Migration lokaler Dienste in die Cloud

Einführung in Cloud-Migrationsstrategien am Beispiel einer Quarkus-App

---

Prof. Dr. Thomas Schuster

Berufungsvortrag – THU, 2025

Technische Hochschule Ulm

Agenda

Lernziele

Nach der Einheit sind die Studierenden in der Lage, ...

- den Bedarf und die Vorteile einer Migration lokaler Systeme in die Cloud zu erläutern.
- typische Herausforderungen und Erfolgsfaktoren bei Cloud-Migrationen zu benennen.
- verschiedene Migrationsstrategien (z.B. Rehost, Replatform) zu unterscheiden.
- die grundlegenden Schritte einer Lift-and-Shift-Migration praktisch umzusetzen.
- Infrastruktur mit Terraform für Cloud-Deployments zu definieren.

Motivation

Kontext: Verteilte und webbasierte Systeme

Heute: **Migration** der bestehenden Anwendung in die Cloud

- Umsetzung: Infrastruktur-Provisionierung mit Terraform
- Analyse: Bewertung von Migrationsstrategien
- Ziel: Verständnis für Cloud-Betriebsmodelle

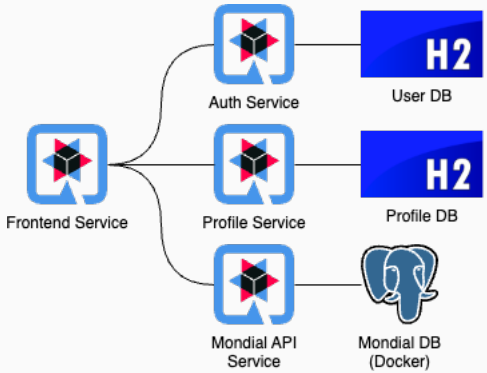


Abbildung 1: Unsere bisherige Microservice Entwicklung

Motivation für die Cloud-Migration

Warum sollten wir lokale Microservices in die Cloud migrieren?

- **Kosteneffizienz**: Weniger Fixkosten für Hardware und Betrieb
- **Skalierbarkeit**: Dynamische Anpassung an Lastspitzen
- **Verfügbarkeit**: Nutzung globaler Cloud-Infrastrukturen

Motivation für die Cloud-Migration

Warum sollten wir lokale Microservices in die Cloud migrieren?

- **Kosteneffizienz**: Weniger Fixkosten für Hardware und Betrieb
- **Skalierbarkeit**: Dynamische Anpassung an Lastspitzen
- **Verfügbarkeit**: Nutzung globaler Cloud-Infrastrukturen

Zusätzliche Vorteile:

- **Automatisierung**: Continuous Integration/Deployment vereinfacht
- **Sicherheit und Compliance**: Cloud-Anbieter bieten integrierte Lösungen
- **Innovationsgeschwindigkeit**: Schnellere Bereitstellung neuer Features

Migrationsstrategien

Die 7 *R* der Migration

Die 7 *R* nach Amazon Web Services (AWS, [1])

- Retire
- Retain
- Rehost (Lift & Shift)
- Relocate
- Repurchase
- Replatform
- Refactor (or re-architect)

Die 7 *R* der Migration

Die 7 *R* nach Amazon Web Services (AWS, [1])

- Retire
- Retain
- Rehost (Lift & Shift)
- Relocate
- Repurchase
- Replatform
- Refactor (or re-architect)

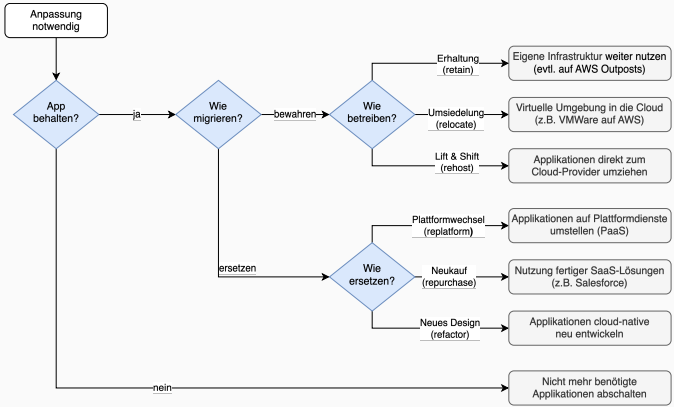


Abbildung 2: Vorgehensweise gemäß 7 *R* und [6]

Strategie im Fokus: Rehost (Lift & Shift)

Was ist Lift & Shift?

