Kubernetes (K8s)

Eine Presentation von

Florian Thom

Gliederung

- Ausgangssituation
- Kubernetes Überblick
- Objekte
  - Haupt-Objekte
  - Persistenz
  - Automatische Skalierung
  - Weitere
- Kubectl
- Quellen
- Anhang: Praxisbeispiel

Container:
Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run.
- https://cloud.google.com/containers/

Ausgangssituation

= Applikation innerhalb einfacher Container (via Docker, Singularity, LXC, …)

- Hauptvorteile durch Container
  - Plattformunabhängigkeit: “Built it once, run it anywhere”
  - Isolation der Applikation
  - Möglichkeit zur Skalierung
- Verbleibende Herausforderungen
  - Wie erfolgt praktische Skalierung?
  - Verteilung auf verschiedene Nodes?
  - Auswahl spezieller Nodes (CPU/RAM) ?
  - Umgang mit abgestürzten / nicht gebrauchstüchtigen Containern?
  - Softwareupdates?
- Lösung: Gebrauch einer Software, welche die Container managed
  - Benutzung eines Orchestrierungssystem für Container

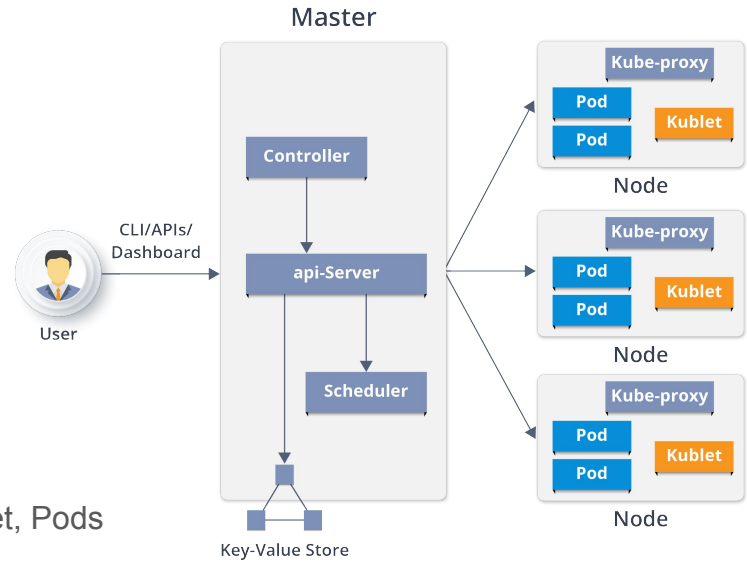
“Container Orchestration refers to the automated arrangement, coordination, and management of software containers”
- https://medium.com/onfido-tech/container-orchestration-with-kubernetes-an-overview-da1d39ff2f91

Kubernetes Überblick

“Kubernetes (k8s) is an open-source system for automating deployment, scaling, and management of containerized applications [...]” und bietet somit „ [...] production-grade container orchestration“ als Plattform

- https://kubernetes.io/

- Features:
  - Selfhealing
  - Unterstützte Rollouts and rollbacks
  - Horizontal-, vertical- and cluster-scaling
  - Automated scheduling / binpacking (Behällterproblem)
  - ...
- Steuerung des K8s-Clusters durch REST-Schnittstelle / API
  - offizieller CLI-Client = “kubectl”
- Cluster:
  - Master: API-Server, Controller, Scheduler, etcd-store
  - Slave: Container-Runtime(z.B. Docker), Kube-Proxy, Kubelet, Pods



Vereinfachte Architektur eines K8s-Clusters
https://www.quora.com/What-is-the-architecture-of-Kubernetes

Objekte

WordCloud der Objekt-Arten (kind's)

"Kubernetes Objects are persistent entities in the Kubernetes system […]"

- Allgemein auch "Ressource" genannt
- Jedes Objekt hat
  1. "spec" + Basisinformationen (definieren wir)
     - definiert "desired state" eines Objektes
     - wird in (.yaml) File (auch Manifest-File) festgelegt
  2. "status" (wird durch k8s definiert + geupdated)
     - Beschreibt aktuellen Status eines Objektes

- Manifest-File
  - Aufbau:
    1. "apiVersion" = API-Version der K8s - API
    2. **"kind" = Definiert die Art der Ressource**
    3. "Metadata" = Namen der Resource und ähnliches
    4. "spec": = Definiert "desired state" einer Ressource

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Beispiel Manifest-File eines Deployment-Objektes
https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

