CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Container-Orchestrierung

(5) Kubernetes (Teil III: Weitere Workloads)

PROF. DR.
NANE KRATZKE

creative commons

CLOUD NATIVE COMPUTING

Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.

KAPITEL 9

Container-Plattformen

9.1 Scheduling

- Heterogenität von Workloads
- Scheduling-Algorithmen
- Scheduling-Architekturen

9.2 Orchestrierung

- Definition von Betriebszuständen
- Regelkreis: Desired vs Current State

9.3 Inside Kubernetes

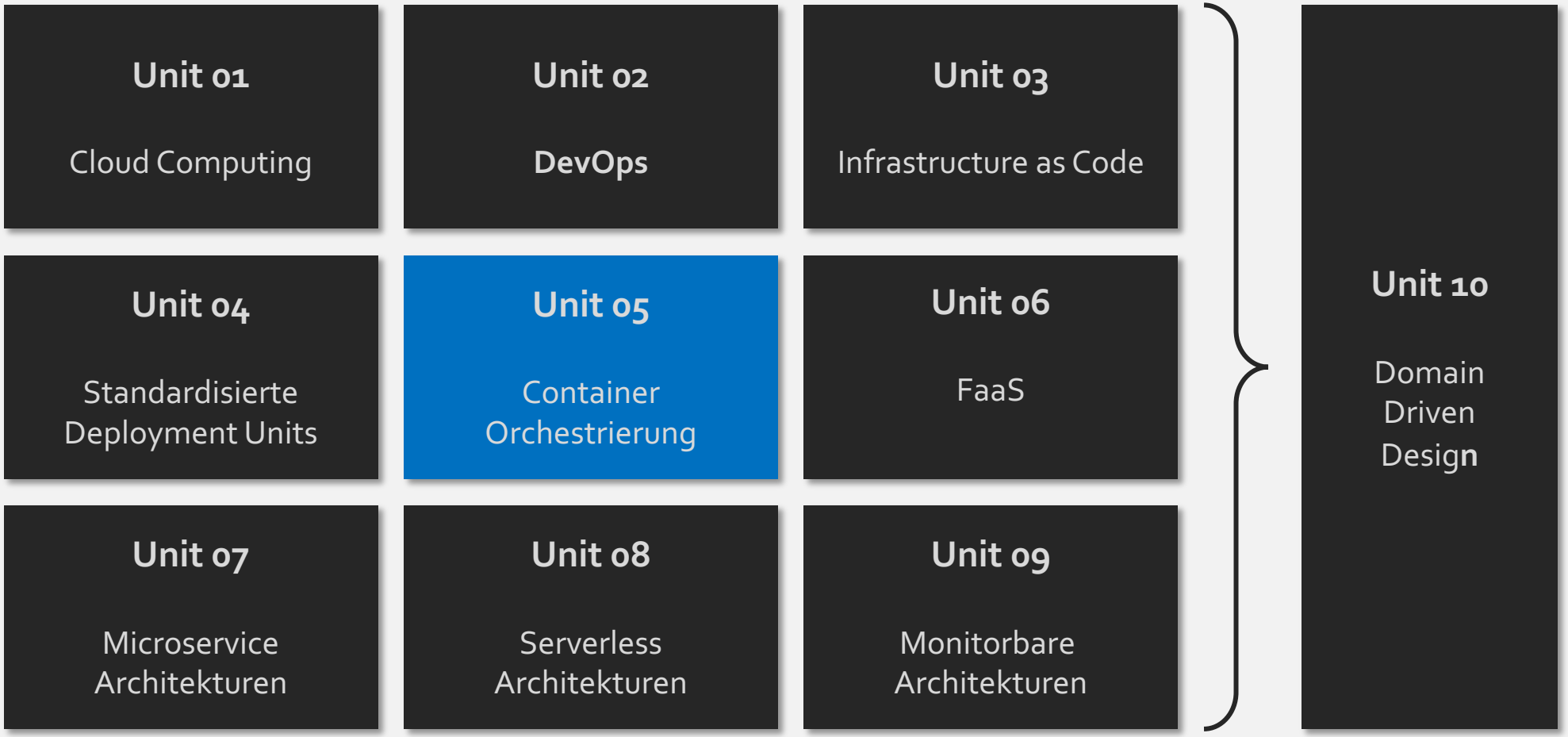
- Kubernetes-Architektur und Ressourcen
- Workloadarten
- Scheduling Constraints
- Automatische Skalierung von Workloads
- Exponierung von Services
- Health Checking
- Persistenz
- Isolation von Workloads