- **Verschieben** von **Anwendungen** ohne wesentliche Änderungen von On-Premise auf Cloud-Infrastruktur.

Warum damit starten?

- Häufiger Einstiegspunkt: Schnellster Weg in die Cloud
- Schnelle (vermeintliche) Erfolge: Basis-Skalierbarkeit/-Verfügbarkeit

Strategie im Fokus: Rehost (Lift & Shift)

Was ist Lift & Shift?

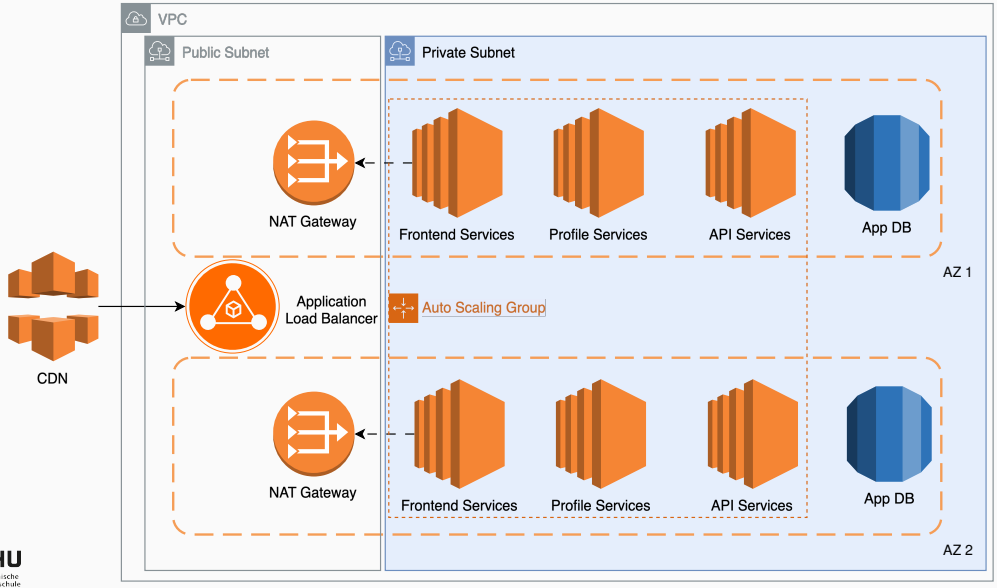
- **Verschieben** von **Anwendungen** ohne wesentliche Änderungen von On-Premise auf Cloud-Infrastruktur.
- Typische **Ziele**:
  - Virtuelle Maschinen (z.B. AWS EC2, Azure VM, GCP Compute Engine)
  - Datenbanken oft auf Managed Services (z.B. AWS RDS)

Warum damit starten?

- Häufiger Einstiegspunkt: Schnellster Weg in die Cloud
- Schnelle (vermeintliche) Erfolge: Basis-Skalierbarkeit/-Verfügbarkeit
- Geringer initialer Aufwand: Kaum Code-Änderungen nötig.

Anwendungsbeispiel

Architekturübersicht:Zielbild in der Cloud



Live-Demo: Lift & Shift

Was wir zeigen werden

- Unsere Anwendung (Quarkus Java App).
- Definition der Cloud-Infrastruktur mit **Terraform** (EC2, Security Groups).
- `terraform plan` und `terraform apply`: Infrastruktur erstellen.
- Deployment der Dienste auf (zunächst) zwei EC2 Instanzen.
- Zugriff auf die laufende Anwendung und die Datenbank.

» Let's roll ...«

Beispiel: Infrastructure as Code mit Terraform

```
provider "aws" {
    region = "eu-central-1" # Beispielregion Frankfurt
}

resource "aws_instance" "webserver" {
    ami           = "ami-0abcdef1234567890" # Beispiel Amazon Linux AMI ID
    instance_type = "t3.micro"              # Free Tier Instanzklasse
}
# Security Group und Netzwerk müssen auch definiert werden (hier ausgelassen)

resource "aws_instance" "app_server" {
    ami                    = data.aws_ami.ubuntu_24_04.id
    instance_type          = var.instance_type
    key_name               = var.key_name
    vpc_security_group_ids = [aws_security_group.app_sg.id]
    iam_instance_profile   = aws_iam_instance_profile.ec2_profile.name
```

