CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Infrastructure as Code

(5) IaC Tools: Terraform

creative commons

CLOUD NATIVE COMPUTING

Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.

KAPITEL 7

Infrastructure as Code

7 Infrastructure as Code

7.1 Virtualisierung

- Virtualisierung von Hardware-Infrastruktur
- Virtualisierung von Software-Infrastruktur

7.2 Provisionierung

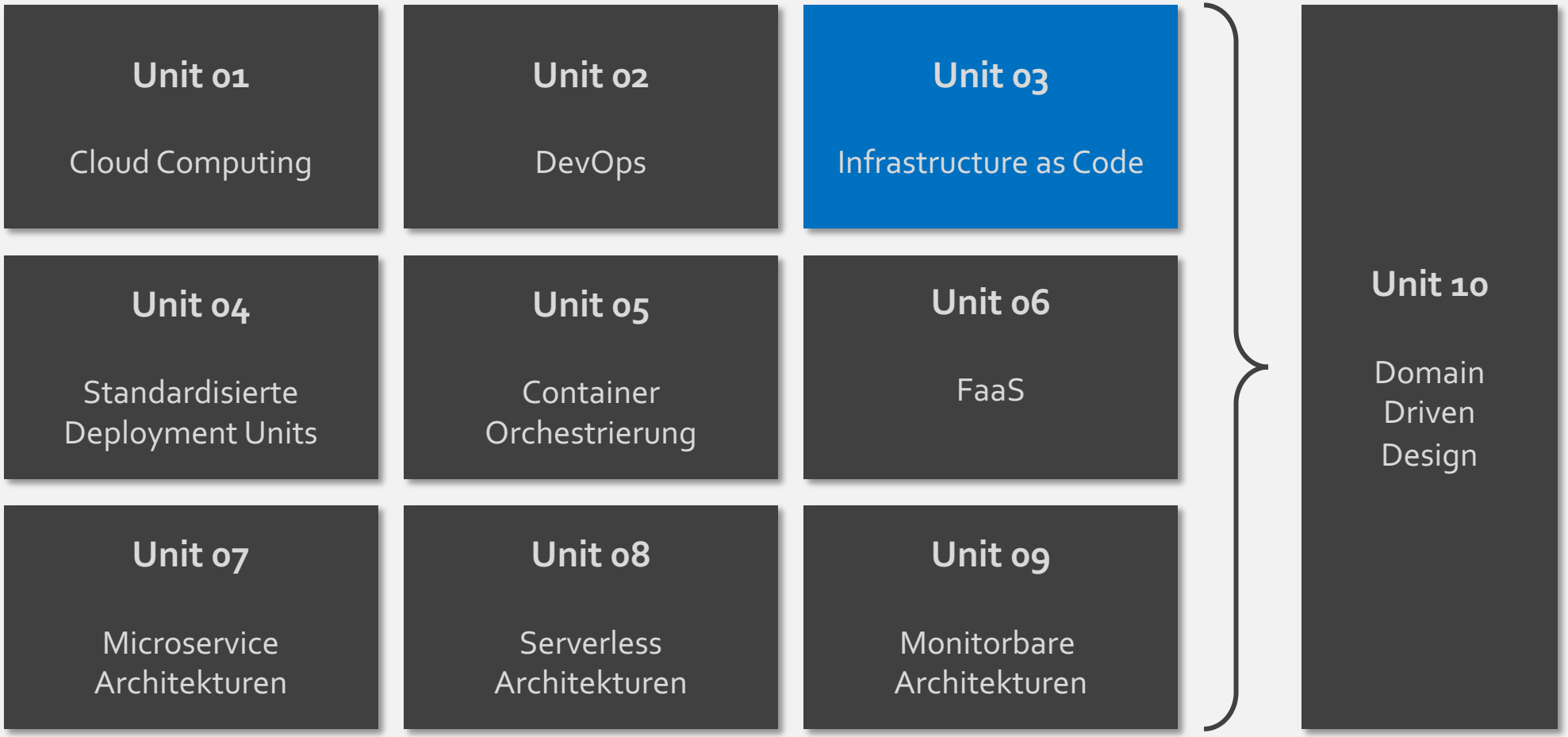
- Immutable Infrastructure
- Infrastructure as Code
- Provisionierung von lokalen Umgebungen
- Provisionierung von Multi-Host Umgebungen

