Tobias Koch

Kubernetes

Technische Übersicht und Best Practices

Das Titelbild dieses Dokuments stammt von

Growtika

unsplash.com/photos/GSiEeoHcNTQ

Veröffentlicht unter der Unsplash-Lizenz.

Eine Namensnennung ist nicht erforderlich, wird hier jedoch mit Dank gemacht.

Inhaltsverzeichnis

1 Grundlagen

1.1 Pod

Die kleinste und einfachste Kubernetes-Einheit, die eine oder mehrere Container beinhaltet, die zusammen auf einer Node ausgeführt werden.

Befehl	Beschreibung
`kubectl get pods`	Alle Pods im Standard-Namespace auflisten
`kubectl describe pod <pod-name>`	Details zu einem bestimmten Pod anzeigen
`kubectl delete pod <pod-name>`	Einen Pod löschen
`kubectl logs <pod-name>`	Logs eines Pods anzeigen
`kubectl exec -it <pod-name> – <command>`	Befehle in einem Pod ausführen

Beispielkonfiguration: Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: beispiel-pod
spec:
  containers:
    - name: beispiel-container
      image: beispiel:latest
      ports:
        - containerPort: 80
```

- **image** legt fest, welches Image verwendet wird
- Nach dem Doppelpunkt steht die Versionsbezeichnung, oder „latest“ für die neueste Version
- In der Produktion sollte die Version spezifiziert werden
- **ports** gibt an, auf welchen Ports der Container Anfragen entgegennimmt
- Sie dienen als Referenz und machen den Port *nicht* automatisch von außen erreichbar

Nützliche Links und Ressourcen

Dokumentation: https://kubernetes.io/docs/concepts/workloads/pods/

1.2 Deployment

Verwaltet und skaliert eine Gruppe von Pods und stellt sicher, dass eine bestimmte Anzahl von Pods immer läuft.

Befehl	Beschreibung
`kubectl get deployments`	Alle Deployments auflisten
`kubectl describe deployment <deployment-name>`	Details anzeigen
`kubectl delete deployment <deployment-name>`	Deployment löschen
`kubectl rollout restart deployment <deployment-name>`	Deployment neu starten
`kubectl scale deployment <deployment-name> --replicas=<number>`	Anzahl der Pods ändern
`kubectl set image deployment <deployment-name> <container-name>=<new-image>`	Image aktualisieren

Beispielkonfiguration: Deployment

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

- **replicas** legt die Anzahl gleichzeitig laufender Pods fest
- **selector** ordnet Pods über Label-Matching dem Deployment zu
- **template** beschreibt den Pod-Aufbau
- In `template.spec` Container, Images und Ports definieren
- Änderungen am Deployment sorgen automatisch für einen geordneten Austausch der Pods

Nützliche Links und Ressourcen

Dokumentation: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

1.3 Rollouts

Ein Rollout ist der Prozess, mit dem Kubernetes eine neue Version eines Deployments ausrollt. Dabei ersetzt Kubernetes alte Pods schrittweise durch neue, um Ausfälle zu vermeiden.

Befehl	Beschreibung
`kubectl rollout status deployment <name>`	Fortschritt eines Rollouts überwachen
`kubectl rollout restart deployment <name>`	Deployment neu starten
`kubectl rollout undo deployment <name>`	Letzten Rollout rückgängig machen
`kubectl rollout history deployment <name>`	Rollout-Historie anzeigen

- Rollouts erfolgen automatisch bei jeder Änderung am Deployment
- Rollbacks sind möglich, solange frühere Revisionen gespeichert sind
- Mit `restart` kann ein Rollout auch ohne Änderung ausgelöst werden

Nützliche Links und Ressourcen

Dokumentation:
https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#updating-a-deployment

1.4 Services

Services abstrahieren die Kommunikation mit Pods. Sie stellen eine dauerhafte IP-Adresse und einen DNS-Namen bereit, unter dem eine Gruppe von Pods erreicht werden kann.

Befehl	Beschreibung
`kubectl get services`	Services auflisten
`kubectl describe service <service-name>`	Details anzeigen
`kubectl delete service <service-name>`	Service löschen
`kubectl expose <resource> –port=<port>`	Service erstellen

1.4.1 Service-Typen

Der Service Typ legt fest, wie der Netzwerkzugriff erfolgt:

Typ	Beschreibung
`ClusterIP`	Standard-Typ. Verbindet Clients innerhalb des Clusters. Von außen nicht erreichbar.
`NodePort`	Öffnet einen festen Port auf jeder Node, leitet Traffic an den Service weiter.
`LoadBalancer`	Nutzt einen externen Load Balancer des Cloud-Providers.
`ExternalName`	Leitet den DNS-Namen auf eine externe Adresse um. Kein echter Proxy.

Beispielkonfiguration: Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: beispiel-service
spec:
  selector:
    app: beispiel-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  type: ClusterIP
```

1.4.2 Headless Services

Eine spezielle Art von Service, die keine Cluster-IP bereitstellen und somit keine Lastverteilung durchführen. Stattdessen ermöglichen sie clientseitiges Load Balancing oder direkten DNS-Zugriff auf die einzelnen Pods. Typische Einsatzszenarien sind:

- StatefulSets mit stabiler Pod-Namensauflösung (siehe 5.2)
- Datenbanken, die Peer-Discovery benötigen
- Monitoring-Tools mit direktem Zugriff auf Pods

Beispielkonfiguration: Headless Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: beispiel-headless-service
spec:
  clusterIP: None
  selector:
    app: beispiel-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Nützliche Links und Ressourcen

Dokumentation: https://kubernetes.io/docs/concepts/services-networking/service/

1.5 Nodes

Nodes sind physische oder virtuelle Maschinen in einem Kubernetes-Cluster. Sie stellen die Rechenressourcen (CPU, RAM, Netzwerk) zur Verfügung und führen die Pods aus.

Befehl	Beschreibung
`kubectl get nodes`	Alle Nodes im Cluster auflisten
`kubectl describe node <node-name>`	Details zu einer bestimmten Node anzeigen
`kubectl cordon <node-name>`	Node für neue Pods sperren
`kubectl uncordon <node-name>`	Node wieder freigeben
`kubectl drain <node-name>`	Node evakuieren und alle Pods darauf verschieben
`kubectl delete node <node-name>`	Node aus dem Cluster entfernen

1.5.1 Node-Typen

Typ	Beschreibung
Master Node	Verantwortlich für die Cluster-Steuerung und das Scheduling von Pods
Worker Node	Führt Pods aus, enthält Kubelet, Kube-Proxy und eine Container Runtime
Etcd Node	Speichert den Cluster-Zustand. Läuft meist auf Master-Nodes, kann auch dediziert sein

1.5.2 Node-Komponenten

Komponente	Beschreibung
Kubelet	Sorgt dafür, dass Pods auf der Node wie definiert laufen
Kube-Proxy	Verantwortlich für die Weiterleitung von Traffic und Netzwerkrichtlinien
Container Runtime	Führt Container aus (z. B. Docker, containerd, CRI-O)
API-Server	Gateway zur Kubernetes-API und Frontend für die Control Plane
Scheduler	Plant neue Pods auf passenden Nodes mit genügend Ressourcen
Controller-Manager	Führt laufende Prozesse aus, um geforderten Zustand stabil zu halten
Cloud-Controller	Kommuniziert mit dem Cloud-Provider

Weitere nützliche Befehle

Befehl	Beschreibung
`kubectl top nodes`	CPU- und RAM-Auslastung der Nodes anzeigen
`kubectl label node <node-name> <label-key>=<label-value>`	Label hinzufügen
`kubectl taint nodes <node-name> <key>=<value>:<taint-effect>`	Node für bestimmte Pods unplanbar machen
`kubectl edit node <node-name>`	Node bearbeiten
`kubectl get node <node-name> -o yaml`	Konfiguration anzeigen
`kubectl patch node <node-name> -p <patch-data>`	Patch anwenden
`kubectl get events –field-selector involvedObject.kind=Node`	Ereignisse der Node anzeigen
`kubectl annotate node <node-name> <annotation-key>=<annotation-value>`	Annotation hinzufügen

Nützliche Links und Ressourcen

Dokumentation: https://kubernetes.io/docs/concepts/architecture/nodes/

1.6 Namespaces

Ermöglichen die Trennung von Ressourcen innerhalb eines Kubernetes-Clusters, um verschiedene Umgebungen oder Teams zu unterstützen.

Befehl	Beschreibung
`kubectl get namespaces`	Alle Namespaces auflisten
`kubectl describe namespace <namespace-name>`	Details zu einem Namespace anzeigen
`kubectl create namespace <namespace-name>`	Neuen Namespace erstellen
`kubectl delete namespace <namespace-name>`	Einen Namespace löschen
`kubectl get pods –namespace=<namespace-name>`	In einem bestimmten Namespace ausführen
`kubectl config set-context –current –namespace=<namespace-name>`	Namespace als Standard setzen
`kubectl config view –minify | grep namespace:`	Aktuellen Namespace anzeigen
`kubectl api-resources –namespaced=true`	Ressourcen in namespaces
`kubectl api-resources –namespaced=false`	Ressourcen außerhalb von namespaces

1.6.1 Best Practices bei Namespaces

- Entwicklungs-, Test- und Produktionsumgebungen in separaten Namespaces trennen
- `ResourceQuotas` und `LimitRanges` definieren, um Ressourcenverbrauch zu kontrollieren
- Zugriff über `RoleBindings` auf Namespace-Ebene granular steuern
- Namespace-Anzahl überschaubar halten, um Komplexität zu vermeiden
- Systemreservierte Namespaces wie `kube-system` und `default` unangetastet lassen
- Klare, sprechende Namen vergeben (`frontend-staging` statt `ns-2`)

Nützliche Links und Ressourcen

Dokumentation:
https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

1.7 Labels und Annotations

Labels und Annotations sind Schlüssel-Wert-Paare, die zur Markierung und Beschreibung von Kubernetes-Ressourcen verwendet werden. Labels werden zur Identifizierung genutzt, während Annotations zusätzliche Informationen tragen. Labels werden zur Selektion und Gruppierung von Ressourcen verwendet, während Annotations zur Speicherung von nicht-selektiven Metadaten dienen.