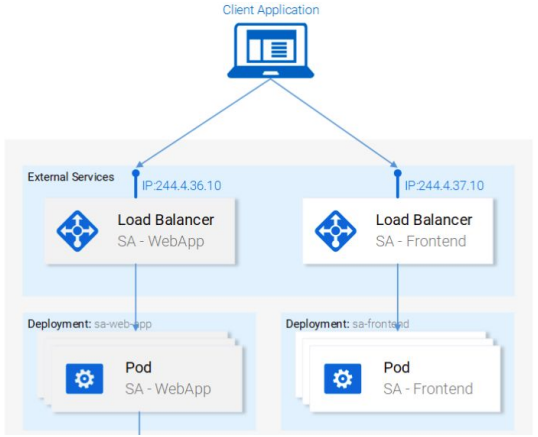
Objekte: Haupt-Objekte

Pod-Objekt

- = kleinste deployable Einheit innerhalb von Kubernetes
- = Abstraktion von Container
- Enthällt 1+ Container
- Container innerhalb eines Pods teilen:
  - Volume, ip, port-space

- Mehr als 1 Container pro Pod?
  - “Usually, you run only one container [...].”
  - Ausnahmen:
    - - “two containers need to share volumes”
    - - “or they communicate with each other using inter-process communication”
    - - “or are otherwise tightly coupled, then that’s made possible using Pods”

https://medium.freecodecamp.org/learn-kubernetes-in-under-3-hours-a-detailed-guide-to-orchestrating-containers-114ff420e882



Übersicht über Hauptkomponenten

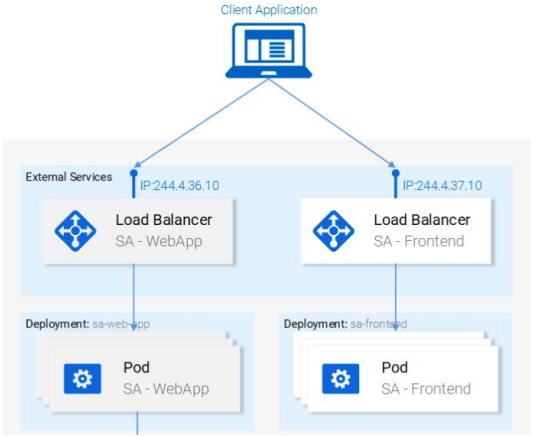
https://medium.freecodecamp.org/learn-kubernetes-in-under-3-hours-a-detailed-guide-to-orchestrating-containers-114ff420e882

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: httpd-simple-page
spec:
  containers:
    - image: flooth/apache-demo:0.1
      name: httpd-simple-page
      ports:
        - containerPort: 80 # (optional)
```

Objekte: Haupt-Objekte

Deployment-Objekt

- = 1 Wrapper über X Pods
- Ziel:
  - Aufrechterhaltung einer definierten Anzahl an gebrauchsfähigen Pods (ReplicaSet)



Übersicht über Hauptkomponenten
https://medium.freecodecamp.org/learn-kubernetes-in-under-3-hours-a-detailed-guide-to-orchestrating-containers-114ff420e882

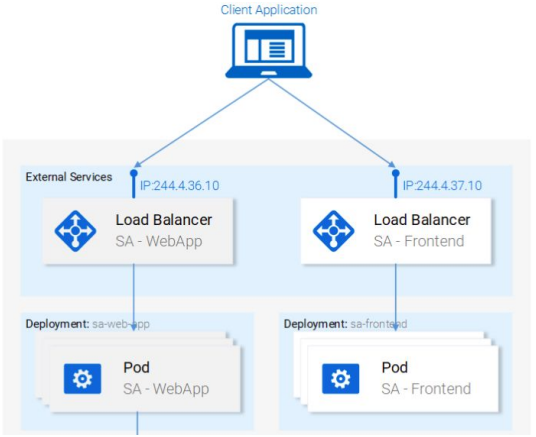
  - Erleichtert Updates: Zero-downtime-update:
    - bei Update werden nacheinander die alten Replicas (des Pods) eingestellt und gleichzeitig die neuen Replicas (mit neuem image/state) aufgesetzt
    - Geringe Abweichung von der “Desired number of Replicas” über Update
      - maximal 25% der “Desired number of Replicas” nicht verfügbar

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: httpd-simple-page
spec:
  replicas: 3
  minReadySeconds: 15
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: httpd-simple-page
    spec:
      containers:
        - image: flooth/apache-demo:0.1
          imagePullPolicy: Always
          name: httpd-simple-page
          ports:
            - containerPort: 80
```

