



INFOSERVE
›eurodata-Gruppe

Introduction to Kubernetes

DevOpsSaar Meetup 2 | 25.10.2018

DR. PHILIPP WALTER | Leiter IT | INFOSERVE GmbH

VORWORT

Einführung in Kubernetes

DR. PHILIPP WALTER

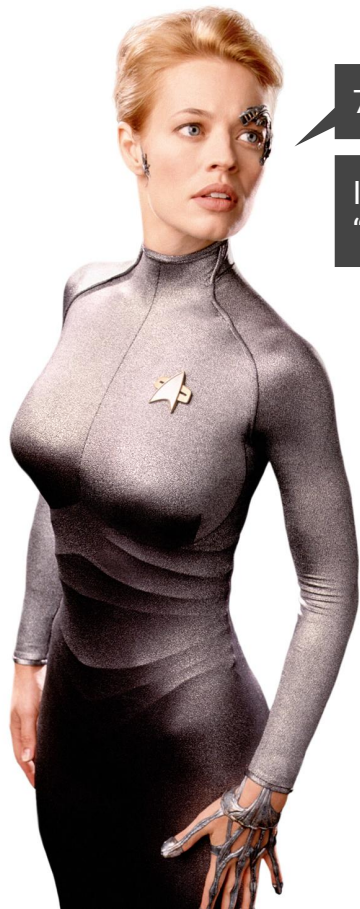
Kubernetes verwaltet Container auf Clustern

Docker : Kubernetes = 1 : N

	DOCKER	KUBERNETES
LAUFZEITUMGEBUNG	Einzelner Host mit Docker-Binaries	Viele Hosts mit Docker- und Kubernetes-Binaries
USE CASES	Software lokal laufen lassen Container-Images bauen und testen	Skalierbare und selbstheilende Produktionsumgebungen
API	docker pull, run, start, stop, ps, images, ...	kubectl + yaml-Specs (Details folgen...)
OBJEKTE	Container, Images, Volumes, Ports	Pods aus 1..n Containern, Services, Clusterknoten, Ingress, Deployments, ...
PARADIGMA	Low-Level-Operationen , die manuell auf einzelnen Containern ausgeführt werden	Definiere einen Sollzustand , den Kubernetes automatisch um jeden Preis zu halten versucht
SETUP	apt-get install docker	kubeadm, GKE, CoreOS, ... (Details folgen)

Warum “Kubernetes”?

Ein bisschen Nerd muss sein: Google und die Borg



7 of 9, Borg

7 Seiten...

Interner Cluster Manager
“Borg” bei Google (2015)



kubernetes

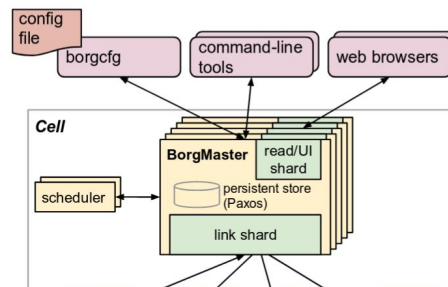
Large-scale cluster management at Google with Borg

Abhishek Verma[†] Luis Pedrosa[‡] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.