Befehl	Beschreibung
`kubectl get pods –show-labels`	Alle Pods mit ihren Labels auflisten
`kubectl label pod <pod-name> <key>=<value>`	Ein Label zu einem Pod hinzufügen
`kubectl label pod <pod-name> <key>-`	Ein Label von einem Pod entfernen
`kubectl annotate pod <pod-name> <key>=<value>`	Eine Annotation zu einem Pod hinzufügen
`kubectl annotate pod <pod-name> <key>-`	Eine Annotation von einem Pod entfernen
`kubectl get pods -l <key>=<value>`	Pods mit einem bestimmten Label auswählen
`kubectl get pods -l 'env in (production, qa)'`	Pods mit einem der Labels auswählen
`kubectl get pods -l 'tier notin (frontend, backend)'`	Pods auswählen, die diese Labels nicht haben

1.7.1 Best Practices für Labels und Annotations

- Syntax korrekt verwenden: Key im Format '<prefix>/<name>', Name max. 63 Zeichen, prefix optional (DNS-Subdomain $\leq$ 253 Zeichen)
- Wenn kein Präfix vorhanden ist kann davon ausgegangen werden dass das Label oder die Annotation privat für Cluster und Nutzer ist
- Präfix verwenden, wenn Labels toolschnittstellenübergreifend oder von Drittanbietertools genutzt werden
- Empfohlene Kubernetes-Labels verwenden
- Labels für das Versionsmanagement verwenden
- Standardisierte Namenskonventionen verwenden
- Nachträgliche Änderungen an Labels vermeiden, da dies zu fehlerhaften Selektoren und unvorhergesehenen Rescheduling-Effekten führen kann
- Labels und Annotationen nicht für sensitive Daten wie Passwörter oder API-Keys verwenden
- Labels systematisch und flächendeckend einsetzen, um granulare Filterung, Auditierung und Management zu ermöglichen
- Labels zur Ressourcen-Selektion, Annotations für ergänzende oder toolbezogene Metadaten nutzen
- Annotations verwenden, um nicht-selektive Metadaten wie Build-IDs, Commit-Hashes, Verantwortliche oder Tool-Informationen abzulegen
- Labels direkt im Pod-Template (z.B. in Deployments) definieren
- Labeling und Annotationen im CI/CD automatisieren, um Konsistenz zu gewährleisten und menschliche Fehler zu vermeiden
- Labels zur Kostenüberwachung verwenden, um unnötige Ausgaben zu vermeiden
- Labels für Debugging und Fehlersuche verwenden, indem man die Labels des Selektors ändert, sodass sie nicht mehr mit den Pods übereinstimmen, die für Probleme verantwortlich sein könnten und sie so aus dem aktiven Cluster herauszulösen

Nützliche Links und Ressourcen

Empfohlene Kubernetes Labels:
https://kubernetes.io/docs/concepts/overview/working-with-objects/common-labels/
https://komodor.com/blog/best-practices-guide-for-kubernetes-labels-and-annotations
https://cast.ai/blog/kubernetes-labels-expert-guide-with-10-best-practices/
https://www.redhat.com/en/blog/kubernetes-labels-best-practices

1.8 Selektoren

Selektoren sind ein essentielles Konzept in Kubernetes, das verwendet wird, um Ressourcen basierend auf bestimmten Kriterien zu filtern oder auszuwählen. Sie werden häufig in verschiedenen Kubernetes-Objekten wie Pods, ReplicaSets und Services verwendet, um eine gezielte Auswahl und Zuweisung zu ermöglichen.

1.8.1 Label-Selektoren

Verwenden Labels, um Ressourcen zu filtern.

Gleichheitsbasierte Selektoren

Filtern Ressourcen basierend auf der genauen Übereinstimmung von Label-Werten.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  labels:
    app: web
spec:
  containers:
    - name: nginx
      image: nginx

---
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  selector:
    app: web
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

In diesem Fall wird der Selektor genutzt um die app ‘web‘ auszuwählen.
Sie können auch innerhalb der Kommandozeile verwendet werden. Es stehen drei Operatioren zur Auswahl

- = oder == : Prüft Übereinstimmung mit Label (die Verwendung von „=“ wird empfohlen)
- != : Prüft fehlende Übereinstimmung mit Label

CLI-Beispiel:

```
kubectl get pods -l env=production
```

Mengenbasierte Selektoren

Filtern Ressourcen basierend auf der Zugehörigkeit von Label-Werten zu einer bestimmten Menge.

YAML	Bedeutung	CLI-Kurzform	Beispiel
`In`	Schlüssel existiert und hat einen der angegebenen Werte	`key in (a,b)`	`app in (web,api)`
`NotIn`	Schlüssel existiert und hat keinen der angegebenen Werte	`key notin (a,b)`	`env notin (prod,qa)`
`Exists`	Schlüssel existiert, Wert egal	`key`	`tier`
`DoesNotExist`	Schlüssel existiert nicht	`!key`	`!partition`

In YAML-Dateien werden die Operatoren groß geschrieben. Im CLI gelten Kurzformen.

Beispielkonfiguration: Mengenbasierter Selektor

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: multi-operator-replicaset
spec:
  replicas: 3
  selector:
    matchExpressions:
      - key: app
        operator: In
        values:
          - frontend
          - backend
      - key: environment
        operator: NotIn
        values:
          - staging
      - key: release
        operator: Exists
  template:
    metadata:
      labels:
        app: frontend
        environment: production
        release: stable
    spec:
      containers:
        - name: nginx
          image: nginx:1.25
```

Kombinierte `matchExpressions`. Pods werden nur erstellt, wenn alle Bedingungen erfüllt sind:

- `app` ist `frontend` oder `backend`
- `environment` ist nicht `staging`
- `release` ist gesetzt (Wert beliebig)

CLI-Beispiel:

```
kubectl get pods -l 'app in (web,api),env notin (staging),release'
```

1.8.2 Feld-Selektoren

Verwenden Felder von Ressourcen, um sie zu filtern. Zum Beispiel können Pods basierend auf ihrem Status oder Namen ausgewählt werden.

```
kubectl get pods --field-selector=status.phase=Running
```

Dieser Befehl listet alle Pods auf, die sich im Status „Running“ befinden.
Es können „=“ und „==“ verwendet werden (empfohlen wird „=“), um zu prüfen, ob die Felder übereinstimmen, und „!=“, um fehlende Übereinstimmungen festzustellen.
Sie können miteinander verkettet werden:

```
kubectl get pods --field-selector=status.phase!=Running,spec.restartPolicy=Always
```

Zudem können mehrere Ressourcentypen gleichzeitig ausgewählt werden:

```
kubectl get statefulsets,services --all-namespaces --field-selector metadata.namespace!=default
```

Durch Feld-Selektoren unterstützte Felder

Die folgende Tabelle listet Ressourcenarten und ihre jeweiligen Felder auf, die im Rahmen von Feld-Selektoren (‘–field-selector‘) verwendet werden können.

Typ	**Feld**

Pod	`spec.nodeName`
	`spec.restartPolicy`
	`spec.schedulerName`
	`spec.serviceAccountName`
	`spec.hostNetwork`
	`status.phase`
	`status.podIP`
	`status.nominatedNodeName`

Event	`involvedObject.kind`
	`involvedObject.namespace`
	`involvedObject.name`
	`involvedObject.uid`
	`involvedObject.apiVersion`
	`involvedObject.resourceVersion`
	`involvedObject.fieldPath`
	`reason`
	`reportingComponent`
	`source`
	`type`

Secret	`type`
Namespace	`status.phase`
ReplicaSet	`status.replicas`
ReplicationController	`status.replicas`
Job	`status.successful`
Node	`spec.unschedulable`
CertificateSigningRequest	`spec.signerName`

Alle Custom Resources (9.2) unterstützen die `metadata.name` und `metadata.namespace` Felder.

1.8.3 Verwendung von Selektoren in Kubernetes-Objekten

Selektoren werden in verschiedenen Kubernetes-Objekten verwendet, um eine gezielte Auswahl und Verwaltung zu ermöglichen:

Deployments

Deployments verwenden Selektoren, um die Pods zu identifizieren, die sie verwalten sollen.

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

Services

Services verwenden Selektoren, um die Pods zu identifizieren, denen sie den Netzwerkverkehr weiterleiten sollen.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  selector:
    app: web
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

ReplicaSets

ReplicaSets verwenden Selektoren, um die Pods zu identifizieren, die sie verwalten sollen.

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: example-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: nginx
          image: nginx
```

1.8.4 Best Practices für die Verwendung von Selektoren

Im Folgenden sind bewährte Vorgehensweisen für den Einsatz von Selektoren in Kubernetes aufgeführt:

- **Eindeutige Labels**: Labels sollten eindeutig gewählt werden, um sicherzustellen, dass Selektoren nur die gewünschten Ressourcen erfassen.
- **Konsistente Benennung**: Eine konsistente Benennungskonvention für Labels und Selektoren erleichtert die Wartung und das Verständnis.
- **Dokumentation**: Die verwendeten Labels und Selektoren sollten nachvollziehbar dokumentiert werden.
- **Vermeidung von Konflikten**: Es ist darauf zu achten, dass keine Konflikte zwischen Selektoren und Labels entstehen, die unerwartetes Verhalten verursachen könnten.

Nützliche Links und Ressourcen

Dokumentation:
https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
https://kubernetes.io/docs/concepts/overview/working-with-objects/field-selectors/
Kubernetes Quellcode:
https://github.com/kubernetes/apimachinery/blob/master/pkg/apis/meta/v1/types.go
Hands-on Guide:
https://devtron.ai/blog/kubernetes-labels-and-selectors-a-definitive-guide-with-hands-on/

2 Konfiguration und Geheimnisse

2.1 ConfigMaps

ConfigMaps werden verwendet, um Konfigurationsdaten von Anwendungen zu speichern.

Befehl	Beschreibung
`kubectl get configmaps`	Alle ConfigMaps auflisten
`kubectl describe configmap <configmap-name>`	Details zu einer ConfigMap anzeigen
`kubectl get configmap <configmap-name>`	ConfigMap anzeigen
`kubectl create configmap <configmap-name>`	Neue ConfigMap erstellen
`kubectl delete configmap <configmap-name>`	Eine ConfigMap löschen
`kubectl edit configmap <configmap-name>`	Eine ConfigMap bearbeiten

2.1.1 Beispiele für die Erstellung von ConfigMaps

Erstellen einer ConfigMap aus einer Datei

Eine ConfigMap kann aus einer oder mehreren Dateien erstellt werden.

```
kubectl create configmap my-config --from-file=config-file.conf
```

Erstellen einer ConfigMap aus Literalwerten

```
kubectl create configmap my-config --from-literal=key1=value1 --from-literal=key2=value2
```

2.1.2 Verwendung von ConfigMaps in Pods

ConfigMaps können in Pods als Umgebungsvariablen oder als Dateien im Dateisystem verwendet werden.

Verwendung als Umgebungsvariable

Verwendung einer ConfigMap in einem Pod als Umgebungsvariable:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: my-image
      env:
        - name: CONFIG_KEY
          valueFrom:
            configMapKeyRef:
              name: my-config
              key: key1
```

Verwendung als Datei im Dateisystem

Verwendung einer ConfigMap in einem Pod als Datei im Dateisystem:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: my-image
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: my-config
```

2.1.3 Anzeigen und Bearbeiten von ConfigMaps

ConfigMap anzeigen:

```
kubectl get configmap my-config -o yaml
```

ConfigMap bearbeiten:

```
kubectl edit configmap my-config
```

2.2 Secrets

Secrets werden verwendet, um sensible Informationen wie Passwörter und Tokens zu speichern.

Befehl	Beschreibung
`kubectl get secrets`	Alle Secrets auflisten
`kubectl describe secret <secret-name>`	Details zu einem Secret anzeigen
`kubectl create secret <type> <secret-name>`	Neues Secret erstellen
`kubectl delete secret <secret-name>`	Ein Secret löschen

2.2.1 Beispiele für die Erstellung von Secrets

Erstellen eines generischen Secrets aus Literalwerten

```
kubectl create secret generic my-secret --from-literal=username=admin --from-literal=password=secret
```

Erstellen eines Secrets aus einer Datei

```
kubectl create secret generic my-secret --from-file=./secret-file.txt
```

Erstellen eines Docker-Registry Secrets

```
kubectl create secret docker-registry my-registry-secret --docker-username=<username>
↪  --docker-password=<password> --docker-email=<email> --docker-server=<server>
```

Erstellen eines TLS Secrets

Ein TLS Secret kann verwendet werden, um TLS-Zertifikate und -Schlüssel zu speichern:

```
kubectl create secret tls my-tls-secret --cert=path/to/cert/file --key=path/to/key/file
```

2.2.2 Verwendung von Secrets in Pods

Secrets können in Pods als Umgebungsvariablen oder als Dateien im Dateisystem verwendet werden.

