In diesem Kapitel werden wir uns mit Docker-Containern im Kubernetes-Netzwerk beschäftigen. Das von Google entwickelte und seit 2014 als Open Source verfügbare Softwarepaket ermöglicht eine flexible Verwaltung von Anwendungen auf Container-Basis. Kubernetes oder k8s, wie es oft abgekürzt geschrieben wird, ist dabei nicht auf Docker als Container-Format beschränkt, sondern kann auch andere Container-Runtimes ausführen (zum Beispiel rkt oder runc).
Ein Kubernetes-Netzwerk besteht aus einem oder mehreren Nodes, die entweder physische Computer oder virtuelle Maschinen sein können. Auf jedem dieser Nodes laufen zumindest drei Services:

- kubelet: koordiniert die Ausführung der Pods, in denen Container laufen.
- Container Runtime: führt die Container aus.
- Kube Proxy: verwaltet Netzwerkregeln für den Node.

Im Unterschied zu Docker gibt es bei Kubernetes eine weitere Abstraktionsebene zwischen dem Betriebssystem und den einzelnen Prozessen: Pods. In einem Pod können ein oder mehrere Container laufen. Container innerhalb eines Pods teilen sich eine IP-Adresse und können über localhost miteinander kommunizieren. In vielen Fällen läuft in einem Pod aber nur ein Container. Sie können sich den Pod dann wie eine weitere Schicht um den Container vorstellen, die in der Kubernetes-Welt gebraucht wird.

Was ist ein Orchestrator

Nehmen wir ein Orchester. Eine Gruppe einzelner Musiker, die unterschiedliche Musikinstrumente spielen. Jeder Musiker und jedes Instrument hat eine eigene Rolle zu spielen, wenn die Musik losgeht. Es gibt Violinen, Cellos, Harfen, Oboen, Flöten, Klarinetten, Trompeten, Posaunen, Trommeln und sogar Triangeln. Jeder ist einzigartig und hat eine essenzielle Rolle im Orchester.
Jedes Instrument individuell und hat keine zugeteilte Rolle – ein Riesendurcheinander.
Eine Dirigentin kommt mit den Notenblättern daher und sorgt für Ordnung. Sie gruppiert die Streichinstrumente in die ersten Reihen, Holzbläser in die Mitte, Blechbläser weiter hinten und Schlagzeug hoch oben und ganz nach hinten. Sie dirigiert ausserdem das Ganze – zeigt jeder Gruppe an, wann zu spielen ist, wie laut oder leise, wie schnell oder langsam.
Kurz gesagt, die Dirigentin nimmt das Chaos, stellt die Ordnung her, und produziert wunderschöne Musik dabei.
Microservices sind genau wie ein Orchester. Jede cloud-native App besteht ja aus vielen kleinen Microservices, die unterschiedliche Aufgaben haben. Manche bedienen Webanfragen, andere authentifizieren Sitzungen, protokollieren, speichern Daten. Aber genau wie beim Orchester brauchen sie jemanden oder etwas, das sie in eine nutzbare App organisiert.
Genau das macht Kubernetes, es nimmt das Durcheinander eigenständiger Microservices und organisiert sie in eine sinnvolle App. Wie bereits erwähnt, kann es die App skalieren, heilen, aktualisieren, und so weiter.
Unter Orchestrierung verstehen wir also:

- Erstellen, Löschen, Neustart von Containern
- Container-Startreihenfolge managen
- Container-Neustart nach Absturz

Der **API Server** ist der einzige Teil eines Kubernetes Clusters, mit dem man direkt interagiert. Wenn man Befehle an den Cluster schickt, gehen diese an den API-Server. Wenn man eine Antwort bekommt, dann kommt diese vom API Server.
Der **Scheduler** berücksichtigt den Ressourcenbedarf eines Pods, wie die CPU oder den Arbeitsspeicher, sowie den Zustand des Clusters. Anschliessend wird der Pod für einen geeigneten Rechenknoten geplant.
Der **etcd Key/Value-Store** speichert den aktuellen Status des Clusters und aller Anwendungen.
Der **Controller Manager** schaut, was im Cluster alles vor sich geht, z.B. ob etwas repariert werden muss, ob ein Container abgestürzt ist, und neu gestartet werden muss, usw.
Der **Cloud Controller** erlaubt es Kubernetes, sich mit Cloud Services wie zum Beispiel Storage und Load Balancern zu integrieren.

