

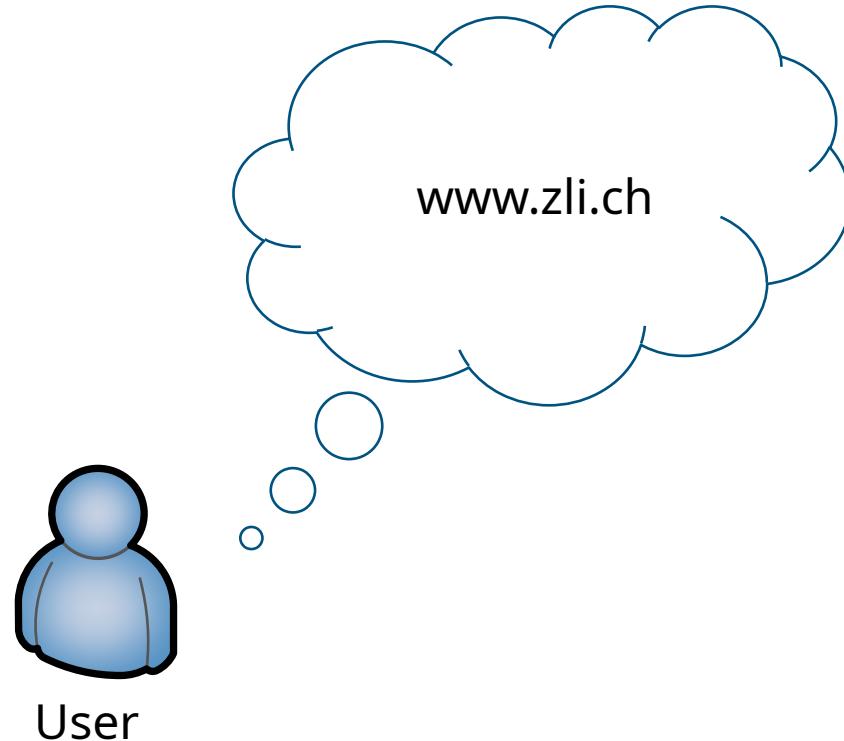
Public Cloud für Anwendungen nutzen

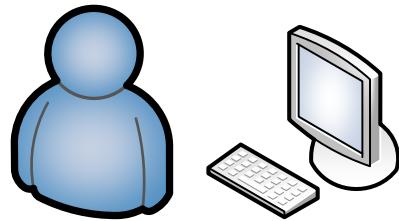
Modul 210

Block 1

Bereitstellung von Daten und Applikationen im Internet

Block 1



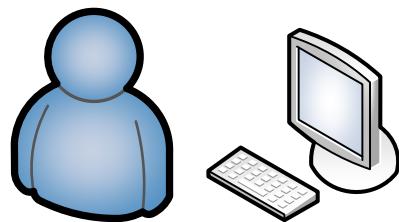


User

Client



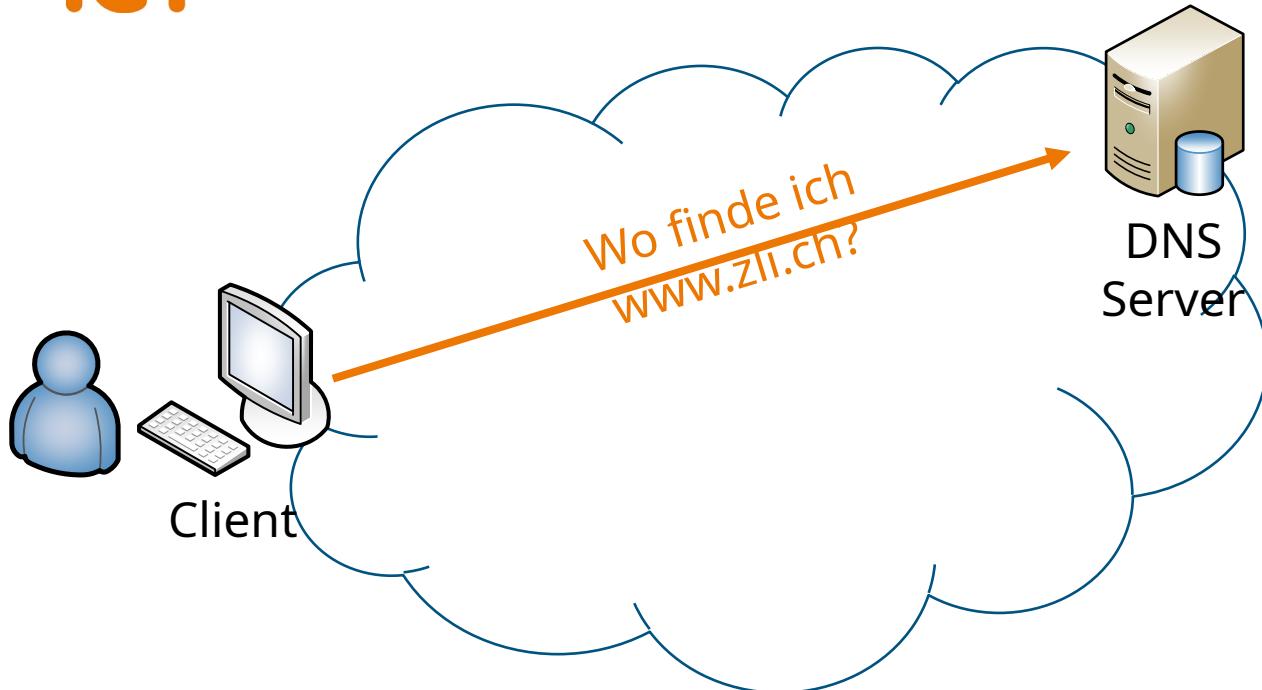
Browser

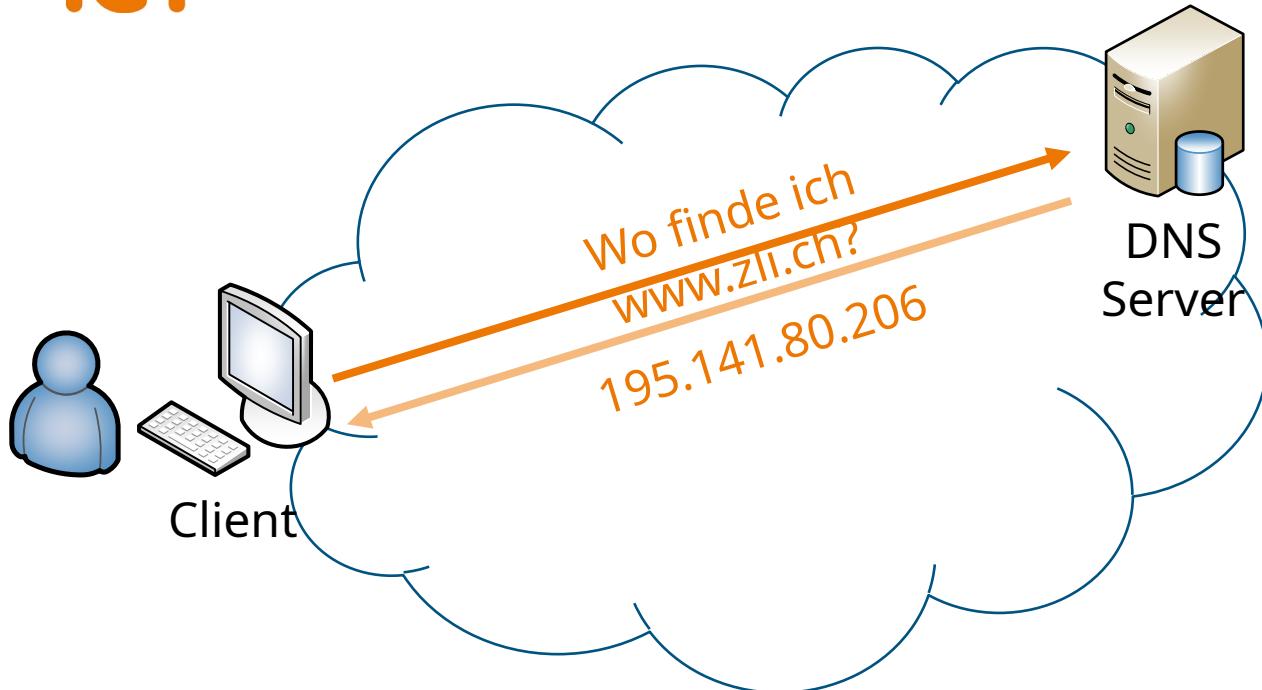


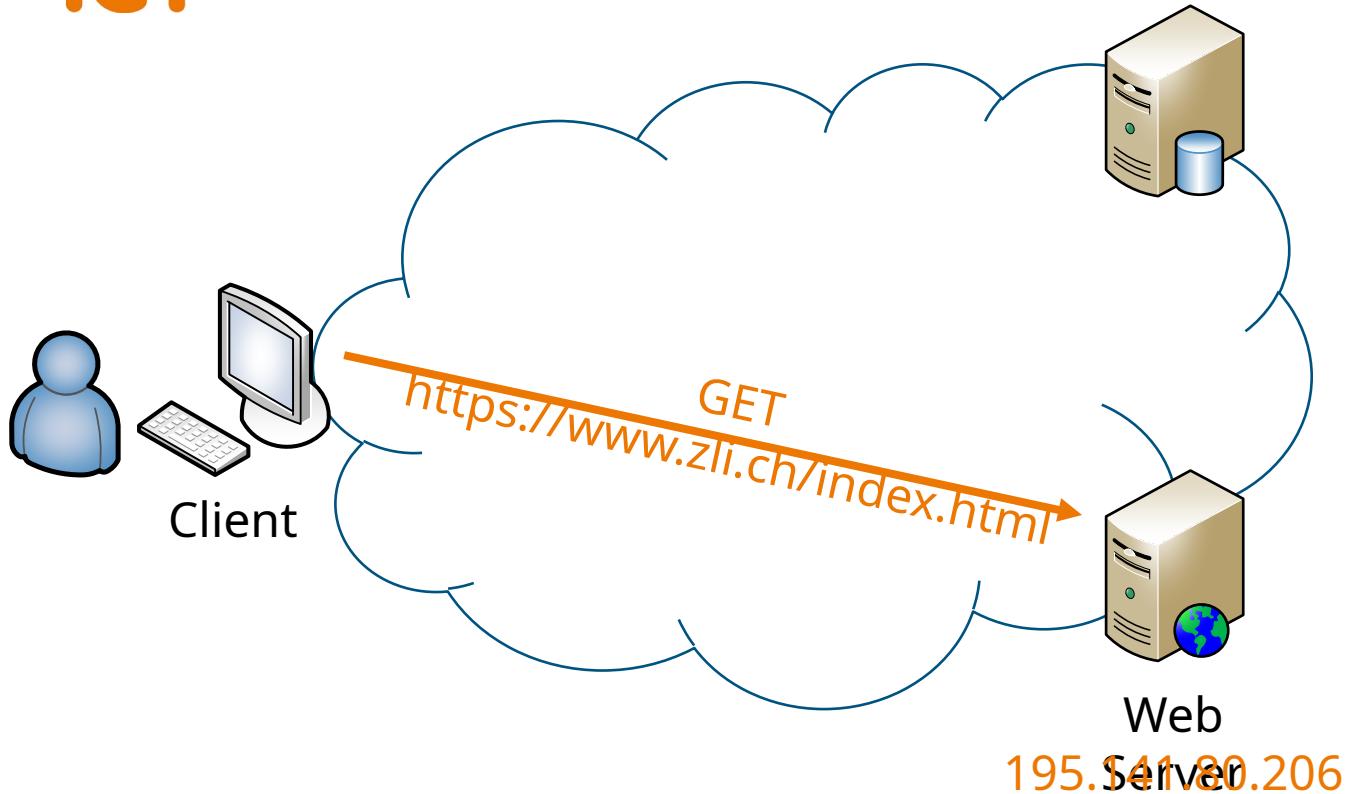
User

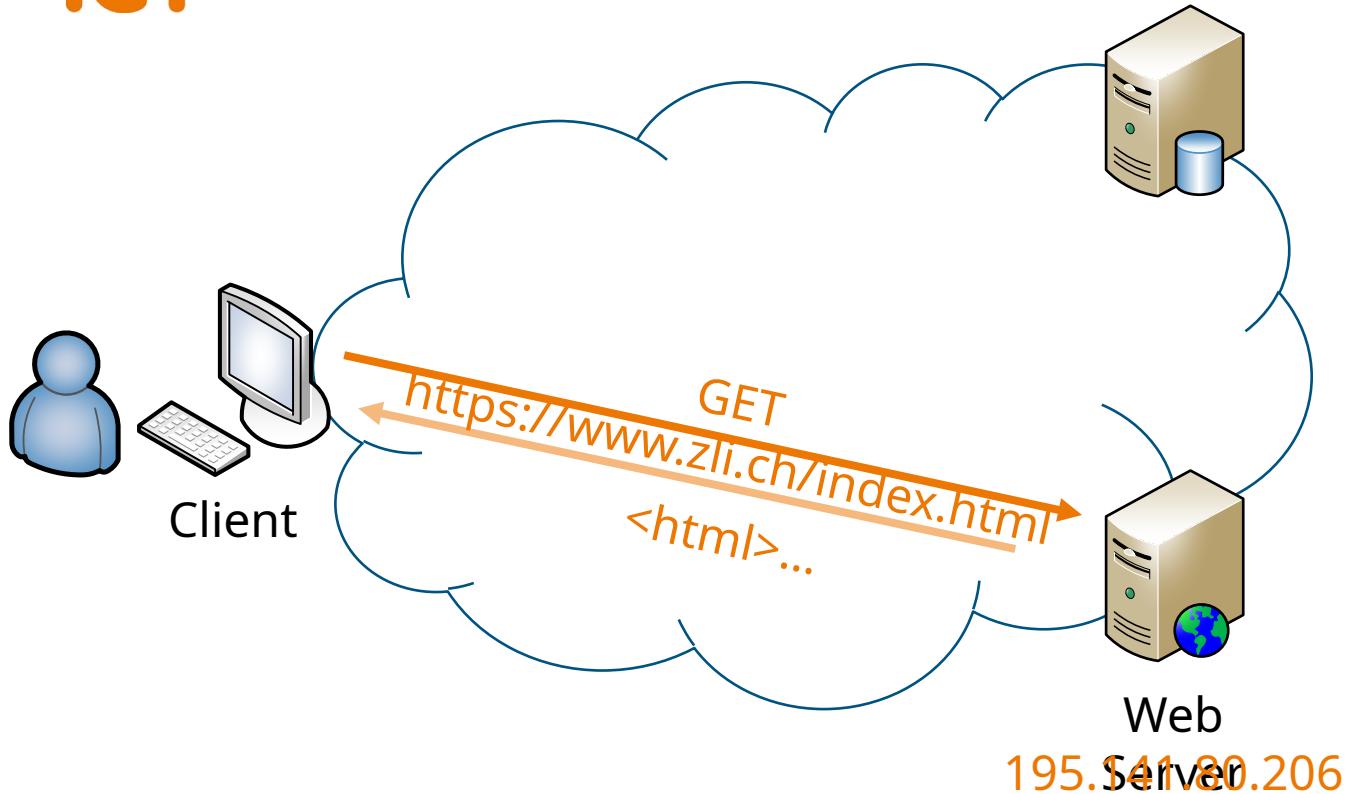
Client

HTTP Anfragen «Klassisch»

