109 Dienste in der Public Cloud betreiben und überwachen

Block 1: Grundlagen des Cloud-Computing

Modulbeschreibung und Ziele

Modulbeschreibung

Modulbeschreibung

- Modul 109 - Dienste in der Cloud betreiben und überwachen
- Modul 210 - Public Cloud für Anwendungen entwickeln

Handlungsziele

- Definiert anhand einer Beispielanwendung eine Cloud-Lösung (einfache Architektur), abgestimmt auf die Zielsetzungen des Unternehmens.
- Beurteilt das vorliegende Sicherheitskonzept mit Benutzer, Rollen, Zugriffen, Auditing, Verschlüsselung, Verantwortlichkeiten (Shared Responsibility) anhand der Beispielanwendung
- Stellt die Beispielanwendung mit Hilfe von virtuellen Servern und/oder Container Technologien mit einem Datenbank Dienst (Plattform Service) bereit.
- Implementiert nach Vorgabe die Überwachung und die Verwaltung der Infrastruktur (Monitoring, Logging, Alarmierung, Remote Management, Patching und Skalierung).
- Implementiert nach Vorgabe die Datensicherung (DR/Backup) der Beispielanwendung mit Datenbank.

Auftrag 1.2: Grundlagen Cloud-Computing

- Wie ist der Begriff Cloud entstanden? Wieso heisst es Cloud? Der Ursprung des Begriffs "Cloud": Der Begriff "Cloud" (Wolke) wird oft auf das Diagramm eines Netzwerks zurückgeführt, das in den 1990er Jahren in Telekommunikationsnetzen verwendet wurde. Die Cloud war eine Metapher für das Internet, die genutzt wurde, um die Komplexität des Untergrunds darzustellen.
- Wie wird der Begriff Cloud definiert, z.B. gemäss NIST? Cloud-Computing ist ein Modell, das es erlaubt, bei Bedarf jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können.
- Welches sind die 5 Merkmale einer Cloud?
  - **On demand self-service**: Nutzung auf Abruf - Der Nutzer hat jederzeit Zugriff auf die Cloud-Dienste.
  - **Broad network access**: Zugriff mit bekannten Technologien - Der Nutzer kann über Internetverbindung und entsprechende Clients auf die Dienste zugreifen.
  - **Ressource pooling**: Zusammenlegung von Ressourcen - Die Serverkapazitäten werden gebündelt in einer einheitlichen Cloud zur Verfügung gestellt.
  - **Rapid elasticity**: Ressourcenanpassung - Die zur Verfügung stehenden Ressourcen werden angepasst und es entsteht der Eindruck eines unbegrenzten Speicherplatzes.
  - **Measured service**: Überwachung des Dienstes - Die einzelnen Cloud-Server werden immer wieder überwacht und optimiert.
- Welche Cloud Dienstleistungen kennen Sie? Datenlagerung und Backup, Webdienste, E-Mail und Kalenderdienste, Identity and Access Management (IAM), Datenbankdienste
- Welche Cloud Anbieter kennen Sie? Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, IBM Cloud und Oracle Cloud.
- Welche Cloud Deployment Modelle kennen Sie?
  - **Private Cloud**: Eine exklusive, von einem Unternehmen intern genutzte Cloud-Infrastruktur.
  - **Public Cloud**: Eine für alle Nutzer öffentlich zugängliche Cloud-Infrastruktur über das Internet.
  - **Hybrid Cloud**: Eine Kombination aus privaten und öffentlichen Cloud-Ressourcen.
- Was sind Cloud Service Modelle?
  - **Infrastructure as a Service (IaaS)**: Hierbei stellt der Anbieter grundlegende Infrastrukturen wie Server, Speicherplatz oder Netzwerkkomponenten zur Verfügung.
  - **Platform as a Service (PaaS)**: Bei PaaS wird neben der Infrastruktur auch eine Plattform zum Entwickeln und Betreiben von Anwendungen bereitgestellt.
  - **Software as a Service (SaaS)**: In diesem Modell wird die Anwendung selbst als Dienst über das Internet bereitgestellt.
- Weshalb soll ich Dienste aus der Cloud beziehen? Was sind die Vorteile?
  - Kosteneinsparungen durch den Wegfall von Vorabinvestitionen
  - Skalierbarkeit und Flexibilität
  - Zugang von überall
  - Einfache Aktualisierungen und Wartung