Master und Worker Nodes

Aktuell ist Kubernetes daran dieses Prinzip aufzuweichen, sodass zukünftig alle Nodes gleichwertig sind, trotzdem ist es im Moment noch wichtig der Unterschied von Master und Worker zu kennen.
Ein Kubernetes Cluster ist eine beliebige Menge an Geräten mit installiertem Kubernetes. Die Geräte können physische Server, virtuelle Maschinen, Cloud Instanzen, Laptops, Raspberry Pis und vieles mehr sein. Wenn man Kubernetes auf mehreren dieser Maschinen installiert und sie miteinander verbindet, hat man ein Kubernetes Cluster. Man kann dann Anwendungen in dem Cluster deployen.
Für gewöhnlich nennt man Maschinen in einem Kubernetes Cluster Nodes.
Ein Kubernetes Cluster hat zwei Sorten von Nodes:

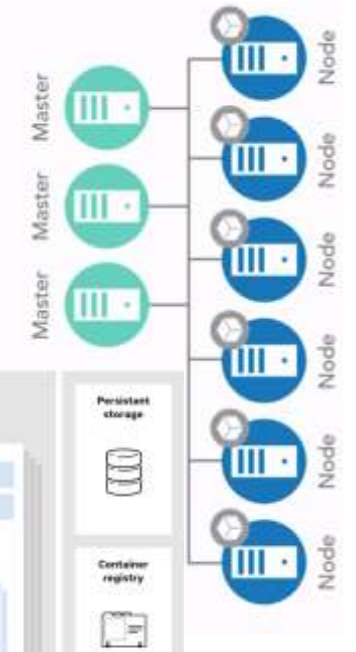
- Master Nodes
- Worker Nodes

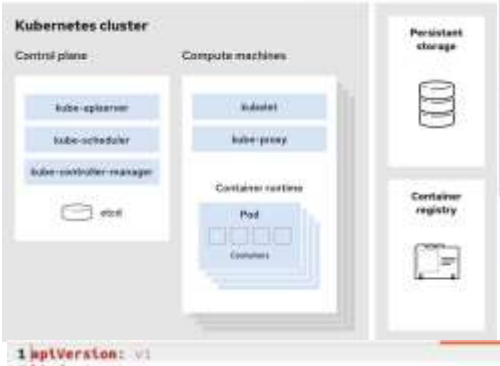
Auf der offiziellen Seite kubernetes.io werden Sie aber vergeblich nach einem Download-Button fahnden. Mit Kubernetes verhält es sich wie mit dem Linux-Kernel: Den können Sie auch irgendwo aus einem schmucklosen Archiv herunterladen, werden damit aber zunächst nur sehr wenig anstellen können. Wie auch Linux wollen Sie Kubernetes in Form einer Kubernetes-Distribution haben. Die bündelt all das, was zum Betrieb notwendig ist und verdrahtet die Komponenten schon mal sinnvoll. Kubernetes-Distributionen gibt es mittlerweile viele und ihre Wahl ist zur Wissenschaft geworden. Die meistgenutzten fallen für Selbstbetreiber schon mal raus, sie heissen GKE (Google Kubernetes Engine) AKS (Azure Kubernetes Service) und EKS (Amazon Elastic Kubernetes Service) und stecken in den Managed-Kubernetes-Angeboten der drei Branchenriesen im Cloudgeschäft.
Vor allem in der Entwicklungs- oder Testphase Ihrer Anwendung kann die Ausführung von K8s mühsam sein, und die Nutzung eines verwalteten Kubernetes-Dienstes kann kostspielig sein. Um den Betrieb von Kubernetes, insbesondere in Entwicklungs- und Testumgebungen, zu vereinfachen, benötigen wir ein Tool, das diese Komplexität reduziert. Heutzutage gibt es viele Tools, die behaupten, den Zweck von Kubernetes in einfacherer Form für kleinere Umgebungen zu erfüllen. Mit solchen Tools können Kubernetes-Entwickler ihre Anwendungen einfach testen und sicherstellen, dass die Dinge in der Produktion genauso gut funktionieren wie in der Entwicklungs-/Testumgebung. Von diesen Tools sind minikube, microk8s, kind und k3s einige der vertrauenswürdigsten, die die Erwartungen erfüllen.