9.4 Zusammenfassung

CLOUD NATIVE COMPUTING

INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



KUBERNETES

Überblick über die wichtigsten Kubernetes-Ressourcen

CLOUD NATIVE COMPUTING

Ausführbare Jobs/Workloads		Persistenz	Isolieren von Anwendungen mittels Namespaces	Exponieren von Anwendungen und Diensten
				Primär Intra-Cluster 
Deployments	**(Cron-)Jobs**	Persistent Volume Claim	Quotas und Limit Ranges	Services
				Primär Extra-Cluster 
Daemon Sets	Stateful Sets	Persistent Volume	Role Based Access Model	Ingress

JOBS

Ausführbare Workloads

Jobs sind meist länger laufende Aufgaben (z.B. das Trainieren eines neuronalen Netzes), die einmalig ausgeführt werden.

Ein Job erzeugt hierzu ein oder mehrere Pods und trägt Sorge dafür, dass eine vorgegebene Anzahl von diesen erfolgreich terminiert. Es können mehrere Pods parallel ausgeführt werden.

Wird ein Job gelöscht, werden auch alle von ihm erzeugten Pods gelöscht.

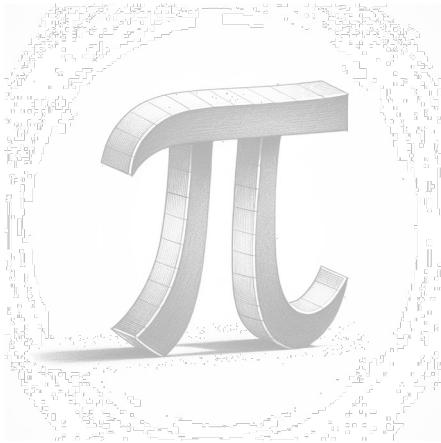
```
# Ausbringen eines Jobs
# (z.B. zur Berechnung der ersten 2000 Nachkommastellen von PI)
kubectl apply -f xample-job.yaml

# Einsehen des Berechnungsergebnisses in den Logs
kubectl logs -l job-name=pi

# Löschen des Jobs
kubectl delete jobs/pi
```

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(100)"]
      restartPolicy: Never      # Container nie neu starten!
```

Manifest: xample-job.yaml

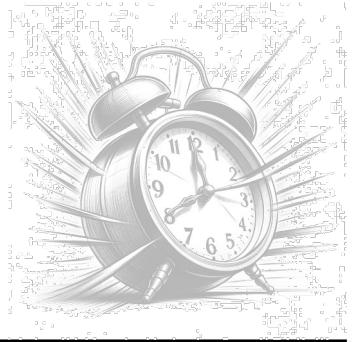


Quelle: https://kubernetes.io/docs/concepts/workloads/controllers/job/

CRON-JOBS

Ausführbare Workloads

CLOUD NATIVE COMPUTING



Cron-Jobs sind periodisch auszuführende Aufgaben (z.B. Erstellung einer Monatsabrechnung).

Die Periodizität von Jobs wird dabei nach dem gleichen Schema angegeben, dass auch von Linux/UNIX-Cron-Jobs bekannt ist (* * * * *).

Ein Cron-Job erzeugt dazu zu den so festgelegten Zeitpunkten Jobs, die wie beschrieben ausgeführt werden.

Manifest: xample-cron-job.yaml

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/5 * * * *"
  jobTemplate:
    spec:
      template:
        metadata: { labels: { cron: job }}
        spec:
          containers:
          - name: hello
            image: busybox
            args:
              - /bin/sh
              - -c
              - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

```
# A crontab file has five fields for specifying:

*   *   *   *   *   command to be executed
-   -   -   -   -
|   |   |   |   |
|   |   |   |   +---------------   DAY OF WEEK   (0-6) (Sunday=0)
|   |   |   +-------------   MONTH   (1-12)
|   |   +---------   DAY OF MONTH   (1-31)
|   +------   HOUR   (0-23)
+---   MINUTE   (0-59)
```

Dieser Beispiel-CronJob gibt alle fünf Minuten die aktuelle Uhrzeit und eine Hallo-Nachricht aus.

Quelle: https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

KUBERNETES

Überblick über die wichtigsten Kubernetes-Ressourcen

CLOUD NATIVE COMPUTING

Ausführbare Jobs/Workloads	Persistenz	Isolieren von Anwendungen mittels Namespaces	Exponieren von Anwendungen und Diensten
 Deployments	 Persistent Volume Claim	 Quotas und Limit Ranges	Primär Intra-Cluster  Services
 (Cron-)Jobs			
 **Daemon Sets**	 Persistent Volume	 Role Based Access Model	Primär Extra-Cluster  Ingress
 Stateful Sets			

DAEMON SETS

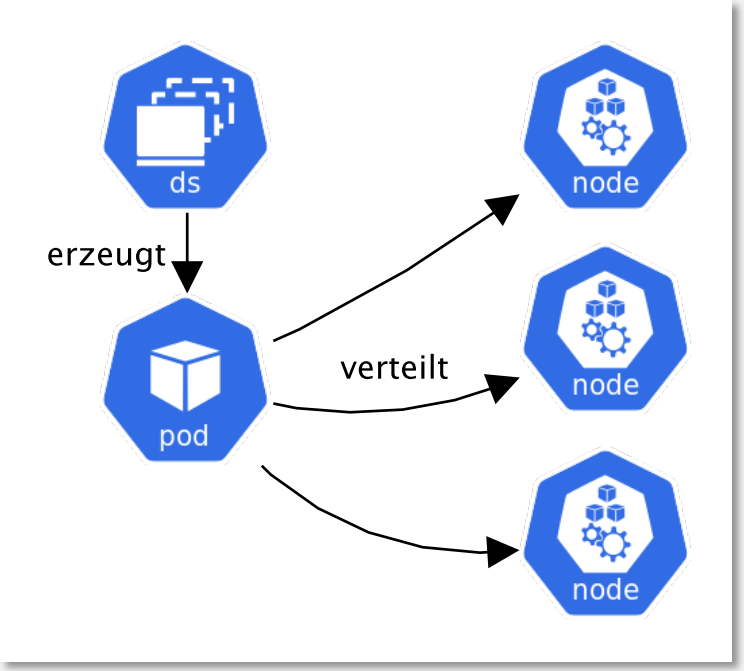
Ausführbare Workloads

Ein Daemon Set stellt sicher, dass auf allen (oder ausgewählten) Nodes der Kubernetes Infrastruktur eine Kopie eines Pods läuft.

