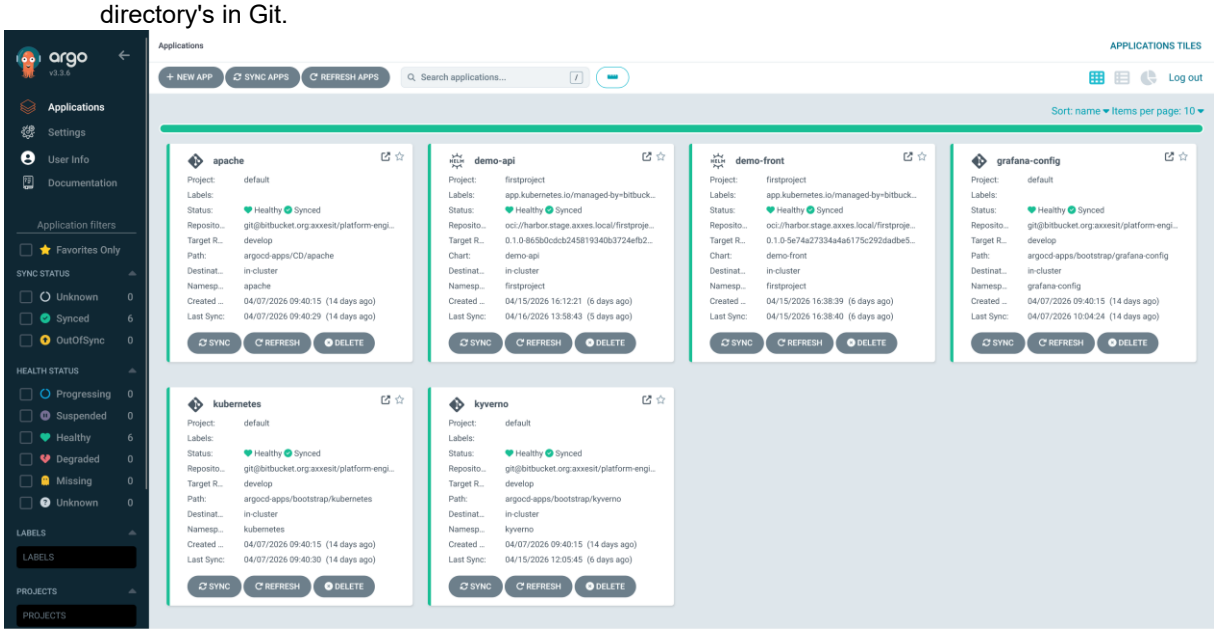
De wachtwoorden van deze serviceaccounts worden opgeslagen als beveiligde variabelen in Bitbucket en nooit in Git.

4.2. GitOps Delivery: Argo CD

4.2.1. ApplicationSet en App-of-Apps-patroon

Na de bootstrap neemt Argo CD het volledige applicatiebeheer over via het **App-of-Apps-patroon**. Dit patroon werkt als volgt: één root-ApplicationSet definieert hoe Argo CD alle subdirectory's in een aangewezen directory moet verwerken. Elke subdirectory bevat de Helm-chart of Kubernetes-manifesten voor één platformcomponent of teamapplicatie.
Wanneer een platformengineer of ontwikkelaar een nieuwe applicatie wil toevoegen aan het platform, volstaat het om een nieuwe directory aan te maken in Git met de benodigde manifesten. De ApplicationSet-controller detecteert de nieuwe directory automatisch en creëert een Argo CD Application die de inhoud synchroniseert naar de cluster. Er is geen handmatige configuratie in Argo CD nodig.
Dit patroon biedt twee belangrijke voordelen:

1. **Clusterherstellbaarheid vanuit één Git-referentie**: het volledige cluster kan worden hersteld door Argo CD te laten wijzen naar de root van de applicatiedirectory. Alle platformcomponenten en teamapplicaties worden automatisch opnieuw gesynchroniseerd.
2. **Schaalbaar multi-app-beheer**: het toevoegen van tientallen of honderdtallen applicaties vereist geen wijziging aan Argo CD zelf. De ApplicationSet-controller schaalt mee met het aantal directory's in Git.

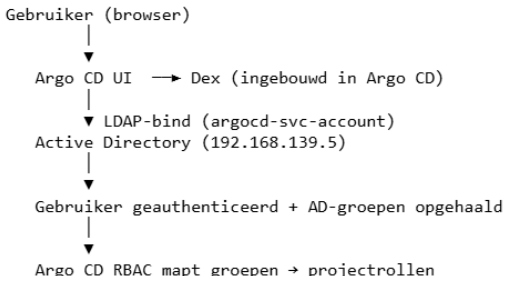


Figuur 5: ArgoCD gebruikersinterface

4.2.2. Active Directory-integratie (Dex/OIDC)

Argo CD gebruikt **Dex** als ingebouwde identity provider om gebruikers te authenticeren tegen Active Directory via het LDAP-protocol. Het authenticatieproces verloopt als volgt:

Wanneer een gebruiker inlogt via de Argo CD-webinterface, wordt het verzoek doorgestuurd naar Dex. Dex voert een LDAP-bind uit met het argocd-svc-serviceaccount tegen Active Directory. Als de gebruiker zich succesvol authenticeert, haalt Dex de groepslidmaatschappen van de gebruiker op uit Active Directory. Argo CD gebruikt deze groepen vervolgens om te bepalen welke rechten de gebruiker heeft.



Figuur 6: Authenticatiestroom van Argo CD via Dex naar Active Directory

De rolmapping is als volgt geconfigureerd:

AD-groep	Argo CD-rol	Rechten
ArgoAdmin	role:admin	Volledig beheerderstoegang over alle projecten en de Argo CD-webinterface
ArgoReadOnly	role:readonly	Alleen-lezentoegang tot alle applicaties en resources
<project-name>	proj:<project-name>:developer	Volledig applicatiebeheer binnen dat specifieke project

Tabel 13: Mapping van AD-groepen naar Argo CD-rollen

De derde rij in de tabel is bijzonder: wanneer een nieuw project wordt aangemaakt via de projectmanagementpipeline (wordt later beschreven), wordt automatisch een Argo CD-project gecreëerd met een developer-rol. Elke AD-groep met dezelfde naam als het project krijgt automatisch die rol toegewezen. Dit betekent dat er geen handmatige rolconfiguratie nodig is wanneer een nieuw team wordt onboard.

4.3. Security Plane

4.3.1. Vault: initialisatie en configuratie

HashiCorp Vault is de gecentraliseerde secret engine van het platform. Vault draait in HA-modus (High Availability) met drie replica's die onderling communiceren. Elke replica heeft een eigen Longhorn-PVC (Persistent Volume Claim) voor opslag.
De initialisatie van Vault verloopt via de configuratiepipeline en is volledig idempotent: als Vault al is geïnitialiseerd, worden de initialisatiestappen overgeslagen. De volledige initialisatieprocedure omvat de volgende stappen:

1. **Status controleren**: de pipeline controleert of Vault al is geïnitialiseerd. Zo ja, worden stappen 2–4 overgeslagen.
2. **Initialisatie**: vault operator init genereert vijf unseal-sleutels met een drempelwaarde van drie. Dit betekent dat minstens drie van de vijf sleutels nodig zijn om Vault te ontgrendelen.
3. **Unseal-sleutels opslaan**: alle vijf unseal-sleutels en het root-token worden opgeslagen in een Kubernetes Secret in de vault-namespace.
4. **Vault ontgrendelen**: alle drie Vault-pods worden ontgrendeld met sleutels 1, 2 en 3.
5. **KV v2 secret-engine activeren**: op het pad secret/ wordt een Key-Value v2 secret-engine geactiveerd voor platform-brede secrets.
6. **LDAP-authmethode configureren**: Vault wordt gekoppeld aan Active Directory via een LDAP-bind. De AD-groep VaultAdmin wordt gemapt op een admin-policy met volledige lees- en schrijfrechten.

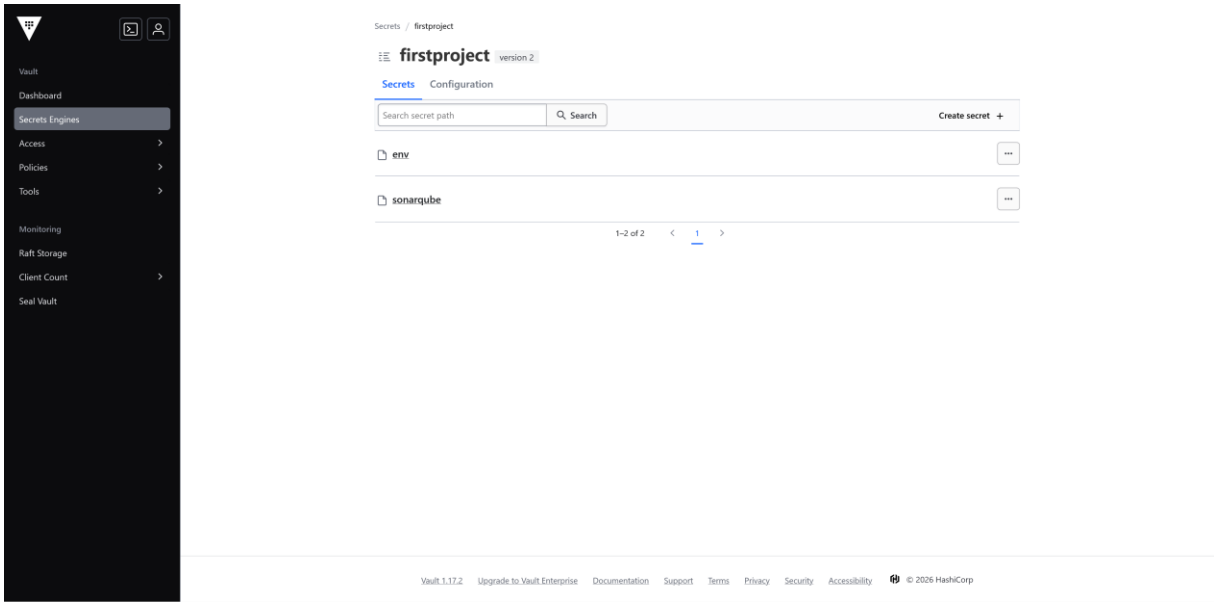
7. **Kubernetes-authmethode configureren**: applicatiepods en de External Secrets Operator kunnen via een Kubernetes serviceaccount secrets lezen zonder wachtwoord. Tokens hebben een TTL (Time To Live) van één uur en worden automatisch vernieuwd.

Dag-2 beheer: ontgrendelen na herstart
Wanneer Vault-pods herstarten, bijvoorbeeld na een nodefout, starten ze in sealed toestand. De configuratiepipeline kan opnieuw worden uitgevoerd om alleen de verzegelde pods te ontgrendelen. De pipeline controleert per pod de seal-status en ontgrendelt uitsluitend de pods die dat nodig hebben.

Dagelijks gebruik door teams
Teams kunnen hun eigen secrets beheren via de Vault-webinterface op https://vault.stage.axxes.local. Het proces is als volgt:

1. Inloggen met LDAP (AD-credentials)
2. Navigeren naar de secrets-engine van het eigen project (bijvoorbeeld firstproject) → env
3. Key-value pair toevoegen of bijwerken (bijvoorbeeld ENVIRONMENT: production)
4. ESO synchroniseert de wijzigingen naar de bijbehorende Kubernetes Secret binnen één minuut
5. Pods nemen de nieuwe waarden op bij de volgende herstart

Er is geen tussenkomst van het platformteam nodig voor het dagelijks secretbeheer.



Figuur 7: Vault Gebruikersinterface voor een project

4.3.2. External Secrets Operator (ESO)

De External Secrets Operator vormt de brug tussen Vault en Kubernetes. ESO synchroniseert secrets vanuit Vault naar standaard Kubernetes Secrets per projectnamespace, zodat applicatiepods secrets kunnen consumeren als omgevingsvariabelen zonder rechtstreeks met de Vault-API te communiceren. Dit werkt via twee Kubernetes Custom Resource Definitions (CRD's):

- **SecretStore**: definieert per namespace hoe ESO verbinding maakt met Vault. Dit omvat het Vault-adres, de authenticatiemethode (Kubernetes service account), en het mount path van de secrets-engine.
- **ExternalSecret**: specificeert welke Vault-paden gesynchroniseerd moeten worden naar welke Kubernetes Secret. Het verversingsinterval is geconfigureerd op één minuut.