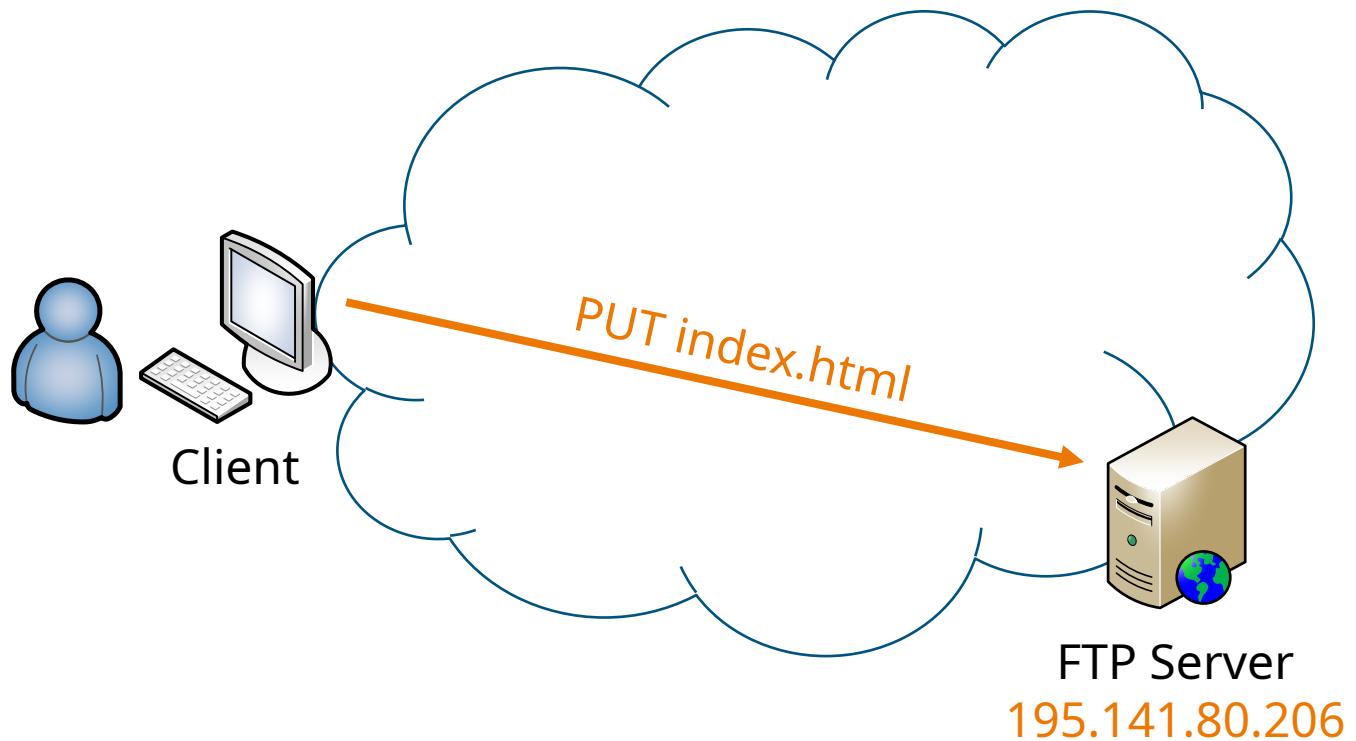








Deployment «Klassisch»

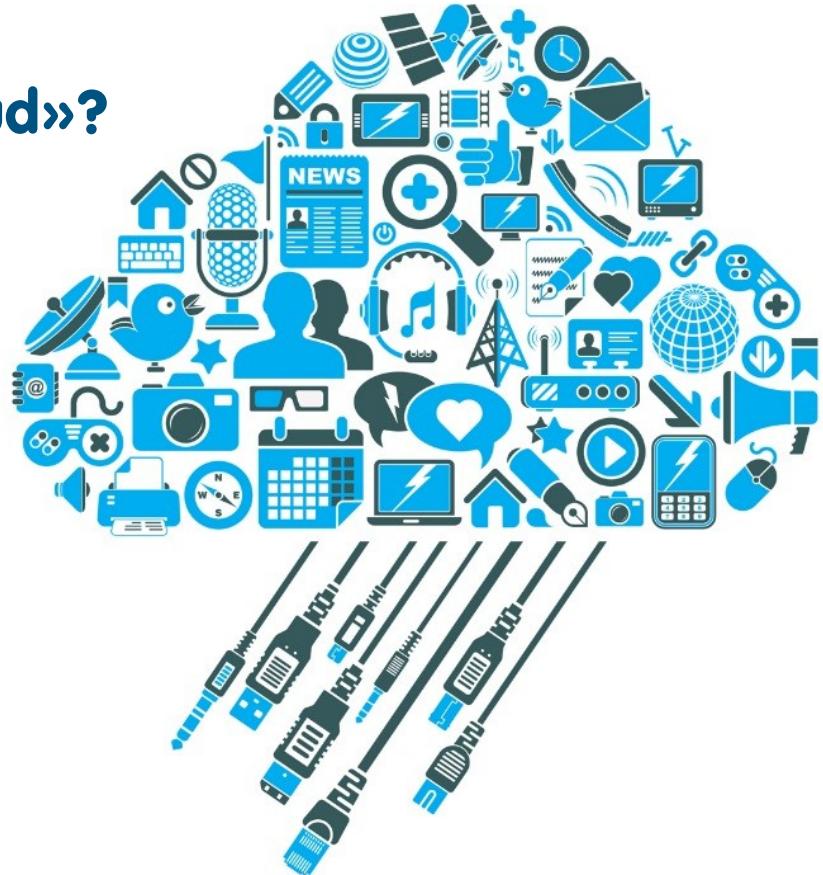


Begriffe und Definitionen

Block 1

Was versteht man unter «Cloud»?

Was versteht man unter «Cloud»?



Wie ist der Begriff entstanden? Wieso heisst es «Cloud»?

Wie ist der Begriff entstanden? Wieso heisst es «Cloud»?

- › Historischer wurde das Internet schon immer als «Wolke» dargestellt
- › Der Verbund von Computern besteht aus ganz vielen kleinen Teilen, wie bei einer Wolke. Ein einzelner Wassertropfen verschwindet im grossen Ganzen, aber gemeinsam ist es eine Wolke.



Ist der Begriff «Cloud» ein synonym für Internet?

- › Ja!
- › «It's just someone else's computer»
- › Cloud = Dienst/Service, die im Internet ausgeführt werden

Wie wird der Begriff definiert?

Wie wird der Begriff definiert?

Cloud Computing ist ein Modell zur Ermöglichung eines allgegenwärtigen, komfortablen und bedarfsgerechten Netzzugangs zu einem gemeinsam genutzten Pool konfigurierbarer Rechenressourcen (z. B. Netze, Server, Speicher, Anwendungen und Dienste), die mit minimalem Verwaltungsaufwand oder minimaler Interaktion mit dem Dienstanbieter schnell bereitgestellt und freigegeben werden können.

Dieses Cloud-Modell besteht aus
fünf wesentlichen Merkmalen,
drei Dienstmodellen
und vier Bereitstellungsmodellen.

Woran erkennt man eine Cloud?

Die fünf Merkmale nach NIST

Merkmale des Cloud Computing (1/5)

1. On demand self-service

Nutzung auf Abruf - Der Nutzer hat jederzeit Zugriff auf die Cloud-Dienste.

Merkmale des Cloud Computing (2/5)

2. Broad network access

Zugriff mit bekannten Technologien - Der Nutzer kann über Internetverbindung und entsprechende Clients auf die Dienste zugreifen.

Merkmale des Cloud Computing (3/5)

3. Ressource pooling

Zusammenlegung von Ressourcen - Die Serverkapazitäten werden gebündelt in einer einheitlichen Cloud zur Verfügung gestellt.

Economics of Scale

Merkmale des Cloud Computing (4/5)

4. Rapid elasticity

Ressourcenanpassung - Die zur Verfügung stehenden Ressourcen werden angepasst und es entsteht der Eindruck eines unbegrenzten Speicherplatzes.

Merkmale des Cloud Computing (5/5)

5. Measured service

Überwachung des Dienstes - Die einzelnen Cloud-Server werden immer wieder überwacht und optimiert.

Welche Dienstleistungen gibt es in der Cloud?

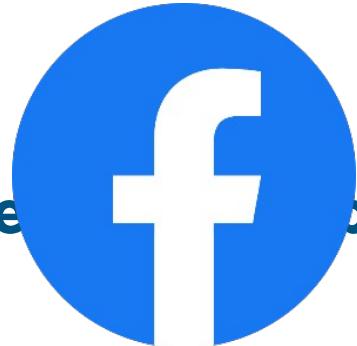
Welche Dienstleistungen gibt es in der Cloud?

- › E-Mail
- › Web-Server
- › Dropbox
- › iCloud
- › etc.



N

ne Dienstle



Google



Office

Wo wird die Infrastruktur für die Cloud betrieben? Die vier Deployment-Modelle

Deployment-Modelle: Public

Public
<p>In diesem Cloud-Computing-Modell stellt ein Drittpartei-Anbieter über das Internet der Öffentlichkeit Cloud-Computing-Ressourcen zur Verfügung. Mit der Public Cloud müssen Unternehmen nicht ihre eigenen Cloud-Server inhouse aufbauen oder warten.</p> <p>EIGENSCHAFTEN: Multi-Tenant-Architektur Nutzungsorientiertes Abrechnungsmodell.</p> <p>HAUPTANBIETER: AWS, Microsoft Azure, Google Cloud Platform</p>

- › Bereitstellung im **geteilten** Rechenzentrum
- › Keine Kontrolle über die Infrastruktur

Deployment-Modelle: Private

Private
Ein Cloud-Computing-Modell, mit dem ein Unternehmen proprietäre Architektur nutzt sowie Cloud-Server, die in ihrem eigenen Rechenzentrum laufen.
EIGENSCHAFTEN: Single-Tenant-Architektur On-Premises-Hardware Direkte Kontrolle über die zugrundeliegende Cloud-Infrastruktur
HAUPTANBIETER: HPE, VMware, Dell EMC, IBM, Red Hat, Microsoft, OpenStack

- › Bereitstellung im **eigenen** Rechenzentrum
- › Volle Kontrolle über die Infrastruktur

Deployment-Modelle: Hybrid

Hybrid

Ein Cloud-Computing-Modell, bestehend aus einem Mix an On-Premises, Private Cloud und Drittpartei-Public-Cloud-Services mit Orchestrierung zwischen den beiden Plattformen.

EIGENSCHAFTEN:

Cloud-Bursting-Fähigkeiten
Public- und Private-Cloud-Umgebungen

HAUPTANBIETER:

Mischung aus
Public- und Private-Cloud-Anbietern

- › Mischung aus Public- und Private-Cloud Anbietern

Deployment-Modelle: Community

- › Mehrere Firmen schliessen sich zusammen um gemeinsam eine Cloud zu betreiben

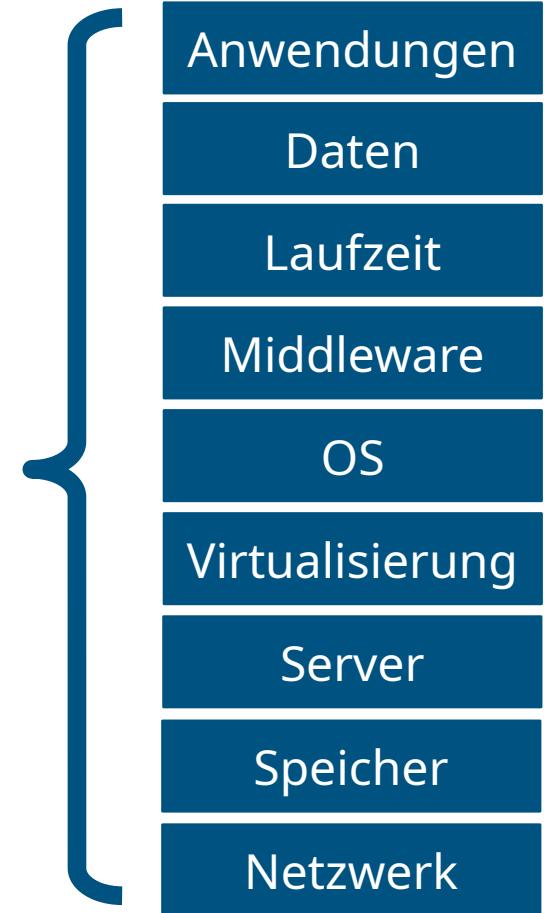
Was wird zur Verfügung gestellt?

Die vier Service-Modelle

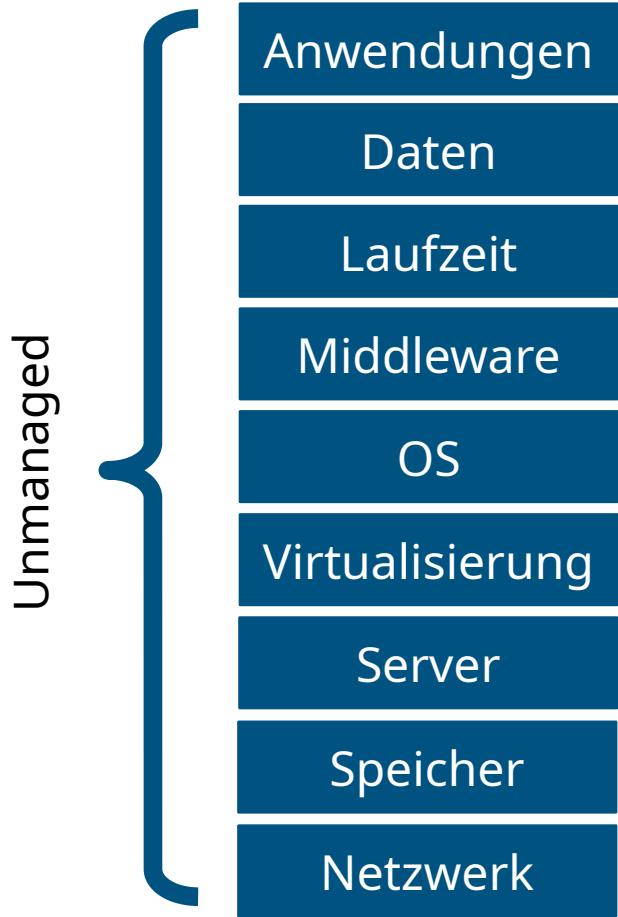
On Premises

- › Alles wird selbst gekauft, installiert, verwaltet und gewartet.
- › z.B.: Applikationen laufen auf den eigenen Servern im Keller.