Werden Nodes dem Cluster hinzugefügt, werden so auch Pods auf diesen automatisch ausgeführt. Werden Nodes entfernt, werden entsprechende Pods durch den Garbage Collector entfernt.

Typische Anwendungsfälle sind häufig zentrale Dienste des Clusters (Kubernetes Add-Ons) wie bspw.:

- Clustered Storage Daemons pro Cluster Node
- Log Collection Daemon pro Cluster Node
- Node Monitoring Daemon pro Cluster Node
- Ingress Controller pro Cluster Node



Quelle: https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

DAEMON SETS

CLOUD NATIVE COMPUTING

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

DaemonSet

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx-daemonset
  labels:
    app: nginx
spec:

  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

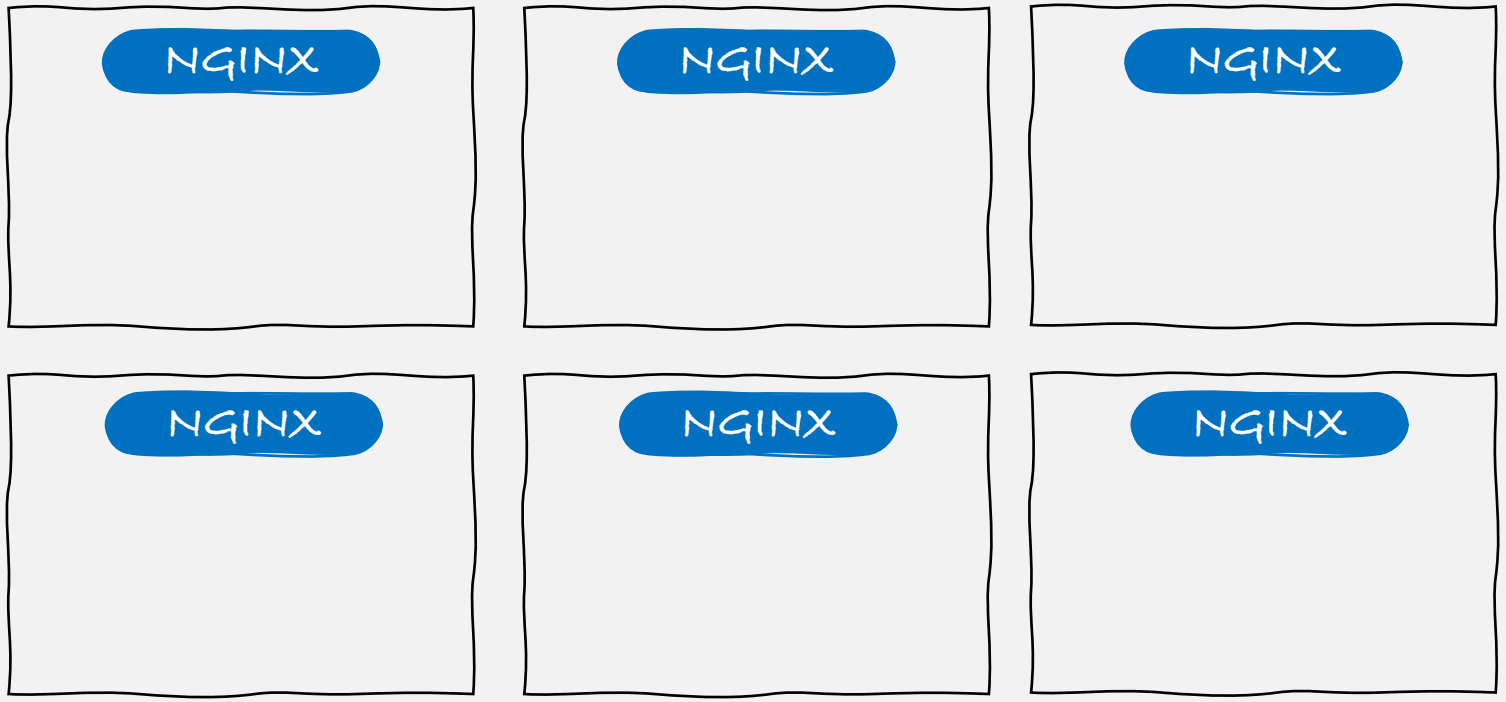
Deployments und Daemon-Sets sind auf den ersten Blick sehr ähnlich, aber …

Deployments adressieren eher Nutzer

DaemonSets adressieren eher Administratoren

SCHEDULING

Von Daemon Sets