Objekte: Haupt-Objekte

Service-Objekt

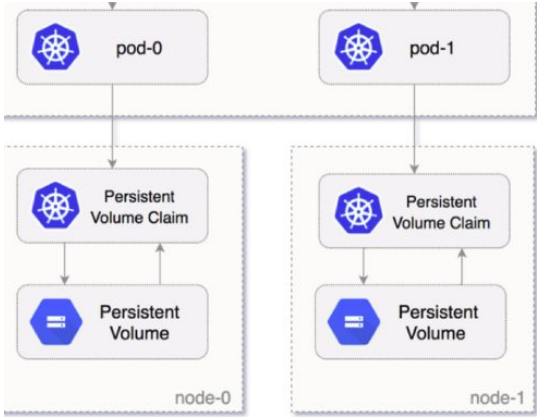
- Ausgangssituation:
  - “each Pod has a unique IP address, those IPs are not exposed outside the cluster”
    - https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/
- = Entrypoint zu einer Menge von Pods innerhalb des Clusters
  - Gewünschte Pods werden per “Label” in Manifest-file getaggt
  - Übergabe des “Label”-Name zum Service via Selector-Field in Service-Manifest-file
- Service-Types:
  - ClusterIP (default): Service wird cluster-intern exposed (mit fester IP)
  - NodePort: Service wird außerhalb des cluster ansprechbar (mit fester IP)
  - Loadbalancer: erstellt Loadbalancer + externe IP + weist beides einem Service zu
  - ...



Übersicht über Hauptkomponenten
https://medium.freecodecamp.org/learn-kubernetes-in-under-3-hours-a-detailed-guide-to-orchestrating-containers-114ff420e882

```yaml
apiVersion: v1
kind: Service
metadata:
  name: httpd-dev-service
spec:
  type: LoadBalancer
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: httpd-simple-page
```

Objekte: Persistenz



https://www.weave.works/blog/kubernetes-faq-configure-storage-for-bare-metal-cluster

PersistentVolume (PV)

- = Abstraktion vom physischen Speicher und Protokoll (NFS,...)
- Low-level → man muss konkreten Speicher kennen
- 1 PV = 1 Speichergröße
- Erstellt vom Operator / Admin

PersistentVolumeClaim (PVC)

- = Verbindung zwischen Pods und PV
- Erstellt vom Developer
- Es wird auf kein spezielles PV verwiesen
    - Nach Definition der “Claims” meiner App (z.B. 100GB) wählt Kubernetes automatisch selbst ein zufälliges PV (welches die Forderungen erfüllt)
    - Lose Kopplung zwischen PV und PVC
    - Forderung muss MINDESTENS erfüllt sein -> keine Aussage darüber wie viel insgesamt gebraucht wird (vom konkreten Volume)

Objekte: Automatische Skalierung

Pods-Layer-Autoscaler

- = skalieren der verfügbaren Ressourcen für die Container
- **Horizontal Pod Autoscaler (HPA)**
  - Skaliert die Anzahl der Pod-Replikas
  - Basiert auf Benutzung von CPU / RAM / RequestCounter / Custom eines Nodes...
- **Vertical Pod Autoscaler (VPA)**
  - Weist Pods dynamisch mehr/weniger CPU/RAM zu
  - Veränderung der Ressourcen → Pods müssen neu-gestartet werden
  - Intern: “Recommender”-Komponente überblickt Vergangenheit und Gegenwart und schlägt neuen Ressourcen-Wert vor
- HPA & VPA nicht kompatibel → können nicht auf gleichen pods arbeiten

```yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: resize-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: image-resizer
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: memory
      targetAverageUtilization: 60
```

Manifest-File für Beispiel HPA

```yaml
apiVersion: poc.autoscaling.k8s.io/v1alpha1
kind: VerticalPodAutoscaler
metadata:
  name: dyn-class-gen-vpa
spec:
  selector:
    matchLabels:
      app: dyn-class-gen
  updatePolicy:
    updateMode: "Auto"
```

Manifest-File für Beispiel VPA

Cluster Autoscaler (CA)

- Skaliert die Anzahl der Nodes innerhalb des Clusters (hoch / runter)
- Trigger:
  - Pods für Node-Pool vorhanden, aber nicht gescheduled, da keine freien Ressourcen vorhanden
- Vorgehen:
  - Anfragen und Integrieren neuer Nodes in Node-Pool (z.B. von AWS, ...)

```
gcloud container clusters create example-cluster \
--zone us-central1-a \
--node-locations us-central1-a,us-central1-b,us-central1-f \
--num-nodes 2 --enable-autoscaling --min-nodes 1 --max-nodes 4
```

Provider-spezifischer CA

Objekte: Weitere

Nicht angesprochene Objekte:

- Ingress
- Secrets
- Namespaces
- Jobs
- Configmaps
- ...