Abstract

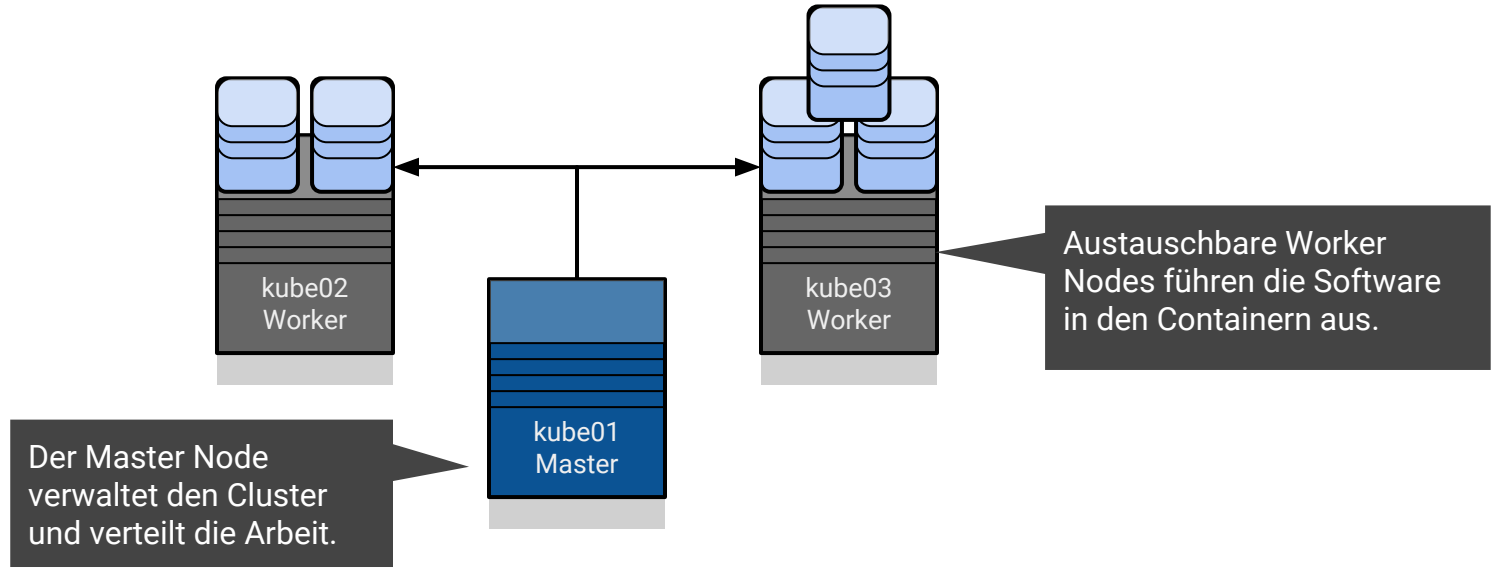
Google’s Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that re-



Kubernetes 101

Laufzeitumgebung: Cluster statt Computer



```
epic ~ kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
kube01	Ready	master	3d	v1.11.2	<none>	CentOS Linux 7 (Core)	3.10.0-862.11.6.el7.x86_64	docker://1.13.1
kube02	Ready	<none>	3d	v1.11.2	<none>	CentOS Linux 7 (Core)	3.10.0-862.11.6.el7.x86_64	docker://1.13.1
kube03	Ready	<none>	3d	v1.11.2	<none>	CentOS Linux 7 (Core)	3.10.0-862.11.6.el7.x86_64	docker://1.13.1

Kubernetes 101

Kubernetes versucht, den Sollzustand zu erhalten, indem Prozesse am Laufen gehalten werden

```
epic ~ kubectl run -i -t busybox --image=busybox:1.28
```

If you don't see a command prompt, try pressing enter.

```
/ # exit
```

Session ended, resume using 'kubectl attach busybox-7999f69f9d-68tv4 -c busybox -i -t' command when the pod is running

```
epic ~ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
busybox-7999f69f9d-68tv4	1/1	Running	1	6m

```
epic ~ kubectl attach busybox-7999f69f9d-68tv4 -c busybox -i -t
```

If you don't see a command prompt, try pressing enter.

```
/ # exit
```

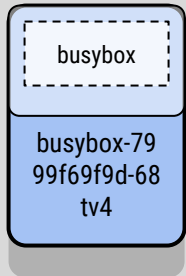
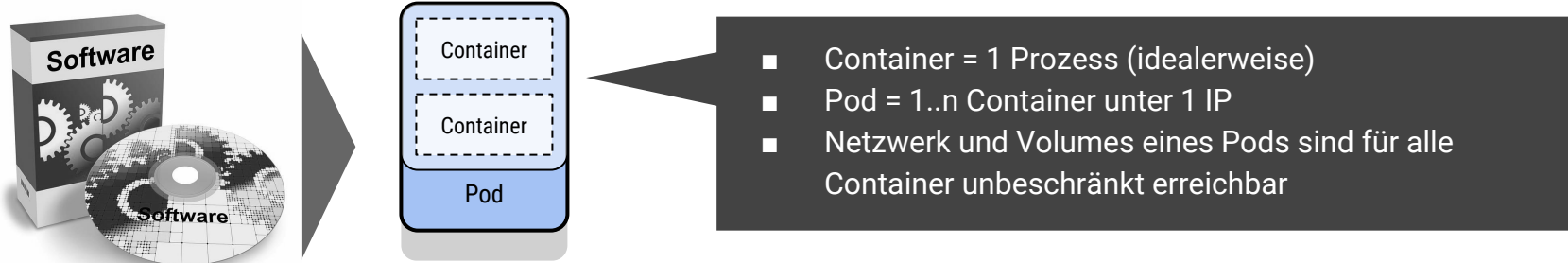
Session ended, resume using 'kubectl attach busybox-7999f69f9d-68tv4 -c busybox -i -t' command when the pod is running

```
epic ~ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
busybox-7999f69f9d-68tv4	1/1	Running	2	7m

Kubernetes 101

Pods bestehen aus Containern und sind der elementare Baustein von Kubernetes