```
> kubectl apply -f nginx-daemon.yaml
```

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx-daemonset
  labels:
    app: nginx
spec:

  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

CLOUD NATIVE COMPUTING

KUBERNETES

Überblick über die wichtigsten Kubernetes-Ressourcen

CLOUD NATIVE COMPUTING

Ausführbare Jobs/Workloads	Persistenz	Isolieren von Anwendungen mittels Namespaces	Exponieren von Anwendungen und Diensten
 Deployments	 Persistent Volume Claim	 Quotas und Limit Ranges	Primär Intra-Cluster  Services
 (Cron-)Jobs			
 Daemon Sets	 Persistent Volume	 Role Based Access Model	Primär Extra-Cluster  Ingress
 **Stateful Sets**			

STATEFUL SETS

Ausführbare Workloads

Stateful Sets dienen dem Ausbringen, Aktualisieren und Skalieren verteilter „Stateful Applications".

Solche Applikationen sind häufig Datenbanken; z.B. etcd (ein verteilter konsistenter Key-Value Store).

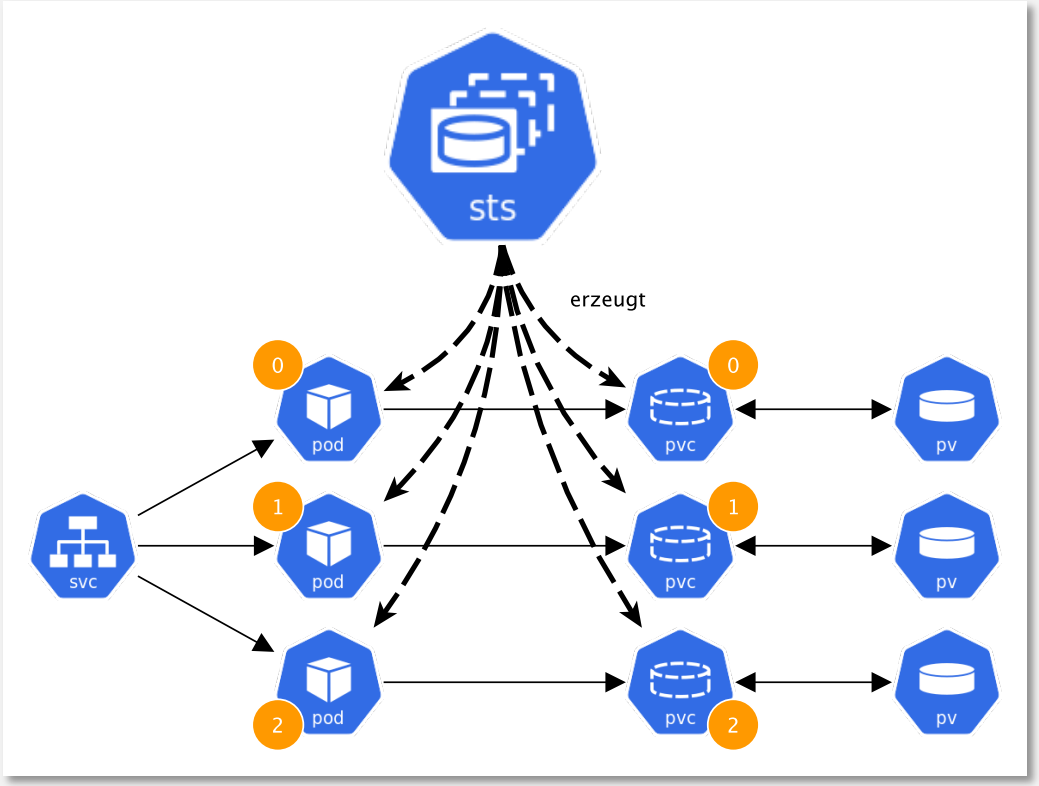
Dabei wird die Reihenfolge der Pod-Erzeugung und Eindeutigkeit der erzeugten Pods garantiert (Sticky Identity).

Obwohl Pods aus derselben Spec generiert werden, haben diese eine persistente Identität, die auch Reschedulings (z.B. im Rahmen von Aktualisierungen) überdauert.

Eine Besonderheit besteht bei Volume Mounts.

Stateful Sets können mittels einem ***Volume Claim Template*** pro Pod einen jeweils eigenen Volume Claim mit derselben Ordinalnummer wie der Pod erzeugen. Auf diese Weise, können Volumes bei einem Update des Pods wieder einem Pod mit derselben Ordinalnummer zugewiesen werden.

Auf diese Weise können auch Stateful Applikationen wie bspw. Datenbanken aktualisiert und skaliert werden.



CLOUD NATIVE COMPUTING

Anmerkung:
Stateful Sets sind nicht per se stateful. Einizig und allein der Controller ist so gestaltet, dass er auf den Betrieb von stateful Applikationen zugeschnitten ist.

Theoretisch ist es möglich stateless Komponenten mit einem Stateful Set zu betreiben.

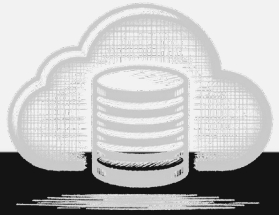
Quelle: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

STATEFUL SETS

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

StatefulSet