WordCloud der Objekt-Arten (kind's)

Kubectl

> “Kubectl is a command line interface for running commands against Kubernetes clusters.”
>
> - https://kubernetes.io/docs/reference/kubectl/overview/

- = Client für Requests zum API-Server des Cluster
- Beispiel-Anfragen:
  - Managing Kubernetes Objects Using Imperative Commands
    - Kubectl run pod_name --image=image_name → e.g. kubectl run ghost --image=ghost
  - Imperative Management of Kubernetes Objects Using Configuration Files
    - kubectl get ressource → e.g. kubectl get pods
    - kubectl create -f manifest-File → e.g. kubectl create -f deployment.yaml
    - Kubectl describe ressource ressource-name → e.g. kubectl describe deployment http-server-one
    - kubectl delete ressource ressource-name → e.g. kubectl delete pods http-server-one
  - Declarative object configuration
    - kubectl apply -f deployment.yaml → kubectl apply -f configs/

Quellen

- https://portworx.com/tutorial-kubernetes-persistent-volumes/
- https://www.youtube.com/watch?v=OulmwTYTaul
- https://cloud.google.com/containers/
- https://medium.com/onfido-tech/container-orchestration-with-kubernetes-an-overview-da1d39ff2f91
- https://kubernetes.io/
- https://medium.freecodecamp.org/learn-kubernetes-in-under-3-hours-a-detailed-guide-to-orchestrating-containers-114ff420e882
- https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/
- https://kubernetes.io/docs/reference/kubectl/overview/
- https://portworx.com/tutorial-kubernetes-persistent-volumes/
- https://portworx.com/basic-guide-kubernetes-storage/

Bilder:

- https://raw.githubusercontent.com/kubernetes/kubernetes/master/logo/logo_with_border.png
- https://www.quora.com/What-is-the-architecture-of-Kubernetes
- https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/
- https://medium.freecodecamp.org/learn-kubernetes-in-under-3-hours-a-detailed-guide-to-orchestrating-containers-114ff420e882
- https://www.weave.works/blog/kubernetes-faq-configure-storage-for-bare-metal-cluster
- https://unofficial-kubernetes.readthedocs.io/en/latest/concepts/storage/persistent-volumes/
-

Vielen Dank für die Aufmerksamkeit!

Anhang

Praxisbeispiel:

1. Create content
2. Create dockerfile
3. Test dockerfile
4. Docker build image: z.B. docker build -t hello-word .
5. Docker push image to (local / remote) registry
6. Docker pull flooth/apache-demo:0.1
7. Schreiben eines/mehrer Manifest-files (deployment and service)
8. Ressourcen erstellen / Manifest-File anwenden

```yaml
# important, this is not a random value, its like docker-compose version
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: httpd-simple-page
spec: # Specification of the deployment
  replicas: 3
  minReadySeconds: 15
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template: # aka how should 1 pod look like?
    metadata:
      labels:
        app: httpd-simple-page
    spec: # Specification of the template
      containers:
        - image: flooth/apache-demo:0.1
          imagePullPolicy: Always
          name: httpd-simple-page
          ports:
            - containerPort: 80

---

apiVersion: v1
kind: Service
metadata:
  name: httpd-dev-single-page
spec:
  type: LoadBalancer
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: httpd-simple-page
```

Manifest-File: Deployment- und Service-Objekt eines http-Beispiels
(wird oft in unterschiedliche Files geteilt)

Anhang

Kubernetes-Cluster-Addons

- K8s Dashboard
  - Grafische Benutzeroberfläche zum schnellen Monitoring des Cluster
- Helm
  - Packet-Manager, vergleichbar mit apt / brew / …
  - Verwaltet viele hilfreiche, schon in Manifest-Files verpackte, Software (-stacks)
    - Beispiel: ELK, Prometheus, Jupyter Hub
- Heapster
  - Zum Überwachen der CPU-Nutzung im Allgemeinen und als Addon im Dashboard
- Freshpod [Development]
  - Startet automatisch Container neu, wenn das dazugehörige Image geupdatet wird
- ...