CLC3 Projekt 🎶

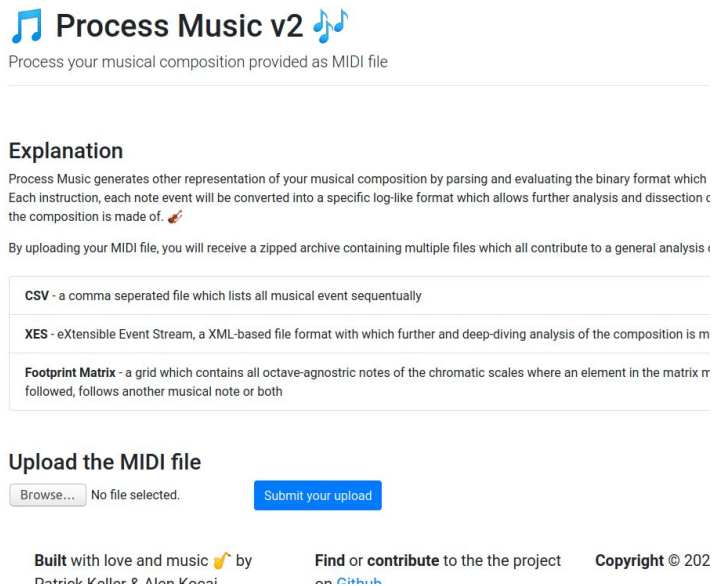
Cloud Process Music - Process your compositions in the cloud

Inhalt

- Cloud Process Music
- WebApp - Architektur
- Cloud Technologien
- Deployment Pipeline
- Kubernetes Konfiguration

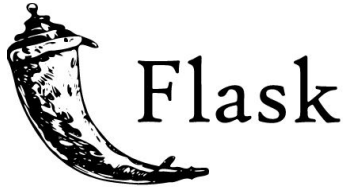
Cloud Process Music

- Frontend zum “Minen” von MIDI - Files
- Upload: MIDI → Download: Zip
- Konvertierung übernimmt Backend
- Schlichte Funktionalität



WebApp - Architektur

- Frontend mittels SPA (React.JS)
- Backend mittels Python (Flask)
- Redis zum Caching
- Docker als Container-Lösung