Unmanaged

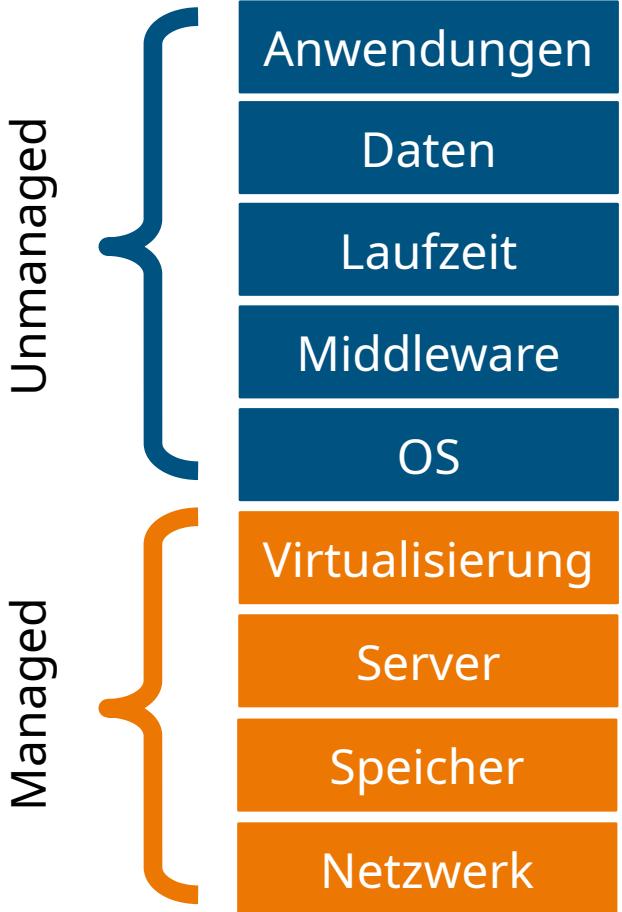


On Premise

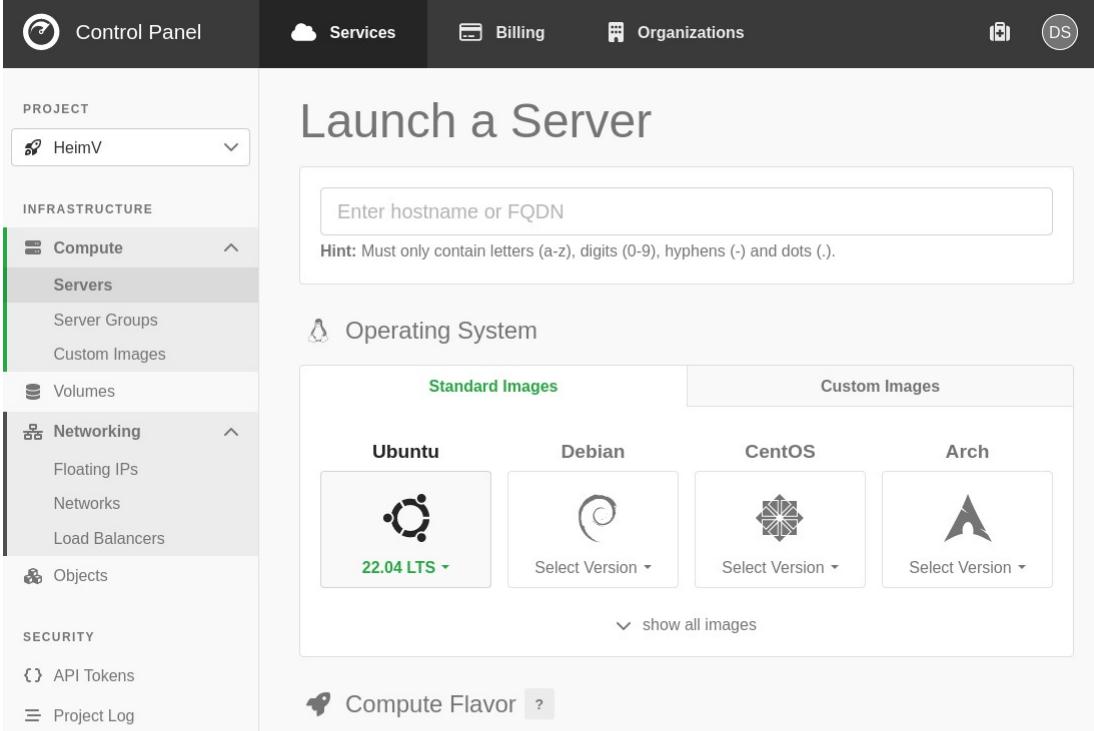


Service-Modelle: IaaS

- › Physische Server werden durch den Anbieter verwaltet und gewartet. Betriebssystem und Software werden selbst verwaltet.
- › z.B.: Applikationen laufen auf Virtual Private Server bei Hetzner, Nodes bei Cloudscale, Amazon E2C



Service-Modelle: IaaS



The screenshot shows a user interface for launching a server. On the left, a sidebar navigation includes: Control Panel, Services, Billing, Organizations, and a DS icon. The main area has tabs for PROJECT (HeimV), INFRASTRUCTURE (Compute, Servers, Server Groups, Custom Images, Volumes), Networking (Floating IPs, Networks, Load Balancers), and SECURITY (API Tokens, Project Log). The Compute tab is selected. The main content area is titled "Launch a Server". It features a search bar for "Enter hostname or FQDN" with a hint: "Hint: Must only contain letters (a-z), digits (0-9), hyphens (-) and dots (.)." Below this is a section for "Operating System" with tabs for "Standard Images" (Ubuntu 22.04 LTS, Debian, CentOS, Arch) and "Custom Images". At the bottom, there is a "Compute Flavor" section.

Unmanaged

Managed

Anwendungen

Daten

Laufzeit

Middleware

OS

Virtualisierung

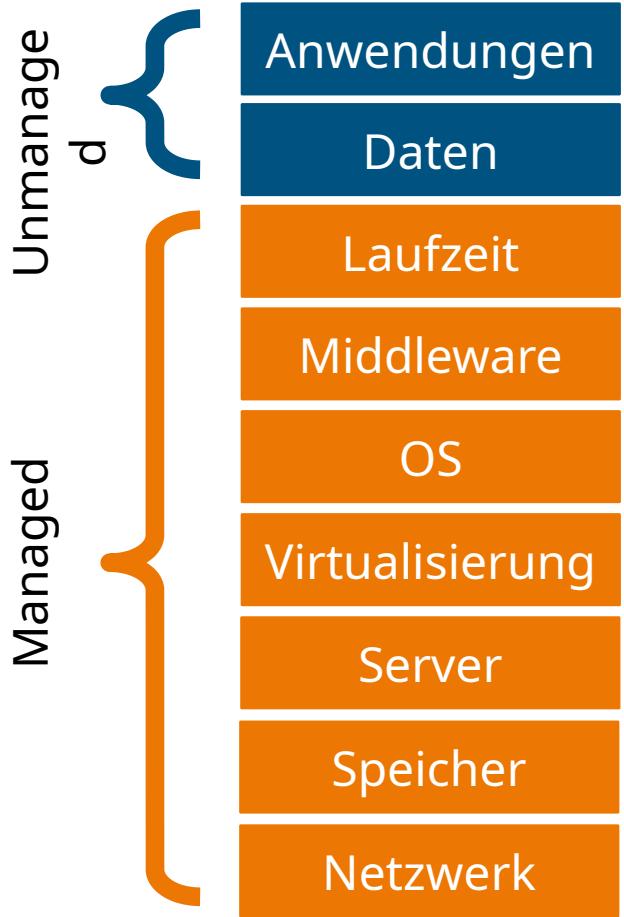
Server

Speicher

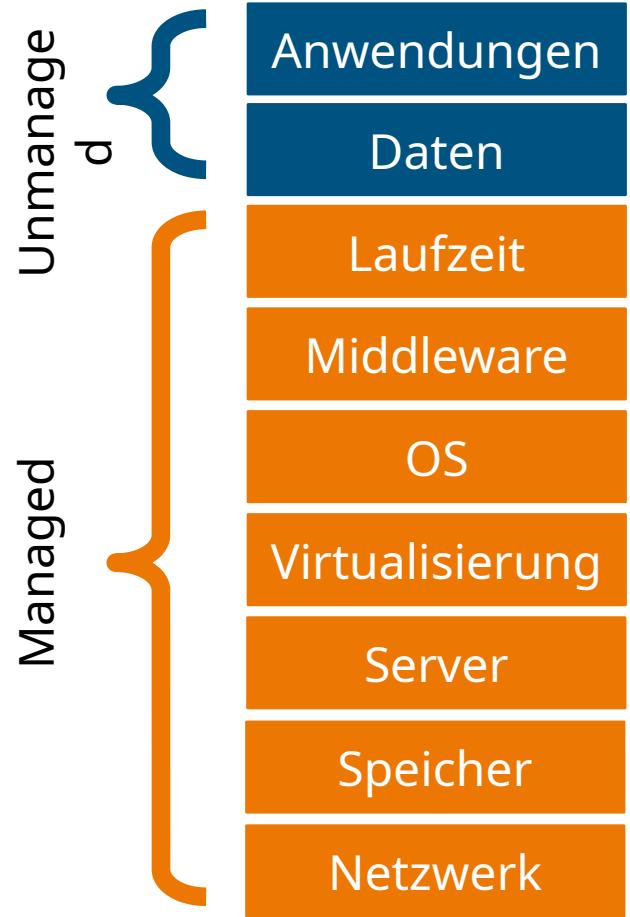
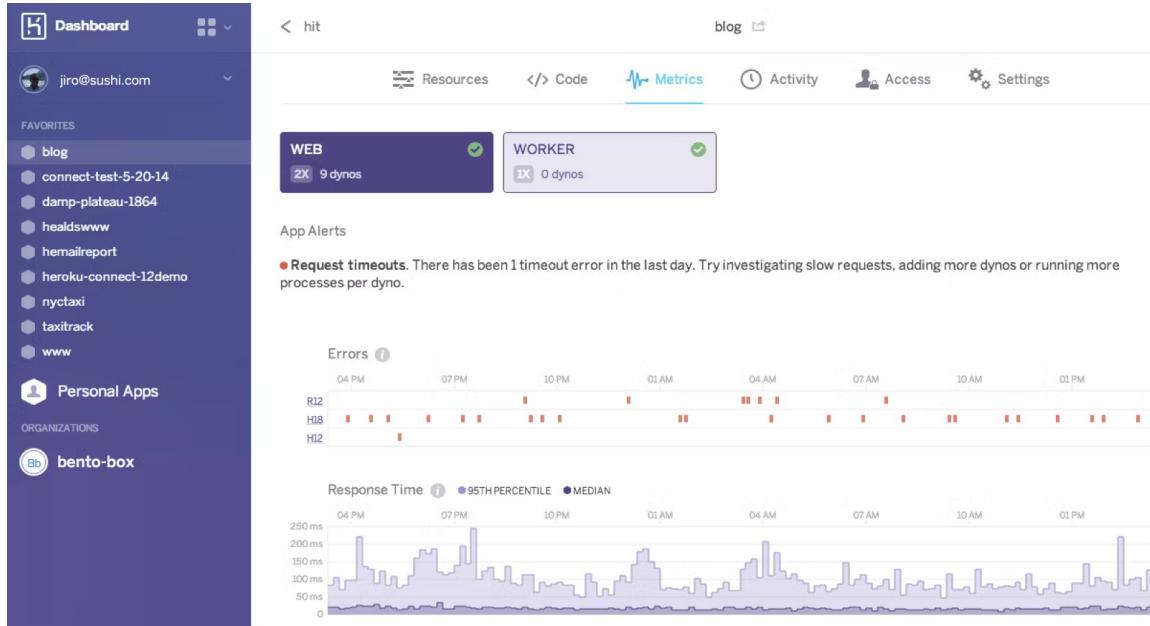
Netzwerk

Service-Modelle: PaaS

- › Laufzeitumgebung für Applikationen wird durch Anbieter zur Verfügung gestellt. Applikation wird selbst verwaltet
- › z.B.: Applikationen laufen auf Heroku

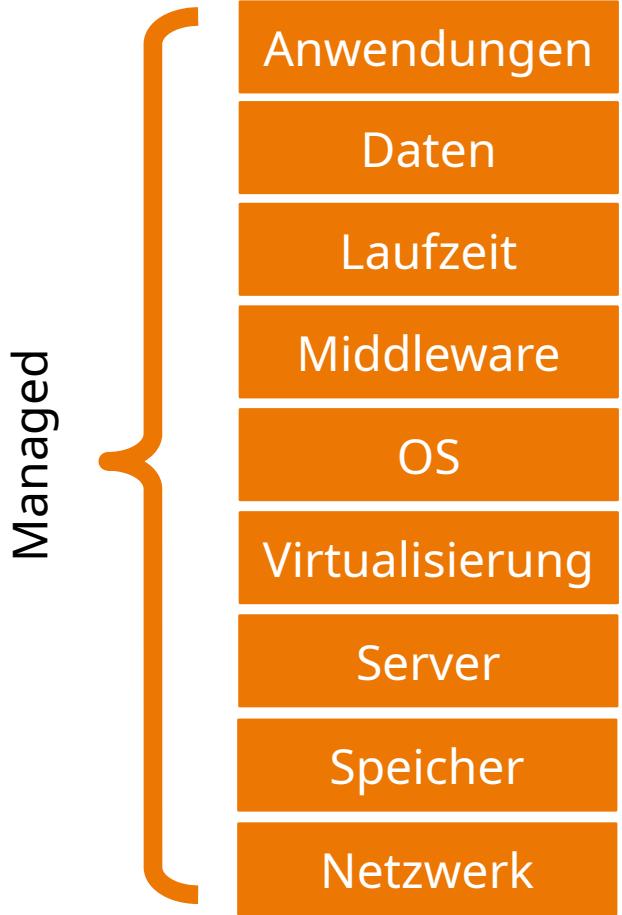


Service-Modelle: PaaS



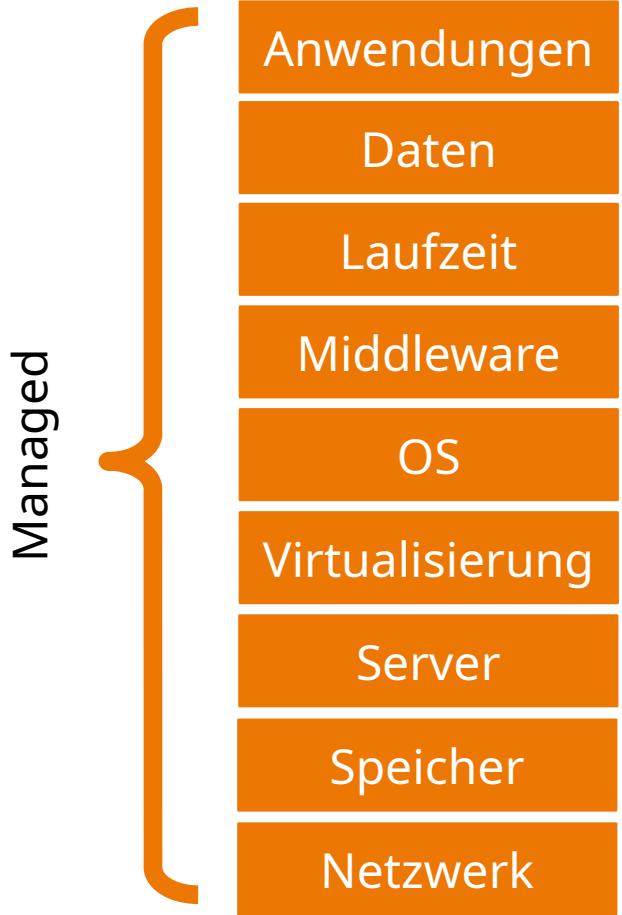
Service-Modelle: SaaS

- › Die gesamte Applikation wird durch den Anbieter zur Verfügung gestellt.
- › z.B.: Google Docs, Dropbox