- Was sind die Nachteile?
  - Mögliche Datenschutz- und Sicherheitsbedenken
  - Abhängigkeit von Internetverbindung und Cloud-Anbieter
  - Potenzielle Kosten bei hoher Datenmenge oder intensiver Nutzung
- Welche Dienstleistungen werden in Ihrem Betrieb On-Premise (eigenes Rechenzentrum) betrieben? Speicher, Webdienste, Citrix, SAP, SNOW
- Wie werden technologische Beiträge in der Cloud geteilt bzw. zur Verfügung gestellt? Technologische Beiträge in der Cloud werden in der Regel über APIs (Application Programming Interfaces) geteilt oder zur Verfügung gestellt. Ein API ermöglicht es, dass unterschiedliche Softwareanwendungen miteinander interagieren. In Bezug auf die Cloud können APIs beispielsweise genutzt werden, um Daten zwischen verschiedenen Cloud-Diensten zu übertragen oder um die Funktionalitäten eines Cloud-Dienstes in eine andere Anwendung zu integrieren.

Block 2: Cloud Anbieter und git

Auftrag 2.2: Git zur Sourcecode- und Konfigurationsverwaltung

Git-Benutzernamen und E-Mail-Adresse konfigurieren:

```
git config --global user.name "Ihr Name"
git config --global user.email "IhreEmail@beispiel.com"
```

Ein Verzeichnis für Ihr Projekt erstellen und in dieses Verzeichnis wechseln:

```
cd beispiel
```

Ein leeres Git-Repository in diesem Verzeichnis initialisieren:

```
git init
```

Eine README-Datei zum Git-Repository hinzufügen:

```
git add README
```

Den Status Ihres Repositories überprüfen:

```
git status
```

Ihren ersten Commit durchführen:

```
git commit -m "Erste Version"
```

Änderungen mit git diff anzeigen und in ein Patch packen:

```
git diff
git diff --no-prefix > patch.diff-
```

Die Historie Ihres Repositories anzeigen:

```
git log
```

Einen neuen Branch erstellen und zwischen Branches wechseln:

```
git checkout -b new_branch
git checkout master
```

Änderungen vom neuen Branch in den Master-Branch übernehmen:

```
git merge new_branch
```

Den Master-Zweig in Main umbenennen:

```
git branch -m master main
git push -u origin main
git push origin --delete master
```

Tags in Git erstellen und anzeigen:

```
git tag
git tag -a v1.4 -m 'my version 1.4'
git show
```

Block 3: Einführung in Container-Technologie

Installation Docker Engine

Dependencies

```
sudo apt update
sudo apt install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

Repository

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Docker Engine

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io -y
```

Anpassungen User (dieser Schritt ist wichtig, da sonst immer mit sudo gearbeitet werden muss)

```
sudo usermod -aG docker $USER
newgrp docker
```

