IT-PS.
Data Science

From Docker to OpenShift

What we have learned while deploying our first application

IT-Power Services GmbH

Clemens Zauchner

Contents

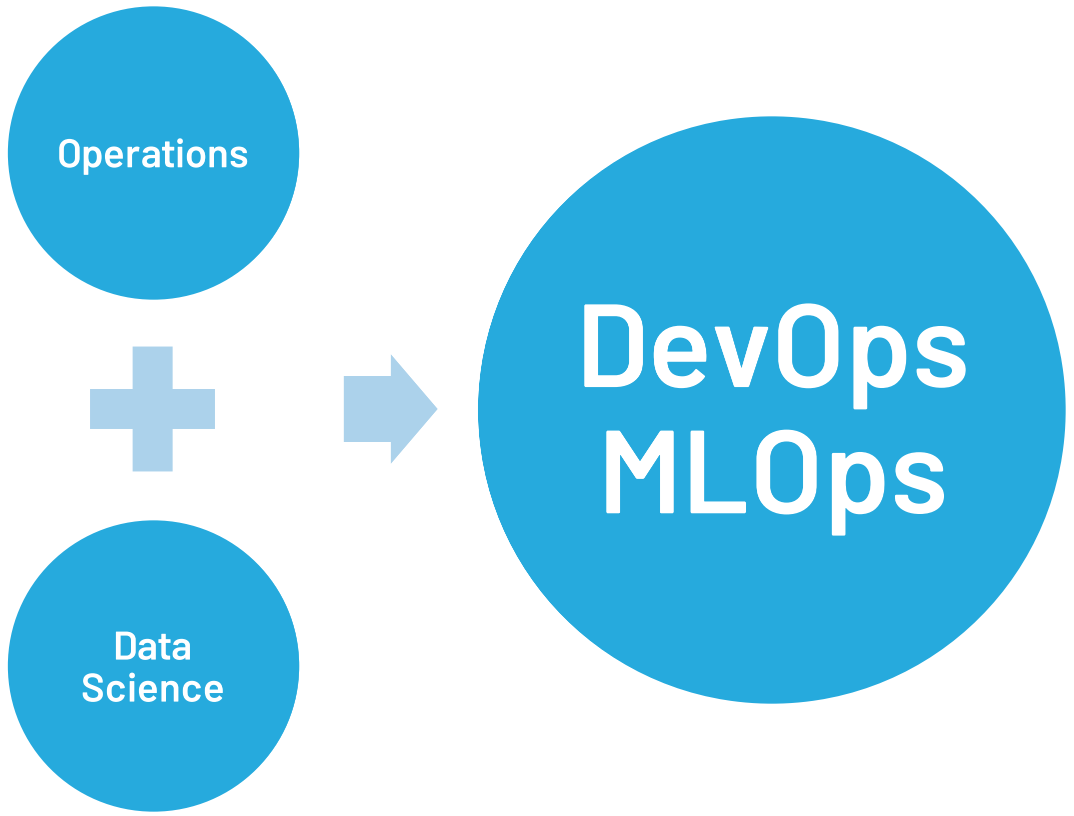
What you can expect from this talk

- What did we deploy?
- How does the deployment look like using Docker Compose / Swarm?
- How did we move from Docker to OpenShift?
- What were the main challenges and how did we deal with them?

IT-Power Services

Bridging the gap between operations and applications

- Operations experts
  - Power-house with high expertise IBM i and Linux systems
  - Private cloud provider with multiple data centres in Austria
  - Services around public cloud
- Data Science and Software Engineering
- DevOps / MLOps
  - CI/CD pipelines
  - Docker, Podman, OpenShift

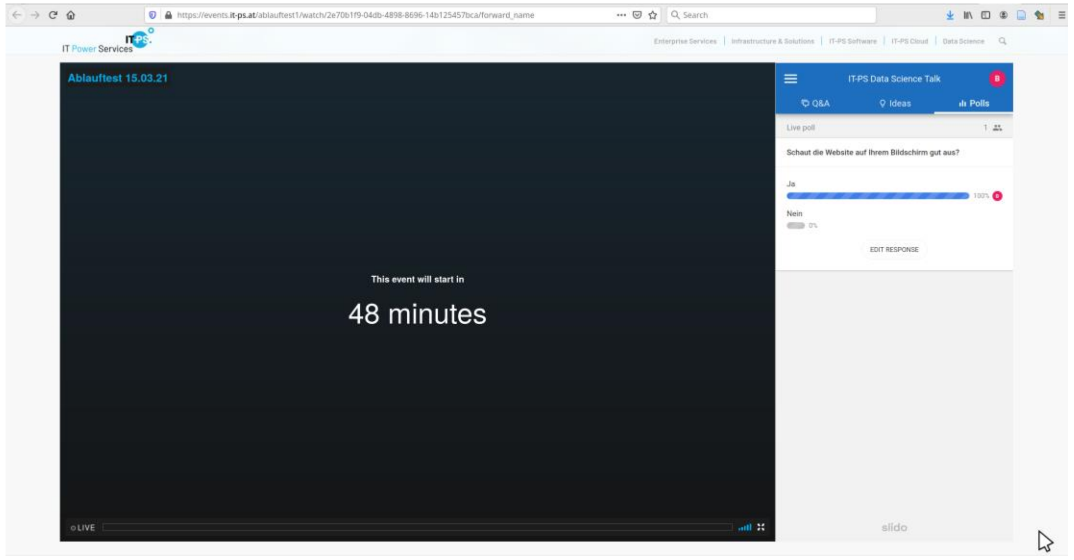


Background

Background: manage an online event

IT-PS Data Science Talk 2021

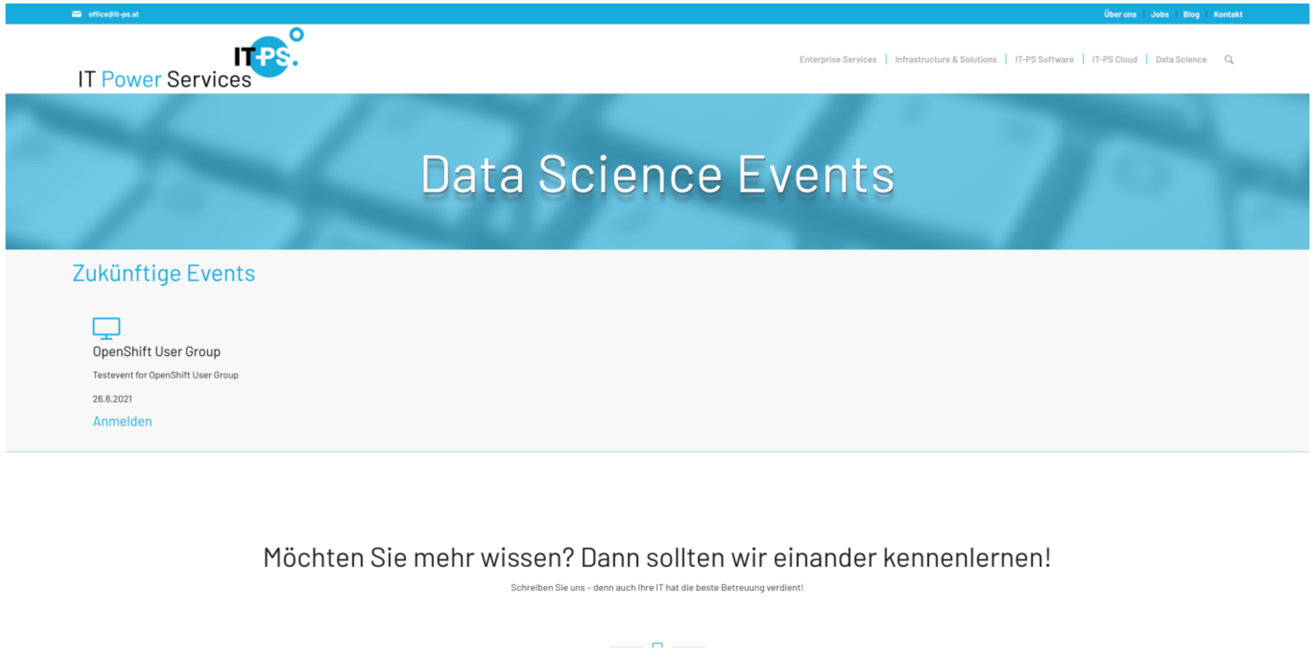
- Manage invitations and participants
  - Including email templates
- Create a virtual "stage"
  - Show video and Q&A side by side



Demo: online event application

Landing page

- The landing page lists past, current and upcoming events



office@it-ps.at
Über uns | Jobs | Blog | Kontakt
IT Power Services
Enterprise Services | Infrastructure & Solutions | IT-PS Software | IT-PS Cloud | Data Science