Verwendung als Umgebungsvariable

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: my-image
      env:
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: my-secret
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: my-secret
              key: password
```

Verwendung als Datei im Dateisystem

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: my-image
      volumeMounts:
        - name: secret-volume
          mountPath: /etc/secret
  volumes:
    - name: secret-volume
      secret:
        secretName: my-secret
```

2.2.3 Transport Layer Security (TLS)

TLS ist ein kryptografisches Protokoll, das für die Sicherung der Kommunikation über ein Computernetzwerk, insbesondere das Internet, verwendet wird. Es stellt sicher, dass die Daten während der Übertragung zwischen Client und Server verschlüsselt und vor unbefugtem Zugriff geschützt sind.

TLS-Zertifikate

TLS-Zertifikate sind digitale Dateien, die die Identität einer Website oder eines Servers bestätigen und einen öffentlichen Schlüssel enthalten. Sie werden von vertrauenswürdigen Zertifizierungsstellen (Certificate Authorities, CAs) ausgestellt. Ein TLS-Zertifikat enthält den öffentlichen Schlüssel des Servers, den Namen des Ausstellers, den Namen des Inhabers und die Gültigkeitsdauer des Zertifikats.

TLS-Schlüssel

Im TLS-Protokoll gibt es zwei Hauptarten von Schlüsseln:

- **Öffentlicher Schlüssel:** Dieser wird im TLS-Zertifikat bereitgestellt und dient zur Verschlüsselung von Nachrichten, die nur mit dem entsprechenden privaten Schlüssel entschlüsselt werden können.
- **Privater Schlüssel:** Dieser bleibt geheim und wird verwendet, um Nachrichten zu entschlüsseln, die mit dem öffentlichen Schlüssel verschlüsselt wurden, sowie um digitale Signaturen zu erzeugen.

TLS-Secret

Ein TLS-Secret ist ein spezielles Kubernetes Secret, das TLS-Zertifikate und Schlüssel enthält. Diese Secrets werden verwendet, um sichere Verbindungen zwischen Kubernetes-Komponenten oder externen Diensten zu ermöglichen. TLS-Secrets enthalten typischerweise die folgenden Daten:

- **tls.crt:** Das TLS-Zertifikat.
- **tls.key:** Der private Schlüssel.

Erstellung und Verwendung von TLS-Secrets

TLS-Secrets können in Kubernetes mit dem folgenden Befehl erstellt werden:

```
kubectl create secret tls tls-secret --cert=path/to/tls.crt --key=path/to/tls.key
```

Sobald das TLS-Secret erstellt wurde, kann es in Pods und anderen Kubernetes-Komponenten verwendet werden, um sichere Verbindungen zu konfigurieren. Zum Beispiel kann ein Ingress-Controller das TLS-Secret verwenden, um HTTPS-Verbindungen zu terminieren.

2.2.4 Anzeigen und Bearbeiten von Secrets

Inhalt eines Secrets anzeigen:

```
kubectl get secret my-secret -o yaml
```

Die Daten in einem Secret werden in einem Base64-kodiertem Format dargestellt.

Dekodierung der Daten:

```
echo 'c29tZXNlY3JldA==' | base64 --decode
```

Bearbeiten eines Secrets:

```
kubectl edit secret my-secret
```

2.2.5 Sicherheit und Best Practices

Im Umgang mit Secrets ist es besonders wichtig, Sicherheitsaspekte zu berücksichtigen. Die folgenden Maßnahmen helfen dabei, den Schutz sensibler Daten sicherzustellen:

- **Secrets verschlüsseln:** Den etcd-Datenspeicher verschlüsseln, z. B. mit AES-256 oder einem externen KMS.
- **Zugriffsrechte beschränken:** Zugriff auf Secrets per RBAC auf das notwendige Minimum reduzieren.
- **Namespaces isolieren:** Secrets in dedizierten Namespaces speichern, um Sichtbarkeit zu begrenzen.
- **Secrets rotieren:** Secrets und Zertifikate regelmäßig erneuern, um ihre Gültigkeit sicherzustellen.
- **Kurzlebige Tokens bevorzugen:** Statt dauerhaft gültiger Secrets besser zeitlich limitierte Zugriffstoken verwenden.
- **Zentrale Verwaltung nutzen:** Tools wie `cert-manager`, Vault oder CSI Drivers einsetzen, um Secrets zentral zu verwalten.
- **Auditlogs aktivieren:** Zugriffe auf Secrets mitprotokollieren, um unautorisierte Aktivitäten zu erkennen.
- **Nur notwendige Mounts konfigurieren:** Secrets nur in Pods einbinden, die sie wirklich benötigen.
- **Keine Protokollierung von Geheimnissen:** Sicherstellen, dass sensible Informationen nicht versehentlich in Logs geschrieben werden.
- **StringData verwenden:** Bei der Definition von Secrets in YAML-Dateien `stringData` anstelle von `data` verwenden, um Klartext direkt zu schreiben.
- **Versionskontrolle vermeiden:** Secrets niemals in Git oder andere Versionskontrollsysteme einchecken

Zusätzliche Ressourcen

Offizielle Dokumentation: https://kubernetes.io/docs/concepts/security/secrets-good-practices/

2.2.6 Verschlüsselung von etcd

Die Verschlüsselung der etcd-Datenbank in Kubernetes ist entscheidend für die Sicherheit sensibler Daten wie Secrets und ConfigMaps. Durch die Verschlüsselung der etcd-Daten wird sichergestellt, dass selbst im Falle eines unbefugten Zugriffs auf die etcd-Datenbank die Daten nicht im Klartext lesbar sind.

Konfiguration für die Verschlüsselung von etcd

Die Verschlüsselung wird durch eine Konfigurationsdatei aktiviert, die die gewünschten Provider (z. B. AES-CBC oder AES-GCM) und Schlüssel angibt:

```yaml
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: <base64-encoded-key>
      - identity: {}
```

Aktivierung

Die Konfigurationsdatei muss gespeichert und der API-Server mit dem Parameter `--encryption-provider-config=<pfad-zur-config-datei>` gestartet werden. Anschließend sollte geprüft werden, ob der API-Server erfolgreich läuft und die Verschlüsselung angewendet wird.

Fehlerbehebung

Typische Fehlerquellen bei der Verschlüsselung:

- Ungültiger oder nicht erreichbarer Pfad zur Konfigurationsdatei
- Syntaxfehler oder fehlende Felder in der YAML-Datei
- Falsch kodierte oder zu kurze Base64-Schlüssel
- Fehler in den API-Server-Logs (mit `journalctl` oder `kubectl logs`)

Best Practices

- **Starke Algorithmen verwenden:** AES-256 mit GCM oder CBC-Modus nutzen.
- **Schlüssel regelmäßig rotieren:** Schlüsselwechsel in der Encryption-Konfigurationsdatei eintragen und Rotation durch Reencryption manuell auslösen.
- **Verschlüsselung testen:** Vor dem Produktiveinsatz die Konfiguration in einer Testumgebung verifizieren.
- **Auditierung aktivieren:** API-Zugriffe auf Secrets und etcd-Zugriffe überwachen.
- **Replikation absichern:** Auch etcd-Replikation über TLS verschlüsseln.
- **Backup-Verschlüsselung beachten:** Sicherstellen, dass Backups von etcd ebenfalls verschlüsselt sind.

Zusätzliche Ressourcen

Offizielle Dokumentation: https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/
Security Best Practices: https://kubernetes.io/docs/concepts/security/overview/

2.3 Kubeconfig

Die `kubeconfig`-Datei regelt den Zugriff auf Kubernetes-Cluster. Sie speichert Konfigurationsinformationen wie Cluster-Adressen, Benutzeranmeldeinformationen und Kontexte.
Ein **Kontext** ist eine Kombination aus einem Cluster, einem Benutzer und einem Namespace. Er definiert die Umgebung, in der `kubectl`-Befehle ausgeführt werden. Durch den Wechsel des Kontexts lässt sich einfach zwischen verschiedenen Clustern oder Rollen umschalten, ohne die Konfiguration manuell zu ändern.

Befehl	Beschreibung
`kubectl config view`	Aktuelle Kubeconfig-Datei anzeigen
`kubectl config get-contexts`	Alle verfügbaren Kontexte auflisten
`kubectl config current-context`	Aktuell verwendeten Kontext anzeigen
`kubectl config use-context <context-name>`	Zu einem anderen Kontext wechseln
`kubectl config set-context <context-name>`	Einen neuen Kontext setzen
`kubectl config delete-context <context-name>`	Einen bestimmten Kontext löschen
`kubectl config rename-context <old-name> <new-name>`	Einen vorhandenen Kontext umbenennen
`kubectl config set-cluster <cluster-name> --server=<server-url>`	Einen neuen Cluster hinzufügen
`kubectl config set-credentials <user-name> --token=<token>`	Benutzerdaten über Token hinzufügen oder ändern
`kubectl config set-credentials <user-name> --username=<username> --password=<password>`	Benutzeranmeldeinformationen setzen
`kubectl config unset <property>`	Eine bestimmte Eigenschaft aus der Kubeconfig-Datei entfernen
`KUBECONFIG=<path/to/kubeconfig> kubectl ...`	Eine spezifische Kubeconfig-Datei für einen Befehl verwenden

Kubeconfig-Datei

```yaml
apiVersion: v1
kind: Config
clusters:
  - cluster:
      certificate-authority-data: <base64-encoded-ca-cert>
      server: https://<cluster-server>
    name: beispiel-cluster
contexts:
  - context:
      cluster: beispiel-cluster
      user: beispiel-benutzer
    name: beispiel-kontext
current-context: beispiel-kontext
users:
  - name: beispiel-benutzer
    user:
      token: <user-token>
```

2.3.1 Verwalten von Konfigurationsdateien

Kubectl verwendet standardmäßig die `$HOME/.kube/config`-Datei zur Speicherung der Konfigurationsinformationen. Es können jedoch auch mehrere Konfigurationsdateien verwendet und diese mit der Umgebungsvariable `KUBECONFIG` verwaltet werden.

Verwenden mehrerer Konfigurationsdateien

`KUBECONFIG`-Umgebungsvariable für mehrere Konfigurationsdateien setzen:

```
export KUBECONFIG=\$HOME/.kube/config:\$HOME/.kube/config-2
```

Konfigurationsdateien zusammenführen:

```
kubectl config view --merge --flatten > \$HOME/.kube/merged-config
export KUBECONFIG=\$HOME/.kube/merged-config
```

Hinzufügen eines neuen Clusters und Kontexts

```
# Neuen Cluster hinzufügen
kubectl config set-cluster my-cluster --server=https://my-cluster-server:6443
↪  --certificate-authority=/path/to/ca.crt
# Benutzeranmeldeinformationen hinzufügen
kubectl config set-credentials my-user --client-certificate=/path/to/client.crt
↪  --client-key=/path/to/client.key
# Einen neuen Kontext setzen
kubectl config set-context my-context --cluster=my-cluster --namespace=my-namespace --user=my-user
# Zu dem neuen Kontext wechseln
kubectl config use-context my-context
```

Überprüfen des aktuellen Kontexts

```
kubectl config current-context
```

2.3.2 Best Practices für das Verwalten von Kubernetes-Konfigurationen

- Aussagekräftige Namen für Kontexte, Cluster und Benutzer wählen.
- `kubeconfig`-Dateien regelmäßig sichern und sicher speichern.
- Verschiedene Umgebungen mit separaten Konfigurationsdateien verwalten.
- Konfigurationen und deren Verwendungszwecke dokumentieren.
- Rollen- und Berechtigungsmanagement (RBAC) zur Zugriffskontrolle nutzen.
- Veraltete oder ungenutzte Einträge regelmäßig entfernen.
- Änderungen an Konfigurationsdateien mit Versionskontrolle (z.B. Git) verwalten
- Sensible Informationen (Tokens, Passwörter) nicht in der Kubeconfig-Datei im Klartext speichern.
- Zugriff auf `.kube/config` auf Lesezugriff für den Nutzer beschränken (z.B. `chmod 600`).
- Automatisierte Rotationen von Zugangstokens mit ServiceAccounts bevorzugen.