Bij het aanmaken van een nieuw project via de projectmanagementpipeline worden de SecretStore en een standaard-ExternalSecret automatisch aangemaakt. De ExternalSecret synchroniseert het pad *"<project>/env"* in Vault naar een Kubernetes Secret genaamd *"<project>-env"* in de projectnamespace.

Ontwikkelaars hoeven niets te weten over Vault of ESO: ze refereren simpelweg naar de Kubernetes Secret in hun platform.yaml en de waarden zijn beschikbaar als omgevingsvariabelen in hun pods.

Op het einde heb heb ik het automatisch herstarten van applicaties gemaakt, zodat wanneer er wijzigingen zijn in Vault deze na de synchronisatieminuut onmiddellijk beschikbaar zijn in de appplicatie.
Deze optie kan worden toegevoegd in de HELM values deployment methode door de reloader aan te zetten. Dit is bijzonder nuttig in combinatie met de External Secrets Operator (ESO): wanneer ESO een Secret ververst na een Vault-wijziging, detecteert Reloader de Secret-update en herstart de bijbehorende pods automatisch. Zo nemen applicaties secretwijzigingen op zonder handmatige tussenkomst.

4.3.3. Kyverno-policies

Kyverno draait als admission controller op het cluster. Een admission controller is een component die elk verzoek aan de Kubernetes-API onderschept en valideert, muteert of aanvult vóórdat het verzoek wordt uitgevoerd. Kyverno dwingt twee policies af:

Policy	Type	Trigger	Effect
generate-default-network-policy	Generate	Een namespace met het label managed-by: idp-platform wordt aangemaakt	Creëert automatisch drie NetworkPolicies: default-deny-ingress, default-deny-egress en allow-dns-egress
require-standard-labels	Validate	Elke Deployment, StatefulSet, DaemonSet, Job of CronJob in een platformbeheerde namespace	Weigert resources die de verplichte labels app.kubernetes.io/name, app.kubernetes.io/team of app.kubernetes.io/managed-by missen
no-run-as Root	Enforce	Elke deployment, container in een namespace aangemaakt door een project	Weigert alle container die proberen te starten met een image waar die runt als de root user.
Only-signed-images	Enforce	Elke deployment, container in een namespace aangemaakt door een project	Enkel images signed via cosign met het clustercertificaat mogen opstarten.

Tabel 14: Kyverno-policies op het platform

De generate-policy verdient bijzondere toelichting. Wanneer het platformteam een nieuw project aanmaakt, wordt er een Kubernetes-namespace gecreëerd met het label managed-by: idp-platform. Kyverno detecteert dit label en genereert automatisch drie NetworkPolicies die al het inkomend en uitgaand verkeer blokkeren, behalve DNS-verkeer. Dit is het default-deny-principe: workloads moeten expliciet het verkeer openen dat ze nodig hebben.

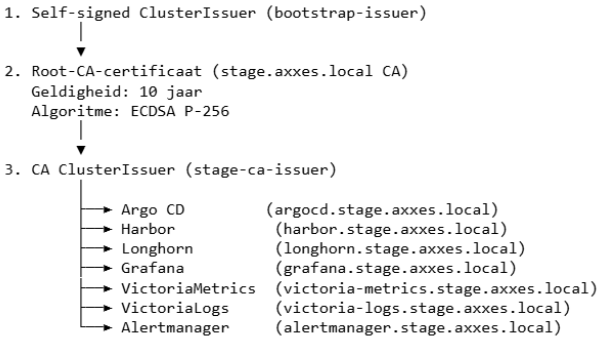
Aan de Policy Engine is een strikte beveiligingsregel toegevoegd die het draaien van containers met root-rechten verbiedt (no-run-as-root). Deze maatregel dwingt af dat alle workloads het "least privilege"-principe volgen, wat de impact van een potentiële inbreuk direct mitigeert.
Vervolgens worden enkel container images toegelaten, die gecertifieerd zijn met een certificaat ondertekend door cosign met het clustercertificaat. Standaard worden de Harbor mirrored en developer repository builded images ondertekend met dit certificaat en zijn hierdoor mogelijk om te gebruiken in het platform.

4.3.4. TLS en certificaten

Alle ingress-verkeer op het platform is TLS-beveiligd. In plaats van commerciële certificaten te gebruiken, beheert het platform een eigen certificaatketen via cert-manager, een Kubernetes-operator die het aanvragen, vernieuwen en distribueren van TLS-certificaten automatiseert.

De certificaatketen is als volgt opgebouwd:

De keten werkt in drie stappen. Eerst genereert een self-signed ClusterIssuer het root-CA-certificaat. Dit root-CA wordt vervolgens gebruikt als basis voor een CA ClusterIssuer, die op zijn beurt individuele TLS-certificaten uitgeeft voor elke platformcomponent. Doordat alle ingresscertificaten zijn ondertekend door dezelfde root-CA, volstaat het om dat ene root-certificaat te vertrouwen op een werkstation om alle platform-URL's als vertrouwd te beschouwen in de browser.
Het root-CA-certificaat wordt na de bootstrap automatisch geüpload naar Bitbucket Downloads, samen met de kubeconfig en talosconfig, zodat ontwikkelaars het eenvoudig kunnen downloaden en installeren.

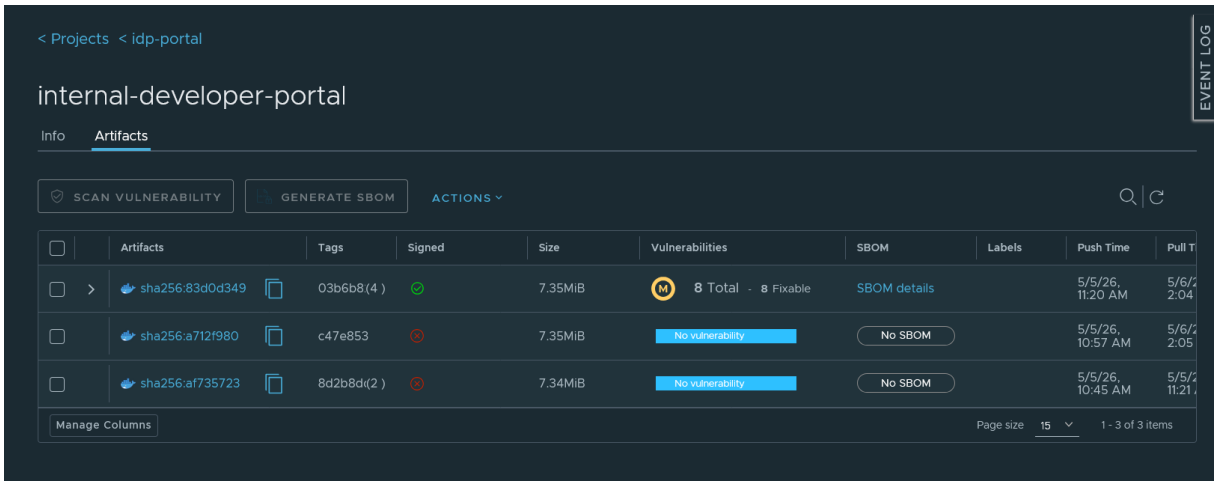


Figuur 8: TLS-certificaatketen van het platform

4.3.5. Container Image Signing met Cosign

Om de supply-chain security te waarborgen is Cosign geïntroduceerd. Tijdens het build-proces in de Bitbucket pipeline worden ontwikkelde container images cryptografisch ondertekend voordat ze in Harbor terechtkomen. Dit waarborgt de integriteit van de images en garandeert dat exact de in de pipeline gecontroleerde code (Shift-Left security) ook daadwerkelijk uitgerold wordt in productie. Een goede vergelijking hiervoor is de authentiek waszegel op een envelop, die bewijst dat de brief onderweg niet geopend is, Cosign werkt als een digitale handtekening op alle code. Het geeft ons platform de garantie dat de applicatie die uiteindelijk op de server belandt, exáct de applicatie is die wij hebben getest in de deployment-pipeline en dat hackers er onderweg niet ongemerkt kwaadaardige code aan hebben toegevoegd

Ook de images mirrored naar Harbor worden eerst ondertekend met dit certificaat om de integriteit te waarborgen. Verder wordt dit certificaat gecontroleerd alvorens een conatiner opgestart wordt in de cluster, doormiddel van een Kyverno policy die deze zegel zal controleren en eventueel weigeren als het niet correct is.



Figuur 9: Harbor gebruikersinterface, voor een project, image signing & scanning

4.3.6. Trivy Operator: Clusterscanning

De Trivy-scan in de CI/CD-pipeline (beschreven in sectie 4.6.3) controleert een containerimage op het moment dat het gebouwd wordt. Dit dekt echter slechts een moment in de tijd: nieuwe kwetsbaarheden die later ontdekt worden in een image die al in productie draait, worden door de pipeline niet meer opgemerkt.

De **Trivy Operator** vult deze blinde vlek op door continu alle draaiende workloads in het cluster te scannen. De Trivy Operator draait als platformcomponent in de trivy-system-namespace en scant voortdurend alle actieve pods op bekende CVE's (Common Vulnerabilities and Exposures) en Kubernetes-misconfiguraties. De scanresultaten worden opgeslagen als Kubernetes Custom Resource Definitions (CRD's) van het type VulnerabilityReport. Deze rapporten zijn raadpleegbaar via kubectl en worden automatisch uitgelezen door VictoriaMetrics, waardoor de kwetsbaarheden zichtbaar zijn in een dedicated beveiligingsdashboard in Grafana. Platformengineers kunnen zo op één plek zien welke workloads kwetsbaarheden bevatten en hoe ernstig deze zijn, ingedeeld naar de standaard CVSS-ernstcategorieën: Critical, High, Medium en Low.
De twee Trivy-mechanismen zijn bewust complementair ontworpen:

Mechanisme	Wanneer	Doel
Trivy in de CI/CD-pipeline	Bij elke build	Voorkomt dat kwetsbare images worden uitgerold (Shift-Left)
Trivy Operator in het cluster	Continu, na deployment	Bewaakt de draaiende vloot op nieuw ontdekte kwetsbaarheden

Door beide lagen te combineren dekt het platform zowel het moment van deployment als de volledige levensduur van een applicatie in het cluster.

4.3.7. Auto Secret Reloader

Kubernetes-pods nemen wijzigingen in Secrets en ConfigMaps standaard niet automatisch over: een pod die al draait, blijft de waarden gebruiken die bij het opstarten zijn ingeladen. Om een secretwijziging door te voeren, is normaal gesproken een handmatige herstart van de deployment nodig. De **Stakater Reloader** automatiseert dit.

Reloader bewaakt voortdurend alle Secrets en ConfigMaps in het cluster. Zodra een Secret of ConfigMap wijzigt, triggert Reloader automatisch een rolling restart van de bijbehorende deployment. In de base chart is deze functionaliteit als optie beschikbaar per applicatie.

Dit is bijzonder waardevol als sluitstuk van de Vault-ESO-keten. Wanneer een ontwikkelaar een waarde aanpast in Vault, synchroniseert ESO de wijziging binnen één minuut naar het bijbehorende Kubernetes Secret. Reloader detecteert die Secret-update en herstart de pods automatisch. Het volledige proces van Vault-aanpassing tot actieve nieuwe waarde in de applicatie verloopt daarmee zonder enige handmatige tussenkomst van het platformteam of de ontwikkelaar. Dit maakt secretrotatie operationeel risicoloos: teams hoeven niet te onthouden pods te herstarten na een secretwijziging.

4.4. Observability

De observability-stack van het platform biedt out-of-the-box inzicht in de gezondheid van zowel de infrastructuur als de applicaties die erop draaien. De stack bestaat uit vier pijlers: metrics, logging, dashboarding en alerting.