```
epic ~ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
busybox-799f69f9d-68tv4	1/1	Running	2	7m

```
epic ~ kubectl attach busybox-799f69f9d-68tv4 -c busybox -i -t
```

If you don't see a command prompt, try pressing enter.

```
/ #
```

Kubernetes 101

Replica Sets halten Pods am Laufen und werden als Deployments verwaltet

```
epic ~ kubectl describe pod busybox-7999f69f9d-68tv4
```

```
...
```

```
Status:           Running
```

```
IP:              10.116.0.3
```

```
Controlled By:   ReplicaSet/busybox-7999f69f9d
```

```
...
```

```
epic ~ kubectl describe ReplicaSet/busybox-7999f69f9d
```

```
...
```

```
Controlled By:   Deployment/busybox
```

```
Replicas:       1 current / 1 desired
```

```
Pods Status:    1 Running / 0 Waiting / 0 Succeeded / 0 Failed
```

```
...
```

```
epic ~ kubectl describe Deployment/busybox
```

```
...
```

```
Replicas:       1 desired | 1 updated | 1 total | 1 available | 0 unavailable
```

```
StrategyType:    RollingUpdate
```

```
RollingUpdateStrategy: 1 max unavailable, 1 max surge
```

```
...
```


Kubernetes 101

Deployments bündeln Pods und Replica Sets

```
epic ~ kubectl get all
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
deploy/busybox	1	1	1	1	1h

NAME	DESIRED	CURRENT	READY	AGE
rs/busybox-7999f69f9d	1	1	1	1h

NAME	READY	STATUS	RESTARTS	AGE
po/busybox-7999f69f9d-68tv4	1/1	Running	3	1h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	3d

```
epic ~ kubectl delete deployment busybox
```

```
deployment "busybox" deleted
```

```
epic ~ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
po/busybox-7999f69f9d-68tv4	1/1	Terminating	3	1h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	3d

Deployments

Deployments können als YAML oder JSON beschrieben werden. Hurra, Infrastructure as Code!

echo.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: echo
spec:
  replicas: 2
  selector:
    matchLabels:
      pod: echo
  template:
    metadata:
      labels:
        pod: echo
    spec:
      containers:
        - name: echo
          image: jmalloc/echo-server:latest
          ports:
            - containerPort: 8080
```

```
epic ~ kubectl apply -f echo.yaml
```

```
deployment "echo" created
```

```
epic ~ kubectl get all
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
deploy/echo	2	2	2	2	6m

NAME	DESIRED	CURRENT	READY	AGE
rs/echo-8594ff85cb	2	2	2	6m

NAME	READY	STATUS	RESTARTS	AGE
po/echo-8594ff85cb-vg72l	1/1	Running	0	6m
po/echo-8594ff85cb-xbl9t	1/1	Running	0	6m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	5d

```
epic ~ kubectl scale deployment echo --replicas 3
```

```
deployment "echo" scaled
```

```
epic ~ kubectl get all
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
deploy/echo	3	3	3	3	7m