Data Science Events

Zukünftige Events

OpenShift User Group

Testevent for OpenShift User Group

26.6.2021

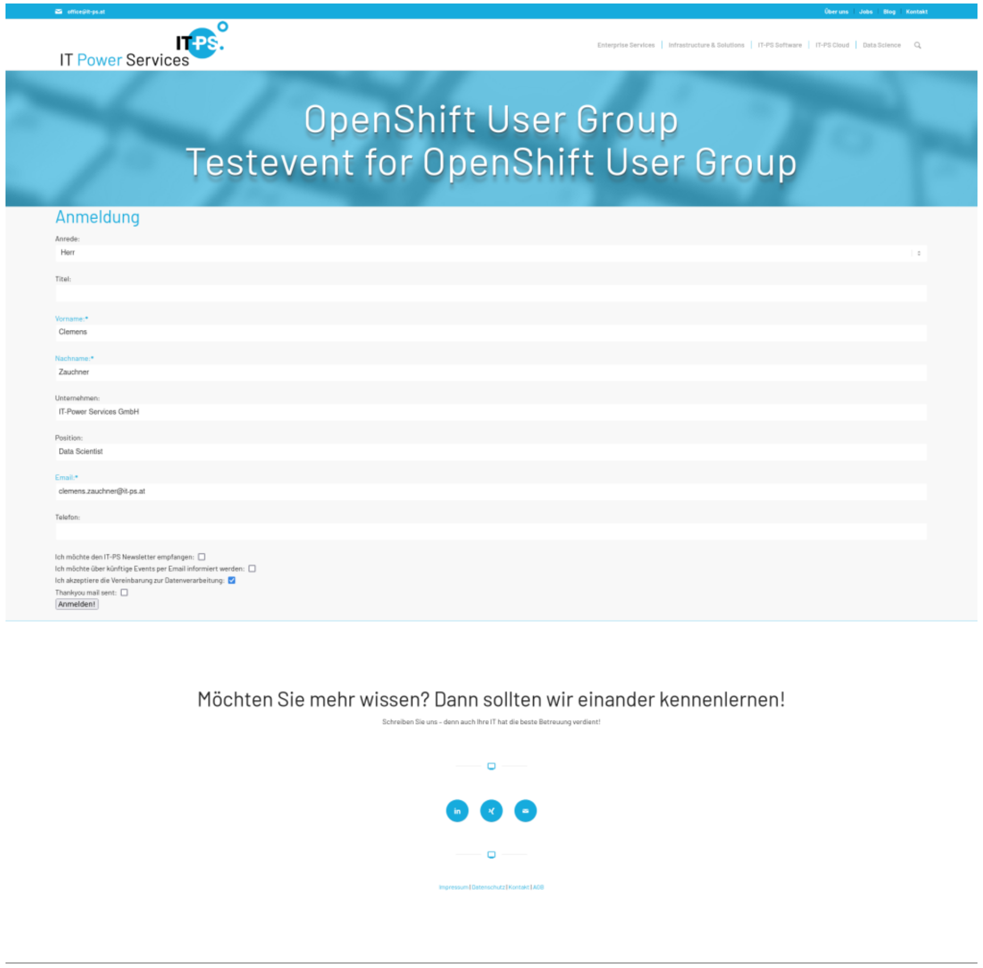
Anmelden

Möchten Sie mehr wissen? Dann sollten wir einander kennenlernen!

Schreiben Sie uns – denn auch Ihre IT hat die beste Betreuung verdient!

Demo: online event application

Registration form



office@it-ps.at
Über uns | Jobs | Blog | Kontakt
IT Power Services
Enterprise Services | Infrastructure & Solutions | IT-PS Software | IT-PS Cloud | Data Science

OpenShift User Group
Testevent for OpenShift User Group

Anmeldung

Anrede: Herr

Titel:

Vorname:* Clemens

Nachname:* Zauchner

Unternehmen: IT-Power Services GmbH

Position: Data Scientist

Email:* clemens.zauchner@it-ps.at

Telefon:

Ich möchte den IT-PS Newsletter empfangen: ☐
Ich möchte über künftige Events per Email informiert werden: ☐
Ich akzeptiere die Vereinbarung zur Datenverarbeitung: ☑
Thankyou mail sent: ☐

Anmelden!

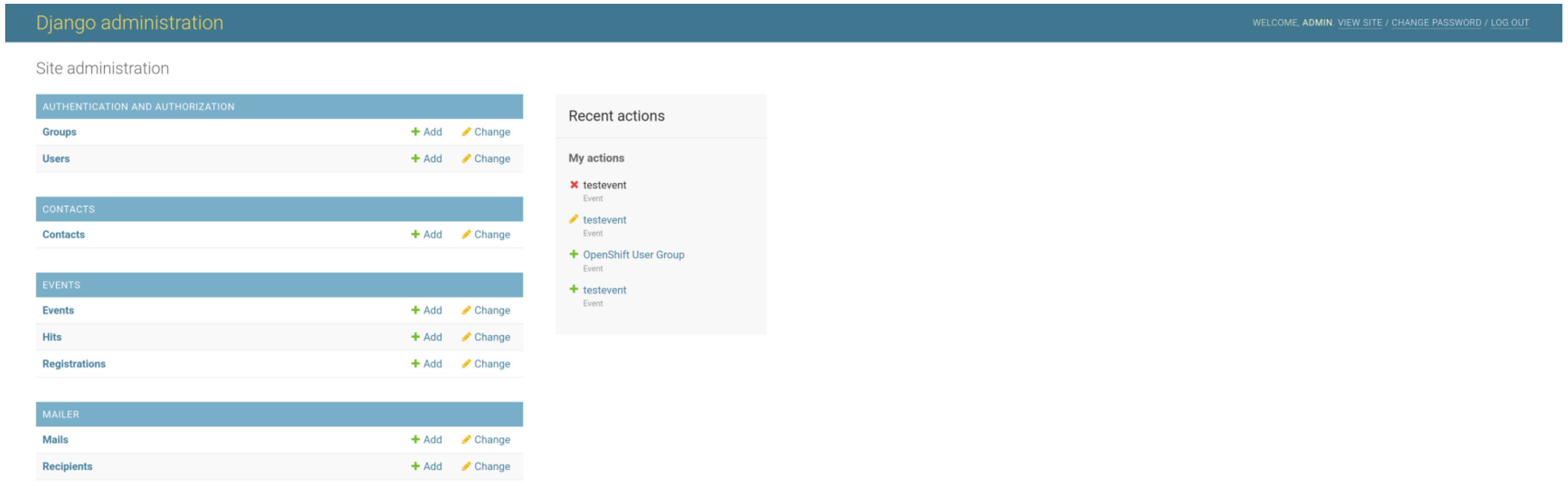
Möchten Sie mehr wissen? Dann sollten wir einander kennenlernen!

Schreiben Sie uns – denn auch Ihre IT hat die beste Betreuung verdient!

Demo: online event application

Django admin area

- Django offers an admin page
  - Manage event details
  - Manage registrations
  - Send mails



Django administration

WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

CONTACTS		
Contacts	+ Add	Change

EVENTS		
Events	+ Add	Change
Hits	+ Add	Change
Registrations	+ Add	Change

MAILER		
Mails	+ Add	Change
Recipients	+ Add	Change

Recent actions

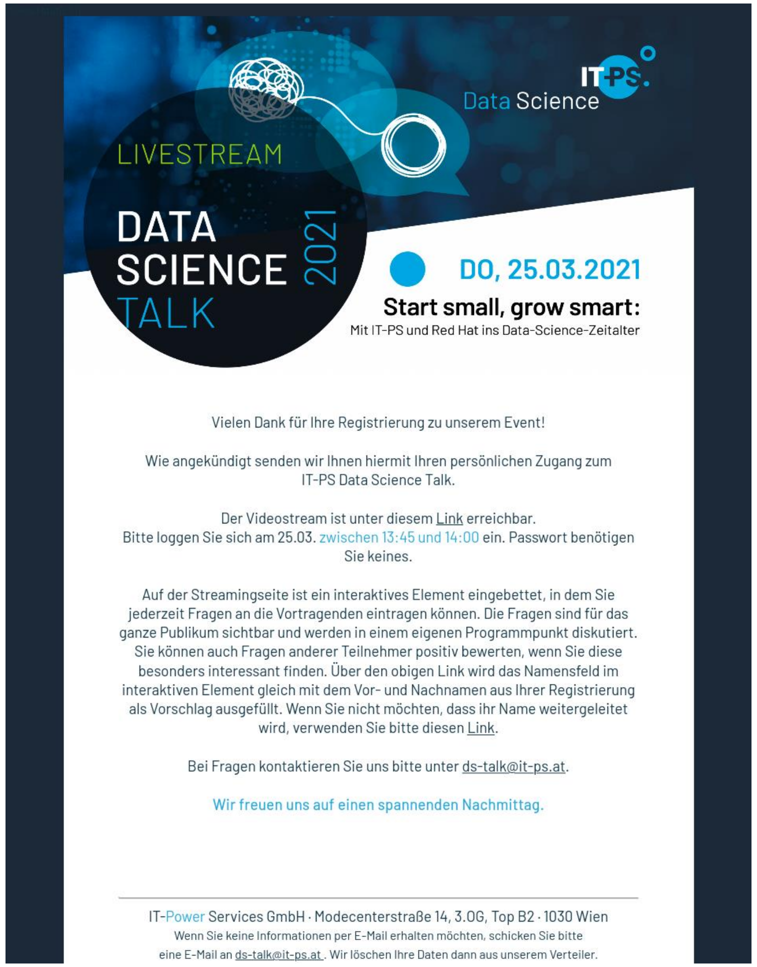
My actions

- ✖ testevent
  Event
- ✎ testevent
  Event
- + OpenShift User Group
  Event
- + testevent
  Event

Demo: online event application

Mailer: create and send emails

- HTML email templates
- Customisation via admin page
- Distribution via admin page
- Unsubscribe action via embedded link



DO, 25.03.2021

Start small, grow smart:

Mit IT-PS und Red Hat ins Data-Science-Zeitalter

Vielen Dank für Ihre Registrierung zu unserem Event!