Auftrag 3.1: Grundlagen Container-Technologie

1. Was ist Container-Technologie oder Container-Virtualisierung? Container-Technologie oder Container-Virtualisierung ist ein Ansatz zur Softwareentwicklung, bei dem ein Anwendung und ihre Abhängigkeiten in einem Prozess gekapselt werden, der von der zugrunde liegenden Betriebssysteminfrastruktur isoliert ist. Jeder Container teilt sich das Betriebssystem-Host und in der Regel auch die Binärdateien und Bibliotheken, ist aber in seiner Ausführung vom Rest des Systems getrennt.
2. Was sind die Vor- und Nachteile der Container-Technologie zu virtuellen Servern (VM)? Vorteile von Containern gegenüber VMs: • Geringerer Ressourcenbedarf: Container benötigen weniger Systemressourcen als VMs, da sie weniger Overhead haben, da sie das Betriebssystem nicht duplizieren müssen. • Schnellere Startzeiten: Container können in Sekundenbruchteilen gestartet und gestoppt werden, während VMs Minuten benötigen. • Portabilität: Anwendungen in Containern können einfach zwischen verschiedenen Systemen und Cloud-Plattformen verschoben werden. Nachteile von Containern gegenüber VMs: • Geringere Isolierung: Da Container das gleiche Betriebssystem teilen, sind sie weniger isoliert als VMs. • Spezifische Betriebssystemabhängigkeiten: Container sind oft an das zugrunde liegende Betriebssystem gebunden, während VMs beliebige Betriebssysteme ausführen können.
3. Welche Produkte kennen Sie im Zusammenhang mit virtuellen Servern und Containern? Im Zusammenhang mit virtuellen Servern sind Produkte wie VMware vSphere, Microsoft Hyper-V und Oracle VM bekannt. Bezüglich Containern sind Docker, Kubernetes, Red Hat OpenShift und Rancher einige der bekanntesten Technologien.
4. Wie unterscheiden sich die Technologien VM und Container in Bezug auf Bereitstellung, Speicherplatz, Portabilität, Effizienz und Betriebssystem (Kernel)? • Bereitstellung: Container können schneller bereitgestellt werden als VMs, da sie weniger Ressourcen benötigen. • Speicherplatz: Container benötigen weniger Speicherplatz als VMs, da sie das Betriebssystem nicht duplizieren müssen. • Portabilität: Container sind portabler als VMs, da sie die gesamte Anwendung und ihre Abhängigkeiten in einem einzigen Paket kapseln. • Effizienz: Container sind effizienter als VMs in Bezug auf Systemressourcen, da sie weniger Overhead haben. • Betriebssystem (Kernel): VMs können verschiedene Betriebssysteme auf demselben physischen Host ausführen, während Container das Host-Betriebssystem teilen müssen.
5. Können virtuelle Server immer durch Container ersetzt werden? Nein, es gibt Szenarien, in denen VMs eine bessere Wahl sind. Zum Beispiel, wenn vollständige Isolierung oder spezifische Betriebssystemanforderungen erforderlich sind. Außerdem können bestimmte

Anwendungen oder Systeme aufgrund ihrer spezifischen Anforderungen oder Abhängigkeiten besser für VMs geeignet sein.

6. Was ist unterschied zwischen Self-Managed und Fully Managed? Ein Self-Managed-Service bedeutet, dass der Nutzer selbst für die Installation, Wartung, Aktualisierungen und Sicherheit verantwortlich ist. Der Nutzer hat mehr Kontrolle, aber auch mehr Verantwortung. Ein Fully Managed Service wird vollständig vom Anbieter verwaltet. Der Anbieter kümmert sich um die Installation, Wartung, Aktualisierungen und Sicherheit. Dies kann die Belastung für den Nutzer reduzieren, nimmt ihm jedoch auch einen Teil der Kontrolle ab.

Auftrag 3.2: Webserver mit Docker

Lassen Sie ihre HTML Webseite in einem Container laufen. Nutzen Sie dazu Container Images von Docker Hub, z.B. das Image von nginx: https://hub.docker.com/_/nginx

```
docker run -p 8080:80 -v /home/minikube/index.html:/usr/share/nginx/html/index.html index:latest
```

```
version: '3.8'

services:
  nginx:
    image: index:latest
    ports:
      - 8080:80
    volumes:
      - /home/minikube/index.html:/usr/share/nginx/html/index.html
```

Block 4: Container Orchestration

Auftrag 4.1: Container-Orchestrierung

1. Warum braucht man Container-Orchestrierung? Container-Orchestrierung wird benötigt, um komplexe Anwendungen mit vielen Containern zu verwalten, zu skalieren, zu überwachen und hochverfügbar zu machen.
2. Wie funktioniert Container-Orchestrierung? Bei der Container-Orchestrierung werden Tools und Plattformen verwendet, um die Bereitstellung, Verwaltung und Kommunikation von Containern zu automatisieren und zu koordinieren.
3. Welche Container-Orchestrierung Technologien kennen Sie? Einige bekannte Container-Orchestrierungstechnologien sind Docker Swarm, Kubernetes, Apache Mesos und Amazon ECS (Elastic Container Service).
4. Was versteht man unter "Scaling Containers"? "Scaling Containers" bezieht sich auf die Fähigkeit, die Anzahl der Containerinstanzen, die eine Anwendung ausführen, dynamisch zu erhöhen oder zu verringern, um die Anforderungen des Workloads zu erfüllen.
5. Was gibt es für Deployment-Strategien? Es gibt verschiedene Deployment-Strategien wie Rolling Update, Blue-Green Deployment, Canary Deployment und A/B-Testing, die es ermöglichen, Anwendungen schrittweise zu aktualisieren oder neue Versionen bereitzustellen, um Ausfallzeiten zu minimieren und einen reibungslosen Übergang sicherzustellen.