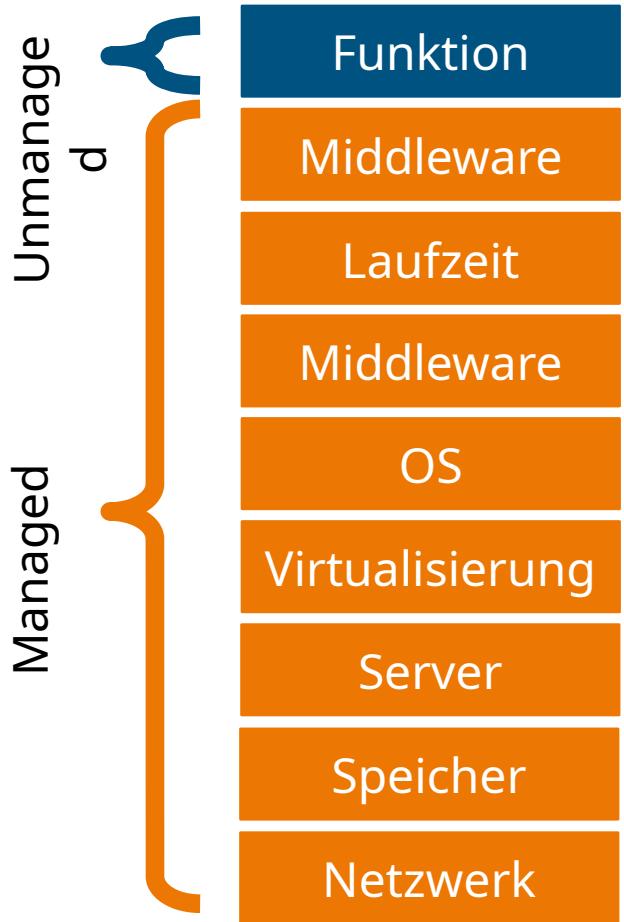
Service-Modelle: SaaS

- › Die gesamte Applikation wird durch den Anbieter zur Verfügung gestellt.
- › z.B.: Google Docs, Dropbox



*Service-Modelle: FaaS

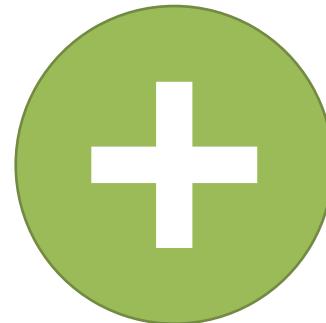
- › Laufzeitumgebung inkl. Middleware (z.B. HTTP Server) wird zur Verfügung gestellt.
Nur die eigentliche Funktion selbst wird selbst verwaltet
- › z.B.: Google Cloud Functions, Firebase



Welche Vorteile haben Cloud-Modelle?

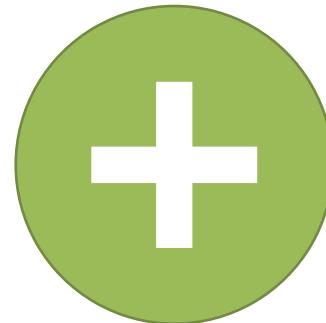
Vorteile von Cloud-Lösungen

- › Bezahlung nach Verbrauch
- › Geringe Gesamtkosten
- › Geringer Wartungsaufwand
- › Weltweiter Zugang



Vorteile von Cloud-Lösungen

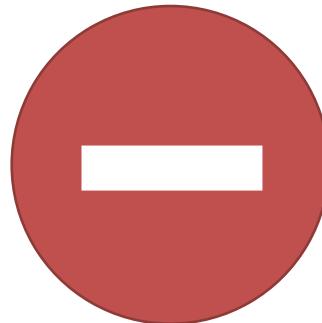
- › Garantierte Betriebskontinuität und Verfügbarkeit
- › Hohe Flexibilität und Elastizität
- › Kurze Markeinführungszeit



Welche Nachteile haben Cloud-Modelle?

Nachteile von Cloud-Lösungen

- › Bezahlung nach Verbrauch
- › Sehr hohe Komplexität
- › Weniger Kontrolle



Block 2

Cloud Anbieter

Block 2

Welche Cloud-Anbieter gibt es?

Welche Cloud-Anbieter gibt es?

- › Amazon AWS
- › Google Cloud Service GCS
- › Microsoft Azure
- › Alibaba Cloud
- › Cloudscale, Flow, ...

Präsentation zu Cloud-Anbieter

- › Firma: Mutterkonzern, Hauptsitz, Gründungsjahr?
- › Infrastruktur: Anzahl Regionen und Standorte, ev. Gamsamtanzahl Server?
- › Angebot: Wie viele Dienste, beliebteste Dienste?
- › Finanzen: Umsatz im letzten Jahr
- › Einfache VM mit 1GB RAM: Name des Produktes, Kosten pro Monat?

Cloud Anbieter: Amazon Web Services AWS



Gründung	2006
Sitz	Seattle USA
Umsatz	45 Mrd. USD
Dienste	> 200
Rechenzentren	77

Cloud Anbieter: Microsoft Azure



Gründung	2008
Sitz	Redmond USA
Umsatz	51 Mrd. USD
Dienste	> 200
Rechenzentren	> 160

Cloud Anbieter: Alibaba Cloud



Gründung	2009
Sitz	Hangzhou CN
Umsatz	9 Mrd. USD
Dienste	
Rechenzentren	24

Cloud Anbieter: Google Cloud Platform



Gründung	2008
Sitz	Mountain View USA
Umsatz	12 Mrd. USD
Dienste	
Rechenzentren	24

Cloud Anbieter: Cloudscale



Gründung	2008
Sitz	Zürich
Umsatz	?
Dienste	
Rechenzentren	2

Welche Cloud-Anbieter gibt es?

- › Demo Amazon, Cloudscale, Flow

Welche Cloud Dienste betreibt Ihre Firma selbst?

Welche Cloud Dienste betreibt Ihre Firma selbst?