7.3 Zusammenfassung

CLOUD NATIVE COMPUTING

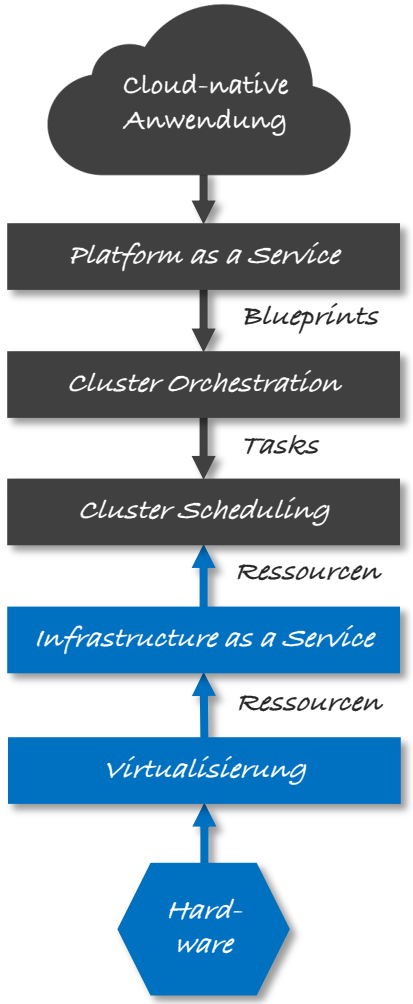
INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



INHALTE

- Virtualisierung
- Infrastructure as a Service
- Provisionierung in IaaS-basierte Infrastrukturen
- Infrastructure as Code
  - Definition, Ansätze und Methoden
  - Überblick gängiger Provisionierungswerkzeuge
  - Single VM-Provisioning mittels Vagrant
  - Multi VM-Provisioning mittels Terraform



INFRASTRUCTURE AS CODE

Tool-Überblick

Tool	Veröffentlicht	Methode	Ansatz	Geschrieben in	Anmerkungen
CFEngine	Northern.tech **1993**	Pull	deklarativ	C	-
Puppet	Puppet **2005**	Pull	deklarativ	Ruby	-
Chef	Chef **2009**	Pull	deklarativ + imperativ	Ruby	-
Vagrant	HashiCorp **2010**	Push	deklarativ + imperativ	Ruby	-
SaltStack	SaltStack **2011**	Push + Pull	deklarativ + imperativ	Python	Mittlerweile VMware
Ansible	Red Hat **2012**	Push + Pull	deklarativ + imperativ	Python	-
Terraform	HashiCorp **2014**	Push	deklarativ	Go	-
Otter	Inedo **2015**	Push	deklarativ + imperativ	.NET	Windows!

INFRASTRUCTURE AS CODE

Terraform

- Entwickelt von HashiCorp (wie auch Vagrant)
- Open Source, in Go geschrieben
- Kommandozeilenwerkzeug
- Deklarativ und Push-basierter Ansatz
- Direkte Anbindung vieler Public und Private Cloud Infrastrukturen und Plattformen (u.a. AWS, GCE, Azure, vSphere, OpenStack, Kubernetes, uvm.)

