AVANTech-Day 2026

AVANTEC
Competence. Security. Trust.

Zscaler Automatisierung

Christian Schnittert
Security Engineer,
AVANTEC

Inhalt

Um was geht es in dieser Präsentation?

AVANTEC
Competence. Security. Trust.

- Zscaler API use cases
- Zscaler API Zugriffsarten
- Zscaler mit Terraform Deep Dive
  - Aufbau
  - Wie führe ich es aus?
  - CI/CD Integration
  - Demo
- Allfällige Erweiterungen
  - URL-Kategorien Pflege
  - Integration mit ITSM
- Fazit & Lessions Learned
- Q&A

Zscaler API use cases

AVANTEC
Competence. Security. Trust.

- Policy-Konfiguration
  - Ausnahmen schneller implementieren
  - Temporäre “Test-” Policies regelmässig überschreiben
- Threat Intelligence Integration
  - Externe IOCs implementieren
  - Kompromittierte Benutzer blockieren
- Infrastructure as Code (IaC)
  - Auditierbar
  - Skalierbar
  - Versioniert

Zscaler API benutzen

AVANTEC
Competence. Security. Trust.

```python
import requests
import urllib3

from cred import CLIENT_ID, CLIENT_SECRET, VANITY_DOMAIN

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

CLIENT_ID = CLIENT_ID
CLIENT_SECRET = CLIENT_SECRET
VANITY_DOMAIN = VANITY_DOMAIN
API_BASE_URL = "https://zsapi.zscaler.net/api/v1" # Adjust to your specific ZIA cloud
AUTH_URL = f"https://{VANITY_DOMAIN}.zslogin.net/oauth2/v1/token"

def get_access_token() -> Any:
    payload = {
        "grant_type": "client_credentials",
        "client_id": CLIENT_ID,
        "client_secret": CLIENT_SECRET,
        "audience": "https://api.zscaler.com"
    }
    response = requests.post(AUTH_URL, data=payload, verify=False)
    response.raise_for_status()
    return response.json().get("access_token")

def main() -> None:
    token = get_access_token()
    headers = {
        "Authorization": f"Bearer {token}",
        "Content-Type": "application/json"
    }

    categories_resp = requests.get(f"{API_BASE_URL}/urlCategories", headers=headers, verify=False)
    categories_resp.raise_for_status()
    categories = categories_resp.json()
    print(categories)
    ## do whatever...
```

Direkt (Maximale Flexibilität mit viel Code)

```python
from zscaler import ZscalerClient

config = {
    "clientId": CLIENT_ID,
    "clientSecret": CLIENT_SECRET,
    "vanityDomain": VANITY_DOMAIN, # xxx if xxx.zslogin.net
    "verify": False,
    "logging": {"enabled": False, "verbose": False},
}

def main() -> None:

    with ZscalerClient(config) as client:
        categories, _resp, _err = client.zia.url_categories.list_categories()

        for category in categories:
            ## do whatever
            print(category.configured_name)
            if category.id is not None:
                client.zia.url_categories.add_urls_to_category(category.id, urls=["url1.com", "url2.com"])

if __name__ == "__main__":
    main()
```

Zscaler SDK (Viel Flexibilität mit weniger Code)

```
resource "zia_url_categories" "example" {
  super_category       = "USER_DEFINED"
  configured_name      = "MCAS Unsanctioned Apps"
  description          = "MCAS Unsanctioned Apps"
  keywords             = ["microsoft"]
  custom_category      = true
  type                 = "URL_CATEGORY"
  urls = [
    "added.by.terraform"
  ]
}
```

Terraform (Konfiguration vorgeben, kein Code)

Zscaler mit Terraform Deep Dive

Aufbau

AVANTEC
Competence. Security. Trust.

- main.tf
  - Primäre Infrastruktur-Konfiguration
  - Definiert welche Ressourcen provisioniert werden müssen
- providers.tf
  - Definiert die Plattform (in dem Fall Zscaler/ZIA)
  - Definiert Version / Authentifizierung
- variables.tf
  - Definiert Input Variablen
  - Typen, Beschreibung, Standardwerte.
- *.tfvars
  - Beinhaltet konkrete Variablen
  - tfvars mit Logindaten sollten nicht versioniert werden
  - *.auto.tfvars werden automatisch verwendet

Zscaler mit Terraform Deep Dive

Aufbau - Beispiele

AVANTEC
Competence. Security. Trust.

