

# Secure-by-default: ein Plattform-Engineering- Baukasten

INNOVATE. INTEGRATE. EXCEED.



inovex

**Wir glauben:**

Kognitive Last ist zu einem der größten  
Sicherheitsrisiken von cloud-nativen  
IT-Architekturen geworden

- Anbieter für smarte Schließsysteme für den Heimbedarf
- Mittelständischer Betrieb (500 MA)
- 7 Entwickler Teams



## **Bob**

Neustarter bei SecureHome

Fokus auf Backend Entwicklung

Tech Stack: Spring Boot, Postgres, Docker



## Bobs Auftrag:

Entwicklung einer cloud-nativen  
Anwendung für die Verwaltung  
von biometrischen Schlüsseln  
für smarte Türschlösser



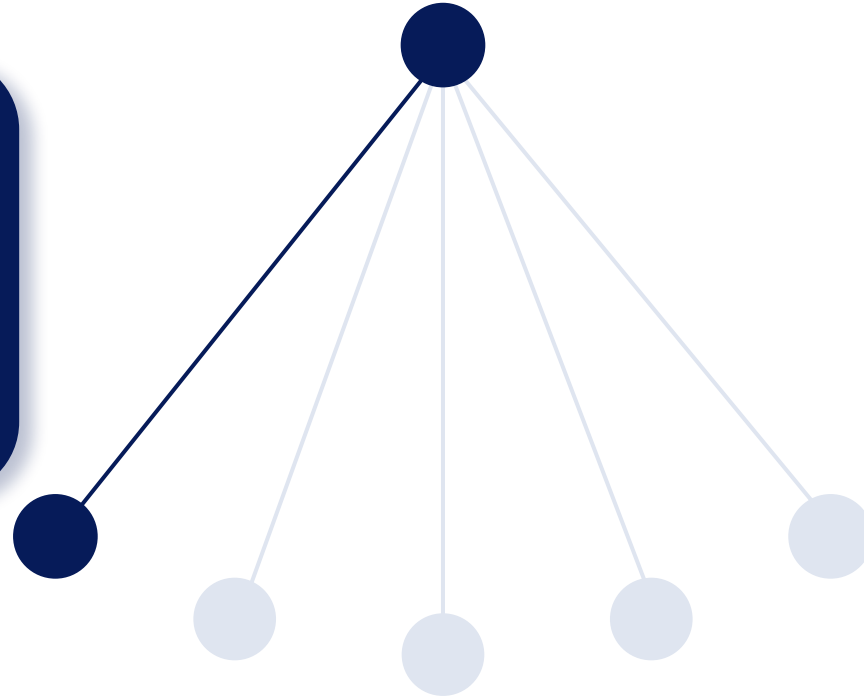
# SecureHome Rückblick

## Zentrale Kontrolle

### (TicketOps):

Starke Kontrolle und  
Konsistenz, aber oft  
langsam und ein

**Innovationshemmnis.**



## Dezentrale Autonomie

### (DevOps):

Hohe Geschwindigkeit und  
Team-Autonomie, ohne  
zentrale Leitplanken

besteht das Risiko von  
Inkonsistenz und

**Sicherheitslücken.**



inovex

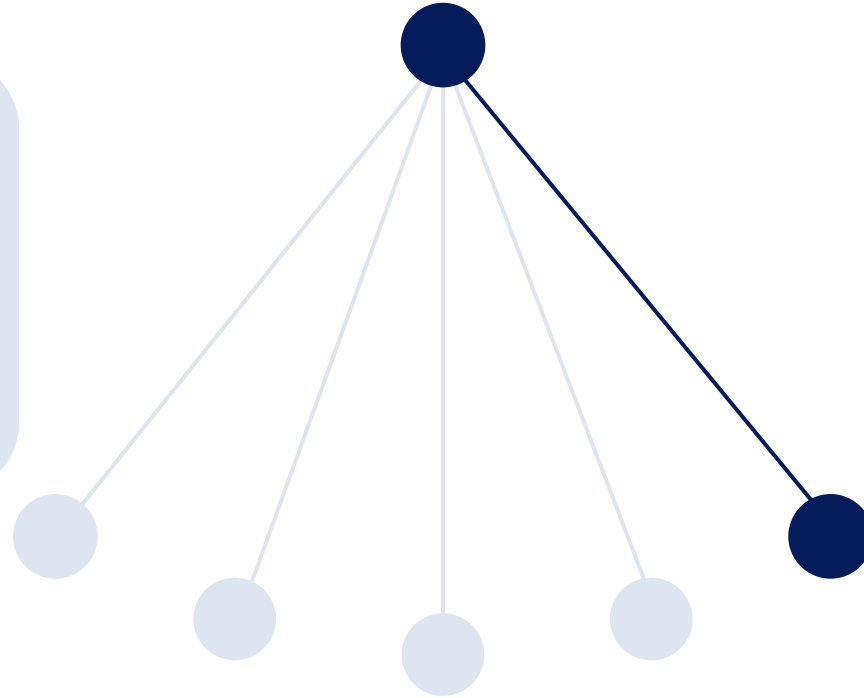