```
...
```

Exkurs: Labels und Selektoren

Man kann alles beschriften und damit auswählen

Selector: app=infos

app = infos
stage = prod
tier = web

app = infos
stage = test
tier = web

app = infos
stage = prod
tier = app

app = infos
stage = test
tier = app

Selector: app=ed, tier=web

app = ed
stage = prod
tier = web

app = ed
stage = test
tier = web

app = ed
stage = prod
tier = app

app = ed
stage = test
tier = app

Selector:
tier=db

app = infos
stage = prod
tier = db

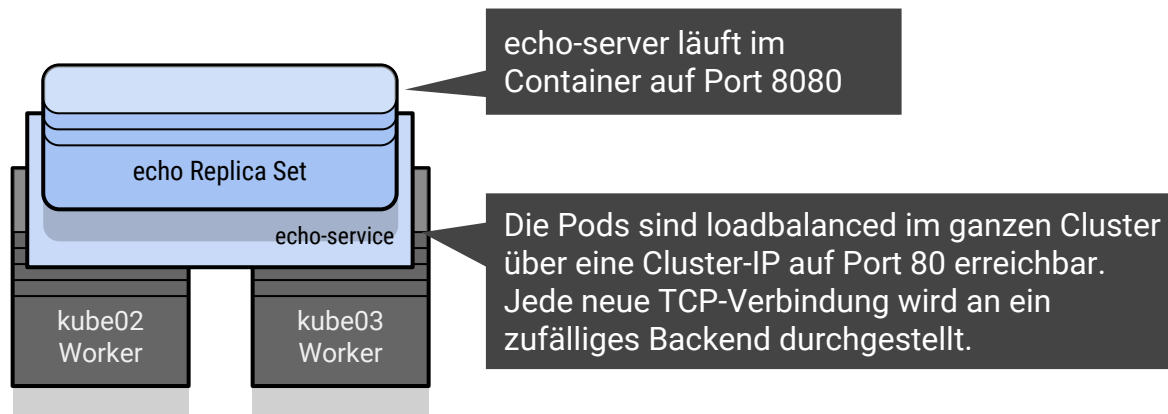
app = infos
stage = test
tier = db

app = ed
stage = prod
tier = db

app = ed
stage = test
tier = db

Services

Services ermöglichen Layer-4-Load-Balancing über Pods



echo-service.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: echo-service
spec:
  selector:
    pod: echo
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
    type: NodePort
```

```
epic ~ kubectl apply -f echo-service.yaml
service "echo-service" created
```

```
epic ~ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
echo-service	NodePort	10.104.0.183	<none>	80:31763/TCP	6m
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	5d

Services

Die Cluster-IP eines Service kann überall im Cluster auch über seinen DNS-Namen aufgelöst werden

```
epic ~ kubectl run -i -t busybox --image=busybox:1.28
```

If you don't see a command prompt, try pressing enter.

```
/ # nslookup echo-service
```

```
Server:      10.96.0.10
```

```
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local
```

```
Name:        echo-service
```

```
Address 1: 10.104.0.183 echo-service.default.svc.cluster.local
```

```
/ # watch -n 1 wget -qO - echo-service | grep served
```

```
Request served by echo-8594ff85cb-vg72l
```

```
Request served by echo-8594ff85cb-xbl9t
```

```
Request served by echo-8594ff85cb-xbl9t
```

```
Request served by echo-8594ff85cb-vg72l
```

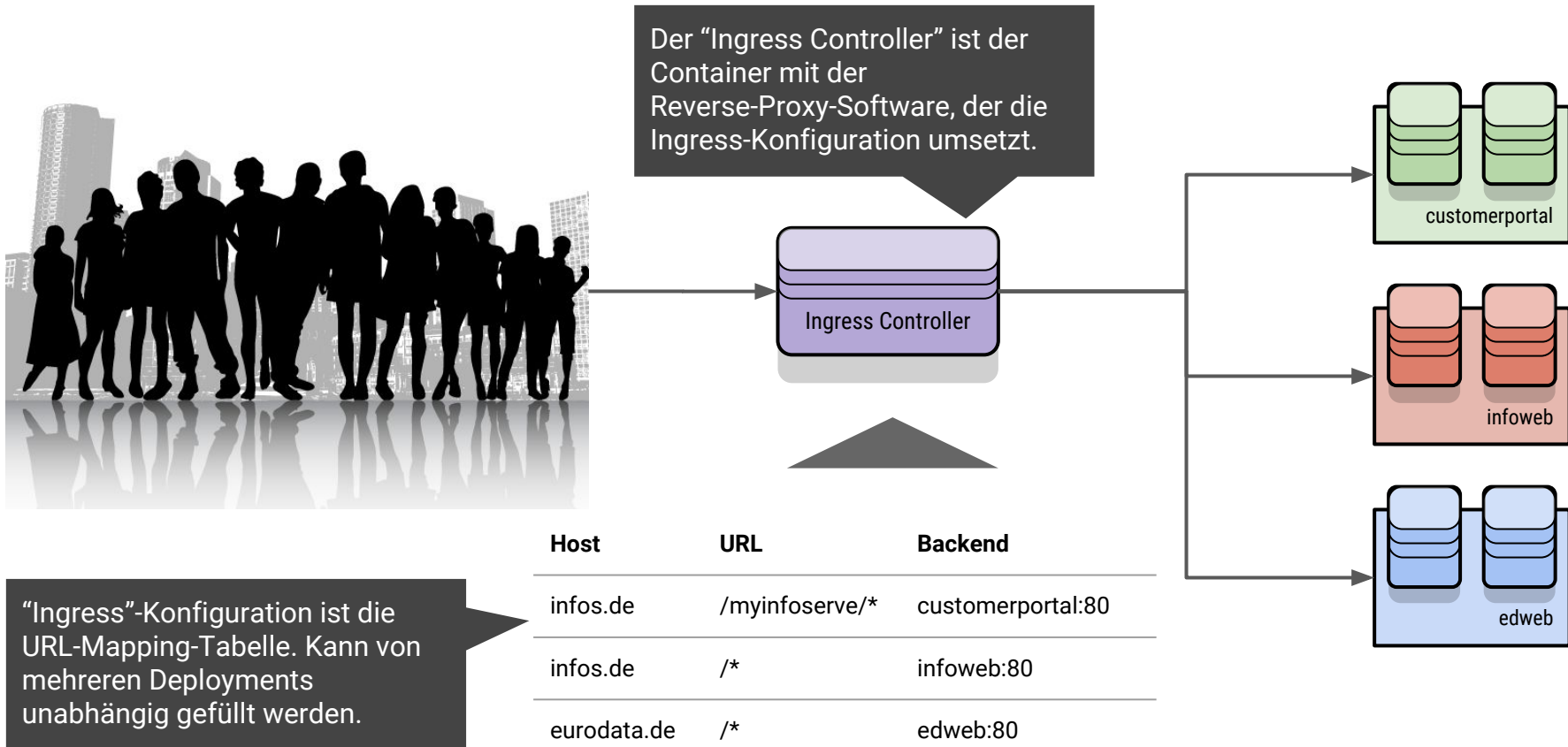
```
Request served by echo-8594ff85cb-vg72l
```