```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx-statefulset
  labels:
    app: nginx
spec:
  serviceName: "nginx"
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-storage
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: nginx-storage
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

Mit volumeClaimTemplates wird für jede Pod-Instanz ein eigenes Volume (persistenter Speicher) erstellt, die dieselber Ordinalnummer wie der Pod erhält.

CLOUD NATIVE COMPUTING

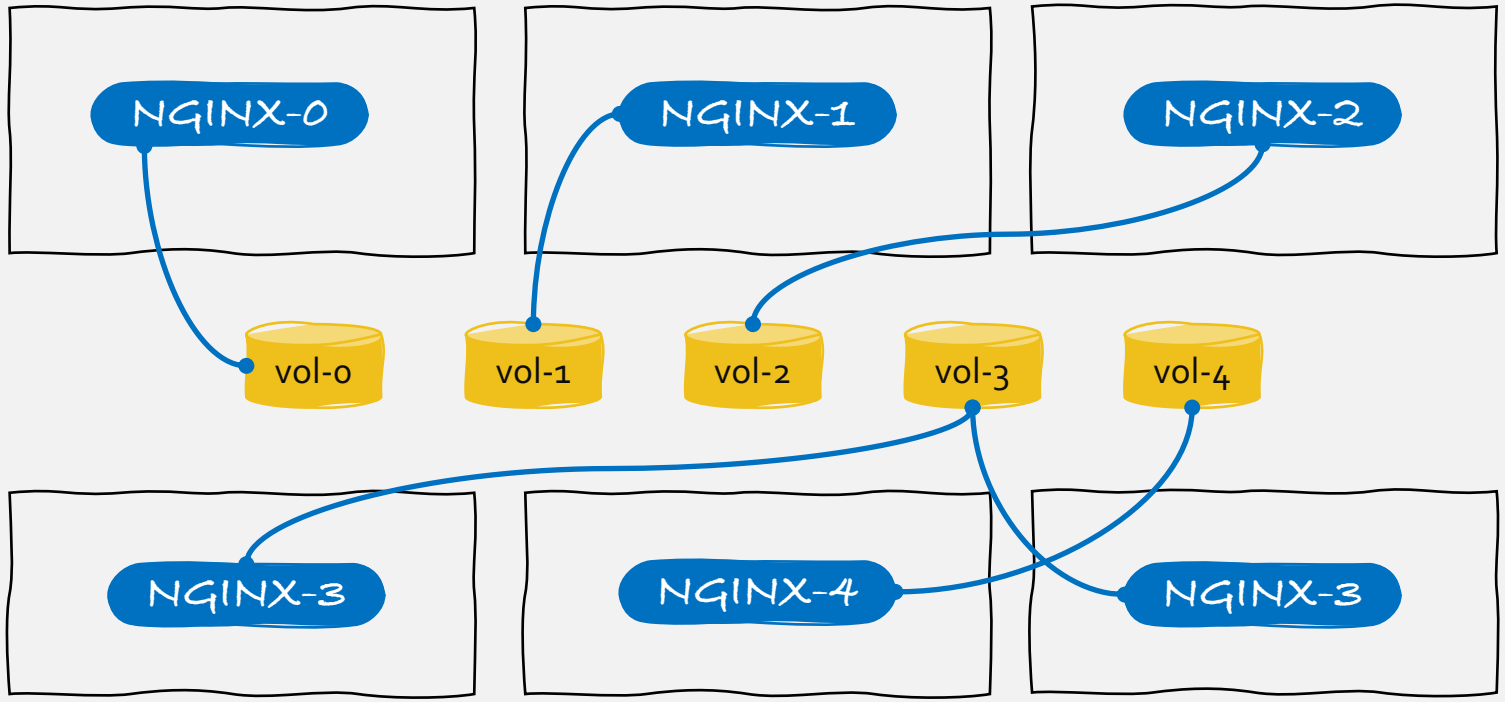
StatefulSets benötigen einige Angaben mehr als stateless Deployments.

*Der **serviceName** ist erforderlich, um jedem Pod einen eindeutigen, geordneten Hostnamen <serviceName>-<nr> geben zu können. Dies ermöglicht es den Pods, sich gegenseitig zu erkennen. Es ermöglicht ferner Volumes eindeutig mounten zu können.*

Diese Volumes können dann eindeutig Pods zugeordnet und gemounted werden.

SCHEDULING

Von Stateful Sets




```
> kubectl apply -f nginx-sts.yaml
> kubectl scale --replicas 5 -f nginx-sts.yaml
> kubectl scale --replicas 3 -f nginx-sts.yaml
> kubectl scale --replicas 4 -f nginx-sts.yaml
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx-statefulset
  labels:
    app: nginx
spec:
  serviceName: "nginx"
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
        volumeMounts:
        - name: vol
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: vol
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

AUSBLICK

Überblick über die wichtigsten Kubernetes-Ressourcen

CLOUD NATIVE COMPUTING

Ausführbare Jobs/Workloads	Exponieren von Anwendungen und Diensten	Persistenz	Isolieren von Anwendungen mittels Namespaces
 Deployments	Primär Intra-Cluster  **Services**	 **Persistent Volume Claim**	  **Quotas und Limit Ranges**
 (Cron-)Jobs			
 Daemon Sets	Primär Extra-Cluster  **Ingress**	 **Persistent Volume**	 **RBAC**  **Network Policies**
 Stateful Sets			

KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de