Wie angekündigt senden wir Ihnen hiermit Ihren persönlichen Zugang zum IT-PS Data Science Talk.

Der Videostream ist unter diesem Link erreichbar.
Bitte loggen Sie sich am 25.03. zwischen 13:45 und 14:00 ein. Passwort benötigen Sie keines.

Auf der Streamingseite ist ein interaktives Element eingebettet, in dem Sie jederzeit Fragen an die Vortragenden eintragen können. Die Fragen sind für das ganze Publikum sichtbar und werden in einem eigenen Programmpunkt diskutiert. Sie können auch Fragen anderer Teilnehmer positiv bewerten, wenn Sie diese besonders interessant finden. Über den obigen Link wird das Namensfeld im interaktiven Element gleich mit dem Vor- und Nachnamen aus Ihrer Registrierung als Vorschlag ausgefüllt. Wenn Sie nicht möchten, dass ihr Name weitergeleitet wird, verwenden Sie bitte diesen Link.

Bei Fragen kontaktieren Sie uns bitte unter ds-talk@it-ps.at.

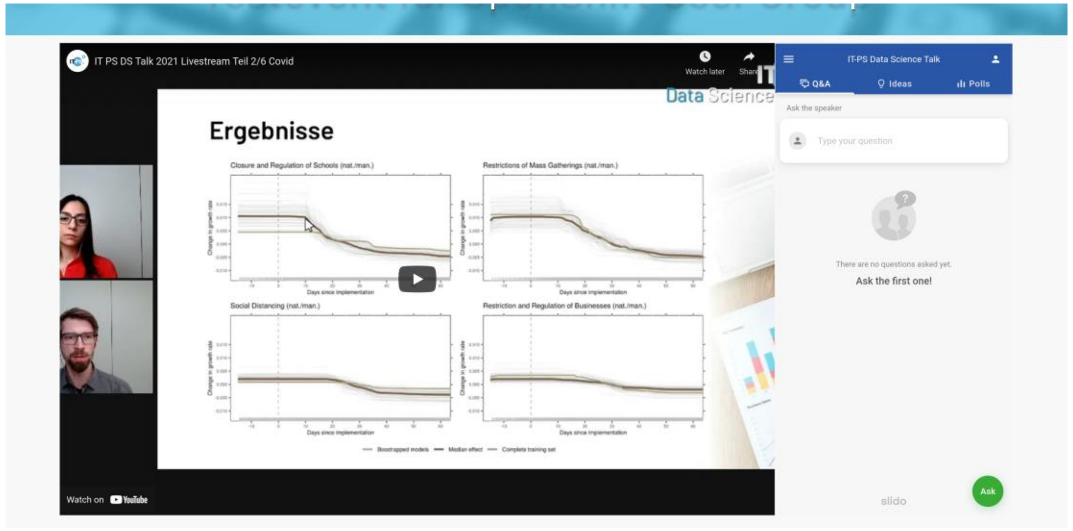
Wir freuen uns auf einen spannenden Nachmittag.

IT-Power Services GmbH · Modecenterstraße 14, 3.OG, Top B2 · 1030 Wien
Wenn Sie keine Informationen per E-Mail erhalten möchten, schicken Sie bitte eine E-Mail an ds-talk@it-ps.at . Wir löschen Ihre Daten dann aus unserem Verteiler.

Demo: online event application

Watch page

- The "watch page" is available for registered users only
- Embeds
  - YouTube / Vimeo iframe
  - Slido Q&A



Tech stack: high level overview

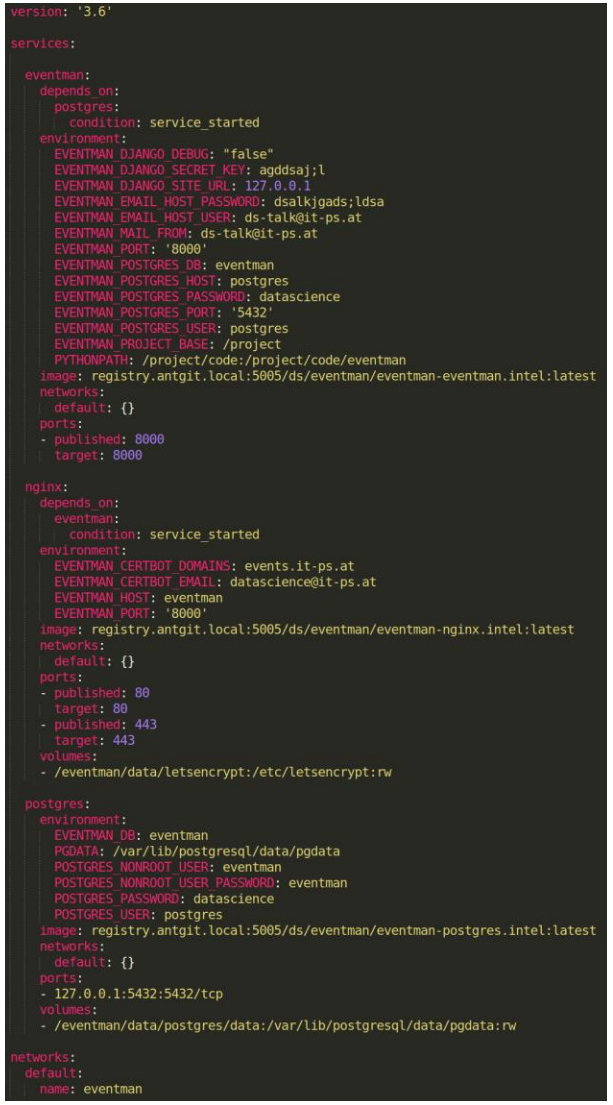
Django application with PosgreSQL database behind Nginx

- Django
  - Python based web framework
  - Model-template-view pattern
- PosgreSQL DB
  - Stores all relevant information for event
  - Managed by Django
- Nginx
  - Webserver
- UWSGI
  - Web Server Gateway Interface
  - Link between webserver and python

Containerisation and Orchestration

Docker Compose / Swarm

- 4 Services
  - Application init
  - Postgres
  - Eventman (Python Django Application)
  - Nginx
- 1 Network
- Some services expose ports on the host
- Some services read / write data on disk

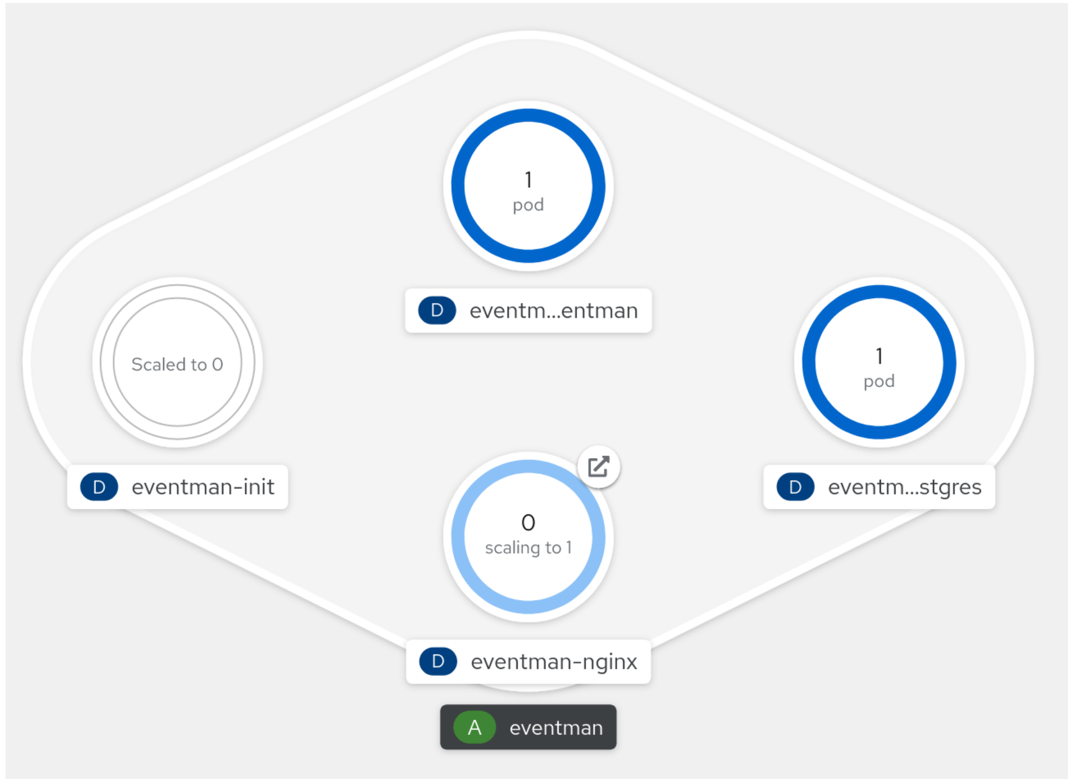


```yaml
version: '3.6'

services:

  eventman:
    depends_on:
      postgres:
        condition: service_started
    environment:
      EVENTMAN_DJANGO_DEBUG: "false"
      EVENTMAN_DJANGO_SECRET_KEY: agddsaj;l
      EVENTMAN_DJANGO_SITE_URL: 127.0.0.1
      EVENTMAN_EMAIL_HOST_PASSWORD: dsalkjgads;ldsa
      EVENTMAN_EMAIL_HOST_USER: ds-talk@it-ps.at
      EVENTMAN_MAIL_FROM: ds-talk@it-ps.at
      EVENTMAN_PORT: '8000'
      EVENTMAN_POSTGRES_DB: eventman
      EVENTMAN_POSTGRES_HOST: postgres
      EVENTMAN_POSTGRES_PASSWORD: datascience
      EVENTMAN_POSTGRES_PORT: '5432'
      EVENTMAN_POSTGRES_USER: postgres
      EVENTMAN_PROJECT_BASE: /project
      PYTHONPATH: /project/code:/project/code/eventman
    image: registry.antgit.local:5005/ds/eventman/eventman-eventman.intel:latest
    networks:
      default: {}
    ports:
    - published: 8000
      target: 8000

  nginx:
    depends_on:
      eventman:
        condition: service_started
    environment:
      EVENTMAN_CERTBOT_DOMAINS: events.it-ps.at
      EVENTMAN_CERTBOT_EMAIL: datascience@it-ps.at
      EVENTMAN_HOST: eventman
      EVENTMAN_PORT: '8000'
    image: registry.antgit.local:5005/ds/eventman/eventman-nginx.intel:latest
    networks:
      default: {}
    ports:
    - published: 80
      target: 80
    - published: 443
      target: 443
    volumes:
    - /eventman/data/letsencrypt:/etc/letsencrypt:rw

  postgres:
    environment:
      EVENTMAN_DB: eventman
      PGDATA: /var/lib/postgresql/data/pgdata
      POSTGRES_NONROOT_USER: eventman
      POSTGRES_NONROOT_USER_PASSWORD: eventman
      POSTGRES_PASSWORD: datascience
      POSTGRES_USER: postgres
    image: registry.antgit.local:5005/ds/eventman/eventman-postgres.intel:latest
    networks:
      default: {}
    ports:
    - 127.0.0.1:5432:5432/tcp
    volumes:
    - /eventman/data/postgres/data:/var/lib/postgresql/data/pgdata:rw

networks:
  default:
    name: eventman
```

The deployment in OpenShift

Topology view from OpenShift

- 3 deployments
- Init deployment
  - Should have been a pod
  - See details later
- NGINX exposes service via route



Moving from Docker to OpenShift

Kompose

Moving from yml to yml

- Kompose is a tool to help users who are familiar with docker-compose move to Kubernetes
  - OpenShift can be selected as provider
- Provides a great starting point to generate yml for all components

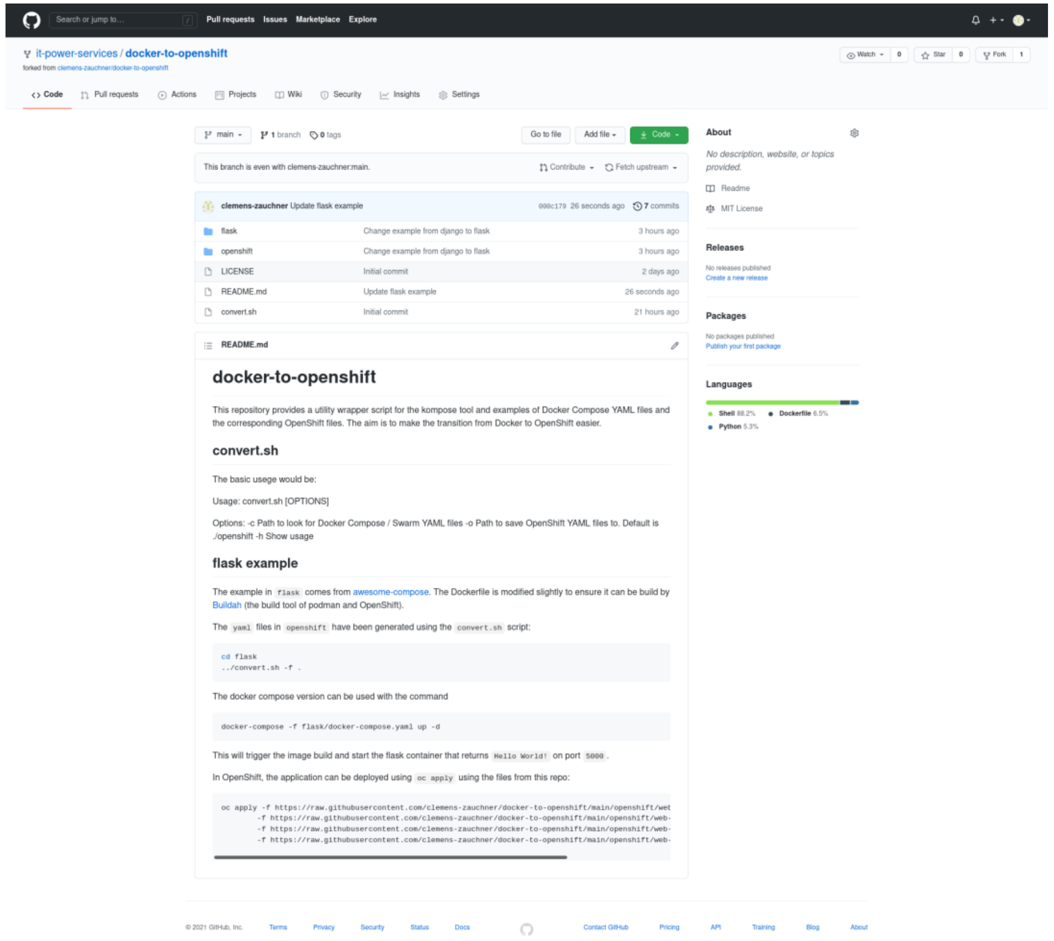
Basic usage:

```
kompose convert \
--out=path/to/out/dir \
--provider=openshift \
-f=docker-compose.yml
```

Single service example

Code samples

- All code samples can be found in our GitHub
- https://github.com/it-power-services/docker-to-openshift



it-power-services / docker-to-openshift
forked from clemens-zauchner/docker-to-openshift

This branch is even with clemens-zauchner:main.

clemens-zauchner Update flask example 000c179 26 seconds ago 7 commits

flask	Change example from django to flask	3 hours ago
openshift	Change example from django to flask	3 hours ago
LICENSE	Initial commit	2 days ago
README.md	Update flask example	26 seconds ago
convert.sh	Initial commit	21 hours ago

About

No description, website, or topics provided.

Readme

MIT License

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

● Shell 88.2% ● Dockerfile 6.5% ● Python 5.3%

README.md

docker-to-openshift

This repository provides a utility wrapper script for the kompose tool and examples of Docker Compose YAML files and the corresponding OpenShift files. The aim is to make the transition from Docker to OpenShift easier.

convert.sh

The basic usege would be:

Usage: convert.sh [OPTIONS]

Options: -c Path to look for Docker Compose / Swarm YAML files -o Path to save OpenShift YAML files to. Default is ./openshift -h Show usage

flask example

The example in `flask` comes from awesome-compose. The Dockerfile is modified slightly to ensure it can be build by Buildah (the build tool of podman and OpenShift).

The `yaml` files in `openshift` have been generated using the `convert.sh` script:

```
cd flask
../convert.sh -f .
```

The docker compose version can be used with the command

```
docker-compose -f flask/docker-compose.yaml up -d
```

This will trigger the image build and start the flask container that returns `Hello World!` on port `5000`.

In OpenShift, the application can be deployed using `oc apply` using the files from this repo:

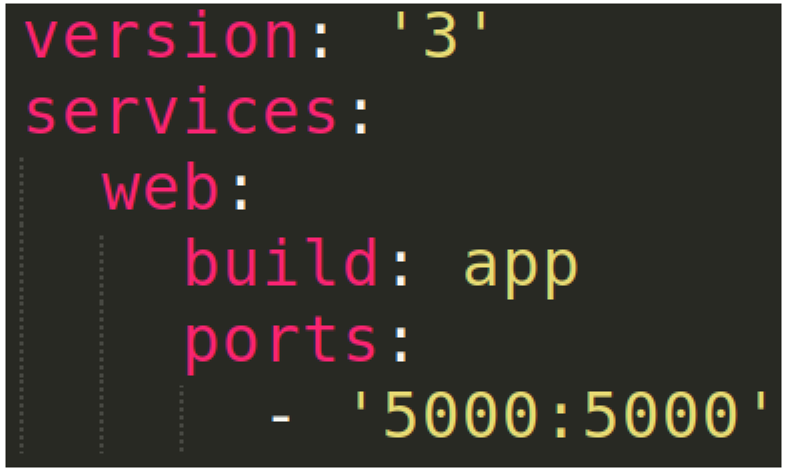
```
oc apply -f https://raw.githubusercontent.com/clemens-zauchner/docker-to-openshift/main/openshift/wet
        -f https://raw.githubusercontent.com/clemens-zauchner/docker-to-openshift/main/openshift/web-
        -f https://raw.githubusercontent.com/clemens-zauchner/docker-to-openshift/main/openshift/web-
        -f https://raw.githubusercontent.com/clemens-zauchner/docker-to-openshift/main/openshift/web-
```

© 2021 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

Single service example

Python Flask API

- Docker Compose includes instructions to
  - Create a service called "web"
  - Image is built from "app" context
  - Exposes port 5000
- The Flask API exposes one Endpoint that returns the string "Hello World!"

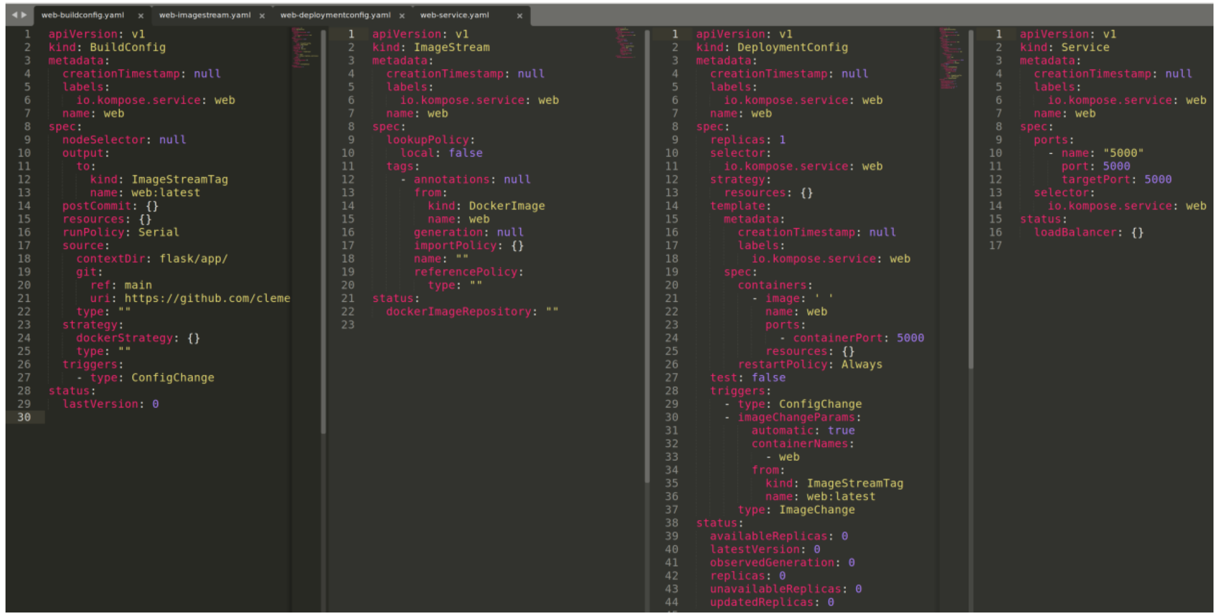


```yaml
version: '3'
services:
  web:
    build: app
    ports:
      - '5000:5000'
```

Single service example

Python Flask API

- Kompose translates that to
  - Build config
  - Imagestream
  - Deployment config
  - Service
- If you want to expose the service outside the cluster this has to be configured manually by creating a route
- The application can be deployed using oc apply



```yaml
apiVersion: v1
kind: BuildConfig
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: web
  name: web
spec:
  nodeSelector: null
  output:
    to:
      kind: ImageStreamTag
      name: web:latest
  postCommit: {}
  resources: {}
  runPolicy: Serial
  source:
    contextDir: flask/app/
    git:
      ref: main
      uri: https://github.com/cleme
    type: ""
  strategy:
    dockerStrategy: {}
    type: ""
  triggers:
    - type: ConfigChange
status:
  lastVersion: 0
```

```yaml
apiVersion: v1
kind: ImageStream
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: web
  name: web
spec:
  lookupPolicy:
    local: false
  tags:
    - annotations: null
      from:
        kind: DockerImage
        name: web
      generation: null
      importPolicy: {}
      name: ""
      referencePolicy:
        type: ""
status:
  dockerImageRepository: ""
```

```yaml
apiVersion: v1
kind: DeploymentConfig
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: web
  name: web
spec:
  replicas: 1
  selector:
    io.kompose.service: web
  strategy:
    resources: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: web
    spec:
      containers:
        - image: ' '
          name: web
          ports:
            - containerPort: 5000
          resources: {}
      restartPolicy: Always
  test: false
  triggers:
    - type: ConfigChange
    - imageChangeParams:
        automatic: true
        containerNames:
          - web
        from:
          kind: ImageStreamTag
          name: web:latest
      type: ImageChange
status:
  availableReplicas: 0
  latestVersion: 0
  observedGeneration: 0
  replicas: 0
  unavailableReplicas: 0
  updatedReplicas: 0
```

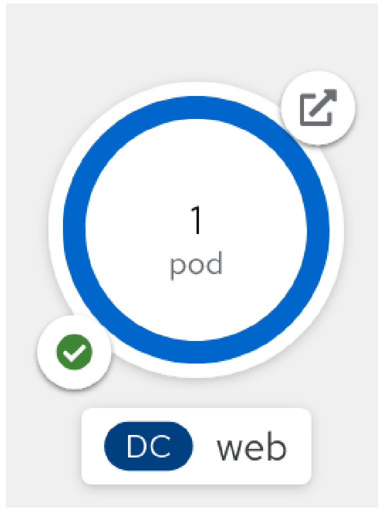
```yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: web
  name: web
spec:
  ports:
    - name: "5000"
      port: 5000
      targetPort: 5000
  selector:
    io.kompose.service: web
status:
  loadBalancer: {}
```

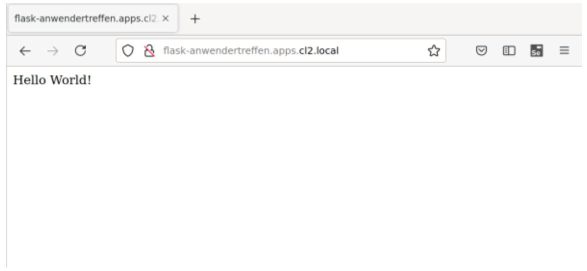
Single service example

Python Flask API

- The topology of the application is very simple

- The example returns the expected "Hello World!" string



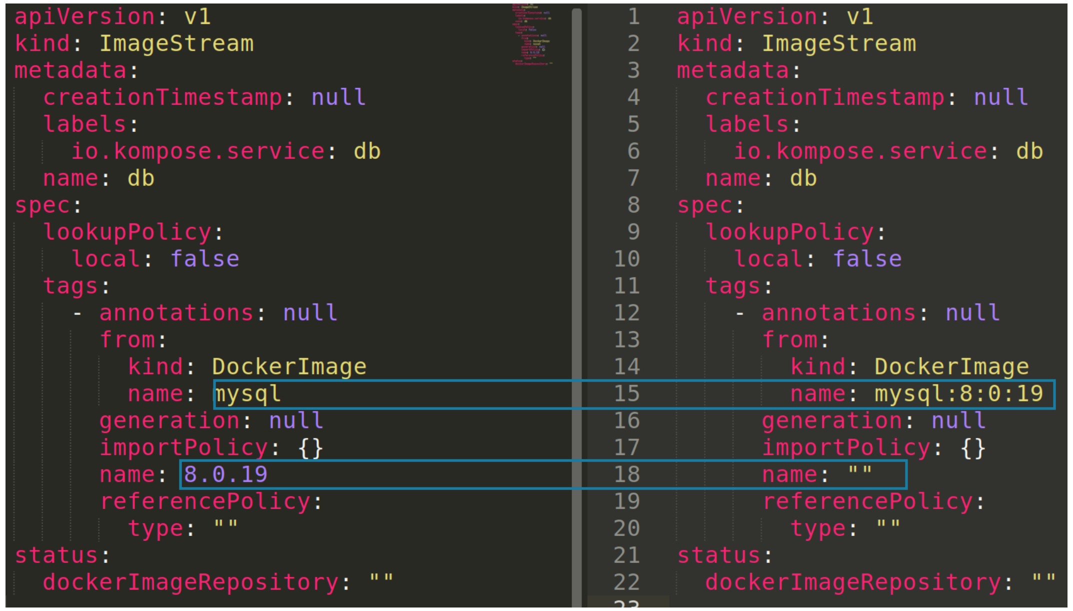

flask-anwendertreffen.apps.cl2

flask-anwendertreffen.apps.cl2.local

Hello World!

Disadvantages of using Kompose

- K8s or minikube required
- Translation is tricky, especially when concepts don't map 1:1
- Kubernetes not opinionated, many ways to do one thing
- Docker Compose files have to be very explicit
  - e.g. restart policy determines type
- The way image streams are created leads to unresolved images



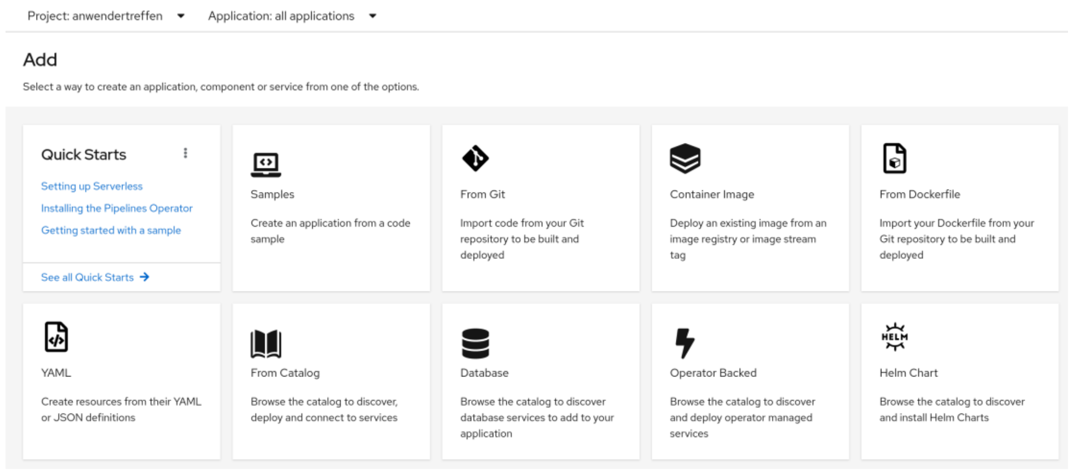
```yaml
apiVersion: v1
kind: ImageStream
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: db
  name: db
spec:
  lookupPolicy:
    local: false
  tags:
    - annotations: null
      from:
        kind: DockerImage
        name: mysql
      generation: null
      importPolicy: {}
      name: 8.0.19
      referencePolicy:
        type: ""
status:
  dockerImageRepository: ""
```

```yaml
apiVersion: v1
kind: ImageStream
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: db
  name: db
spec:
  lookupPolicy:
    local: false
  tags:
    - annotations: null
      from:
        kind: DockerImage
        name: mysql:8:0:19
      generation: null
      importPolicy: {}
      name: ""
      referencePolicy:
        type: ""
status:
  dockerImageRepository: ""
```

Using the OpenShift GUI

A very good starting point

- Many options
  - Deploy an image from a registry
  - Import repo, build and deploy
  - …



Project: anwendertreffen ▾ Application: all applications ▾

Add

Select a way to create an application, component or service from one of the options.

Quick Starts

Setting up Serverless

Installing the Pipelines Operator

Getting started with a sample

See all Quick Starts →

Samples

Create an application from a code sample

From Git

Import code from your Git repository to be built and deployed

Container Image

Deploy an existing image from an image registry or image stream tag

From Dockerfile

Import your Dockerfile from your Git repository to be built and deployed

YAML

Create resources from their YAML or JSON definitions

From Catalog

Browse the catalog to discover, deploy and connect to services

Database

Browse the catalog to discover database services to add to your application

Operator Backed

Browse the catalog to discover and deploy operator managed services

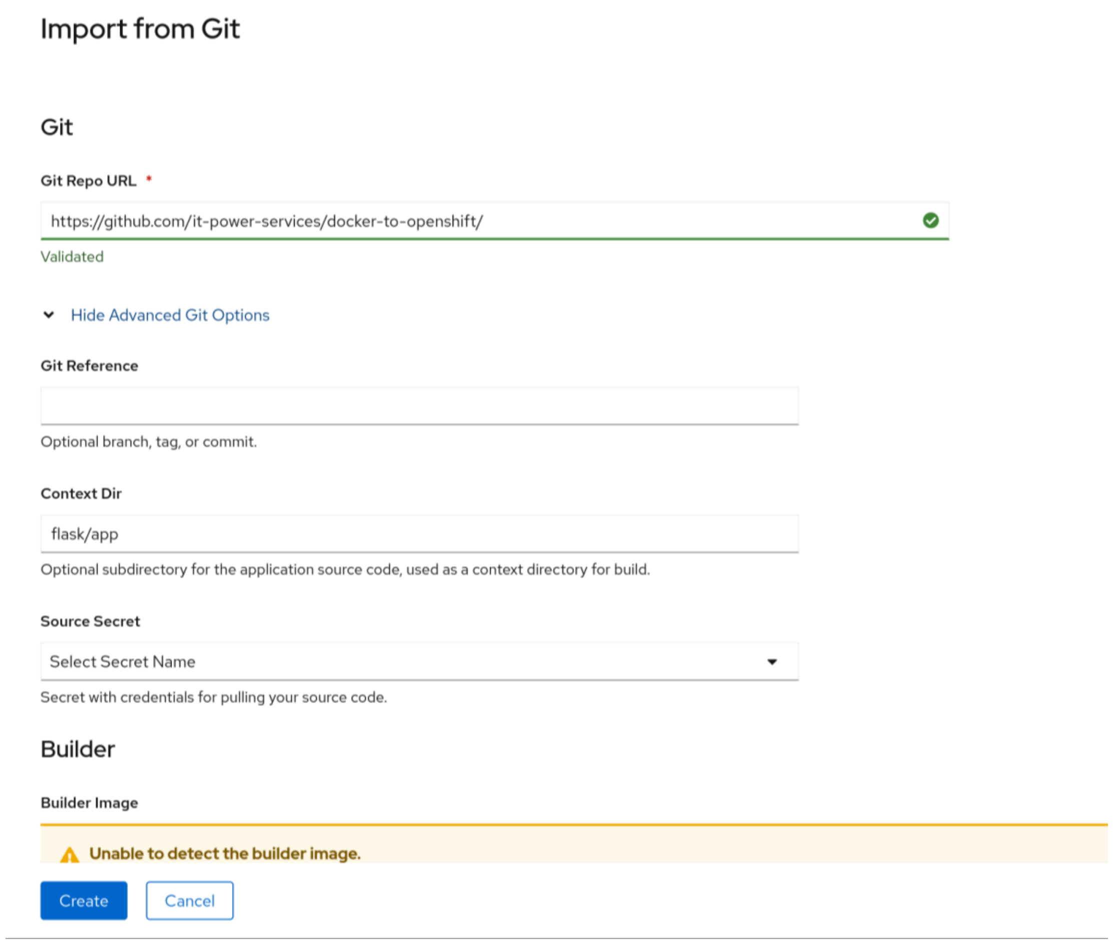
Helm Chart

Browse the catalog to discover and install Helm Charts

Using the OpenShift GUI

Import from Git

- Select the repo url
- Specify the build context
- Select a build image



Import from Git

Git

Git Repo URL *

https://github.com/it-power-services/docker-to-openshift/

Validated

Hide Advanced Git Options

Git Reference

Optional branch, tag, or commit.

Context Dir

flask/app

Optional subdirectory for the application source code, used as a context directory for build.

Source Secret

Select Secret Name

Secret with credentials for pulling your source code.

Builder

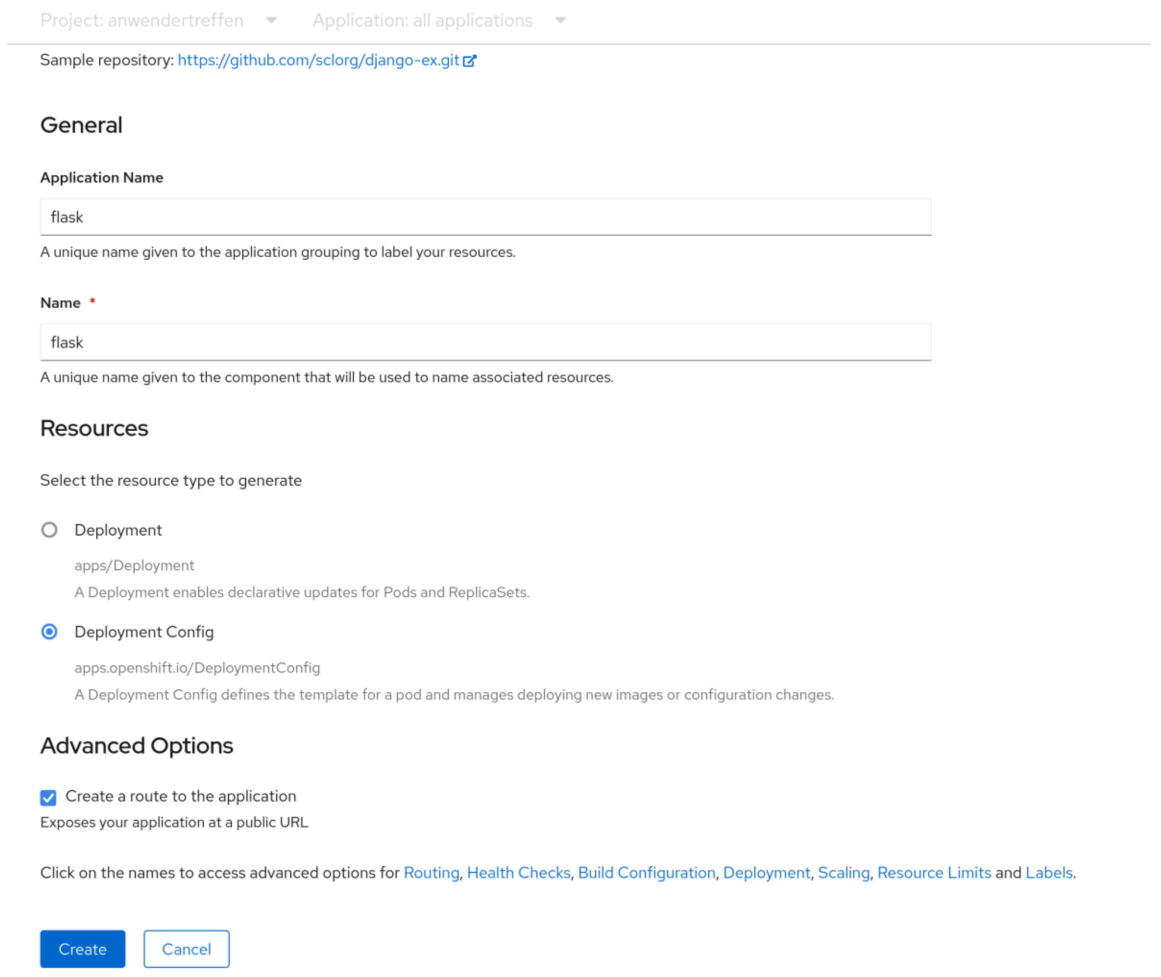
Builder Image

Unable to detect the builder image.

Using the OpenShift GUI

Import from Git

- Provide names for the application
- Choose deployment of deployment config
- Optionally create route to service



Project: anwendertreffen ▾ Application: all applications ▾

Sample repository: https://github.com/sclorg/django-ex.git

General

Application Name

flask

A unique name given to the application grouping to label your resources.

Name *

flask

A unique name given to the component that will be used to name associated resources.

Resources

Select the resource type to generate

( ) Deployment

apps/Deployment

A Deployment enables declarative updates for Pods and ReplicaSets.

(●) Deployment Config

apps.openshift.io/DeploymentConfig

A Deployment Config defines the template for a pod and manages deploying new images or configuration changes.

Advanced Options

[✓] Create a route to the application

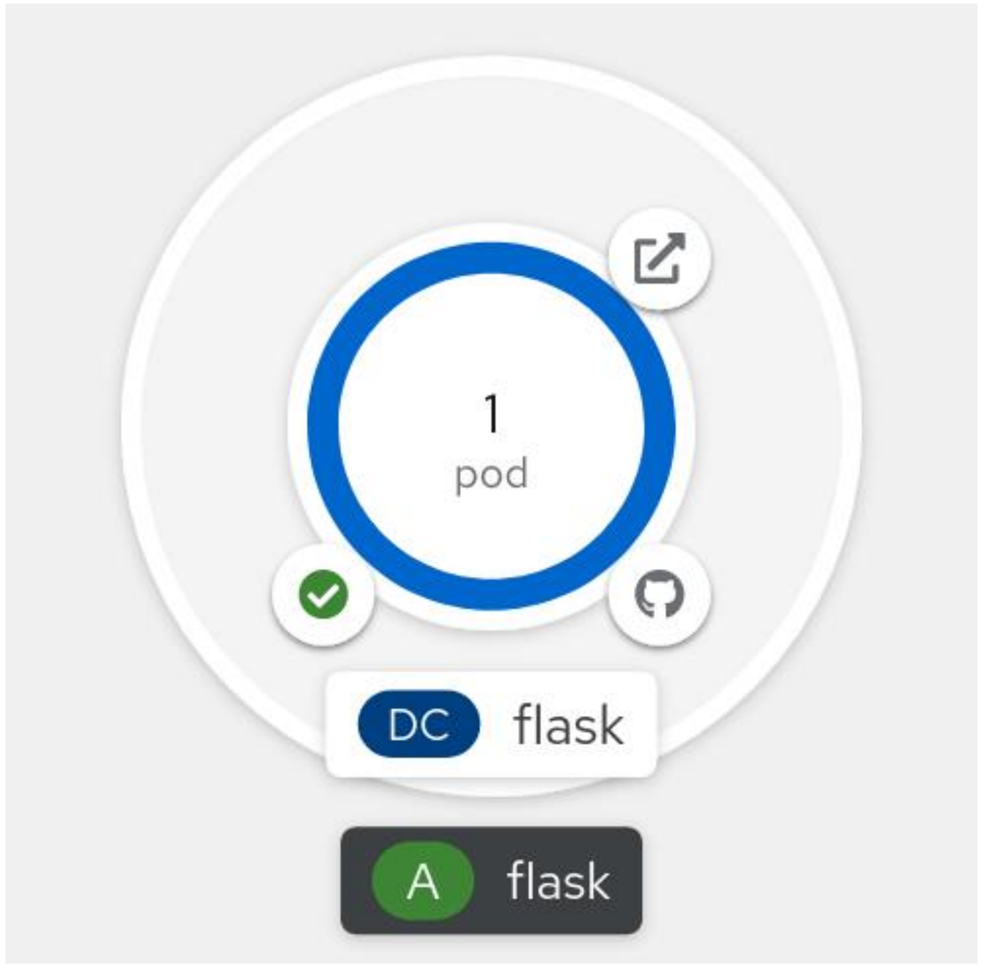
Exposes your application at a public URL

Click on the names to access advanced options for Routing, Health Checks, Build Configuration, Deployment, Scaling, Resource Limits and Labels.

Using the OpenShift GUI

Resulting topology

- Results are similar
- But: service and route needed to be configured to change the port from 8080 to 5000



Using the OpenShift GUI

Advantages and disadvantages

Advantages

- Easier to get familiar with concepts
- Easier to get overview of where things go wrong
- Many obstacles more ironed out
    - e.g. insecure registries

Disadvantages

- Hard to reproduce

IT+PS. Data Science

Challenges and possible solutions

Challenges in the process

New concepts in Kubernetes

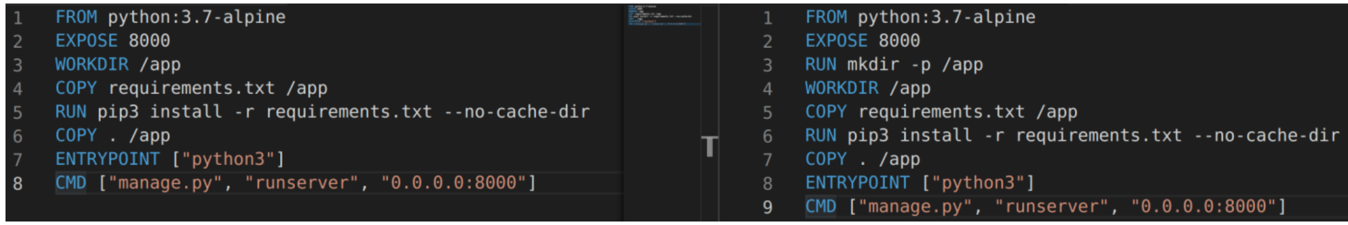
- In Docker: Services and containers (tasks)
- In OpenShift more concepts and they don't map 1:1 to Docker concepts
- Getting the head around not straight forward

- Tools like kompose or the OpenShift GUI can help to get familiar with them

Challenges in the process

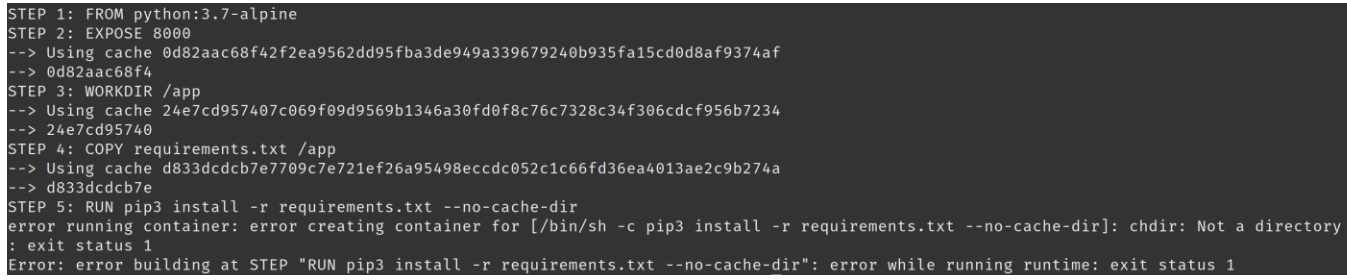
Image build using Buildah

- Images in OpenShift are built using Buildah
- Not all Dockerfiles can be built
- Podman build can help to debug the build process locally



```
FROM python:3.7-alpine
EXPOSE 8000
WORKDIR /app
COPY requirements.txt /app
RUN pip3 install -r requirements.txt --no-cache-dir
COPY . /app
ENTRYPOINT ["python3"]
CMD ["manage.py", "runserver", "0.0.0.0:8000"]
```

```
FROM python:3.7-alpine
EXPOSE 8000
RUN mkdir -p /app
WORKDIR /app
COPY requirements.txt /app
RUN pip3 install -r requirements.txt --no-cache-dir
COPY . /app
ENTRYPOINT ["python3"]
CMD ["manage.py", "runserver", "0.0.0.0:8000"]
```



```
STEP 1: FROM python:3.7-alpine
STEP 2: EXPOSE 8000
--> Using cache 0d82aac68f42f2ea9562dd95fba3de949a339679240b935fa15cd0d8af9374af
--> 0d82aac68f4
STEP 3: WORKDIR /app
--> Using cache 24e7cd957407c069f09d9569b1346a30fd0f8c76c7328c34f306cdcf956b7234
--> 24e7cd95740
STEP 4: COPY requirements.txt /app
--> Using cache d833dcdcb7e7709c7e721ef26a95498eccdc052c1c66fd36ea4013ae2c9b274a
--> d833dcdcb7e
STEP 5: RUN pip3 install -r requirements.txt --no-cache-dir
error running container: error creating container for [/bin/sh -c pip3 install -r requirements.txt --no-cache-dir]: chdir: Not a directory
: exit status 1
Error: error building at STEP "RUN pip3 install -r requirements.txt --no-cache-dir": error while running runtime: exit status 1
```

Challenges in the process

File permissions

- In Docker, usually everything is run as root
- In entrypoints of DB containers, often there is a chown of the data directory
- The OpenShift user will not have permissions to do so
- Solution: with PostgreSQL, specify env variable `PGDATA` to not point to `/var/lib/postgresql/data`

- Using official OpenShift images is the better option

Challenges in the process

Exposing port(s) in container

- Ports below 1024 are privileged ports
- Many Docker images (e.g. wordpress, apache, nginx) use port 80 by default
- Since the container is running with the OpenShift user, this will lead to a permission denied error
- Adapting the image / config is necessary to run them

- Using official OpenShift images is the better option

Challenges in the process

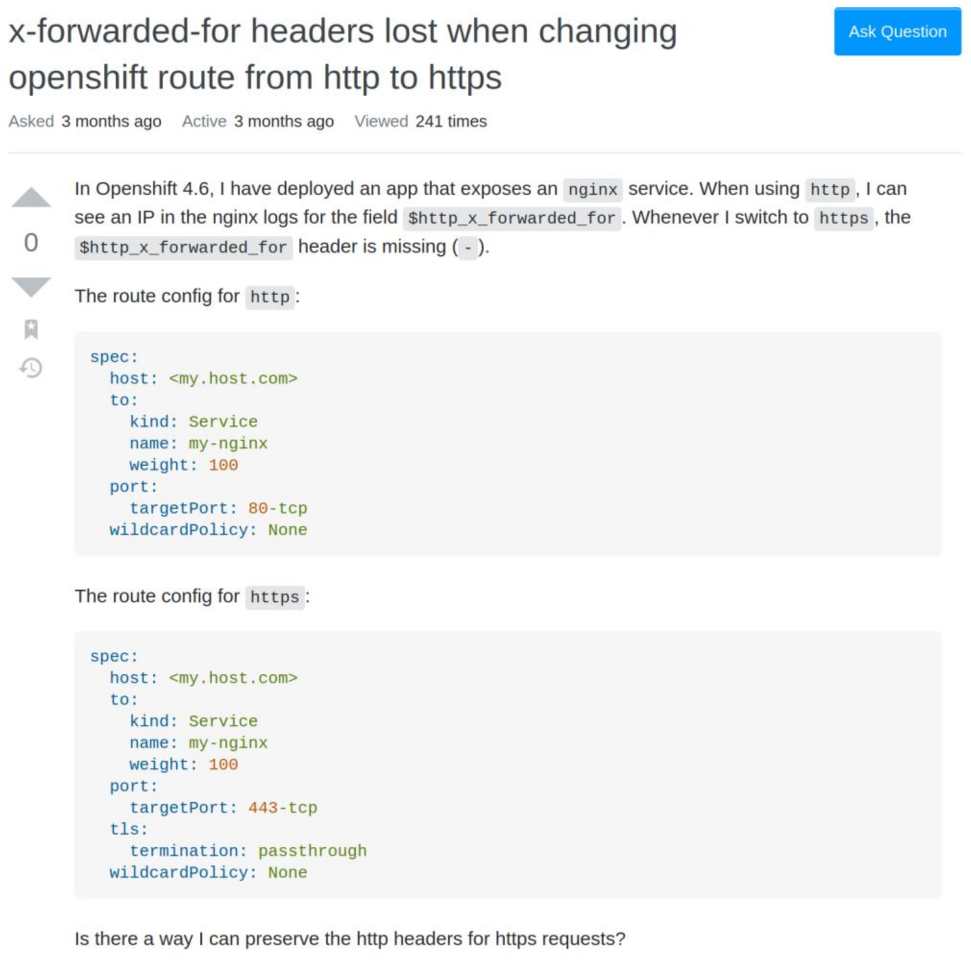
TLS termination and SSL certificates

- 3 ways of serving a certificate to clients
  - Re-encrypt: ingress serves certificate and re-encrypts traffic to pod
  - Edge: ingress serves certificate but does not re-encrypt
  - Passthrough: traffic is passed to pod which handles certificates
- Our NGINX deployment handles letsencrypt certificates automatically
- Requirement: make `/admin` page accessible only via VPN
- Problem: x-forwarded-for headers do not get passed on to nodes, nginx sees IP of ingress controller
- Workaround: edge termination without automatic certificate renewal

Challenges in the process

TLS termination and SSL certificates

- My question on stackoverflow is still not fully answered
- https://stackoverflow.com/questions/66473285/x-forwarded-for-headers-lost-when-changing-openshift-route-from-http-to-https



x-forwarded-for headers lost when changing openshift route from http to https

In Openshift 4.6, I have deployed an app that exposes an `nginx` service. When using `http`, I can see an IP in the nginx logs for the field `$http_x_forwarded_for`. Whenever I switch to `https`, the `$http_x_forwarded_for` header is missing (`-`).

The route config for `http`:

```
spec:
  host: <my.host.com>
  to:
    kind: Service
    name: my-nginx
    weight: 100
  port:
    targetPort: 80-tcp
  wildcardPolicy: None
```

The route config for `https`:

```
spec:
  host: <my.host.com>
  to:
    kind: Service
    name: my-nginx
    weight: 100
  port:
    targetPort: 443-tcp
  tls:
    termination: passthrough
  wildcardPolicy: None
```

Is there a way I can preserve the http headers for https requests?

Summary

- Getting started can feel overwhelming
- There are tools that make the transition easier
- Using the GUI first and trying the same at the CLI afterwards is a good way to learn
- Creating a [minimal, reproducible example] helps to iron out bugs

Photo by Valentin Betancur from Unsplash www.it-ps.at

Clemens Zauchner
Senior Data Scientist
+43 660 92 77 981
clemens.zauchner@it-ps.at

IT-Power Services GmbH
Modecenterstraße 14, 1030 Wien