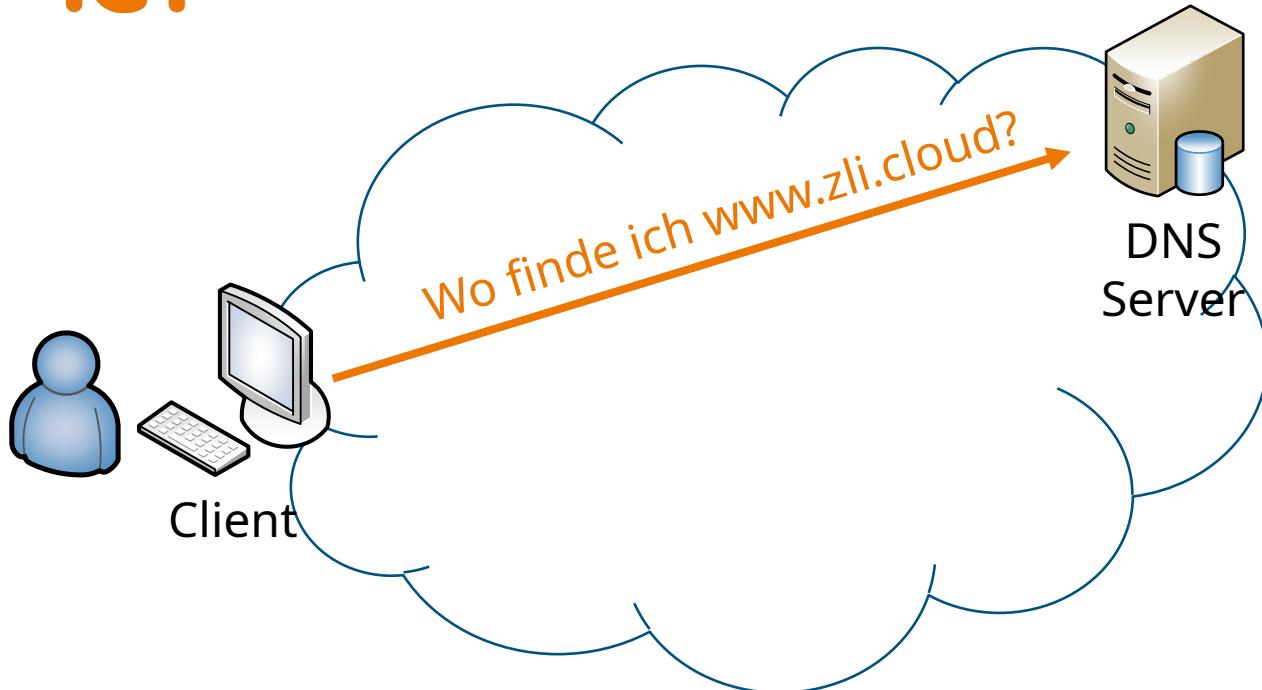
- >
- >
- >

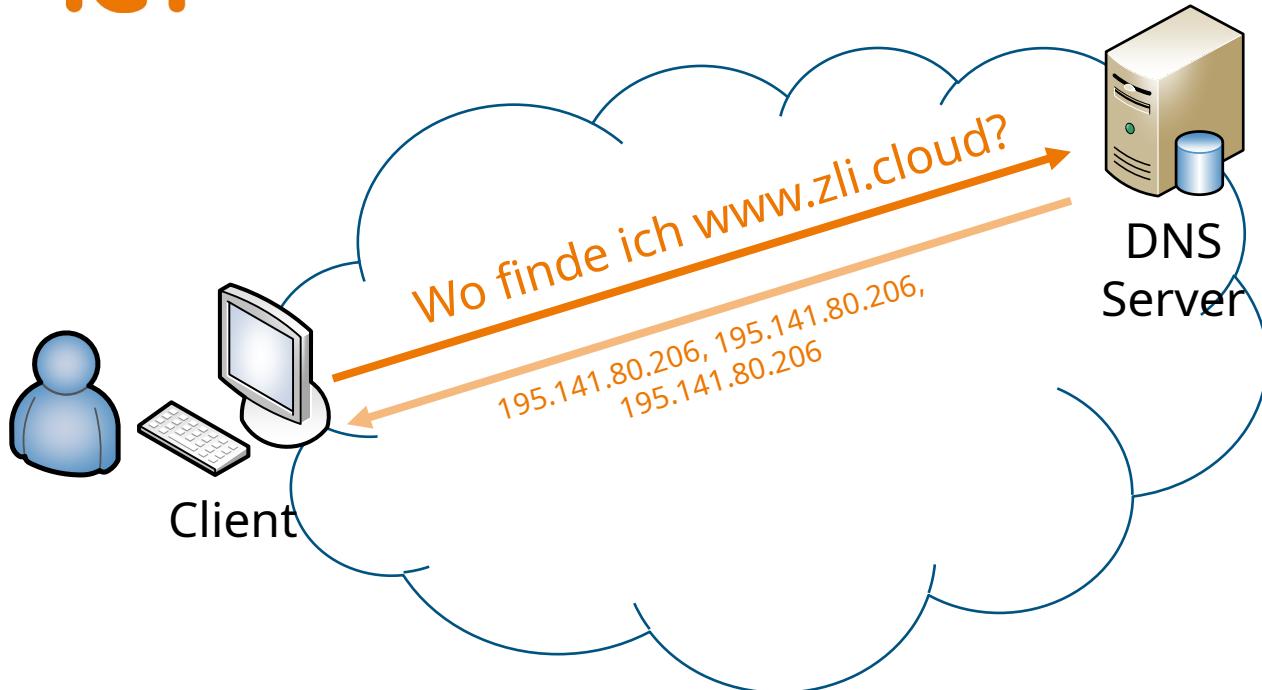
Applikationen deployen mit git

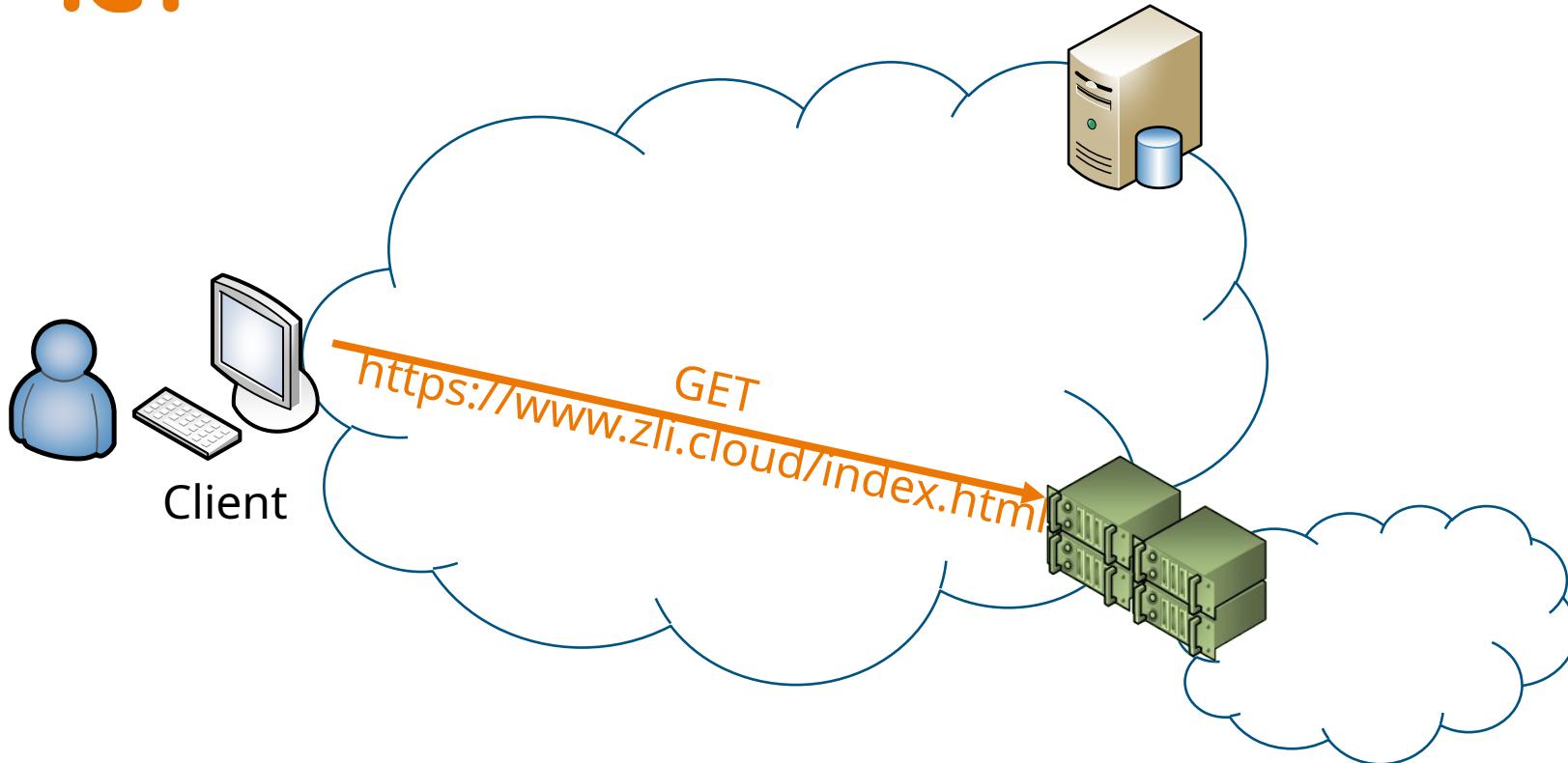
Block 2

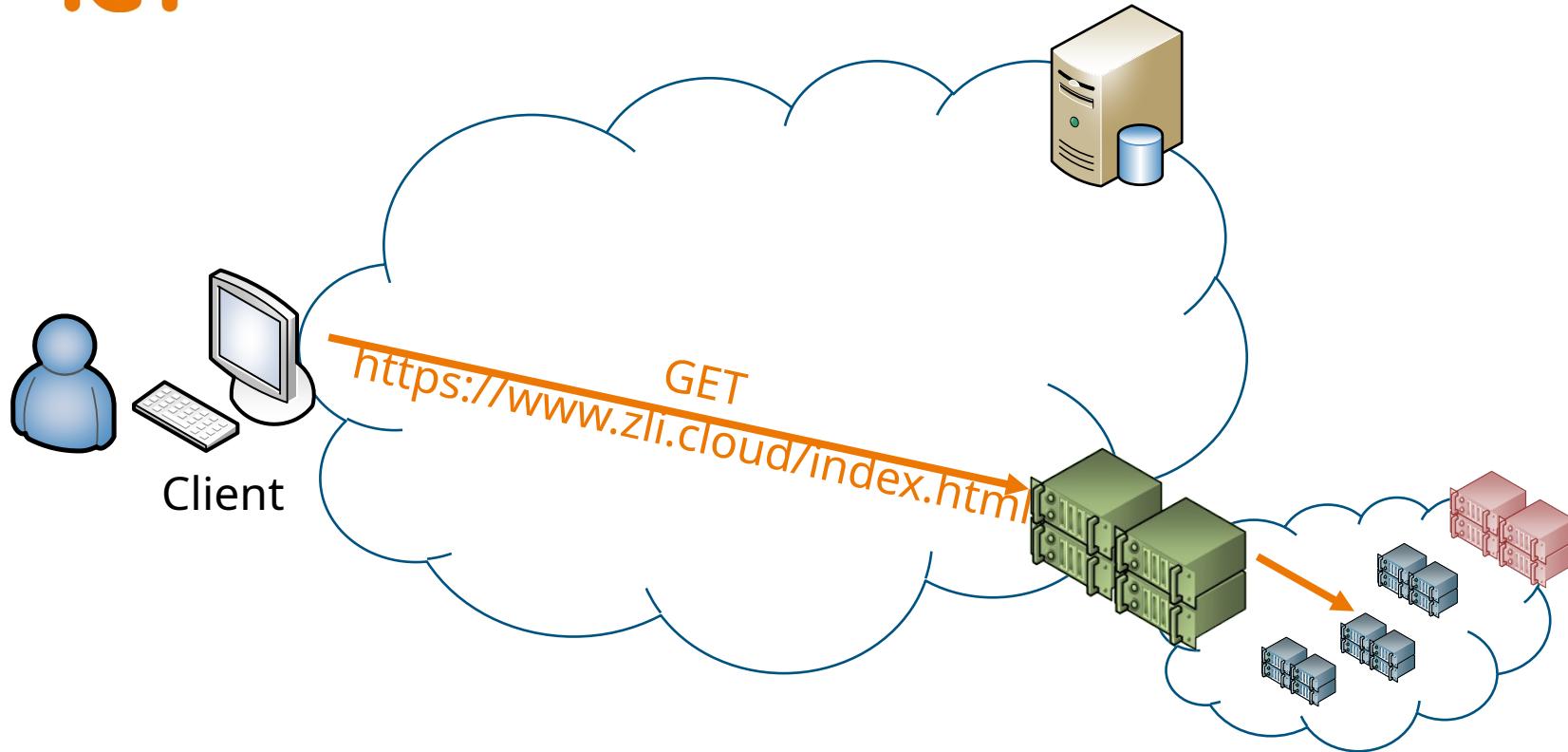
Wie kommt eine Applikation in die Cloud?

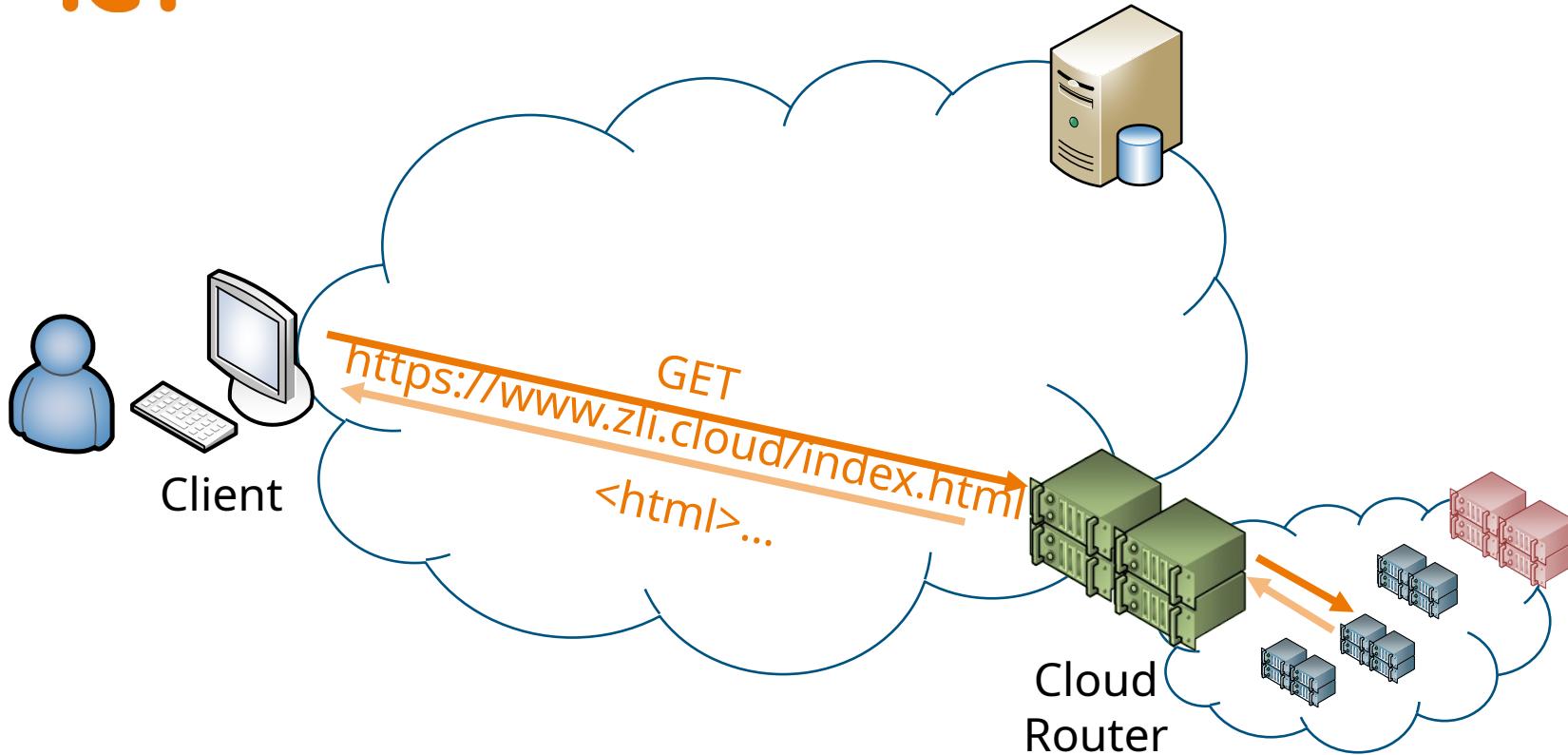
HTTP Anfragen «Cloud»



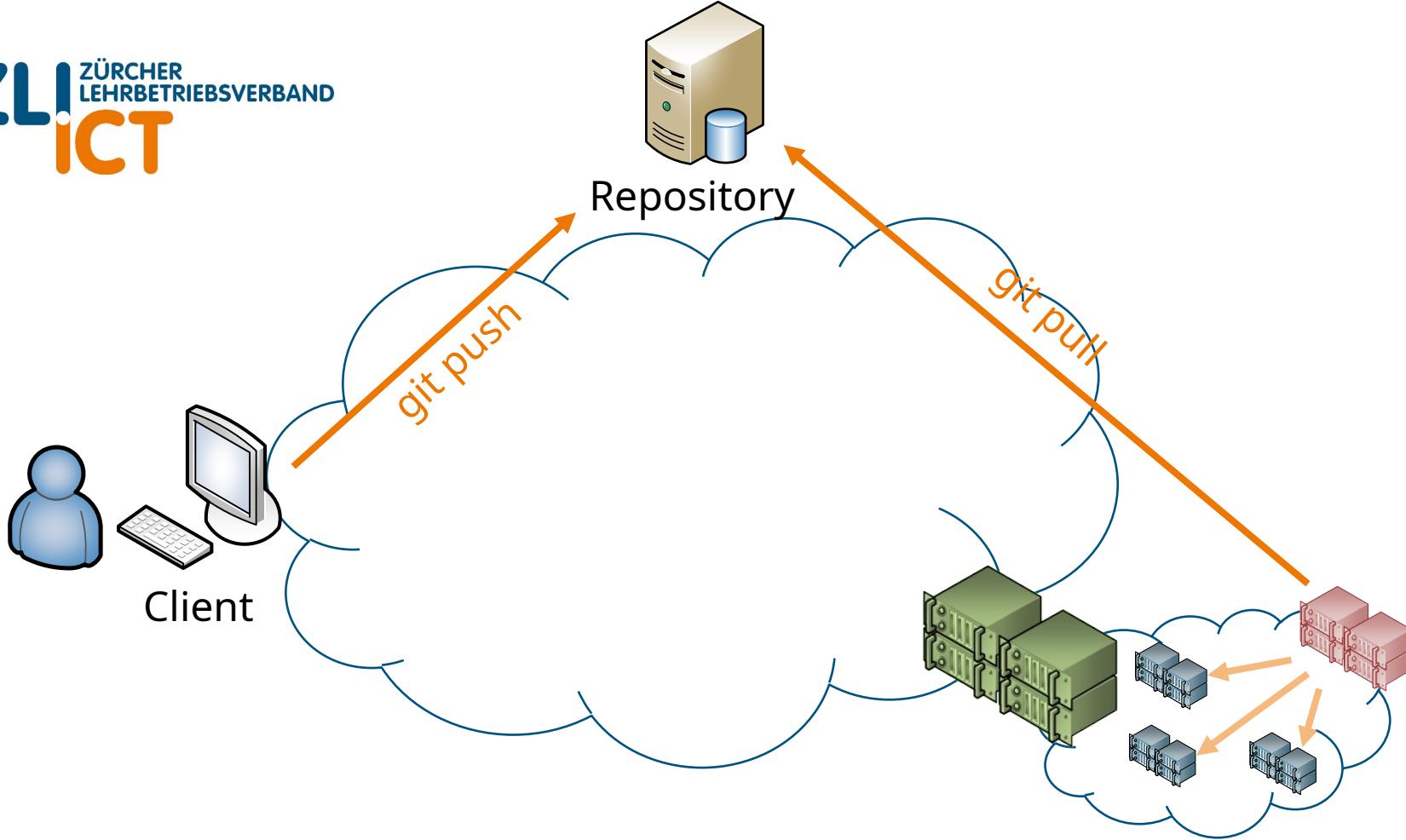








Deployment «Cloud»



**Wie werden technologische Beiträge in der Cloud geteilt
bzw. zur Verfügung gestellt? (1/3)**

- › Code Repositories
 - z.B. GitHub, GitLab, usw.

GitHub

 Bitbucket



GitLab



Azure DevOps

**Wie werden technologische Beiträge in der Cloud geteilt
bzw. zur Verfügung gestellt? (2/3)**

- › Binary-/ Container Repositories
z.B. dockerhub



Wie werden technologische Beiträge in der Cloud geteilt
bzw. zur Verfügung gestellt? (3/3)

- › Anbieterspezifische Marketplaces



Block 3

Einführung in die Container Technologie

Block 3

Was ist Container-Technologie oder Container-Virtualisierung?

- › Eine standardisierte Software-Einheit.
- › Die Container-Virtualisierung erlaubt es, eine Anwendung unabhängig von einem Host-Betriebssystem zu betreiben
- › Im Container verpackt ist also eine komplette Anwendung inklusive Code, Abhängigkeiten und Konfigurationen in einem klar festgelegten Format.

Was sind die Vor – und Nachteile der Technologien?

Virtuelle Server	Container
Prozessisolierung auf Hardwareebene	Prozessisolierung auf Betriebssystemebene
VM bietet vollständige Isolierung der Anwendungen vom Hostbetriebssystem	Container können gewisse Ressourcen mit dem Host-Betriebssystem teilen
Jeder virtuelle Server hat ein eigenes Betriebssystem	Jeder Container kann Betriebssystem-Ressourcen gemeinsam nutzen
Hochfahren in Minuten	Hochfahren in Sekunden
Mehr Ressourcenverbrauch	Geringerer Ressourcenverbrauch
Vorkonfigurierte VMs sind schwer zu finden und zu verwalten	Vorgefertigte Container für Anwendungen bereits verfügbar

Was sind die Vor – und Nachteile der Technologien?