Zusätzliche Ressourcen

Offizielle Dokumentation: https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/

2.3.3 Nützliche Befehle zur Fehlerbehebung

Befehl	Beschreibung
`kubectl config current-context`	Den aktuell verwendeten Kontext anzeigen
`kubectl config view –minify`	Die aktive Konfiguration ohne redundante Informationen anzeigen
`kubectl config get-clusters`	Alle definierten Cluster in der Konfigurationsdatei auflisten
`kubectl config get-users`	Alle definierten Benutzer in der Konfigurationsdatei auflisten
`kubectl config get-contexts`	Alle definierten Kontexte in der Konfigurationsdatei auflisten
`kubectl config delete-cluster <cluster-name>`	Einen bestimmten Cluster aus der Konfigurationsdatei löschen
`kubectl config delete-user <user-name>`	Einen bestimmten Benutzer aus der Konfigurationsdatei löschen
`kubectl config rename-context <old-name> <new-name>`	Einen Kontext umbenennen

2.3.4 Wechseln zwischen verschiedenen Kontexten

```
# Wechseln zum Entwicklungs-Kontext
kubectl config use-context dev-context

# Überprüfen des aktuellen Kontexts
kubectl config current-context

# Wechseln zum Produktions-Kontext
kubectl config use-context prod-context

# Überprüfen des aktuellen Kontexts
kubectl config current-context
```

2.3.5 Verwendung von Kontexten in Skripten

```
#!/bin/bash

# Setzen des Kubernetes-Kontexts
kubectl config use-context dev-context

# Überprüfen des aktuellen Kontexts
current_context=$(kubectl config current-context)
echo "Aktueller Kontext: $current_context"

# Ausführen von Kubernetes-Befehlen im aktuellen Kontext
kubectl get pods -n my-namespace

# Wechseln zum Produktions-Kontext
kubectl config use-context prod-context

# Überprüfen des aktuellen Kontexts
current_context=$(kubectl config current-context)
echo "Aktueller Kontext: $current_context"

# Ausführen von Kubernetes-Befehlen im Produktions-Kontext
kubectl get pods -n my-namespace
```

3 Skalierung und Autoscaling

3.1 Autoscalers

Kubernetes unterstützt verschiedene Mechanismen zur automatischen Skalierung von Workloads und Infrastruktur. Diese Skalierungsarten lassen sich in drei Hauptkategorien unterteilen:

- **Horizontal Pod Autoscaler (HPA):** Skalierung der Anzahl von Pods.
- **Vertical Pod Autoscaler (VPA):** Anpassung der Ressourcenlimits von Pods.
- **Cluster Autoscaler:** Skalierung der Anzahl der Nodes im Cluster.

3.1.1 Horizontal Pod Autoscalers (HPA)

Der HPA skaliert die Anzahl von Pods in einem Deployment, StatefulSet oder ReplicaSet basierend auf Metriken wie CPU-Auslastung oder benutzerdefinierten Indikatoren.

`kubectl get hpa`	Liste aller HPAs
`kubectl autoscale deployment <deployment-name>`	Deployment automatisch skalieren
`kubectl describe hpa <hpa-name>`	Details eines HPAs anzeigen
`kubectl delete hpa <hpa-name>`	HPA löschen
`kubectl edit hpa <hpa-name>`	HPA bearbeiten
`kubectl scale –replicas=<number> deployment <deployment-name>`	Manuelle Skalierung der Pods

Beispielkonfiguration eines Horizontal Pod Autoscalers

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: beispiel-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: beispiel-deployment
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

3.1.2 Vertical Pod Autoscalers (VPA)

Automatische Anpassung der Ressourcenanforderungen (CPU und Speicher) von Pods anhand der tatsächlichen Nutzung über die Zeit.

`kubectl get vpa`	Liste aller VPAs
`kubectl describe vpa <vpa-name>`	Details eines VPAs anzeigen
`kubectl delete vpa <vpa-name>`	VPA löschen
`kubectl apply -f <vpa-config.yaml>`	VPA aus YAML erstellen
`kubectl edit vpa <vpa-name>`	VPA bearbeiten
`kubectl get vpa -o yaml`	YAML-Ausgabe aller VPAs
`kubectl patch vpa <vpa-name> –patch '<json-patch>'`	VPA patchen

Beispielkonfiguration eines Vertical Pod Autoscalers:

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: beispiel-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: beispiel-deployment
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

3.1.3 Cluster Autoscalers

Der Cluster Autoscaler erhöht oder reduziert automatisch die Anzahl der Nodes im Cluster, je nach Ressourcenbedarf der Pods. Nicht schedulbare Pods oder unterausgelastete Nodes sind typische Trigger.

`kubectl get cluster-autoscaler`	Liste aller Cluster Autoscaler
`kubectl describe cluster-autoscaler <autoscaler-name>`	Details anzeigen
`kubectl delete cluster-autoscaler <autoscaler-name>`	Löschen eines Autoscalers
`kubectl apply -f <autoscaler-config.yaml>`	Autoscaler konfigurieren
`kubectl edit cluster-autoscaler <autoscaler-name>`	Autoscaler bearbeiten
`kubectl get cluster-autoscaler -o yaml`	YAML-Ausgabe
`kubectl patch cluster-autoscaler <autoscaler-name> –patch '<json-patch>'`	Patch anwenden

Beispielkonfiguration eines Cluster Autoscalers:

```yaml
apiVersion: autoscaling.k8s.io/v1
kind: ClusterAutoscaler
metadata:
  name: beispiel-cluster-autoscaler
spec:
  scaleDown:
    enabled: true
    unneededTime: 10m
    utilizationThreshold: 0.5
```

3.2 Manuelle Skalierung

Die manuelle Skalierung erlaubt es, die Anzahl der Replikate eines Deployments, StatefulSets, ReplicaSets oder ReplicationControllers unabhängig von automatischen Mechanismen anzupassen.

Befehl	Beschreibung
`kubectl scale deployment <deployment-name> --replicas=<number>`	Replikate eines Deployments setzen
`kubectl scale statefulset <statefulset-name> --replicas=<number>`	Replikate eines StatefulSets setzen
`kubectl scale rc <replication-controller-name> --replicas=<number>`	Replikate eines ReplicationControllers setzen
`kubectl scale replicaset <replicaset-name> --replicas=<number>`	Replikate eines ReplicaSets setzen
`kubectl get deployment <deployment-name> -o=jsonpath='.spec.replicas'`	Aktuelle Replikate eines Deployments anzeigen
`kubectl get statefulset <statefulset-name> -o=jsonpath='.spec.replicas'`	Aktuelle Replikate eines StatefulSets anzeigen
`kubectl get rc <replication-controller-name> -o=jsonpath='.spec.replicas'`	Aktuelle Replikate eines ReplicationControllers anzeigen
`kubectl get replicaset <replicaset-name> -o=jsonpath='.spec.replicas'`	Aktuelle Replikate eines ReplicaSets anzeigen

Eine umfangreiche Erklärung zu jsonpath befindet sich bei 10.4.9

> **Achtung:** Die manuelle Skalierung überschreibt temporär Einstellungen eines `HorizontalPodAutoscaler`. Bei aktiver automatischer Skalierung kann sich die manuell gesetzte Replikazahl daher wieder ändern.

3.2.1 Best Practices für die Skalierung

- **Ressourcenanforderungen definieren:** `resources.requests` und `resources.limits` für alle Pods angeben, um fundierte Entscheidungen durch Autoscaler zu ermöglichen.
- **Metriken überwachen:** CPU- und Speicherauslastung regelmäßig prüfen und gegebenenfalls benutzerdefinierte Metriken einbinden (z. B. via Prometheus Adapter).
- **Initiale Replikazahl festlegen:** Eine Ausgangsgröße wählen, die eine schnelle Verfügbarkeit sicherstellt, ohne erst bei Lastanstieg skalieren zu müssen.
- **HPA und VPA nicht kombinieren:** HPA und VPA nicht gleichzeitig auf dieselben Deployments anwenden, da sich ihre Wirkungen überschneiden können.
- **Cluster-Autoscaler begrenzen:** Minimal- und Maximalanzahl an Nodes setzen, um unkontrolliertes Skalieren oder Ressourcenmangel zu vermeiden.
- **Cooldown-Zeiten konfigurieren:** Eine sinnvolle Downscale-Stabilisierung (z. B. mit `--horizontal-pod-autoscaler-downscale-stabilization`) einstellen, um häufiges Hin- und Herschalten zu verhindern.
- **Skalierungsverhalten testen:** Änderungen an Skalierungsregeln in einer Testumgebung prüfen, bevor sie produktiv eingesetzt werden.

Weitere Ressourcen

HPA: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
Autoscaling Workloads: https://kubernetes.io/docs/concepts/workloads/autoscaling/

4 Speicherverwaltung

4.1 PersistentVolumes

Stellen dauerhaften Speicher bereit, der unabhängig vom Lebenszyklus von Pods existiert.

4.1.1 Erstellung und Verwaltung von PersistentVolumes

`kubectl get pv`	Alle PersistentVolumes auflisten
`kubectl describe pv <pv-name>`	Details zu einem PersistentVolume anzeigen
`kubectl delete pv <pv-name>`	PersistentVolume löschen
`kubectl apply -f <pv-config.yaml>`	PersistentVolume erstellen
`kubectl edit pv <pv-name>`	PersistentVolume bearbeiten
`kubectl get pv -o yaml`	Alle PersistentVolumes anzeigen
`kubectl patch pv <pv-name> -patch '<json-patch>'`	PersistentVolume aktualisieren

Beispielkonfiguration eines PersistentVolumes

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-example
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: /mnt/data
```

Dies erstellt ein PersistentVolume mit 10 GiB Speicherplatz, das auf dem Hostpfad `/mnt/data` basiert.

4.2 PersistentVolumeClaims

PVs werden in Pods über PersistentVolumeClaims (PVCs) verwendet:

`kubectl get pvc`	PersistentVolumeClaims auflisten
`kubectl describe pvc <pvc-name>`	Details anzeigen
`kubectl delete pvc <pvc-name>`	PersistentVolumeClaim löschen
`kubectl apply -f <pvc-config.yaml>`	PersistentVolumeClaim erstellen
`kubectl edit pvc <pvc-name>`	PersistentVolumeClaim bearbeiten
`kubectl get pvc -o yaml`	PersistentVolumeClaims anzeigen
`kubectl patch pvc <pvc-name> -patch '<json-patch>'`	PersistentVolumeClaim aktualisieren

Verwendung eines PersistentVolumeClaims in einem Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: my-image
      volumeMounts:
        - mountPath: /mnt/data
          name: my-volume
  volumes:
    - name: my-volume
      persistentVolumeClaim:
        claimName: pvc-example
```

4.2.1 Reclaim Policies

Die Reclaim-Policy eines PersistentVolumes bestimmt, was mit dem Volume passiert, wenn es nicht mehr benötigt wird. Sie werden in der YAML-Datei einer PersistentVolume unter dem Punkt „`persistentVolumeReclaimPolicy`“ festgelegt. Die möglichen Werte sind:

- `Retain`: Das Volume bleibt bestehen und muss manuell gelöscht werden.
- `Recycle`: Das Volume wird gelöscht und neu erstellt (veraltet und oft nicht unterstützt).
- `Delete`: Das Volume und die dahinterliegende Speicherressource werden gelöscht.

