Deployment Management

- Erstellung eines Deployments
- Bearbeitung eines Deployments
- Deployments veröffentlichen mittels Loadbalancer
- Deployment Skalieren
- Erstellung eines Namespaces
- Deployments veröffentlichen mittels Ingress

Erstellung eines Deployments

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster

Erstellung

```
kubectl create deployment <deployment-name> -n <namespace> --image <image-name>
# Beispiel
kubectl create deployment nginx -n nginx --image nginx
```

Überprüfung

```
kubectl get deployment <deployment-name> -n <namespace>
# Beispiel
kubectl get deployment nginx -n nginx
```

Bearbeitung eines Deployments

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster
- Bereits erstelltes Deployment (How To)

Bearbeitung

Öffnen der aktuellen Konfiguration

Achtung: Wenn die änderungen gespeichert werden, werden diese direkt vom Cluster Übernommen.

```
kubectl edit deployment <deployment-name> -n <namespace>
# Beispiel
kubectl edit deployment nginx -n nginx
```

Die Bearbeitung

Passen Sie die Änderungen in dieser neu geöffneten Datei nach Ihrem wunsch an.

Beispielweise wurde der Port hinzugefügt.

Die Änderung

Fügen Sie die Änderung am Richtigen Ort hinzu, beispiel folgt unten.

```
ports:
- containerPort: 80
  protocol: TCP
```

Die komplette Datei

Wenn Sie die Datei schliessen werden die Änderungen übernommen. Wenn Sie jedoch eine Änderung an der YAML datei tätigen, welche nicht angepasst werden kann wird sich die Datei erneuert öffnen und anschliessend im Obersten Sektor den Fehler als Kommentar anzeigen.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "2"
  creationTimestamp: "2021-12-23T19:25:25Z"
  generation: 3
  labels:
    app: nginx
  name: nginx
  namespace: nginx
  resourceVersion: "255984"
  uid: c45eed25-d12f-4e32-b1fb-8965f372796b
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
```

```
    spec:
      containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  availableReplicas: 3
  conditions:
  - lastTransitionTime: "2021-12-23T19:25:25Z"
    lastUpdateTime: "2021-12-23T19:27:27Z"
    message: ReplicaSet "nginx-7848d4b86f" has successfully progressed.
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  - lastTransitionTime: "2021-12-23T19:28:47Z"
    lastUpdateTime: "2021-12-23T19:28:47Z"
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  observedGeneration: 1
  readyReplicas: 1
  replicas: 1
  updatedReplicas: 1
```

Überprüfung

```
kubectl get deployment <deployment-name> -n <namespace> -o <output-optionen>
# Beispiel
kubectl get deployment nginx -n nginx -o yaml
```

Deployments veröffentlichen mittels Loadbalancer

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster
- Bereits erstelltes Deployment (How To)

Veröffentlichung

Bei der veröffentlichung wird ein neuer Service vom Typ Loadbalancer erstellt, welcher die Port-Weiterleitung vom Host zum Container ermöglicht. Der Loadbalancer routed den Trafic vom allen nodes über einen Gewissen Port zu einem Host, auf welchem der veröffentlichte Container verfügbar ist.

```
kubectl expose deployment <deployment-name> --port=<Hostport> --target-port=<ContainerPort> --type=<Loadbalancer> -n <namespace>
# Beispiel
kubectl expose deployment nginx --port=8300 --target-port=80 --type=LoadBalancer -n nginx
```

Überprüfung

```
kubectl get service -n <namespace>
# Beispiel
kubectl get service -n nginx
```

Deployment Skalieren

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster
- Bereits erstelltes Deployment (How To)

Skalieren

Im falle einer Skalierung werden die Pods skaliert. Dies bedeutet das sämtliche Container in den Pods multipliziert werden.

```
kubectl scale --replicas=<anzahl> deployment <deployment-name> -n <namespace>
# Beispiel
kubectl scale --replicas=5 deployment nginx -n nginx
```

Überprüfung

```
kubectl get deployment <deploymentname> -n <namespace>
# Beispiel
kubectl get deployment nginx -n nginx
```

Erstellung eines Namespaces

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster

Erstellung

```
kubectl create namespace <namespace>
# Beispiel
kubectl create namespace nginx
```

Überprüfung

Wenn der Namespace existiert sollte er mit folgendem Command angezeigt werden können.

```
kubectl get namespaces
```

Deployments veröffentlichen mittels Ingress

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster
- Bereits erstelltes Deployment (How To)
- Bereits vorhandener Loadbalancer (How To)

Erstellung eines Ingress

Ein Ingress ist eine Art V-Host einfach für ein Kubernetes Cluster. Es muss nicht unbedingt ein Loadbalancer dazwischen sein, es kann auch eine Cluster IP oder ähnliches sein.

```
kubectl create ingress <ingress-name> --rule="<cname+directory>=<Loadbalancer:Port>" -n <namespace>
# Beispiel
kubectl create ingress nginx --rule="nginx.voser.local/=nginx:8300" -n nginx
```

Überprüfung

```
kubectl get ingress -n <namespace>
# Beispiel
kubectl get ingress -n nginx
```