Virtuelle Server	Container
Die Anpassung von vorkonfigurierten VMs erfordert Arbeit	Benutzerdefiniertes Setup mit Containern ist einfach zu erstellen
VMs sind in der Regel grösser, da sie ein vollständiges Betriebssystem enthalten.	Container sind klein und haben nur eine Container-Engine auf dem Host-Betriebssystem
VMs können leicht auf einen neuen Host verschoben werden	Container werden zerstört und neu erstellt, anstatt sie zu verschieben
Die Erstellung von VMs nimmt relativ viel Zeit in Anspruch	Container können in Sekundenschnelle erstellt werden
Virtualisierte Apps sind schwieriger zu finden und es dauert länger, sie zu installieren und auszuführen.	Containerisierte Anwendungen können innerhalb weniger Minuten gefunden und installiert werden

**Welche Produkte kennen Sie im Zusammenhang mit
virtuellen Servern und Container?**

Welche Produkte kennen Sie im Zusammenhang mit virtuellen Servern und Container?

Virtual Machines



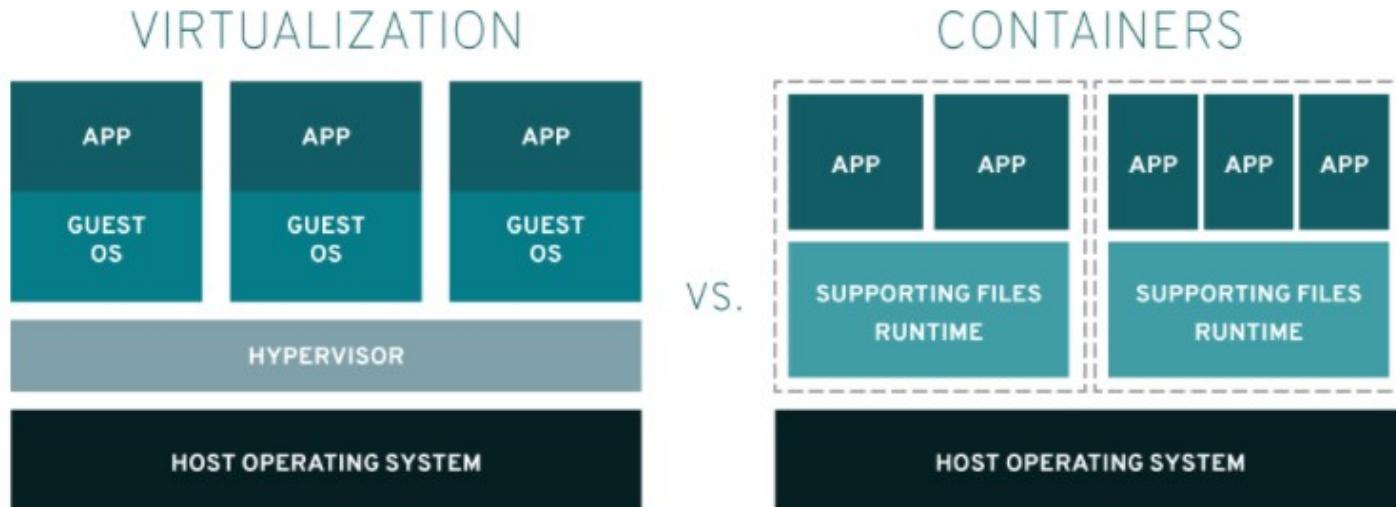
Containers



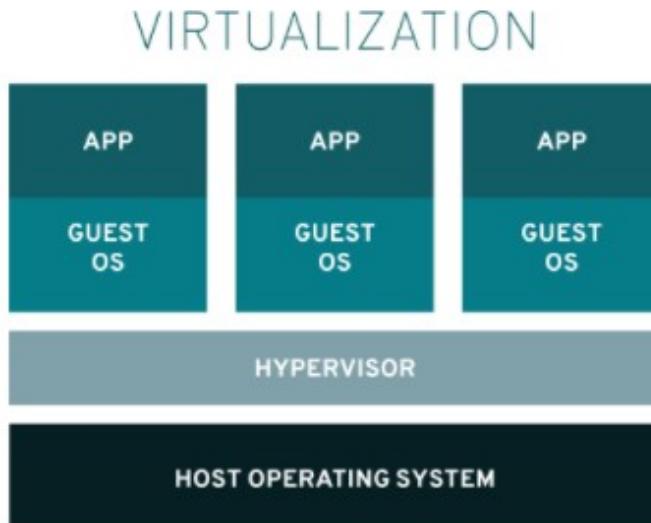
Wie unterscheiden sich die Technologien in Bezug auf Bereitstellung, Speicherplatz, Portabilität, Effizienz und Betriebssystem (Kernel)?

Dimensionen	Virtuelle Server	Container
Bereitstellung	Minuten	Millisekunden
Speicherplatz	Mehr Platz, da die Daten für das Betriebssystem dupliziert werden	Weniger Platz, da Container Images geteilt werden. Es wird lediglich die Applikation gespeichert
Portabilität	Virtuelle Server können auf Hardwarelevel verschoben werden solange der identische Hypervisor verwendet wird	Container können auf Laptops, auf Hardware im Rechenzentrum oder in der Cloud betrieben werden
Effizienz	Benötigt mehr RAM, CPU und Speicherplatz	Geringerer Ressourcenverbrauch
Betriebssystem (Kernel)	Dediziert	Geteilt

Können virtuelle Server durch Container ersetzt werden?

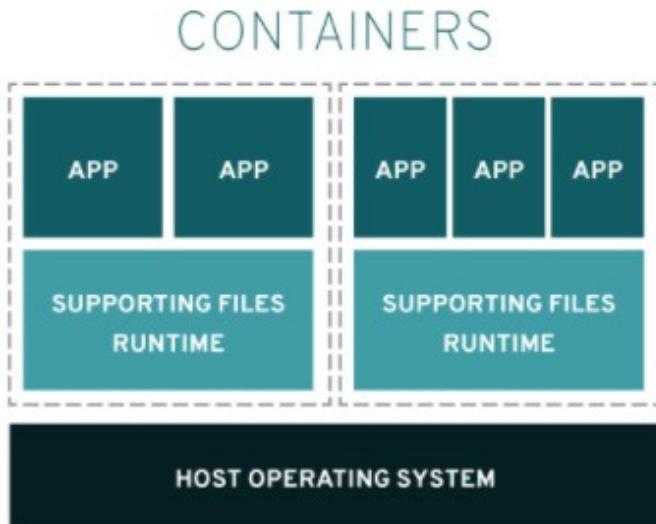


Können virtuelle Server durch Container ersetzt werden?



- Traditionelle, Legacy- und monolithische Workloads unterzubringen
- Riskante Entwicklungszyklen zu isolieren
- Infrastrukturressourcen (wie Netzwerke, Server und Daten) zu provisionieren
- ein Betriebssystem unter einem anderen Betriebssystem (z. B. Unix unter Linux) auszuführen

Können virtuelle Server durch Container ersetzt werden?



- cloudnative Apps zu entwickeln
- Microservices zu paketieren
- DevOps- oder CI/CD-Praktiken einzuführen
- skalierbare IT-Projekte in einer uneinheitlichen IT-Umgebung zu verschieben, die unter einem gemeinsamen Betriebssystem läuft

Self-Managed vs. Fully Managed Containers

Self-Managed vs. Fully Managed Containers

Self-Managed	Fully Managed
Volle Kontrolle über System und Daten	Kleinste Kontrolle über System und Daten
Höchste System Administrationskosten und Komplexität	Niedrigste System Administrationskosten und Komplexität
Zusätzliche Hardwarekosten	Keine Hardwarekosten
Zusätzliche DB und OS Lizenzkosten	Keine zusätzlichen DB und OS Lizenzkosten notwendig, bereits in der Miete inbegriffen
Kein SLA	Klar definiertes SLA

Self-Managed vs. Fully Managed Containers

Self-Managed	Fully Managed
Keine automatische Skalierung	Automatische Skalierung gemäss SLA
DR und Backup müssen zusätzlich konfiguriert werden	DR und Backup gemäss SLA
Updates müssen manuell eingepflegt werden	Updates werden automatisch eingespielt
Vollständige Transparenz des Systems	Starkes Vendor-Locking
Marginales Vendor-Locking	Wenig Transparenz des Systems

Block 4

Container Orchestrierung

Block 4

Warum braucht man Container-Orchestrierung?

Warum braucht man Container-Orchestrierung?

Provisionierung und Deployment

Konfiguration und Planung

Ressourcenzuweisung

Container-Verfügbarkeit

Skalieren oder Entfernen von Containern

Load Balancing und Traffic Routing

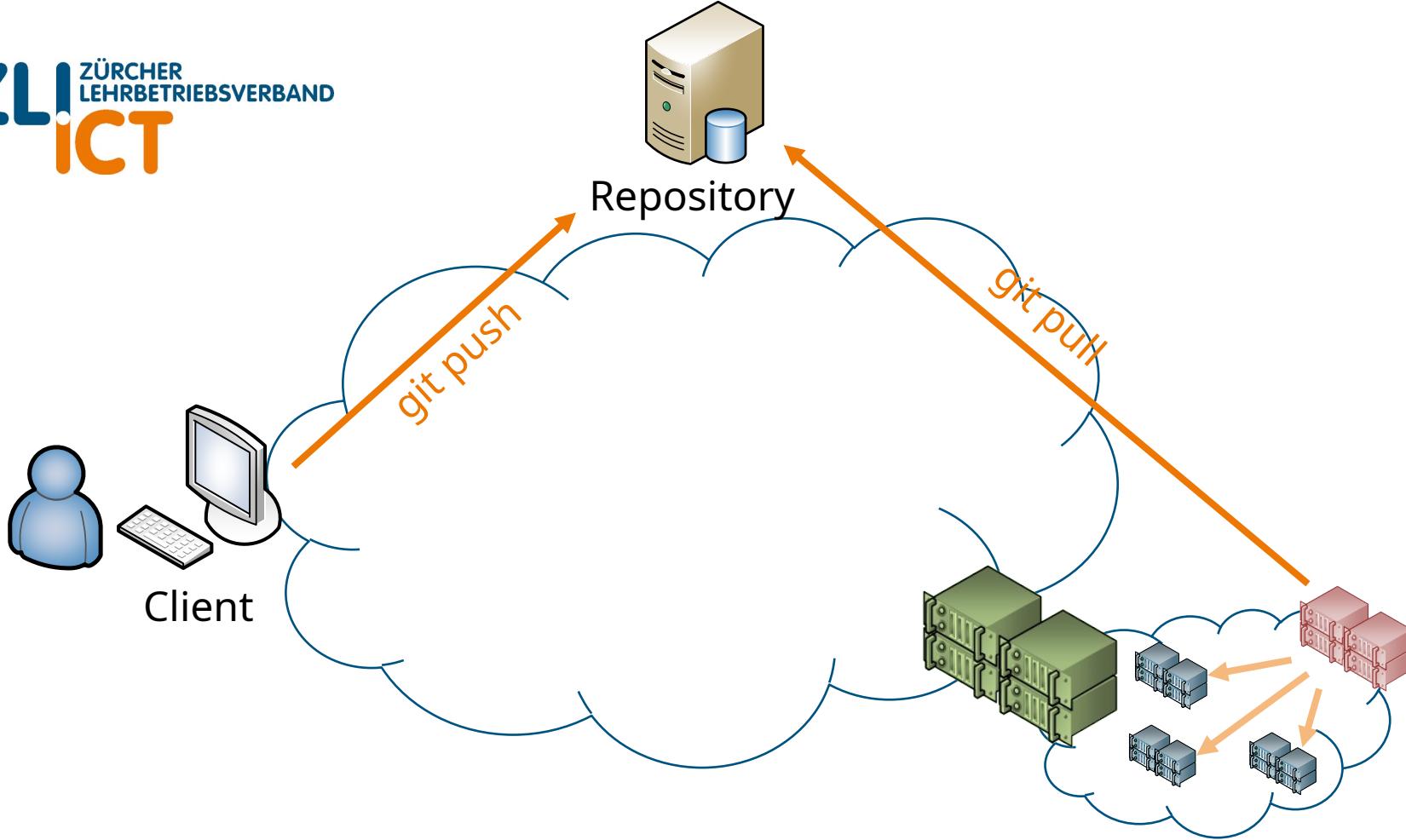
Load Balancing und Traffic Routing

Überwachen des Containerzustands

Konfigurieren von Anwendungen

Konfigurieren von Anwendungen

Sichern von Interaktionen zwischen Containern



Wie funktioniert Container-Orchestrierung

Beschreibung der **Konfiguration** einer Anwendung in einer YAML- oder JSON-Datei. Die Konfigurationsdatei teilt dem Konfigurationsmanagement-Tool mit, wo sich die Container-Images befinden, wie ein Netzwerk eingerichtet wird und wo Protokolle gespeichert werden.

Wie funktioniert Container-Orchestrierung

Beim **Deployment** eines neuen Containers plant das Container-Management-Tool automatisch das Deployment in einem Cluster, berücksichtigt alle definierten Anforderungen oder Einschränkungen und findet den richtigen Host. Das Orchestrierungs-Tool verwaltet dann den Lifecycle des Containers basierend auf den Spezifikationen, die in der Erstellungsdatei festgelegt wurden.

Wie funktioniert Container-Orchestrierung

Patterns verwalten die Konfiguration, den Lifecycle und die Skalierung von containerbasierten Anwendungen. Diese wiederholbaren Patterns sind die Tools, die ein Entwickler zum Erstellen vollständiger Systeme benötigt.

Wie funktioniert Container-Orchestrierung

Die Container-Orchestrierung kann in jeder **Umgebung** verwendet werden, in der Container ausgeführt werden, z. B. auf lokalen Servern und Public oder Private Cloud-Umgebungen

Welche Container-Orchestrierung Technologien kennen Sie?

Welche Container-Orchestrierung Technologien kennen Sie?



kubernetes



**RED HAT[®]
OPENSHIFT**



**Amazon
EKS**



Azure Kubernetes Service (AKS)



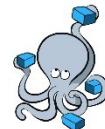
Google Kubernetes Engine



RANCHER[®]



MARATHON



**docker
Compose**



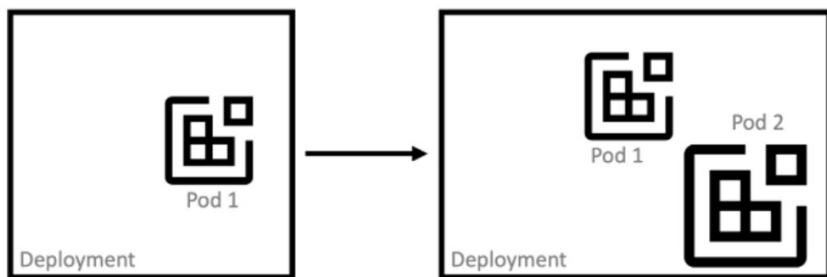
minikube



Nomad

Was versteht man unter «scaling» von Containern?