main.tf:

```
resource "zia_url_categories" "categories" {
  for_each = var.url_categories

  super_category  = each.value.super_category
  configured_name = each.value.configured_name
  description     = each.value.description
  keywords        = each.value.keywords
  custom_category = each.value.custom_category
  type            = each.value.type
  urls            = each.value.urls
}
```

providers.tf:

```
terraform {
  required_providers {
    zia = {
      version = "~> 4.0.0"
      source  = "zscaler/zia"
    }
  }
}

provider "zia" {
  client_id     = var.zia_client_id
  client_secret = var.client_secret
  vanity_domain = var.zia_vanity_domain
  zscaler_cloud = var.zia_cloud
}
```

variables.tf:

```
variable "url_categories" {
  type = map(object({
    super_category      = string
    configured_name     = string
    description         = string
    keywords            = list(string)
    custom_category     = bool
    type                = string
    urls                = list(string)
  }))
}
```

categories.tfvars:

```
url_categories = {
  "category_1" = {
      super_category      = "USER_DEFINED"
      configured_name     = "allowed_urls"
      description         = "Terra URLs"
      keywords            = ["microsoft"]
      custom_category     = true
      type                = "URL_CATEGORY"
      urls = [
        "allowed.terra1.com",
        "allowed.terra2.com",
        "allowed.terra3.com"
      ]
  },
  "category_2" = {
      super_category      = "USER_DEFINED"
      configured_name     = "blocked_urls"
      description         = "Terra URLs"
      keywords            = []
      custom_category     = true
      type                = "URL_CATEGORY"
      urls = [
        "blocked.terra1.com",
        "blocked.terra2.com"
      ]
  }
}
```

Zscaler mit Terraform Deep Dive

Wie führe ich es aus?

- Terraform Schritte:
  - init
    - Richtet Verzeichnis ein
    - Verbindet sich je nach Aufbau mit backend-Speicher
    - Lädt Provider / Module runter
  - plan
    - Abgleich (IST / SOLL)
    - Zeigt was er anpassen will (ähnlich zu einem Dry-Run)
    - Dient zur Überprüfung
  - apply
    - Führt die im plan beschriebenen Schritte aus
    - Erstellt / modifiziert oder zerstört Infrastruktur
    - Gibt definierte Rückgabewerte zurück

init:

```
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of zscaler/zia from the dependency lock file
- Using previously-installed zscaler/zia v4.0.10

Terraform has been successfully initialized!
```

plan:

```
Terraform will perform the following actions:

  # zia_url_categories.categories["category_3"] will be created
  + resource "zia_url_categories" "categories" {
      + category_id                          = (known after apply)
      + configured_name                      = "my_category_3"
      + custom_category                      = true
      + custom_ip_ranges_count               = (known after apply)
      + custom_urls_count                    = (known after apply)
      + description                          = "Terra URLs"
      + editable                             = (known after apply)
      + id                                   = (known after apply)
      + ip_ranges_retaining_parent_category_count = (known after apply)
      + super_category                       = "USER_DEFINED"
      + type                                 = "URL_CATEGORY"
      + urls                                 = [
          + "url1.com",
          + "url2.com",
        ]
      + urls_retaining_parent_category_count = (known after apply)

      + url_keyword_counts (known after apply)
    }
```

apply:

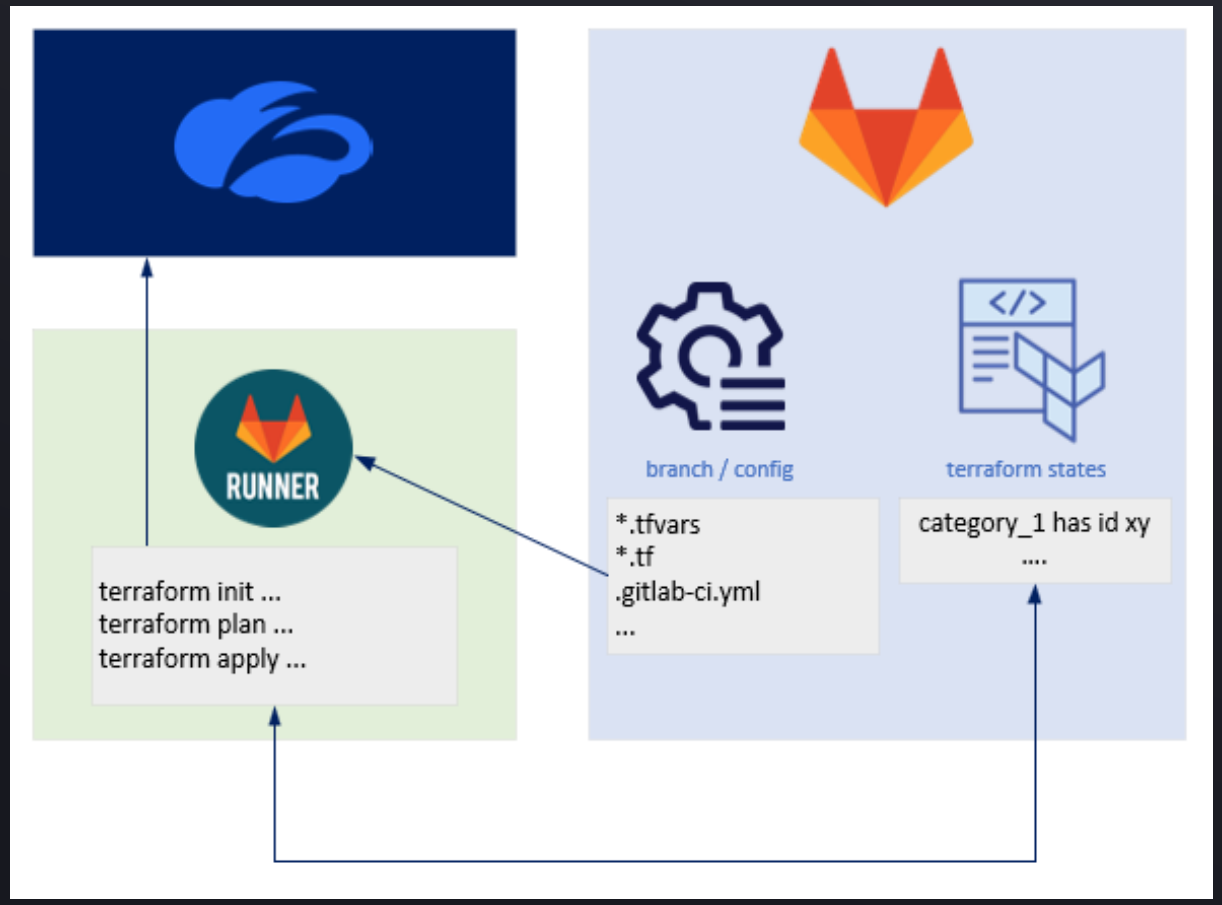
```
zia_url_categories.categories["category_3"]: Creating...
zia_url_categories.categories["category_3"]: Creation complete after 7s [id=CUSTOM_07]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Zscaler mit Terraform Deep Dive

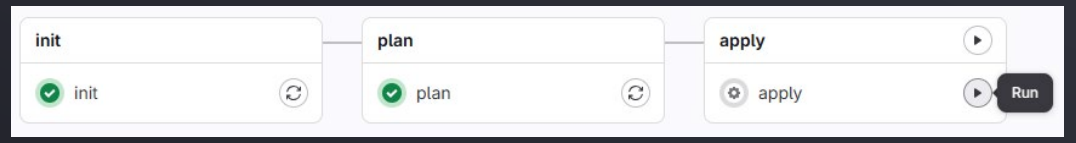
CI/CD Integration

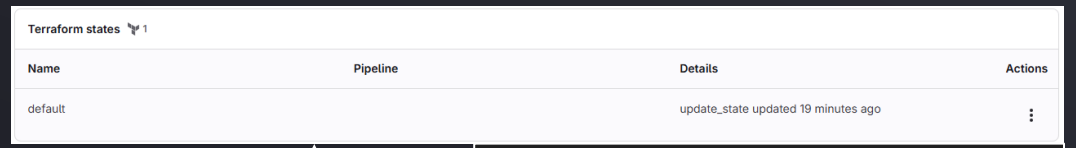
AVANTEC
Competence. Security. Trust.

Grober Ablauf:



Gitlab Pipeline:





Terraform state


```
{
  "mode": "managed",
  "type": "zia_url_categories",
  "name": "categories",
  "provider": "provider[\"registry.terraform.io/zscaler/zia\"]",
  "instances": [
    {
      "index_key": "category_1",
      "schema_version": 0,
      "attributes": {
        "category_id": "CUSTOM_06",
        "configured_name": "allowed_urls",
        "custom_category": true,
        "custom_ip_ranges_count": 0,
        "custom_urls_count": 3,
        "db_categorized_urls": [],
        "description": "Terra URLs",
        "editable": true,
        "id": "CUSTOM_06",
        "ip_ranges": [],
        "ip_ranges_retaining_parent_category": [],
        "ip_ranges_retaining_parent_category_count": 0,
        "keywords": [
          "microsoft"
        ],
```


Terraform configuration (tfvars)

```
url_categories = {
  "category_1" = {
      super_category      = "USER_DEFINED"
      configured_name     = "allowed_urls"
      description         = "Terra URLs"
      keywords            = ["microsoft"]
      custom_category     = true
      type                = "URL_CATEGORY"
      urls = [
        "allowed.terra1.com",
        "allowed.terra2.com",
        "allowed.terra3.com"
      ]
  },
```

**Falls der State verloren gehen sollte, müssen die Kategorie-IDs manuell nachgetragen werden!*

Zscaler mit Terraform Deep Dive

AVANTEC
Competence. Security. Trust.

DEMO

Allfällige Erweiterungen

URL-Kategorien Pflege

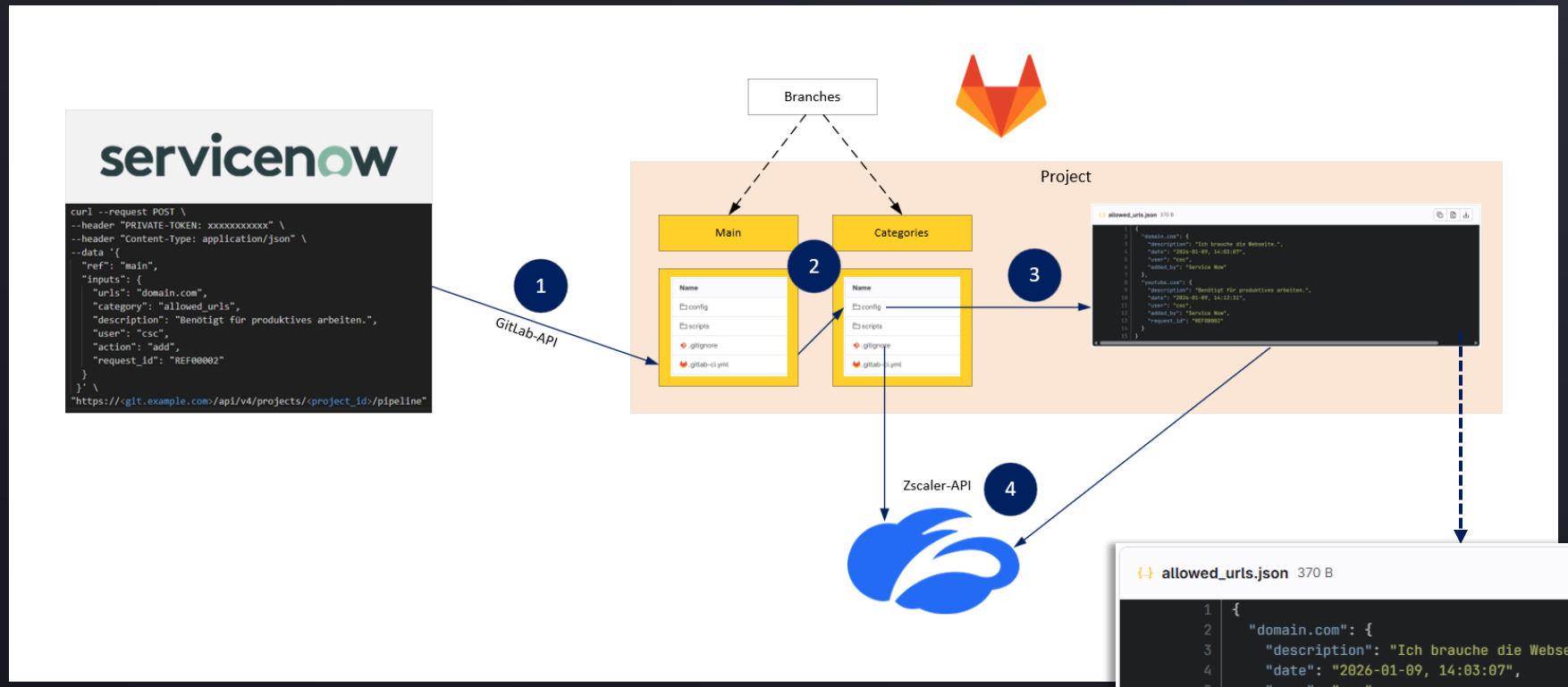
- Option 1: Direkte Anpassungen in der tfvars Konfiguration:
  - Terraform liest normal die tfvars und implementiert sie.
  - Validierung von Input muss bedacht werden.
  - Fehleranfällig. (z. B. Syntax-Fehler bei grossen Listen)
- Option 2: Manuelle Pflege:
  - Terraform erstellt Kategorien, ignoriert allerdings Änderungen an URLs.
  - URLs können in Zscalers Webui hinzugefügt werden.
  - Gefahr von State Drift
- Option 3: In Terraform URL-Changes ignorieren und zweite Automatisierung laufen lassen:
  - Terraform ist «nur» für den Grundbaustein verantwortlich.
  - Validierung kann bei der Automatisierung direkt verbaut werden.
  - Gut für grosse Anzahl Anfragen / ITSM Tools.

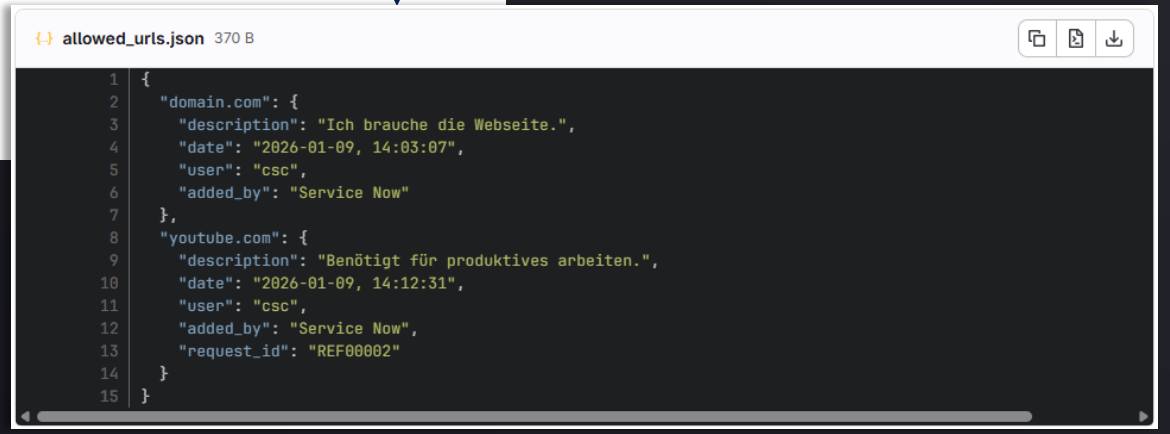
```
urls = [
  "allowed.terra1.com", ## REF0001, meier müller
  "allowed.terra2.com", ## REF0002, thomas meier
  "allowed.terra3.com"  ## REF0003, hans thomas
]
```

```
resource "zia_url_categories" "categories" {
  for_each = var.url_categories

  super_category  = each.value.super_category
  configured_name = each.value.configured_name
  description     = each.value.description
  keywords        = each.value.keywords
  custom_category = each.value.custom_category
  type            = each.value.type
  urls            = each.value.urls

  ## Ignoriert Änderungen an URL Liste:
  lifecycle {
      ignore_changes = [
        urls
      ]
    }
}
```

Allfällige Erweiterungen

ITSM-Anbindung (via Option 3)





Fazit & Lessions Learned

AVANTEC
Competence. Security. Trust.

- Start Small
  - Nicht sofort komplexe Firewall-Policies automatisieren.
  - Mit einfachen Listen (z.B. URL-Kategorien) starten und Prozesse etablieren
- Vorsicht vor «State Drift»
  - Manuelle Anpassungen und parallel laufende Terraform-Pipelines vertragen sich nicht gut.
  - «Single Source of Truth»
- Prozess vor Skript
  - Anbindung an ITSM-Tools benötigt Freigabe- und Feedback-prozesse (hat der Change funktioniert)

AVANTEC
Competence. Security. Trust.

Q&A

AVANTech-Day 2026

AVANTEC
Competence. Security. Trust.

DANKE

BeyondTrust · CHECK POINT · CROWDSTRIKE · digicert · netskope · THALES · tenable · xorlab · zscaler

AVANTEC
Competence. Security. Trust.

Nächste Session 16:00

Session 4

Was ist PAM mit Zero Standing Privileges (ZSP)? — **Frei, 4. OG**

Produkt: BeyondTrust
Referenten: Michael Scherzinger, Jessica Warland

Kontrolle und Schutz im Zeitalter von KI – Der Zscaler-Ansatz für GenAI Security — **Cancellara, 4. OG**

Produkt: zscaler
Referent: Jonas Kugler

How to deploy Check Ponit SASE in 30 minutes — **Holdener, 5. OG**

Produkt: CHECK POINT
Referent: Tobias Wälchli

Einblick ins Cyber Defence CenterPRA — **Odermatt, 4. OG**

Referent: Tobias Balschun