4.4.1. Metrics: VictoriaMetrics

VictoriaMetrics draait als metrics-backend en is volledig Prometheus-compatibel: bestaande scrape-configuraties, dashboards en alertregels die voor Prometheus zijn geschreven, werken ongewijzigd met VictoriaMetrics.
VictoriaMetrics scrapet automatisch alle Kubernetes-endpoints die de annotatie prometheus.io/scrape: "true" dragen. Platformcomponenten zoals Traefik (ingresscontroller), Longhorn (storage), Argo CD (GitOps) en cert-manager (TLS) exposeren standaard metrics die door VictoriaMetrics worden verzameld. Ontwikkelteams die hun eigen applicatiemetrics willen exposeren, hoeven enkel de juiste annotatie op hun pods te plaatsen, VictoriaMetrics detecteert en scrapt ze automatisch.
De keuze voor VictoriaMetrics boven het meer gangbare Prometheus is onderbouwd in hoodstuk 3. De belangrijkste voordelen in de praktijk zijn het significant lagere geheugengebruik (relevant voor de beperkte on-premises-resources) en de eenvoudigere HA-configuratie zonder externe componenten als Thanos of Mimir.

4.4.2. Logging: VictoriaLogs en Vector

Voor gecentraliseerde logging worden twee verzamelmechanismen ingezet:

1. **VictoriaLogs Collector**: verzamelt containerlogs van alle pods en nodes en stuurt deze naar VictoriaLogs als opslagbackend. De collector draait als DaemonSet op elke node in het cluster en verzamelt logs via het standaard Kubernetes-logpad (/var/log/pods/). De retentie is ingesteld op zeven dagen.
2. **Vector**: wordt uitsluitend ingezet voor het verzamelen van **Kubernetes API-auditlogs**. De Talos-machineconfiguratie bevat een volledig auditbeleid dat alle API-aanroepen vastlegt. Vector draait als DaemonSet op de control-plane-nodes, leest de auditlogs uit /var/log/audit/audit.log en stuurt deze door naar VictoriaLogs. Deze auditlogs registreren wie een API-verzoek heeft gedaan, welke resource werd aangesproken, wanneer het verzoek plaatsvond en wat het resultaat was (toegestaan, geweigerd, fout).

De logs, zowel containerlogs als API-auditlogs, zijn doorzoekbaar via de VictoriaLogs-webinterface op https://victoria-logs.stage.axxes.local. Engineers kunnen containerlogs filteren op namespace, pod, container en tijdsperiode zonder kubectl logs te gebruiken of directe clustertoegang te hebben. Platformengineers kunnen daarnaast de API-auditlogs raadplegen voor compliance- en troubleshootingdoeleinden.

4.4.3. Dashboarding: Grafana

Grafana is geconfigureerd met VictoriaMetrics als primaire datasource en biedt een reeks vooraf geconfigureerde dashboards:

- **Clustergezondheid**: CPU-, geheugen- en schijfgebruik per node, nodestatus en kubelet-gezondheid
- **Kubernetes-workloads**: podstatus, containerrestarts, resourcegebruik per namespace
- **Platformcomponenten**: Traefik-verzoekstatistieken, Longhorn-volumestatus, Argo CD-synchronisatiestatus

Grafana-authenticatie verloopt via LDAP tegen Active Directory, met dezelfde AD-groepen die ook voor Argo CD en Harbor worden gebruikt. Dit betekent dat een ontwikkelaar met één set AD-credentials toegang heeft tot Argo CD, Grafana, Harbor en Vault, een consistente aanmeldervaring over alle platformcomponenten.

Figuur 10: Grafana voorbeeld dashboard

4.4.4. Alerting: Alertmanager naar Microsoft Teams

Alertmanager ontvangt alerts van VictoriaMetrics wanneer vooraf gedefinieerde drempelwaarden worden overschreden en stuurt meldingen naar **Microsoft Teams** via een webhook-integratie. De volgende alertregels zijn voorbeelden van standaard geconfigureerde regels:

- **Node-onbeschikbaarheid**: melding wanneer een clusternode niet meer bereikbaar is
- **Hoge CPU- of geheugenbelasting**: melding wanneer een node langdurig boven een drempelwaarde zit
- **PersistentVolume-capaciteit**: melding wanneer een Longhorn-volume bijna vol is
- **Pod-crashloops**: melding wanneer een pod herhaaldelijk crasht en herstart
- **+ Nog veel meer**

Deze regels zijn standaart regels gegeven door de HELM-charts van VictoriaLogs, ook de standaart regels voor longhorn (Storage) zijn toegvoegd via ArgoCD application. Ook kunnen er nog manueel regels toegvoegd werden en deze worden uitgerolt via ArgoCD. Dit maakt het mogelijk om specifieke alerts te ontvangen, aangepast voor elke applicatie.

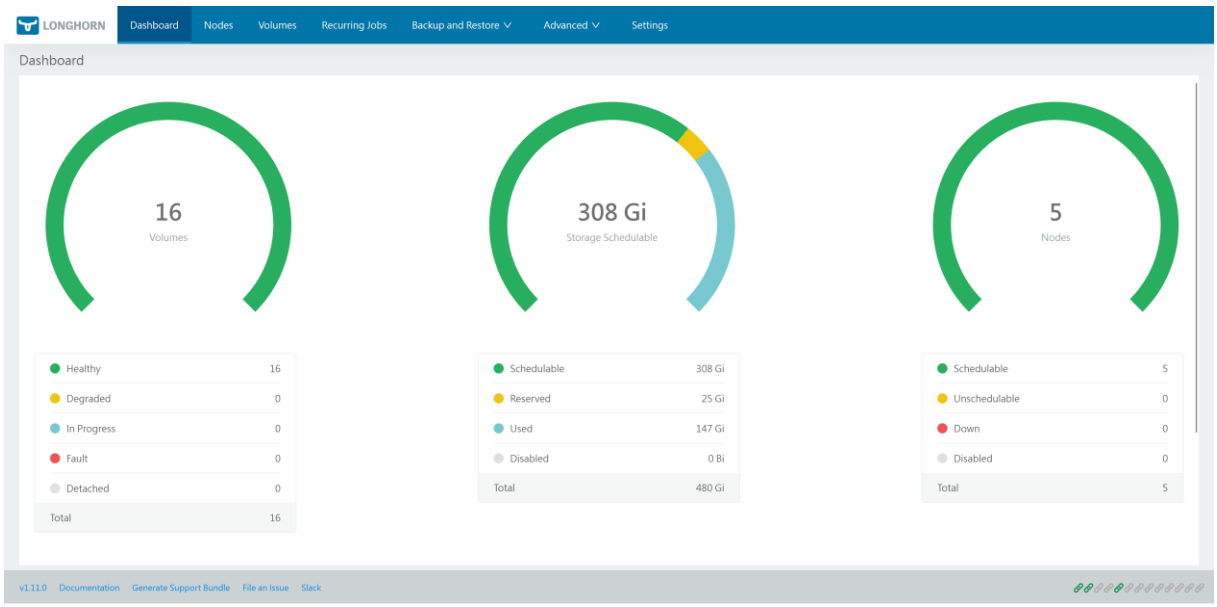
4.5. Data Plane

4.5.1. Storage, Longhorn

Longhorn is de gedistribueerde blokopslagoplossing van het platform. Het biedt Persistent Volumes (PV's) aan stateful workloads zoals Vault, Harbor, CloudNativePG-databases en VictoriaMetrics. Longhorn repliceert data over meerdere nodes, zodat het verlies van één node niet leidt tot gegevensverlies.

Longhorn is geïnstalleerd via Helm in fase 3 van de bootstrap. Om correct te functioneren op Talos Linux zijn specifieke machineconfiguraties nodig (zie tabel 12 in sectie 4.1.1): de kernelmodules nbd, iscsi_tcp en configfs worden bij het opstarten geladen en de Talos-extensies iscsi-tools en util-linux-tools zijn geïnstalleerd.

De Longhorn-managementinterface is beschikbaar op https://longhorn.stage.axxes.local en biedt een overzicht van alle volumes, hun replicatiestatus, snapshots en backups.



Figuur 11: Longhorn gebruikersinterface

4.5.2. Backupstrategie & Disaster Recovey

Het platform hanteert een tweelaagsgegevensbescherming die zowel snelle herstelacties als bescherming tegen clusteruitval biedt:

Type	Opslaglocatie	Snelheid	Retentie	Beschermt tegen
Snapshots	Lokaal op Longhorn (op de clusternodes)	Snel aan te maken en te herstellen	7 dagen	Gebruikersfouten, per ongeluk verwijderde data
Backups	RustFS (externe S3-compatibele objectopslag)	Langzamer, maar overleeft clusteruitval	4 weken	Nodefout, clusteruitval, corruptie

Tabel 15: Backupstrategie met Longhorn en RustFS

RustFS is een lichtgewicht S3-compatibele objectopslagserver die buiten het cluster draait. Longhorn uploadt backups naar een RustFS-bucket via het S3-protocol. De credentials voor RustFS zijn opgeslagen als Kubernetes Secret in de longhorn-system-namespace.

Beide typen gegevensbescherming zijn geautomatiseerd via recurring jobs die bij de bootstrap worden geconfigureerd. Handmatige snapshots kunnen worden aangemaakt via de Longhorn-webinterface of via kubectl.

Naast het regulier back-uppen van applicatiedata, is het platform tevens voorbereid op 'worst-case' scenarios. Bijvoorbeeld het verliezen van de volledige Kubernetes control-plane. Hiervoor is een volwaardig Disaster Recovery proces ingericht dat draait om het herstellen van de fundamentele Kubernetes status (etcd).

- **Geautomatiseerde etcd-snapshots:** Via een Kubernetes CronJob draait op de achtergrond de Sidero Labs talos-backup tool. Deze maakt dagelijks een .snap snapshot van de volledige etcd-database (het "brein" van de cluster).
- **Off-cluster Opslag:** Om te garanderen dat backups overleven wanneer het cluster faalt, maakt de backup-pod gebruik van S3-compatibele externe opslag (RustFS). Deze opslag draait onafhankelijk van de cluster.
- **Bare-Metal Recovery:** In het geval van een totale datacenter crash, illustreert het bijhorende DR-runbook de gestroomlijnde recovery procedure van Talos Linux: Er wordt via de originele Terraform-code een "schone" control-plane node (met behoud van het oorspronkelijke IP-adres) opgetrokken. In plaats van een standaard nieuwe bootstrap start de administrator vervolgens de server op via de Talos API mét de gedownloade RustFS snapshot: talosctl bootstrap --recover-from=<downloaded_file.snap>.

Vanaf het moment dat het fundamentele etcd brein succesvol aan deze nieuwe primaire node is geïnjecteerd, hoeven de overige nodes slechts standaard toegevoegd (ge-joined) te worden aan het cluster. Zij zullen zich automatisch synchroniseren, waarmee het voormalige platform, inclusief ArgoCD state, Longhorn volume-definities en applicaties weer volledig "tot leven" is gewekt uit de assen van de crash. Deze naadloze herbouwbaarheid toont wederom de immense waarde van *Everything-as-Code* (Terraform infrastructuur) gecombineerd met rigoureuze back-ups (etcd state).

4.5.3. Database: CloudNativePG

CloudNativePG is een Kubernetes-operator die het volledige levenscyclusbeheer van PostgreSQL-databases automatiseert. Wanneer een ontwikkelteam in hun platform.yaml een spec.database-blok opneemt, wordt automatisch een PostgreSQL-cluster aangemaakt.

Een voorbeeld van een databasespecificatie in platform.yaml:

```yaml
spec:
  database:
    enabled: true
    instances: 2        # Primary + 1 replica
    storage:
      size: 10Gi
```

Op basis van deze specificatie genereert het platform een CloudNativePG Cluster-resource die het volgende regelt:

- **Primary-replica-topologie**: de operator beheert automatisch de primary/replica-verhouding, inclusief streaming replicatie en automatische failover wanneer de primary uitvalt
- **Longhorn-backed opslag**: elke PostgreSQL-instance krijgt een eigen Longhorn PVC voor gedistribueerde, gerepliceerde blokopslag
- **Credentials-injectie**: de databaseverbindingsgegevens (host, poort, gebruikersnaam, wachtwoord) worden automatisch als Kubernetes Secret aangemaakt door de operator en als omgevingsvariabelen geïnjecteerd in de applicatiepod
- **S3-backup**: CloudNativePG kan WAL-archieven (Write-Ahead Logs) en base-backups pushen naar RustFS via het S3-protocol, als aanvullende databasespecifieke backup bovenop de Longhorn-volumebackups

4.5.4. Message Queuing: RabbitMQ

Naast een relationele database via CloudNativePG ondersteunt het platform ook asynchrone berichtenwachtrijen. Wanneer een applicatie berichten tussen services wil uitwisselen zonder directe koppeling, kan een team een eigen **RabbitMQ**-instantie aanvragen. Net als bij de database-aanvraag in platform.yaml hoeft een developer hiervoor geen Kubernetes-resources handmatig te schrijven: het activeren van één vlag in het values.yaml-bestand volstaat.