Kubernetes - minikube - Was ist das?

minikube ist eine lokale Kubernetes Installation mit dem Ziel, einfach und schnell Kubernetes zu lernen, und simple Kubernetes Umgebungen zu entwickeln und auszutesten.

Systemvoraussetzungen: 2 CPUs oder mehr Mindestens 2GB RAM Mindestens 20GB Disk Space Internet Verbindung Container oder virtuelle Umgebung: Docker, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, oder VMware Fusion/Workstation

---

Installation minikube

Dependencies

```
sudo apt install curl wget apt-transport-https -y
```

Download und Installation minikube

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Download und Installation kubectl

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Ein paar minikube Addons

```
minikube start --driver=docker
minikube addons enable ingress
minikube addons enable dashboard
minikube addons enable metrics-server
```

Reboot Linux

```
sudo reboot
```

Start minikube und Dashboard

```
minikube start --driver=docker
minikube dashboard --url
```

Um das Dashboard von einem anderen Computer zu öffnen kann man den folgenden Befehl benutzen um die Ports zu Mappen:

```
ssh -L <Port>:127.0.0.1:<Port> <User>@<IP>
```

Auftrag 4.2 Container Orchestration mit Docker Compose

1. Folgen Sie dem Tutorial https://docs.docker.com/compose/gettingstarted/
2. Erstellen Sie ein Diagram und beschreiben Sie die einzelnen verwendeten Container
3. Beantworten Sie folgende Fragen
    1. Was ist Redis? Redis ist eine Open-Source-Datenbank, die als Schlüssel-Wert-Speicher dient und schnellen Zugriff auf Daten ermöglicht. Es wird häufig als Zwischenspeicher oder Caching-Mechanismus eingesetzt.
    2. Welche Ports werden genutzt? Redis verwendet standardmäßig den Port 6379 für die Kommunikation zwischen Client und Server. Dieser Port wird in der Regel für den Zugriff auf Redis-Dienste genutzt.
    3. Was ist die Bedeutung von ENV im DOCKERFILE? ENV (Environment) in einem Dockerfile ermöglicht das Setzen von Umgebungsvariablen innerhalb des Container-Images. Diese Variablen können während der Laufzeit verwendet werden, um bestimmte Konfigurationswerte oder Parameter anzupassen, wie z.B. Datenbankverbindungszeichenfolgen oder API-Schlüssel.

Auftrag 4.3: minikube - Kubernetes ganz einfach (Crash Kurs)

Die installierte Kubernetes Umgebung muss nach jedem Reboot/Neustart des Systems mit den folgenden Befehlen gestartet werden.

```
minikube start --driver=docker
minikube dashboard --url
```

- Erstellen Sie in ihrem privaten Git Account ein Repository mit dem Namen dev_minikube
- Erstellen Sie im Homeverzeichnis des Benutzers ein Verzeichnis dev_minikube
- Initialisieren Sie ihr Git Repository in diesem Verzeichnis, erstellen die 4 benötigten Files (Git Repository der Konfigurationsfiles) für Deployment und Services und stellen Sie sicher, dass diese Konfigurationsfiles in ihrem Git Account auf github.com gespeichert sind.

Sind die Konfigurationsfiles einmal erstellt, gestaltet sich das eigentliche Deployment sichtlich einfach:

```
kubectl apply -f mongo-config.yaml
kubectl apply -f mongo-secret.yaml
kubectl apply -f mongo.yaml
kubectl apply -f webapp.yaml
```

Das wars auch schon. Jetzt müssen Sie nur noch herausfinden, über welchen URL bzw. IP Adresse und Port auf Ihren neu deployten Service zugegriffen werden kann.

