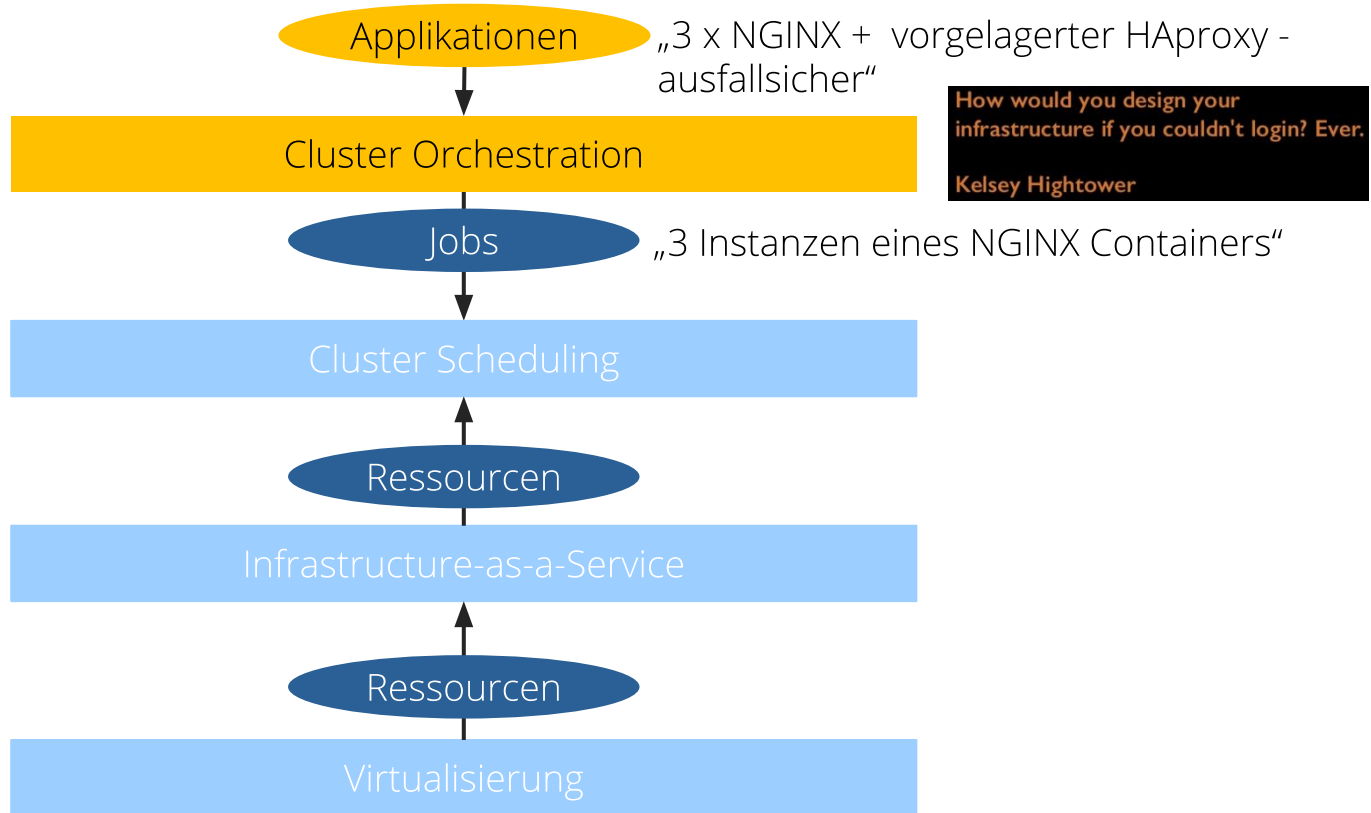


# Cloud Computing Orchestrierung

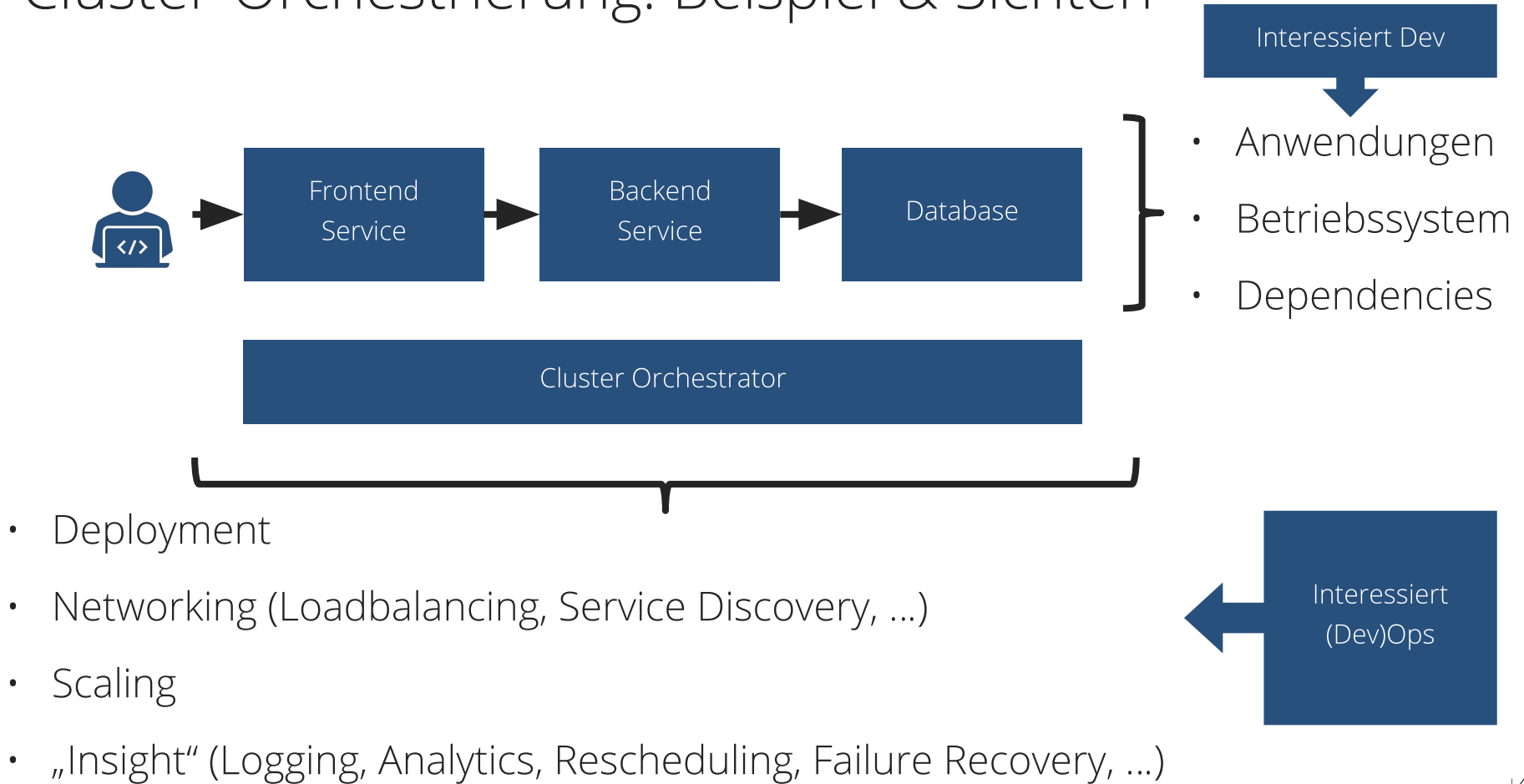


# Orchestrierung - Definition, Aufgaben, Konzepte

# Big Picture: Wir sind nun auf Applikationsebene



# Cluster-Orchestrierung: Beispiel & Sichten



# Cluster-Orchestrierung

- Ziel: Eine Anwendung, die in mehrere Betriebskomponenten (Container) aufgeteilt ist, auf mehreren Knoten laufen lassen.  
„Running Containers on Multiple Hosts“.  
DockerCon SF 2015: Orchestration for Sysadmins
- Führt Abstraktionen zur Ausführung von Anwendungen mit ihren benötigten Schnittstellen und Betriebskomponenten ein.
- Orchestrierung ist keine statische, einmalige Aktivität wie die Provisionierung, sondern eine dynamische, kontinuierliche Aktivität.
- Orchestrierung hat den Anspruch, alle Standard-Betriebsprozeduren einer Anwendung zu automatisieren.