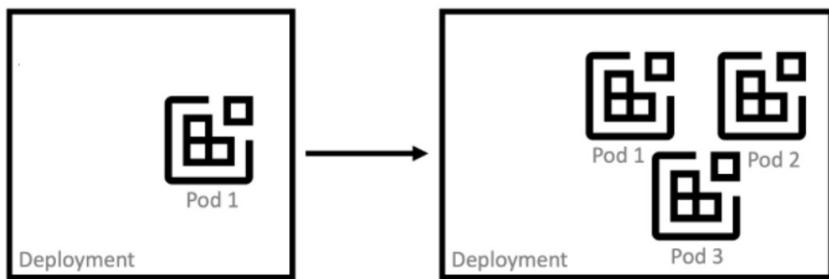
Vertikale Skalierung (Scale up)



Einige Pods können **mehr Ressourcen** nutzen können als andere.

Vertikale Skalierung eignet sich in diesem Zusammenhang eher für zustandsbehaftete Anwendungen, bei denen die Anfragen von einem bestimmten Pod bearbeitet werden müssen. Bei zustandslosen Anwendungen gibt es diese Anforderung nicht, so dass es einfacher ist, die horizontale Skalierung zu verwenden, und Sie haben den Vorteil, dass jeder Pod auf dieselbe Weise definiert ist.

Horizontale Skalierung (Scale out)



Es werden **mehr Instanzen** der Pods gestartet.

Horizontale Skalierung ist das, was Kubernetes am besten kann. Dafür wurde es ja auch entwickelt! Pods, die in einer Kubernetes-Bereitstellung ausgeführt werden, können mühelos repliziert werden, sodass eine horizontale Skalierung innerhalb von Sekunden möglich ist. Der in Kubernetes integrierte Load Balancer ermöglicht eine nahtlose Konfiguration. Mit dem horizontalen Pod-Autoscaler (HPA) können Sie die minimale und maximale Anzahl der Pods angeben, die Sie ausführen möchten, sowie die CPU- oder Speicherauslastung, die Ihre Pods anstreben sollten.

Block 5

Applikationen für die Cloud bereitstellen mit den 12 Faktoren

Block 5

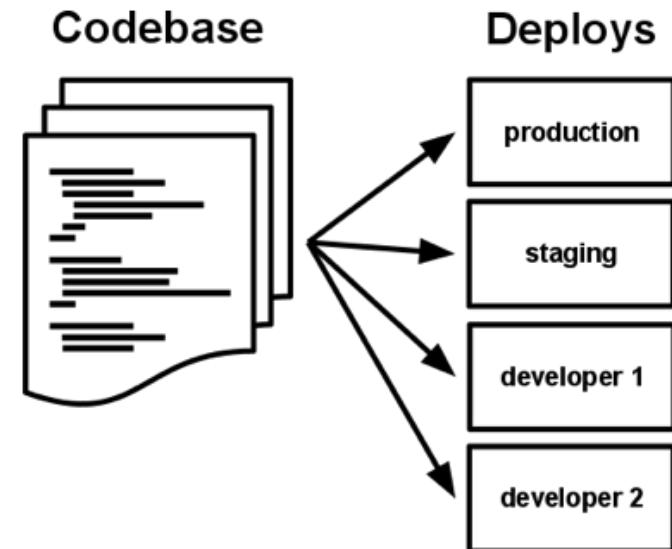
Was ist eine 12-Factor App

Heute wird Software oft als Dienst geliefert - auch Web App oder Software-As-A-Service genannt. Die Zwölf-Faktoren-App ist eine Methode um Software-As-A-Service Apps zu bauen die:

- **deklarative Formate** benutzen für die Automatisierung der Konfiguration, um Zeit und Kosten für neue Entwickler im Projekt zu minimieren;
- einen **sauberen Vertrag** mit dem zugrundeliegenden Betriebssystem haben, maximale Portierbarkeit zwischen Ausführungsumgebungen bieten;
- sich für das **Deployment** auf modernen Cloud-Plattformen eignen, die Notwendigkeit von Servern und Serveradministration vermeiden;
- die **Abweichung minimieren** zwischen Entwicklung und Produktion, um Continuous Deployment für maximale Agilität ermöglichen;
- und **skalieren** können ohne wesentliche Änderungen im Tooling, in der Architektur oder in den Entwicklungsverfahren.

Faktor 1: Codebase

Eine im Versionsmanagementsystem verwaltete Codebase, viele Deployments



Faktor 2: Abhängigkeiten

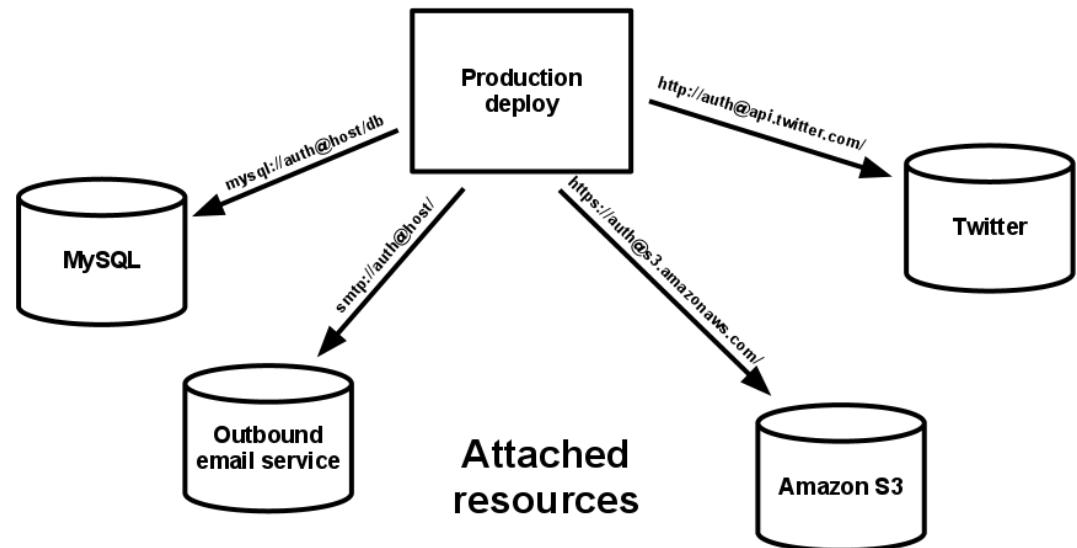
Abhängigkeiten explizit deklarieren und isolieren

Faktor 3: Konfiguration

Die Konfiguration in Umgebungsvariablen ablegen

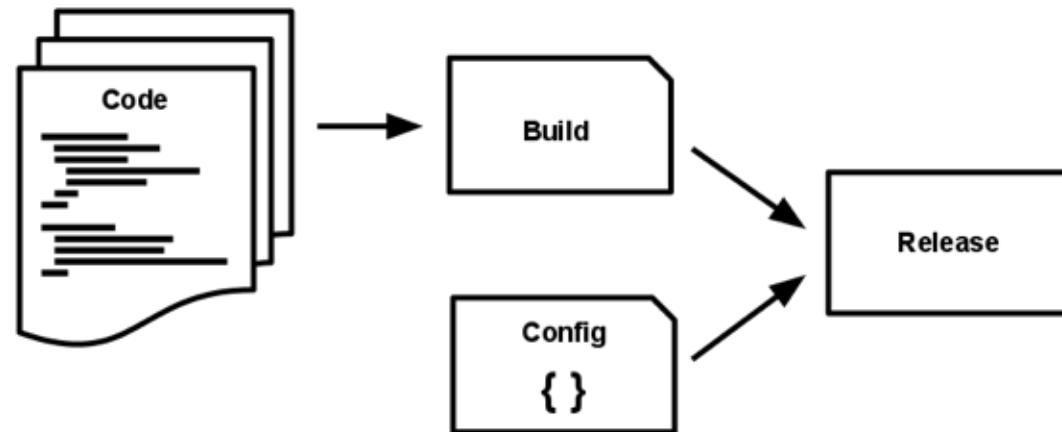
Faktor 4: Unterstützende Dienste

Unterstützende Dienste als angehängte Ressourcen behandeln



Faktor 5: Phasen Build, release, run

Build- und Run-Phase strikt trennen



Faktor 6: Prozesse

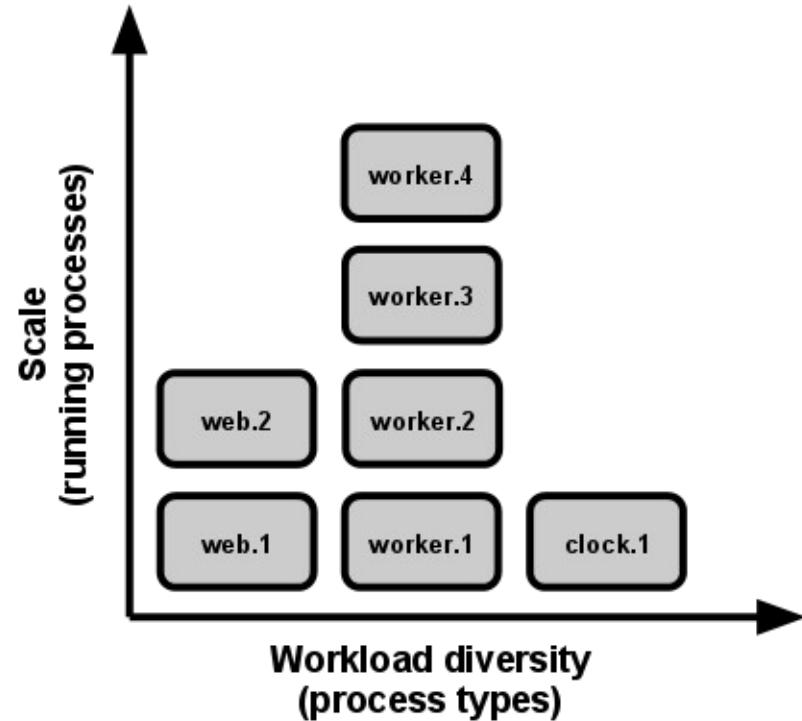
Die App als einen oder mehrere Prozesse ausführen

Faktor 7: Bindung an Ports

Dienste durch das Binden von Ports exportieren

Faktor 8: Nebenläufigkeit

Mit dem Prozess-Modell skalieren



Faktor 9: Einweggebrauch

Robuster mit schnellem Start und problemlosen Stopp

Faktor 10: Dev-Prod-Vergleichbarkeit

Entwicklung, Staging und Produktion so ähnlich wie möglich halten

Faktor 11: Logs

Logs als Strom von Ereignissen behandeln

Faktor 12: Admin-Prozesse

Admin/Management-Aufgaben als einmalige Vorgänge behandeln

Block 6

Kubernetes

Kubernetes Demo

Begriffe 1

- Pod: Container-Einheit



```
webapp.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp-deployment
  labels:
    app: webapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
        - name: webapp
          image: nanajanashia/k8s-demo-app:v1.0
          ports:
            - containerPort: 3000
          env:
            - name: USER_NAME
              valueFrom:
                secretKeyRef:
                  name: mongo-secret
                  key: mongo-user
            - name: USER_PWD
              valueFrom:
                secretKeyRef:
                  name: mongo-secret
                  key: mongo-password
            - name: DB_URL
              valueFrom:
                configMapKeyRef:
                  name: mongo-config
                  key: mongo-url
  ---  
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  type: NodePort
  selector:
    app: webapp
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
      nodePort: 30100
```

```
mongo-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: mongo-secret
type: Opaque
data:
  mongo-user: bW9uZ291c2VY
  mongo-password: bW9uZ29wYXNzd29yZA==
```

```
mongo-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongo-config
data:
  mongo-url: mongo-service
```

```
mongo.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-deployment
  labels:
    app: mongo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo
  template:
    metadata:
      labels:
        app: mongo
    spec:
      containers:
        - name: mongodb
          image: mongo:5.0
          ports:
            - containerPort: 27017
          env:
            - name: MONGO_INITDB_ROOT_USERNAME
              valueFrom:
                secretKeyRef:
                  name: mongo-secret
                  key: mongo-user
            - name: MONGO_INITDB_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mongo-secret
                  key: mongo-password
```

```
---  
apiVersion: v1
kind: Service
metadata:
  name: mongo-service
spec:
  selector:
    app: mongo
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
```

Container Port: Wie erreiche ich den Container im Pod

Target Port: Wie erreiche ich den Container im Pod | Service

Port: Wie erreiche ich den Service

NodePort: Wie erreiche ich die App

secretKeyRef: Verweis zu Username/Passwort für Datenbank

Login

configMapKeyRef: Verweis zu URL zur Datenbank (ENV

DB_URL)

Fortgeschrittene Handlungsziele

Sicherheitskonzept beurteilen

- Benutzer, Rollen, Zugriffe, Verantwortlichkeiten
- Auditing
- Verschlüsselung

Backup Slides

2.5 Container-Orchestrierung

- **Was gibt es für Deployment Strategien?**

Basis-Bereitstellung

Bei der Basisbereitstellung werden alle Knoten innerhalb einer Umgebung gleichzeitig mit einer einzigen neuen Serviceversion aktualisiert.

Wann sollten Sie Basic Deployments verwenden?

- Ihre Anwendung/Ihr Dienst ist nicht geschäfts-, unternehmens- oder umsatzkritisch
- Sie stellen ausserhalb der Geschäftszeiten bereit und niemand nutzt die Anwendung/den Dienst
- Sie experimentieren mit Bereitstellungen und es ist in Ordnung, wenn die Anwendung/der Dienst fehlschlägt.

Vorteile

- Einfach und schnell.

Nachteile

- Risiko, Ausfälle, langsameres Rollback.
- Vor nicht allzu langer Zeit war Basic Deployment die Art und Weise, wie Entwickler Anwendungen ausrollten. Normalerweise aktualisiert jemand in Ops die Server um Mitternacht und dann hofft man, dass alles gut geht.



2.5 Container-Orchestrierung

- Was gibt es für Deployment Strategien?

Rolling Deployment

Bei einem Rolling Deployment werden alle Knoten innerhalb einer einzelnen Umgebung schrittweise einzeln oder in N Stapeln mit einer neuen Serviceversion aktualisiert.

Wann sollten Rolling Deployments verwendet werden?

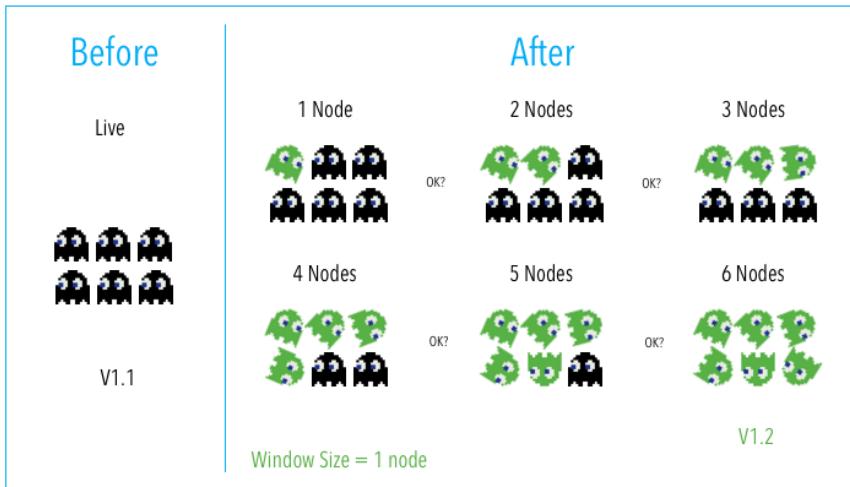
- Wenn Sie sowohl neue als auch alte Bereitstellungen unterstützen müssen.
- Lastausgleichsszenarien, die eine geringere Ausfallzeit erfordern.