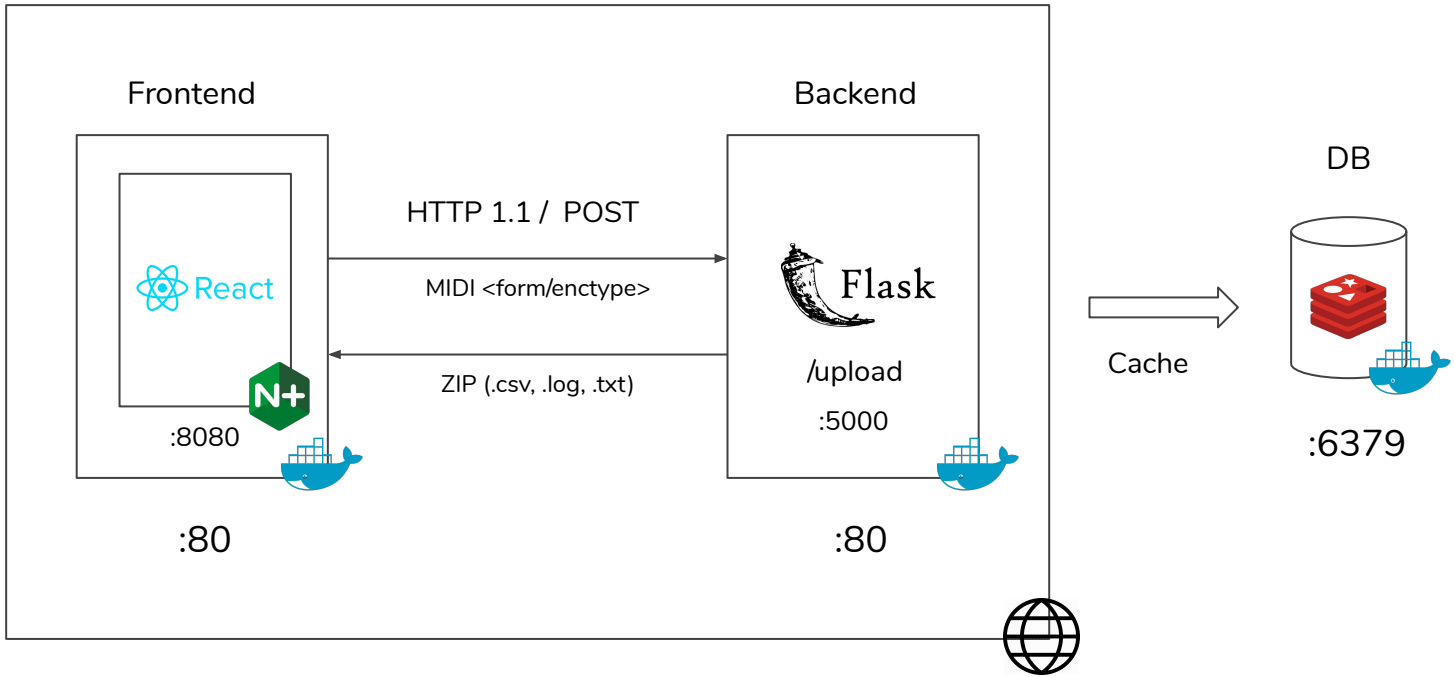


WebApp - Architektur II

- Frontend via Multi-Stage Docker Image
  - Nginx mit Port 8080
- Backend als Python Image
  - Flask mit Port 5000
- Redis Base Image
  - Port 6379

```
### [ Build stage ]====================================

FROM node:12.2.0-alpine as build

WORKDIR /app
COPY . /app

RUN npm install --silent
RUN npm run build

### [ Run stage ]=====================================

FROM nginx:1.16.0-alpine as run

LABEL maintainer "Alen Kocaj <alen.kocaj@posteo.at>"

RUN mkdir /etc/nginx/html

COPY --from=build /app/build /etc/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf

CMD ["nginx", "-g", "daemon off;"]
```

Öffentlich (auch du kannst zugreifen)



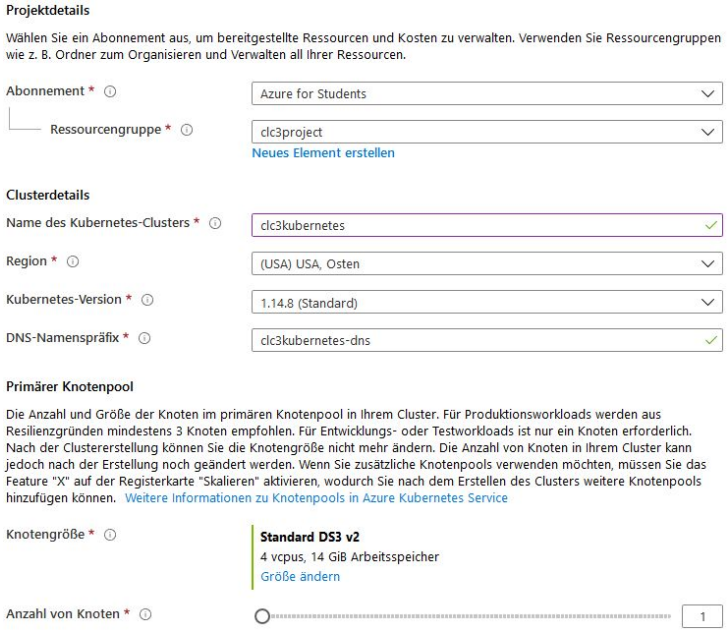
Cloud Technologien

- AWS EKS
  - fully-managed Kubernetes service (PaaS)
  - Provisionierung mittels `eksctl`
  - Config definiert Instanzen
    - 3 Nodegruppen mit je 2 `**m5.large**` VMs
    - AWS EC2 General Purpose
  - Lokale Orchestrierung mittels `kubectl`