4.3 StorageClasses

StorageClasses definieren die verschiedenen Klassen von Speicher, die in einem Kubernetes-Cluster verwendet werden können. Sie ermöglichen die dynamische Bereitstellung von PersistentVolumes.

Befehle zur Verwaltung von StorageClasses

Befehl	Beschreibung
`kubectl get storageclass`	Alle verfügbaren StorageClasses auflisten
`kubectl describe storageclass <storageclass-name>`	Details zu einer bestimmten StorageClass anzeigen
`kubectl create -f <storageclass.yaml>`	Eine neue StorageClass anhand einer YAML-Datei erstellen
`kubectl delete storageclass <storageclass-name>`	Eine StorageClass löschen
`kubectl annotate storageclass <storageclass-name> <key>=<value>`	Eine Annotation zu einer StorageClass hinzufügen
`kubectl edit storageclass <storageclass-name>`	Eine StorageClass im Editor bearbeiten
`kubectl patch storageclass <storageclass-name> -p <patch-data>`	Eine StorageClass mit Patch-Daten ändern

Beispielkonfiguration einer StorageClass

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

Referenzierung der StorageClass durch einen PVC

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  storageClassName: fast
```

4.3.1 Beispiele für verschiedene StorageClasses

NFS StorageClass

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs
provisioner: example.com/nfs
parameters:
  server: 192.168.1.1
  path: /exported/path
```

GCE Persistent Disk StorageClass

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  replication-type: none
```

Azure Disk StorageClass

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: azure-disk
provisioner: kubernetes.io/azure-disk
parameters:
  skuName: Standard_LRS
  location: eastus
```

AWS EBS StorageClass

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: aws-ebs
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ext4
```

4.3.2 Standard-StorageClass festlegen

In Kubernetes kann eine StorageClass als Standard festgelegt werden. Dies bedeutet, dass alle PVCs, die keine spezifische StorageClass angeben, diese Standard-StorageClass verwenden. Um eine Standard-StorageClass festzulegen, wird die Annotation „`storageclass.kubernetes.io/is-default-class`“ auf `true` gesetzt:

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
```

Best Practices für StorageClasses

- Eindeutige Benennung
- Sorgfältige Verwaltung der Zugriffsrechte
- Nutzung von Labels und Annotations, um zusätzliche Metadaten zu speichern und die Verwaltung zu erleichtern.
- Überwachen der Nutzung von PersistentVolumes und StorageClasses und Rotierung der zugrunde liegenden Ressourcen, um die Sicherheit und Performance zu gewährleisten.

Nützliche Links und Ressourcen

Kubernetes StorageClass Dokumentation:
https://kubernetes.io/docs/concepts/storage/storage-classes/
Kubernetes Task: Change the Default StorageClass
https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
Kubernetes Blog: Dynamic Provisioning and StorageClasses in Kubernetes:
https://kubernetes.io/blog/2017/03/dynamic-provisioning-and-storage-classes-kubernetes/
Kubernetes Examples: Persistent Volume Provisioning:
https://github.com/kubernetes/examples/tree/master/staging/persistent-volume-provisioning/

4.4 Verteilte Dateisysteme

Verteilte Dateisysteme wie HDFS (Hadoop Distributed File System) sind darauf ausgelegt, große Datenmengen über mehrere Knoten hinweg zu speichern und zu verwalten. Sie bieten hohe Verfügbarkeit, Fehlertoleranz und parallelen Zugriff auf die Daten, was sie ideal für Big-Data-Anwendungen macht.

Einführung in verteilte Dateisysteme wie HDFS

HDFS ist ein verteiltes Dateisystem, das für die Speicherung sehr großer Dateien konzipiert ist. Es teilt jede Datei in mehrere Blöcke auf, die über verschiedene Knoten verteilt werden. Jeder Block wird normalerweise mehrfach repliziert, um Fehlertoleranz zu gewährleisten.

4.4.1 Verwendung von verteilten Dateisystemen in Kubernetes

In Kubernetes können verteilte Dateisysteme als Speicherlösung für Anwendungen verwendet werden, die große Datenmengen verarbeiten müssen. Sie können Kubernetes Persistent Volumes (PV) und Persistent Volume Claims (PVC) nutzen, um Speicherplatz in verteilten Dateisystemen zu verwalten.

Beispielkonfiguration für verteiltes Dateisystem mit PersistentVolume

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: hdfs-pv
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteMany
  hdfs:
    namenode: "hdfs://namenode.example.com:8020"
    path: "/data"
```

YAML-Datei für verteiltes Dateisystem mit PersistentVolumeClaim

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: hdfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
```

Verwaltung und Best Practices

- Fehlertoleranz durch ausreichende Replikation der Daten sicherstellen
- Regelmäßige Überwachung der Gesundheit und Leistung des Dateisystems
- Aufteilung von großen Datenmengen in kleinere Blöcke, um die Parallelisierung zu maximieren und die Verarbeitung zu beschleunigen
- sorgfältige Verwaltung der Zugriffsrechte
- Robuste Backup- und Wiederherstellungsstrategien implementieren, um Datenverlust zu vermeiden

Beispiele für verteilte Dateisysteme

- **Ceph**: Ein verteiltes Speichersystem, das sowohl Block-, Datei- als auch Objektspeicher unterstützt. Es ist bekannt für seine Skalierbarkeit und Zuverlässigkeit.
- **GlusterFS**: Ein weiteres verteiltes Dateisystem, das für seine Flexibilität und Einfachheit bekannt ist.
- **Amazon S3**: Ein verteiltes Objektspeichersystem von Amazon Web Services, das oft in Verbindung mit Kubernetes verwendet wird.

Integration von verteilten Dateisystemen in Kubernetes

Die Integration von verteilten Dateisystemen in Kubernetes kann mithilfe von CSI (Container Storage Interface) Treibern erfolgen.

YAML-Datei für ein CephFS PersistentVolume

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cephfs-pv
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteMany
  csi:
    driver: cephfs.csi.ceph.com
    volumeHandle: my-cephfs-volume
    volumeAttributes:
      clusterID: my-cluster-id
      fsName: my-cephfs
      pool: my-pool
```

YAML-Datei für ein CephFS PersistentVolumeClaim

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cephfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
```

4.4.2 Nützliche Links und Ressourcen

- Kubernetes Volumes Dokumentation
- Kubernetes Persistent Volumes Dokumentation
- Kubernetes Storage Classes Dokumentation
- Kubernetes Blog: Local Persistent Volumes Beta
- Kubernetes Examples: Persistent Volume Provisioning
- Rook: Cloud-Native Storage Orchestrator for Kubernetes

- Ceph: Distributed Storage in Kubernetes
- Gluster Documentation

4.5 Container Storage Interface (CSI) Treiber

Der Container Storage Interface (CSI) ist ein Standard, der es Speichersystemen ermöglicht, sich nahtlos in Container-Orchestrierungsplattformen wie Kubernetes zu integrieren. CSI-Treiber sind Plugins, die es Kubernetes ermöglichen, verschiedene Speicherlösungen dynamisch zu verwalten und zu orchestrieren.

4.5.1 Vorteile von CSI-Treibern

- **Flexibilität**: CSI-Treiber ermöglichen die Integration einer Vielzahl von Speichersystemen, einschließlich Cloud-Speicher, verteilten Dateisystemen und traditionellen Speicherlösungen.
- **Portabilität**: Anwendungen können einfacher zwischen verschiedenen Kubernetes-Umgebungen migriert werden, da die Speicherintegration standardisiert ist.
- **Erweiterbarkeit**: Neue Speichertechnologien können durch die Entwicklung spezieller CSI-Treiber leicht in Kubernetes integriert werden.

4.5.2 Integration von verteilten Dateisystemen in Kubernetes

Die Integration von verteilten Dateisystemen in Kubernetes kann mithilfe von CSI-Treibern erfolgen.

Integration von CephFS unter Verwendung des CSI-Treibers

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cephfs-pv
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteMany
  csi:
    driver: cephfs.csi.ceph.com
    volumeHandle: my-cephfs-volume
    volumeAttributes:
      clusterID: my-cluster-id
      fsName: my-cephfs
      pool: my-pool
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cephfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
```

Durch die Verwendung von CSI-Treibern können verschiedene verteilte Dateisysteme nahtlos in Kubernetes integriert werden, was eine flexible und skalierbare Speicherlösung bietet.

4.5.3 Beispiele für CSI-Treiber

- **cephfs.csi.ceph.com**: Ermöglicht die Integration von CephFS in Kubernetes.
- **glusterfs.csi.gluster.org**: Ermöglicht die Integration von GlusterFS.
- **aws.ebs.csi.aws.com**: Ermöglicht die Integration von Amazon EBS.
- **gce-pd.csi.storage.gke.io**: Ermöglicht die Integration von Google Persistent Disks.
- **csi.s3.amazonaws.com**: Ermöglicht die Integration von Amazon S3.

4.5.4 Best Practices für die Verwendung von CSI-Treibern

- Treiberkompatibilität sicherstellen
- CSI-Treiber auf dem neuesten Stand halten
- Performance Tests durchführen
- Backup- und Wiederherstellungsstrategien, die speziell für die verwendeten CSI-Treiber optimiert sind implementieren

4.5.5 Nützliche Links und Ressourcen

- Kubernetes CSI Volumes Dokumentation
- Offizielle Kubernetes CSI Dokumentation
- Kubernetes Blog: Container Storage Interface (CSI) GA
- Kubernetes CSI GitHub Repository

5 Workloads und Ressourcen

5.1 DaemonSets

Stellen sicher, dass eine Kopie eines Pods auf jeder Node (oder einer ausgewählten Gruppe von Nodes) im Cluster läuft.

Befehl	Beschreibung
`kubectl get daemonsets`	Alle DaemonSets auflisten
`kubectl describe daemonset <daemonset-name>`	Details zu einem DaemonSet anzeigen
`kubectl create -f <filename>`	DaemonSet aus einer Datei erstellen
`kubectl apply -f <filename>`	DaemonSet aus einer Datei erstellen, oder aktualisieren
`kubectl delete daemonset <daemonset-name>`	Ein DaemonSet löschen

YAML-Datei für ein 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:latest
          ports:
            - containerPort: 80
```

5.1.1 Anwendungsfälle für DaemonSets

- Protokollierung: Ein Protokollierungs-Agent muss auf jeder Node laufen, um die Logs von Anwendungen zu sammeln.
- Überwachung: Überwachungs-Agents sammeln Metriken von Nodes und Pods und senden sie an ein zentrales Überwachungssystem.
- Netzwerk-Dienste: Bereitstellung von Netzwerkdiensten wie DNS oder CNI-Plugins, die auf jeder Node laufen müssen.

5.1.2 Best Practices für DaemonSets

- Ressourcenanforderungen und limits setzen, um die Stabilität des Clusters zu gewährleisten.
- Node-Selector verwenden, wenn DaemonSets nur auf bestimmten Nodes ausgeführt werden sollen
- PodAntiAffinity verwenden, um sicherzustellen dass DaemonSet-Pods nicht auf denselben Nodes wie andere kritische Pods laufen.
- Rolling Updates verwenden, um eine reibungslose Aktualisierung der DaemonSet-Pods ohne Ausfallzeiten zu gewährleisten.