Onder de motorkap provisioneert de base chart een RabbitmqCluster-resource via de officiële **RabbitMQ Cluster Operator**. Het platform genereert automatisch een standaardgebruiker en bijbehorend wachtwoord. De verbindingsgegevens zoals hostnaam, poort, gebruikersnaam en wachtwoord worden als omgevingsvariabelen rechtstreeks in de applicatiecontainer geïnjecteerd, zonder dat de developer iets handmatig hoeft te configureren of secrets hoeft op te slaan.

Een aandachtspunt bij het gebruik van RabbitMQ is de opstarttijdvolgorde. De broker een warmup-fase nodig voordat hij verbindingen kan accepteren. Een applicatiecontainer start doorgaans sneller op dan de broker. Applicaties moeten daarom bij het opstarten verbindingspogingen herhalen met een wachttijd tussen pogingen in, in plaats van direct te falen als de eerste verbinding mislukt.

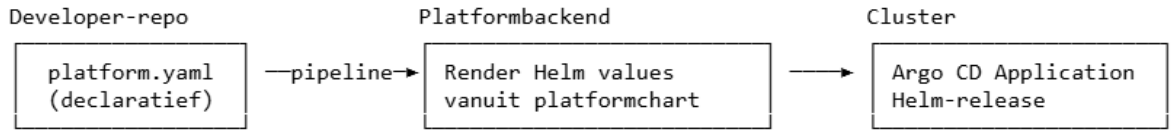
4.6. Golden Path en Developer Experience

Om de complexiteit voor ontwikkelteams te minimaliseren, definieert het platform een zogenoemde Golden Path. Dit is het aanbevolen, sterk geautomatiseerde pad dat een applicatie volgt van broncode tot productie. Door deze 'gouden standaard' in te richten als een Self-Service Infrastructure, hoeven ontwikkelaars niet buiten de lijntjes te kleuren en garandeert het platformteam onzichtbaar dat alle bedrijfsstandaarden en beveiligingseisen (zoals Shift-Left Security) automatisch worden toegepast.

In dit platform zijn meerdere deployment golden path strategieën geïmplementeerd. Dee eerste strategie is gebaseerd op een eigen ontworpen platform.yaml configuration file, de tweede strategie is gebaseerd op het templaten in HELM charts. Op deze tweede stretegie is er ook een multi environment deployment gebaseerd voor een meer volwaardige deployment.

4.6.1. platform.yaml: het declaratieve contract

In plaats van zelf Kubernetes-manifesten, Helm-charts, Dockerfiles en pipelineconfiguraties te schrijven, plaatst een ontwikkelaar één declaratief bestand, “platform.yaml”, in de repository van de applicatie.
Dit bestand beschrijft de applicatie op een hoog abstractieniveau:



Figuur 12: De golden path van platform.yaml naar een draaiende applicatie

Het platform.yaml-bestand bevat secties voor:

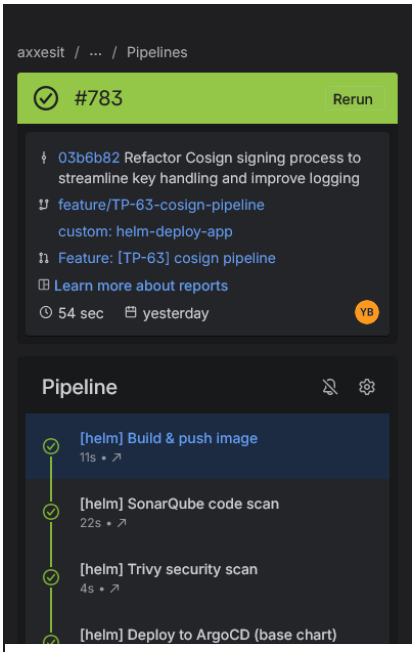
- **Applicatie-identiteit**: naam, team, project
- **Container-image**: registry, repository, tag
- **Ingress**: hostname en TLS-configuratie
- **Secrets**: verwijzingen naar Kubernetes Secrets (gesynchroniseerd vanuit Vault door ESO)
- **Database**: optioneel PostgreSQL-cluster via CloudNativePG (zie sectie 4.5.3)

- **Opslag**: optionele Longhorn-backed PersistentVolumeClaims
- **Resources**: CPU- en geheugenlimieten

De platformrenderer, een script in de CI/CD-pipeline, deze valideert het platform.yaml-bestand, genereert een Helm values-bestand op basis van de opgegeven specificatie, verpakt het als Helm-chart en pusht het naar Harbor. Vervolgens wordt een Argo CD Application-manifest geschreven naar de GitOps-repository. Argo CD detecteert de wijziging en synchroniseert de applicatie naar de cluster.

4.6.2. Evolutie: Van platform.yaml naar Helm Base Chart

In het begin was het Golden Path uitsluitend ontworpen rondom het platform.yaml bestand. Het doel was de Cognitive Load maximaal te verlagen doordat ontwikkelaars geen specifieke sturing of kennis rondom Kubernetes of Helm nodig hadden. Hun abstracte wensen werden door in-house pipeline-scripts (RenderPlatformApp.sh) vertaald naar specifieke Helm charts.



Figuur 13: Bitbucket pipeline, deploy applicatie vie HELM strategie

Na feedback tijdens een demo-sessie kwam hier echter een belangrijk pijnpunt naar voren. Helm is immers ontworpen als een native templating tool en de gemaakte CI/CD scripts waren in feite eigen templating-logica (van platform.yaml naar values.yaml) daarbóvenop aan het doen. Dit "dubbel templaten" was niet alleen dubbel werk, maar resulteerde er ook in dat voor éérdere wijziging (zoals een nieuwe image-tag) een compleet nieuwe unieke per-applicatie Helm chart gegenereerd en gepubliceerd werd in de Harbor registry. Harbor liep hierdoor vol met identieke charts per commit.

De oplossing native values.yaml en Shared Base Chart Om dit op te lossen is als volwaardig alternatief het Helm Base Chart-patroon geïmplementeerd. Hierbij beheert het platformteam centraal één grote, gedeelde base chart (platform-app). Ontwikkelaars leveren nu simpelweg een standaard values.yaml bestand aan in hun repository. Er is geen extra "render script" meer: de pipelines injecteren enkel een nieuwe image tag, dit is de commit hash van de laatste commit, deze wordt rechtstreeks in het Application manifest voor Argo CD geplaatst. Argo CD wijst naar de centrale base chart in Harbor en voedt dit direct in de eigen engine met de door de developer aangeleverde waarden.

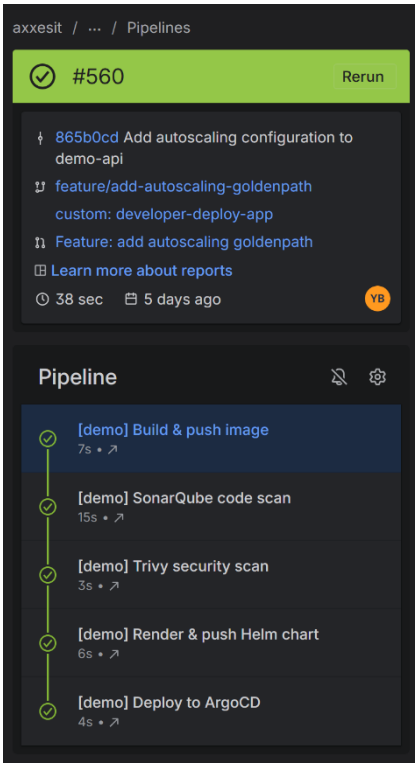
Hierdoor behouden teams dezelfde enterprise standaarden, maar is "double templating" geëlimineerd. Ook de Harbor storage blijft schoon omdat de chart één enkele gecentraliseerde versie is. Teams hebben hierbij bovendien direct toegang tot hun applicatie specifieke monitoring middels vmrule.yaml alert-injecties.

Dit patroon bleek zo robuust dat deze native values.yaml weg inmiddels succesvol als fundering wordt gebruikt voor de later geïmplementeerde multi-environment en multi-stage deployments.

4.6.3. Developerpipeline (Build → Scan → Deploy)

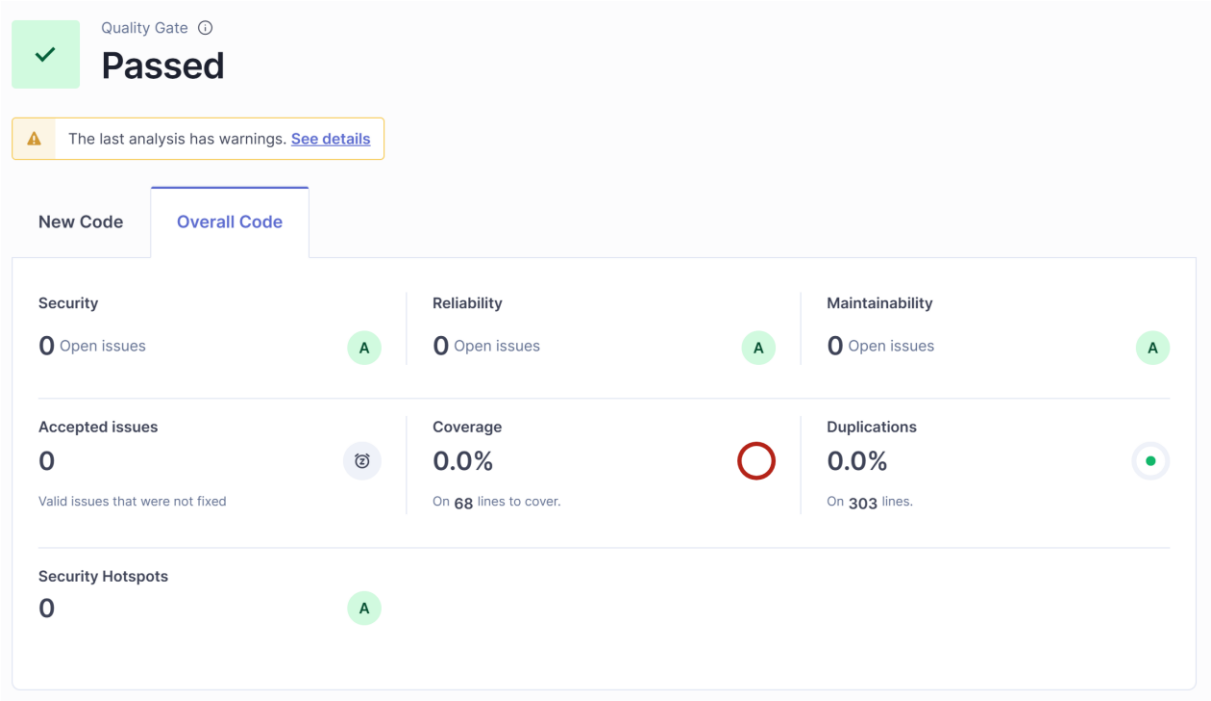
De standaardpipeline voor developers dekt de volledige levenscyclus van broncode tot draaiende applicatie in vijf stappen

Figuur 14: De vijf stappen van de developerpipeline



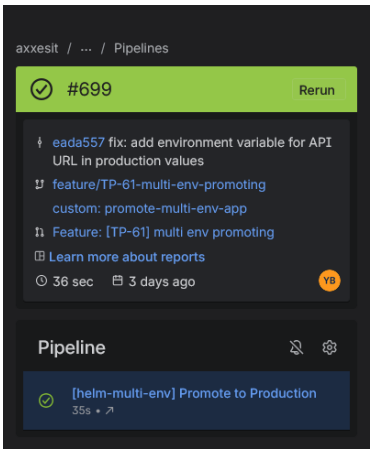
Figuur 15: Bitbucket pipeline, deploy applicatie via Platform.yaml strategie

Stap 1: Build: de applicatie wordt gebouwd als container-image en gepusht naar het teamproject in Harbor (harbor.stage.axxes.local/<project>/<app>:<sha>).
Stap 2: SonarQube scan: de pipeline leest het SonarQube-adres en -token uit Vault. Als het token leeg is (het team heeft SonarQube nog niet geconfigureerd), wordt de scan overgeslagen zonder de pipeline te laten falen. Als het token wel is ingesteld, wordt een codekwaliteitsscan uitgevoerd en wacht de pipeline op het resultaat van de quality gate. Een falende quality gate blokkeert de deployment.
Stap 3: Trivy scan: het gebouwde container-image wordt gescand op bekende kwetsbaarheden via Trivy, dat ingebouwd. De pipeline faalt als er meer dan drie kritieke kwetsbaarheden worden gevonden. Dit voorkomt dat onveilige images in productie terechtkomen.
Stap 4: Render chart: het platform.yaml-bestand wordt gevalideerd tegen de golden path-specificatie. Vervolgens wordt een Helm-chart gegenereerd met de opgegeven waarden en gepusht naar Harbor als OCI-artefact. Deze stap is niet meer nodig met de HELM templating strategie van deployen.
Stap 5: Deploy: de pipeline schrijft een Argo CD Application-manifest en voert deze uit op de cluster. ArgoCD rendert de Helm-chart met de opgegeven values en past de resulterende Kubernetes-manifesten toe op het cluster. De applicatie is live.