Was bedeutet cloud-native?

Eine cloud native App muss folgendes unterstützen:

- Skalierung nach Bedarf
- Self-healing
- Rolling Updates ohne Ausfallzeit
- Funktion wo auch immer es Kubernetes gibt






```yaml
apiVersion: v1
kind: Service
metadata:
  name: todo-app-service
  labels:
    name: todo-app
spec:
  selector:
    app: todo-app
  type: LoadBalancer
  # loadBalancerIP is optional. MetalLB will automatically allocate an IP
  # from its pool if not specified. You can also specify one manually.
  # loadBalancerIP: x.y.z.a
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 3000
    - name: https
      protocol: TCP
      port: 443
      targetPort: 3000
```

Dashboard

Mit `microk8s dashboard-proxy` starten wir das Dashboard.

```
$microk8s kubectl scale --replicas=1 rc redis-slave
```

docker network create todoapp_network

```
docker run --net=todoapp_network --name=redis-master -d redis-master:v1
docker run --net=todoapp_network --name=redis-slave -d redis-slave:v1
docker run --net=todoapp_network --name=frontend -d -p 3000:3000 todo-app:v1
```

```
1  FROM nginx:1.10.1-alpine
2  COPY index.html /usr/share/nginx/html
3  EXPOSE 8080
4  CMD ["nginx", "-g", "daemon off;"]
```

1. Basic Nginx Dockerfile:

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
```

Vendor Lock-in

Ein Begriff, den Sie ebenfalls kennen sollten, ist Vendor Lock-in.
Hat man sich nach gründlichem Nachdenken für eine Software entschieden, bleibt immer eine Angst: Was tun, wenn der Anbieter nächstes Jahr das Angebot einstellt oder unverhältnismässig verteuert? Dahinter steckt die Befürchtung, dass man mit einer Entscheidung in einem Herstelleruniversum eingesperrt ist. Mit Open-Source-Software aus dem Cloud-Native-Umfeld kann man dem grösstenteils aus dem Weg gehen. Wer sich etwa für Kubernetes als Plattform für seine Container entscheidet und dafür ein gemanagtes Kubernetes anmietet, kann die Umgebung zügig von einem zum anderen Anbieter verschieben. Vendor Lock-in gibt es aber auch bei Cloudanbietern, wenn man sich für Spezallösungen abseits von Speicherplatz und Rechenleistung entscheidet. Dazu gehören einige Datenbanken oder Dienste aus dem KI-Bereich, die man bei Cloudanbietern anmieten kann. Wer Vendor Lock-in vermeiden will, sollte nur Server und Speicherplatz mieten und damit Open-Source-Software selbst betreiben. Das macht mehr Arbeit beim Betrieb, verhindert später aber Probleme beim Umzug.

Semantic Versioning:

Semantic Versioning: Software-Versionsnummern sind meist nicht willkürlich, sondern folgen einem Schema. Besonders im schnelllebigen Cloudumfeld ist das wichtig. Die Versionsnummer 2.5.11 etwa besteht aus der Major-Version 2, der Minor-Version 5 und der Patch-Version 11. Patches sind kleinere Reparaturen, die Sicherheitsprobleme beseitigen oder Fehler abstellen. Minor-Änderungen bringen neue Funktionen mit, verändern aber keine bestehenden Funktionen. Entscheiden sich Entwickler für ein neues Major-Update, kann es elementare Änderungen an der Software geben, die manuelle Arbeit beim Umstieg erfordern. Wer Software im Cloudumfeld nutzt, sollte dieses Schema verinnerlichen und zum Beispiel nicht ohne Tests eine neue Major-Version einer Datenbank einsetzen.
Nun wollen wir aber loslegen: Schiff ahoi!

Bei **Deployments** wie wir es bei der App selber haben, können wir einfach in der Datei todo-app-deploy.yaml die Anzahl bei replicas auf 2 verringern und die Datei mit:

```
$ microk8s kubectl apply -f todo-app-deploy-v2.yaml
```

2. Nginx with SSL Dockerfile:

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
COPY ssl.crt /etc/nginx/ssl.crt
COPY ssl.key /etc/nginx/ssl.key
```

3. Nginx with PHP-FPM Dockerfile:

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
COPY site.conf /etc/nginx/conf.d/
RUN apt-get update && apt-get install -y \
    php-fpm \
    && rm -rf /var/lib/apt/lists/*
```

Welches Problem löst Docker?

Docker ermöglicht schnelles Prototyping (neue Infrastrukturen können nach kurzer Zeit getestet werden) neuer Anwendungen und unkompliziertes Skalierung bestehender Anwendungen. (LeistungLeistung der Container können schnell und einfach verändert werden). Genau das wollen wir aktuell mit DevOps erreichen. Das Kürzel DevOps steht für das Ziel, Software schneller und qualitativ hochwertiger entwickeln und ausliefern zu können. DevOps soll Entwicklung (Development), IT Betrieb (Operations) und Qualitätssicherung in den Unternehmen zusammenführen.

Best Build Practices

- Container sind kurzlebig und jederzeit reproduzierbar
- Wer hat's gemacht? (MAINTAINER bzw. LABEL-Direktive)
- Wie ist es bezeichnet? (--tag)
- Verwenden eines eigenen Build-Ordners pro Template
- Verwendung eines .dockerignore-Files
- Schlanke Images (Ein Image sollte klein, simpel und so schlank/kompakt wie möglich sein.)
- Nur ein Prozess pro Container
- Anzahl der Layer minimieren/niedrig halten

Der Syntax dazu ist:

`docker image tag name-des-images:v1 pfad-zu-registry/ordner-für-image/name-des-image:v1`

Taggen Sie alle drei Images, gemäss ihrem Projekt und ihrer gewünschten Ordnerstruktur. Hier ein Beispiel:

`docker image tag redis-master:v1 registry.gitlab.com/thomas-staub/m347todo/to-do-app/redis-master:v1`

```
sudo docker login -u nicolas.dumermuth -p (gitlab-access-token)
sudo docker image tag todo-app:v2 gitlab.iet-gibb.ch:5050/nicolas.dumermuth/m169_inf2021b_nicolas_dumermuth/todo-app:v2
sudo docker push gitlab.iet-gibb.ch:5050/nicolas.dumermuth/m169_inf2021b_nicolas_dumermuth/todo-app:v2
sudo docker run -d -p:3000:3000 --name frontend gitlab.iet-gibb.ch:5050/nicolas.dumermuth/m169_inf2021b_nicolas_dumermuth/todo-app:v2
```

Run a new Container	Manage Containers	Manage Images	Info & Stats

Start a new Container from an image
`docker run IMAGE`
`docker run nginx`
	Show a list of running containers
	`docker ps`
		Download an image
		`docker pull IMAGE[:TAG]`
		`docker pull nginx`
			Show the logs of a container
			`docker logs CONTAINER`
			`docker logs web`

...and assign it a name
`docker run --name CONTAINER IMAGE`
`docker run --name web nginx`
	Show a list of all containers
	`docker ps -a`
		Upload an image to a repository
		`docker push IMAGE`
		`docker push myimage:1.0`
			Show stats of running containers
			`docker stats`

...and map a port
`docker run -p HOSTPORT:CONTAINERPORT IMAGE`
`docker run -p 8080:80 nginx`
	Delete a container
	`docker rm CONTAINER`
	`docker rm web`
		Delete an image
		`docker rmi IMAGE`
			Show processes of container
			`docker top CONTAINER`
			`docker top web`

...and map all ports
`docker run -P IMAGE`
`docker run -P nginx`
	Delete a running container
	`docker rm -f CONTAINER`
	`docker rm -f web`
		Show a list of all images
		`docker images`
			Show installed docker version
			`docker version`

...and start container in background
`docker run -d IMAGE`
`docker run -d nginx`
	Delete stopped containers
	`docker container prune`
		Delete dangling images
		`docker image prune`
			Get detailed info about an object
			`docker inspect NAME`
			`docker inspect nginx`

...and assign it a hostname
`docker run --hostname HOSTNAME IMAGE`
`docker run --hostname srv nginx`
	Stop a running container
	`docker stop CONTAINER`
	`docker stop web`
		Delete all unused images
		`docker image prune -a`
			Show all modified files in container
			`docker diff CONTAINER`
			`docker diff web`

...and add a dns entry
`docker run --add-host HOSTNAME:IP IMAGE`
	Start a stopped container
	`docker start CONTAINER`
	`docker start web`
		Build an image from a Dockerfile
		`docker build DIRECTORY`
		`docker build .`
			Show mapped ports of a container
			`docker port CONTAINER`
			`docker port web`

...and map a local directory into the container
`docker run -v HOSTDIR:TARGETDIR IMAGE`
`docker run -v ~/:/usr/share/nginx/html nginx`
	Copy a file from a container to the host
	`docker cp CONTAINER:SOURCE TARGET`
	`docker cp web:/index.html index.html`
		Tag an image
		`docker tag IMAGE NEWIMAGE`
		`docker tag ubuntu ubuntu:18.04`	

...but change the entrypoint
`docker run -it --entrypoint EXECUTABLE IMAGE`
`docker run -it --entrypoint bash nginx`
	Copy a file from the host to a container
	`docker cp TARGET CONTAINER:SOURCE`
	`docker cp index.html web:/index.html`
		Build and tag an image from a Dockerfile
		`docker build -t IMAGE DIRECTORY`
		`docker build -t myimage .`	

	Start a shell inside a running container
	`docker exec -it CONTAINER EXECUTABLE`
	`docker exec -it web bash`
		Save an image to .tar file
		`docker save IMAGE > FILE`
		`docker save nginx > nginx.tar`	

	Rename a container
	`docker rename OLD_NAME NEW_NAME`
	`docker rename 096 web`
		Load an image from a .tar file
		`docker load -i TARFILE`
		`docker load -i nginx.tar`	

	Create an image out of container
	`docker commit CONTAINER`
	`docker commit web`		

`docker system prune --all --volumes`

docker network ls

Remove one or more networks:

docker network rm [NETWORK]

Show information on one or more networks:

docker network inspect [NETWORK]

Connects a container to a network:

docker network connect [NETWORK] [CONTAINER]

Disconnect a container from a network:

docker network disconnect [NETWORK] [CONTAINER]

```yaml
services:
  portainer:
    image: portainer/portainer-ce:latest
    container_name: portainer
    restart: unless-stopped
    security_opt:
      - no-new-privileges:true
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - ./portainer-data:/data
    ports:
      - 9000:9000
```

```yaml
version: "3"
services:
  todoapp:
    build: ./web-frontend
    ports:
      - "3000:3000"
    depends_on:
      - redis-master
      - redis-slave
  redis-slave:
    build: ./redis-slave
    depends_on:
      - redis-master
  redis-master:
    build: ./redis-master
```

Vergewissern Sie sich, dass alle nötigen Addons installiert sind.

`microk8s enable dashboard dns registry ingress`

Mit kubectl get node können wir kontrollieren, ob das Cluster erfolgreich gestartet wurde

`$ microk8s kubectl get nodes`

Folgender Befehl gibt ihnen die Informationen zum Cluster zurück.

`$ microk8s kubectl cluster-info`

Im nächsten Schritt installieren wir einen Loadbalancer.

`$ microk8s enable metallb`

Sie werden nach einem IP Range gefragt. Geben Sie hier die IP ihres Clusters ein. Sofern Sie alles lokal eingerichtet haben, verwenden Sie 127.0.0.172-127.0.0.240

`$ microk8s kubectl get componentstatus`

Wir starten nunden Redis Master Replication-Controller, dies kann einige Zei

`$ microk8s kubectl create -f ./redis-master-controller.yaml`

\# Überprüfung ob der Redis-Master Pod gestartet wurde

`$ microk8s kubectl get pods`

\# Überprüfen ob der Service erstellt wurde

`$ microk8s kubectl get service`

redis-master-controller.yaml

```yaml
kind: ReplicationController
apiVersion: v1
metadata:
  name: redis-master
  labels:
    name: redis-master
spec:
  replicas: 1
  selector:
    name: redis-master
  template:
    metadata:
      labels:
        name: redis-master
    spec:
      containers:
      - name: redis-master
        image: staubth/redis-master:v1
        ports:
        - containerPort: 6379
          protocol: TCP
```

todo-app-deploy-v2.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-app-deployment
  labels:
    app: todo-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: todo-app

  minReadySeconds: 20
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: todo-app
    spec:
      containers:
      - name: todo-app
        image: staubth/todo-app:v2
        ports:
        - containerPort: 3000
          protocol: TCP
```

```yaml
kind: Service
apiVersion: v1
metadata:
  name: redis-slave
  labels:
    name: redis-slave
spec:
  ports:
  - port: 6379
    targetPort: 6379
    protocol: TCP
  selector:
    name: redis-slave
```

```
docker-compose -f deploy/docker-compose/docker-compose.yml up -d
```

Schlüsselwort	Bedeutung
ADD	kopiert Dateien in das Dateisystem des Images.
CMD	führt das angegebene Kommando beim Container-Start aus.
COPY	kopiert Dateien aus dem Projektverzeichnis in das Image.
ENTRYPOINT	führt das angegebene Kommando immer beim Container-Start aus.
ENV	setzt eine Umgebungsvariable.
EXPOSE	gibt die aktiven Ports des Containers an.
FROM	gibt das Basis-Image an.
LABEL	legt eine Zeichenkette fest.
RUN	führt das angegebene Kommando aus.
USER	gibt den Account für RUN, CMD und ENTRYPOINT an.
VOLUME	gibt Volume-Verzeichnisse an.
WORKDIR	legt das Arbeitsverzeichnis für RUN, CMD, COPY etc. fest.

```dockerfile
FROM alpine:3.2
MAINTAINER Johannes M. Scheuermann <johannes.scheuermann@inovex.de>

COPY ./bin/todo-app /app/todo-app
COPY ./public /app/public

WORKDIR /app
CMD ["./todo-app"]
EXPOSE 3000
```

FROM
...
RUN
...
COPY
...
CMD

Virtuelle Maschine	Docker-Container
Prozessisolation auf Hardwareebene	Prozessisolation auf Betriebssystemebene
Jede VM verfügt über ein separates Betriebssystem	Jeder Container kann das Betriebssystem gemeinsam nutzen
Bereit in Minuten	Bereit in Sekunden
VMs haben nur wenige GB	Container sind leicht (KBs / MBs)
Vorgefertigte VMs sind schwer zu finden	Vorgefertigte Docker-Container sind leicht verfügbar
VMs können problemlos auf einen neuen Host verschoben werden	Container werden zerstört und neu erstellt, anstatt sich zu bewegen
Das Erstellen einer VM dauert relativ lange	Container können in Sekunden erstellt werden
Mehr Ressourcennutzung	Weniger Ressourcenverbrauch