Beispiel: Infrastructure as Code mit Terraform ii

```
17    # User Data Script zum Einrichten der Instanz
18    user_data = <<-EOF
19      #!/bin/bash
20      # Aktiviere detailliertes Logging aller Befehle
21      set -x
22
23      # Logdatei für dieses Skript
24      LOG_FILE="/var/log/user-data-setup.log"
25      ...
26
27      echo "Installiere notwendige Tools (docker.io, awscli, openjdk-21)..."
28      sudo apt-get install unzip docker.io openjdk-21-jdk -y
29      ...
```

Beispiel: Infrastructure as Code mit Terraform iii

```
30        sudo -u ubuntu nohup java -jar "${var.jar_s3_key}" > app.log 2>&1 &
31        if [ $? -ne 0 ]; then
32          echo "FEHLER: Start der Anwendung fehlgeschlagen!"
33          # exit 1
34        else
35          echo "Anwendung erfolgreich gestartet."
36        fi
37        echo "User Data Skript beendet am $(date)"
38        EOF
39  }
```

- Beschreibt den gewünschten Zustand der Infrastruktur.
- Terraform plant und führt die nötigen API-Aufrufe aus.

Technische & Wirtschaftliche Analyse

Analyse des Lift & Shift Ansatzes

- ⛔ Verpasste Cloud-Vorteile: Keine Nutzung höherwertiger Dienste (PaaS, Serverless, ...). App ist nicht optimiert für Cloud Umfeld.

Analyse des Lift & Shift Ansatzes

- **Verpasste Cloud-Vorteile**: Keine Nutzung höherwertiger Dienste (PaaS, Serverless, ...). App ist nicht optimiert für Cloud Umfeld.
- **Operativer Aufwand**: Security Hardening, Patching, Scaling, Backup-Konfiguration, Monitoring

Analyse des Lift & Shift Ansatzes

- **Verpasste Cloud-Vorteile**: Keine Nutzung höherwertiger Dienste (PaaS, Serverless, ...). App ist nicht optimiert für Cloud Umfeld.
- **Operativer Aufwand**: Security Hardening, Patching, Scaling, Backup-Konfiguration, Monitoring
- **Begrenzte Skalierbarkeit**: VM-Ebene (größere Instanz = vertikal; mehr Instanzen = horizontal). Keine feingranulare Skalierung wie bei Serverless.

Analyse des Lift & Shift Ansatzes

- **Verpasste Cloud-Vorteile**: Keine Nutzung höherwertiger Dienste (PaaS, Serverless, ...). App ist nicht optimiert für Cloud Umfeld.
- **Operativer Aufwand**: Security Hardening, Patching, Scaling, Backup-Konfiguration, Monitoring
- **Begrenzte Skalierbarkeit**: VM-Ebene (größere Instanz = vertikal; mehr Instanzen = horizontal). Keine feingranulare Skalierung wie bei Serverless.
- **Kosten**: Laufende VM (oft 24/7), bei Überprovisionierung (durch „Sicherheitszuschlag“) u.U. teurer als On-Prem.

Kostenmodell: Lift & Shift

Anwendung:

- 4 Quarkus Services:
  - frontend
  - profile
  - authentication
  - db-api
- 1 PostgreSQL Datenbank

Region: AWS ‘eu-central-1‘ (Frankfurt)

Lift & Shift Ansatz (Annahmen):

- 1 EC2 Instanz pro Service (4x t3.micro, Linux)

Kostenmodell: Lift & Shift

Anwendung:

- 4 Quarkus Services:
  - frontend
  - profile
  - authentication
  - db-api
- 1 PostgreSQL Datenbank

Region: AWS ‘eu-central-1‘ (Frankfurt)

Lift & Shift Ansatz (Annahmen):

- 1 EC2 Instanz pro Service (4x t3.micro, Linux)
- 1 Managed DB Instanz (RDS db.t3.micro, PostgreSQL, Single-AZ)

Kostenmodell: Lift & Shift

Anwendung:

- 4 Quarkus Services:
  - frontend
  - profile
  - authentication
  - db-api
- 1 PostgreSQL Datenbank

Region: AWS ‘eu-central-1‘ (Frankfurt)

Lift & Shift Ansatz (Annahmen):

- 1 EC2 Instanz pro Service (4x t3.micro, Linux)
- 1 Managed DB Instanz (RDS db.t3.micro, PostgreSQL, Single-AZ)
- Speicher: EBS gp3 (EC2), RDS Storage gp3

Kostenmodell: Lift & Shift

Anwendung:

- 4 Quarkus Services:
  - frontend
  - profile
  - authentication
  - db-api
- 1 PostgreSQL Datenbank

Region: AWS ‘eu-central-1‘ (Frankfurt)

Lift & Shift Ansatz (Annahmen):

- 1 EC2 Instanz pro Service (4x t3.micro, Linux)
- 1 Managed DB Instanz (RDS db.t3.micro, PostgreSQL, Single-AZ)
- Speicher: EBS gp3 (EC2), RDS Storage gp3
- Betrieb: 24/7 (730 Std./Monat)

Kostenmodell: Lift & Shift

Anwendung:

- 4 Quarkus Services:
  - frontend
  - profile
  - authentication
  - db-api
- 1 PostgreSQL Datenbank

Region: AWS ‘eu-central-1‘ (Frankfurt)

Lift & Shift Ansatz (Annahmen):

- 1 EC2 Instanz pro Service (4x t3.micro, Linux)
- 1 Managed DB Instanz (RDS db.t3.micro, PostgreSQL, Single-AZ)
- Speicher: EBS gp3 (EC2), RDS Storage gp3
- Betrieb: 24/7 (730 Std./Monat)
- Preismodell: On-Demand (flexibel, teurer); 100 GB Datentransfer im Free Tier

Kostenmodell: Komponenten & Kalkulation (Beispiel) i

Komponente	Spezifikation	Preis (ca.)
EC2 Compute	4×t3.micro (Linux, On-Demand)	28,24€
EBS Storage (EC2)	4×30GiB gp3	11,16€
RDS Compute	1×db.t3.micro (PostgreSQL, Single-AZ)	13,90€
RDS Storage	1×100GiB gp3	7,44€
Datenübertragung (Out)	100GB (nach 100GB Free Tier)	8,37€
Gesamt		**ca. 69,11€**

Tabelle 1: Geschätzte monatliche AWS-Kosten (Stand: April 2025)

Kostenmodell: Komponenten & Kalkulation (Beispiel) ii

Das ist eine Vereinfachte Schätzung

Preise der Cloud-Anbieter können sich ändern und wir haben einige vereinfachte Annahmen getroffen. Für genauere Kalkulationen am besten Kalkulator des Anbieters nutzen, bspw. **AWS Pricing Calculator** [2].

Der nächste Schritt: Replatform (”Lift & Reshape”)

- Idee: Anwendung mit *kleineren* Anpassungen auf Cloud-Plattformdiensten betreiben, um Cloud-Vorteile besser zu nutzen.
- Beispiele:
  - Anwendung auf PaaS (z.B. AWS Elastic Beanstalk, Azure App Service, Google App Engine).
  - Anwendung in Containern (z.B. AWS Fargate/ECS, Azure Container Instances, Google Cloud Run/GKE Autopilot).
  - Wechsel zu optimierten Datenbanken (z.B. AWS Aurora).
- Vorteile: Weniger Management-Aufwand (OS, Patches), bessere Skalierung und Integration mit Cloud-Services (Monitoring, Logging).
- Aufwand: **Moderat** - Anpassungen an der Anwendung oder Deployment-Prozess (Dockerfile erstellen) nötig.

Der Königsweg: Refactor / Re-Architect

- Idee: Grundlegend überarbeiten oder neu entwickeln, um *maximal* von Cloud-nativen Architekturen zu profitieren.
- Beispiele:
  - Monolithen in Microservices transformieren (haben wir schon)
  - Serverless-Funktionen nutzen (z.B. AWS Lambda) [5]
  - NoSQL Datenbanken nutzen (z.B. DynamoDB, Cosmos DB).
  - Event-gesteuerte Architekturen (z.B. SQS, Kinesis).
- Vorteile: Höchste Skalierbarkeit (feingranular, automatisch), Ausfallsicherheit (Design for Failure), Kostenoptimierung (Pay-per-Use), Agilität (Teams)
- Aufwand: **Hoch** - Signifikante Entwicklungsarbeit, neues Denken (verteilte Systeme, Eventual Consistency) erforderlich.

Zusammenfassung und Ausblick

Zusammenfassung

- Die Cloud-Migration bietet ein Spektrum an Strategien (die 7 R's).
- **Lift & Shift (Rehost)** ist ein häufiger, einfacher Startpunkt, aber **limitiert** in Bezug auf Cloud-Vorteile und kann Kosten-/Aufwandsfallen bergen.
- **Replatform** und **Refactor** ermöglichen eine tiefere Cloud-Integration und bessere Nutzung der Vorteile, erfordern aber mehr Aufwand.
- Die Wahl der Strategie ist kontextabhängig und basiert auf Zielen, Ressourcen und der Anwendung selbst [3].

Wichtige Werkzeuge und Praktiken

- **Infrastructure as Code (IaC)**
  - Definition der Infrastruktur in Code (z.B. Terraform, CloudFormation).
  - Wert: Reproduzierbarkeit, Versionierung, Automatisierung, weniger Fehler.
- **CI/CD Pipelines**
  - Automatisierte Builds, Tests und Deployments (z.B. GitHub Actions, Jenkins, GitLab CI, CodePipeline).
  - Wert: Schnellere Releases, höhere Qualität, konsistente Prozesse.
- **Monitoring & Logging**
  - Umfassende Überwachung (Metriken, Logs, Traces; z.B. CloudWatch, Datadog).
  - Wert: Performance-Optimierung, Fehleranalyse, Verständnis des Systems.
- **DevOps-Kultur**
  - Zusammenarbeit zwischen Entwicklung (Dev) und Betrieb (Ops).
  - Wert: Agilität, schnelle Feedbackzyklen, gemeinsame Verantwortung.

Lernkontrollfragen i

1. Nennen Sie drei wesentliche Vorteile, die Unternehmen durch eine Migration ihrer Dienste in die Cloud erwarten.
2. Erklären Sie den grundsätzlichen Unterschied zwischen den Migrationsstrategien „Rehost“ („Lift & Shift“) und „Replatform“.
3. Was versteht man unter „Lift & Shift“? Beschreiben Sie kurz, welche Cloud-Dienste typischerweise für eine einfache Anwendung bei diesem Ansatz genutzt werden.
4. Trotz der einfachen Umsetzung: Welche Nachteile oder Risiken hat der „Lift & Shift“-Ansatz oft im Vergleich zu Cloud-nativen Ansätzen? Nennen Sie mindestens zwei.

Lernkontrollfragen ii

5. Was ist der Hauptzweck von Infrastructure as Code (IaC), und welche Vorteile bietet der Einsatz von Werkzeugen wie Terraform dafür?
6. Für welche Art von Migrationsziel würden Sie eine „Refactor/Re-Architect“-Strategie in Betracht ziehen, und was sind die Hauptmerkmale dieses Ansatzes?
7. Warum ist kontinuierliches Monitoring im Cloud-Betrieb unerlässlich?
8. Beschreiben Sie das „Shared Responsibility Model“ [4] am Beispiel einer EC2-Instanz (Lift & Shift). Wofür ist der Cloud-Anbieter verantwortlich, wofür der Nutzer?
9. Welche Risiken sind mit der Abhängigkeit von einem Cloud-Anbieter verbunden?

Referenzen i

[1] AWS. ***About the Migration Strategies - AWS Prescriptive Guidance.*** URL: https://docs.aws.amazon.com/prescriptive-guidance/latest/large-migration-guide/migration-strategies.html (besucht am 27. 04. 2025).

[2] AWS. ***AWS Pricing Calculator.*** URL: https://calculator.aws/#/ (besucht am 27. 04. 2025).

[3] Simon Heinrich u. a. **„A Total Cost of Ownership Model for Cloud Computing Infrastructure“.** In: (3. Jan. 2023), S. 5779–5788. URL: https://hdl.handle.net/10125/103337 (besucht am 19. 04. 2025).

[4] ***Modell der geteilten Verantwortung – Amazon Web Services (AWS).*** Amazon Web Services, Inc. URL: https://aws.amazon.com/de/compliance/shared-responsibility-model/ (besucht am 15. 01. 2025).

Referenzen ii

[5] Quarkus. ***AWS Lambda Functions with Quarkus.*** URL: https://quarkus.io/guides/aws-lambda (besucht am 27. 04. 2025).

[6] ***Understanding the "7 Rs" - Cloud Migration Strategies.*** Community.aws. URL: https://community.aws/content/2cKbgI3WsAYTiDM0J48uEMVqOVW/understanding-the-7-rs (besucht am 27. 04. 2025).