Figuur 16: SonarQube gebruikersinterface, voor een applicatie in een project

Na het implementeren van de native HELM templating deployment variant, is hierop ook een multi environment deployment gemaakt. Om een volwassen releasecyclus te faciliteren, zijn de CI/CD-pipelines hiervoor uitgebreid met functionaliteit voor Multi-Environment Deployments. Binnen een project kunnen nu diverse gescheiden omgevingen (zoals staging en productie) naast elkaar bestaan. Een geautomatiseerde Promotion Flow zorgt ervoor dat applicaties na seen succesvolle validatie naadloos gepromoveerd kunnen worden van staging naar productie, inclusief het geautomatiseerd aanmaken van multi-environment gerelateerde namespaces en Argo CD inrichting.



Figuur 17: Bitbucket pipeline, promoting staging to production&

4.6.4. Projectmanagement (namespace, Argo CD, Vault, Harbor)

Een **project** op het platform is de logische eenheid die alles bundelt wat een team nodig heeft om te werken. Het omvat:

Onderdeel	Wat wordt aangemaakt
Kubernetes-namespace	<project> - geïsoleerd met resourcequota's en default-deny-NetworkPolicies
Argo CD-project	<project> - begrenst waar het team naar kan deployen en welke repositories het mag gebruiken
Vault secrets-engine	KV v2 op pad <project>/ met standaardsecrets (APP_NAME, ENVIRONMENT, LOG_LEVEL, DEBUG) en RBAC-policies
Harbor-project	harbor.stage.axxes.local/<project>/
ESO SecretStore + ExternalSecret	Synchroniseert <project>/env → Kubernetes Secret <project>-env, vernieuwd elke minuut

Tabel 16: Onderdelen die worden aangemaakt bij een nieuw project

Al het bovenstaande wordt aangemaakt via een dedicated NewProject-pipeline in Bitbucket. Deze pipeline accepteert de projectnaam als parameter en voert sequentieel de volgende scripts uit:

1. Kubernetes-namespace aanmaken met het label managed-by: idp-platform
2. Argo CD AppProject aanmaken met een developer-rol
3. Vault KV v2 secrets-engine aanmaken met standaardsecrets en RBAC-policies
4. Harbor-project aanmaken
5. ESO SecretStore en ExternalSecret aanmaken in de projectnamespace

Het Argo CD-project wordt geconfigureerd met een developer-rol die volledige applicatiebeheersrechten biedt **uitsluitend binnen het eigen project**:

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: <project-name>
  namespace: argocd
spec:s
  destinations:
    - namespace: <project-name>-*
      server: https://kubernetes.default.svc
  sourceRepos:
    - '*'
  roles:
    - name: developer
      description: Developer role for <project-name>
      policies:
        - p, proj:<project-name>:developer, applications, *, <project-name>/*, allow
```

Figuur 18: Argo CD AppProject-manifest met developer-rol

Leden van de overeenkomstige AD-groep (een groep met dezelfde naam als het project) krijgen automatisch de developer-rol toegewezen via de Dex/OIDC-mapping die beschreven is in sectie 4.2.2.

Een complementaire DeleteProject-pipeline verwijdert alle bovenstaande onderdelen wanneer een project wordt afgesloten, zodat er geen verweesde resources achterblijven.

4.6.5. Service-onboardinggids

De service-onboardinggids is een vereiste voor Axxes en beschrijft stap voor stap hoe een ontwikkelteam hun eerste applicatie deployt op het IDP. De gids gaat ervan uit dat het platformteam de NewProject-pipeline al heeft uitgevoerd. Tabel 17 toont wat een team ontvangt na het aanmaken van een project.

Onderdeel	Details
Kubernetes-namespace	<project> - geïsoleerd met quota's en default-deny-NetworkPolicies
Argo CD-project	<project> - de AD-groep van het team heeft deployrechten
Harbor-project	harbor.stage.axxes.local/<project>/ + robot-accountcredentials
Vault secrets-engine	<project>/ - standaardsecrets voorgevuld op <project>/env
ESO auto-sync	<project>/env → K8s Secret <project>-env, vernieuwd elke minuut
kubectl-toegang	Via Dex OIDC - ontwikkelaar logt in met AD-credentials via de browser

Tabel 17: Wat een team ontvangt na projectcreatie

De werkwijze die een ontwikkelaar volgt is samengevat als:

```
platform.yaml schrijven → Pipeline configureren → Code pushen → Pipeline bouwt/scant/deployt → Applicatie live
```

De onboardinggids beschrijft elke stap met voorbeelden: hoe het platform.yaml-bestand eruitziet voor een eenvoudige webapplicatie, hoe de pipeline wordt geconfigureerd, hoe secrets worden beheerd via de Vault-webinterface en hoe de applicatie kan worden gemonitord via Argo CD en Grafana.

Na het toevoegen van de tweede deployment strategie via standaard HELM values, kunnen developers ook via deze manier een applicatie opstarten op het platform. Hiervoor schrijven developers rechtsreeks de HELM values file en wordt deze niet gegenereerd vanuit het platform.yaml script

4.6.6. Developertroubleshootinggids

Een andere vereiste van Axxes is de developertroubleshootinggids biedt ontwikkelteams een gestructureerd pad om veelvoorkomende problemen te diagnosticeren zonder het platformteam direct in te schakelen. De gids beschrijft vier scenario's:

Applicatie is onbereikbaar
Controle van de ingress-configuratie (is de IngressRoute correct?), de NetworkPolicies (staat het verkeer vanuit de ingress-controller toe?) en de DNS-resolutie (wijst het domein naar het juiste MetalLB-IP?).

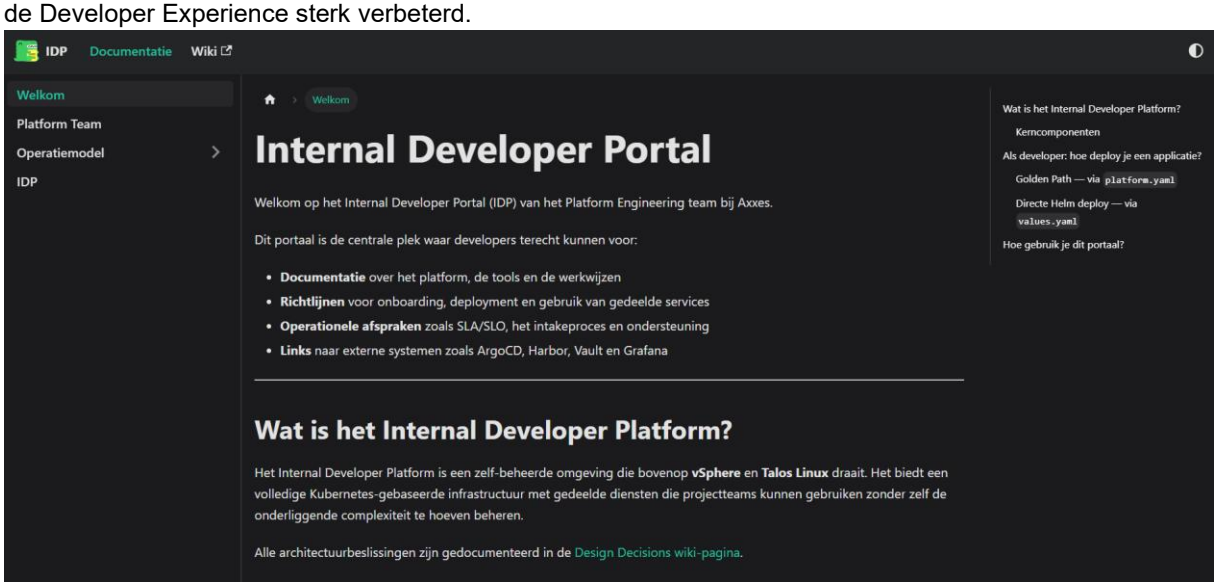
Pods crashen of starten niet
Loganalyse via de Argo CD-webinterface (die logs, events en resourcediffs toont zonder terminaltoegang) of via kubectl logs. Veelvoorkomende oorzaken zijn ontbrekende secrets, onvoldoende resourcelimieten of een verkeerd container-image.

Secrets niet beschikbaar
Controle van de ESO-synchronisatiestatus: bestaat de ExternalSecret in de namespace? Staat de SecretStore op Ready? Is het verwachte Vault-pad gevuld? Als een sleutel ontbreekt in Vault, kan de ontwikkelaar deze zelf toevoegen via de Vault-webinterface, ESO synchroniseert binnen één minuut.

Pipeline faalt
Per stap (build, SonarQube, Trivy, render, deploy) worden de meest voorkomende fouten en oplossingen beschreven. Bijvoorbeeld: een falende Trivy-scan duidt op kritieke kwetsbaarheden in het basis-image. De oplossing is het basis-image bijwerken (door het platformteam een nieuwere versie te laten spiegelen via de Harbor-mirroringpipeline) of de kwetsbare afhankelijkheid te verwijderen.

4.6.7. IDP-Portal: De First-Line of Support

Om de communicatie met ontwikkelteams te stroomlijnen is er een Internal Developer Portal (IDP-Portal) gelanceerd. Dit platform, gebouwd met Docusaurus, fungeert als de eerste laags ondersteuning voor ontwikkelaars. Het biedt een centraal overzicht van de platformdocumentatie, de SLA/SLO-afspraken, contactgegevens van het platformteam en een heldere uitleg van de onboarding- en probleemoplossingsprocessen. Door deze Self-Service documentatie wordt de communicatie efficiënter en de Developer Experience sterk verbeterd.



Figuur 19: Internal Developer Portal, developer hulplijn website

4.6.8. Progressive Delivery: Argo Rollouts

De standaard Kubernetes-deploymentstrategie vervangt pods naar een nieuwe versie via een rolling update zonder zicht op de gevolgen voor eindgebruikers. Er is geen ingebouwd mechanisme om slechts een deel van het verkeer naar de nieuwe versie te sturen of om de uitrol te pauzeren totdat een validatiestap is doorlopen. **Argo Rollouts** vervangt dit standaardgedrag door een stapsgewijze deploymentaanpak met twee strategieën: canary en blue-green.

Canary-deployment

Bij een canary-deployment wordt verkeer geleidelijk van de stabiele versie naar de nieuwe versie verschoven. De stappen zijn volledig configureerbaar in het values.yaml-bestand van de applicatie. Een typische canary-configuratie stuurt eerst twintig procent van het verkeer naar de nieuwe versie, pauzeert zodat het platformteam of de ontwikkelaar de gezondheid kan controleren, en verhoogt daarna de verhouding stapsgewijs totdat de volledige vloot is bijgewerkt. Elke stap kan ofwel handmatig worden gepromoteerd ofwel automatisch na een geconfigureerde wachttijd.

De base chart maakt bij een canary-configuratie automatisch alle benodigde Kubernetes-resources aan: een Rollout-resource die de standaard Deployment vervangt, een stabiele service die altijd naar de lopende versie wijst, een canary-service die naar de nieuwe versie wijst en een TraefikService die het verkeer gewogen verdeelt op basis van de ingestelde percentages. Argo Rollouts past de gewichten van de TraefikService automatisch aan bij elke stap.

Blue-green-deployment

Bij een blue-green-deployment wordt de nieuwe versie volledig naast de bestaande versie uitgerold zonder dat er al verkeer naartoe gaat. Pas na een expliciete promotie schakelt de stabiele service in één beweging over naar de nieuwe pods. Dit patroon is geschikt voor deployments die een handmatige goedkeuring vereisen voordat ze live gaan.

Bedieningsvlak

Promotie en terugdraaien zijn beschikbaar via drie kanalen: de kubectl Argo Rollouts-plugin, de Argo CD-webinterface en het toegevoegde Argo Rollouts-dashboard op https://argo-rollouts.stage.axxes.local. In dat dashboard is de actuele stap, de gewichtsverdeling en de podverdeling per rollout zichtbaar.

4.6.9. PR Preview Environments

Om feature branches te kunnen testen op een volledig operationele clusteromgeving zonder staging of productie aan te raken, zijn **PR Preview Environments** geïmplementeerd. Een preview-omgeving is een kortstondige, volledig gedeployde instantie van een applicatie die automatisch wordt aangemaakt wanneer een pull request wordt geopend en automatisch wordt verwijderd wanneer de pull request wordt gesloten of gemerged.

Werking
Twee componenten werken samen om preview-omgevingen te realiseren. De eerste component is een dedicated Bitbucket pull-requests-pipeline die automatisch wordt getriggerd bij elke push naar een PR-branch. Deze pipeline bouwt een containerimage getagd met de commit-SHA en maakt de benodigde Harbor pull secret aan in de preview-namespace. De tweede component is een Argo CD preview-environments ApplicationSet die open pull requests bewaakt. Zodra een PR-branch voldoet aan de vereiste naamconventie, maakt Argo CD automatisch een Application aan in een geïsoleerde namespace specifiek voor die pull request.

Branch-naamconventie
De naam van de branch is het stuurmechanisme voor het activeren van een preview-omgeving. Branches moeten het patroon preview/<app-name>/<beschrijving> volgen om automatisch door de ApplicationSet te worden opgepikt. Branches die niet met preview/ beginnen worden genegeerd.

Automatische overschrijvingen
Argo CD leest de values.yaml van de PR-branch en past een vaste set overschrijvingen toe om de preview-omgeving lichtgewicht en geïsoleerd te houden. Het aantal replica's wordt teruggebracht naar één, HPA, VPA en Reloader worden uitgeschakeld, secrets worden verwijderd en er worden geen PVC's aangemaakt. Elke preview-omgeving krijgt een eigen ingress-URL in de vorm https://<app-name>-pr-<PR#>.stage.axxes.local, waarbij TLS automatisch wordt ingeschakeld via de platform-CA.

Levenscyclus

Gebeurtenis	Actie
PR opent	Argo CD maakt namespace en deployt de applicatie automatisch
Push naar PR-branch	Pipeline herbouwt de image; Argo CD synchroniseert opnieuw
PR gesloten of gemerged	Argo CD verwijdert automatisch de namespace en alle bijbehorende resources

4.7. Ondersteunende infrastructuur

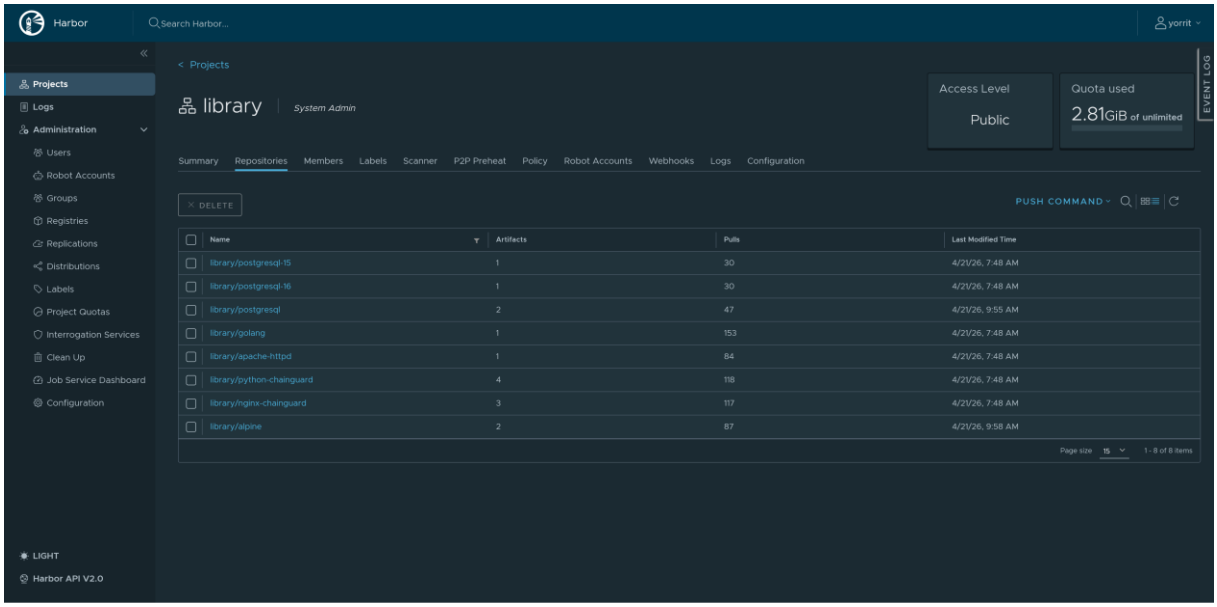
4.7.1. Harbor: Image Mirroring

Het platform bevat een geautomatiseerde pipeline (harbor-mirror-images) die goedgekeurde upstream-images spiegelt naar Harbor. Het primaire doel is beveiligingscontrole: platformengineers beoordelen upstream-images en spiegelen alleen goedgekeurde versies naar de interne registry. Ontwikkelaars pullen uitsluitend vanuit Harbor, directe pulls vanuit Docker Hub of andere externe registries zijn niet toegestaan. De pipeline leest een images.yaml-bestand dat de te spiegelen images definieert en voert een shellscript uit dat de images pullt vanuit de oorspronkelijke bron, hertagd naar het Harbor-formaat en pusht naar harbor.stage.axxes.local/library/.
Tabel 18 toont de op dit moment gespiegelde images.

Image	Bron	Opmerkingen
alpine	docker.io/library/alpine	Lichtgewicht basisimage
golang:1.22-alpine	docker.io/library/golang:1.22-alpine	Build-toolchain voor Go-applicaties
nginx-chainguard	cgr.dev/chainguard/nginx:latest	Gehard nginx-image met minimale aanvalsvector
python-chainguard	cgr.dev/chainguard/python:latest	Gehard Python-image met minimale aanvalsvector
apache-httpd	docker.io/library/httpd	Apache HTTP-server
postgresql:17	ghcr.io/cloudnative-pg/postgresql:17	Database-basisimage voor CloudNativePG
postgresql:16	ghcr.io/cloudnative-pg/postgresql:16	Database-basisimage voor CloudNativePG
postgresql:15	ghcr.io/cloudnative-pg/postgresql:15	Database-basisimage voor CloudNativePG

Tabel 18: Gespiegelde upstream-images in Harbor

Naast beveiligingscontrole biedt image mirroring nog twee voordelen: het vermijdt de rate limits van Docker Hub (die publieke pulls beperken tot 100 per zes uur per IP-adres) en het maakt het platform air-gap-klaar, als de internetverbinding wegvalt, blijven alle benodigde images beschikbaar in de interne registry.



Figuur 20: Harbor upstream image mirooring registry

4.7.2. DNS en Active Directory

Een eigen Active Directory-domein (stage.axxes.local) is opgezet op een Windows Server-VM als centrale identiteitsbron voor het platform. Alle platformcomponenten die gebruikersauthenticatie vereisen, Argo CD (via Dex), Grafana, Harbor en Vault, authenticeren tegen dit AD via LDAD.
Op dezelfde Windows Server-VM draait een DNS-server die alle interne domeinnamen van het platform oplost. De DNS-records wijzen naar de MetalLB-loadbalancer-IP's die Traefik als ingresscontroller gebruikt. Wanneer een ontwikkelaar https://argocd.stage.axxes.local opent in de browser, lost de DNS-server dit op naar het MetalLB-IP, dat Traefik doorstuurt naar de Argo CD-pods in het cluster.
Het opzetten van een eigen AD en DNS was niet voorzien in het oorspronkelijke projectplan, maar bleek noodzakelijk om een realistische, zelfstandige platformomgeving te creëren. Dit is één van de scope-uitbreidingen die gedurende de stage zijn toegevoegd.

4.7.3. Pipelinesplitsing over twee VLAN-runners

De bootstrappipeline is opgesplitst over twee zelfgehoste Bitbucket-runners vanwege de VLAN-segmentatie van de on-premises-omgeving:

Runner	Locatie (VLAN)	Verantwoordelijkheid
Windows-runner	vCenter-VLAN	Deployt VM's vanuit OVA op vSphere, injecteert Talos-machineconfiguraties, exporteert outputs als pipelineartefact
Linux-runner	Lab-VLAN	Verbindt met de Talos API, bootstrapt etcd, installeert applicaties via Helm, configureert platformcomponenten

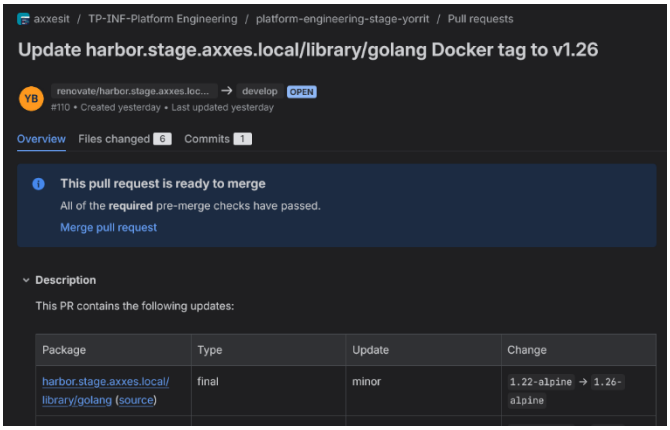
Tabel 19: Bitbucket-runners en hun verantwoordelijkheden

State wordt doorgegeven tussen de runners als een Bitbucket-pipelineartefact (vm_creation.outputs.json). De eerste runner (Windows) schrijft de outputs van de VM-creatie naar dit JSON-bestand en de tweede runner (Linux) leest het in om te weten welke IP-adressen de nodes hebben en hoe de Talos-clientconfiguratie eruitziet.

Deze splitsing was een onverwachte uitdaging vroeg in de stage. Het initiële ontwerp ging uit van één runner met toegang tot beide netwerken, maar de VLAN-segmentatie maakte dit onmogelijk. De oplossing, het doorgeven van Terraform-outputs als JSON-artefact tussen runners, is robuust gebleken en draait zonder problemen in de dagelijkse werking. De volledige achtergrond is beschreven in de lessons learned (hoofdstuk 5).

4.7.4. Geautomatiseerd Afhankelijkheidsbeheer met Renovate