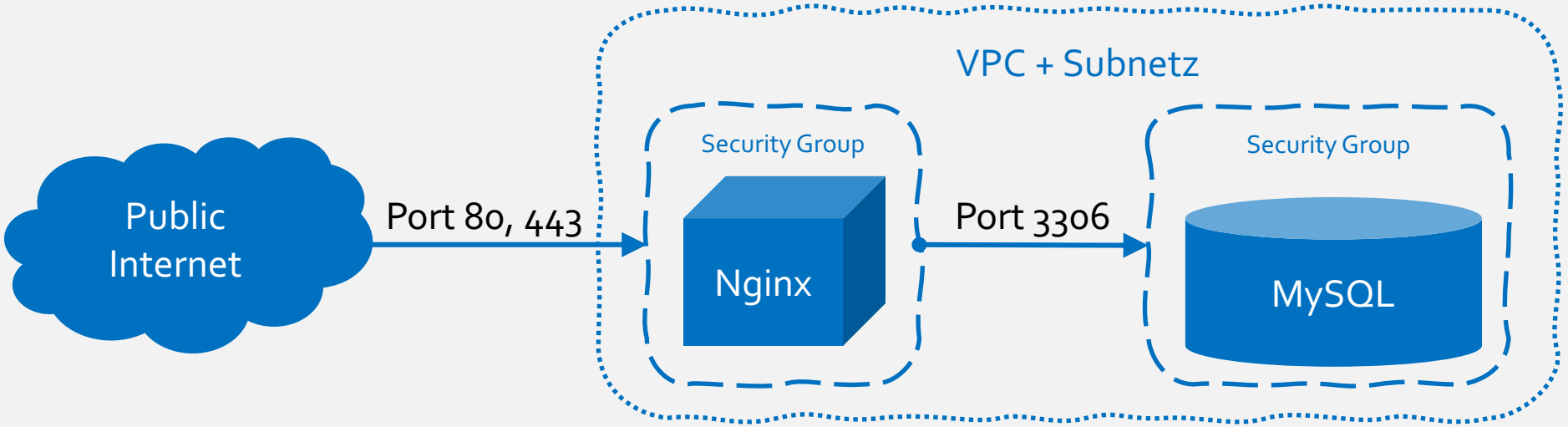

```
Terraform will perform the following actions:

  # kubernetes_pod.example will be updated in-place
  ~ resource "kubernetes_pod" "test" {
      id = "default/terraform-test"

      metadata {
          generation       = 0
          labels           = {
              "app" = "MyApp"
          }
          name             = "terraform-test"
          namespace        = "default"
          resource_version = "650"
          self_link        = "/api/v1/namespaces/default/pods/terraform-test"
          uid              = "5130ef35-7c09-11e9-be7c-080027f59de6"
      }

  ~ spec {
```

Vagrant ist gut für Single Node/VM Installationen (Städte).

Terraform „formiert ganze Planeten" (also Multi-Node Installationen).