Vorteile

- Einfach, relativ leicht rückgängig zu machen, geringeres Risiko als bei der Basisbereitstellung.
- Allmählicher Rollout der Anwendung mit zunehmendem Datenverkehr.

Nachteile

- Verifizierung von Gates zwischen Knoten schwierig und langsam.
- App/DB muss sowohl neue als auch alte Artefakte unterstützen. Manuelle Überprüfungen/Verifizierungen bei jedem Inkrement können viel Zeit in Anspruch nehmen.
- Verlorene Transaktionen und abgemeldete Benutzer sind ebenfalls zu berücksichtigen.



2.5 Container-Orchestrierung

- **Was gibt es für Deployment Strategien?**

Blue/Green-Bereitstellung

Beim Blue/Green Deployment werden zwei identische Umgebungen, Blue (Staging) und Green (Production) genannt, gleichzeitig mit verschiedenen Versionen eines Dienstes betrieben.

Wann sollte man Blue/Green Deployments verwenden?

- Wenn Sie eine Überprüfung in einer vollständigen Produktionsumgebung durchführen möchten.
- Wenn Sie keine Ausfallzeiten wünschen.

Vorteile

- Einfach, schnell, gut verständlich und leicht zu implementieren: Die Umstellung erfolgt fast augenblicklich.
- Geringeres Risiko im Vergleich zu anderen Bereitstellungsstrategien.
- Schnelles Rollback

Nachteile

- Die Replikation einer Produktionsumgebung kann komplex und teuer sein (z. B. nachgelagerte Abhängigkeiten von Microservices).
- Die QA/UAT-Testabdeckung erkennt möglicherweise nicht alle Anomalien und Regressionen in der blauen Umgebung.
- Ein Ausfall kann weitreichende Auswirkungen auf das Geschäft haben, bevor das Rollback einsetzt.
- Aktuelle Transaktionen und Sitzungen gehen durch den physischen Wechsel von einem Rechner, der den Datenverkehr bedient, zu einem anderen verloren.



2.5 Container-Orchestrierung

- Was gibt es für Deployment Strategien?

Canary-Bereitstellung

Bei der Canary-Bereitstellung werden alle Knoten in einer Umgebung in kleinen Phasen schrittweise aktualisiert, wobei in jeder Phase eine Verifizierung erforderlich ist, um zur nächsten Phase überzugehen.

Wann sollten Canary-Bereitstellungen verwendet werden?

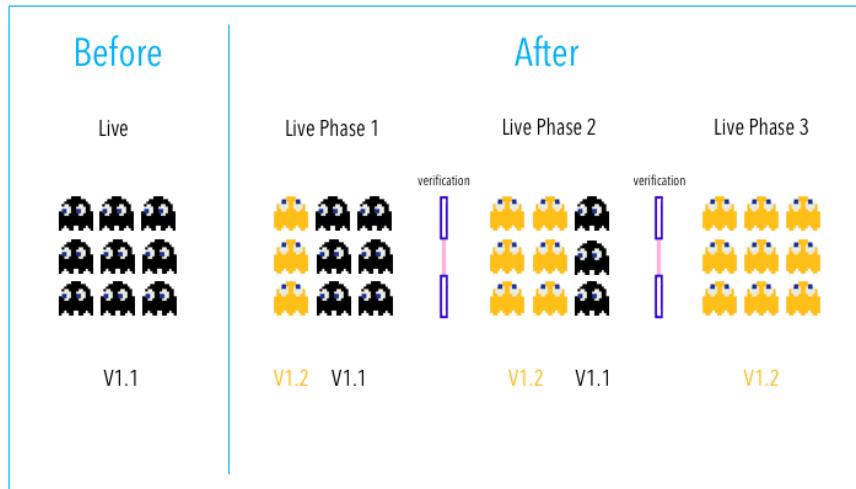
Wenn Sie überprüfen möchten, ob die neue Version der Anwendung in Ihrer Produktionsumgebung korrekt funktioniert.

Vorteile:

- Bereitstellung in kleinen Phasen.
- Geringstes Risiko im Vergleich zu allen anderen Bereitstellungsstrategien.
- Testen Sie in der Produktion mit echten Benutzern und Anwendungsfällen.
- Führen Sie zwei Serviceversionen nebeneinander aus und vergleichen Sie.
- Kostengünstiger als Blue/Green, da keine zwei Produktionsumgebungen erforderlich sind.
- Schnelles und sicheres Rollback.

Nachteile:

- Die Skripterstellung für Canary-Einsätze kann komplex sein.
- Manuelle Verifizierung kann Zeit in Anspruch.
- Erforderliche Überwachung und Instrumentierung für das Testen in der Produktion (APM, Log, Infra, Endbenutzer, usw.).
- Datenbankkompatibilität (Schemaänderungen, Abwärtskompatibilität).

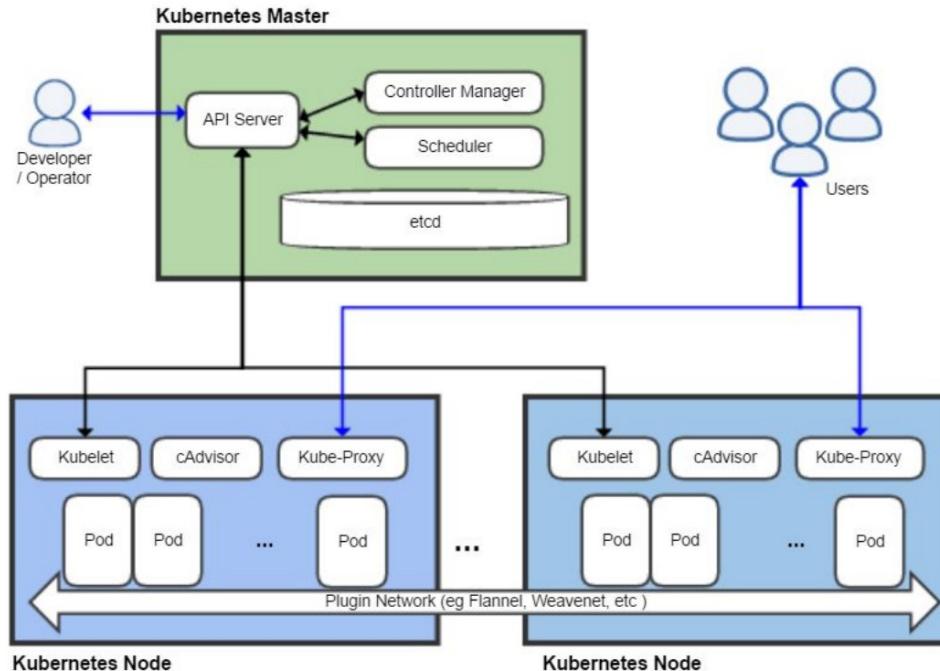


Wo kann nun ein Container unterstützen?

- Applikation muss per Container zur Verfügung stehen (z.B DockerHub)
- Installation wurde durch Vendor durchgeführt, somit kann System rascher in Betrieb genommen werden
- On Prem Betrieb des Containers möglich, somit vollständige Kontrolle über System und Daten
- Kompromiss zwischen System Komplexität und Transparenz

Orchestration Example - Kubernetes

- platform for automating deployment, scaling and management of containerized applications
- Initially written and designed by Google
- Borrows heavily from Google longs experience with managing containers



3.3: Characteristics

- Application which is optimized for running on a cloud infrastructure having essential characteristics of being scalable and resilient
- Being economically efficiently
- Designed as a distributed application built up from stateless components
- Each phase in the application life-cycle has to be adapted and optimized for running in a cloud

Difference Capex / Opex

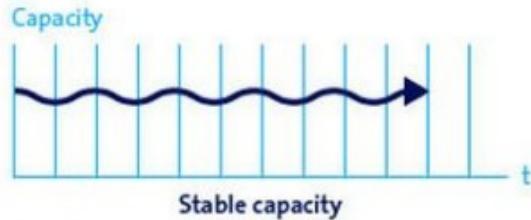
Capital Expenses (CAPEX)

- Expenditures creating future benefits.
- Incurred when a business spends money either to buy fixed assets
- CAPEX is a cost which cannot be deducted in the year in which it is paid or incurred and must be capitalized

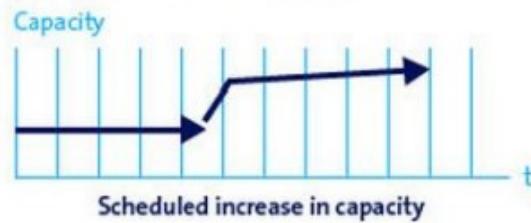
Operational Expenses (OPEX)

- Ongoing cost for running a product, business, or system
- Day-to-day expense
- Operating expenses is the sum of an operating expenses for a period of time (e.g. month or year)

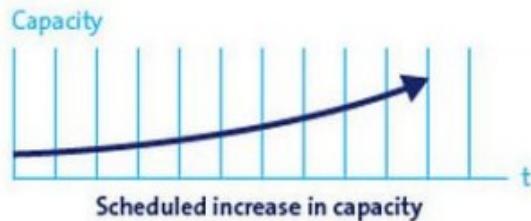
ICT Consumption models



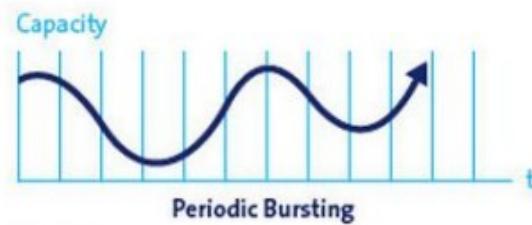
Stable capacity



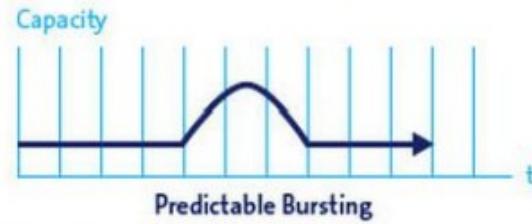
Scheduled increase in capacity



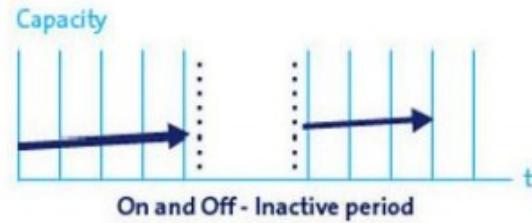
Scheduled increase in capacity



Periodic Bursting



Predictable Bursting



On and Off - Inactive period

Cloud Storage

- From Block Storage to Blob Storage
- CAP Theorem

Cloud Security

Serverless Functions

Docker

- Open source software engine to **commoditize** linux containers
- Features
 - Portability: portable deployment of application across machines
 - Application-centric: user facing function geared towards application deployment
 - Build automation: create containers from build files
 - Versioning support: software development best-practices
 - Component reuse: any container can be used as a base, specialized and saved
 - Sharing: support for public / private repositories of containers
 - Tools: CLI / rest API for interacting with docker

Container

- A container is a runnable instance of an image
- A container can be created, run, stopped, moved, deleted using Docker API or CLI
- A container can be attached to one or more networks and to storage
- Containers are relatively well isolated from each other, as well as from the host machine
- A container is defined by its image as well as any configuration options provide when it is created or run
- When a container is removed, any changes to its state that are not stored in persistent storage disappear

Images

- An image is a collection of read-only content for creating a docker container
- Docker container are based on own images or those created by others and published in a registry
- To build an image, a dockerfile is created with a simple syntax for defining the steps needed to create the image and run int
- Each instruction in a dockerfile creates a layer in the image
- When you change the dockerfile and rebuild the image, only those layers which have changed are rebuilt
- Docker image are therefore lightweight, small and fast, when compared to virtualization technologies
- Docker images are a series of read-only layers
- Each layer has its own unique identifier
- The first image in the stack is called a base image
 - All other layers are stacked on top of this layer
- The image layers are stored in the docker hosts local storage area
- Containers are a combination of a docker image with a thin writable layer added to the top
- All writes that add new or modified existing data are stored in the writable layer
 - When the container is deleted the writable layer is also deleted
 - Changes related to image content create new layers
 - The original image content remains unchanged
- Because each container has its own thin writable container layer and all data is stored this container layer, multiple containers can share the same underlaying image and yet have their own data state

Container Definition - Dockerfile

Dockerfile

- Recommended, portable way to create a custom docker image
- By default is a text called dockerfile
- Follows very simple syntax
 - FROM: an existing image
 - RUN: command
 - ADD: a config file from the host machine
 - EXPOSE: port by default
 - CMD: run command by default when container starts, operator does not override it
 - ENTRYPOINT: command will always be executed when container starts
- docker built -t <my_container_image_name> .

Environment Variables

Environment variables

- Images are typically generic services images
- Docker allows the operator to set environment variables in a new container
 - By default some variables are automatically added
 - HOME: as default user within the container is root, default is /
 - HOSTNAME: hostname associated with the container
 - PATH:
 - TERM: xterm if the container is allocated a pseudo terminal entry point
- Can only be done at container startup time
- Useful for linking containers

Container Image Registry

Docker Registry

- A central feature of docker is the docker registry
 - Images live inside repositories, and repositories live on registry
- Registry code is open source, so anyone can run his own private registry
- Images are stored using a specific naming format
 - <username>/<image-name>:<tag>
- The tag can be any numbering scheme
 - Tag images differently for each version
- Commands
 - logins: login to registry, required to use docker hub
 - pull <image_name:tag>: download an image to local registry
 - tag name source_image: creates a new tagged image based on a existing image
 - push <image_name:tag>: uploads a built image to a docker registry
 - search <keyword>: search dockerhub for a key word

Kubernetes Concepts

- Clusters: set of machines where pods are deployed, managed and scaled
 - Nodes are connected via a flat network
 - Typical cluster sizes range from 1-200 nodes
- Pod - a pod consists of one or more containers
 - Guaranteed to be co-located on the same machine
 - A pod is basic unit of scheduling
- Controller - reconciliation loop that drives actual cluster state toward the desired cluster state
 - Replication controller: handles replication and scaling of pods
- Service: set of pods that work together
 - Custom and built-in service are supported
- Label - object reference
 - A user can assign key-value pair to any API object in the system
 - Label selector: a query against a label that returns matching objects

Sicherheitskonzept beurteilen

- Benutzer, Rollen, Zugriffe, Verantwortlichkeiten
- Auditing
- Verschlüsselung