Om de veiligheid en actualiteit van het platform proactief te waarborgen, draait Renovate als een cronjob on-cluster. Renovate scant de Git-repositories continu op verouderde pakketten, Helm-charts en container images en stelt automatisch Pull Requests voor om deze bij te werken. Belangrijke configuraties (zoals Vault API variabelen en het GitHub-token benodigd voor het ophalen van release notes) worden veilig opgehaald via de Vault/External Secrets Operator integratie.



Figuur 21: Renovate PullRequest update seggestie

5. Lessons Learned

Dit hoofdstuk beschrijft de belangrijkste lessen die tijdens de stage zijn opgedaan. Het gaat niet om fouten, maar om situaties waarin de oorspronkelijke aanpak moest worden bijgestuurd, waarin onverwachte complexiteit opdook, of waarin een pragmatische oplossing werd gevonden voor een probleem dat niet voorzien was in het projectplan.

5.1. VLAN-segmentatie en pipelinesplitsing

Het probleem
Het oorspronkelijke ontwerp ging uit van één zelfgehoste Bitbucket-runner die toegang had tot zowel de vSphere/vCenter-API als het Talos-clusternetwerk. Bij de eerste implementatie bleek dat de on-premises-omgeving is opgedeeld in twee gescheiden VLAN's: het vCenter-beheernetwerk en het lab-VLAN waar het Talos-cluster draait. Geen enkele runner kan beide netwerken bereiken.

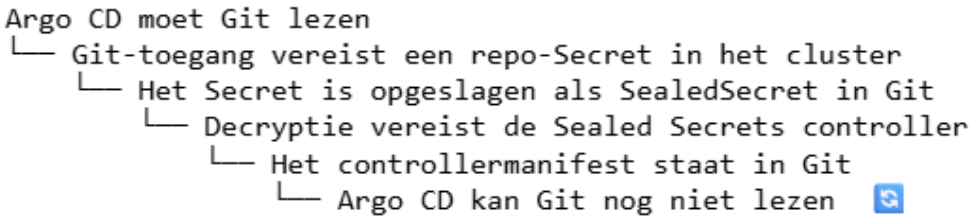
De impact
De volledige bootstrappipeline, die in het oorspronkelijke ontwerp als één doorlopend Terraform-proces was opgezet, moest worden opgesplitst over twee runners: een Windows-runner in het vCenter-VLAN voor de VM-creatie en een Linux-runner in het lab-VLAN voor de clusterbootstrap en applicatie-installatie.

De oplossing
Terraform-outputs uit fase 1 (IP-adressen, machineconfiguraties, clustersecrets) worden geëxporteerd als een JSON-bestand (vm_creation.outputs.json) en als Bitbucket-pipelineartefact doorgegeven aan de volgende fase. De tweede runner leest dit artefact in en gebruikt de waarden als invoer voor fase 2. Deze aanpak is robuust gebleken: het artefact is deterministisch (dezelfde Terraform-state levert altijd hetzelfde JSON-bestand op) en de pipeline faalt expliciet als het artefact ontbreekt of ongeldig is.

De les
Bij het ontwerpen van een CI/CD-pipeline voor een on-premises-omgeving is het essentieel om vroegtijdig de netwerktopologie in kaart te brengen. De aanname dat één runner alle systemen kan bereiken, gaat in veel bedrijfsomgevingen niet op. Het opsplitsen van pipelines over meerdere runners met artefactoverdracht is een patroon dat breder toepasbaar is dan alleen deze stage.

5.2. Chicken-and-egg-bootstrapprobleem

Het probleem
Een zelfgehost platform heeft vaak circulaire afhankelijkheden. Het meest prominente voorbeeld: Argo CD moet een SSH-sleutel hebben om de private Git-repository te klonen, maar die SSH-sleutel moet als Kubernetes Secret bestaan vóórdat Argo CD kan starten. Tegelijkertijd is het de bedoeling dat Argo CD zélf alle Secrets beheert via GitOps.



De oplossing
De bootstrappipeline doorbreekt de cirkel door de eerste iteratie buiten het GitOps-model om af te handelen. Terraform installeert eerst Argo CD en injecteert vervolgens het repo-Secret rechtstreeks via

kubectl apply. Pas daarna kan Argo CD de ApplicationSet synchroniseren vanuit Git en het normale GitOps-beheer overnemen.

De les
Bij het bouwen van een GitOps-platform moet expliciet worden nagedacht over de bootstrapvolgorde. Niet alles kan via GitOps worden beheerd, er is altijd een initiële opstart nodig die buiten het model om wordt geïnjecteerd. Het is belangrijk om deze uitzondering bewust en gedocumenteerd te maken, zodat toekomstige platformengineers begrijpen waarom bepaalde stappen buiten Argo CD om verlopen.

5.3. Scope-uitbreidingen

Drie onderdelen die niet in het oorspronkelijke projectplan stonden, bleken gedurende de stage noodzakelijk:

1. **Active Directory en DNS**: het opzetten van een eigen AD-domein (stage.axxes.local) en DNS-server was nodig om een realistische, zelfstandige platformomgeving te creëren met consistente authenticatie over alle platformcomponenten.
2. **Bitbucket-runners**: het configureren van twee zelfgehoste runners (Windows en Linux) op gescheiden VLAN's was nodig vanwege de netwerksegmentatie die pas bij de eerste implementatie aan het licht kwam.
3. **RustFS als backupdoel**: het opzetten van een S3-compatibele objectopslagserver buiten het cluster was nodig om Longhorn-backups te kunnen maken die een clusteruitval overleven.

Deze uitbreidingen hebben de scope van de stage vergroot maar hebben tegelijkertijd het platform realistischer en completer gemaakt. Ze zijn gedocumenteerd als bewuste toevoegingen en niet als scope creep, omdat ze direct voortkwamen uit functionele of niet-functionele vereisten die anders niet vervuld konden worden.

5.4. Resiliency door Stress en Node Failure Testing

Om de werkelijke robuustheid van de georkestreerde architectuur te valideren, is er besloten om niet uit te gaan van aannames, maar de cluster fysiek op de proef te stellen door middel van uitvoerige testfases en resilience testing uit te voeren: **Baseline HTTP Load Testing** (k6) en **Node Failure Testing**.

HTTP Load Testing (K6): Er werd een gecontroleerde stress-test uitgevoerd op een demo-applicatie om het verzadigingspunt van het cluster te bepalen. Uit een initiële stresstest (*smoke test*) bleek de responstijd van het cluster (inclusief Ingress, Traefik en de CloudNativePG database) extreem consistent: 95% latencies op de applicatie-endpoint bleven onder de 15ms, met 0% foutmarge. Dit bewijst dat intern verkeer zonder enige performancehinder afgehandeld wordt.

Node Resilience Testing (Graceful & Power-cycle): Vervolgens werden zes verschillende faalscenario's uitgetest, gericht op worker- én control-plane-nodes. Hierin werden zowel "Graceful" herstarts via de API gesimuleerd, als onverwachte "Power-cycle crashes" waarbij de netwerkverbinding of stroomtoevoer onmiddellijk werd doorgesneden.

- **Graceful Restarts:** Bij elegante afkoppeling van enkelvoudige nodes en zelfs bij mitigatie van twéé gelijktijdige nodes, behaalde het platform een vlekkeloze 100.00% HTTP success rate. Talos Linux wist de pods keurig via PodDisruptionBudgets (PDB) te migreren (draining) zonder dat eindgebruikers verbindingen verloren.
- **Hard Crashes (Power-Cycle):** Het systeem functioneerde opmerkelijk robuust bij harde crashes van een afzonderlijke node. De detectietijd bevond zich rond de 15s-93s waarna pods vrijwel direct (1-16s) succesvol hersteld werden. Zelfs bij een abrupte exit van een control-plane master lag de fail-rate van HTTP-requests verwaarloosbaar laag (99.63% succes, ofwel slechts 1 dropped request exact op het moment van de crash).
- **Grenzen van het 5-node cluster:** De limiet van het geadopteerde design (3 masters, 2 workers) werd blootgelegd tijdens de *two-node hard crash*, waarbij gelijktijdig een worker en een master de

stroom verloren. Hier werd een tijdelijke PDB-schending geconstateerd en een aanzienlijke wachtrij in Longhorn-verstoring (faulted engines en unexpected detaches). Hoewel ook daar alles automatisch wíst te herstellen na heropstart, markeert het dat het down-schalen van >40% van het complete systeem zonder vooraankondiging te groot is voor ononderbroken traffic en policy-garanties.

In zijn algemeenheid heeft deze testfase onomstotelijk aangetoond dat de combinatie van Talos, Longhorn en Traefik zeer betrouwbare en dynamisch robuuste veerkracht vertoont bij het overgrote deel van potentiële infrastructuurfouten.

5.5. De Waarde van een Gecombineerd Deployment Model

De introductie van de geavanceerde Helm "Base Chart"-methode naast de oorspronkelijke platform.yaml abstractie heeft een waardevol inzicht opgeleverd. Hoewel een zwaar geabstraheerd IDP perfect is voor beginnende teams die maximale Cognitive Load Reductie zoeken, vragen ervaren ops-gerichte teams uiteindelijk toch naar de flexibiliteit van native Kubernetes tooling (zoals complexe Helm values en custom alert regels). Door beide in een Golden Path "self-service" aan te bieden, wordt de adoptiegraad significant vergroot zonder standaarden los te laten.

5.6. TLS-Trust en Self-Signed Certificates

Omdat "Zero-Trust" en on-premises security prioriteiten waren, is direct gekozen voor volledige TLS-versleuteling (HTTPS) over alle interne netwerken van het platform. Hier liep het platform bootstrapproces tegen sommige obstakels aan. Omdat de verbindingen beveiligd zijn door een interne (self-signed) root Certificate Authority, weigerden diverse scripts, pipelines of IDP tools (zoals Dex OIDC authenticatie voor Kubernetes API of webbrowsers) om via deze eindpunten te communiceren (Fout: *x509: certificate signed by unknown authority*).s

6. Besluit

Deze stage had als doel een MVP van een Internal Developer Platform te ontwerpen, documenteren en realiseren op een on-premises VMware vSphere-omgeving bij Axxes. Het platform moest volledig declaratief beheerd worden volgens GitOps-principes, met geautomatiseerde CI/CD-pipelines, observability, beveiligingsbeleid en een golden path voor ontwikkelteams.

6.1. Bereikte resultaten

Het resultaat is een werkend on-premises Kubernetes-platform dat het volledige traject dekt van infrastructuurprovisioning tot applicatiedeployment:

- **Geautomatiseerde clusterprovisioning**: het volledige platform kan vanaf nul worden opgebouwd via één Bitbucket-pipeline, opgesplitst over drie fasen (VM-creatie, clusterbootstrap, applicatie-installatie). Een terraform destroy gevolgd door een nieuwe pipeline-run levert een identiek platform op.
- **GitOps-gebaseerd applicatiebeheer**: Argo CD beheert alle platformcomponenten en teamapplicaties via het App-of-Apps-patroon. Wijzigingen aan het platform worden aangebracht door Git-commits, niet door handmatige commando's.
- **Gecentraliseerd geheimenbeheer**: HashiCorp Vault biedt per-project-geïsoleerde secrets met LDAP-gebaseerde toegangscontrole en auditlogging. External Secrets Operator synchroniseert secrets automatisch naar Kubernetes.
- **Observability**: VictoriaMetrics, Grafana, VictoriaLogs en Alertmanager bieden out-of-the-box metrics, logging, dashboarding en alerting, inclusief meldingen naar Microsoft Teams.
- **Beveiligingsbeleid**: Kyverno dwingt default-deny-NetworkPolicies en verplichte labels af. Alle ingress-verkeer is TLS-beveiligd via cert-manager. Talos Linux biedt een immutable OS zonder SSH-toegang.
- **Golden path**: ontwikkelteams kunnen via één platform.yaml-bestand en een gestandaardiseerde pipeline een applicatie deployen, inclusief container-image, ingress, secrets, database en monitoring.
- **Projectmanagement**: het aanmaken van een nieuw team (namespace, Argo CD-project, Vault-engine, Harbor-project, ESO-synchronisatie) is volledig geautomatiseerd via een dedicated pipeline.
- **Golden path evolutie:** het platform biedt niet alleen een platform.yaml methode, maar bied ook direct een Helm Variant Deployment via een centraal beheerde base chart. Dit elimineerde dubbel templating-werk en verschoonde de Harbor registries per applicatie. Opleveren is hiermee super elastisch: voor zowel beginnende developers als heavy-users is er een passend pad.
- **Multi-Environment & Promotion Flows:** de pipelines zijn geëvolueerd naar volwaardige multi-stage straten waarbij gescheiden omgevingen (staging, productie) en de bijhorende Argo CD configuraties volledig zonder tussenkomst worden uitgerold en doorgeschoven.
- **Zero-Trust & Afhankelijkheidsbeheer:** de introductie van Cosign zorgt voor cryptografische (supply-chain) garantie bij elke nieuwe applicatierelease, een strenge Kyverno 'no-run-as-root' policy dwingt veiligheid binnen containers af en een in-cluster implementatie van Renovate voorziet het hele platform doorlopend van automatische updateverzoeken.