IP Adresse des minikube Nodes herausfinden:

```
minikube ip
```

Port des Webapp Services (Nodeport) herausfinden:

```
kubectl describe service webapp
```

URL wäre dann zusammengesetzt: http://192.168.49.2:30100

Falls die Webseite nicht erreichbar ist, kann der folgende Befehl ausgeführt werden: `

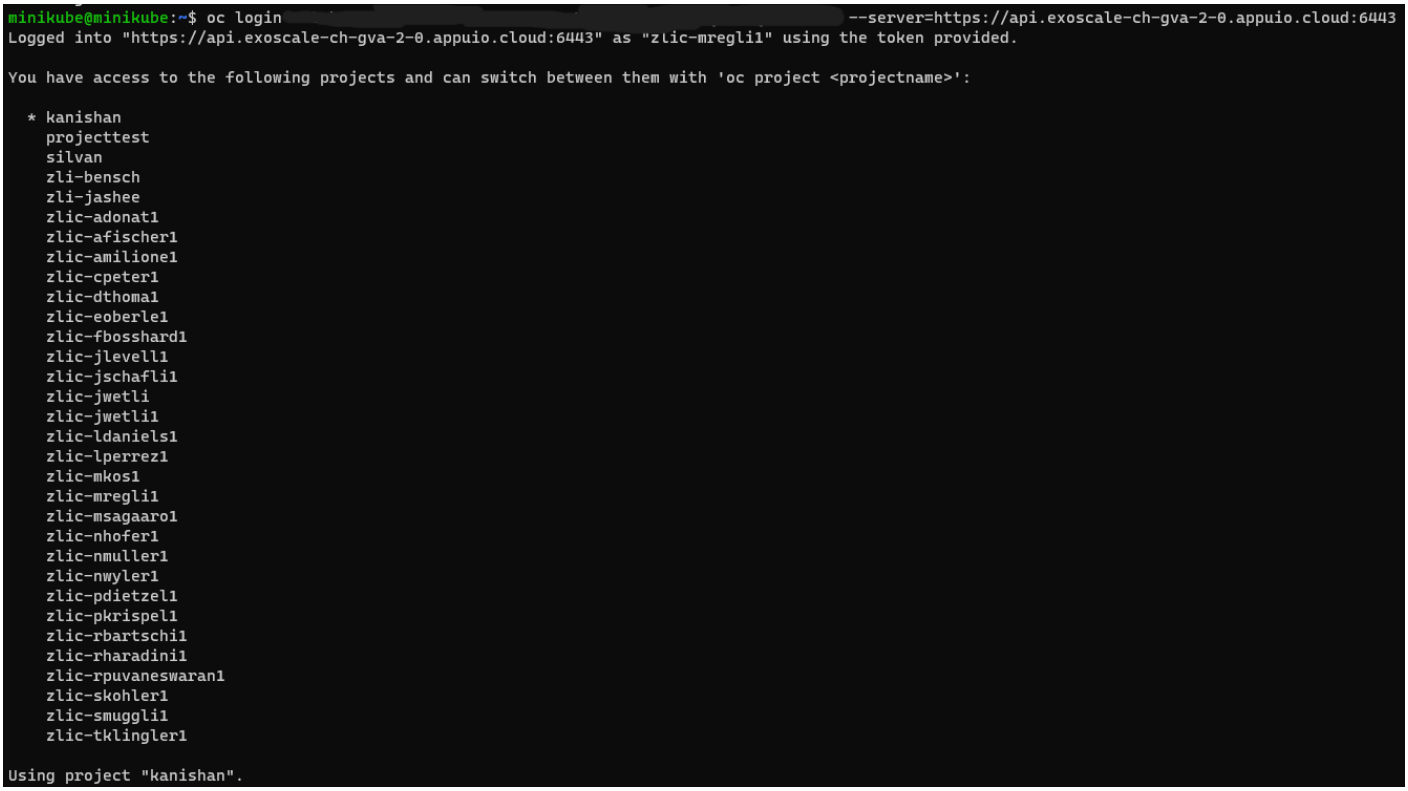
```
minikube service webapp-service
```

Block 6: Openshift Plattform kennenlernen

Auftrag 6.1: OpenShift APPUiO Projekt einrichten

Container Registry Login

Als Container-Registry, verwenden wir die Github Container Registry. Um Images zu verwalten und in OpenShift abzufragen, befolgen Sie folgende Schritte:



Schritt 1 - Token Erstellen

Erstellen Sie unter folgendem Link einen (Classic) Personal Access Token (PAT) mit den write:packages Berechtigungen.