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: process-music-cluster
  region: eu-central-1

nodeGroups:
  - name: ng-1
    instanceType: m5.large
    desiredCapacity: 2
    ssh:
        publicKeyPath: /home/alen/go/sr
  - name: ng-2
    instanceType: m5.large
    desiredCapacity: 2
    ssh:
        publicKeyPath: /home/alen/go/sr
  - name: ng-3
    instanceType: m5.large
```

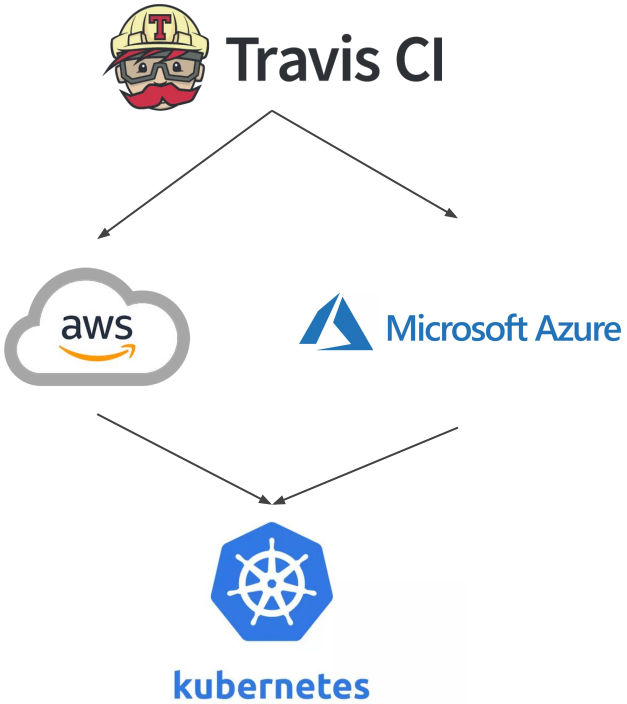
Cloud Technologien II

- Microsoft Azure
  - fully-managed Kubernetes service (PaaS)
  - Provisionierung über Portal
    - 1 Node
    - DS3 v2 (4vcpu, 14GB RAM)
  - Lokale Orchestrierung mittels `kubectl`



Deployment Pipeline

- CI / CD mittels Travis CI
- Erstellung von Docker Images
  - Artefakte pushed auf DockerHub
- Deploy / Update auf AWS & Azure
  - Kubernetes als Orchestierungslösung



Deployment Pipeline II

```
install:
  - mkdir -p $HOME/aws-sdk
  - pip install awscli --upgrade --user --cache-dir $HOME/.cache/pip
  - bash install.sh

services:
  - docker

before_script:
  - mkdir "$HOME/.aws"
  - echo "[default]" > "$HOME/.aws/credentials"
  - echo "aws_access_key_id=${AWS_KEY}" >> "$HOME/.aws/credentials"
  - echo "aws_secret_access_key=${AWS_SECRET}" >> "$HOME/.aws/credentials"

  - echo "[default]" > "$HOME/.aws/config"
  - echo "region=${AWS_REGION}" >> "$HOME/.aws/config"
  - echo "output=${AWS_OUTPUT}" >> "$HOME/.aws/config"

  - export PATH=$PATH:$HOME/aws-sdk
  - aws eks list-clusters
```

Schreib Konfig. manuell via Travis Envs und logge auf AWS

```
#!/bin/bash

curl -o "$HOME/aws-sdk/aws-iam-authenticator" https://amazon-eks.s3-us-
aws-iam-authenticator
chmod +x "$HOME/aws-sdk/aws-iam-authenticator"

if [ ! -f "$HOME/aws-sdk/eksctl" ]; then
    curl --silent --location "https://github.com/weaveworks/eksctl/rele
    _amd64.tar.gz" | tar xz -C "$HOME/aws-sdk"
fi

curl -o "$HOME/aws-sdk/kubectl" https://amazon-eks.s3-us-west-2.amazona
chmod +x "$HOME/aws-sdk/kubectl"

curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

Installiere AWS & Azure CLITools

Deployment Pipeline III

Build Image

Push to Registry

Notify K8s

Login to Azure

```
script:
  - GIT_SHA="$(git rev-parse --short HEAD)"

  - docker build -f ./frontend/Dockerfile -t $AK_REG_USER/frontend:latest  -t $AK_REG_USER/frontend:$GIT_SHA ./frontend
  - docker build -f ./backend/Dockerfile -t $AK_REG_USER/backend:latest -t $AK_REG_USER/backend:$GIT_SHA ./backend
  - docker build -f ./db/Dockerfile -t $AK_REG_USER/db:latest -t $AK_REG_USER/db:$GIT_SHA ./db

  - echo "$AK_REG_PW" | docker login --username $AK_REG_USER --password-stdin
  - docker push $AK_REG_USER/frontend:latest
  - docker push $AK_REG_USER/frontend:$GIT_SHA

  - docker push $AK_REG_USER/backend:latest
  - docker push $AK_REG_USER/backend:$GIT_SHA

  - docker push $AK_REG_USER/db:latest
  - docker push $AK_REG_USER/db:$GIT_SHA

  - export PATH=$PATH:$HOME/aws-sdk
  - kubectl apply -f kube/frontend-k8s.yaml --force
  - kubectl apply -f kube/backend-k8s.yaml --force
  - kubectl apply -f kube/redis-k8s.yaml --force

  - az login --service-principal -u ${AZ_APPID} --password ${AZ_PASSWORD} --tenant ${AZ_TENANT}
  - az aks get-credentials --resource-group ${AZ_RES_GROUP} --name ${AZ_CLUSTER}
  - kubectl apply -f kube/frontend-k8s.yaml --force
  - kubectl apply -f kube/backend-k8s.yaml --force
  - kubectl apply -f kube/redis-k8s.yaml --force
```

Kubernetes Config

- Frontend & Backend als Service & Deployment
  - Service Type: LoadBalancer
  - Static IP für den Zugriff von außen
    - Static URL via AWS

- Redis ohne LoadBalancer (Type: ClusterIP)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend
  labels:
    app: backend
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 5000
  selector:
    app: backend
  type: LoadBalancer
```

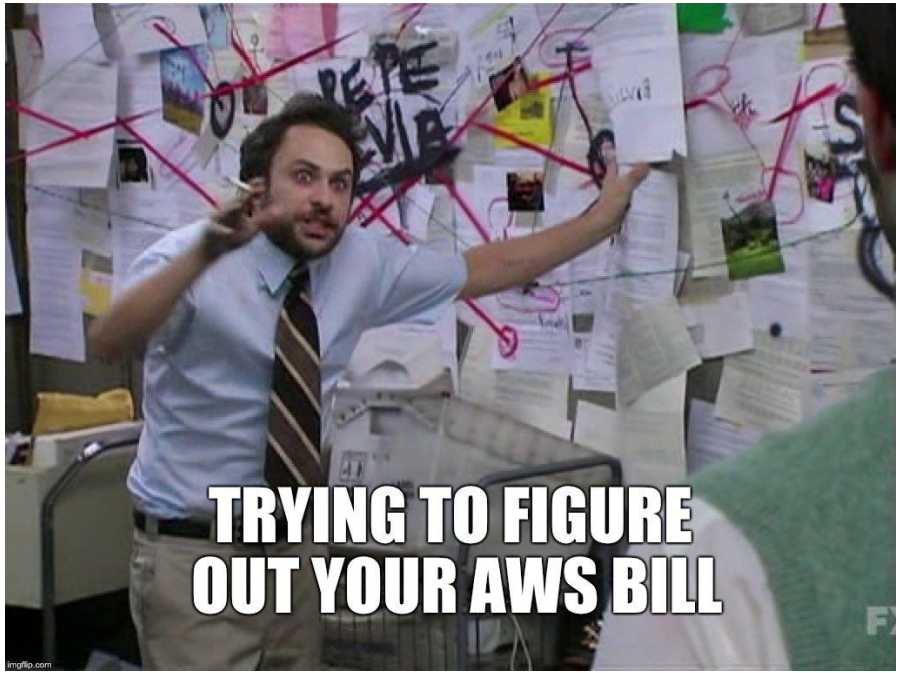
```
backend      LoadBalancer   10.100.12.111    a17646bed4bf511eabe96068898d0fcb-983216779.eu-central-1.elb.amazonaws.com
frontend     LoadBalancer   10.100.227.182   a16acd7964bf511eabe96068898d0fcb-57024942.eu-central-1.elb.amazonaws.com
```

🎶 DEMO 🎶

Lesson learned

- Windows vs. Linux
  - Pfade
  - OS
- AWS vs. Azure
  - Azure Login
- Container Kommunikation
  - Backend & Redis

Lesson learned II



Vielen Dank für die Aufmerksamkeit!