- **Developer Experience (IDP-Portal):** om de kloof tussen platformbeheerders en ontwikkelaars resoluut te dichten is er een actief intern ontwikkelaarsportaal in leven geroepen op basis van Docusaurus, wat dient als de cruciale eerste lijn van "self-service" documentatie.
- **Continu kwetsbaarheidsscanning:** de Trivy Operator bewaakt alle draaiende workloads in het cluster en rapporteert nieuwe CVE's via Grafana, als aanvulling op de Trivy-scan in de CI/CD-pipeline.
- **Progressive Delivery:** Argo Rollouts biedt canary- en blue-green-deployments als opt-in golden path-functionaliteit, waardoor teams gecontroleerd kunnen uitrollen zonder downtime.

- **PR Preview Environments:** feature branches resulteren automatisch in een volledig operationele preview-omgeving met eigen URL en TLS, die bij het sluiten van de pull request automatisch wordt opgeruimd.
- **Automatische Secret-propagatie:** Stakater Reloader herstart pods automatisch bij Vault- en ESO-secretwijzigingen, waardoor het volledige Vault-naar-pod-traject zonder handmatige tussenkomst verloopt.
- **RabbitMQ als Self-Service:** ontwikkelteams kunnen een asynchrone berichtenwachtrij aanvragen via één vlag in hun values.yaml, zonder Kubernetes-resources handmatig te schrijven.
- **Schijfbeheer via kubelet GC:** een Talos machineconfiguratispatch configureert proactieve garbagecollection van containerimages, waarmee schijfvulling op de clusternodes wordt voorkomen.

6.2. Meerwaarde voor Axxes

De primaire waarde van deze stage voor Axxes ligt niet in het één-op-één overnemen van het gebouwde platform, maar in de leeropbrengst en de herbruikbare patronen. Het project levert:

- **ADR's (Architecture Decision Records)**: elke technologiekeuze is onderbouwd met een vergelijking van alternatieven en een expliciete motivatie, herbruikbaar als referentiemateriaal bij klantopdrachten.
- **Standaardisatierichtlijnen**: voor Git, Kubernetes-naamgeving en Terraform, toepasbaar onafhankelijk van het specifieke platform.
- **Runbooks**: stapsgewijze operationele gidsen voor elke platformcomponent, van bootstrap tot dag-2-beheer.
- **Golden path en onboardinggids**: een concreet voorbeeld van hoe een developer experience er op een IDP uit kan zien, inclusief pipeline-templates en troubleshootinggidsen.

6.3. Reflectie op het proces

De stage heeft bevestigd dat Platform Engineering een discipline is die niet alleen technische kennis vereist, maar ook een sterk bewustzijn van de ontwikkelaarservaring. De keuze voor tools is slechts een deel van het verhaal, de manier waarop die tools worden samengevoegd tot een coherent, gedocumenteerd en reproduceerbaar geheel bepaalt uiteindelijk de waarde van het platform.
De lessons learned (hoofdstuk 5) tonen aan dat een on-premises-omgeving specifieke uitdagingen met zich meebrengt die in een cloudgebaseerde omgeving niet of minder spelen: netwerksegmentatie, het ontbreken van managed services en de noodzaak om ondersteunende infrastructuur (AD, DNS, S3-opslag) zelf op te zetten. Deze ervaringen zijn waardevolle lessen voor toekomstige platformtrajecten bij Axxes en hun klanten die eveneens on-premises werken.

6.4. Toekomstige uitbreidingen

Het platform biedt ruimte voor verdere uitbreiding na de stageperiode:

- **IDP-frontend**: een self-service-webinterface waarmee teams zelf projecten kunnen aanmaken, applicaties kunnen deployen en de status van hun resources kunnen bekijken, zonder de Bitbucket-pipelines rechtstreeks te moeten triggeren.
- **Multi-clusterondersteuning**: Argo CD en Vault zijn beide geschikt voor multi-clusterbeheer, wat het platform zou kunnen uitbreiden naar meerdere omgevingen (ontwikkeling, staging, productie).

Begrippenlijst (Glossary)

TERM	DEFINITIE
COGNITIVE LOAD (REDUCTION)	De hoeveelheid mentale inspanning die vereist is om een taak uit te voeren. In softwareontwikkeling probeert men dit te verlagen door infrastructuurcomplexiteit weg te nemen bij de ontwikkelaar.
DECLARATIEF	Een configuratiemodel waarbij men de gewenste eindtoestand (de "what") beschrijft, in plaats van de exacte stappen om daar te komen (de "how"). De tooling zorgt vervolgens voor de uitvoering.
DEVELOPER EXPERIENCE (DX)	De algehele ervaring, efficiëntie en tevredenheid van ontwikkelaars bij het gebruik van interne tools, processen en systemen.
EVERYTHING AS CODE (EAC)	De praktijk waarbij infrastructuur, configuratie, beveiligingsbeleid en pipelines volledig in code (vaak in Git) worden gedefinieerd en beheerd.
GITOPS	Een operationeel model waarbij Git fungeert als de 'single source of truth'. Geautomatiseerde processen (zoals Argo CD) zorgen ervoor dat de infrastructuur en applicaties exact overeenkomen met wat in Git beschreven staat.
GOLDEN PATH	Een vooraf gedefinieerd, ondersteund en best-practice pad voor ontwikkelaars om software of infrastructuur te bouwen en te deployen, bedoeld om complexiteit weg te nemen.
IMMUTABLE OS	Een besturingssysteem (zoals Talos Linux) dat na het opstarten niet meer gemodificeerd kan worden (read-only). Updates vereisen het vervangen van de gehele image, wat zorgt voor hogere veiligheid en reproduceerbaarheid.
INTERNAL DEVELOPER PLATFORM (IDP)	Een verzameling van gecentraliseerde tools, API's en diensten die door een platformteam wordt aangeboden om ontwikkelaars te helpen bij het versnellen van de software delivery.
PLATFORM AS A PRODUCT	Een denkwijze waarbij interne infrastructuurtools en het IDP worden gebouwd met dezelfde focus op gebruikersbehoeften, productmanagement en klanttevredenheid (in dit geval de ontwikkelaars) als commerciële producten.
PLATFORM ENGINEERING	De technische discipline en praktijk van het bouwen van toolchains en workflows (zoals een IDP) die softwareontwikkeling versnellen en standaardiseren in complexe organisaties.
SELF-SERVICE INFRASTRUCTURE	Een cloud- of platformmodel waarbij ontwikkelteams on-demand infrastructuurbronnen (zoals databases of namespaces) kunnen uitrollen zonder menselijke tussenkomst van een ops-team.
SHIFT-LEFT SECURITY	Het principe van het verplaatsen van beveiligingscontroles en testfases naar een eerder (linker) moment in het ontwikkelproces, zoals rechtstreeks in de CI/CD-pipeline.
CONTAINER / CONTAINER IMAGE	Een container is een lichtgewicht, geïsoleerde runtime-eenheid die een applicatie samen met al haar afhankelijkheden verpakt. Een container image is het onveranderlijke pakket (blueprint) waaruit een container wordt gestart, opgebouwd uit lagen en opgeslagen in een container registry (zoals Harbor)
INGRESS	Een Kubernetes-object dat extern HTTP(S)-verkeer routeert naar interne services binnen het cluster, op basis van regels zoals hostnamen en URL-paden. Een Ingress Controller (zoals Traefik) voert deze regels effectief uit.
HELM	Een pakketbeheerder voor Kubernetes waarmee applicaties worden gedefinieerd, geïnstalleerd en beheerd via herbruikbare templates (zogenaamde Helm Charts). Dit vereenvoudigt het deployen van complexe applicaties met meerdere Kubernetes-resources.
NAMESPACE	Een logische scheiding binnen een Kubernetes-cluster waarmee resources (zoals pods, services en secrets) worden gegroepeerd en geïsoleerd. Dit maakt multi-tenancy en toegangscontrole mogelijk.

Verantwoording AI-gebruik

Bij het opstellen van dit realisatiedocument is gebruikgemaakt van een generatieve AI-tool (een large language model). Hieronder wordt transparant toegelicht waarvoor de tool is ingezet en welke beperkingen daarbij in acht zijn genomen.

Waarvoor de AI-tool is gebruikt

De AI-tool is ingezet als hulpmiddel bij het **structureren en redigeren** van dit document, niet voor het bedenken van de inhoud zelf. Concreet is de tool gebruikt voor:

- het opstellen en verfijnen van de **hoofdstuk- en sectiekoppen** en de algemene documentstructuur;
- het **herformuleren van mijn eigen teksten** naar een meer formele en professionele schrijfstijl;
- het **verbeteren van de leesbaarheid**, zinsopbouw, spelling en consistentie van de tekst.

De volledige technische inhoud, de ontwerpkeuzes, de implementatie, de testresultaten en de lessons learned zijn gebaseerd op mijn eigen werk tijdens de stage. De AI-tool heeft uitsluitend gediend ter ondersteuning van de redactie en vormgeving van bestaande, door mij aangeleverde informatie.

Gebruikte prompts (voorbeelden)

De tool is iteratief aangestuurd met prompts in de geest van:

- "Stel een logische hoofdstuk- en sectiestructuur voor een realisatiedocument over een Internal Developer Platform op basis van de meegegeven project documentatie."
- "Herschrijf de volgende tekst naar een formelere en professionelere schrijfstijl, met behoud van de technische inhoud."
- "Controleer deze tekst op leesbaarheid, zinsbouw en consistentie en stel verbeteringen voor."

Beperkingen en onzekerheden

Een AI-tool levert geen gezaghebbende of definitieve informatie en kan fouten of onnauwkeurigheden bevatten. De gegenereerde en geredigeerde teksten zijn daarom telkens nagelezen en waar nodig bijgestuurd, zodat de inhoud aansluit bij wat tijdens de stage daadwerkelijk is uitgevoerd. De technische inhoud en de ontwerpkeuzes in dit document komen voort uit het eigen werk; de AI-tool is enkel ondersteunend ingezet bij de structuur en formulering, en niet als bron van technische kennis.

Literatuurlijst

Sidero Labs. (2026). Talos Linux documentation. https://www.talos.dev/v1.9/

The Harbor Authors. (2026). Harbor documentation. https://goharbor.io/docs/

Argo CD Authors. (2026). Argo CD documentation. https://argo-cd.readthedocs.io/en/stable/

VictoriaMetrics. (2026). VictoriaMetrics documentation. https://docs.victoriametrics.com/

HashiCorp. (2026). Vault documentation. https://developer.hashicorp.com/vault/docs

External Secrets Authors. (2026). External Secrets Operator documentation. https://external-secrets.io/

Longhorn Authors. (2026). Longhorn documentation. https://longhorn.io/docs/

CloudNativePG Authors. (2026). CloudNativePG documentation. https://cloudnative-pg.io/documentation/

Kyverno Authors. (2026). Kyverno documentation. https://kyverno.io/docs/

Traefik Labs. (2026). Traefik documentation. https://doc.traefik.io/traefik/

MetalLB Authors. (2026). MetalLB documentation. https://metallb.universe.tf/

Sigstore Authors. (2026). Cosign documentation. https://docs.sigstore.dev/cosign/

cert-manager Authors. (2026). cert-manager documentation. https://cert-manager.io/docs/

HashiCorp. (2026). Terraform vSphere provider documentation.
https://registry.terraform.io/providers/hashicorp/vsphere/latest/docs

Renovate. (2026). Renovate documentation. https://docs.renovatebot.com/

Grafana Labs. (2026). Grafana documentation. https://grafana.com/docs/grafana/latest/

Prometheus Authors. (2026). Alertmanager documentation.
https://prometheus.io/docs/alerting/latest/alertmanager/

Grafana Labs. (2026). k6 documentation. https://grafana.com/docs/k6/latest/

Docusaurus. (2026). Docusaurus documentation. https://docusaurus.io/docs

Argo Rollouts Authors. (2026). Argo Rollouts documentation. https://argoproj.github.io/rollouts/

RabbitMQ Authors. (2026). RabbitMQ Cluster Operator documentation.
https://www.rabbitmq.com/kubernetes/operator/operator-overview

Stakater. (2026). Reloader documentation. https://github.com/stakater/Reloader

Aqua Security. (2026). Trivy Operator documentation. https://aquasecurity.github.io/trivy-operator/

Platform Engineering (2026). Documentation https://platformengineering.org/

Maes, Stijn. (Stagementor). (2026). Persoonlijke communicatie [Mondeling overleg].

Tijtgat, Toon. (Platform Engineering Team Lead). (2026). Persoonlijke communicatie [Mondeling overleg].