https://github.com/settings/tokens

Schritt 2 - Docker Login

Loggen Sie sich nun lokal in die Container Registry ein. Als Passwort dient ihnen der zuvor erstellte PAT.

```
docker login ghcr.io -u <Github Username>
```

```
minikube@minikube:~$ docker login ghcr.io -u masluse
Password:
WARNING! Your password will be stored unencrypted in /home/minikube/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Schritt 3 - OpenShift Login

Damit OpenShift ihre privaten Container-Images herunterladen kann, müssen sie für ihr Projekt die Zugangsdaten definieren. Stellen Sie sicher, dass Sie sich auf dem korrekten Projekt befinden:

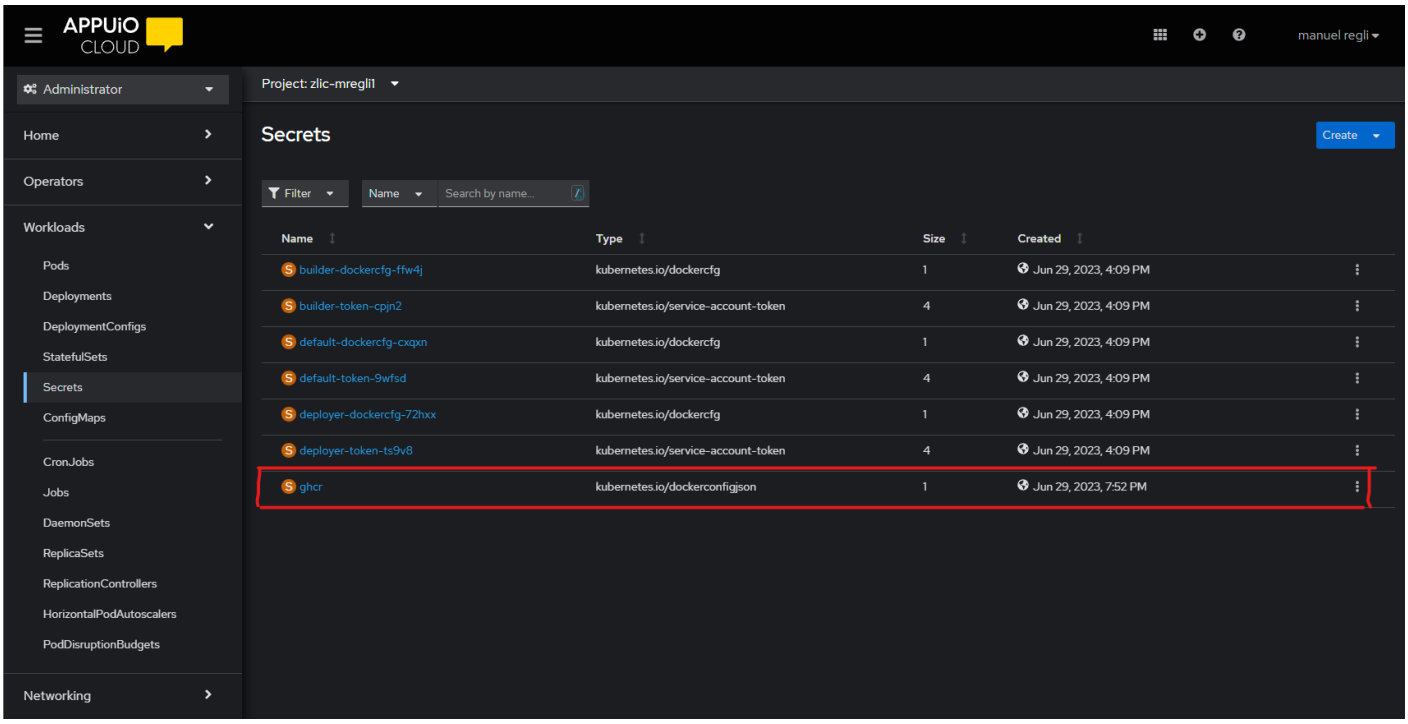
```
oc project <openshift username>
```

```
minikube@minikube:~$ oc project zlic-mregli1
Now using project "zlic-mregli1" on server "https://api.exoscale-ch-gva-2-0.appuio.cloud:6443".
```

Erstellen Sie anschliessend ein Secret mit den Credentials und konfigurieren Sie die Credentials als Default:

```
oc create secret docker-registry ghcr --docker-server=ghcr.io --docker-username=<Github Username> --docker-password=<Github PAT>
oc secrets link default ghcr --for=pull
```

```
minikube@minikube:~$ oc create secret docker-registry ghcr --docker-server=ghcr.io --docker-username=masluse --docker-password=
secret/ghcr created
minikube@minikube:~$ oc secrets link default ghcr --for=pull
minikube@minikube:~$
```



Auftrag 6.2: HTML-Seite auf OpenShift deployen

1. Erstellen Sie einen neuen Ordner und kopieren Sie die HTML Seite aus Auftrag 1.1 und das Dockerfile aus Auftrag 3.2. Ersetzen Sie das Baseimage "nginx" mit "nginxinc/nginx-unprivileged".

```
cd
mkdir auftrag-6.2
cd auftrag-6.2
echo '<H1> Auftrag 6.2: HTML-Seite auf OpenShift deployen </H1>' > index.html
echo 'FROM nginxinc/nginx-unprivileged' > Dockerfile
echo 'WORKDIR /usr/share/nginx/html' >> Dockerfile
echo 'COPY index.html index.html' >> Dockerfile
```

2. Builden und testen Sie den Container, sodass dieser die Webseite auf Port 8080 publiziert.

```
docker build -t ghcr.io/<Github Username>/html-page:v1 .
docker run -p 8080:8080 ghcr.io/<Github Username>/html-page:v1
```

3. Pushen Sie den Container in die Github Container Registry:

```
docker push ghcr.io/<Github Username>/html-page:v1
```

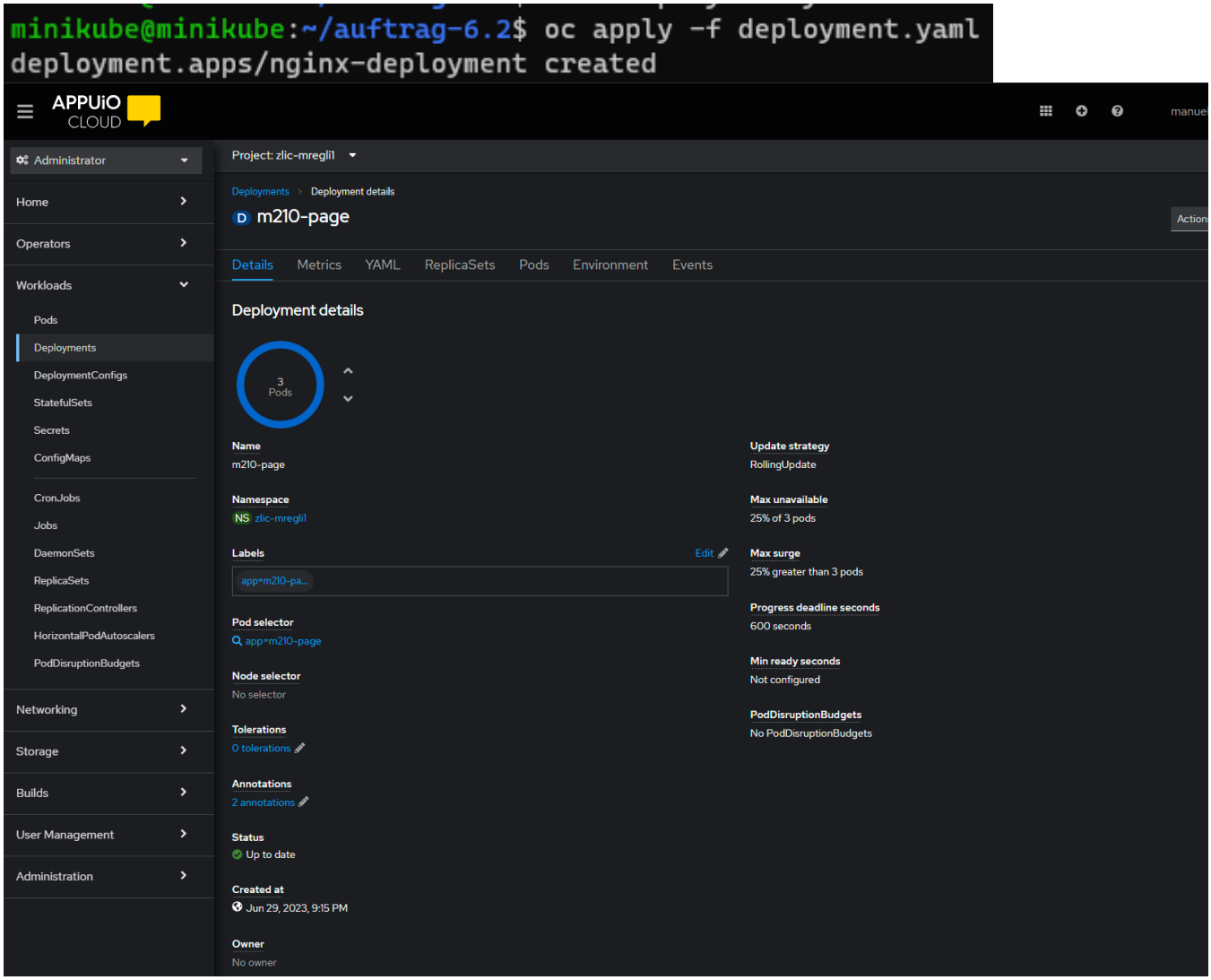
```
minikube@minikube:~/auftrag-6.2$ docker push ghcr.io/masluse/html-page:v1
The push refers to repository [ghcr.io/masluse/html-page]
088024538e55: Pushed
5f70bf18a086: Pushed
aefa4ac94b0d: Pushed
2957d90a4b00: Pushed
27e29e7dc201: Pushed
ef67c084d4c1: Pushed
b5cd47658bf8: Pushed
5f3343bb4bed: Pushed
86dc269365df: Pushed
ac4d164fef90: Pushed
v1: digest: sha256:4443a088a44e4116ef91a6348951eaa209d070f0f97d9c81c16ec6b785273d53 size: 2
```

4. Erstellen Sie ein Deployment für den Container, wie Sie es in Auftrag 4.2 gelernt haben. Und applizieren Sie dieses:

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: m210-page
labels:
    app: m210-page
spec:
replicas: 3
selector:
    matchLabels:
    app: m210-page
template:
    metadata:
    labels:
        app: m210-page
    spec:
    containers:
    - name: m210-page
        image: ghcr.io/masluse/html-page:v1
```

```
    ports:
    - containerPort: 8080
```

```
oc apply -f deployment.yaml
```




```
apiVersion: apps/v1
kind: Deployment
metadata:
name: m210-page
labels:
    app: m210-page
spec:
replicas: 3
selector:
    matchLabels:
    app: m210-page
template:
```

```
    metadata:
    labels:
        app: m210-page
    spec:
    containers:
    - name: m210-page
        image: ghcr.io/masluse/html-page:v1
        ports:
        - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
labels:
    app: m210-page
name: m210-page
spec:
ports:
- name: 8080-tcp
    port: 8080
    protocol: TCP
    targetPort: 8080
selector:
    app: m210-page
sessionAffinity: None
type: ClusterIP
---
apiVersion: route.openshift.io/v1
kind: Route
metadata:
labels:
    app: m210-page
name: m210-page
spec:
port:
    targetPort: 8080-tcp
to:
    kind: Service
    name: m210-page
tls:
```

```
    termination: edge
    insecureEdgeTerminationPolicy: Redirect
```

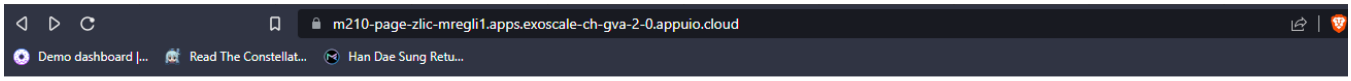
```
oc apply -f deployment.yaml
```

```
minikube@minikube:~/auftrag-6.2$ oc apply -f deployment.yaml
deployment.apps/nginx-deployment unchanged
service/m210-page created
route.route.openshift.io/m210-page created
```

5. Rufen Sie die erstellte Route ab und testen Sie die Webseite:

```
oc get routes
```

https://m210-page-zlic-mregli1.apps.exoscale-ch-gva-2-0.appuio.cloud/



Auftrag 6.2: HTML-Seite auf OpenShift deployen