TERRAFORM BEISPIEL

Provisionierung von Nginx und MySQL

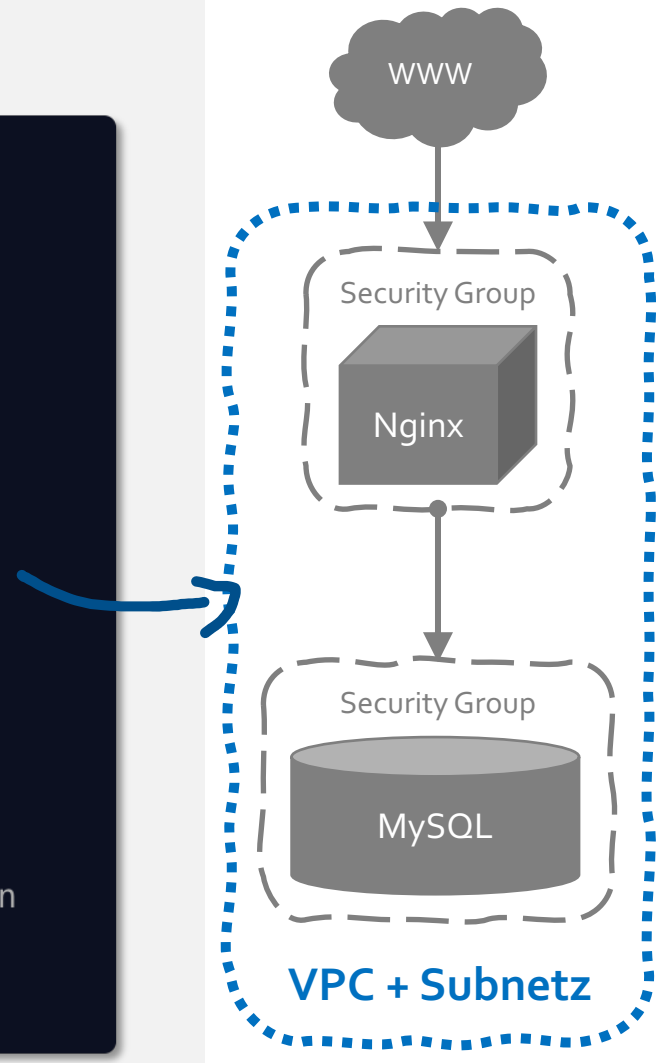


- Es sollen zwei AWS-Instanzen erzeugt werden.
- Auf einer Instanz soll Nginx installiert werden.
- Auf der anderen Instanz soll MySQL installiert werden.
- MySQL soll nur von der Nginx Instanz erreicht werden können.
- Nginx hingegen soll von außen und Port 80 und 443 erreichbar sein.
- Erforderliche Subnetz und SSH-Schlüsselpaare sollen ebenfalls erzeugt werden.

PROVISIONIERUNG VON NGINX UND MYSQL

1. Provider festlegen und VPC + Subnetz für das Deployment einrichten

```hcl
# AWS-Provider konfigurieren
provider "aws" {
  region = "us-east-1"  # Ändern Sie die Region nach Bedarf
}

# VPC erstellen
resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
  enable_dns_support = true
  enable_dns_hostnames = true
}

# Subnetz erstellen
resource "aws_subnet" "my_subnet" {
  vpc_id     = aws_vpc.my_vpc.id
  cidr_block = "10.0.1.0/24"        # Passen Sie die CIDR-Block-Adresse nach Bedarf an
  availability_zone = "us-east-1a"  # Wählen Sie die gewünschte Verfügbarkeitszone
}
```

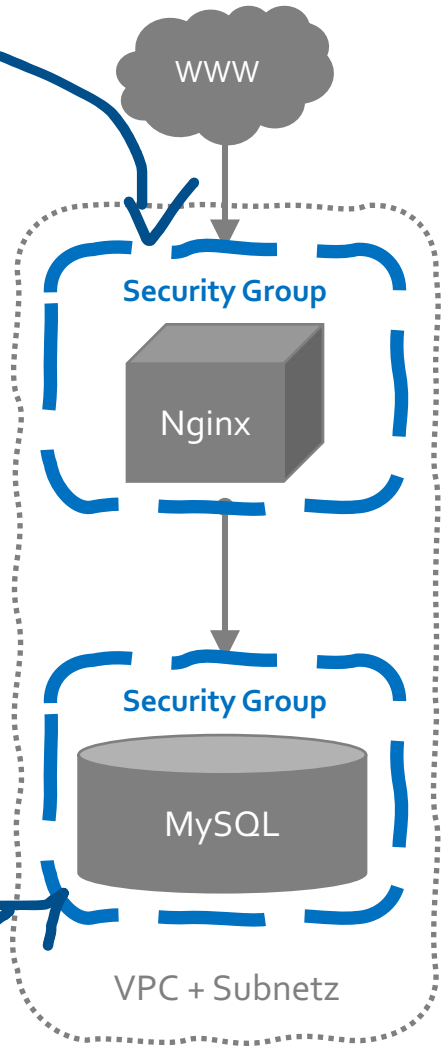


PROVISIONIERUNG VON NGINX UND MYSQL

2. Sicherheitsgruppen erstellen

```hcl
# Sicherheitsgruppe für Nginx
resource "aws_security_group" "nginx_sg" {
  name        = "nginx-sg"
  description = "Security group for Nginx"
  vpc_id      = aws_vpc.my_vpc.id

  # Inbound-Regeln für HTTP zulassen
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

```hcl
# Sicherheitsgruppe für MySQL
resource "aws_security_group" "mysql_sg" {
  name        = "mysql-sg"
  description = "Security group for MySQL"
  vpc_id      = aws_vpc.my_vpc.id