**Blaupause der Anwendung**, die den gewünschten Betriebszustand der Anwendung beschreibt: Betriebskomponenten (Container), deren Betriebsanforderungen sowie die angebotenen und benötigten Schnittstellen.



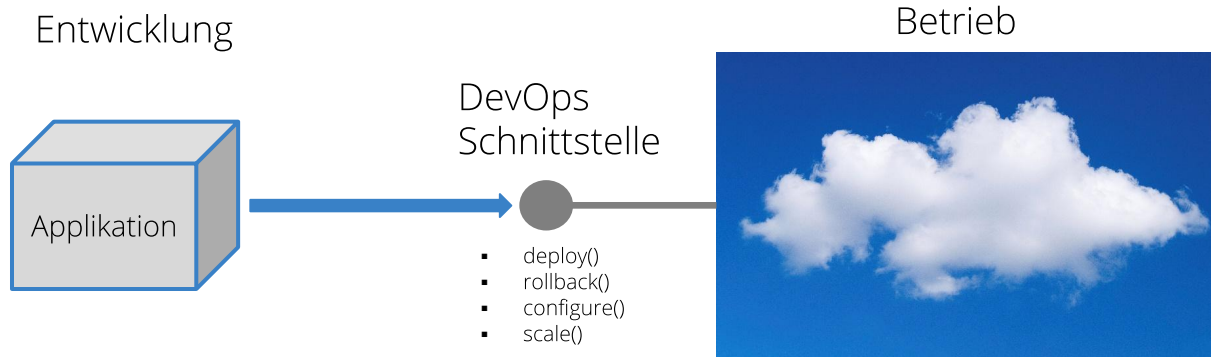
Cluster-Orchestrierer



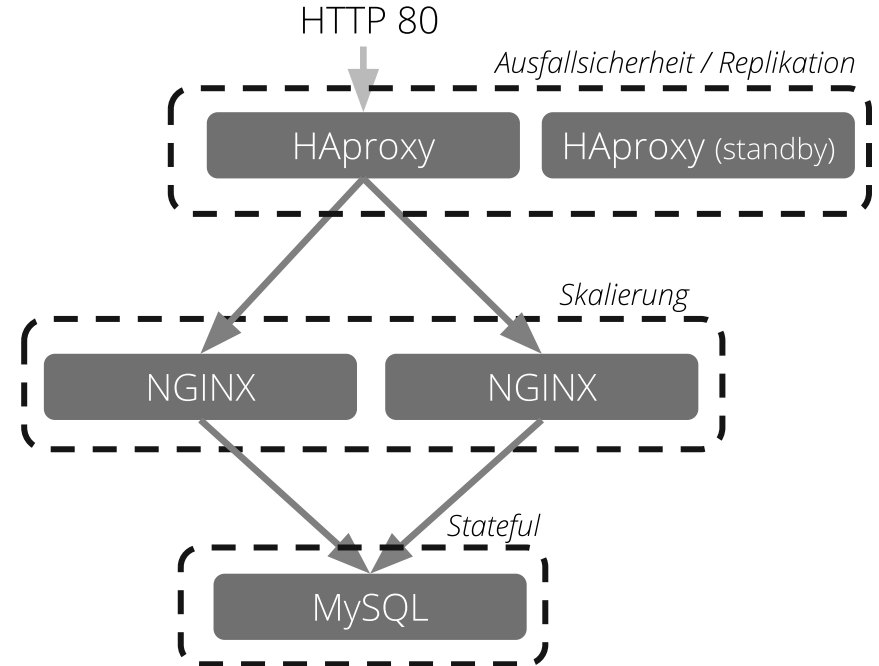
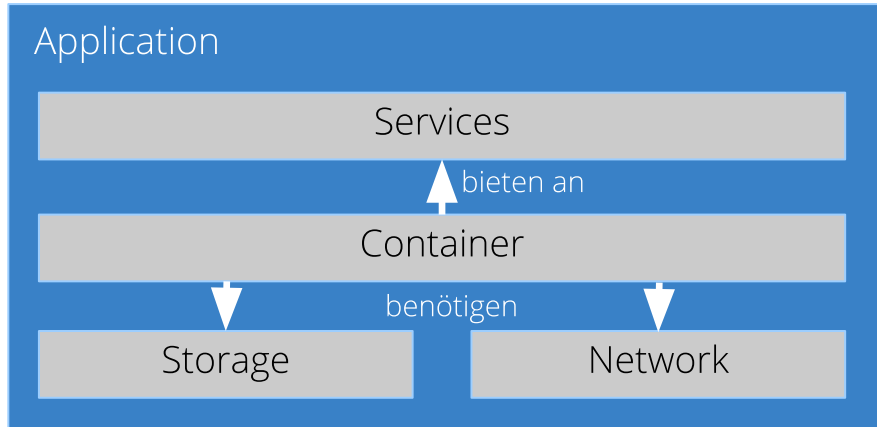
## Steuerungsaktivitäten im Cluster:

- Start von Containern auf Knoten (□ Scheduler)
- Verknüpfung von Containern
- ...

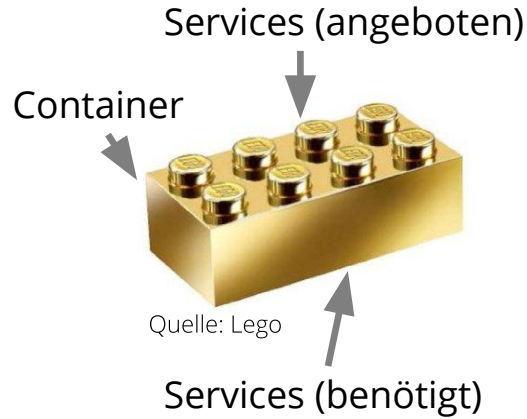
# Ein Cluster-Orchestrierer bietet eine Schnittstelle zwischen Betrieb und Entwicklung für ein Cluster an



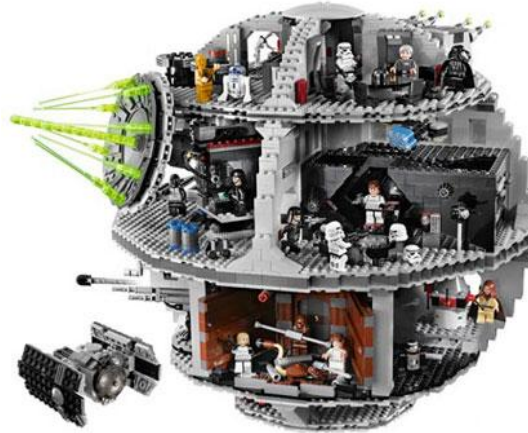
# Dafür benötigt er für jede Anwendung eine Blaupause



# Analogie 1: Lego Star Wars



Cluster-Orchestrierer

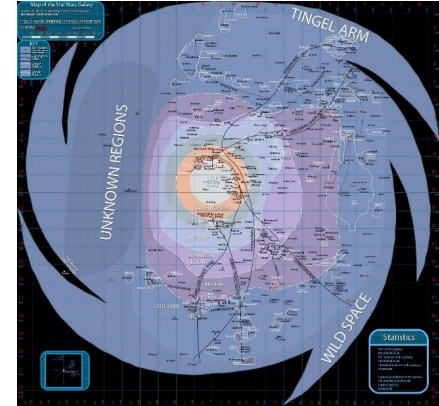


Quelle: Lego



Blaupause

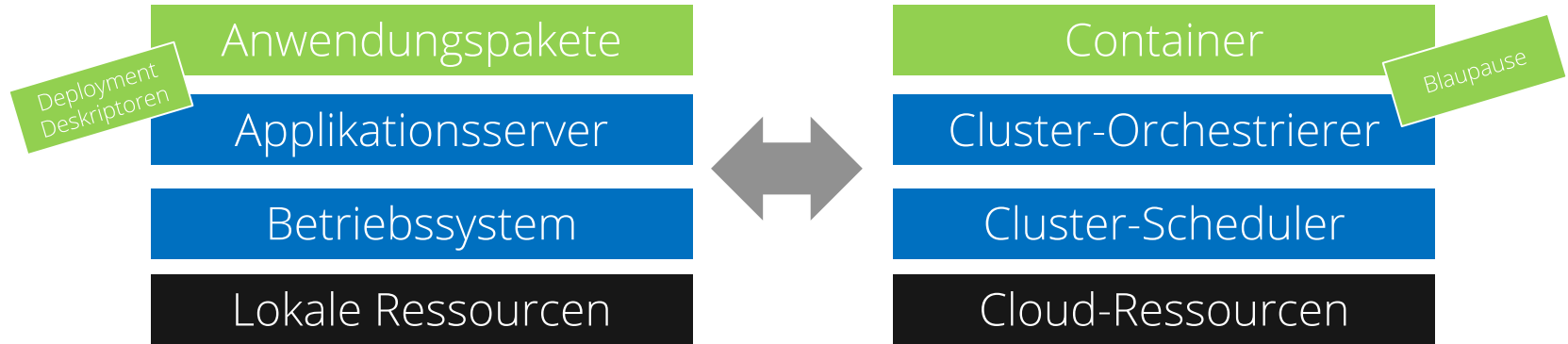
Cluster-Scheduler



Quelle: wikipedia.de



# Analogie 2: Applikationsserver



# Cluster-Orchestrierer automatisieren verschiedene Betriebsaufgaben für Anwendungen auf einem Cluster (1/2):

- Container-Logistik: Verwaltung und Bereitstellung von Containern.
- Package-Management: Verwaltung und Bereitstellung von Applikationen.
- Bereitstellung von Administrationsschnittstellen (Remote-API, Kommandozeile).
- Management von Services: Service Discovery, Naming, Load Balancing.
- Automatismen für Rollout-Workflows wie z.B. Canary Rollout.
- Monitoring und Diagnose von Containern und Services.

# Cluster-Orchestrierer automatisieren verschiedene Betriebsaufgaben für Anwendungen auf einem Cluster (2/2):

- Scheduling von Containern mit applikationsspezifischen Constraints (z.B. Deployment- und Start-Reihenfolgen, Gruppierung, ...)
- Aufbau von notwendigen Netzwerk-Verbindungen zwischen Containern.
- Bereitstellung von persistenten Speichern für zustandsbehaftete Container.
- (Auto-)Skalierung von Containern.
- Re-Scheduling von Containern im Fehlerfall (Auto-Healing) oder zur Performance-Optimierung.



# Übung: Orchestrierungsmuster

- Separation of Concerns mit modularen Containern



# Kubernetes



# kubernetes

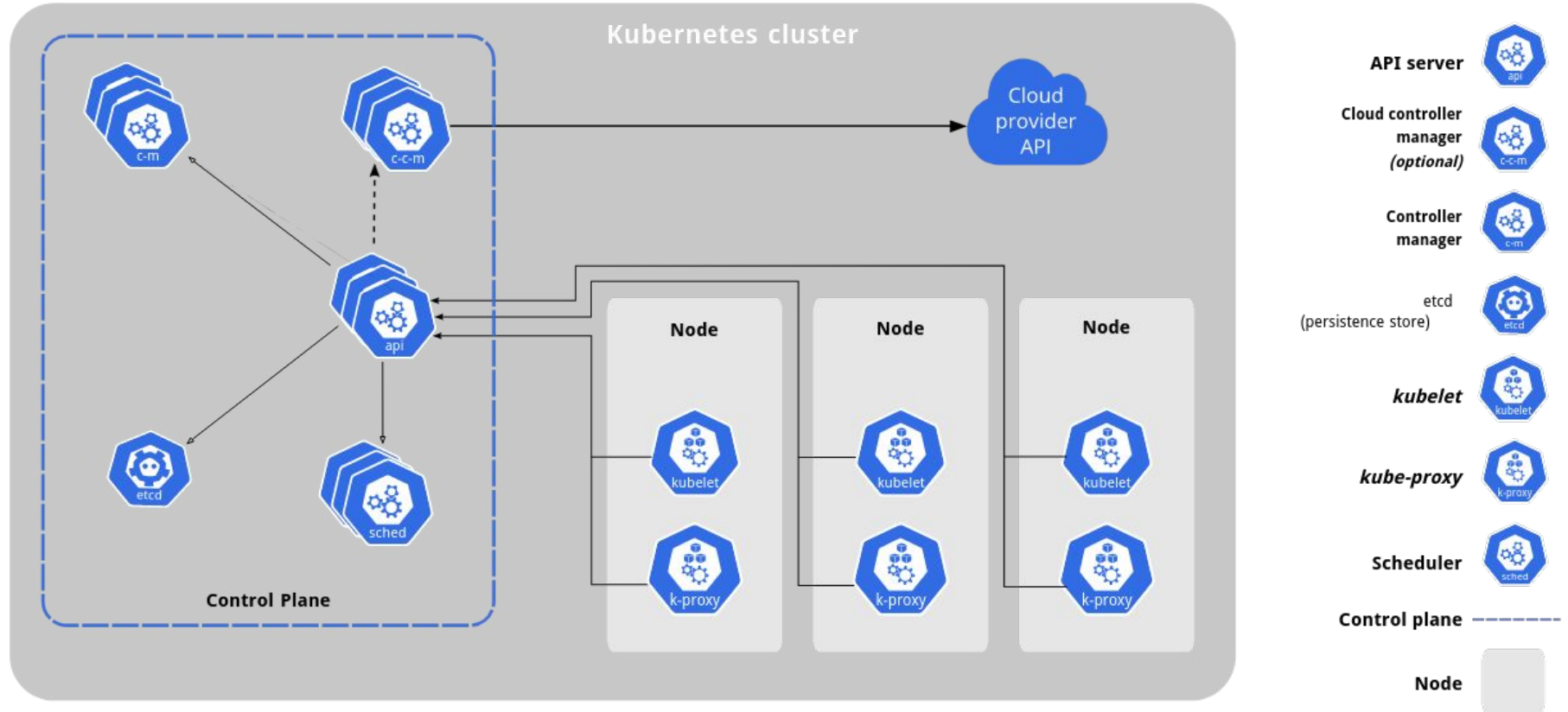
Kubernetes (auch als K8s bezeichnet) ist ein **Open-Source**-System zur **Automatisierung** der **Bereitstellung**, **Skalierung** und **Verwaltung** von **Container-Anwendungen**, das ursprünglich von Google entworfen und an die Cloud Native Computing Foundation (CNCF) gespendet wurde. Es zielt darauf ab, eine „Plattform für ... Anwendungscontainer[n] auf **verteilten Hosts**“ zu liefern. Es unterstützt eine Reihe von Container-Tools, einschließlich **Docker**.

[Quelle: Wikipedia](#)

# Wieso?!

- Ungelöst: Effiziente Nutzung der Hardware in einem Rechenzentrum
  - Ein Server pro Team? Was passiert, wenn ein Team keinen ganzen braucht?
- Virtualisierung und Elastizität löst das Problem der Hardware-Beschaffung
- Ungelöst: Development & Deployment der Anwendungen
- Trend: Weg vom Monolithen, hin zu Microservices
  - Verstärkt das Problem des Deployments
  - Wie finden sich die Microservices gegenseitig?
- Trend: Zero-downtime Deployments
- Lösung: Kubernetes als Standard
  - Anwendung in Container verpacken
  - Deployment beschreiben
  - Kubernetes kümmert sich um den Betrieb: startet neu, skaliert, etc.
  - Zero-Downtime und Rollbacks inkl.

# Architektur von Kubernetes





# Terminologie

- **Pods** are the smallest deployable units of computing that you can create and manage in Kubernetes.
- Kubernetes runs your workload by placing containers into Pods to run on Nodes. A **node** may be a virtual or physical machine, depending on the cluster.
- A **service** is an abstract way to expose an application running on a set of Pods as a network service.

# Aufgaben der Bausteine auf einem Node

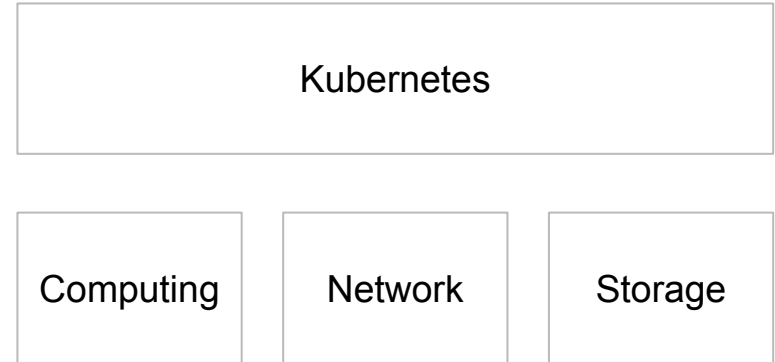
- **Container runtime:** "The container runtime is the software that is responsible for running containers. Kubernetes supports several container runtimes, one of them is Docker"
- **kubelet:** "An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod."
- **kube-proxy:** "kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept."

# Aufgaben der Bausteine in der Control Plane

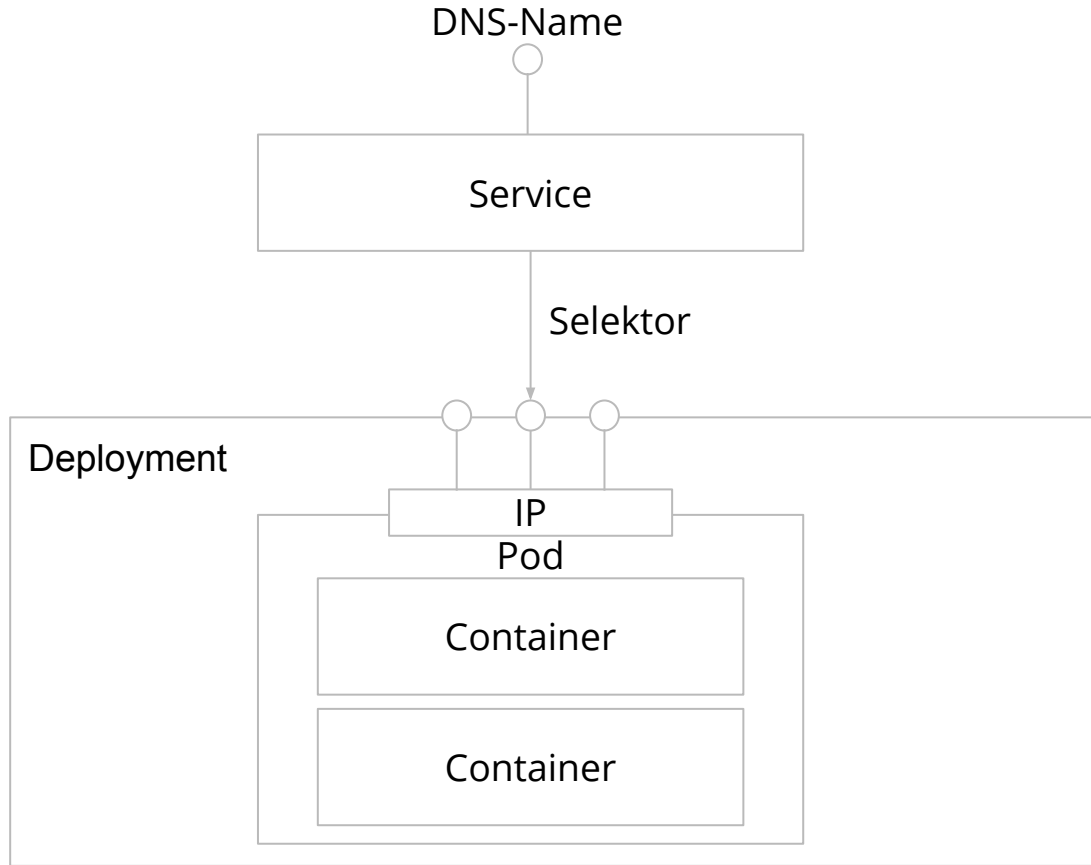
- **API Server:** "The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane."
- **etcd:** "Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data."
- **scheduler:** "Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on."
- **Controller Manager:** "Control Plane component that runs controller processes."
  - Node controller: Responsible for noticing and responding when nodes go down.
  - Replication controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
  - ...
- **Cloud Controller Manager** (optional): "A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that just interact with your cluster."

# Kubernetes virtualisiert:

- Computing durch die Container Runtime
- Network durch virtuelle IPs und Overlay Networks
- Storage durch Persistent Volumes



# Die wichtigsten Abstraktionen von Kubernetes



# Die wichtigsten Abstraktionen von Kubernetes

**Service:** Endpunkt unter einem definierten DNS-Namen, der Aufrufe an die Pods im Hintergrund verteilt (Load Balancing, Failover). Die für einen Service relevanten Pods werden über ihre Labels selektiert, z.B.: `role = apache`, `env != test`, `tier in (web, app)`

**Pod:** Gruppe an Containern, die auf dem selben Knoten laufen und sich eine Netzwerk-Schnittstelle inklusive einer dedizierten IP, persistente Volumes und Umgebungsvariablen teilen. Ein Pod ist die atomare Scheduling-Einheit in K8s. Ein Pod kann über sog. Labels markiert werden, das sind frei definierbare Schlüssel-Wert-Paare.

**Deployment:** Klammer um einen gewünschten Zielzustand im Cluster in Form eines Pods mit dazugehörigem Replication Controller. Ein Deployment bezieht sich nicht auf Services, da diese in der K8s-Philosophie einen von Pods unabhängigen Lebenszyklus haben.



# Live-Demo: Lokales Kubernetes mit minikube

Release v0.23.7

Notes

Thank you to all that contributed with patches and enhancements, for 2023 I'll try to make some of them decide as final. But if you don't mind, please let me know if you have any suggestions or if you want to be part of the team (please help me with the code, the support, the design and the development of the project) or if you want to be part of the team (please help me with the code, the support, the design and the development of the project).


On 2023 I'll try to make some of them decide as final. But if you don't mind, please let me know if you have any suggestions or if you want to be part of the team (please help me with the code, the support, the design and the development of the project).

Maintenance Release!

Resolved Issues/Features

Resolved PRs:

© 2023 Endless Software LLC. All rights reserved under Apache 2.0





# Minikube

- [Minikube](#) installiert ein lokales Kubernetes-Cluster auf dem eigenen Rechner
- Eignet sich super zum Ausprobieren von Kubernetes
- [kubectl](#) ist ein Client für die Kubernetes API, die verwendet wird, um Kubernetes zu konfigurieren
- [K9s](#) ist eine Terminal UI für Kubernetes



# Live-Demo: Pods & Deployment



# Pods & Deployments

- [Pods](#) sind die kleinste schedulbare Einheit in Kubernetes
- Ein [Deployment](#) beschreibt, aus welchen Containern der Pod besteht und wie er konfiguriert ist

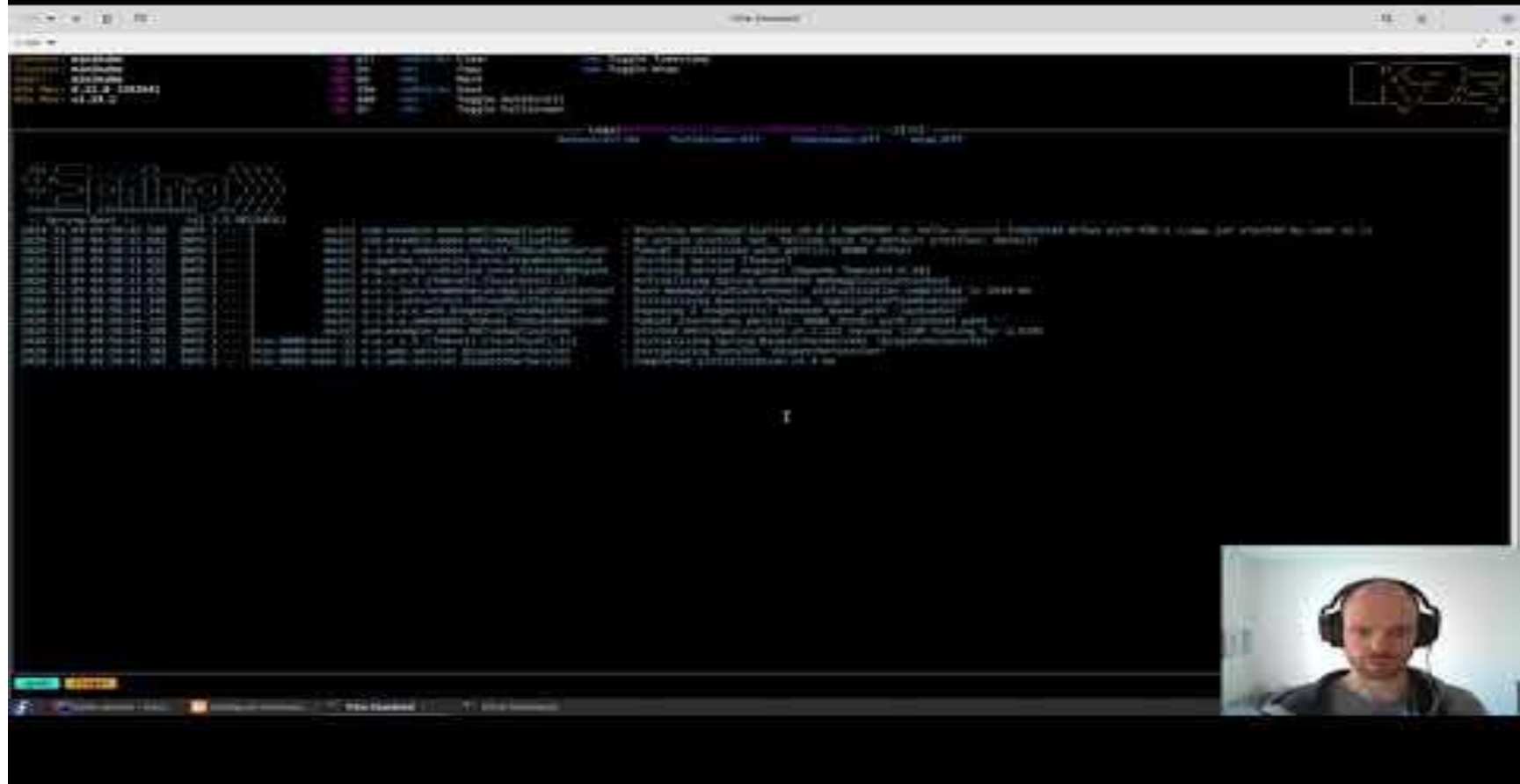


# Übung, Aufgabe 2 & 3

## 30 Minuten



# Live-Demo: Probes



<https://youtu.be/IDD9twPu-hs>

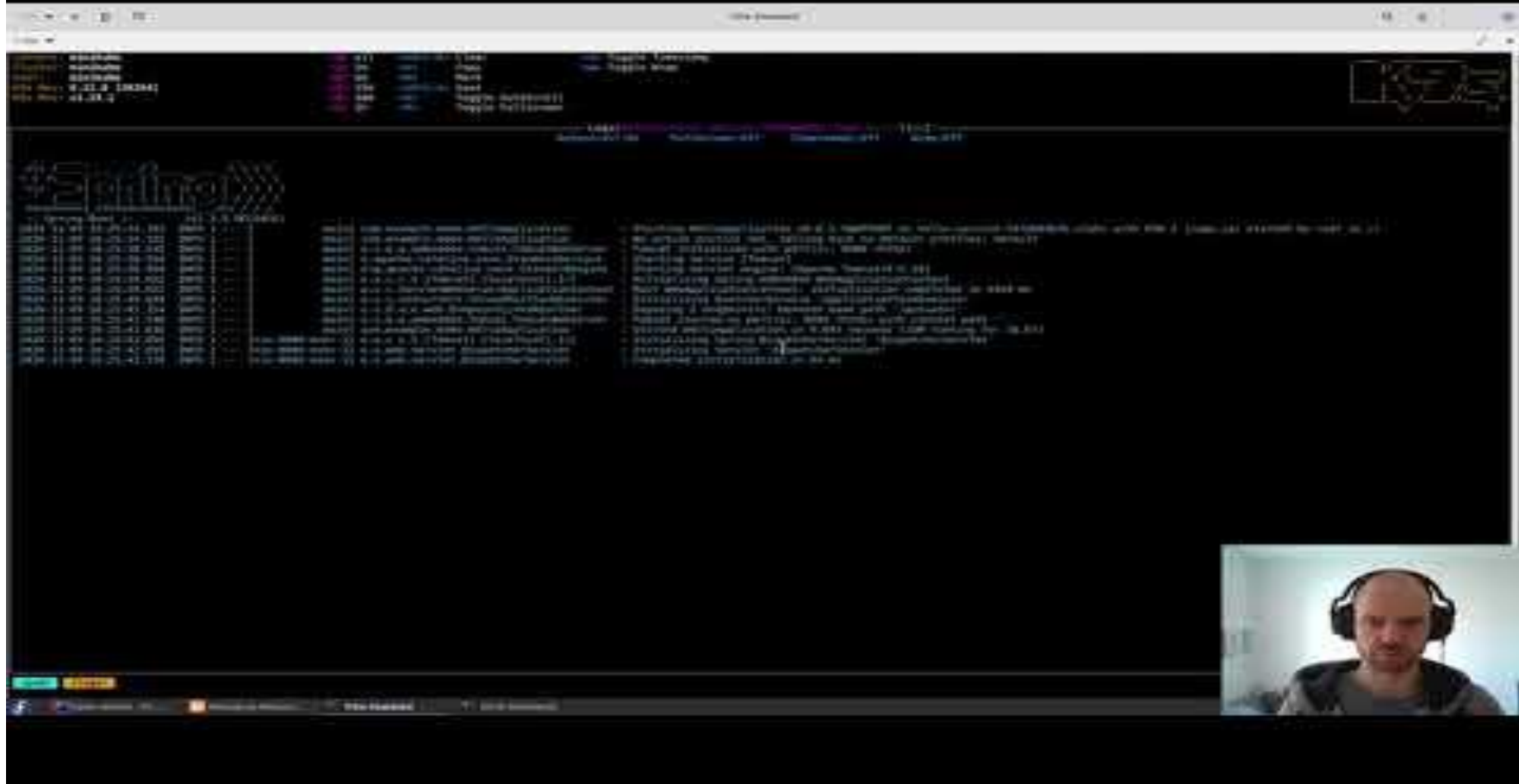
# Probes

- <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes>
- Liveness probes prüfen, ob der Container noch am Leben ist. Falls nicht, wird dieser neu gestartet.
- Readiness probes prüfen, ob der Container bereit ist, Traffic zu bekommen. Falls nicht, wird dieser aus dem Load Balancing genommen.
- Mit einer Startup Probe kann das Problem von langsam startenden Containern mitigiert werden





# Live-Demo: Resource Constraints



<https://youtu.be/dUyAueNWA1k>

# Resource constraints

- <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>
- Resource Requests helfen dem Scheduler, einen geeigneten Node für den Pod zu finden
- Resource Limits schützen vor amoklaufenden Containern, in dem sie den Container abschießen, sollte dieser die Memory-Limits erreichen
- Limits können auf Speicher und auf CPU gesetzt werden

# Resource Constraints: CPU

Limits and requests for CPU resources are measured in cpu units. One cpu, in Kubernetes, is equivalent to 1 vCPU/Core for cloud providers and 1 hyperthread on bare-metal Intel processors.

Fractional requests are allowed. A Container with `spec.containers[].resources.requests.cpu` of 0.5 is guaranteed half as much CPU as one that asks for 1 CPU. The expression 0.1 is equivalent to the expression 100m, which can be read as "one hundred millicpu". Some people say "one hundred millicores", and this is understood to mean the same thing. A request with a decimal point, like 0.1, is converted to 100m by the API, and precision finer than 1m is not allowed. For this reason, the form 100m might be preferred.

CPU is always requested as an absolute quantity, never as a relative quantity; 0.1 is the same amount of CPU on a single-core, dual-core, or 48-core machine.

Quelle: <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>



# Übung, Aufgabe 4

## 15 Minuten



# Live-Demo: Services

© 2004 Blackwell Publishing Ltd, *Journal of Internal Medicine* 255: 103–110



- Other
- Testing started
- Concepts
- How to use
- Client Architecture
- Guidelines
- Workflow
- Services, Load Balancing and Monitoring**
- Services
  - Service Discovery
  - Microservice Architecture
  - Connecting Applications
  - API Gateway
  - Service Mesh
  - APIs
  - Service Virtualization
  - Service Framework
  - Building a Service Mesh
  - Service Mesh in Kubernetes
  - Service Mesh in AWS
- Storage
  - Cloud Storage
  - Security
  - Performance
  - Integration with Data Lake

### Service

When submitting your application, you agree to transfer your application to our confidential service. We reserve the right to use your information for other purposes. We reserve the right to use your information for other purposes. We reserve the right to use your information for other purposes.

### Motivation =>

Scholarship funds are created and managed by NAFSA for the benefit of your member. Funds are restricted resources. Please add a designated title to your grant use. Full credit will be given to your title.

Each Pyl post is sent to the Pyl address, listed in a `__pylmail__` file. The set of Pyls receiving it are treated as one could be different from the set of Pyls receiving Pyl distribution as a general rule.

[illegible]

© 2004 Blackwell Publishing Ltd

### Service resources

For Klemmerstein, a *barrier* is an abstract notion without definition. It signals a set of facts and a policy by which to decide how to overcome this problem or rationalize a certain behavior. The set of facts suggested by a barrier is usually interpreted by a decision. To learn about what we mean by defining these new phenomena, see <http://www.klemmerstein.com/2012/05/01/what-is-a-barrier/>.

The following comments are statements of legal positions and do not constitute an offer of insurance or financial advice. Please contact your broker.

The Service will publish answers to this document as

- ☐ [Send this page](#)
- ☐ [Print this page](#)
- ☐ [Download this page](#)



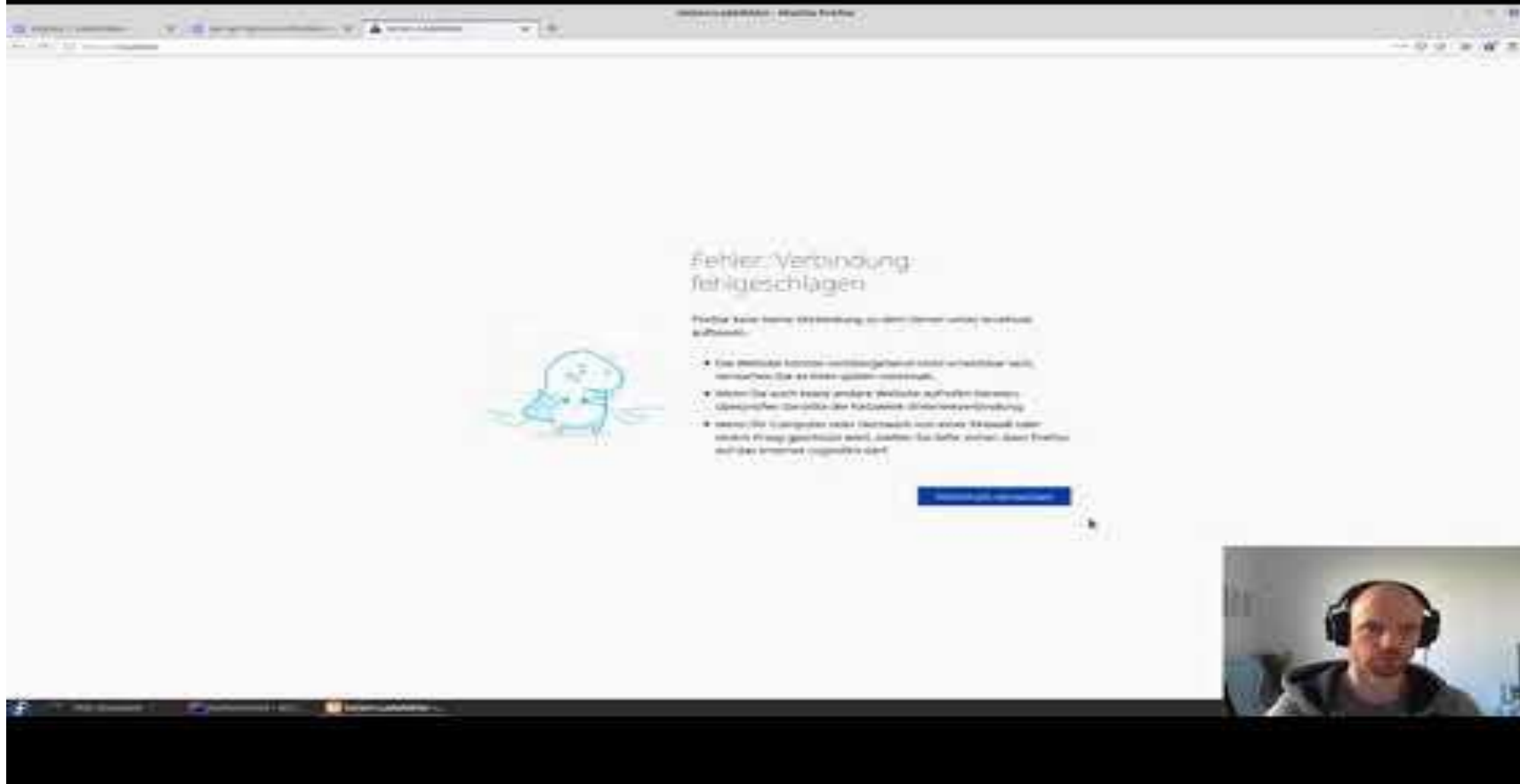
# Services

- <https://kubernetes.io/docs/concepts/services-networking/service/>
- Services machen Anwendungen in Pods für andere Pods im Kubernetes-Cluster zugreifbar
- Services verwenden Label selectors, um die Pods zu finden, auf die der Service verweist
- Services load-balancen zwischen mehreren Pods





# Live-Demo: Ingresses



<https://youtu.be/lkBag5znVjE>

# Ingress

“An API object that manages **external access to the services** in a cluster, typically HTTP. Ingress may provide load balancing, SSL termination and name-based virtual hosting”

<https://kubernetes.io/docs/concepts/services-networking/ingress/>

- Durch einen Ingress kann ein Service von außerhalb des Clusters erreicht werden



# Übung, Aufgabe 5

## 15 Minuten



# Live-Demo: Config Maps

### Define container environment variables using ConfigMap data

Define a container environment variable with data from a single ConfigMap:

† Differ significantly ( $P < 0.05$ ) from the control group.

doi:10.1371/journal.pone.0142811.g001

2. Except for the variables that were defined in the CopyrightMap to the JSTOR.org environment, all other variables in the final version of the map.

© 2004 Blackwell Publishing Ltd *Journal of Internal Medicine* 255: 103–110

```

10: # Create a new instance of the class
11: # and return it
12: def __new__(cls, *args, **kwargs):
13:     # Create a new instance of the class
14:     # and return it
15:     return super().__new__(cls, *args, **kwargs)
16:
17: # Create a new instance of the class
18: # and return it
19: def __init__(self, *args, **kwargs):
20:     # Create a new instance of the class
21:     # and return it
22:     super().__init__(*args, **kwargs)
23:
24: # Create a new instance of the class
25: # and return it
26: def __str__(self):
27:     # Create a new instance of the class
28:     # and return it
29:     return str(self)
30:
31: # Create a new instance of the class
32: # and return it
33: def __repr__(self):
34:     # Create a new instance of the class
35:     # and return it
36:     return repr(self)
37:
38: # Create a new instance of the class
39: # and return it
40: def __eq__(self, other):
41:     # Create a new instance of the class
42:     # and return it
43:     return self == other
44:
45: # Create a new instance of the class
46: # and return it
47: def __neq__(self, other):
48:     # Create a new instance of the class
49:     # and return it
50:     return self != other
51:
52: # Create a new instance of the class
53: # and return it
54: def __lt__(self, other):
55:     # Create a new instance of the class
56:     # and return it
57:     return self < other
58:
59: # Create a new instance of the class
60: # and return it
61: def __gt__(self, other):
62:     # Create a new instance of the class
63:     # and return it
64:     return self > other
65:
66: # Create a new instance of the class
67: # and return it
68: def __le__(self, other):
69:     # Create a new instance of the class
70:     # and return it
71:     return self <= other
72:
73: # Create a new instance of the class
74: # and return it
75: def __ge__(self, other):
76:     # Create a new instance of the class
77:     # and return it
78:     return self >= other
79:
80: # Create a new instance of the class
81: # and return it
82: def __hash__(self):
83:     # Create a new instance of the class
84:     # and return it
85:     return hash(self)
86:
87: # Create a new instance of the class
88: # and return it
89: def __getitem__(self, item):
90:     # Create a new instance of the class
91:     # and return it
92:     return self[item]
93:
94: # Create a new instance of the class
95: # and return it
96: def __setitem__(self, item, value):
97:     # Create a new instance of the class
98:     # and return it
99:     self[item] = value
100:
101: # Create a new instance of the class
102: # and return it
103: def __delitem__(self, item):
104:     # Create a new instance of the class
105:     # and return it
106:     del self[item]
107:
108: # Create a new instance of the class
109: # and return it
110: def __contains__(self, item):
111:     # Create a new instance of the class
112:     # and return it
113:     return item in self
114:
115: # Create a new instance of the class
116: # and return it
117: def __iter__(self):
118:     # Create a new instance of the class
119:     # and return it
120:     return iter(self)
121:
122: # Create a new instance of the class
123: # and return it
124: def __reversed__(self):
125:     # Create a new instance of the class
126:     # and return it
127:     return reversed(self)
128:
129: # Create a new instance of the class
130: # and return it
131: def __len__(self):
132:     # Create a new instance of the class
133:     # and return it
134:     return len(self)
135:
136: # Create a new instance of the class
137: # and return it
138: def __sizeof__(self):
139:     # Create a new instance of the class
140:     # and return it
141:     return self.__sizeof__()
142:
143: # Create a new instance of the class
144: # and return it
145: def __dir__(self):
146:     # Create a new instance of the class
147:     # and return it
148:     return dir(self)
149:
150: # Create a new instance of the class
151: # and return it
152: def __getattr__(self, name):
153:     # Create a new instance of the class
154:     # and return it
155:     return getattr(self, name)
156:
157: # Create a new instance of the class
158: # and return it
159: def __setattr__(self, name, value):
160:     # Create a new instance of the class
161:     # and return it
162:     setattr(self, name, value)
163:
164: # Create a new instance of the class
165: # and return it
166: def __delattr__(self, name):
167:     # Create a new instance of the class
168:     # and return it
169:     delattr(self, name)
170:
171: # Create a new instance of the class
172: # and return it
173: def __copy__(self):
174:     # Create a new instance of the class
175:     # and return it
176:     return copy.copy(self)
177:
178: # Create a new instance of the class
179: # and return it
180: def __deepcopy__(self, memo):
181:     # Create a new instance of the class
182:     # and return it
183:     return copy.deepcopy(self, memo)
184:
185: # Create a new instance of the class
186: # and return it
187: def __reduce__(self):
188:     # Create a new instance of the class
189:     # and return it
190:     return self.__reduce__()
191:
192: # Create a new instance of the class
193: # and return it
194: def __reduce_ex__(self, protocol):
195:     # Create a new instance of the class
196:     # and return it
197:     return self.__reduce_ex__(protocol)
198:
199: # Create a new instance of the class
200: # and return it
201: def __weakref__(self):
202:     # Create a new instance of the class
203:     # and return it
204:     return self.__weakref__
205:
206: # Create a new instance of the class
207: # and return it
208: def __subclasshook__(self, other):
209:     # Create a new instance of the class
210:     # and return it
211:     return self.__subclasshook__(other)
212:
213: # Create a new instance of the class
214: # and return it
215: def __subclasscheck__(self, other):
216:     # Create a new instance of the class
217:     # and return it
218:     return self.__subclasscheck__(other)
219:
220: # Create a new instance of the class
221: # and return it
222: def __subclasses__(self):
223:     # Create a new instance of the class
224:     # and return it
225:     return self.__subclasses__()
226:
227: # Create a new instance of the class
228: # and return it
229: def __bases__(self):
230:     # Create a new instance of the class
231:     # and return it
232:     return self.__bases__()
233:
234: # Create a new instance of the class
235: # and return it
236: def __base__(self):
237:     # Create a new instance of the class
238:     # and return it
239:     return self.__base__
240:
241: # Create a new instance of the class
242: # and return it
243: def __mro__(self):
244:     # Create a new instance of the class
245:     # and return it
246:     return self.__mro__
247:
248: # Create a new instance of the class
249: # and return it
250: def __mro_entries__(self, other):
251:     # Create a new instance of the class
252:     # and return it
253:     return self.__mro_entries__(other)
254:
255: # Create a new instance of the class
256: # and return it
257: def __getattribute__(self, name):
258:     # Create a new instance of the class
259:     # and return it
260:     return self.__getattribute__(name)
261:
262: # Create a new instance of the class
263: # and return it
264: def __setattribute__(self, name, value):
265:     # Create a new instance of the class
266:     # and return it
267:     self.__setattribute__(name, value)
268:
269: # Create a new instance of the class
270: # and return it
271: def __delattribute__(self, name):
272:     # Create a new instance of the class
273:     # and return it
274:     del self.__delattribute__(name)
275:
276: # Create a new instance of the class
277: # and return it
278: def __contains__(self, item):
279:     # Create a new instance of the class
280:     # and return it
281:     return item in self
282:
283: # Create a new instance of the class
284: # and return it
285: def __iter__(self):
286:     # Create a new instance of the class
287:     # and return it
288:     return iter(self)
289:
290: # Create a new instance of the class
291: # and return it
292: def __reversed__(self):
293:     # Create a new instance of the class
294:     # and return it
295:     return reversed(self)
296:
297: # Create a new instance of the class
298: # and return it
299: def __len__(self):
300:     # Create a new instance of the class
301:     # and return it
302:     return len(self)
303:
304: # Create a new instance of the class
305: # and return it
306: def __sizeof__(self):
307:     # Create a new instance of the class
308:     # and return it
309:     return self.__sizeof__()
310:
311: # Create a new instance of the class
312: # and return it
313: def __dir__(self):
314:     # Create a new instance of the class
315:     # and return it
316:     return dir(self)
317:
318: # Create a new instance of the class
319: # and return it
320: def __getattr__(self, name):
319:     # Create a new instance of the class
320:     # and return it
321:     return getattr(self, name)
322:
323: # Create a new instance of the class
324: # and return it
325: def __setattr__(self, name, value):
326:     # Create a new instance of the class
327:     # and return it
328:     setattr(self, name, value)
329:
330: # Create a new instance of the class
331: # and return it
332: def __delattr__(self, name):
333:     # Create a new instance of the class
334:     # and return it
335:     delattr(self, name)
336:
337: # Create a new instance of the class
338: # and return it
339: def __copy__(self):
340:     # Create a new instance of the class
341:     # and return it
342:     return copy.copy(self)
343:
344: # Create a new instance of the class
345: # and return it
346: def __deepcopy__(self, memo):
347:     # Create a new instance of the class
348:     # and return it
349:     return copy.deepcopy(self, memo)
350:
351: # Create a new instance of the class
352: # and return it
353: def __reduce__(self):
354:     # Create a new instance of the class
355:     # and return it
356:     return self.__reduce__()
357:
358: # Create a new instance of the class
359: # and return it
360: def __reduce_ex__(self, protocol):
361:     # Create a new instance of the class
362:     # and return it
363:     return self.__reduce_ex__(protocol)
364:
365: # Create a new instance of the class
366: # and return it
367: def __weakref__(self):
368:     # Create a new instance of the class
369:     # and return it
370:     return self.__weakref__
371:
372: # Create a new instance of the class
373: # and return it
374: def __subclasshook__(self, other):
375:     # Create a new instance of the class
376:     # and return it
377:     return self.__subclasshook__(other)
378:
379: # Create a new instance of the class
380: # and return it
381: def __subclasscheck__(self, other):
382:     # Create a new instance of the class
383:     # and return it
384:     return self.__subclasscheck__(other)
385:
386: # Create a new instance of the class
387: # and return it
388: def __subclasses__(self):
389:     # Create a new instance of the class
390:     # and return it
391:     return self.__subclasses__()
392:
393: # Create a new instance of the class
39
```

- [Share this page](#)
- [Create a new page](#)
- [Add a new column](#)



# Config Maps

- [Config Maps](#) werden verwendet, um eine Anwendung in Kubernetes zu konfigurieren
- Die Werte von Config-Maps können entweder als Umgebungsvariablen oder als Dateien im Container erscheinen



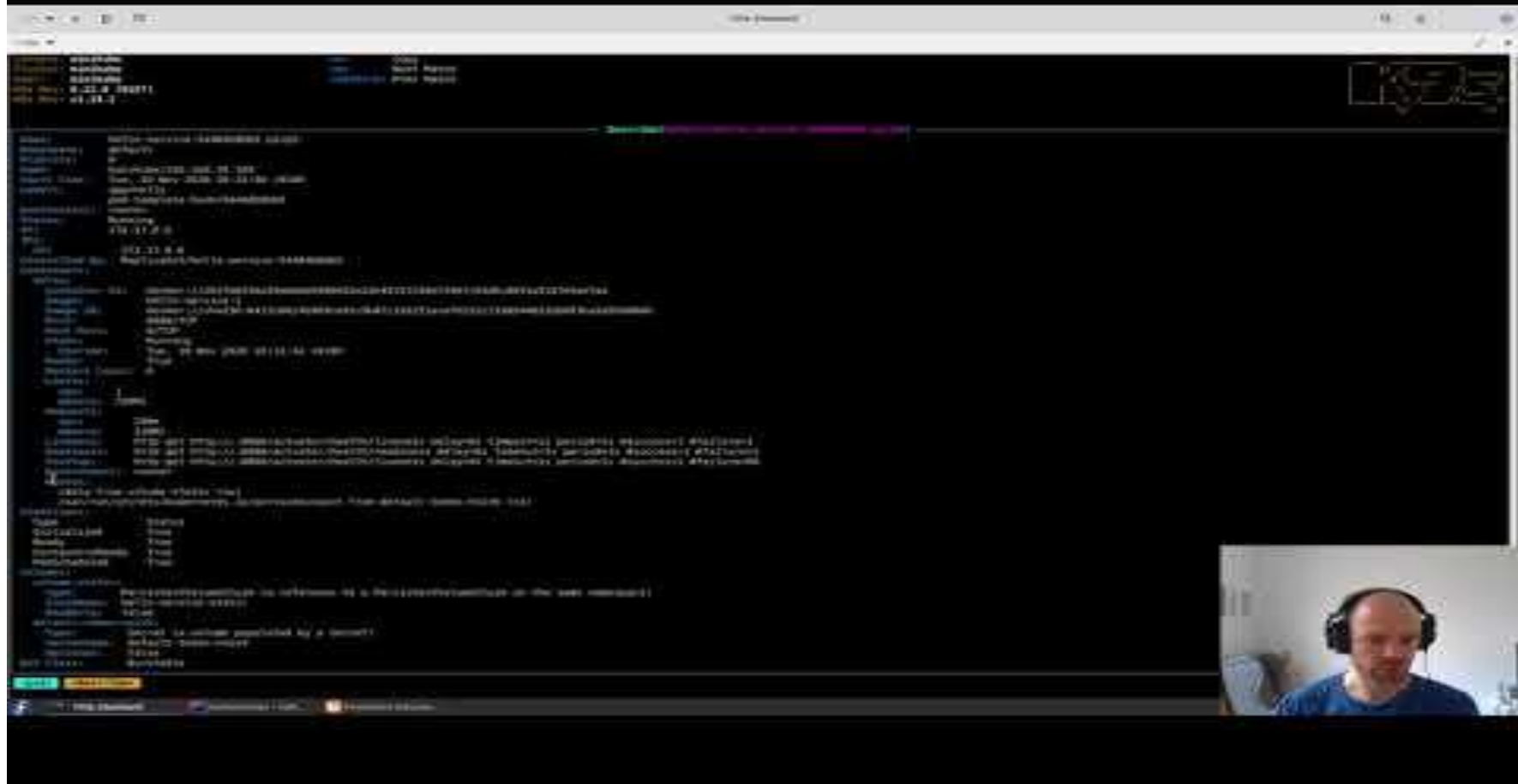
# Übung, Aufgabe 6

## 15 Minuten





# Live-Demo: Persistent Volumes



[https://youtu.be/P35\\_Ya2VxME](https://youtu.be/P35_Ya2VxME)

# Volumes

- Persistent Volumes stellen Speicherplatz bereit
  - Diese können entweder statisch oder dynamisch provisioniert werden
- Persistent Volume Claims fordern ein Persistent Volume an
- Über ein Volume kann ein Pod ein Persistent Volume Claim verwenden
- Volume mounts lassen das Volume in einem Container über einen Linux-Mount erscheinen



# Übung, Aufgabe 7

## 15 Minuten



# Live-Demo: Helm, der Package Manager

YouTube video player interface showing a video titled "The Document" with a duration of 1:15. The video content is a document with two columns of text, likely a transcript or a list of items. The text is partially obscured by a large black redaction box on the right side of the screen. A small inset video in the bottom right corner shows a person wearing a headset, presumably the speaker or a participant in the recording.

Document Content (Left Column):

- 1.1.1
- 1.1.2
- 1.1.3
- 1.1.4
- 1.1.5
- 1.1.6
- 1.1.7
- 1.1.8
- 1.1.9
- 1.1.10
- 1.1.11
- 1.1.12
- 1.1.13
- 1.1.14
- 1.1.15
- 1.1.16
- 1.1.17
- 1.1.18
- 1.1.19
- 1.1.20
- 1.1.21
- 1.1.22
- 1.1.23
- 1.1.24
- 1.1.25
- 1.1.26
- 1.1.27
- 1.1.28
- 1.1.29
- 1.1.30
- 1.1.31
- 1.1.32
- 1.1.33
- 1.1.34
- 1.1.35
- 1.1.36
- 1.1.37
- 1.1.38
- 1.1.39
- 1.1.40
- 1.1.41
- 1.1.42
- 1.1.43
- 1.1.44
- 1.1.45
- 1.1.46
- 1.1.47
- 1.1.48
- 1.1.49
- 1.1.50
- 1.1.51
- 1.1.52
- 1.1.53
- 1.1.54
- 1.1.55
- 1.1.56
- 1.1.57
- 1.1.58
- 1.1.59
- 1.1.60
- 1.1.61
- 1.1.62
- 1.1.63
- 1.1.64
- 1.1.65
- 1.1.66
- 1.1.67
- 1.1.68
- 1.1.69
- 1.1.70
- 1.1.71
- 1.1.72
- 1.1.73
- 1.1.74
- 1.1.75
- 1.1.76
- 1.1.77
- 1.1.78
- 1.1.79
- 1.1.80
- 1.1.81
- 1.1.82
- 1.1.83
- 1.1.84
- 1.1.85
- 1.1.86
- 1.1.87
- 1.1.88
- 1.1.89
- 1.1.90
- 1.1.91
- 1.1.92
- 1.1.93
- 1.1.94
- 1.1.95
- 1.1.96
- 1.1.97
- 1.1.98
- 1.1.99
- 1.1.100

Document Content (Right Column):

- 1.1.1
- 1.1.2
- 1.1.3
- 1.1.4
- 1.1.5
- 1.1.6
- 1.1.7
- 1.1.8
- 1.1.9
- 1.1.10
- 1.1.11
- 1.1.12
- 1.1.13
- 1.1.14
- 1.1.15
- 1.1.16
- 1.1.17
- 1.1.18
- 1.1.19
- 1.1.20
- 1.1.21
- 1.1.22
- 1.1.23
- 1.1.24
- 1.1.25
- 1.1.26
- 1.1.27
- 1.1.28
- 1.1.29
- 1.1.30
- 1.1.31
- 1.1.32
- 1.1.33
- 1.1.34
- 1.1.35
- 1.1.36
- 1.1.37
- 1.1.38
- 1.1.39
- 1.1.40
- 1.1.41
- 1.1.42
- 1.1.43
- 1.1.44
- 1.1.45
- 1.1.46
- 1.1.47
- 1.1.48
- 1.1.49
- 1.1.50
- 1.1.51
- 1.1.52
- 1.1.53
- 1.1.54
- 1.1.55
- 1.1.56
- 1.1.57
- 1.1.58
- 1.1.59
- 1.1.60
- 1.1.61
- 1.1.62
- 1.1.63
- 1.1.64
- 1.1.65
- 1.1.66
- 1.1.67
- 1.1.68
- 1.1.69
- 1.1.70
- 1.1.71
- 1.1.72
- 1.1.73
- 1.1.74
- 1.1.75
- 1.1.76
- 1.1.77
- 1.1.78
- 1.1.79
- 1.1.80
- 1.1.81
- 1.1.82
- 1.1.83
- 1.1.84
- 1.1.85
- 1.1.86
- 1.1.87
- 1.1.88
- 1.1.89
- 1.1.90
- 1.1.91
- 1.1.92
- 1.1.93
- 1.1.94
- 1.1.95
- 1.1.96
- 1.1.97
- 1.1.98
- 1.1.99
- 1.1.100

# Helm

- <https://helm.sh/>
- Paket-Manager für Kubernetes
- Erlaubt das Installieren von Charts in den Cluster
- Charts gibt es für alles mögliche: Redis, PostgreSQL, etc.
- Es gibt eine [Suchmaschine für Charts](#)
- Charts können über den `--set` oder den `--values` Parameter angepasst werden



# Übung, Aufgabe 8

## 20 Minuten