# SecureHome Rückblick

## Zentrale Kontrolle

### (TicketOps):

Starke Kontrolle und  
Konsistenz, aber oft  
langsam und ein

**Innovationshemmnis.**



## Dezentrale Autonomie

### (DevOps):

Hohe Geschwindigkeit und  
Team-Autonomie, ohne  
zentrale Leitplanken

besteht das Risiko von  
Inkonsistenz und

**Sicherheitslücken.**



inovex

# Explosion der Komplexität



**2018**



**2025**

# Kognitive Last als Risikofaktor

## **Security:**

SAST, DAST, SCA, IAM, RBAC, Secret Management, Encryption, OWASP Top 10, Zero Trust, Threat Modeling, CVEs, SBOM....

## **Infrastruktur & Betrieb:**

CI/CD-Pipelines, Infrastructure as Code (IaC),  
Kubernetes-Konfiguration, Datenbanken, Backup, Monitoring, Alerting,  
PKI, Updates von Tools und Dependencies...

## **Governance:**

CRA, NIS2, KRITIS, ISO 27001, DSGVO, Audit Logs, Incident Response,  
Disaster Recovery....

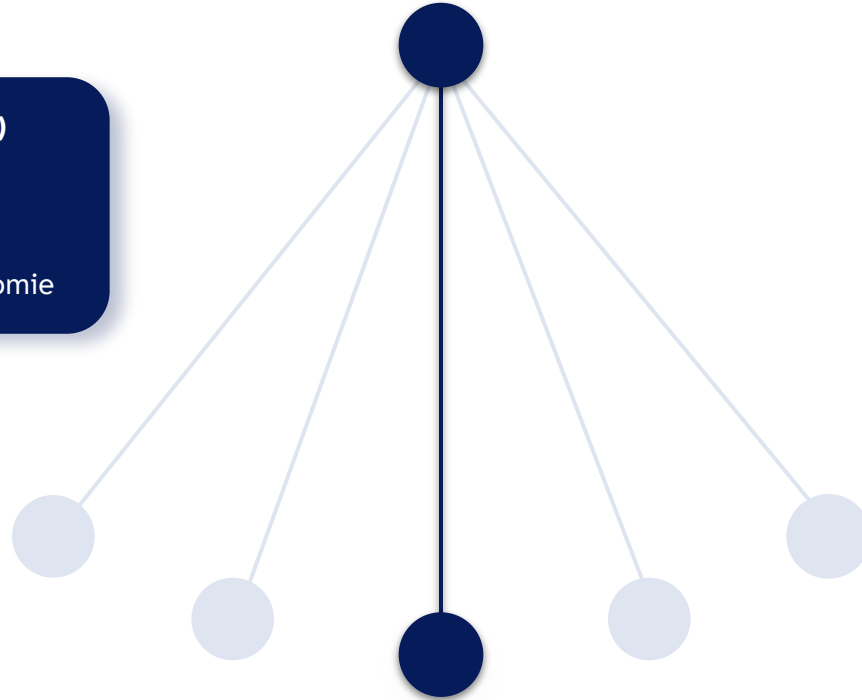


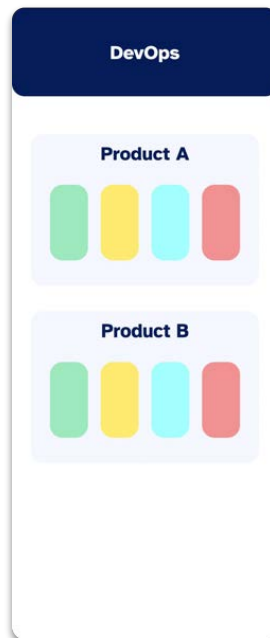
Die Frage ist nicht, **ob** Bob hier etwas übersehen wird, sondern **wann**

# Die Dev(Sec)Ops Evolution - Platform Engineering

## IDP (Platform Engineering)

Balance zwischen zentralen,  
sicheren Standards und  
maximaler Entwickler-Autonomie





**Product Management**



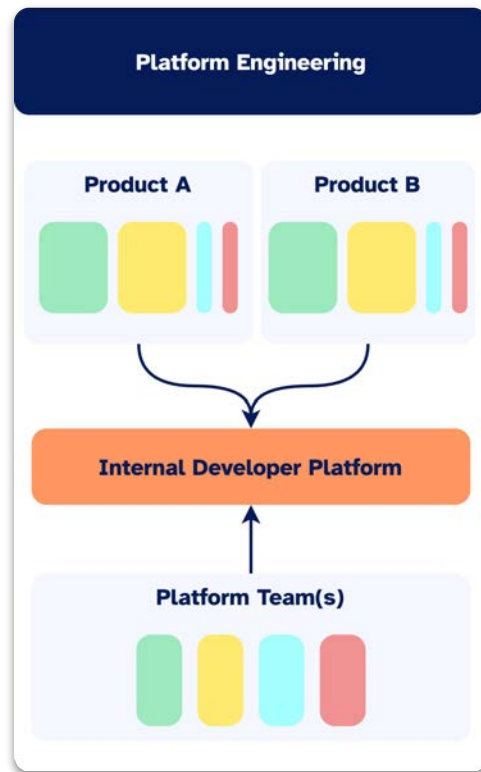
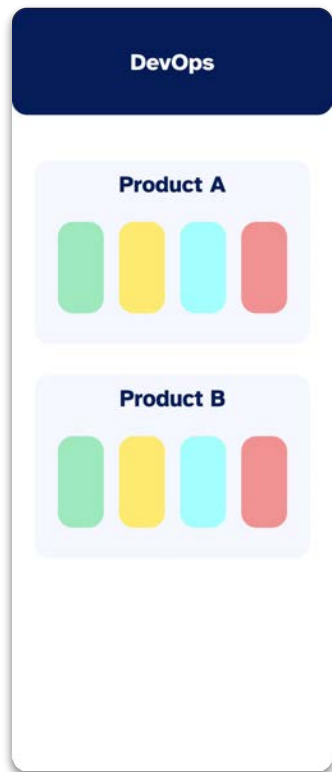
**Software Development**



**Security & Testing**



**Infrastructure & Operations**

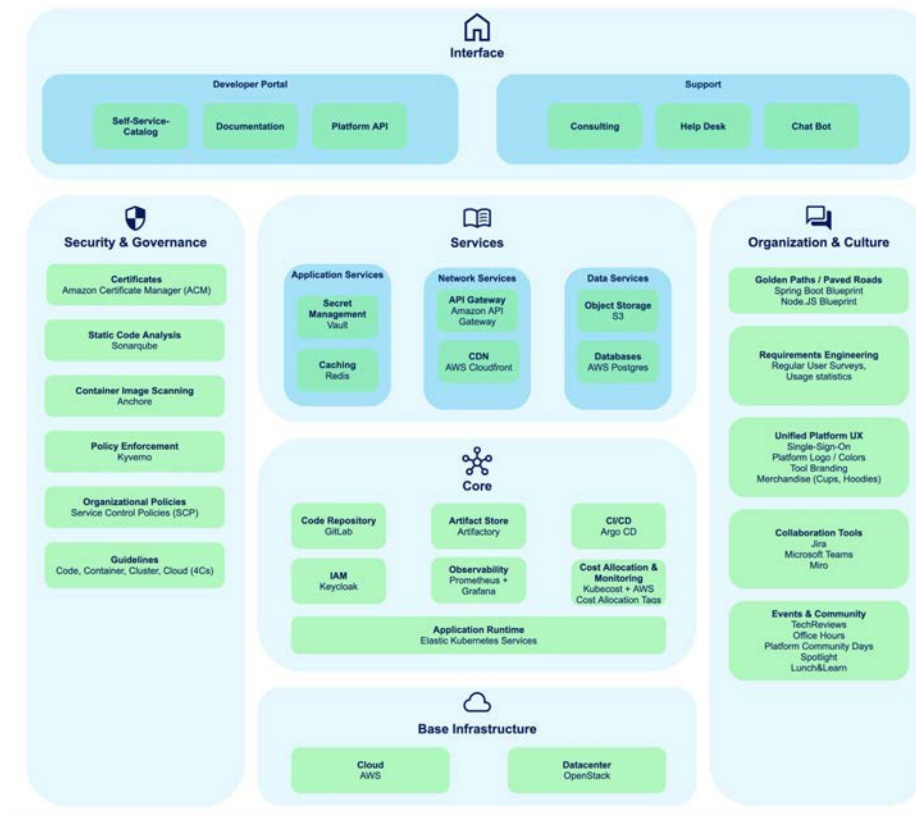


# Merkmale einer Internen Entwicklerplattform (IDP)

- **Developer Experience**  
Stellt die Zufriedenheit und Produktivität der Entwickler in den Mittelpunkt
- **Golden Paths / Paved Roads**  
Bieten standardisierte und sichere Wege, die Best Practices bündeln und die Komplexität verbergen
- **Plattformen sind Produkte**  
Die IDP wird wie ein internes Produkt mit eigenem Produktmanagement und Branding behandelt



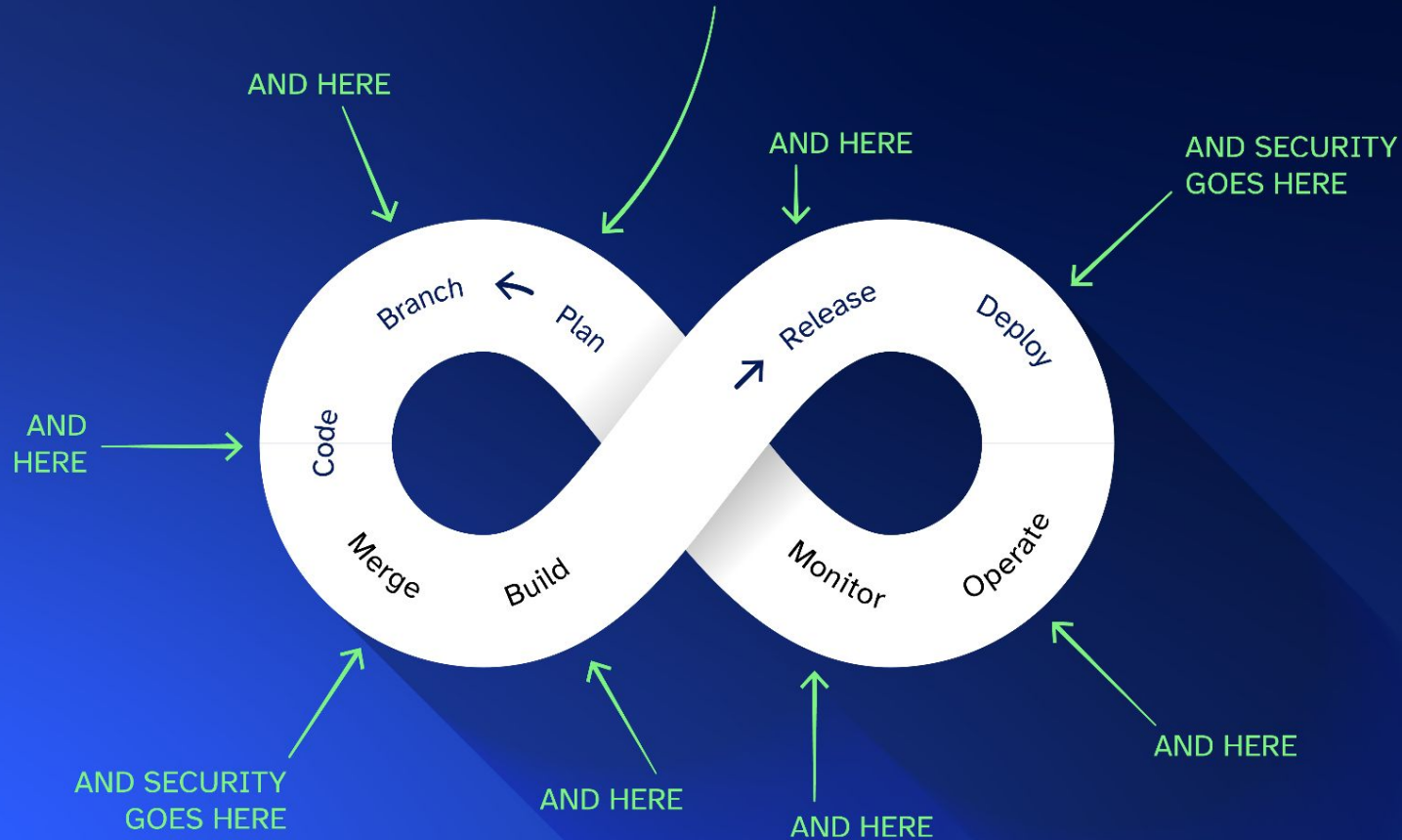


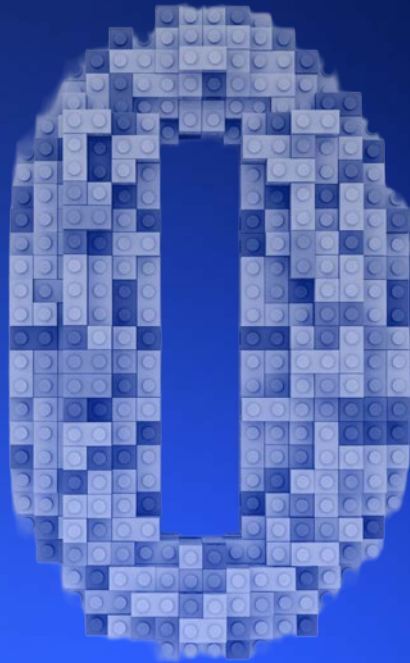


# Wie hilft Platform Engineering nun Bob mit seiner kognitiven Last?



# SECURITY GOES HERE





Initiierung

# Herausforderungen

Übliche “Checkliste des Chaos”:

- **Berechtigungen** nach und nach; “die gleichen wie die:der da drüben”
- Wo ist überhaupt dieses Gitlab? Oder andere Dienste? Welche gibt es überhaupt?
- Wie bekomme ich da ein Code Repository? Wohin pushe ich meine Docker Images?
- Welche **Regeln** muss ich befolgen? Wie kommt die Anwendung ins Asset Management und andere Compliance- und BWL-Dinge?
- Kann ich Teile von anderen Teams übernehmen (z.B. build pipelines)?



# Plattform-Ansatz

- Dokumentation
  - Welche Bausteine gibt es schon
  - Bausteine passen zusammen
- “Working with the garage door open”, Innersourcing
- Plattform-Orchestrierung, z.B. “matrose”
  - Eigenes Go-Tool
  - Input: Liste der Projekte mit Feature Toggles
  - Legt Gruppen/Repos/Namespaces in Keycloak/Gitlab/Artifactory/Grafana/Kubernetes an und setzt Berechtigungen



# Berechtigungsverwaltung

- **konsistent** für alle Plattformkomponenten
- **simpel vs least privilege**
  - Erfahrung: Team-Ebene oder “Funktion-in-Team-Ebene” ist guter Mittelweg
- Datei mit Teams/Gruppen kann für Anlage und Berechtigung von Code-Repос, Kubernetes-Namespace usw. **wiederverwendet** werden
- **Approver** / Ansprechpartner für Veränderungen an Gruppen

YAML mit Gruppen &  
Mitgliedern + terraform



Keycloak-Gruppen aus Code  
anlegen + selbst verwalten



## Konkrete Bausteine: Keycloak-Gruppen

- Gruppenanlage über matrose
- Starres Mapping: Gruppe X zu Namespace X (1:1) → mehr Gruppen
- Jeder in der Gruppe konnte andere Mitglieder hinzufügen & entfernen
- Auditierung über Keycloak-Logs

Keycloak-Gruppen aus Code  
anlegen + selbst verwalten



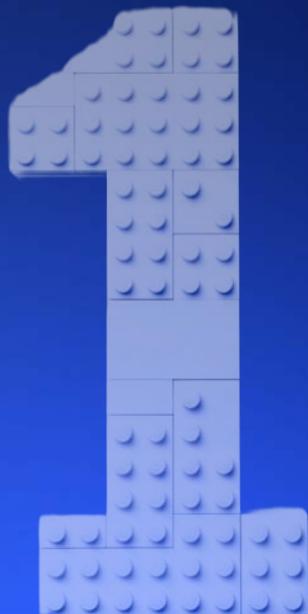
## Konkrete Bausteine: GitOps-Stil

- YAML in Git → Auditierbar
- ~1 Gruppe pro Team, dafür komplizierteres Mapping
- Approver → min. 2 für Urlaubsvertretung
- Allowlisted Groups:
  - Team kann Berechtigungen weiterdelegieren
  - Minimale Berechtigungen + Sichtbarkeit für Plattformteam

```
- name: "team-happy-feet"
  approvers:
    - "linda@secure.home"
    - "bob@secure.home"
  members:
    - "bob@secure.home"
    - "dave@secure.home"
```

YAML mit Gruppen &  
Mitgliedern + terraform





Entwicklung

# Herausforderungen

- Anforderungen und Standards identifizieren
- Generische unternehmensweite **Richtlinien** finden, die nicht zusammen passen
- Orientierung in der Service-Landschaft
  - Jedes Team löst die selben Probleme nochmal
  - Teams wollen Autonomie bei der Entwicklung
  - **Tool-Auswahl**
- Verantwortlichkeiten identifizieren
  - Welche Security-Mechanismen müssen von Bob selbst berücksichtigt werden?



# Plattform-Ansatz

- **Golden Paths:** CI-Templates, bereitgestellte Tools
  - renovate, OWASP Dependency-Track, SonarQube, trivy, ...
  - Feedback nah am Entwickler-Workflow (MR)
- Security-Guidelines
  - Mit Begründungen
  - Automatisiert, wo möglich (Policies-as-Code)
- Klare Verantwortlichkeiten (Shared Responsibilities Modell)
- **Community:** TechRadar, Code-Guidelines, Events



# Konkrete Bausteine: Compliance out-of-the-box

- Viele, lange, trockene Richtlinien
- trotzdem noch viel Freiheit

Scaffolding / Blueprint



Basis-Images



# Konkrete Bausteine: Scaffolding / Blueprints

- Ähnliche Services → passendere Checklisten, Synergien
- Abstraktion secure-by-default
  - helm-Chart, Gitlab-CI-Pipelines mit SAST/DAST, ...
  - Ejection muss möglich sein
  - Ausnahmen dokumentieren (z.B. in ADR)

Scaffolding / Blueprint



# Konkrete Bausteine: Basis-Images

- Meistens: handvoll erlaubte Betriebssysteme + wenige Basics (Zertifikate)
- Zwei Builds um Updates durch zu propagieren

Basis-Images





Betrieb

# Herausforderungen

- Kontinuierliche Weiterentwicklung, Anpassungsaufwand für Entwickler
- ITSM, Rufbereitschaft & Incident Management
- **Firewall-Regeln**, TLS
- Observability
- Disaster Recovery
- Secret-Management
- Service anbieten, Dokumentation pflegen



# Plattform-Ansatz

- Versionierung der Plattform
- Automatisiertes ITSM (Changes anlegen, Assets pflegen)
- Audit-Logs built-in
- Vordefinierte Dashboards
- Service-Abhängigkeiten aus Firewall-Regeln

# Konkrete Bausteine: Firewall-Regeln

- Erfahrung: muss schnell sein, sonst “mal lieber mehr erlauben ...”
- Layer 3/4
  - DNS-Namen?
  - Azure Service Tags o.Ä.?
- Genehmigung, zumindest durch Ziel der Regel

Zentrales YAML



Team-verwaltete  
NetworkPolicies (abstrahiert)



## Konkrete Bausteine: Firewall-Regeln zentral

- YAML in Git, Entwickler können MRs öffnen
- Von einer Umgebung zur nächsten kopierbar
- Bei anderen Teams spicken
- Einzelnes Team hat Verwaltungsaufwand, benötigt Zeit

Zentrales YAML



```
spokes:
  - id: "abcdef"
    name: "example-dev"
    approvers:
      - "linda@secure.home"
      - "bob@secure.home"
    services:
      - name: "api"
        fqdns:
          - "hostname.internal"
        port: 443
        access_allowed_from_spokes:
          - "uvwxyz"
```

## Konkrete Bausteine: Firewall-Regeln verteilt

- Initiator meist Quelle des Traffics, Genehmigung durch Ziel  
→ Kommunikation zwischen Teams
- Schwieriger zu debuggen  
→ Unterstützung durch Dashboard oder Tools
- Keine Sichtbarkeit bei anderen Teams

```
namespace: "example-dev"
```

```
services:
```

- name: "api"  
port: 443  
access\_allowed\_from:
  - "uvwxyz"

Team-verwaltete  
NetworkPolicies (abstrahiert)





# Deprovisionierung

# Herausforderungen

- Offboarding eines Entwicklers → **Secrets rotieren**
- Periodische Reviews für z.B. Berechtigungen, genutzte Ressourcen → automatisierte Sunset-Policies
- Ownership weitergeben
- Service abkündigen → Abhängigkeiten informieren
- Archivierung



## Konkrete Bausteine: Secrets rotieren

- Bestenfalls Secrets vermeiden, z.B. on-demand generieren
- Vollständige Liste der Secrets
- Wenig dupliziert
- Anleitung wie zu rotieren (automatisiert)
- Bootstrapping: wie wird der Zugriff auf Secrets geprüft?

Verschlüsselt in Repo



Integrierter Vault



## Konkrete Bausteine: in Repo

- Wenig Infrastruktur
- Rotation in Datei → über Pipeline ausrollen
- Geteilte Secrets über Repos
- **Allmächtiges Secret** in CI

```
backup_azure_sas_token: ENC[AES256_GCM,data:...]  
sops:
```

```
  age:  
    - recipient: "age1aa"  
      enc: |  
        -----BEGIN AGE ENCRYPTED FILE-----  
        ...  
        -----END AGE ENCRYPTED FILE-----  
    - recipient: "age1bb"  
    ...
```

Verschlüsselt in Repo  
(sops o.Ä.)



## Konkrete Bausteine: integrierter Vault

- **Integriert!** → Zugriff über die Identität der CI (z.B. Gitlab ID Token)
- Zentrale Stelle
- Rotiertes Secret wird automatisch an allen Stellen bemerkt und aktualisiert
  - Umsetzung der Automatisierung **aufwendig**
  - Synchronisierung der Rotation
- Benötigt Infrastruktur

Integrierter Vault



And Bob deployed happily ever after ....



# Kernbotschaften

## 1. Handelt pragmatisch, nicht dogmatisch.

Beginnt mit einer "Thinnest Viable Platform" (TVP), die das dringendste Problem löst und schafft damit **Vertrauen** und **Momentum**.

## 2. Baut eine Plattform, die Entwickler lieben.

Verbergt Komplexität hinter einfachen und sicheren "Golden Paths", um die Developer Experience zu verbessern.

**Gemeinsam.**

### 3. Die Reduktion kognitiver Last ist selbst eine Sicherheitsmaßnahme

Die beste Sicherheit ist die, über die Entwickler:innen **nicht** nachdenken müssen.

# Vielen Dank!

**Simon Dreher**  
Security & Cloud Platform Engineer

📞 +49 152 3318 1222

✉️ [simon.dreher@inovex.de](mailto:simon.dreher@inovex.de)



**Pascal Petsch**  
Cloud Platform Engineer

📞 +49 173 3181 081

✉️ [pascal.petsch@inovex.de](mailto:pascal.petsch@inovex.de)

