

...

- Was ? Erweiterter Einstieg in Hashicorp Terraform (etwas AWS lastig ...)
- Ziel ? Generelles Verständnis und die Fähigkeit zur Erstellung eigener IaC - Deployments
- Wer seid ihr ? Fachlicher Hintergrund ? Programmierkenntnisse ?
- Bei Fragen – bitte sofort fragen.
- Euer Workshop – Eure Geschwindigkeit.
- erinnert mich bitte an Pausen.

# Hashicorp Terraform - IaC.

## IBM kauft Hashicorp für 6,4 Milliarden US-Dollar

*"Durch die Kombination des Portfolios und der Expertise von IBM mit den Fähigkeiten und Talenten von Hashicorp entsteht eine umfassende Hybrid-Cloud-Plattform für das KI-Zeitalter,,*

IBM's CEO Arvind Krishna

# Hashicorp Terraform - Alternativen.



# Pulumi

<https://www.pulumi.com/>

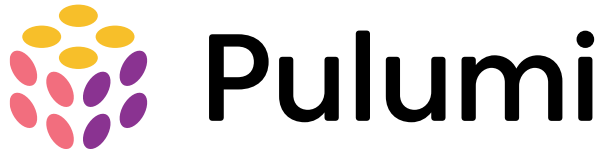


# OpenTofu



<https://opentofu.org/>

# Hashicorp Terraform - Alternativen.



```
import pulumi
from pulumi_aws import ec2

# AMI für Amazon Linux 2 in der Region us-east-1
ami = ec2.get_ami(most_recent=True,
                  owners=["amazon"],
                  filters=[{"name": "name", "values": ["amzn2-ami-hvm-*-x86_64-ebs"]}))

# Sicherheitsgruppe erstellen
group = ec2.SecurityGroup('web-sg',
                           description='Enable HTTP access',
                           ingress=[
                               {'protocol': 'tcp', 'from_port': 80, 'to_port': 80, 'cidr_blocks': ['0.0.0.0/0']},
                           ])

# EC2-Instanz erstellen
instance = ec2.Instance('web-instance',
                          instance_type='t2.micro',
                          vpc_security_group_ids=[group.id],
                          ami=ami.id,
                          tags={
                              'Name': 'Pulumi-EC2',
                          })

# Öffentliche IP-Adresse der Instanz exportieren
pulumi.export('public_ip', instance.public_ip)
```

# Hashicorp Terraform - IaC.

```
# version/local backend
#
terraform {
  required_version = ">= 0.12"
  backend "local" {
    path = "state/terraform.tfstate"
  }
}

# aws client
provider "aws" {
  profile = var.profile
  region = var.region
}

# module call
module "stack" {
  source      = "../modules/stack"
  vpc_cidr    = "128.0.0.0/16"

  subnet_cidrs = ["128.0.1.0/24", "128.0.2.0/24", "128.0.3.0/24"]
  av_zones     = ["eu-central-1a", "eu-central-1b", "eu-central-1c"]

  namespace = "teccle"

  ec2 = {
    "instance_ami"    = "ami-0a02ee601d742e89f"
    "instance_type"   = "t2.nano"
    "instance_count"  = 6
    "ebs_device"      = "/dev/sdb"
    "ebs_vol_size"    = 1
    "ebs_vol_type"    = "gp2"
  }
}
...
```



HashiCorp  
**Terraform**  
**Infrastructure**  
**as**  
**Code**

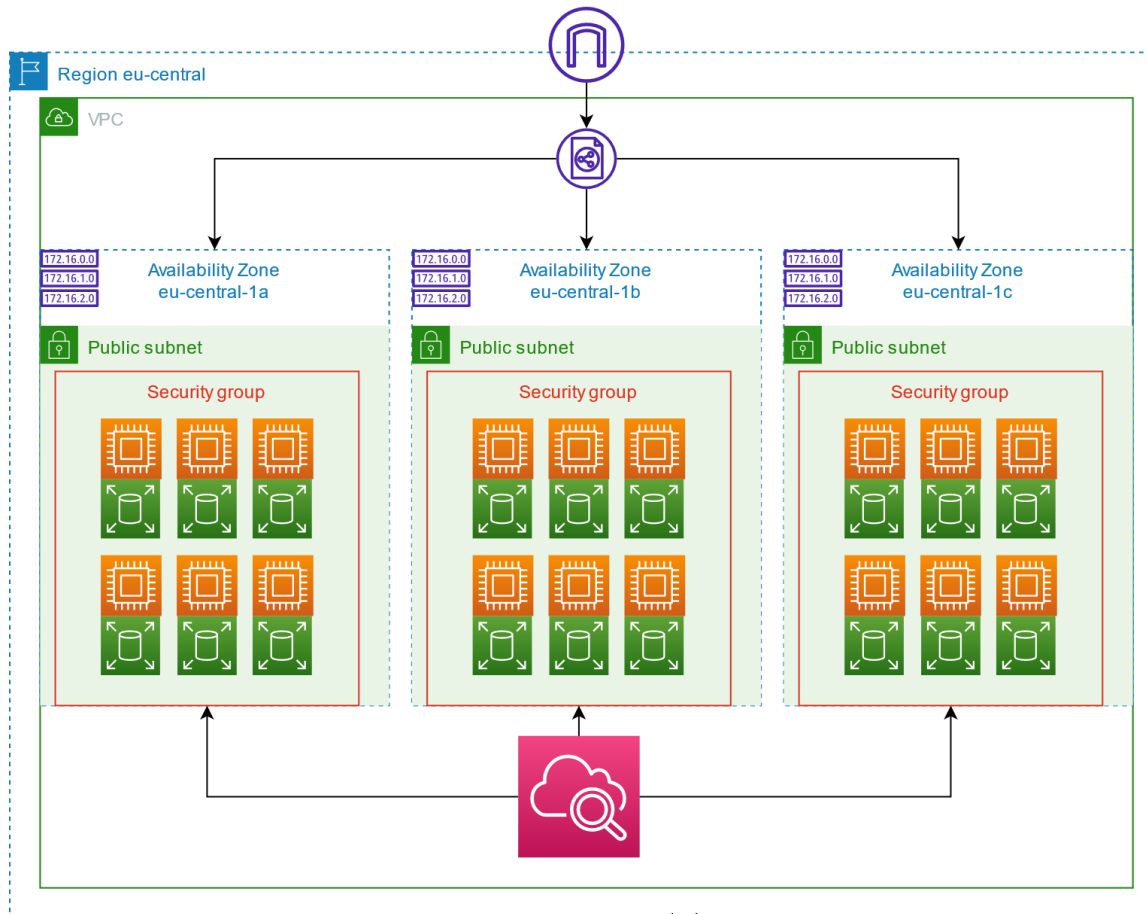
[heiko.stein@etomer.com](mailto:heiko.stein@etomer.com) // principal it-architect

## Worüber reden wir ?

# Demo !!!

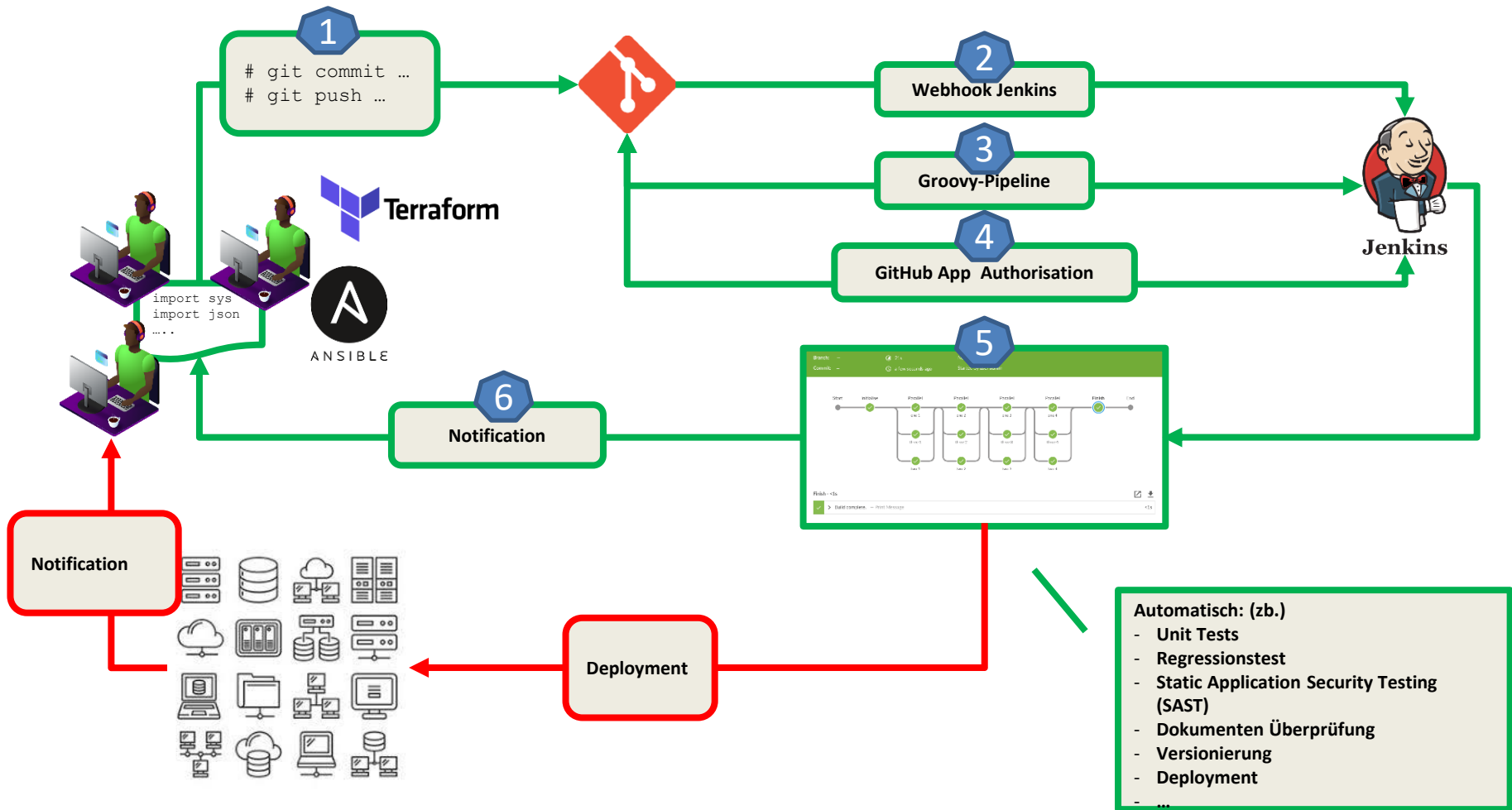


## Demo.



- Infrastruktur Deployment
  - **Hashicorp Terraform**
    - AWS Provider
      - VPC/AZ/IGW
      - Routes/Subnets/S G's
      - EC2/EBS/ELB
      - Cloudwatch
- Applikations Deployment
  - **Ansible**
    - Customizing OS
    - Setup Apache Webserver
    - Konfiguration Apache Webserver
    - Konfiguration Content
    - Start Apache Webserver

## Demo.





## Agenda.

**1**

**Was ist Terraform?**

**2**

**Termini/Konzepte in Terraform**

**3**

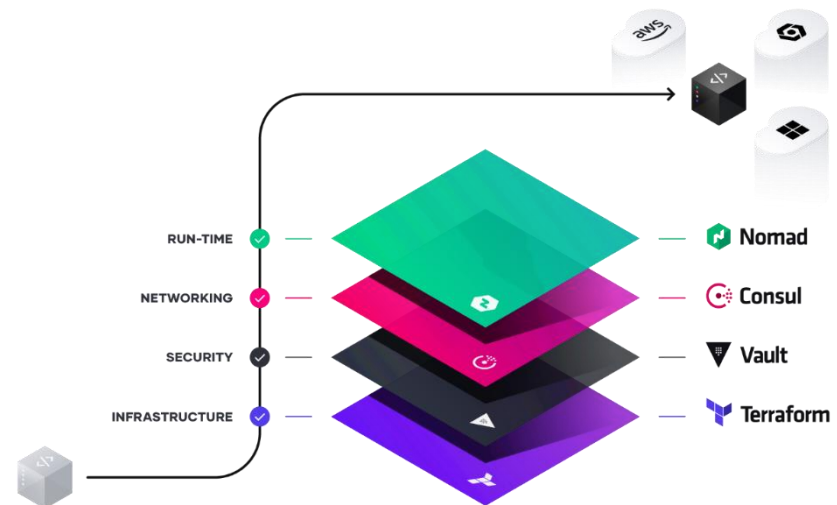
**Einstieg HCL**

**4**

**Tipps**

# Was ist Terraform ?

- Terraform ist ein Infrastruktur as Code-Tool von HashiCorp.
  - <https://www.hashicorp.com/>
- Erstmals im Juli 2014 von Hashicorp veröffentlicht
  - <https://www.hashicorp.com/products/terraform>
- CNCF Member
  - <https://landscape.cncf.io/card-mode?organization=hashi-corp>