5.1.3 Weitere nützliche Befehle für DaemonSets

Befehl	Beschreibung
`kubectl rollout status daemonset/<daemonset-name>`	Den Rollout-Status eines DaemonSets anzeigen
`kubectl edit daemonset <daemonset-name>`	Ein DaemonSet direkt im Editor bearbeiten
`kubectl scale –replicas=<number> daemonset <daemonset-name>`	Die Anzahl der Pods in einem DaemonSet ändern
`kubectl get daemonset <daemonset-name> -o yaml`	Die vollständige YAML-Konfiguration eines DaemonSets anzeigen
`kubectl patch daemonset <daemonset-name> -p <patch-data>`	Einen Patch auf ein DaemonSet anwenden

5.1.4 Nützliche Links und Ressourcen

- Kubernetes DaemonSet Dokumentation
- Kubernetes Tasks für DaemonSets

5.2 StatefulSets

Verwalten den Zustand und die Identität von Pods, insbesondere für zustandsbehaftete Anwendungen.

Befehl	Beschreibung
`kubectl get statefulsets`	Alle StatefulSets auflisten
`kubectl describe statefulset <statefulset-name>`	Details zu einem StatefulSet anzeigen
`kubectl apply -f <dateiname>`	StatefulSet aus Datei erstellen
`kubectl delete statefulset <statefulset-name>`	Ein StatefulSet löschen

Beispielkonfiguration für ein StatefulSet

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

5.2.1 Anwendungsfälle für StatefulSets

- Datenbanken: StatefulSets bieten stabile Netzwerk-IDs und persistente Speicher, die für Datenbanken erforderlich sind.
- Verteilte Dateisysteme: Verteilte Dateisysteme wie HDFS (siehe 4.4) benötigen stabile IDs und persistente Speicher.
- Clustered Anwendungen: Anwendungen, die auf Clustering angewiesen sind, benötigen stabile Netzwerk-IDs.

5.2.2 Best Practices für StatefulSets

- Headless Services (siehe 1.4.2) verwenden, um DNS-Namen für individuelle Pods zu ermöglichen
- Ressourcenanforderungen und -limits setzen, um die Stabilität des Clusters zu gewährleisten.
- PersistentVolumes verwenden, um sicherzustellen, dass der Speicher nach Pod-Neustarts erhalten bleibt.
- PodManagementPolicy verwenden, um die Verwaltung der Pods feiner zu steuern.

5.2.3 Weitere nützliche Befehle für StatefulSets

Befehl	Beschreibung
`kubectl rollout status statefulset/<statefulset-name>`	Rollout-Status anzeigen
`kubectl edit statefulset <statefulset-name>`	StatefulSet bearbeiten
`kubectl scale –replicas=<number> statefulset <statefulset-name>`	Anzahl der Pods ändern
`kubectl get statefulset <statefulset-name> -o yaml`	Konfiguration anzeigen
`kubectl patch statefulset <statefulset-name> -p <patch-data>`	Patch anwenden

5.2.4 Nützliche Links und Ressourcen

- Kubernetes StatefulSet Dokumentation
- Kubernetes Task: Skalieren von StatefulSets
- Kubernetes Tutorial: Basic StatefulSet

5.3 ReplicaSets

ReplicaSets stellen sicher, dass eine definierte Anzahl von Pod-Replikaten zu jeder Zeit läuft. Sie sind der Nachfolger von ReplicationControllers und bieten zusätzliche Selektionsmöglichkeiten.

Befehl	Beschreibung
`kubectl get replicasets`	ReplicaSets auflisten
`kubectl describe replicaset <replicaset-name>`	Details anzeigen
`kubectl create -f <replicaset.yaml>`	ReplicaSet aus Datei erstellen
`kubectl delete replicaset <replicaset-name>`	ReplicaSet löschen
`kubectl scale replicaset <replicaset-name> –replicas=<number>`	Anzahl der Pods skalieren
`kubectl get pods –selector=<label>=<value>`	Zugeordnete Pods anzeigen
`kubectl edit replicaset <replicaset-name>`	ReplicaSet bearbeiten
`kubectl patch replicaset <replicaset-name> -p <patch-data>`	ReplicaSet patchen

YAML-Datei für ein ReplicaSet

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

5.3.1 Anwendungsfälle für ReplicaSets

- Sicherstellen der Hochverfügbarkeit: ReplicaSets stellen sicher, dass eine bestimmte Anzahl von Pod-Replikaten immer läuft, um Ausfallzeiten zu minimieren.
- Lastverteilung: Durch das Erstellen mehrerer Replikate kann die Last auf mehrere Pods verteilt werden.
- Skalierung: Einfaches horizontales Skalieren von Anwendungen durch Ändern der Anzahl der Replikate.

5.3.2 Best Practices für ReplicaSets

- Labels und Selektoren verwenden, um sicherzustellen, dass nur die gewünschten Pods vom ReplicaSet verwaltet werden.
- Ressourcenanforderungen und -limits setzen, um die Stabilität und Effizienz des Clusters zu gewährleisten.
- ReplicaSets regelmäßig überwachen, um sicherzustellen, dass sie wie erwartet funktionieren und die gewünschte Anzahl von Pods läuft.
- den Befehl „`kubectl rollout`“ verwenden, um Änderungen und Updates an ReplicaSets zu überwachen und zu kontrollieren.

5.3.3 Weitere nützliche Befehle für ReplicaSets

Befehl	Beschreibung
`kubectl rollout status replicaset/<replicaset-name>`	Den Rollout-Status eines ReplicaSets anzeigen
`kubectl rollout history replicaset/<replicaset-name>`	Die Rollout-Historie eines ReplicaSets anzeigen
`kubectl set image replicaset/<replicaset-name> <container-name>=<new-image>`	Das Image eines Containers in einem ReplicaSet aktualisieren
`kubectl get rs -o wide`	Detaillierte Informationen zu allen ReplicaSets anzeigen
`kubectl get pods -l <label>=<value>`	Pods anzeigen, die einem bestimmten ReplicaSet zugeordnet sind
`kubectl delete pod –force –grace-period=0 <pod-name>`	Einen Pod sofort löschen, damit das ReplicaSet ihn neu erstellen kann
`kubectl scale replicaset nginx-replicaset –replicas=5`	Skalierung eines ReplicaSets über einen Konsolenbefehl

YAML-Datei für die Skalierung eines ReplicaSets

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
spec:
  replicas: 5 # Skalierung auf 5 Replikate
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

5.3.4 Fehlerbehebung bei ReplicaSets

- **Pod startet nicht:** Die Logs des Pods mit „`kubectl logs <pod-name>`“ überprüfen, um mögliche Fehlerursachen zu identifizieren.
- **Nicht genügend Ressourcen:** Sicherstellen, dass genug Ressourcen im Cluster verfügbar sind, oder Ressourcenanforderungen des ReplicaSets anpassen
- **Konflikte bei Labels:** Die Label-Selektoren überprüfen, um sicherzustellen, dass keine Konflikte mit anderen ReplicaSets oder Workloads bestehen.
- **Unzureichende Replikate:** „`kubectl get rs`“ und „`kubectl describe rs <replicaset-name>`“ verwenden, um den Status und eventuelle Probleme zu überprüfen.

5.3.5 Nützliche Links und Ressourcen

- Kubernetes ReplicaSet Dokumentation

5.4 Jobs

Ermöglichen die einmalige Ausführung von Aufgaben, bis sie erfolgreich abgeschlossen sind.

Befehl	Beschreibung
`kubectl get jobs`	Alle Jobs auflisten
`kubectl describe job <job-name>`	Details zu Job anzeigen
`kubectl create job -f <job.yaml>`	Job erstellen
`kubectl apply job -f <job.yaml>`	Job erstellen, oder aktualisieren
`kubectl delete job <job-name>`	Job löschen

YAML-Datei für einen Job

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: example-job
spec:
  template:
    spec:
      containers:
        - name: example
          image: busybox
          command:
            - echo "Hello, Kubernetes!"
      restartPolicy: Never
  backoffLimit: 4
```

5.4.1 Anwendungsfälle für Jobs

- Datenverarbeitung: Ausführen von einmaligen Datenverarbeitungsaufgaben oder Batch-Jobs.
- Migrationen: Durchführen von Datenbank- oder Schema-Migrationen.
- Wartungsaufgaben: Periodische Wartungsaufgaben, die einmalig ausgeführt werden müssen.

5.4.2 Best Practices für Jobs

- eine „`restartPolicy`“ von „`Never`“ oder „`OnFailure`“ verwenden, um sicherzustellen, dass fehlgeschlagene Jobs nicht endlos neu gestartet werden.
- „`backoffLimit`“ verwenden, um die maximale Anzahl von Wiederholungsversuchen für einen fehlgeschlagenen Job zu begrenzen.
- den Status von Jobs regelmäßig überwachen, um sicherzustellen, dass sie wie erwartet abgeschlossen werden.
- Labels und Annotations verwenden, um Jobs besser zu organisieren und zu identifizieren.

5.4.3 Weitere nützliche Befehle für Jobs

Befehl	Beschreibung
`kubectl get job <job-name> -o yaml`	Die vollständige YAML-Konfiguration eines Jobs anzeigen
`kubectl logs job/<job-name>`	Logs eines Jobs anzeigen
`kubectl get pods –selector=job-name=<job-name>`	Die Pods anzeigen, die zu einem bestimmten Job gehören
`kubectl wait –for=condition=complete job/<job-name>`	Warten, bis ein Job abgeschlossen ist
`kubectl patch job <job-name> -p <patch-data>`	Einen Patch auf einen Job anwenden

5.4.4 Fehlerbehebung bei Jobs

- **Job startet nicht:** Die Logs des Jobs mit `kubectl logs job/<job-name>` überprüfen, um mögliche Fehlerursachen zu identifizieren.
- **Pod des Jobs bleibt hängen:** Den Status und die Logs des Pods, der dem Job zugeordnet ist überprüfen.
- **Job wird zu oft wiederholt:** Sicherstellen, dass die `backoffLimit` korrekt gesetzt ist, um unnötige Wiederholungen zu vermeiden.
- **Job läuft zu lange:** `activeDeadlineSeconds` verwenden, um die maximale Laufzeit eines Jobs zu begrenzen.

5.4.5 Nützliche Links und Ressourcen

- Kubernetes Job Dokumentation
- Kubernetes Tasks für Jobs

5.5 CronJobs

Ermöglicht die zeitgesteuerte Ausführung von Jobs basierend auf Cron-Syntax. CronJobs sind nützlich für wiederkehrende Aufgaben wie regelmäßige Backups, Batch-Verarbeitungen oder andere zeitgesteuerte Prozesse. Sie bieten eine Möglichkeit, Jobs zu bestimmten Zeiten oder in regelmäßigen Intervallen automatisch auszuführen.

Befehl	Beschreibung
`kubectl get cronjobs`	Alle CronJobs auflisten
`kubectl describe cronjob <cronjob-name>`	Details zu einem CronJob anzeigen
`kubectl create -f cronjob.yaml`	CronJob erstellen
`kubectl apply -f cronjob.yaml`	CronJob erstellen oder aktualisieren
`kubectl delete cronjob <cronjob-name>`	Einen CronJob löschen

Grundlagen eines CronJobs

Ein CronJob erstellt auf Basis eines vordefinierten Zeitplans automatisch Jobs. Der Zeitplan wird in der Cron-Syntax angegeben, die aus fünf Feldern besteht, die die Minute, Stunde, Tag des Monats, Monat und Tag der Woche definieren.

YAML-Syntax für CronJobs

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: beispiel-cronjob
spec:
  schedule: '0 0 * * *'
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: beispiel-container
              image: beispiel-image
              args:
                - /bin/sh
                - -c
                - echo "Dies ist ein Beispiel-CronJob"
          restartPolicy: OnFailure
```

5.5.1 Anwendungsfälle für CronJobs

- Regelmäßige Backups: Automatisierung von Datenbank- oder Dateisystem-Backups.
- Batch-Verarbeitung: Durchführung von periodischen Datenverarbeitungsaufgaben.
- Wartungsaufgaben: Automatisierte Wartungsprozesse wie Log-Rotation oder Cleanup-Skripte.
- Monitoring und Updates: Regelmäßige Überprüfungen und Aktualisierungen von Systemzuständen.