  # Inbound-Regel für MySQL (Port 3306)
  # von der Nginx-Instanz zulassen
  ingress {
    from_port   = 3306
    to_port     = 3306
    protocol    = "tcp"
    security_groups = [aws_security_group.nginx_sg.id]
  }
}
```



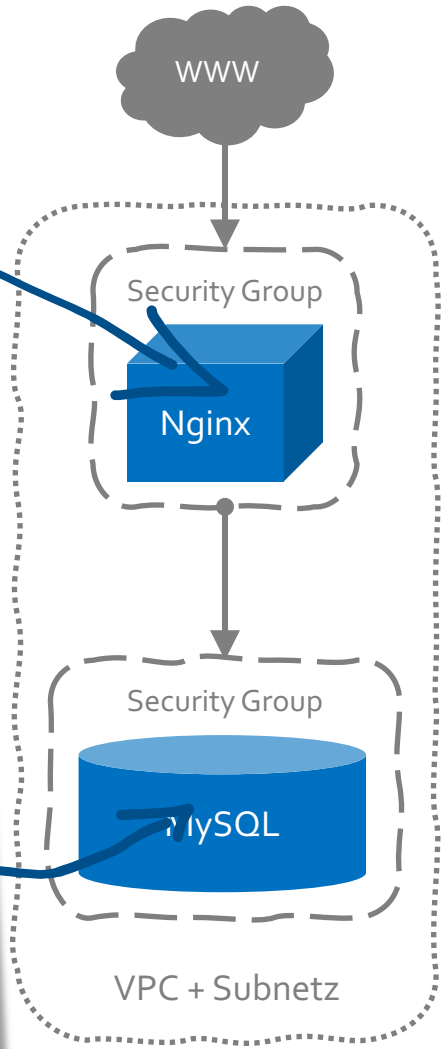
PROVISIONIERUNG VON NGINX UND MYSQL

3. Instanzen erstellen und mit SSH-Schlüssel versehen

```
# EC2-Instanz für Nginx erstellen
resource "aws_instance" "nginx_instance" {
  ami           = "ami-0c55b159cbfafe1f0"  # Amazon Linux 2 AMI
  instance_type = "t2.micro"               # Instanzgröße
  subnet_id     = aws_subnet.my_subnet.id
  key_name      = aws_key_pair.my_key_pair.key_name
  security_groups = [aws_security_group.nginx_sg.id]

  user_data = <<-EOF
              #!/bin/bash
              # Skript für die Installation von Nginx auf Amazon Linux 2
              sudo yum install -y nginx
              sudo systemctl start nginx
              sudo systemctl enable nginx
              EOF
}
```

```
# Schlüsselpaar erstellen
resource "aws_key_pair" "my_key_pair" {
  key_name   = "my-key-pair"
  public_key = file("~/.ssh/id_rsa.pub")  # Pfad zu Ihrem öffentlichen Schlüssel
}
```

```
# EC2-Instanz für MySQL erstellen
resource "aws_instance" "mysql_instance" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.my_subnet.id
  key_name      = aws_key_pair.my_key_pair.key_name
  security_groups = [aws_security_group.mysql_sg.id]

  user_data = <<-EOF
              #!/bin/bash
              # Skript für die Installation von MySQL
              sudo yum install -y mysql-server
              sudo systemctl start mysqld
              sudo systemctl enable mysqld
              EOF
}
```



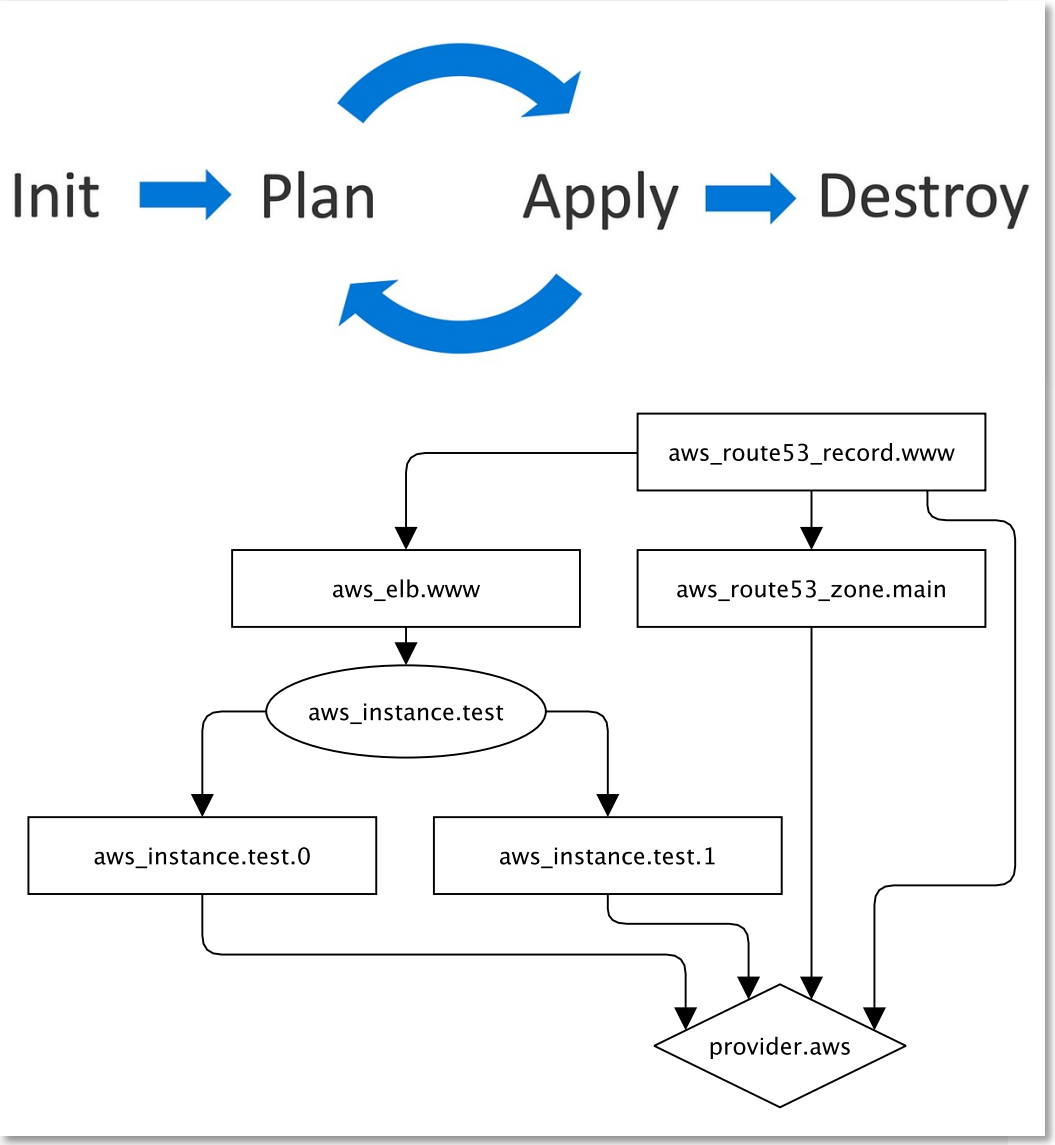
TERRAFORM

Deklarative Programmierung von Infrastrukturen

Write:
Beschreibung eines Zielzustands über eine domänenspezifische Sprache HCL (HashiCorp Configuration Langugage)

Plan (`terraform plan`):
Ist-Zustand ermitteln. Notwendige Änderungen planen (entsprechend Abhängigkeiten geordnet und parallelisiert)

Apply (`terraform apply`):
Idempotente Herstellung des Zielzustands. Der Zustand (.tfstate) Datei wird dabei lokal oder einem Remote Store (S3, HTTP, Postgres DB, ...) gespeichert.



Beispiel eines Terraform Plans (Directed Acyclic Graph, DAG) zur Ausbringung eines AWS-Infrastructure Deployments

TERRAFORM

Die Kern-Entitäten in Terraform Skripten

CLOUD NATIVE COMPUTING

Provider:
Schnittstelle zum Infrastruktur-Provider.
Hier werden üblicherweise die Access Credentials, Zugriffsprotokolle, sowie die Data Centers (Zonen) hinterlegt.

```
provider "google" {
  credentials = file(var.gce_credentials)
  project = var.gce_project
  region = var.gce_region
  zone = var.gce_zone
}
```

Data Sources:
Zugriff/bzw. Bekanntmachung von Informationen, die nicht durch Terraform selber in der Infrastruktur angelegt werden, sondern extern und global definiert werden/wurden (und nicht anpassbar sind).

- Netzwerknamen
- Datacenter Namen

```
data vsphere_network "private" {
  name          = "ei-vm-clients"
  datacenter_id = data.vsphere_datacenter.this.id
}

data vsphere_network "public" {
  name          = "dmz_ei_vm2"
  datacenter_id = data.vsphere_datacenter.this.id
}
```

Diese Entität wird meist in privaten Infrastrukturen benötigt, weniger in Public Cloud Infrastrukturen.

TERRAFORM

Die Kern-Entitäten in Terraform Skripten

CLOUD NATIVE COMPUTING

Ressourcen:

- Anzulegende Ressourcen in der Infrastruktur
- Diese sind Infrastruktur bzw. Provider-spezifisch

Beispiele (Azure):

- Virtual Machines
- Virtual Network (VNet)
- Blob Storage
- Azure SQL-Database
- Load Balancers
- Network Security Groups
- Azure DNS Zones

```
resource "google_compute_instance" "bootstrap" {
    name = "k8s-bootstrap"
    machine_type = var.bootstrap_machine_type

    metadata = {
        ssh-keys = "ubuntu:${file(var.ssh_key)}"
    }

    metadata_startup_script = file("resources/install.sh")

    boot_disk {
        initialize_params {
            image = "ubuntu-os-cloud/ubuntu-2004-lts"
        }
    }
```

TERRAFORM

Nur Versuch macht kluch ...

CLOUD NATIVE COMPUTING

Klonen Sie dieses Repository:

```
git clone https://git.mylab.th-luebeck.de/cloud-native/lab-iaas-iac.git
```

Kubernetes (MicroK8S)

```
> cd cluster
> terraform init
> terraform apply
```

Ergänzen Sie anschließend dies zur config.auto.tfvars

```
small_workers=2
xl_workers=2
```

um Ihren Cluster von einem Knoten auf fünf Knoten hochzuskalieren.

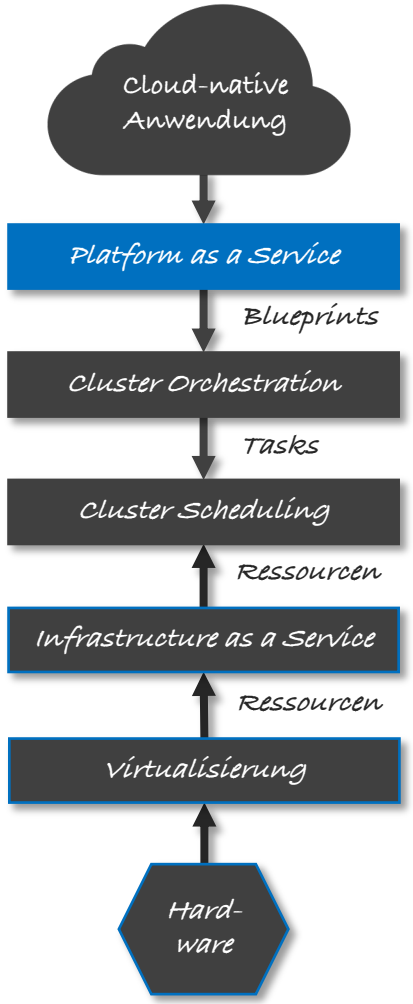
```
> terraform apply
```

Provisioning und Skalierung von Kubernetes Cluster in GCE.

AUSBLICK

Überblick über Units und Themen dieses Moduls

Unit 01 Cloud Computing	**Unit 02** DevOps	**Unit 03** Infrastructure as Code	**Unit 10** Domain Driven Design
Unit 04 Standardisierte Deployment Units	**Unit 05** Container Orchestrierung	**Unit 06** FaaS	
Unit 07 Microservice Architekturen	**Unit 08** Serverless Architekturen	**Unit 09** Monitorbare Architekturen	



KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de