## Was ist Terraform ?

- Terraform ist als Framework, mit folgenden Merkmalen implementiert:
  - Automatische Provisionierung von (Cloud) Infrastrukturen
    - Erstellen, Änderung und Versionierung
  - Deklarative, idempotente Definition der Infrastrukturen in HCL
    - HashiCorp Configuration Language
  - Unterstützt die Client-Architektur, sodass keine zusätzliche Konfigurationsverwaltung auf einem Server erforderlich ist
    - z.B #aws; #az
  - Terraform ist nicht **cloud-agnostisch**
    - Plugins für die jeweilige Cloud stellt der Cloudanbieter zur Verfügung
  - Modular, via GoLang erweiterbar -> <https://www.terraform.io/plugin/sdkv2>
  - Nutzung des CDK für eigenen Code -> <https://www.terraform.io/cdktf>

## Lizenz – Trouble ?

- HashiCorp hatte im August 2023 angekündigt, Terraform künftig unter der Business Source License (BSL-1.1) zu veröffentlichen, die keine Open-Source-Lizenz ist.
- Die BSL verbietet, u.a. ein gehostetes Terraform-Produkt als Konkurrenz zu HashiCorp anzubieten.
- Als Reaktion entstand der Fork OpenTofu.
- OpenTofu wurde von der Linux Foundation adoptiert, die die Weiterentwicklung betreut – langfristiges Ziel ist, daraus ein Projekt der Cloud Native Computing Foundation (CNCF) zu machen.
- Dort wiederum ist auch HashiCorp Mitglied.

<https://www.cncf.io/?s=opentofu>

## Was ist Terraform ? – Neues Lizenzmodel – Business Source License.

- Bisher standen die Tools des Unternehmens unter der Mozilla Public License (MPL).
- Die BSL wurde ursprünglich von MariaDB als Versuch entwickelt, einen Mittelweg zwischen Open Source und kommerzieller Software zu finden. (<https://www.hashicorp.com/bsl>)
- Der Quellcode bleibt einsehbar, darf auch modifiziert werden.
- Im Wesentlichen gewährt die BSL die Freiheit, den Quellcode der Software einzusehen, zu modifizieren, abgeleitete Werke zu erstellen, ihn weiterzuverbreiten und in nicht-produktiver Umgebung zu nutzen.
- Für die produktive Nutzung überlässt die BSL es dem Lizenzgeber, ob und zu welchen Bedingungen eine solche Verwendung erlaubt ist.
- "Der Lizenzgeber kann weitere Nutzungsrechte gewähren, die eine begrenzte Produktionsnutzung ermöglichen" heißt es im Text der Lizenz.
- Hashicorp erlaubt die kommerzielle Nutzung eingeschränkt:
- **"Alle Produktionsnutzungen sind zulässig, mit Ausnahme des Hostings oder der Einbettung der Software in ein Angebot, das mit den Produkten oder Dienstleistungen von Hashicorp konkurriert."**
- Kurz gesagt: Als Anbieter einer SaaS-Plattform, die Tools von Hashicorp verwendet, muss man lizenzieren.
- In der BSL gibt es auch noch einen Fallback, nachdem die unter ihr stehende Werke nach spätestens vier Jahren automatisch auf eine als "Change License" angegebenen Lizenz wechseln.

# Was ist Terraform ? – (Nicht) agnostisch !



```
resource "aws_instance" "dev" {
  ami           = var.ec2["instance_ami"]
  instance_type = var.ec2["instance_type"]

  count = length(var.subnet_cidrs) * var.ec2["instance_count"]
  subnet_id = element(aws_subnet.dev[*].id, count.index)

  vpc_security_group_ids = [aws_security_group.dev.id]
  key_name                = aws_key_pair.dev.key_name

  ebs_block_device {
    device_name = var.ec2["ebs_device"]
    volume_type = var.ec2["ebs_vol_type"]
    volume_size = var.ec2["ebs_vol_size"]
  }

  tags = {
    Name = join("-", [var.namespace, "ec2", count.index, element(var.av_zones,
count.index)])
  }
}
```



```
resource "azurerm_linux_virtual_machine" "poc" {
  name = join("-", [var.namespace, count.index, element(var.av_zones,
count.index)])
  availability_set_id = null

  resource_group_name = azurerm_resource_group.poc.name
  location             = azurerm_resource_group.poc.location
  size                 = "Standard_F2"
  admin_username       = var.vm_admin_username
  disable_password_authentication = true
  network_interface_ids = [element(azurerm_network_interface.poc[*].id,
count.index)]
  computer_name        = join("-", [var.namespace, count.index,
element(var.av_zones, count.index)])
  count                = var.vm_count
  zone                 = element(var.av_zones, (count.index))

  admin_ssh_key {
    username = var.vm_admin_username
    public_key = data.azure_rm_ssh_public_key.ssh_pubkey.public_key
  }

  os_disk {
    caching = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }

  source_image_reference {
    publisher = "RedHat"
    offer     = "RHEL"
    sku       = "8_4"
    version   = "latest"
  }

  tags = {
    env = var.namespace
  }
}
```

- Terraform ist methodisch Cloud-agnostisch.
- Es bietet keine anbieterunabhängige Abstraktion für Cloud-Ressourcen (!)

## Was ist Terraform ? – Terraform Packages.



- SaaS offering provided by HashiCorp
- Same features as the Enterprise version
- Fastest way to adopt Terraform
- Support

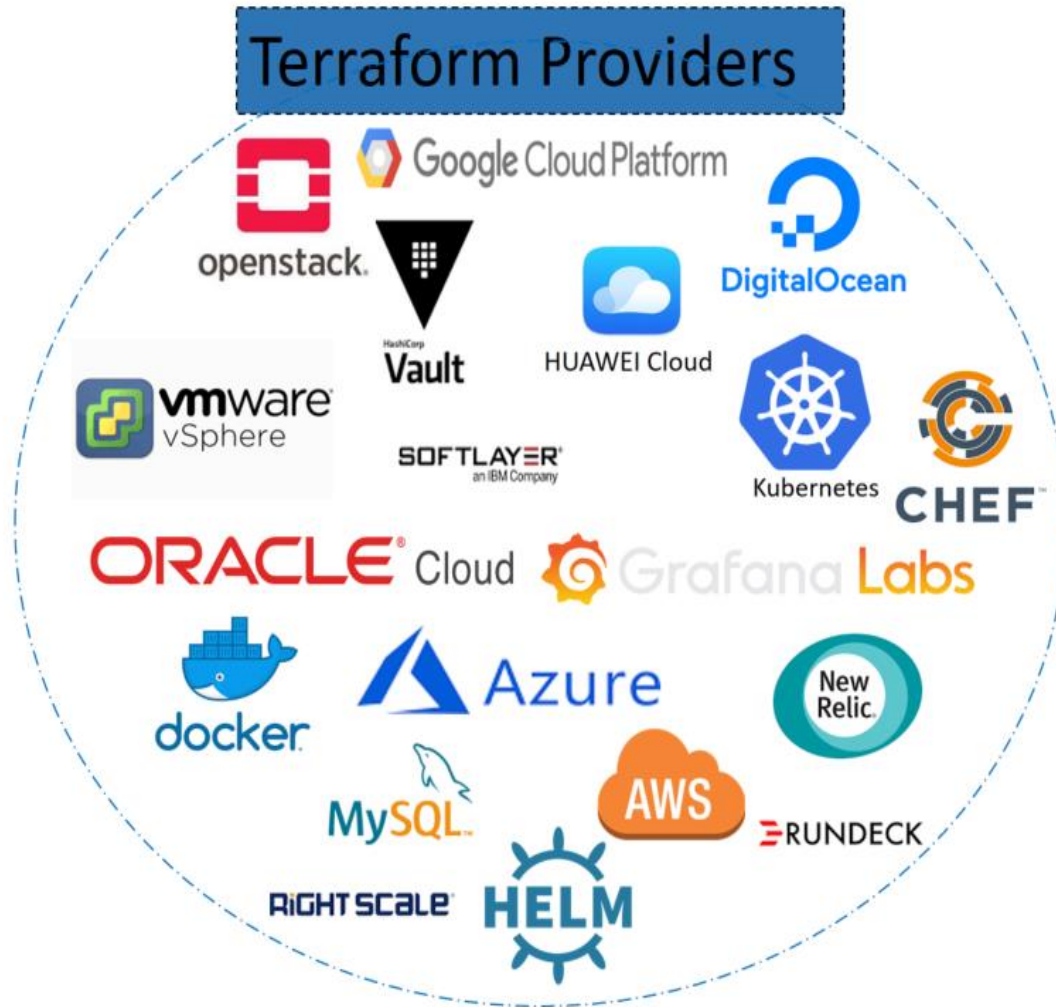


- Self-hosted Enterprise appliance
- Provide Terraform as a centralized shared service
- CLI, GUI, API
- Native VCS interactions
- Audit-trail, SSO, RBAC
- Policy enforcement
- Support



- Freemium version of Terraform
- Single binary to be installed on your machine
- Provide command line tool to create workloads based on IaC written in HCL