5.5.2 Best Practices für CronJobs

- eine eindeutige „`schedule`“ verwenden, um sicherzustellen, dass CronJobs nicht gleichzeitig ausgeführt werden.
- eine geeignete „`concurrencyPolicy`“ setzen, um die gleichzeitige Ausführung von Jobs zu steuern (z.B. „`Forbid`“ oder „`Replace`“).
- „`successfulJobsHistoryLimit`“ und „`failedJobsHistoryLimit`“ verwenden, um die Anzahl der gespeicherten erfolgreichen und fehlgeschlagenen Jobs zu begrenzen.
- CronJobs regelmäßig überwachen, um sicherzustellen, dass sie wie erwartet ausgeführt werden.
- Ressourcenanforderungen und -limits setzen, um die Stabilität des Clusters zu gewährleisten.

5.5.3 Weitere nützliche Befehle für CronJobs

Befehl	Beschreibung
`kubectl get cronjob <cronjob-name> -o yaml`	Die vollständige YAML-Konfiguration eines CronJobs anzeigen
`kubectl logs job/$(kubectl get jobs -selector=cronjob-name=<cronjob-name> -o jsonpath=".items[0].metadata.name")`	Logs des letzten Jobs eines CronJobs anzeigen
`kubectl create job –from=cronjob/<cronjob-name> <job-name>`	Manuelles Erstellen eines Jobs aus einem CronJob
`kubectl get jobs –selector=cronjob-name=<cronjob-name>`	Alle Jobs eines bestimmten CronJobs auflisten
`kubectl patch cronjob <cronjob-name> -p <patch-data>`	Einen Patch auf einen CronJob anwenden
`kubectl delete job –selector=cronjob-name=<cronjob-name>`	Alle Jobs eines bestimmten CronJobs löschen

5.5.4 Fehlerbehebung bei CronJobs

- **Job startet nicht:** Logs des CronJobs und des erstellten Jobs mit folgendem Befehl überprüfen, um mögliche Fehlerursachen zu identifizieren:
  `kubectl logs job/$(kubectl get jobs –selector=cronjob-name=<cronjob-name> -o jsonpath=".items[0].metadata.name")`
- **CronJob wird nicht ausgeführt:** sicherstellen, dass der Zeitplan „`schedule`“ korrekt ist und keine Syntaxfehler enthält. Zeit und Datumseinstellungen des Clusters überprüfen.
- **Jobs bleiben hängen:** Die Pods, die von den Jobs erstellt wurden überprüfen, um zu sehen, ob sie in einem Fehlerzustand sind. „`kubectl describe pod <pod-name>`“ für detaillierte Informationen verwenden.
- **Mehrere Instanzen eines Jobs:** „`concurrencyPolicy`“ auf „`Forbid`“ oder „`Replace`“ setzen, um die gleichzeitige Ausführung mehrerer Instanzen eines Jobs zu verhindern.
- **CronJob führt Jobs zu häufig aus:**Zeitplan „`schedule`“ überprüfen und korrigieren, um sicherzustellen, dass er den gewünschten Ausführungsintervall widerspiegelt.

5.5.5 Erweiterte Konfigurationsoptionen für CronJobs

Erweiterte YAML-Syntax für CronJobs

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: erweiterter-cronjob
spec:
  schedule: "*/5 * * * *"
  concurrencyPolicy: Forbid
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      backoffLimit: 4
      template:
        spec:
          containers:
            - name: beispiel-container
              image: busybox
              args:
                - /bin/sh
                - -c
                - echo "Dies ist ein erweiterter CronJob"
          restartPolicy: OnFailure
```

5.5.6 Nützliche Links und Ressourcen

- Kubernetes CronJob Dokumentation
- Cron Expression Generator & Explainer

6 Probes

Bei Kubernetes werden Probes genutzt um herauszufinden ob ein Container läuft, bereit oder gestartet ist. Um diese Probes nutzen zu können müssen API-Endpunkte in der containerisierten Applikation definiert werden. Die Konvention ist es den Endpunkt für die Liveness Probe „livez“ zu nennen und den Endpunkt für die Readiness Probe „readyz“. Die Startup Probe verwendet üblicherweise denselben Endpunkt wie die Liveness Probe. Die Bezeichnung „healthz“ ist sein Kubernetes v1.16 veraltet.

6.1 Python FastAPI-Endpunkte

Um die Beispiele zu illustrieren wird FastAPI genutzt. Dafür müssen zuerst das „FastAPI“ und „Response“ Objekt importiert werden. „time“ wird in diesem Fall genutzt um eine Prüfung zu simulieren.

Python Code für FastAPI

```python
from fastapi import FastAPI
from fastapi.responses import PlainTextResponse
import time

app = FastAPI()
start_time = time.time()
```

6.2 Parameter für Probes

In der YAML-Datei gibt es folgende Felder, die für Probes definiert werden können:

Feld	Beschreibung
`initialDelaySeconds`	Anzahl der Sekunden nach dem Start des Containers, bevor die Startup-, Liveness- oder Readiness-Probe gestartet wird. Falls eine Startup-Probe definiert ist, beginnen die Verzögerungen der Liveness- und Readiness-Probes erst nach deren erfolgreichem Abschluss. Wenn `periodSeconds` größer ist als `initialDelaySeconds`, wird dieser Wert ignoriert. Standardwert: `0`, Minimum: `0`.
`periodSeconds`	Intervall (in Sekunden), in dem die Probe durchgeführt wird. Standardwert: `10`, Minimum: `1`. Solange ein Container nicht bereit ist, kann die Readiness-Probe häufiger als konfiguriert ausgeführt werden, um den Pod schneller bereitzustellen.
`timeoutSeconds`	Zeitlimit (in Sekunden), nach dem eine Probe als fehlgeschlagen gilt. Standardwert: `1`, Minimum: `1`.
`successThreshold`	Minimale Anzahl aufeinanderfolgender Erfolge, die erforderlich sind, damit eine zuvor fehlgeschlagene Probe als erfolgreich gilt. Standardwert: `1`, Minimum: `1`. Für Liveness- und Startup-Probes muss der Wert `1` sein.
`failureThreshold`	Anzahl aufeinanderfolgender Fehler, nach denen Kubernetes den Container als nicht bereit/nicht gesund betrachtet. Standardwert: `3`, Minimum: `1`. Bei Liveness- und Startup-Probes wird der Container nach Überschreitung neugestartet. Bei Readiness-Probes wird der Container weiter betrieben, jedoch als nicht bereit markiert.
`terminationGracePeriodSeconds`	Gibt an, wie lange der kubelet nach dem Auslösen des Shutdowns wartet, bevor der Container erzwungen beendet wird. Standardmäßig wird der pod-spezifische Wert übernommen (Standard: `30`). Minimum: `1`.

6.3 Liveness Probe

Python Code für livez-Endpunkt

Die Funktion livez besteht aus einem Endpunkt der den Statuscode 200 „ok“ zurückgibt, um zu zeigen dass alles wie erwartet funktioniert. Kubernetes akzeptiert hier alle Codes von 200-399. Schlägt der Aufruf fehl wird der Container neu gestartet, sobald der failureThreshold erreicht wurde. Dabei wartet die Liveness Probe nicht auf einen Erfolg durch andere Probes.

```python
@app.get("/livez", response_class=PlainTextResponse)
async def livez():
    return PlainTextResponse(content="ok", status_code=200)
```

Konfiguration für Liveness HTTP Probe

Die Liveness probe wird konfiguriert indem man einen Endpunkt, hier „livez“ und einen Port über den der Endpunkt läuft festlegt. Es wird ein initialDelay festlegt, vor dem keine Prüfung erfolgt und danach wird periodisch der Endpunkt erneut aufgerufen, um sicherzustellen dass die API weiterhin läuft.
Eine HTTP-Liveness Probe kann auch einen benannten Port verwenden.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: liveness-example-http
spec:
  containers:
    - name: app
      image: your-image:latest
      livenessProbe:
        httpGet:
          path: /livez
          port: 8080
        failureThreshold: 1
        initialDelaySeconds: 3
        periodSeconds: 10
```

Konfiguration für eine Liveness TCP Probe

Der Probe-Typ ‘tcpSocket‘ prüft, ob ein TCP-Handshake am angegebenen Port erfolgreich ist. Da kein Protokoll-Austausch erfolgt kann diese Variante auch für Dienste wie Redis, PostgreSQL oder eigene Binärprotokolle verwendet werden.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: liveness-example-tcp
spec:
  containers:
    - name: app
      image: your-image:latest
      livenessProbe:
        tcpSocket:
          port: 8080
        failureThreshold: 3
        initialDelaySeconds: 15
        periodSeconds: 10
```

Konfiguration für eine Liveness gRPC Probe

Implementiert die Applikation das gRPC Health Checking Protocol kann auch eine gRPC Probe verwendet werden.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: etcd-with-grpc
spec:
  containers:
    - name: etcd
      image: registry.k8s.io/etcd:3.5.1-0
      command:
        - /usr/local/bin/etcd
        - --data-dir
        - /var/lib/etcd
        - --listen-client-urls
        - http://0.0.0.0:2379
        - --advertise-client-urls
        - http://127.0.0.1:2379
        - --log-level
        - debug
      ports:
        - containerPort: 2379
      livenessProbe:
        grpc:
          port: 2379
        initialDelaySeconds: 10
```

Dabei muss Folgendes beachtet werden:

- Der Port muss explizit gesetzt werden und es muss die Portnummer verwendet werden.
- Der ‘service‘-Parameter wird zur Unterscheidung zwischen Liveness und Readiness genutzt
- Ein gRPC-Endpunkt kann für mehrere Probe-Typen genutzt werden
- Es gibt keine Authentifizierung und keine Fehlermeldungscodes
- Ein Fehler jeglicher Art führt zu einem Fehlschlag
- Fehlkonfigurationen führen sofort zu einem Fehler
- Der Dienst muss auf der Pod-IP lauschen
- Falls das Feature-Gate ‘ExecProbeTimeout=false‘ ist ignoriert ‘grpc-health-probe‘ das ‘timeoutSeconds‘-Limit

6.4 Readiness Probe

Python Code für readyz-Endpunkt

Die Funktion readyz prüft ob die API bereit ist. Zum Beispiel könnte sie testen ob ein Zugriff auf eine Datenbank möglich ist. Ist dies der Fall wird auch hier der Statuscode 200 zurückgegeben. Ansonsten wird der Statuscode 503 „Service Unavailable“ zurückgegeben. Kubernetes akzeptiert alle Codes von 400 bis 599 als Fehlschlag.
Je nach Resultat wird die Node

```python
@app.get("/readyz", response_class=PlainTextResponse)
async def readyz():
    current_time: float = time.time()-start_time
    # Beispielhafte Prüfung (Platzhalter für echte Verfügbarkeitslogik)
    if current_time < 10:
        return PlainTextResponse(content=f"error: {current_time}", status_code=503)
    else:
        return PlainTextResponse(content="ok", status_code=200)
```

Konfiguration für Readiness Probe

Die Readiness Probe verwendet das gleiche Prinzip mit Endpunkt, initialDelay und periodSeconds, wie die Liveness Probe.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: readiness-example
spec:
  containers:
    - name: app
      image: your-image:latest
      readinessProbe:
        httpGet:
          path: /readyz
          port: 8080
        initialDelaySeconds: 10
        periodSeconds: 5
```

6.5 Startup Probe

Die Startup Probe funktioniert genauso wie eine Liveness Probe, mit dem Unterschied dass sie die Ausführung von Readiness und Lifeness Probes verhindert, bis der Start erfolgreich war.
Sie wird genauso wie eine Liveness Probe konfiguriert, es ist also üblich den '/livez' Endpunkt zu verwenden, jedoch können auch hier TCP oder gRPC verwendet werden.

Beispielkonfiguration einer Startup Probe

Diese Startup Probe ist so definiert dass alle 5 Sekunden der 'livez' Endpunkt aufgerufen wird. Nach 30 Fehlschlägen, also 150 Sekunden, gilt der Start als gescheitert.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: startup-example
spec:
  containers:
    - name: app
      image: your-image:latest
      startupProbe:
        httpGet:
          path: /livez
          port: 8080
        failureThreshold: 30
        periodSeconds: 5
```

6.6 Best Practices für Probes

- **Initiale Verzögerung konfigurieren**, um ausreichend Startzeit zu bieten.
- **Angemessene Timeout-Werte setzen**, um kurzzeitige Verzögerungen oder Lastspitzen nicht fälschlich als Fehler zu interpretieren (`timeoutSeconds` typischerweise 2-5 Sekunden).
- **Leichtgewichtige Endpunkte verwenden**, um den Overhead der Probe möglichst gering zu halten.
- **Readiness-Probes für Abhängigkeiten nutzen**, anstatt die Anwendung durch Liveness-Probes neu zu starten, wenn externe Dienste nicht erreichbar sind.
- **Auf Readiness-Probes nicht verzichten**, wenn die Anwendung komplexe Initialisierungsschritte durchläuft oder auf externe Komponenten angewiesen ist.
- **Startup-Probes bei langer Initialisierungszeit einsetzen**.
- **TCP-Probes gezielt einsetzen**, da sie lediglich die Port-Erreichbarkeit prüfen und keine Aussage über die tatsächliche Funktionsfähigkeit des Dienstes treffen.
- **gRPC-Probes nur verwenden**, wenn das gRPC Health Check Protocol implementiert ist. Andernfalls auf HTTP-Probes oder benutzerdefinierte Mechanismen ausweichen.
- **Parameter wie `failureThreshold` und `periodSeconds` anpassen**, um das Verhalten der Probe an die Stabilität und Antwortzeit des Dienstes anzupassen.
- **Probes in Testumgebungen validieren**, um Fehlkonfigurationen und unerwartetes Verhalten frühzeitig zu erkennen.

Nützliche Links und Ressourcen

Kubernetes Dokumentation:
https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/
mdn web docs: https://developer.mozilla.org/de/docs/Web/HTTP/Reference/Status
FastAPI: https://fastapi.tiangolo.com/

7 Sicherheit und Zugriffskontrolle

7.1 Roles und RoleBindings

Ermöglicht die Verwaltung von Berechtigungen innerhalb von Namespaces. Roles definieren die Berechtigungen für Ressourcen innerhalb eines Namespaces, während RoleBindings diese Rollen an Benutzer, Gruppen oder ServiceAccounts binden.

Befehl	Beschreibung
`kubectl get roles`	Alle Roles im aktuellen Namespace auflisten
`kubectl get rolebindings`	Alle RoleBindings im aktuellen Namespace auflisten
`kubectl describe role <role-name>`	Details zu einer Role anzeigen
`kubectl describe rolebinding <rolebinding-name>`	Details zu einer RoleBinding anzeigen
`kubectl create -f role.yaml`	Rolle erstellen
`kubectl apply -f role.yaml`	Rolle erstellen oder aktualisieren
`kubectl create -f rolebinding.yaml`	RoleBinding erstellen
`kubectl apply -f rolebinding.yaml`	RoleBinding erstellen, oder aktualisieren
`kubectl delete role <role-name>`	Rolle löschen
`kubectl delete rolebinding <rolebinding-name>`	RoleBinding löschen

Beispielkonfiguration für eine Role

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: "default"
  name: "beispiel-role"
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get","list","watch"]
```

Beispielkonfiguration für ein RoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: beispiel-rolebinding
  namespace: default
subjects:
  - kind: User
    name: example-user
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: beispiel-role
  apiGroup: rbac.authorization.k8s.io
```

7.1.1 Anwendungsfälle für Roles und RoleBindings

- Beschränken des Zugriffs auf bestimmte Ressourcen innerhalb eines Namespaces.
- Delegieren von Verwaltungsaufgaben an bestimmte Benutzer oder ServiceAccounts.
- Implementieren von Least Privilege Prinzipien, um die Sicherheit zu erhöhen.

7.1.2 Best Practices für Roles und RoleBindings

- spezifische Rollen mit minimalen Berechtigungen verwenden, um das Prinzip der minimalen Rechte zu gewährleisten.
- alle RoleBindings und deren Zweck dokumentieren, um die Verwaltung zu erleichtern.
- regelmäßig die vergebenen Berechtigungen überwachen und überprüfen, um sicherzustellen, dass sie noch notwendig sind.
- Gruppen nutzen, um Berechtigungen effizienter zu verwalten.

7.1.3 Weitere nützliche Befehle für Roles und RoleBindings

Befehl	Beschreibung
`kubectl get roles -o yaml`	YAML-Konfiguration aller Roles im aktuellen Namespace anzeigen
`kubectl get rolebindings -o yaml`	YAML-Konfiguration aller RoleBindings im aktuellen Namespace anzeigen
`kubectl auth can-i <verb> <resource>`	Überprüfen, ob der aktuelle Benutzer eine bestimmte Aktion auf einer Ressource ausführen kann
`kubectl create role <role-name> –verb=<verb> –resource=<resource>`	Eine Role mit einem bestimmten Verb und einer Ressource erstellen
`kubectl create rolebinding <rolebinding-name> –role=<role-name> –user=<user-name>`	Eine RoleBinding für eine bestimmte Role und einen Benutzer erstellen

Nützliche Links und Ressourcen

- Kubernetes RBAC Dokumentation
- Kubernetes Task: Configuring Service Accounts for Pods
- Kubernetes Blog: Using RBAC for Kubernetes Authorization

7.2 ClusterRoles und ClusterRoleBindings

Ermöglicht die Verwaltung von Cluster-weiten Berechtigungen. ClusterRoles definieren die Berechtigungen für Ressourcen im gesamten Cluster, während ClusterRoleBindings diese Rollen an Benutzer, Gruppen oder ServiceAccounts im gesamten Cluster binden.

Befehl	Beschreibung
`kubectl get clusterroles`	Alle ClusterRoles auflisten
`kubectl get clusterrolebindings`	Alle ClusterRoleBindings auflisten
`kubectl describe clusterrole <clusterrole-name>`	Details zu einer ClusterRole anzeigen
`kubectl describe clusterrolebinding <clusterrolebinding-name>`	Details zu einer ClusterRoleBinding anzeigen
`kubectl create -f clusterrole.yaml`	ClusterRole erstellen
`kubectl apply -f clusterrole.yaml`	ClusterRole erstellen, oder aktualisieren
`kubectl create -f clusterrolebinding.yaml`	ClusterRoleBinding erstellen
`kubectl apply -f clusterrolebinding.yaml`	ClusterRoleBinding erstellen oder aktualisieren
`kubectl delete clusterrole <clusterrole-name>`	Eine ClusterRole löschen
`kubectl delete clusterrolebinding <clusterrolebinding-name>`	Eine ClusterRoleBinding löschen

Beispielkonfiguration für eine ClusterRole

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: beispiel-clusterrole
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch"]
```

Beispielkonfiguration für ein ClusterRoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: beispiel-clusterrolebinding
subjects:
  - kind: User
    name: example-user
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: beispiel-clusterrole
  apiGroup: rbac.authorization.k8s.io
```

7.2.1 Anwendungsfälle für ClusterRoles und ClusterRoleBindings

- Zuweisen von administrativen Berechtigungen für Cluster-weite Operationen.
- Delegieren von Berechtigungen für spezifische Cluster-weite Aufgaben an bestimmte Benutzer oder ServiceAccounts.
- Implementieren von Sicherheitsrichtlinien auf Cluster-Ebene.

7.2.2 Best Practices für ClusterRoles und ClusterRoleBindings

- spezifische Rollen mit minimalen Berechtigungen verwenden, um das Prinzip der minimalen Rechte zu gewährleisten.

- alle ClusterRoleBindings und deren Zweck dokumentieren, um die Verwaltung zu erleichtern.
- regelmäßig die vergebenen Berechtigungen überwachen und überprüfen, um sicherzustellen, dass sie noch notwendig sind.
- Gruppen nutzen, um Berechtigungen effizienter zu verwalten.
- Namenskonventionen für ClusterRoles und ClusterRoleBindings verwenden, um ihre Zwecke klar zu kommunizieren.

7.2.3 Weitere nützliche Befehle für ClusterRoles und ClusterRoleBindings

Befehl	Beschreibung
`kubectl get clusterroles -o yaml`	YAML-Konfiguration aller ClusterRoles anzeigen
`kubectl get clusterrolebindings -o yaml`	YAML-Konfiguration aller ClusterRoleBindings anzeigen
`kubectl auth can-i <verb> <resource> –all-namespaces`	Überprüfen, ob der aktuelle Benutzer eine bestimmte Aktion auf einer Ressource in allen Namespaces ausführen kann
`kubectl create clusterrole <clusterrole-name> –verb=<verb> –resource=<resource>`	Eine ClusterRole mit einem bestimmten Verb und einer Ressource erstellen
`kubectl create clusterrolebinding <clusterrolebinding-name> –clusterrole=<clusterrole-name> –user=<user-name>`	Eine ClusterRoleBinding für eine bestimmte ClusterRole und einen Benutzer erstellen

Nützliche Links und Ressourcen

- Kubernetes RBAC Dokumentation
- Kubernetes Task: Configuring Service Accounts for Pods
- Kubernetes Blog: Using RBAC for Kubernetes Authorization

7.3 ServiceAccounts

ServiceAccounts werden verwendet, um Pods mit bestimmten Berechtigungen auszustatten. Sie ermöglichen Pods den Zugriff auf den Kubernetes API-Server und andere Ressourcen unter Verwendung von Role-Based Access Control (RBAC).

Befehl	Beschreibung
`kubectl get serviceaccounts`	Alle ServiceAccounts im aktuellen Namespace auflisten
`kubectl describe serviceaccount <serviceaccount-name>`	Details zu einem ServiceAccount anzeigen
`kubectl create serviceaccount <serviceaccount-name>`	Einen neuen ServiceAccount erstellen
`kubectl apply -f serviceaccount.yaml`	Serviceaccount aus Datei erstellen, oder aktualisieren
`kubectl delete serviceaccount <serviceaccount-name>`	Einen ServiceAccount löschen

Beispielkonfiguration für einen ServiceAccount

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: beispiel-serviceaccount
  namespace: default
```

7.3.1 Anwendungsfälle für ServiceAccounts

- Ermöglichen von Pods, sich gegenüber dem Kubernetes API-Server zu authentifizieren.
- Zuweisen spezifischer Berechtigungen zu Pods mittels RBAC.
- