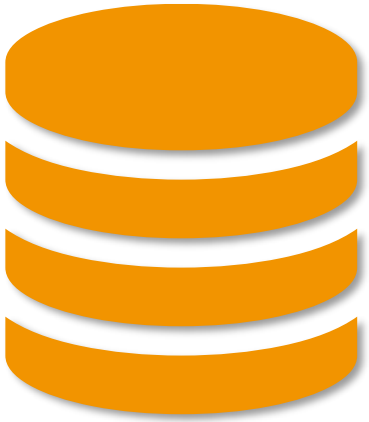

Ostbayerische Technische Hochschule
Amberg-Weiden

Cloud Grundlagen [EXT]

Hinweis

Ostbayerische Technische Hochschule
Amberg-Weiden

- Bitte beachten Sie das Urheberrecht!
- Alle Materialien dieser Vorlesung sind – auch wenn sie nicht ausdrücklich gekennzeichnet sind – urheberrechtlich geschützt.
- Sie dienen ausschließlich Ihrem persönlichen Gebrauch im Rahmen dieser Vorlesung.
- Die Materialien dürfen insbesondere nicht weiter verbreitet werden.
- Eigene Aufzeichnungen (Video, Foto, Ton) der Vorlesung sind nicht gestattet.

Ostbayerische Technische Hochschule
Amberg-Weiden

Cloud Grundlagen

Important Cloud Principles

- **Design for failure**
  - Hardware will fail
  - Outages will occur

  - **Strategies:**
    - Have a coherent backup and restore strategy for your data and automate it
    - Build process threads that resume on reboot
    - Allow the state of the system to re-sync by reloading messages from queues
    - Keep pre-configured and pre-optimized virtual images
      to support (2) and (3) on launch/boot
    - Avoid in-memory sessions or stateful user context, move that to data stores.

[https://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf]

Important Cloud Principles

Ostbayerische Technische Hochschule Amberg-Weiden

- **Decouple your components**
  - The more loosely coupled the components of the system, the bigger and better it scales
  - **Strategies:**
    - Decouple your components, by building asynchronous systems and scaling horizontally
    - Build a loosely coupled system using messaging queues
- **Implement elasticity**
  - Scale only when needed
  - **Strategies:**
    - Proactive Cyclic Scaling: Periodic scaling that occurs at fixed interval (daily, weekly, monthly, quarterly)
    - Proactive Event-based Scaling: Scaling just when you are expecting a big surge of traffic requests due to a scheduled business event (new product launch, marketing campaigns)
    - Auto-scaling based on demand. By using a monitoring service, your system can send triggers to take appropriate actions so that it scales up or down based on metrics (utilization of the servers or network i/o, for instance)

The 12-Factor App

By **Adam Wiggins**, **Heroku** cofounder, author of 12factor.net

Ostbayerische Technische Hochschule Amberg-Weiden

- I. Codebase
  - One codebase tracked in revision control, many deploys
  - 1:1 relationship between code repository and app
- II. Dependencies
  - Explicitly declare and isolate dependencies
  - Use packaging (gem, pip, maven,...)
- III. Config
  - Store config in the environment
  - Strict separation of config from code
- IV. Backing services
  - Treat backing services as attached resources
  - No code dependency to a resource instance
- V. Build, release, run
  - Strictly separate build and run stages
  - cf. Continuous Delivery
- VI. Processes
  - Execute the app as one or more stateless processes
  - Externalize state into data stores or databases (no sticky sessions)

[Adam Wiggins - https://12factor.net/]

The 12-Factor App

Ostbayerische Technische Hochschule Amberg-Weiden

- VII. Port binding
  - Export services via port binding
  - Self containment
- VIII. Concurrency
  - Scale out via the process model
- IX. Disposability
  - Maximize robustness with fast startup and graceful shutdown
- X. Dev/Prod parity
  - Keep development, staging, and production as similar as possible
  - cf. DevOps and Continuous Delivery
- XI. Logs
  - Treat logs as event streams
  - Centralize logs in a central system
- XII. Admin processes
  - Run admin/management tasks as one-off processes / scripts
  - e.g. no 12 page instruction for a database migration

Noch Fragen?