## Was ist Terraform ? – Provider.

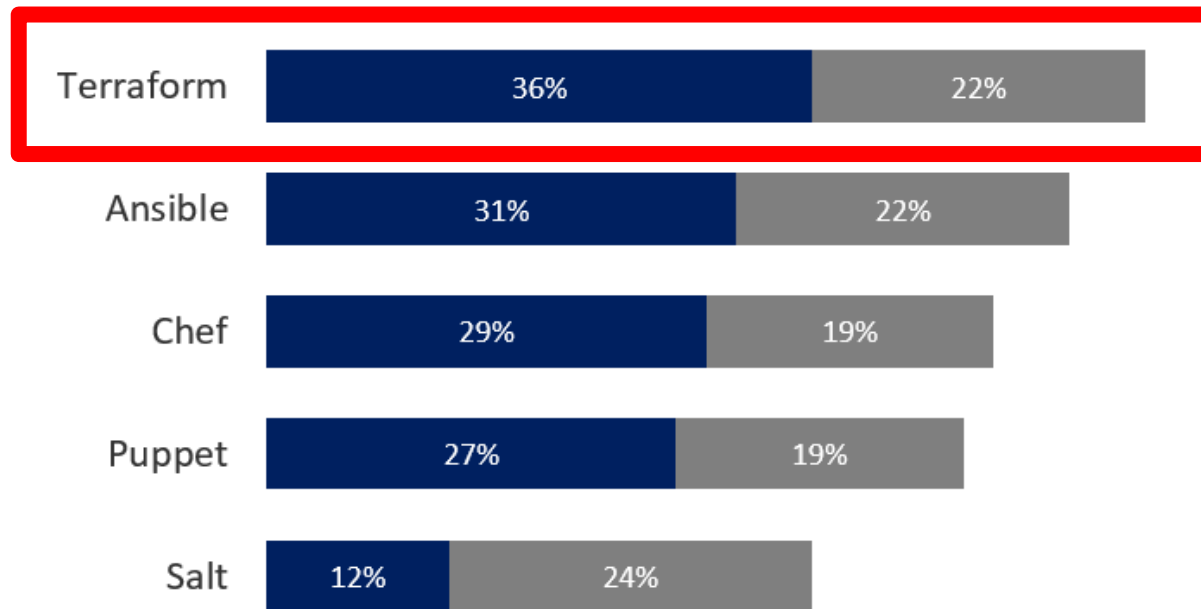




## Was ist Terraform – Nutzung ?

### Verwendete Konfigurationstools

Prozent aller Befragten



■ Im Einsatz

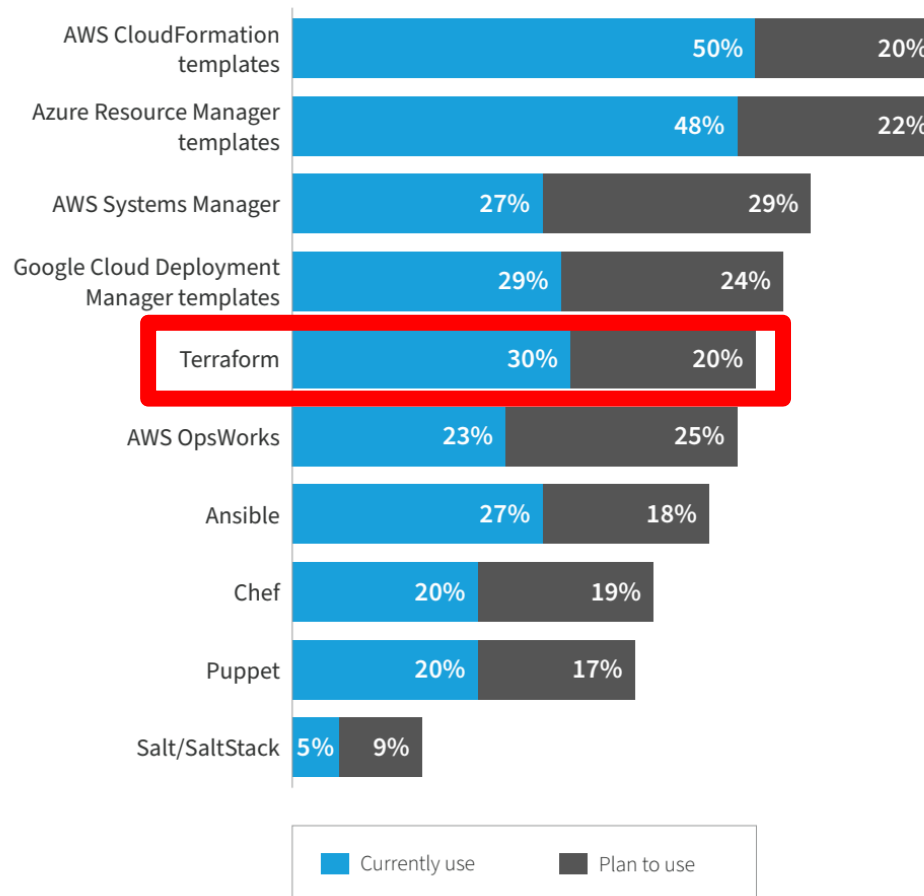
■ Nutzung geplant

N=750

Quelle: State of the Cloud Report 2021 von Flexera

# Was ist Terraform – Nutzung ?

Current and planned configuration tools for all organizations



N=753  
Source: Flexera 2022 State of the Cloud Report

## Was ist Terraform ? – Vorteile.

- Einbindung ein bestehende Workflows, Versionsverwaltung etc. problemlos möglich
- Terraform Provider ermöglichen Unterstützung zahlreicher Plattformen
- Funktionale Trennung von Orchestrierung und Systemkonfiguration-Management mit gut abbildbaren Übergabeschnittstellen
  - z.B. Terraform -> Ansible
- Deklarative Beschreibung der Zielumgebung, kein imperativer Ansatz
- Baukasten-Prinzip mit in sich geschlossenen Codeblöcken
- Selbständige Auflösung von Abhängigkeiten zwischen definierten Infrastrukturelementen (das ist echt hilfreich !!!)
- Automatisches Persistieren des Lifecycle der Deployments
- Umsetzung in HCL als einfache, Interpreter basierte Deklarationssprache
- Modularisierung der Umgebungsdeklarationen zur Nachnutzung möglich

## Was ist Terraform – Einstieg ?

- **E-Learning** -> <https://learn.hashicorp.com/>
- **Installation** -> <https://www.terraform.io/downloads>
- **Dokumentation** -> <https://www.terraform.io/docs>
- **Community** -> <https://www.terraform.io/community>

## Was ist Terraform ?

### Abgrenzung:

**Terraform ist ein Tool zur Infrastruktur-Orchestrierung, für Konfigurationsmanagement sind andere Tools besser geeignet**

	Chef	Puppet	Ansible	SaltStack	Terraform
<b>Code</b>	OSS	OSS	OSS	OSS	OSS
<b>Provider (Cloud)</b>	aws,azure,gcp,oracle, *	aws,azure,gcp,oracle, *	aws,azure,gcp,oracle, *	aws,azure,gcp,oracle, *	aws,azure,gcp,oracle, *
<b>Type</b>	Config Mgmt	Config Mgmt	Config Mgmt	Config Mgmt	<b>Orchestration</b>
<b>Infrastruktur</b>	mutable	mutable	mutable	mutable	<b>immutable/ mutable</b>
<b>Language</b>	procedural	declarative	declarative	declarative	declarative
<b>Architecture</b>	Client/Server	Client/Server	Client	Client/Server	Client

# Immutable vs. Mutable.

## Immutable Infrastrukturen

- bestehen aus unveränderlichen Komponenten, die bei jeder Bereitstellung ausgetauscht werden, anstatt „in place“ aktualisiert zu werden. Diese Komponenten werden von einem gemeinsamen Image gestartet, das einmal pro Bereitstellung erstellt wird und getestet und validiert werden kann. Das gemeinsame Image kann durch Automatisierung erstellt werden, muss es aber nicht. Die Unveränderlichkeit ist unabhängig von einem Tool oder Workflow für die Erstellung der Images.

## Mutable Infrastrukturen

- Eine veränderbare Infrastruktur kann je nach Bedarf aktualisiert und konfiguriert werden. Es ist möglich Patches anzuwenden, die Konfiguration zu aktualisieren, sie zu vergrößern oder zu verkleinern usw.

## Agenda.

1

**Was ist Terraform?**

2

**Termini/Konzepte in Terraform**

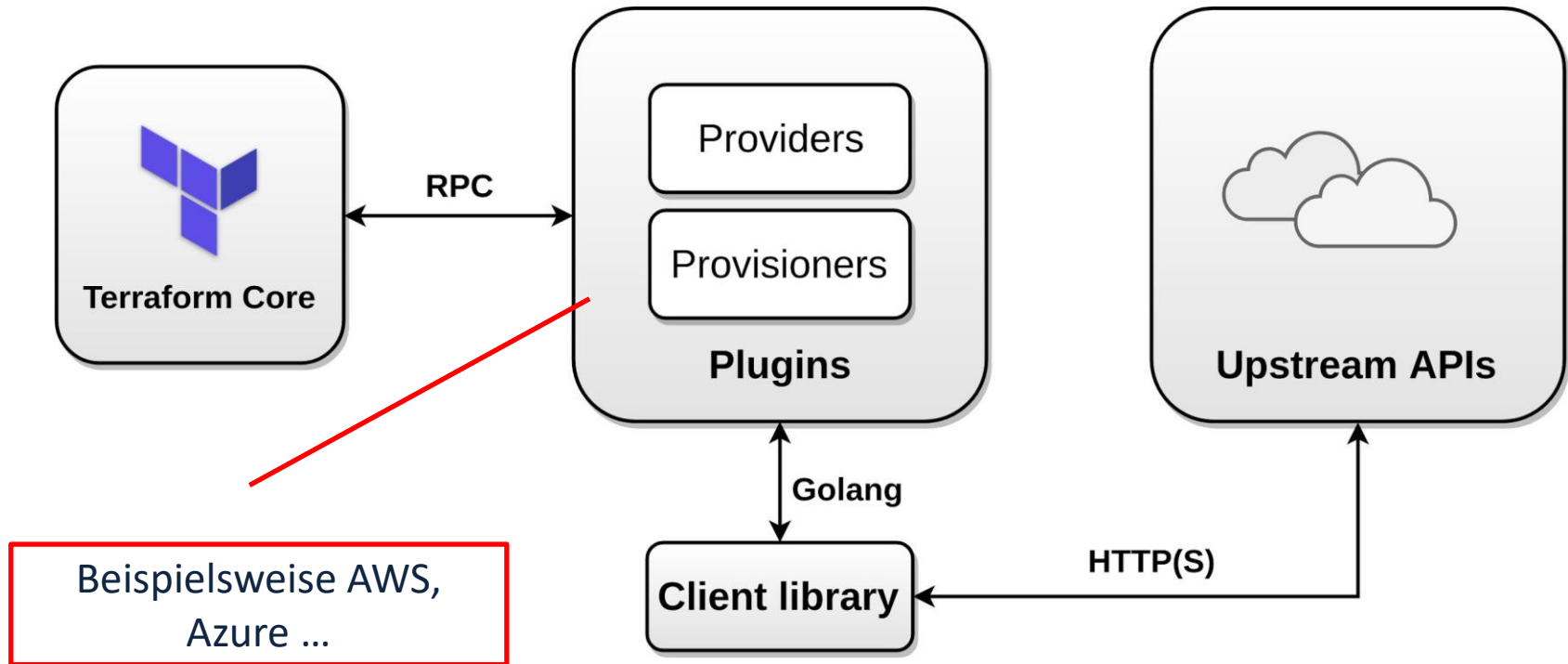
3

**Einstieg HCL**

4

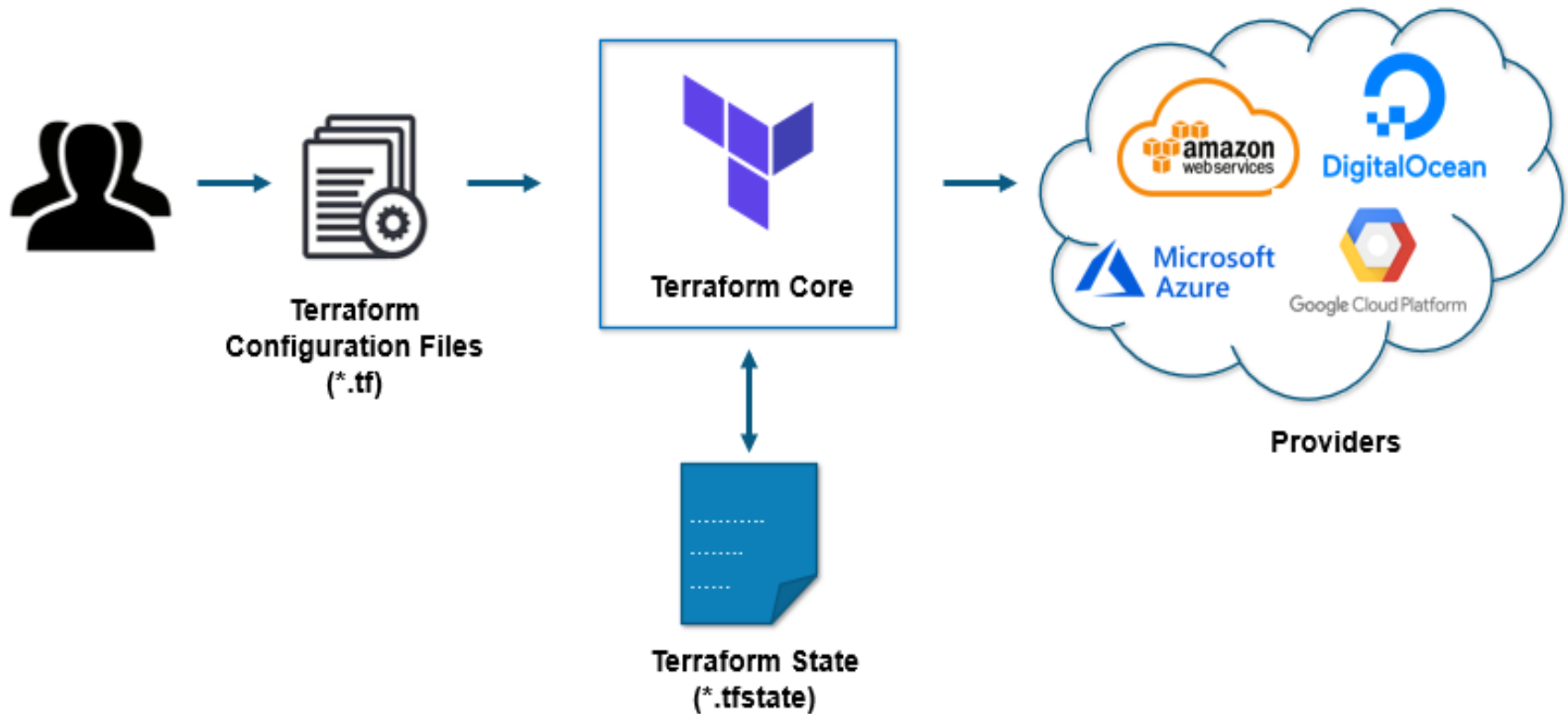
**Tipps**

## Termini/Konzepte in Terraform - Architektur.





## Termini/Konzepte in Terraform - Deployment.



## Termini/Konzepte in Terraform – Deployment Phasen.

- „**terraform init**“ initialisiert das Arbeitsverzeichnis, das aus allen Konfigurationsdateien besteht.
- „**terraform plan**“ wird verwendet, um einen Ausführungsplan zu erstellen, um einen gewünschten Zustand der Infrastruktur zu erreichen.



- „**terraform apply**“ führt dann die im Plan definierten Änderungen in der Infrastruktur durch um die Infrastruktur in den gewünschten Zustand zu versetzen. (deklarativ)
- „**terraform destroy**“ löscht:
  - alle Infrastruktur-Ressourcen
  - alle Ressourcen nach der Apply-Phase als tainted markiert sind

## Termini/Konzepte in Terraform.

**State** dient zur Speicherung der Informationen über die verwaltete Infrastruktur.

**Provider** Plugins, um mit Cloud-Anbietern usw. und anderen APIs zu interagieren. Jeder Provider fügt eine Reihe von Ressourcentypen und/oder Datenquellen hinzu, die Terraform verwalten kann

**Expressions** dienen der Referenz auf verwendete Values/Variablen bzw. zur Berechnung/Modifikation/Ausführungssteuerung.

**Functions** sind built-in Funktionen zur Transformation und Kombination von Values/Variablen.

**Module** sind die Methode zur Nachnutzung und Paketierung von HCL Code. Ein Modul besteht aus einer Menge von [.tf | .tf.json] Dateien in einem Verzeichnis.

**Ressourcen** sind das wichtigste Element in HCL. Jeder Ressourcenblock beschreibt ein oder mehrere Infrastrukturobjekte.

**Datasources** erlauben Zugriff auf Informationen außerhalb von Terraform definiert sind bzw. in anderen separaten Terraform-Konfiguration.

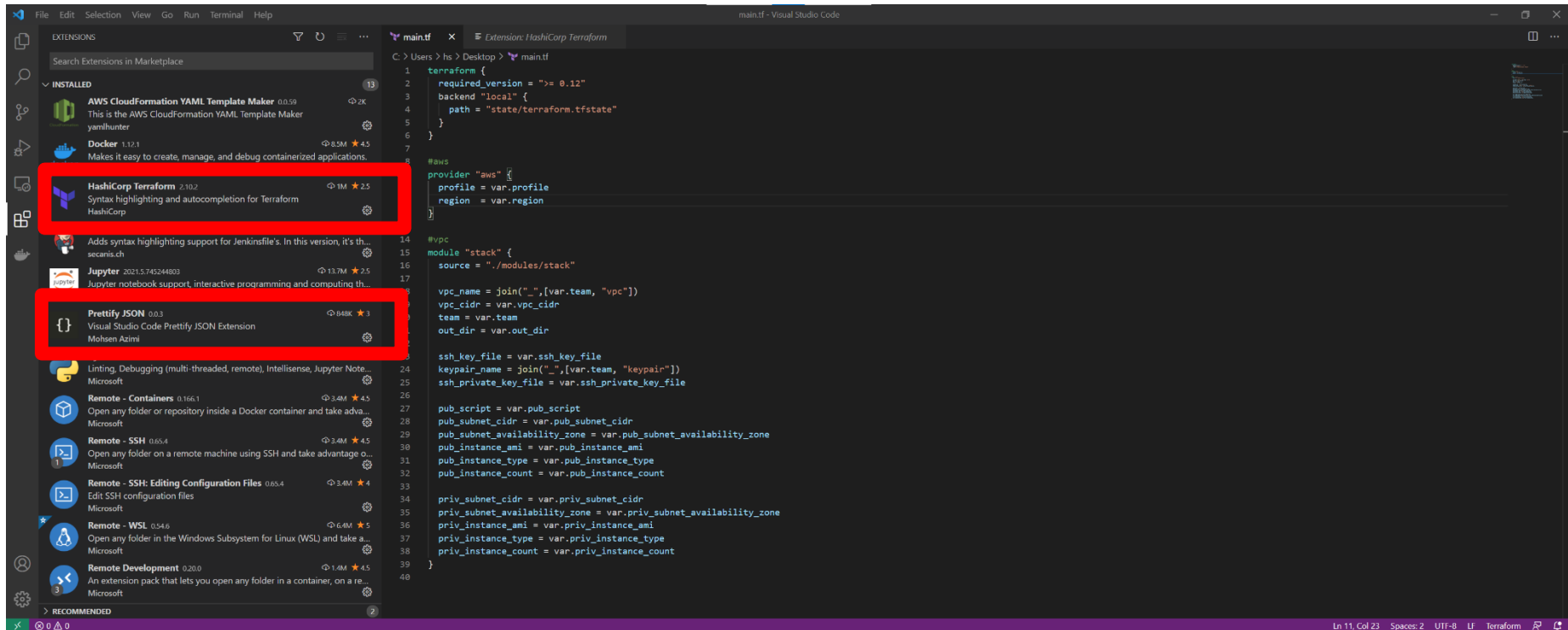
**Variablen** fungieren als Eingabe Argumente, Return-Values und Hilfsvariablen im HCL-Code.

**Settings** sind ein spezieller Terraform-Konfigurationsblocktyp, um einige Verhaltensweisen von Terraform selbst zu konfigurieren.



# Tools.

- Visual Studio Code/Codium + Terraform Extension + JSON
  - Win1[0|1] + WSL2, + Virtualbox, ...
  - Linux
  - MacOS X



## Agenda.

1

**Was ist Terraform?**

2

**Termini/Konzepte in Terraform**

3

**Einstieg HCL**

4

**Tipps**

## Einstieg HCL.

- Grundsätzlicher Aufbau Terraform Configuration File (\*.tf)

```
<RESOURCE TYPE> <RESOURCE NAME> "<RESOURCE LABEL>" {  
  # Block body  
  <IDENTIFIER> = <EXPRESSION> # Argument  
}  
...  
...  
<RESOURCE TYPE> <RESOURCE NAME> "<RESOURCE LABEL>" {  
  # Block body  
  <IDENTIFIER> = <EXPRESSION> # Argument  
}
```

## Einstieg HCL.

- Einfachste Terraform Configuration-File Anwendung

```
# 11
total 12
drwxrwxr-x. 3 hs hs 68 May 3 16:46 ./
drwxrwxr-x. 3 hs hs 24 May 3 16:38 ../
drwxrwxr-x. 8 hs hs 163 May 3 16:38 .git/
-rw-rw-r--. 1 hs hs 85 May 3 16:38 .gitignore
-rw-rw-r--. 1 hs hs 1323 May 3 16:38 main.tf
-rw-rw-r--. 1 hs hs 1142 May 3 16:38 README.md
```

- [main.tf](#)

## Termini/Konzepte in Terraform – Demo 0.

[https://github.com/git67/tf4teccle/blob/feature/0\\_step/README.md](https://github.com/git67/tf4teccle/blob/feature/0_step/README.md)





## Einstieg HCL - Resources.

```

      .001.^
      u$0N=1
      z00BAI
      |. .=^
      ;s<| | |
      NRX~=-\
      z0c^X^
      ^B0s^^
      00$H~
      n$0=XN; .
      iBBB0vU1=~\
      $000cRr~vul
      FAHZugr~
      ZZUFABFI.^
      ;BRHV n$U~
      ^ARN1 ^0si
      'Onv~ 01.'
      c0qr rs.\
      aUu\ ul.\
      ^R0- :.\
      nn^~ -=.^|-
      =1^' .. ..

```

# Resources

## Einstieg HCL - Resources.

- **Resources**
  - **DER** meistgenutzte Sprach-Konstrukt in HCL !
  - Ein Block von einem oder mehreren Infrastrukturobjekten (Compute-Instanzen, virtuelle Netzwerke, etc.), die bei der Konfiguration und Verwaltung der Infrastruktur verwendet werden.
  - Resources bilden den Syntax und die Semantik von Objekten in Abhängigkeit vom Provider (AWS/Azure/...) ab

```
# vpc
resource ...{
...
}
```

## Einstieg HCL - Resources.

- **Resources**
  - Abbildung von Meta-Argumenten
    - depends\_on
    - (count|for\_each)
    - Provider
      - aws, azure, ...
    - Lifecycle

```
resource "azurerm_resource_group" "example" {  
  # ...  
  
  lifecycle {  
    create_before_destroy = true  
    prevent_destroy = false  
    ignore_changes = [  
      tags,  
    ]  
  }  
}
```

- Provisioners ...

## Einstieg HCL - Resources.

- Resources

```
# ec2
resource "aws_instance" "dev" {
  ami           = var.ec2["instance_ami"]
  instance_type = var.ec2["instance_type"]

  count          = length(var.subnet_cidrs) * var.ec2["instance_count"]
  subnet_id      = element(aws_subnet.dev[*].id, count.index)

  vpc_security_group_ids = [aws_security_group.dev.id]
  key_name               = aws_key_pair.dev.key_name

  ebs_block_device {
    device_name = var.ec2["ebs_device"]
    volume_type = var.ec2["ebs_vol_type"]
    volume_size = var.ec2["ebs_vol_size"]
  }

  tags = {
    Name = join("_", [var.namespace, "ec2", count.index, element(var.av_zones, count.index)])
  }
}
```

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance>

## Einstieg HCL – Resources - Referenzierung.

- Resources - Referenzierung

```
# vpc
resource "aws vpc" "dev" {
  cidr_block      = var.vpc_cidr
  enable_dns_support = true
  enable_dns_hostnames = true
  tags = {
    Name = join("_", [var.namespace, "vpc"])
  }
}

# internet gw
resource "aws internet gateway" "dev" {
  vpc_id = aws vpc.dev.id
  tags = {
    Name = join("_", [var.namespace, "igw"])
  }
}
```

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc#id>

## Einstieg HCL – Resources - Meta-Arguments.

- Meta-Arguments

```
resource "aws_instance" "pub" {  
...  
  count = var.pub_instance_count  
...  
}
```

<https://www.terraform.io/language/meta-arguments/count>

```
resource "azurerm_public_ip" "poc_lb_ip" {  
  name                = join("_", [var.namespace, "pub_ip_lb"])  
  location             = var.location  
  resource_group_name = var.rg_name  
  allocation_method   = "Static"  
  sku                 = "Standard"  
  
  tags = {  
    env = var.namespace  
  }  
  
  depends_on = [azurerm_resource_group.poc]  
}
```

[https://www.terraform.io/language/meta-arguments/depends\\_on](https://www.terraform.io/language/meta-arguments/depends_on)

## Einstieg HCL - Resources - Provisioners.

- **Provisioners**
  - Abbildung von Provisioners zur Implementation von Post-Creation Aktionen
  - Hooks für Aktionen auf dem lokalen oder auf remote Systeme, um Server oder andere Infrastrukturobjekte für den Betrieb vorzubereiten.
    - file, remote-exec, local-exec
    - chef, puppet, salt, ...
    - cloud-init
  - <https://www.terraform.io/docs/language/resources/provisioners/index.html>
  - Imho **nicht** „Best Practice“ !!

## Einstieg HCL - Resources - Provisioners.

```
provisioner "file" {  
  source      = var.pub_script  
  destination = "/tmp/pub_script.sh"  
  
  connection {  
    type      = local.conn_data.type  
    user      = local.conn_data.user  
    private_key = local.conn_data.private_key  
    host      = self.public_dns  
  }  
}
```

```
provisioner "remote-exec" {  
  inline = [  
    "chmod 755 /tmp/pub_script.sh",  
    "/tmp/pub_script.sh ~/.ssh/priv_ec2"  
  ]  
  
  connection {  
    type      = local.conn_data.type  
    user      = local.conn_data.user  
    private_key = local.conn_data.private_key  
    host      = self.public_dns  
  }  
}
```

<https://www.terraform.io/language/resources/provisioners/syntax>

```
provisioner "local-exec" {  
  command = <<-EOC  
    /usr/bin/mkdir -p ${var.out_dir}  
    [ -f ${var.out_dir}/hosts ] && /usr/bin/sed -i "/ ${self.tags["Name"]} /d" ${var.out_dir}/hosts  
    /usr/bin/echo "${self.public_ip} ${self.tags["Name"]} ${self.public_dns}" >> ${var.out_dir}/hosts  
  EOC  
}
```



## Einstieg HCL – Resources – Demo 1.

[https://github.com/git67/tf4teccle/blob/feature/1\\_step/README.md](https://github.com/git67/tf4teccle/blob/feature/1_step/README.md)



## Einstieg HCL - Variables.

## Variables

```

  \
  .001.^
  u$0N=1
  z00BAI
  |..=^
  ;s<
  NAX^=-\
  z0c^<X^
  ^B0s^~^
  00$H^
  n$0=XN;.\
  iBBB0vU1=~'\
  ^$000cRr^vul
  FAHZuqr-'
  ZZUFABFI.\
  ;BRHv n$U^~
  ^ARN1 ^0si
  'Onv^ 01.'
  c0qr rs.\
  aUU\ ul\
  ^RO- :.\
  nn^ -=.~|-^
  =1^'.. \

```

## Einstieg HCL - Variables.

- **HCL Variables**
  - Variablen müssen vor Nutzung deklariert werden
  - Deklaration einer Variablen umfasst min. Name und Value
  - Die Deklaration kann in separaten Files oder im eigentlichen Modul stattfinden
  - Sonderfall:
    - Explizite Wertzuweisung (\*.tfvars)
  - [variables list vs skalar.pptx](#)

## Einstieg HCL – Variables Precedence.

1. Environment variables
2. terraform.tfvars file
3. terraform.tfvars.json
4. \*.auto.tfvars || \*.auto.tfvars.json files
5. Alle -var and -var-file cli options



## Einstieg HCL – Variable Types.

- **HCL Variable Types**

- **string:**
  - Eine Folge von Unicode-Zeichen, die einen Text darstellen, wie z.B. „Hallo Welt,,
- **number:**
  - Ein numerischer Wert. Der Typ number kann sowohl ganze Zahlen wie 15 als auch Bruchzahlen wie 6,283185 darstellen.
- **bool:**
  - ein boolescher Wert, entweder True oder False. Bool-Werte können in bedingter Logik verwendet werden.
- **list / tupel:**
  - eine Folge von Werten, wie ["eins", "zwei"]. Elemente in einer Liste oder einem Tupel werden durch aufeinanderfolgende ganze Zahlen, beginnend mit Null, identifiziert.
- **map / object:**
  - Eine Gruppe von Werten, die durch benannte Bezeichnungen identifiziert werden, z. B. {Name = "Mabel", Alter = 52}.
  - Maps bestehen aus Elementen gleichen Typs, Objects können Elemente variante Types enthalten.

## Einstieg HCL - Input Variables - \*.tf vs \*.tfvars.

- **\*.tf**

- Variablen Deklaration und Zuweisung des Typ
- Vergabe Default Value

```
variable "profile" {  
    type          = string  
    description = "profile aws-cli"  
    default       = „NONE"  
}
```

- **\*.tfvars**

- Zuweisung Variablen-Value
  - Voraussetzung ist die erfolgte Deklaration in sofern deklariert -> \*.tf
- Überschreibt Default Value aus \*.tf
- Übergabe an Terraform Modul via:
  - Argument
    - -var-file=„<\*.tfvars>“
  - Automatisch
    - terraform.tfvars
    - .auto.tfvars

```
profile = "devops"
```

## Einstieg HCL - Input Variables.

- **Input Variables**

- Variablen (Key/Value), werden von Terraform(-Ressourcen) verwendet, um Anpassungen zu ermöglichen.
- Type Constraints
- Custom Validation Rules
- <https://www.terraform.io/docs/language/values/index.html>

```
variable "profile" {  
  type      = string  
  description = "profile aws-cli"  
  default    = "devops"  
}  
  
variable "ssh_credentials" {  
  description = "ssh key files"  
  type        = map(any)  
  default = {  
    "pub_key"  = "./files/keys/ec2-user.pub"  
    "priv_key" = "./files/keys/ec2-user"  
  }  
}
```

## Einstieg HCL - Input Variables.

- **Input Variables**
  - Argumentenvektor

```
# terraform apply -var name="value"  
# terraform plan -var-file="../../variables.tfvars"
```

- Zeichenkette (str)

```
variable "name" {  
  type = string  
  default = "value"  
}  
  
access = var.name          -> "value"
```

- Liste (arr)

```
variable "names" {  
  type = list  
  default = ["hans", "paul", "otto"]  
}  
  
access = var.names[1]      -> "paul"
```



## Einstieg HCL - Input Variables.

- **Input Variables**

- Map (key/value; dict)

```
variable "names" {  
  type = map (any)  
  default = {  
    "hans"  = "der feine root-user"  
    "paul"  = "urlaubsaulhilfe"  
    "otto"  = "gaertner"  
  }  
}  
  
access = var.names["hans"]          -> "der feine root-user"
```

- Boolean (true/false)

```
variable "ausfuehrung_installation" {  
  default = true  
}  
  
access = var.ausfuehrung_installation    -> true
```

## Einstieg HCL - Input Variables.

- **Input Variables**
  - Complex Variables → Object

```
variable "rg_config" {
  type = object({
    create_rg = bool
    name      = string
    location  = string
  })
}

resource "azurerm_resource_group" "demo_rg" {
  count      = var.rg_config.create_rg ? 1 : 0
  name      = var.rg_config.name
  location  = var.rg_config.location
  tags      = { Purpose = "Demo-RG", Automation = "true" }
}
```

## Einstieg HCL - Input Variables.

- **Input Variables**
  - Complex Variables → List Object

```
variable "storage_config" {  
  type = list(object({  
    name                        = string  
    account_kind               = string  
    account_tier               = string  
    account_replication_type   = string  
    access_tier                = string  
    enable_https_traffic_only  = bool  
    min_tls_version            = string  
    is_hns_enabled             = bool  
  }))  
}
```

## Einstieg HCL - Input Variables.

- **Input Variables**
  - Custom Validation Rule

```
# ec2
variable "ec2" {
  description = "ec2 attributes"
  type        = map(any)
  default = {
...
    "ebs_vol_size" = 0
...
  }
  validation {
    condition      = var.ec2["ebs_vol_size"] > 0
    error_message = "the ebs_vol_size have to be larger then 0"
  }
}
```

## Einstieg HCL - Output Variables.

- **Output Variables**

- Dies sind Rückgabewerte eines Terraform-Moduls, die von anderen Konfigurationen verwendet werden können bzw. z.B. nach stdout geschrieben werden können
- Bei Nutzung von Modulen ist die Enkapsulation der Values zu beachten ...
- <https://www.terraform.io/docs/language/values/outputs.html>

```
output "elb_fqdn" {
  description = "elb endpoint fqdn"
  value       = aws_elb.dev.dns_name
}

output "vpc_cidr_block" {
  description = "The CIDR block of the VPC"
  value       = concat(aws_vpc.dev.*.cidr_block, [""])[0]
}

output "ec2_public_fqdns" {
  value = module.stack.ec2_public_fqdns
}
```

## Einstieg HCL - Output Variables.

- **Output Variables**

```
# vpc
resource "aws_vpc" "vpc_devops" {
  cidr_block = var.cidr_vpc
  enable_dns_support = true
  enable_dns_hostnames = true
  tags = {
    Name = join("_",[var.namespace, "vpc"])
  }
}
```

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc#id>

```
output "vpc" {
  description = "vpc id"
  value      = aws_vpc.vpc_devops.id
}
```

## Einstieg HCL - Local Variables.

- **Local Variables**

- Variablen (Key/Value), werden von Terraform(-Ressourcen) lokal verwendet
- Geltungsbereich innerhalb des Modul
- Type Constraints
- Custom Validation Rules
- <https://www.terraform.io/docs/language/values/outputs.html>

```
locals {  
  conn_data = {  
    type = "ssh",  
    user = "ec2-user",  
    private_key = file(var.ssh_private_key_file),  
  }  
}  
...  
provisioner "file" {  
  ...  
  type = local.conn_data.type  
  ...  
}
```

## Einstieg HCL - Variables – Demo 2.

[https://github.com/git67/tf4teccle/blob/feature/2\\_step/README.md](https://github.com/git67/tf4teccle/blob/feature/2_step/README.md)





## Einstieg HCL - State.

# State

```
  ``
  .001.^
  u$0N=1
  z00BAI
  |..=``
  ;s<'''
  NAX~=-\
  z0c^<X^
  ~B0s~^^
  00$H~'
  n$0=XN;.\
  iBBB0vU1=~'\
  `$$00cRr`vul
  FAHZuqr-'
  ZZUFABFI.\
  ;BRHv n$U^~
  `ARN1 ^0si
  'Onv~ 01.'
  c0qr rs.\
  aUU\ ul\
  `RO- :.\
  nn~` -=.~|-\
  =1^'.. \..`
```

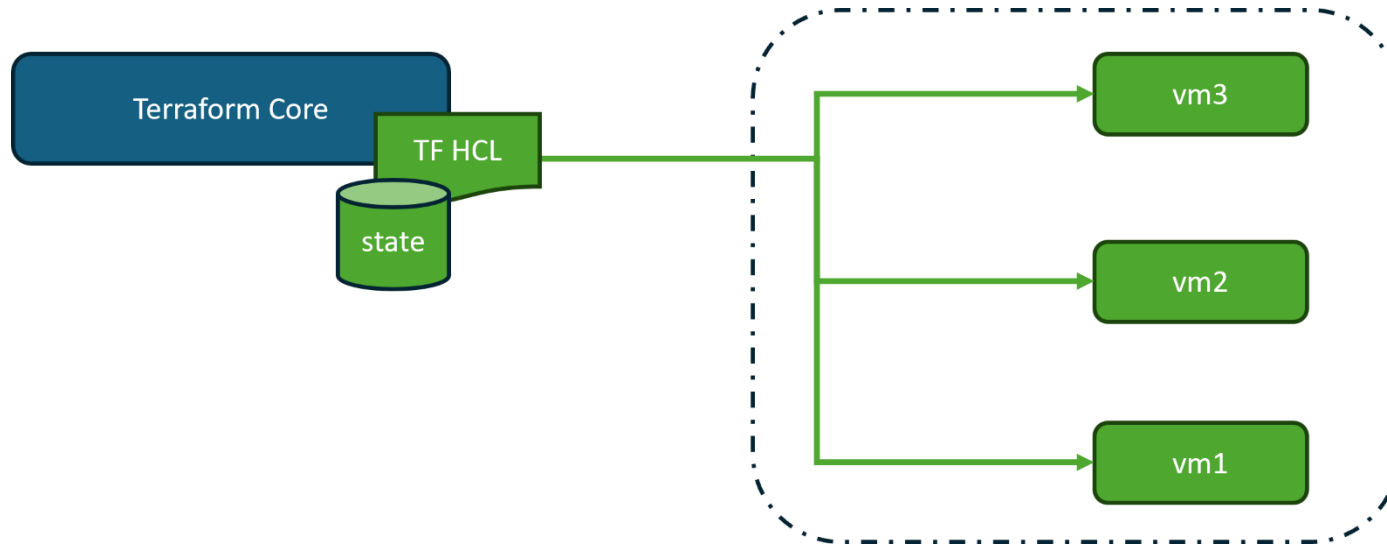
## Einstieg HCL - State.

- **State**

- Zwischengespeicherte Informationen über die von Terraform verwaltete Infrastruktur und die zugehörigen Konfigurationen.
- Dieser Status wird standardmäßig in einer lokalen Datei namens "terraform.tfstate" gespeichert, sollte aber remote gespeichert werden.
- States sollten pro Deployment separiert werden ...
- Wenn Terraform ein entferntes Objekt als Reaktion auf eine Konfigurationsänderung erstellt, wird es die Identität dieses entfernten Objekts mit einer bestimmten Ressourceninstanz aufzeichnen und dieses Objekt dann möglicherweise als Reaktion auf zukünftige Konfigurationsänderungen aktualisieren oder löschen.
- <https://www.terraform.io/docs/language/state/index.html>

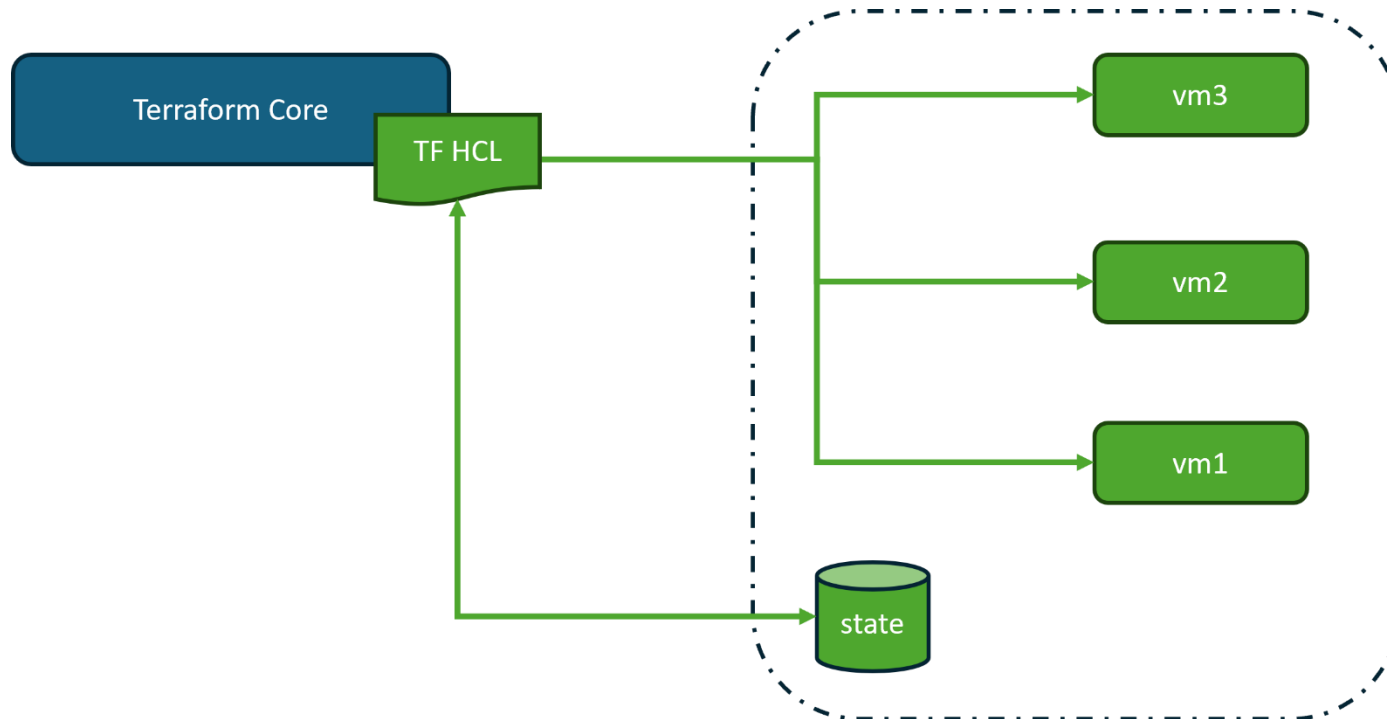
## Einstieg HCL – State Varianten.

- **Local State**



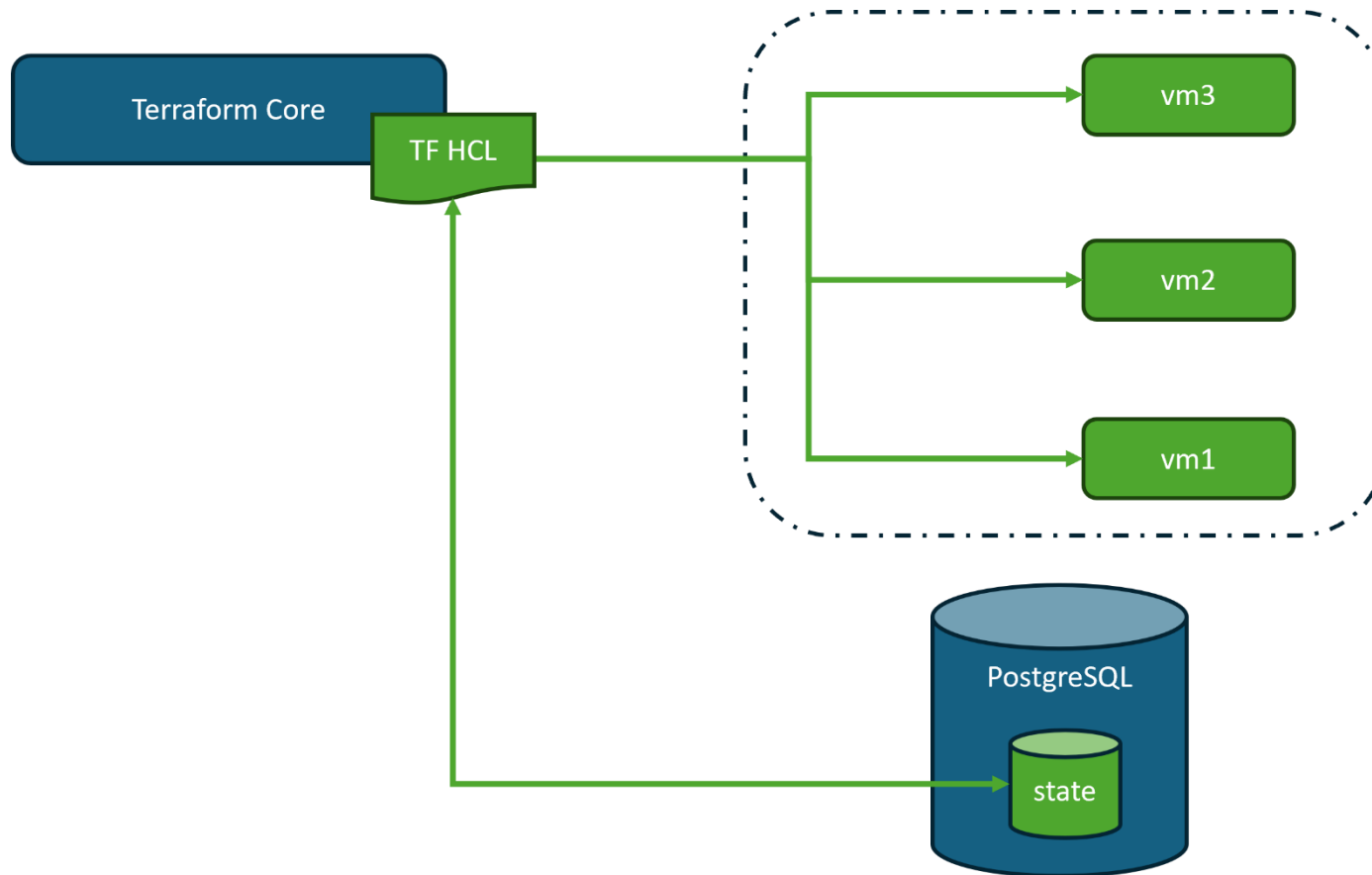
## Einstieg HCL – State Varianten.

- Remote State



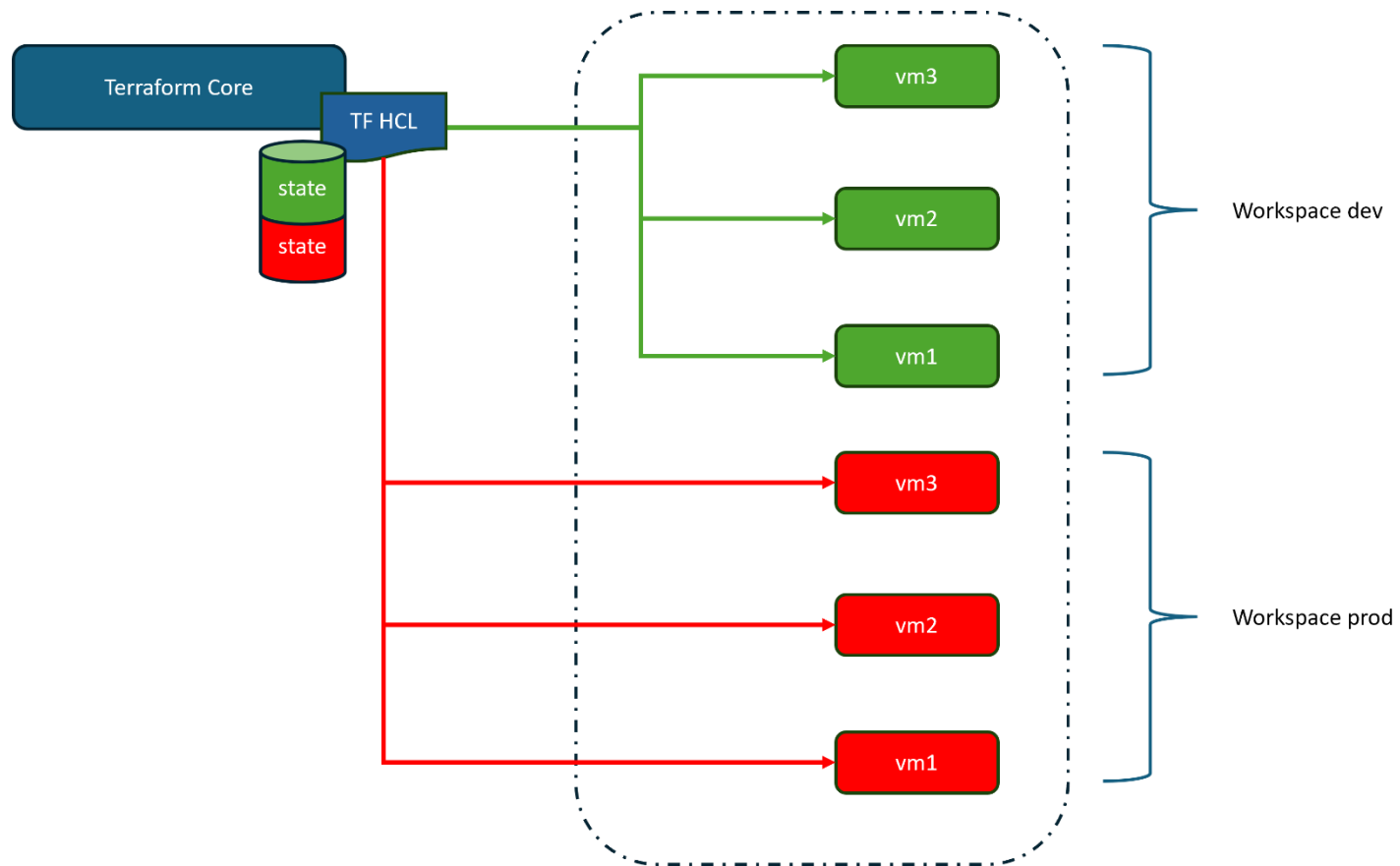
## Einstieg HCL – State Varianten.

- DB-State (PostgreSQL)



## Einstieg HCL – State Varianten.

- Separation aller Varianten via Workspaces (Beispiel local State)



# Einstieg HCL - State.

- State

```
# terraform state list
# terraform show
# terraform state show aws_vpc.vpc_devops
# terraform show -json
{"format_version":"1.0","terraform_version":"1.2.6","values":{"outputs":{"vpc":{"sensitive":false,"value":"vpc-03a26dfb213a3b731","type":"string"},"vpc-info":{"sensitive":false,"value":"vpc-03a26dfb213a3b731 : teccle_vpc","type":"string"}},"root_module":{"resources":[{"address":"aws_subnet.subnet_public_devops","mode":"managed","type":"aws_subnet","name":"subnet_public_devops","provider_name":"registry.terraform.io/hashicorp/aws","schema_version":1,"values":{"arn":"arn:aws:ec2:eu-central-1:860092230784:subnet/subnet-0f389d0dd0dafc82c","assign_ipv6_address_on_creation":false,"availability_zone":"eu-central-1a","availability_zone_id":"eucl-az2","cidr_block":"10.1.0.0/24","customer_owned_ipv4_pool":"","enable_dns64":false,"enable_resource_name_dns_a_record_on_launch":false,"enable_resource_name_dns_aaaa_record_on_launch":false,"id":"subnet-0f389d0dd0dafc82c","ipv6_cidr_block":"","ipv6_cidr_block_association_id":"","ipv6_native":false,"map_customer_owned_ip_on_launch":false,"map_public_ip_on_launch":true,"outpost_arn":"","owner_id":"860092230784","private_dns_hostname_type_on_launch":"ip-name","tags":{"Name":"teccle_subnet"},"tags_all":{"Name":"teccle_subnet"},"timeouts":null,"vpc_id":"vpc-03a26dfb213a3b731"},"sensitive_values":{"tags":{},"tags_all":{},"depends_on":["aws_vpc.vpc_devops]},"address":"aws_vpc.vpc_devops","mode":"managed","type":"aws_vpc","name":"vpc_devops","provider_name":"registry.terraform.io/hashicorp/aws","schema_version":1,"values":{"arn":"arn:aws:ec2:eu-central-1:860092230784:vpc/vpc-03a26dfb213a3b731","assign_generated_ipv6_cidr_block":false,"cidr_block":"10.1.0.0/16","default_network_acl_id":"acl-059f362c8ef3d87a0","default_route_table_id":"rtb-04a77d5134b870ced","default_security_group_id":"sg-017eaa65a6f465b28","dhcp_options_id":"dopt-9158adf8","enable_classiclink":false,"enable_classiclink_dns_support":false,"enable_dns_hostnames":true,"enable_dns_support":true,"id":"vpc-03a26dfb213a3b731","instance_tenancy":"default","ipv4_ipam_pool_id":null,"ipv4_netmask_length":null,"ipv6_association_id":"","ipv6_cidr_block":"","ipv6_cidr_block_network_border_group":"","ipv6_ipam_pool_id":"","ipv6_netmask_length":0,"main_route_table_id":"rtb-04a77d5134b870ced","owner_id":"860092230784","tags":{"Name":"teccle_vpc"},"tags_all":{"Name":"teccle_vpc"},"sensitive_values":{"tags":{},"tags_all":{}}}}]}

# terraform show -json| jq '.values.root_module.resources[0].address'
"aws_subnet.subnet_public_devops"
"aws_vpc.vpc_devops"
```

## Einstieg HCL - State – Back End.

- Lokales Back End

```
terraform {  
  required_version = ">= 0.12"  
  backend "local" {  
    path = "state/terraform.tfstate"  
  }  
}
```

- Remote Back End (AWS S3/DynamoDB)

```
terraform {  
  backend "s3" {  
    bucket          = "23443208983-terraform-states"  
    key             = "terraform/state"  
    region          = "eu-central-1"  
    shared_credentials_file = "~/.aws/credentials"  
    profile          = "devops"  
    dynamodb_table = "state-tf-locks"  
  }  
}
```



## Einstieg HCL - State – Back End.

- PostgreSQL Backend

- export PG\_CONN\_STR=postgres://postgres:postgres@desktop/state\_db?sslmode=disable
- export PG\_SCHEMA\_NAME=tf\_state

```
terraform {  
  backend "pg" {}  
}
```

- Workspaces

```
# tf workspace list  
* default  
  prod  
# tf workspace new dev  
Created and switched to workspace "dev"!  
# tf workspace list  
  default  
* dev  
  prod
```

## Einstieg HCL - State – Back End.

- PostgreSQL Backend

- export PG\_CONN\_STR=postgres://postgres:postgres@desktop/state\_db?sslmode=disable
- export PG\_SCHEMA\_NAME=tf\_state

```
# docker exec -ti pg_tf bash

root@db91575274f4:/# su - postgres
postgres@db91575274f4:~$ psql
postgres=# \c state_db
state_db=# \dt tf_state.*
               List of relations
 Schema | Name  | Type  | Owner
-----+-----+-----+-----
 tf_state | states | table | postgres

state_db=# select name from tf_state.states;
 name
-----
 default
 prod
 dev
(3 rows)
```

```
# tf workspace list
default
* dev
prod
```

## Einstieg HCL - State – Backup.

- Backup State via pull

```
# tf state pull > terraform.tfstate

# vi backend.tf
terraform {
# backend "pg" {}
  backend "local" {}
}

# tf init --reconfigure

# tf state list
```

## Einstieg HCL - State – Back End – Demo 3.

[https://github.com/git67/tf4teccle/blob/feature/3\\_step/README.md](https://github.com/git67/tf4teccle/blob/feature/3_step/README.md)



## Einstieg HCL - Module.

# Module

```
  ``
  .001.^
  u$0N=1
  z00BAI
  |..=``
  ;s<'''
  NAX~=-\
  z0c^<X^
  ~B0s~^^
  00$H~'
  n$0=XN;.\
  iBBB0vU1=~'\
  ` $000cRr`vul
  FAHZuqr-'
  ZZUFABFI.\
  ;BRHv n$U^~
  `ARN1 ^0si
  'Onv~ 01.'
  c0qr rs.\
  aUU\ ul\
  `RO- :.\
  nn~` -=.~|-\
  =1^'.. \..`
```

## Einstieg HCL - Module.

- **Modul**
  - Ist ein Ordner mit Terraform-Vorlagen in dem alle Konfigurationen hinterlegt sind.
  - Abstraktion durch logische Trennung und Nachnutzbarkeit
  - Aufruf des/der Module erfolgt via Root-Modul
  - <https://www.terraform.io/docs/language/modules/index.html>

# Einstieg HCL – Module - Struktur.

- **Struktur (Beispiel)**

- **Root Module**

```
# ls *tf
main.tf  output.tf  var_aws.tf

#cat main.tf

# module call
module "base" {
  source = "../module/base"

  cidr_vpc = "128.0.0.0/16"

  cidr_subnet = "128.0.1.0/24"

  tag_part = "teccle"

  availability_zone = "eu-central-1a"
}
```

- **Module**

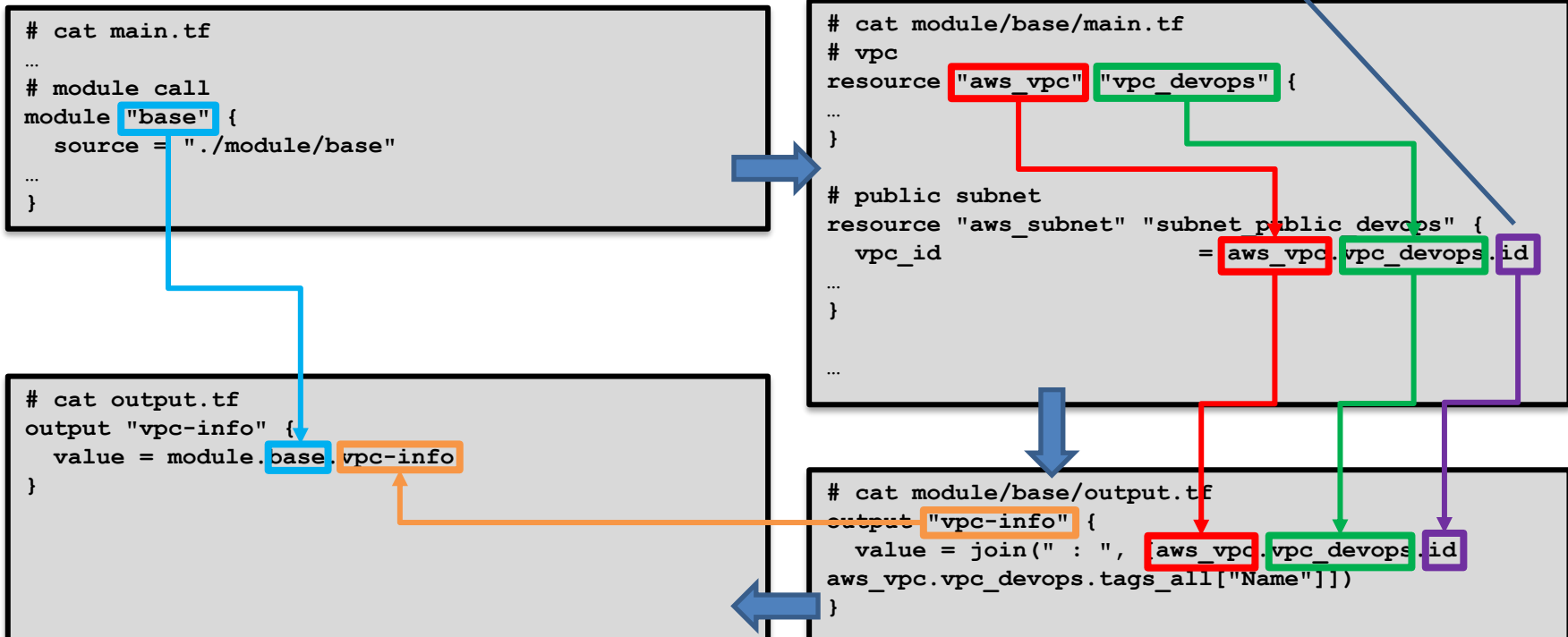
```
# ls module/base/
main.tf  output.tf  variables.tf
```

# Einstieg HCL – Module – Geltungsbereich/Zugriff Variablen.

## • Beispiel

– `aws_vpc.vpc_devops.id`

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc#id>





## Einstieg HCL - Einstieg HCL - Module – Demo 4.

[https://github.com/git67/tf4teccle/blob/feature/4\\_step/README.md](https://github.com/git67/tf4teccle/blob/feature/4_step/README.md)



## Einstieg HCL – Expressions, Functions, Datasources.

# Expressions Functions Datasources

```
  ``  
  .001.^  
  u$0N=1  
  z00BAI  
  |..=^.  
  ;s<|  
  NAX^=-\  
  z0c^<X^  
  ^B0s^~^  
  00$H^  
  n$0=XN;.  
  iBBB0vU1=^|\  
  ^$000cRr^vul  
  FAHZuqr-  
  ZZUFA0FI.  
  ;BRHv n$U^~  
  ARN1 ^0si  
  'Onv^ 01.  
  c0qr rs.  
  aUU^ ul  
  ^RO- :.  
  nn^ -.=^|~  
  =1^'..
```

## Einstieg HCL - Expressions.

- **Expressions**

- Werden zur Referenz bzw. Berechnung von Values innerhalb der Konfiguration verwendet
- Types and Values, Strings and Templates, References to Values, Operators, Function Calls, Conditional Expressions, For, Splat, Dynamic Blocks, Custom Condition Checks, Type Constraints, Version Constraints
- <https://www.terraform.io/docs/language/expressions/index.html>

- **z.B.**

```
subnet_id      = element(aws_subnet.dev.[*].id, count.index)

version = ">= 1.2.0, < 2.0.0"

String: "lalalalal"
Bool: (True|False)
Numbers: 12, 2.5
Null: null
Map: {<KEY> = <VALUE>}

var.geo != "" ? var.geo : "africa"
```

## Einstieg HCL - Functions.

- **Functions**

- Builtins, zur Manipulation/Bearbeitung von Ausdrücken
- Möglichkeit, bereitgestellte Werte zu kombinieren, zu transformieren oder auf andere Weise zu bearbeiten
- Numeric, String, Collection, Encoding ,Filesystem, Date and Time , Hash and Crypto, IP Network ,Conversion Functions
- <https://www.terraform.io/docs/language/functions/index.html>

- **z.B.**

```
Name = join("_", [var.namespace, "igw"])
Name = join("_", [var.namespace, "ec2", count.index, element(var.av_zones, count.index)])
subnet_id = element(aws_subnet.dev.*.id, count.index)
count      = length(var.subnet_cidrs) * var.ec2["instance_count"]
condition = can(regex("^[a-z]+$", var.name))
coalesce(local.from_json, "b")
...
```

## Einstieg HCL - Datasources.

- **Datasources**

- Von Provider-Plugin implementiert, zur Qualifikation von Requests auf externe Objekte, d.h. Elemente ausserhalb der eigenen Terraform Konfiguration/State
- <https://www.terraform.io/docs/language/data-sources/index.html>

- **z.B.**

```
variable "filter_str" {  
  description = "string for filter namespaces"  
  default     = "teccle-*"  
}  
  
# vpc  
data "aws_vpc" "get_vpc_id" {  
  filter {  
    name      = "tag:Name"  
    values    = [var.filter_str]  
  }  
}  
  
output "aws vpc id" {  
  value = data.aws_vpc.get_vpc_id.id  
}
```

## AWS Deployment.

```

      ^\
      .001.^
      u$ON=1
      z00BAI
      |..=.
      ;<|'|'
      NAX~=-\'
      z0c^X^
      ~B0s^^
      @0$H^
      n$0=XN;.^
      iBBB0vU1=~\'
      $000cRr~vuI
      FAHZugr-'
      ZZUFABFI.\
      :BRAHV n$U^-
      \ARM1    @si
      'Onv~    01.'
      c0qr     rs.\
      aUU     ul'\
      \RO-     :.\
      nn       -=.\|-
      =1^'.   ..


```

# Terraform + Ansible

# Erstellung Ansible Inventory.

- Erstellung Ansible Inventory via Terraform Template Funktion

```
# cat output.tf
...
resource "local_file" "inventory" {
  content = templatefile(var.ansible["ansible_inv_template"],
    {
      group_name    = "stack"
      public_ips    = module.stack.ec2_public_ips
      public_fqdns  = module.stack.ec2_public_fqdns
    }
  )
  filename = var.ansible["ansible_inv"]
}
...
```



```
# cat files/templates/ansible_inventory.template
[{{group_name}}]
{% for i, fqdn in public_fqdns ~}
{{fqdn}} ansible_host={{public_ips[i]}}
{% endfor ~}

[all:vars]
ansible_ssh_common_args='-o UserKnownHostsFile=/dev/null -o GSSAPIAuthentication=no -o StrictHostKeyChecking=no'
```



```
# cat ../ansible/inventories/inventory
[stack]
ec2-18-184-235-160.eu-central-1.compute.amazonaws.com ansible_host=18.184.235.160
ec2-54-93-109-196.eu-central-1.compute.amazonaws.com ansible_host=54.93.109.196
ec2-3-71-33-42.eu-central-1.compute.amazonaws.com ansible_host=3.71.33.42

[all:vars]
ansible_ssh_common_args='-o UserKnownHostsFile=/dev/null -o GSSAPIAuthentication=no -o StrictHostKeyChecking=no'
```

# Erstellung Ansible Inventory.

## • Erstellung Ansible Inventory via Terraform Template Funktion

```
# cat main.tf
...
# module call
module "stack" {
  source = "../module/stack"
}
```

```
# cat module/stack/main.tf
# ec2
resource "aws_instance" "dev" {
  ami           = var.ec2["instance_ami"]
  instance_type = var.ec2["instance_type"]
  ...
  tags = {
    Name = join("_", [var.namespace, "ec2", count.index,
                     element(var.av_zones, count.index)])
  }
}
```

```
# cat output.tf
...
resource "local_file" "inventory" {
  content = templatefile(var.ansible["ansible_inv_template"],
    {
      group_name    = "stack"
      public_ips    = module.stack.ec2_public_ips
      public_fqdns  = module.stack.ec2_public_fqdns
    }
  )
  filename = var.ansible["ansible_inv"]
}
```

```
# cat module/stack/output.tf
...
output "ec2_public_ips" {
  description = "List of public IP addresses assigned to the
instances, if applicable"
  value       = aws_instance.dev[*].public_ip
}

output "ec2_public_fqdns" {
  description = "List of public fqdns of ec2 instances"
  value       = aws_instance.dev[*].public_dns
}
```

[https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance#public\\_ip](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance#public_ip)



## Agenda.

**1**

**Was ist Terraform?**

**2**

**Termini/Konzepte in Terraform**

**3**

**Einstieg HCL**

**4**

**Tipps**

## Tipps.

- Vermeidung Post-Creation Provisioners
- Code nachnutzbar in dokumentierten Modules
- Keine „hard coded“ Variablen
- Nutzung von Tools für Q&A, automatisiertes Codehandling
  - CI Pipelines
    - Jenkins, GitHub Actions, Tekton, ...
  - Bump Version -> Semantic Versioning
  - Linting
    - tflint
    - ...
  - Pre-Commit Hooks
- Clean Code
  - [https://www.buecher.de/shop/softwareentwicklung/clean-code-ebook-pdf/martin-robert-c-/products\\_products/detail/prod\\_id/52843385/](https://www.buecher.de/shop/softwareentwicklung/clean-code-ebook-pdf/martin-robert-c-/products_products/detail/prod_id/52843385/)



## Tipps – Naming Schema.

- Name Tags (data sources filter qualification ...)

```
resource "aws_instance" "pub" {  
  ami           = var.pub_instance_ami  
  instance_type = var.pub_instance_type  
  subnet_id     = aws_subnet.pub.id  
  vpc_security_group_ids = [aws_security_group.pub.id]  
  key_name      = aws_key_pair.global.key_name  
  count         = var.pub_instance_count  
  tags = {  
    team = var.team  
    Name = join("_", [var.team, "pub_ec2", count.index])  
  }  
}
```

## Tipps – Dynamic Blocks.

- Dynamic Blocks
  - variables.tf

```
variable "sg_rules_egress" {  
  type = list(object({  
    from_port = number  
    to_port   = number  
    protocol  = string  
    cidr_blocks = list(string)  
  }))  
  default = []  
}
```

- .auto.tfvars

```
sg_rules_ingress = [  
  { from_port = 22, to_port = 22, protocol = "tcp", cidr_blocks = ["0.0.0.0/0"] },  
  { from_port = 8080, to_port = 8080, protocol = "tcp", cidr_blocks = ["0.0.0.0/0"] },  
  { from_port = 8086, to_port = 8086, protocol = "tcp", cidr_blocks = ["0.0.0.0/0"] },  
  { from_port = 161, to_port = 162, protocol = "tcp", cidr_blocks = ["0.0.0.0/0"] },  
  { from_port = 161, to_port = 162, protocol = "udp", cidr_blocks = ["0.0.0.0/0"] },  
]
```

- sg.tf

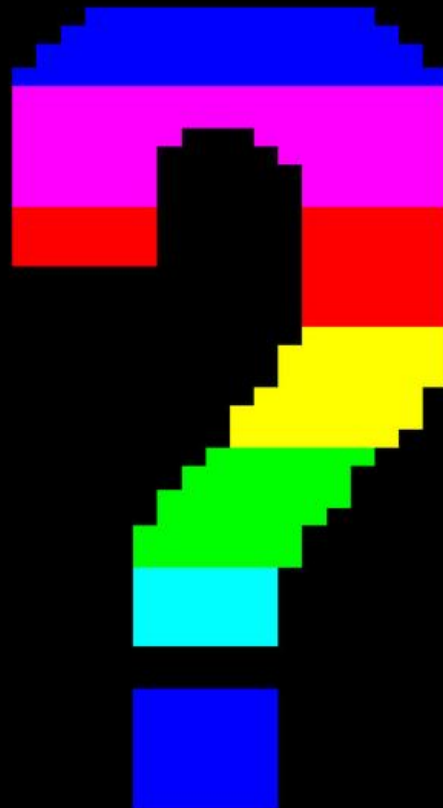
```
resource "aws_security_group" "dev" {  
  ...  
  dynamic "egress" {  
    for_each = var.sg_rules_egress  
  
    content {  
      from_port = egress.value.from_port  
      to_port   = egress.value.to_port  
      protocol  = egress.value.protocol  
      cidr_blocks = egress.value.cidr_blocks  
    }  
  }  
  ...  
}
```

## Informationen, Informationen.

- <https://learn.hashicorp.com/collections/terraform/aws-get-started>
- <https://www.terraform.io/intro/index.html>
- <https://www.terraform-best-practices.com/key-concepts>
- <https://www.terraform.io/downloads.html/>
- <https://www.terraform.io/docs/cli/index.html>
- <https://registry.terraform.io/browse/providers>
- <https://www.terraform.io/docs/language/resources/provisioners/index.html>



# Fragen.



# Vielen Dank für Eure Aufmerksamkeit.

[heiko.stein@etomer.com](mailto:heiko.stein@etomer.com) // principal it-architect



# Pro/Cons. Terraform vs. Cloudformation

Funktion	Terraform	CloudFormation
Unterstützung von AWS-Services	(+) Terraform unterstützt nahezu alle AWS-Services. Es zeichnet sich auch durch eine schnelle Unterstützung neuer Funktionen bei bestehenden Services aus. Gänzlich neue Services werden erst einige Zeit nach deren Veröffentlichung unterstützt.	CloudFormation unterstützt nahezu alle AWS-Services. Es ist eher langsam in Bezug auf Unterstützung neuer Funktionen bestehender Services. Gänzlich neue Services werden gelegentlich gleichzeitig mit deren Veröffentlichung unterstützt.
Nutzererfahrung	Terraform in der Open-Source-Variante ist ein CLI-basiertes Tool und bietet keine Benutzeroberfläche.	(+) Bei CloudFormation steht eine nutzerfreundliche Benutzeroberfläche mit wichtigen Informationen zur Nachvollziehbarkeit von Änderungen zur Verfügung.
Programmiersprache	(+) Die HCL-basierten Templates sind effizienter.	Die JSON- oder YAML-basierten Templates sind etwas schwerfälliger.
Modularisierung	(+) Terraform-Module helfen bei der Erstellung von reproduzierbarer Infrastruktur.	CloudFormation kennt keine Module. Verschachtelte Stacks und Cross-Stack-Referenzen können zum Erreichen der Modularisierung verwendet werden.
Verwaltungsaufwand	Terraform bietet ausgereifte Möglichkeiten für die Erstellung von Infrastruktur, gibt aber keinen Prozess vor. Dieser Prozess sowie alle dafür nötigen Artefakte müssen selbst verwaltet werden.	(+) CloudFormation ist ein durch AWS verwaltetes Service, das den Anwendern einige Entscheidungen und Aufwand rund um dessen Nutzung abnimmt. Dadurch werden zwar auch die Möglichkeiten begrenzt, aber der Verwaltungsaufwand verringert.
Sperrung von Ressourcen zur Vermeidung von paralleler Bearbeitung	Es gibt bei Terraform keinen integrierten Mechanismus, um mehrfache parallele Bearbeitung durch unterschiedliche Anwender zu verhindern. Dies kann zu einem inkonsistenten Zustand der Infrastruktur führen.	(+) CloudFormation besitzt eine integrierte Zustandsverwaltung und Verriegelung des Stacks, um parallele Bearbeitung zu vermeiden.
Importieren von Infrastruktur	(+) Terraform unterstützt den Import und die Verwaltung von Ressourcen, die außerhalb von Terraform erstellt wurden.	Es ist nicht möglich, außerhalb von CloudFormation erstellte Ressourcen im Nachhinein über CloudFormation zu verwalten.
Zustandsverifizierung, Änderungsübersicht und Änderungsverwaltung	(+) Terraform kann mit Hilfe des „plan“-Befehls alle geplanten Änderungen am Ist-Zustand der Infrastruktur ermitteln und eine graphische Darstellung der Abweichungen zur Verfügung stellen. Dabei werden auch historische Änderungen der Infrastruktur erkannt, die nicht durch Terraform durchgeführt wurden.	CloudFormation kann mit Hilfe von Change-Sets eine Übersicht aller Abweichungen des Soll-Zustands laut zuletzt eingespielten Templates (dem Stack) zu geplanten Änderungen erstellen. Dies ist nur bei Ressourcen möglich, die mit CloudFormation erstellt wurden. CloudFormation bietet keine Möglichkeit, gegen den Ist-Zustand der Infrastruktur zu prüfen. Änderungen an Ressourcen, die nicht durch CloudFormation durchgeführt wurden, werden nicht erkannt.
Fehlerbehandlung und Rollback	Terraform bietet kein automatisches Rollback im Fehlerfall. Fehler werden aber auf abhängige Ressourcen isoliert. Nicht abhängige Ressourcen werden immer noch erstellt, aktualisiert oder zerstört. Eine selbst implementierte oder manuelle Wiederherstellung des Zustands ist erforderlich.	(+) CloudFormation führt im Fehlerfall ein automatisches Rollback auf den letzten arbeitsfähigen Stand durch.
Rolling Updates von Auto-Scaling-Gruppen	Terraform unterstützt keine Rolling Updates von Auto-Scaling-Gruppen.	(+) CloudFormation unterstützt Rolling Updates von Auto-Scaling-Gruppen.
Externe Wartebedingungen (z.B. auf die Beendigung eines Shell-Skripts)	Terraform bietet keine Unterstützung von externen Wartebedingungen.	(+) In CloudFormation können externe Wartebedingungen definiert und davon abhängige Ressourcen erst nach deren Erfüllung erstellt oder aktualisiert werden.
Multi-Cloud-Support	(+) Terraform unterstützt mehrere Cloud-Provider.	CloudFormation ist AWS-spezifisch.
Tool-Support und Lizenzierung	Die Open-Source-Version ist kostenfrei. Zwar sind dafür keine Support-SLAs festgelegt, aber im Allgemeinen werden Probleme schnell gelöst. Terraform ist in der kostenpflichtigen Unternehmensversion mit unterschiedlichen Support-Varianten als SaaS oder private Installation erhältlich.	CloudFormation ist ein kostenloses AWS-Service. AWS bietet Support im Rahmen des ausgewählten <a href="#">Support-Plans</a> .