```
...
```

```
/ #
```

Ingress Controller

Layer-7-Load-Balancing mit URL Mappings auf verschiedene Services

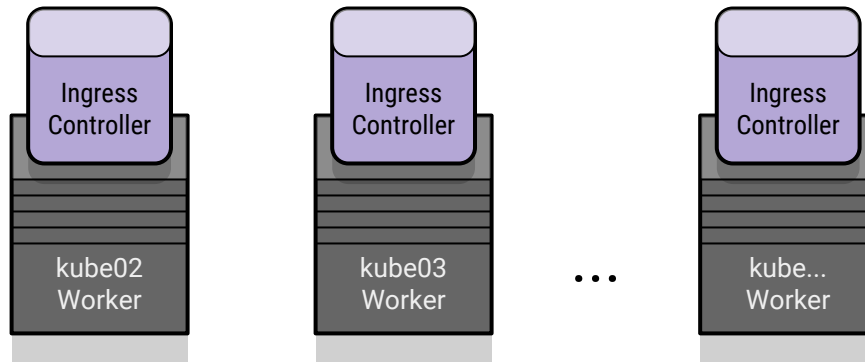


Ingress Controller

Ingress Controller auf eigenem Cluster (kein GKE) deployen

```
traefik-ingress-controller.yaml
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: traefik-ingress-controller
  namespace: kube-system
---
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  ...
```



<https://docs.traefik.io/user-guide/kubernetes/>

```
epic ~ kubectl get all --namespace=kube-system -l k8s-app=traefik-ingress-lb
```

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
ds/traefik-ingress-controller-v1	2	2	2	2	2	<none>	8m

NAME	READY	STATUS	RESTARTS	AGE
po/traefik-ingress-controller-v1-7664s	1/1	Running	0	8m
po/traefik-ingress-controller-v1-pk7bq	1/1	Running	0	8m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
traefik-ingress-service	ClusterIP	10.99.243.201	<none>	80/TCP,443/TCP,8080/TCP	8m	k8s-app=traefik-ingress-lb

Ingress-Ressourcen

Ingress-Ressourcen machen Services unter einer URL und einem Pfad zugänglich

echo-ingress.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: echo-ingress
spec:
  rules:
  - http:
      paths:
      - path: /echo
        backend:
          serviceName: echo-service
          servicePort: 80
```

```
epic ~ kubectl create -f echo-ingress.yaml
ingress "echo-ingress" created
```

```
epic ~ curl kube02/echo
Request served by echo-8594ff85cb-xmbb7
```

HTTP/1.1 GET /echo

```
Host: kube02
X-Forwarded-Port: 80
X-Forwarded-Server: traefik-ingress-controller-qvqbj
X-Real-IP: 10.124.0.0
User-Agent: curl/7.61.0
X-Forwarded-For: 10.124.0.0
X-Forwarded-Host: kube02
Accept-Encoding: gzip
Accept: */*
X-Forwarded-Proto: http
```

```
epic ~ curl kube03/echo
Request served by echo-8594ff85cb-mj65h
```

HTTP/1.1 GET /echo

...

DEMO

Kubernetes-Cluster auf VMware-Basis

DR. PHILIPP WALTER