previder

WHITEPAPER

Azure Landing Zones mit Terraform automatisieren

Ein Beitrag von: Thomas Krampe

it starts here.

Ich glaube, das hat jeder Consultant im Microsoft Azure Umfeld bereits erlebt. Der Kunde meldet sich irgendwann, meistens nach einer unerwarteten Azure-Rechnung, einem Audit, der erste Sicherheitsvorfall, oder einfach der neue CIO, der fragt warum niemand weiß, was eigentlich alles in Azure läuft (und warum).

In meinem Fall war es ein mittelständisches Unternehmen, 350 Mitarbeiter, seit zwei Jahren mehr oder weniger aktiv auf Azure. Gestartet mit einer Subscription für "wir müssen in die Cloud-Sachen", inzwischen laufen dort einige produktive Anwendungen, eine selbst entwickelte Reporting-App, mehrere Test- und Produktivumgebungen mit virtuellen Maschinen, erste AVD (Azure Virtual Desktop) Test VDI's und irgendwo noch ein Überbleibsel aus einem alten Migrationsprojekt. Alles in einer Subscription, alles flach, alles als Global Admin angelegt.

Die Symptome die ich beim ersten Review gefunden habe:

- **Keine Trennung zwischen Produktion und Test und/oder Development auf Netzwerkebene**
- **Ressourcen ohne Tags, Kostenstellen nicht zuordenbar**
- **Jeder Entwickler mit Contributor-Rechten auf die gesamte Subscription**
- **Kein Log Analytics Workspace, Defender for Cloud deaktiviert wegen "zu teuer"**
- **Drei (oder mehr) verschiedene Namenskonventionen für die Ressourcen, keine davon dokumentiert**
- **Der Kunde hat das natürlich nicht absichtlich falsch gemacht. Wie wir sagen würden, es ist historisch gewachsen. Er hat einfach angefangen und niemand hat ihm zu Beginn gesagt, dass Azure (oder jede andere Cloud) eine Grundlage braucht, bevor der erste Workload läuft. Gut, dass er jetzt das ganze mit professioneller Beratung auf saubere Beine stellen möchte.**

Genau hier setzt eine Azure Landing Zone an und Terraform ist zumindest in diesem Artikel, das Werkzeug, mit dem wir das nicht nur einmal aufbauen, sondern so dokumentieren und vor allem automatisieren, dass der Kunde in einem Bruchteil der Zeit eine saubere Basis bekommt.

Warum dieser Artikel nicht mit "terraform init" anfängt

Die meisten Terraform-Tutorials für Azure zeigen, wie eine Resource Group und eine VM erstellt wird. Das ist nett, hilft aber meist nicht weiter, wenn eine echte Unternehmensumgebung aufgebaut oder einem Kunden eine saubere Azure-Grundlage geliefert werden soll. Ja, natürlich können wir für alles die KI benutzen, aber wenn wir nicht Wissen, was genau wir fragen sollen, gibt es einen schlechten Prompt und ein schlechter Prompt liefert ein schlechtes Ergebnis - Trash in, trash out - ihr kennt das.

Dieser Whitepaper startet dort, wo die Arbeit wirklich beginnt, bei der Azure Landing Zone. Ich beschreibe, wie wir mit Terraform eine enterprise-taugliche Azure-Infrastruktur aufbauen, die skaliert, Policy-konform ist und sich per CI/CD-Pipeline automatisiert ausrollen lässt, reproduzierbar, für jeden Mandanten.

Inhaltsverzeichnis

previder

Kapitel 1

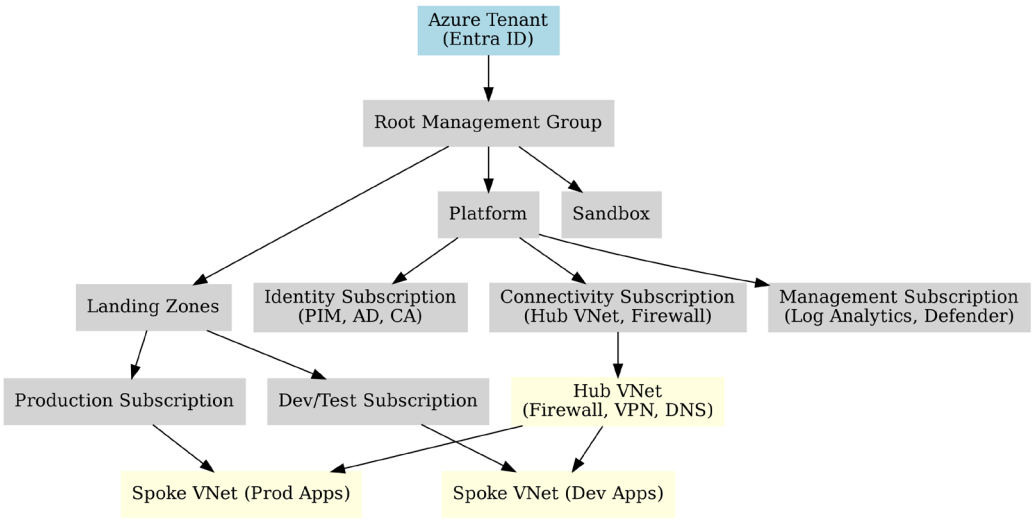
Was ist eine Azure Landing Zone?

it starts here.

Das Cloud Adoption Framework als Grundlage

Das **Cloud Adoption Framework (CAF)** ist Microsofts strukturierte Methodik für die Einführung und den Betrieb von Azure in Unternehmen. Es deckt alles ab, von der initialen Strategie über Migration und Governance bis hin zu laufendem Betrieb. Die Azure Landing Zone ist der operative Arm des CAF. Sie übersetzt die CAF-Prinzipien in eine konkrete, technisch umgesetzte Umgebung, auf der Workloads sicher und compliant laufen können.

Microsoft beschreibt eine Azure Landing Zone als eine skalierbare, modulare Umgebung auf Basis von Best Practices des Cloud Adoption Frameworks. Praktisch übersetzt: Bevor der erste Workload läuft, ist eine definierte Grundlage aus Governance, Netzwerk, Sicherheit und Identität die Voraussetzung, nicht das erste Ticket danach.



Die acht Design Areas

Microsoft strukturiert die Landing Zone in acht Design Areas, die zusammen die vollständige Plattform beschreiben. Dieser Artikel deckt davon sechs ab — Billing/Tenant-Setup und detailliertes Governance-Design sind bewusst ausgelassen, weil sie stark vom jeweiligen Kunden abhängen und kein allgemeines Beispiel erlauben.

Design Area	Inhalt	Abgedeckt
Billing & Tenant	EA/MCA-Struktur, Tenant-Topologie	—
Identity & Access	RBAC, PIM, Break Glass, Entra ID	Ja
Network	Hub-Spoke, Firewall, DNS, Private Endpoints	Ja
Resource Organisation	Management Groups, Subscriptions, Tags	Ja
Security	Defender for Cloud, Policies, Zero Trust	Ja
Management	Log Analytics, Monitoring, Alerting	Ja
Governance	Policy-Zuweisungen, Compliance, Blueprints	Ja
Platform Automation	IaC, CI/CD, Subscription Vending	Ja

Die Schichten der Landing Zone

Eine Landing Zone besteht aus mehreren Schichten:

- Management Groups — Hierarchische Organisationsstruktur über Subscriptions
- Azure Policies — Governance-Regeln, die automatisch durchgesetzt werden (Regionen, SKUs, Tagging)
- RBAC-Grundstruktur — Wer darf was, auf welcher Ebene
- Hub-Spoke-Netzwerk — Zentrales Hub-VNet mit Firewall, angeschlossene Spoke-VNets pro Workload
- Monitoring-Baseline — Log Analytics Workspace, Diagnostic Settings, Defender for Cloud

Das ist alles kein Nice-to-have. Wenn das weggelassen wird, bauen wir auf Sand. Governance-Probleme, unkontrollierte Kosten und Compliance-Lücken entstehen meist immer dort, wo die Landing Zone-Grundlage gefehlt hat.

previder

Kapitel 2

Terraform, Bicep oder Portal unsere ehrliche Einschätzung

it starts here.

Bevor es technisch wird, kurz die Frage, die jeder stellt:

	Terraform	Bicep	ARM portal
Multi-Cloud	Ja	Nein (Azure-only)	Nein
Community & Module	Sehr groß	Wächst	—
State Management	Extern (du bist verantwortlich)	Eingebaut (Deployment Stack)	—
CI/CD-Integration	Erstklassig	Gut	Nein
Lernkurve	Mittel	Niedrig (wenn Azure-only)	Keine

Unsere Empfehlung

Wenn mehrere Kunden betreut werden oder ihr nicht ausschließlich auf Azure setzt, lohnt sich Terraform. Die Investition in die Lernkurve amortisiert sich ab der zweiten Umgebung, weil die Module wieder verwendet werden.

previder

Kapitel 3

Voraussetzungen

it starts here.

Bevor wir anfangen, kurz was auf dem Rechner und im Azure-Tenant vorhanden sein muss:

Lokal installiert:

- Terraform >= 1.9 (aktuell ist die 1.15, das *avm-ptn-alz*-Modul setzt 1.9 als Minimum voraus)
- Azure CLI (eingeloggt via *az login*)

Im Azure-Tenant:

- Eine aktive Azure Subscription
- Ein Account mit Owner-Berechtigung auf Subscription-Ebene als Ausgangspunkt

Wichtig zu den Root Management Group Berechtigungen

Owner auf Subscription-Ebene reicht für das ALZ-Modul nicht. Es braucht Owner auf der Root Management Group (Tenant-Level). Diese Berechtigung ist in Azure standardmäßig für niemanden aktiv, auch nicht für Global Admins. Einmalig aktivieren: Azure Portal → Microsoft Entra ID → Properties → “Access management for Azure resources” auf “Yes” setzen. Danach kann die Rolle per CLI zugewiesen werden (Details im CI/CD-Abschnitt).

previder

Kapitel 4

Das Azure Landing Zones Terraform Modul

it starts here.

Microsoft stellt Terraform-Module im Rahmen der **Azure Verified Modules (AVM)**-Initiative bereit. AVM ist der aktuelle Microsoft-Standard für geprüfte, versionierte und offiziell unterstützte Terraform-Module. Das ältere *caf-enterprise-scale*-Modul wird im August 2026 End of Life gehen und sollte für neue Projekte nicht mehr verwendet werden.

Für Landing Zones sind zwei AVM-Module relevant:

Modul	Inhalt
Azure/avm-ptn-alz/azurerm	Management Groups, Policy-Zuweisungen, RBAC-Grundstruktur
Azure/avm-ptn-alz-managementazurerm	Log Analytics, Defender for Cloud, Monitoring-Baseline

Hinweis für Bestandsprojekte mit *caf-enterprise-scale*

Eine Migration zu *avm-ptn-alz* sollte vor August 2026 geplant werden. Das Modul wird bis dahin weiterhin gepflegt, aber kein neues Projekt sollte damit starten. Der ALZ Terraform Accelerator generiert bereits neuen Code auf Basis von *avm-ptn-alz*.

Die vollständige Modulreferenz ist hier zu finden: Azure Verified Modules — Terraform Pattern Modules.

Wer den manuellen Weg überspringen will

Am Ende des Whitepapers gibt es einen Abschnitt zum **ALZ Terraform Accelerator**, einem Microsoft-Werkzeug das den kompletten Code inkl. Pipeline und Backend-Setup automatisch generiert. Für Greenfield-Projekte ist das der schnellste Einstieg. Der manuelle Weg lohnt sich trotzdem, weil der generierte Code früher oder später angepasst werden muss.

Minimale Verzeichnisstruktur

Minimale Verzeichnisstruktur </> Shell

```
alz/
├── lib/
│   └── alz.alz_architecture_definition.json  # Management Group Hierarchie
├── main.tf           # Modul-Aufruf und Provider
├── variables.tf      # Parameter
├── terraform.tfvars  # Kundenwerte – nicht-sensitive Konfiguration (nicht ins Git!)
├── backend.tf        # Remote State
└── outputs.tf        # IDs und Referenzen
```

Wichtiger Hinweis

Landet das Terraform Projekt in einem Github Repository, was ich ausdrücklich zusammen mit Github Actions empfehle, dann kümmert euch zwingend um die *.gitignore* Datei. Die *terraform.tfvars* enthält Konfigurationswerte wie Region, Kundenpräfix und Tags — aber keine Secrets. Echte Secrets (Passwörter, API-Keys, Zertifikate) gehören nie in eine Datei, weder committed noch lokal. Sie werden über Azure Key Vault mit einem Terraform Data Source oder über Umgebungsvariablen eingelesen.

Backend zuerst, Terraform State in das Azure Storage

Bevor wir auch nur eine Ressource erstellen, richten wir den Terraform Remote State ein. Ein lokal gespeicherter State ist in einem Team eine echte Zeitbombe.

Azure CLI: Terraform Remote State Storage </> Shell

```shell
# Storage Account für den State anlegen (einmalig, manuell oder per Skript)
az group create --name rg-terraform-state --location germanywestcentral

# WICHTIG: Storage Account Namen müssen global eindeutig sein (über alle Azure-Tenants weltweit).
# Den Namen unbedingt anpassen, z.B. mit einem Kundenpräfix: stcontosotfstate
az storage account create \
  --name stterraformstate \
  --resource-group rg-terraform-state \
  --sku Standard_LRS \
  --allow-blob-public-access false \
  --min-tls-version TLS1_2

az storage container create \
  --name tfstate \
  --account-name stterraformstate

# Härtung: Blob Soft Delete (Schutz vor versehentlichem Löschen, 30 Tage Recovery)
az storage account blob-service-properties update \
  --account-name stterraformstate \
  --resource-group rg-terraform-state \
  --enable-delete-retention true \
  --delete-retention-days 30

# Härtung: Blob Versioning (Point-in-Time Recovery bei State-Korruption)
az storage account blob-service-properties update \
  --account-name stterraformstate \
  --resource-group rg-terraform-state \
  --enable-versioning true

# Härtung: Netzwerkzugriff auf bekannte IPs einschränken (z.B. Pipeline-Agent oder eigene IP)
# In Produktivumgebungen empfiehlt sich ein Private Endpoint statt IP-Whitelist
az storage account network-rule add \
  --account-name stterraformstate \
  --resource-group rg-terraform-state \
  --ip-address "EIGENE_IP_ODER_AGENT_IP"

az storage account update \
  --name stterraformstate \
  --resource-group rg-terraform-state \
  --default-action Deny
```

backend.tf </> Hcl

```hcl
# backend.tf
terraform {
  backend "azurerm" {
    resource_group_name  = "rg-terraform-state"
    storage_account_name = "stterraformstate"
    container_name       = "tfstate"
    key                  = "alz/terraform.tfstate"
  }
}
```

Der State wird mit einem Blob-Lease gesperrt, sobald jemand *terraform apply* ausführt. Gleichzeitige Änderungen sind damit ausgeschlossen.

Das Modul aufrufen

Das *avm-ptn-alz*-Modul arbeitet mit einem eigenen *alz*-Provider, der Policy-Definitionen und Archetype-Konfigurationen aus der offiziellen Azure Landing Zones Library lädt. Die Management Group Hierarchie wird in einer lokalen JSON-Datei im *lib/*-Verzeichnis definiert, was sie versionierbar und kundenseitig anpassbar macht.

main.tf: ALZ Modul aufrufen </> Hcl

```hcl
# main.tf
terraform {
  required_version = ">= 1.9"

  required_providers {
    azapi = {
      source  = "Azure/azapi"
      version = "~> 2.0, >= 2.0.1"
    }
    alz = {
      source  = "Azure/alz"
      version = "~> 0.17"
    }
  }
}

provider "azapi" {}

# Der alz-Provider lädt Policy-Definitionen und Archetype-Konfigurationen
# aus der Azure Landing Zones Library (GitHub). ref = Versions-Tag der Library.
provider "alz" {
  library_overwrite_enabled = true
  library_references = [
    {
      path = "platform/alz"
      ref  = "2026.04.2"  # Beim Einsetzen auf aktuellen Release-Tag prüfen
    },
    {
      # Lokales lib/-Verzeichnis für kundenseitige Anpassungen
      custom_url = "${path.module}/lib"
    }
  ]
}

# Tenant-ID für die Root Management Group auslesen
data "azapi_client_config" "current" {}

module "alz" {
  source  = "Azure/avm-ptn-alz/azurerm"
  version = "~> 0.21"

  # Pflichtparameter
  architecture_name  = "alz"                              # Name der Architektur-Definition in lib/
  parent_resource_id = data.azapi_client_config.current.tenant_id
  location           = var.location

  enable_telemetry = false
}
```

Die Management Group Struktur wird in *lib/alz.alz_architecture_definition.json* definiert. Das folgende Beispiel bildet die Standard-ALZ-Hierarchie nach CAF ab und kann mit dem Kundenpräfix angepasst werden:

alz.alz_architecture_definition.json </> JSON

```
{
  "$schema": "https://raw.githubusercontent.com/Azure/Azure-Landing-Zones-Library/main/schemas/architecture_definition.json",
  "name": "alz",
  "management_groups": [
    { "archetypes": ["root"],          "display_name": "Contoso ALZ",    "id": "contoso-alz",          "parent_id": null },
    { "archetypes": ["platform"],      "display_name": "Platform",       "id": "contoso-platform",     "parent_id": "contoso-alz" },
    { "archetypes": ["management"],    "display_name": "Management",     "id": "contoso-management",   "parent_id": "contoso-platform" },
    { "archetypes": ["connectivity"], "display_name": "Connectivity",   "id": "contoso-connectivity", "parent_id": "contoso-platform" },
    { "archetypes": ["identity"],      "display_name": "Identity",       "id": "contoso-identity",     "parent_id": "contoso-platform" },
    { "archetypes": ["landing_zones"],"display_name": "Landing Zones",  "id": "contoso-landingzones", "parent_id": "contoso-alz" },
    { "archetypes": ["corp"],          "display_name": "Corp",           "id": "contoso-corp",         "parent_id": "contoso-landingzones" },
    { "archetypes": ["online"],        "display_name": "Online",         "id": "contoso-online",       "parent_id": "contoso-landingzones" },
    { "archetypes": ["sandbox"],       "display_name": "Sandbox",        "id": "contoso-sandbox",      "parent_id": "contoso-alz" }
  ]
}
```

Ein *terraform apply* erstellt die komplette Management-Group-Hierarchie, weist die CAF-Standard-Policies zu und richtet das Hub-Netzwerk ein. Rechne mit 10 bis 20 Minuten Laufzeit und je nach Konfiguration mit mehreren Dutzend bis über hundert Ressourcen, die Terraform anlegt. Der erste Apply ist dabei der langsamste, weil Azure die Policy-Definitionen verteilen und die Management Group Hierarchie aufbauen muss. Folge-Applies laufen deutlich schneller durch, weil Terraform nur noch die Differenz zum bestehenden State ausführt.

Das Ergebnis im Azure Portal sieht so aus, die gesamte Hierarchie auf einen Blick:

Contoso Automotive GmbH	...	Verwaltungsgruppe	az-demo
Außer Betrieb	...	Verwaltungsgruppe	az-demo-decommissioned
Landing Zones	...	Verwaltungsgruppe	az-demo-landingzones
Nicht-Produktion	...	Verwaltungsgruppe	az-demo-lz-nichtproduktion
Entwicklung	...	Verwaltungsgruppe	az-demo-np-entwicklung
Test & Qualitätssicherung	...	Verwaltungsgruppe	az-demo-np-test
Produktion	...	Verwaltungsgruppe	az-demo-lz-produktion
Corp (ERP & Fertigung)	...	Verwaltungsgruppe	az-demo-prod-corp
IoT & Fertigungsdaten	...	Verwaltungsgruppe	az-demo-prod-iot
Online (Kundenportal & B2B)	...	Verwaltungsgruppe	az-demo-prod-online
Platform	...	Verwaltungsgruppe	az-demo-platform
Connectivity	...	Verwaltungsgruppe	az-demo-platform-connectivity
Identity	...	Verwaltungsgruppe	az-demo-platform-identity
Management	...	Verwaltungsgruppe	az-demo-platform-management
Sandbox	...	Verwaltungsgruppe	az-demo-sandbox

Landing Zones mit Produktions- und Nicht-Produktions-Bereichen:

Name	Typ	ID
Landing Zones	Verwaltungsgruppe	az-demo-landingzones
Nicht-Produktion	Verwaltungsgruppe	az-demo-lz-nichtproduktion
Entwicklung	Verwaltungsgruppe	az-demo-np-entwicklung
Test & Qualitätssicherung	Verwaltungsgruppe	az-demo-np-test
Produktion	Verwaltungsgruppe	az-demo-lz-produktion
Corp (ERP & Fertigung)	Verwaltungsgruppe	az-demo-prod-corp
IoT & Fertigungsdaten	Verwaltungsgruppe	az-demo-prod-iot
Online (Kundenportal & B2B)	Verwaltungsgruppe	az-demo-prod-online

Platform-Bereich mit Management, Connectivity und Identity:

Platform	Verwaltungsgruppe	az-demo-platform
Connectivity	Verwaltungsgruppe	az-demo-platform-connectivity
Identity	Verwaltungsgruppe	az-demo-platform-identity
Management	Verwaltungsgruppe	az-demo-platform-management

previder

Kapitel 5

CI/CD – Authentifizierung ohne Secrets

it starts here.

Bevor wir in die Pipeline-Syntax gehen, das Wichtigste: Wie authentifiziert sich die Pipeline gegenüber Azure?

Der klassische Weg ist ein Service Principal mit *client_secret* in einer Pipeline-Variable zu speichern. Funktioniert, ist aber eine permanente Schwachstelle. Secrets laufen ab, werden versehentlich geloggt oder landen im falschen Repository.

Der bessere Weg ist Workload Identity Federation (OIDC), die Pipeline beweist ihre Identität über ein kurzlebiges, signiertes Token. Azure vertraut diesem Token dann auf Basis einer vorab konfigurierten Vertrauensbeziehung. Kein dauerhaftes Secret, das rotiert werden muss.

Die Einrichtung ist einmalig identisch, egal ob GitHub Actions oder Azure DevOps:

Azure CLI: App Registration en Service Principal </> Shell

```
# App Registration und Service Principal anlegen
az ad app create --display-name "sp-alz-terraform"
APP_ID=$(az ad app list --display-name "sp-alz-terraform" --query "[0].appId" -o tsv)
az ad sp create --id $APP_ID
SP_OBJECT_ID=$(az ad sp show --id $APP_ID --query "id" -o tsv)

# Tenant ID auslesen — wird im scope-Pfad benötigt
TENANT_ID=$(az account show --query tenantId -o tsv)

# Owner auf Root Management Group (für den initialen ALZ-Deploy zwingend erforderlich)
# Die Root Management Group hat immer die gleiche ID wie der Tenant
az role assignment create \
  --role "Owner" \
  --assignee-object-id $SP_OBJECT_ID \
  --scope "/providers/Microsoft.Management/managementGroups/${TENANT_ID}"

# SICHERHEITSHINWEIS: Owner auf Root-MG-Ebene ist das absolute Minimum für den ersten Deploy.
# Nach dem initialen Aufbau der Landing Zone sollte die Rolle auf die tatsächlich benötigten
# Permissions reduziert werden. Microsoft stellt eine Custom Role Definition bereit, die genau
# die für ALZ-Betrieb erforderlichen Berechtigungen enthält:
# https://github.com/Azure/Azure-Landing-Zones-Library/tree/main/platform/alz/role_definitions
```

Hinweis

Die folgenden Befehle verwenden *$APP_ID*, *$SP_OBJECT_ID* und *$TENANT_ID* aus dem Block oben. Wer die Blöcke in einer neuen Shell-Session ausführt, muss diese Variablen zuerst neu setzen: *APP_ID=$(az ad app list --display-name “sp-alz-terraform” --query “[0].appId” -o tsv)*

previder

Kapitel 6

GitHub Actions

it starts here.

Federated Credential für GitHub

Für GitHub Actions werden zwei Federated Credentials benötigt: eines für Pushes auf *main* (der Apply-Trigger) und eines für Pull Requests (der Plan-Trigger). Fehlt das PR-Credential, schlägt der OIDC-Token-Austausch in PR-Runs still fehl.

GitHub Actions: Federated Credential </> Shell

```shell
# Credential für Pushes auf main – löst terraform apply aus
az ad app federated-credential create \
  --id $APP_ID \
  --parameters '{
    "name": "github-alz-main",
    "issuer": "https://token.actions.githubusercontent.com",
    "subject": "repo:contoso/azure-landing-zone:ref:refs/heads/main",
    "audiences": ["api://AzureADTokenExchange"]
  }'

# Credential für Pull Requests – löst terraform plan aus
az ad app federated-credential create \
  --id $APP_ID \
  --parameters '{
    "name": "github-alz-pr",
    "issuer": "https://token.actions.githubusercontent.com",
    "subject": "repo:contoso/azure-landing-zone:pull_request",
    "audiences": ["api://AzureADTokenExchange"]
  }'
```

Im GitHub Repository werden dann nur drei Secrets hinterlegt: *AZURE_CLIENT_ID*, *AZURE_TENANT_ID*, *AZURE_SUBSCRIPTION_ID* — keine Passwörter oder Token, nur IDs.

Pipeline

.github/workflows/terraform-alz.yml </> YAML

```yaml
# .github/workflows/terraform-alz.yml
name: Terraform ALZ

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

permissions:
  id-token: write        # OIDC-Token anfordern
  contents: read
  pull-requests: write   # Plan als PR-Kommentar

env:
  ARM_CLIENT_ID:       $
  ARM_TENANT_ID:       $
  ARM_SUBSCRIPTION_ID: $
  ARM_USE_OIDC:        "true"

jobs:
  terraform:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: alz/

    steps:
      - uses: actions/checkout@v4

      - uses: hashicorp/setup-terraform@v3
        with:
          terraform_version: "~1.15"

      - name: Terraform Init
        run: terraform init

      - name: Terraform Validate
        run: terraform fmt -check && terraform validate

      - name: Security Scan mit Checkov
        uses: bridgecrewio/checkov-action@v12
        with:
          directory: alz/
          framework: terraform
          soft_fail: true   # Pipeline nicht blocken, aber Findings im Log sichtbar machen

      - name: Terraform Plan
        id: plan
        run: terraform plan -out=tfplan -no-color
        continue-on-error: true

      - name: Plan als Pull Request Kommentar
        uses: actions/github-script@v7
        if: github.event_name == 'pull_request'
        with:
          script: |
            github.rest.issues.createComment({
              issue_number: context.issue.number,
              owner: context.repo.owner,
              repo: context.repo.repo,
              body: "```\n$\n```"
            })

      - name: Terraform Apply
        if: github.ref == 'refs/heads/main' && github.event_name == 'push'
        run: terraform apply -auto-approve tfplan
```

Jeder Pull Request zeigt den Plan als Kommentar im Review. Merge auf *main* löst den Apply aus.

Checkov ist ein statischer Code-Analyse-Tool für Infrastructure-as-Code. Es prüft Terraform-Code auf bekannte Fehlkonfigurationen, zum Beispiel Storage Accounts ohne Netzwerkrestriktionen, Key Vaults ohne Soft Delete oder VMs mit öffentlicher IP. Mit *soft_fail: true* blockiert es die Pipeline nicht, macht Findings aber im CI-Log sichtbar. Wer strikteres Verhalten will, setzt *soft_fail: false* — dann bricht die Pipeline bei kritischen Findings ab, bevor ein Plan ausgeführt wird.

previder

Kapitel 7

Azure DevOps Pipelines

it starts here.

Azure DevOps nutzt statt GitHub Secrets eine **Service Connection** mit Workload Identity Federation, die wir direkt im Project Settings anlegen können (Project Settings → Service connections → New → Azure Resource Manager → Workload Identity Federation).

Wichtig

Die Service Connection muss auf Management-Group-Ebene berechtigt werden, nicht nur auf Subscription-Ebene. Das passiert nachträglich per Azure CLI (RBAC-Assignment wie oben).

Federated Credential für Azure DevOps

Azure DevOps: Federated Credential </> Shell

```
# Subject-Format: sc://<Organisation>/<Projekt>/<ServiceConnection-Name>
# Die ORGANISATION_ID ist die GUID der Azure DevOps Organisation –
# zu finden unter: https://dev.azure.com/<Organisation>/_settings/organizationAad
az ad app federated-credential create \
  --id $APP_ID \
  --parameters '{
    "name": "azdo-alz-terraform",
    "issuer": "https://vstoken.dev.azure.com/ORGANISATION_ID",
    "subject": "sc://contoso-org/platform-engineering/alz-terraform",
    "audiences": ["api://AzureADTokenExchange"]
  }'
```

Pipeline

azure-pipelines.yml

```yaml
# azure-pipelines.yml
trigger:
  branches:
    include: [main]

pr:
  branches:
    include: [main]

variables:
  terraformVersion: "1.15.5"
  workingDirectory: "alz"

pool:
  vmImage: ubuntu-latest

stages:
  - stage: Plan
    jobs:
      - job: TerraformPlan
        steps:
          - task: TerraformInstaller@1
            inputs:
              terraformVersion: $(terraformVersion)

          - task: TerraformTaskV4@4
            displayName: Terraform Init
            inputs:
              provider: azurerm
              command: init
              workingDirectory: $(workingDirectory)
              backendServiceArm: "alz-terraform" # Name der Service Connection
              backendAzureRmResourceGroupName: "rg-terraform-state"
              backendAzureRmStorageAccountName: "stterraformstate"
              backendAzureRmContainerName: "tfstate"
              backendAzureRmKey: "alz/terraform.tfstate"

          - task: TerraformTaskV4@4
            displayName: Terraform Plan
            inputs:
              provider: azurerm
              command: plan
              workingDirectory: $(workingDirectory)
              environmentServiceNameAzureRM: "alz-terraform"
              commandOptions: "-out=tfplan -no-color"

          - publish: $(workingDirectory)/tfplan
            artifact: tfplan

  - stage: Apply
    condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/main'))
    jobs:
      - deployment: TerraformApply
        environment: "production"    # Environment mit manuellem Approval-Gate
        strategy:
          runOnce:
            deploy:
              steps:
                - download: current
                  artifact: tfplan

                - task: TerraformTaskV4@4
                  displayName: Terraform Apply
                  inputs:
                    provider: azurerm
                    command: apply
                    workingDirectory: $(workingDirectory)
                    environmentServiceNameAzureRM: "alz-terraform"
                    commandOptions: "$(Pipeline.Workspace)/tfplan/tfplan"
```

Der wesentliche Unterschied zu GitHub Actions, Azure DevOps bietet mit Environments ein natives Approval-Gate. Der Apply-Stage wartet auf eine manuelle Freigabe, bevor er ausgeführt wird. Das ist in regulierten Umgebungen oder bei Kunden mit Change-Management-Prozessen ein echter Vorteil.

previder

Kapitel 8

Wann welche Plattform?

it starts here.

	GitHub Actions	Azure DevOps
Approval-Gate vor Apply	Über Environments (etwas umständlicher)	Nativ, direkt im Environment
Bestehende Infrastruktur	Gut wenn Code bereits auf GitHub	Gut wenn Kunde bereits ADO nutzt
Audit-Trail	GitHub Logs	ADO bietet detailliertere Compliance-Logs
Kosten	Kostenlos für Public Repos, Minuten für Private	Im M365/Azure-Paket oft inklusive

Für neue Projekte ohne Vorgaben: verwende ich GitHub Actions, weil sie deutlich schlanker und einfacher zu konfigurieren sind. Bei Projekten mit bestehendem Azure DevOps und Change-Management-Anforderungen verwende ich dann lieber Azure DevOps mit Environment-Approval.

previder

Kapitel 9

Typische Stolpersteine

it starts here.

1. Secrets im State File

Terraform speichert alle Ressourcen-Attribute im State, auch Passwörter und Tokens die vielleicht niemand explizit ausgegeben hat. Den State File niemals ins das Repository legen, immer ein Remote Backend, Storage Account mit Private Endpoint oder zumindest IP-Einschränkung anlegen.

2. Management Group Deletion Lock

Wenn wir eine Management Group löschen wollen, die noch Subscriptions enthält, schlägt Terraform mit einem kryptischen Fehler fehl. Subscriptions müssen zuerst herausgelöst werden, wichtig für den *terraform destroy* Workflow.

3. Policy-Compliance-Latenzen

Nach dem Zuweisen einer Policy kann es bis zu 30 Minuten dauern, bis Azure alle Ressourcen evaluiert hat. Terraform meldet zwar *apply complete*, aber Compliance-Checks im Portal zeigen noch “Not compliant”. Das ist kein Bug, das ist Azure. In CI/CD-Pipelines *terraform apply* daher nicht mit Compliance-Checks ohne eine angemessene Wartezeit verketten.

4. Kostenfalle Azure Firewall

Das ALZ-Modul kann eine Azure Firewall im Hub bereitstellen. Die Kosten setzen sich aus mehreren Komponenten zusammen:

- **Deployment-Kosten** (reine Betriebszeit, unabhängig vom Traffic): Basic ~300 EUR/Monat, Standard ~900 EUR/Monat, Premium ~1.500 EUR/Monat
- **Firewall Policy:** kostenlos wenn direkt mit der Firewall verwaltet, ~30 EUR/Monat pro Policy wenn Azure Firewall Manager im Einsatz ist
- **Datenverarbeitung:** ca. 0,016 EUR/GB verarbeiteter Traffic

In Demo- oder Lab-Umgebungen die Firewall deaktivieren und durch Network Security Groups ersetzen. NSGs sind kein gleichwertiger Ersatz in Produktivumgebungen (kein Layer-7-Inspection, kein zentrales Logging), reichen aber zum Testen der Governance-Struktur ohne laufende Kosten.

5. Root Management Group Berechtigungen

Das ALZ-Modul benötigt Owner-Berechtigungen auf der Root Management Group (Tenant-Level). Diese Berechtigung ist in frischen Azure-Tenants nicht automatisch vorhanden, auch nicht für Global Admins. Die Aktivierung ist im Voraussetzungen-Abschnitt am Anfang des Artikels beschrieben. Wer das übersehen hat: Azure Portal → Microsoft Entra ID → Properties → “Access management for Azure resources” auf “Yes” setzen, danach die Rolle per CLI zuweisen.

previder

Kapitel 10

Was wir jetzt haben

it starts here.

Nach einem erfolgreichen *terraform apply* haben wir:

- Eine vollständige Management-Group-Hierarchie nach CAF-Standard
- Über 100 Azure Policies aktiv und zugewiesen (Defender, Logging, Tagging, erlaubte Regionen)
- Ein Hub-VNet bereit für Connectivity-Workloads
- Einen Log Analytics Workspace mit Defender for Cloud als Grundlage
- Eine CI/CD-Pipeline die jeden weiteren Change automatisch plant und anwendet
- Das ist die Basis, auf der alle Kundenworkloads sauber laufen. Neue Subscriptions werden in die richtige Management Group eingehängt, erben automatisch alle Policies und sind vom ersten Tag an compliant.

Die Landing Zone adressiert vier der fünf Säulen des **Azure Well-Architected Framework (WAF)** strukturell. Die Performance Efficiency liegt bei den einzelnen Workloads und ist kein Bestandteil der Plattformebene.

WAF-Säule	Wie die Landing Zone sie adressiert
Reliability	Policies erzwingen Redundanzstandards, Regionen und SKUs werden kontrolliert
Security	RBAC, PIM, Private Endpoints, Defender for Cloud, Zero Trust by Default
Cost Optimization	Tags und Policies für Kostenzuordnung, Budget-Alerts als Baseline
Operational Excellence	CI/CD-Pipeline für jeden Change, IaC als Single Source of Truth, Monitoring-BaselinePrivate
Performance Efficiency	Liegt auf Workload-Ebene, nicht auf Plattformebene

previder

Kapitel 11

Subscription Vending — wie neue Workloads an ihre Subscription kommen

it starts here.

Die Landing Zone definiert die Struktur, aber sie beantwortet noch nicht, wie ein neues Entwicklungsteam an seine eigene Subscription kommt und das in der richtigen Management Group, mit den richtigen Policies und Tags, ohne dass jemand manuell im Portal klickt?

Subscription Vending ist das Automatisierungsmuster dafür. Es beschreibt einen vollständig automatisierten Prozess, der auf Anfrage eine neue Subscription provisioniert, sie in die korrekte Management Group einordnet, Basisressourcen wie ein Spoke-VNet anlegt und mit der Hub-Netzwerk-Infrastruktur verbindet. Das Team bekommt eine fertige Umgebung, die vom ersten Tag an den Unternehmensstandards entspricht und zwar nicht, weil jemand es manuell konfiguriert hat, sondern weil die Landing Zone die Policies automatisch durchsetzt.

Technisch wird Subscription Vending üblicherweise über eine Pipeline realisiert, die per Pull Request oder API-Aufruf ausgelöst wird und Terraform (oder Bicep) mit den Parametern des neuen Workloads ausführt. Das AVM-Modul Azure/avm-ptn-alz-management/azurerm und der ALZ Terraform Accelerator enthalten Beispiele für dieses Muster.

Subscription Vending ist kein Pflichtbestandteil der initialen Landing Zone, aber in jeder Umgebung sinnvoll, die mehr als drei oder vier Workload-Teams bedient. Ohne es entsteht früher oder später ein manueller Bottleneck beim Platform-Team.

previder

Kapitel 12

Identity, wer darf was und wann

it starts here.

Die Landing Zone legt die RBAC-Struktur fest, aber RBAC allein reicht nicht. Zwei Themen sind in der Praxis immer wieder der Unterschied zwischen einer sicheren und einer nur nach außen hin ordentlichen Umgebung.

PIM, keine dauerhaften privilegierten Rollen

Privileged Identity Management bedeutet, dass niemand permanent als Owner oder Contributor in einer Subscription sitzt. Stattdessen werden privilegierte Rollen auf Anfrage für einen definierten Zeitraum aktiviert, mit Begründung, mit Genehmigung wenn nötig, und mit vollständigem Audit-Trail. Nach Ablauf der Zeit ist die Rolle weg, ohne dass jemand daran denkt.

In der Praxis hat der Consultant oder der Platform-Engineer dauerhaft Reader-Rechte und aktiviert Owner für die Dauer der Arbeit. Das klingt nach Overhead, ist aber der einzige Weg um permanente privilegierte Konten als Angriffsfläche auszuschließen. PIM ist in Microsoft Entra ID P2 enthalten und sollte in jeder Unternehmensumgebung aktiv sein.

Break Glass Accounts

Zwei Konten, die ausschließlich für Notfälle existieren: keine Conditional Access Policies die sie aussperren könnten, keine MFA per Authenticator-App, weil die an ein Gerät gebunden ist, das im Notfall nicht verfügbar sein könnte. Break Glass Accounts sollten stattdessen mit FIDO2 Hardware-Keys (z.B. YubiKey) oder einer langen Passphrase gesichert werden, die physisch hinterlegt ist, in einem verschlüsselten Tresor oder einem physisch gesicherten Umschlag. Verwendet werden sie ausschließlich dann, wenn alle anderen Zugangswege versagen: globale MFA-Störung, Entra ID Incident, kompromittierter Admin-Account.

Break Glass Accounts gehören in jede Landing Zone, werden aber regelmäßig vergessen. Die Zugangsdaten müssen auch tatsächlich irgendwo liegen, wo sie im Notfall erreichbar sind, ein verschlüsselter Tresor, ein Umschlag im Schrank, abhängig vom Sicherheitskonzept des Kunden. Was nicht funktioniert, sind die Credentials nur im Passwortmanager des Admins vorzuhalten, der vielleicht gerade nicht erreichbar ist.

Service Principals vs. Workload Identity

Der klassische Service Principal mit *client_secret* ist ein dauerhafter Credential, der rotiert werden muss, der versehentlich geloggt wird, der in .env-Dateien landet und den niemand mehr kennt nachdem der Kollege das Unternehmen verlassen hat.

Workload Identity Federation, wie im CI/CD-Abschnitt beschrieben, löst das Problem an der Wurzel. Die Pipeline beweist ihre Identität über ein kurzlebiges, signiertes Token. Azure vertraut diesem Token auf Basis einer einmalig konfigurierten Vertrauensbeziehung. Kein Secret, kein Ablaufdatum, kein manuelles Rotieren. Für alle neuen Deployments sollte das der Standard sein, Service Principals mit Secrets gehören ausgemustert.

previder

Kapitel 13

Netzwerk, was die Landing Zone schützt und warum

it starts here.

Das Hub-Spoke-Netzwerk ist der strukturelle Rahmen. Drei Konzepte bestimmen dabei ob eine Azure-Umgebung tatsächlich sicher ist oder ob sie nur so aussieht.

Private Endpoints

Dienste wie Storage Accounts, Key Vault, SQL Datenbanken und viele weitere Azure-Ressourcen sind standardmäßig über das öffentliche Internet erreichbar. Das ist der Default. Private Endpoints geben diesen Diensten eine private IP-Adresse innerhalb des VNets, der öffentliche Endpunkt kann danach deaktiviert werden.

In der Praxis bedeutet das, ein Storage Account mit einem Private Endpoint ist nur aus dem Netz erreichbar, in dem der Endpoint hängt, und aus verbundenen Netzen. Kein direkter Internetzugriff, kein versehentliches Offenlegen von Daten durch eine falsch konfigurierte Firewall-Regel. Die Landing Zone kann per Azure Policy erzwingen, dass bestimmte Ressourcentypen ausschließlich mit Private Endpoints deployed werden dürfen.

DNS

Private Endpoints funktionieren nur korrekt, wenn DNS stimmt. Azure löst *storageaccount.blob.core.windows.net* standardmäßig zur öffentlichen IP auf. Mit einem Private Endpoint muss DNS dieselbe Adresse zur privaten IP im VNet auflösen, damit der Traffic den richtigen Weg nimmt.

Dafür gibt es Private DNS Zones, eine pro Dienst-Typ (*privatelink.blob.core.windows.net*, *privatelink.vaultcore.azure.net*, usw.), die im Hub-VNet hängen und automatisch die korrekten A-Records eingetragen bekommen, sobald ein Private Endpoint erstellt wird. Das Hub-VNet fungiert dabei als zentraler DNS-Resolver für alle angeschlossenen Spoke-VNets. Wenn dieser DNS-Mechanismus fehlt oder falsch konfiguriert ist, landen Verbindungen trotz Private Endpoint auf der öffentlichen IP, ohne dass man es sofort bemerkt.

Warum offener Internetzugriff ein Risiko ist

In einer flachen Azure-Umgebung ohne Landing Zone sind VMs, Storage Accounts und andere Ressourcen oft direkt aus dem Internet erreichbar, entweder weil Public IP vergeben wurde, weil NSG-Regeln zu weit offen sind oder weil niemand aktiv verhindert hat, dass ein Entwickler schnell etwas testet und es dann produktiv lässt.

Die Landing Zone adressiert das auf mehreren Ebenen:

- Azure Policies verhindern, dass bestimmte Ressourcentypen ohne Private Endpoint deployed werden. NSG-Grundregeln blockieren eingehenden Internetzugriff als Default.
- Ausgehender Internetzugriff läuft durch die Azure Firewall im Hub, die den Traffic inspizieren und loggen kann.
- Kein Spoke-VNet kommuniziert direkt mit dem Internet, sondern immer über das Hub.

Das klingt restriktiv, ist aber genau der Punkt. In einer Unternehmensumgebung sollte jede Verbindung die nicht explizit erlaubt ist, verboten sein. Die Landing Zone verwendet dieses Prinzip in der Grundkonfiguration konsequent.

previder

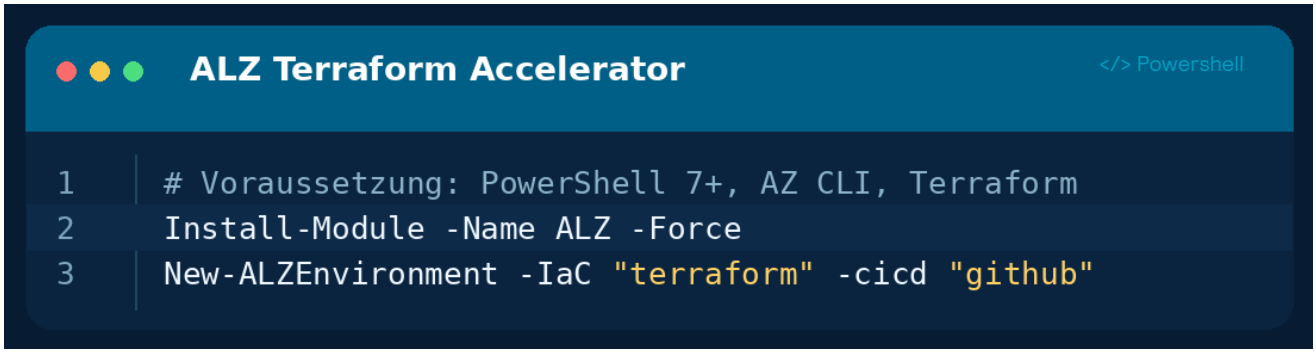
Kapitel 14

ALZ Terraform Accelerator

it starts here.

Wer das alles nicht manuell aufsetzen will, kann den ALZ Terraform Accelerator nutzen. Das ist ein PowerShell-basiertes Scaffolding-Werkzeug von Microsoft, das einen Fragenkatalog stellt (Tenant, Regionen, Netzwerkdesign, GitHub oder Azure DevOps) und daraus alles automatisch generiert:

- Den vollständigen Terraform-Code auf Basis des *avm-ptn-alz*-Moduls
- Die konfigurierte CI/CD-Pipeline inkl. OIDC-Setup
- Das State-Storage-Backend



Der Accelerator ist ideal für Greenfield-Projekte, die exakt dem CAF-Standard folgen. Sobald du eigene Abweichungen brauchst, wirst du den generierten Code anpassen müssen, dafür ist das Verständnis aus diesem Artikel bereits die Grundlage.

previder

Kapitel 15

Weiterführende Links

it starts here.

Je nach Wissensstand des Lesers, kann dieser Artikel tatsächlich auch mehr neue Fragen aufwerfen als er beantwortet. Deshalb habe ich alle Links zu offiziellen Dokumentationen der in diesem Artikel verwendeten Konzepte hier aufgeführt.

- Azure Verified Modules — Übersicht aller Terraform-Module
- AVM Pattern Modul: avm-ptn-alz
- AVM Pattern Modul: avm-ptn-alz-management
- Azure Landing Zones Terraform Modul auf GitHub (caf-enterprise-scale, Legacy)
- ALZ Terraform Accelerator
- Workload Identity Federation — GitHub Actions und Azure
- Workload Identity Federation — Azure DevOps
- CAF Enterprise Scale Wiki
- Azure Landing Zones — Konzept-Dokumentation
- Microsoft Security Adoption Framework (SAF) — ergänzt das CAF um konkrete Zero-Trust-Umsetzung und Security-Reifegradmodell
- Azure Well-Architected Framework — die fünf Säulen für Workload-Design auf der Landing Zone
- Checkov — IaC Security Scanner — statische Analyse für Terraform, Bicep, ARM und weitere IaC-Formate

previder

Previder Deutschland

info@previder.de
www.previder.de

it starts here.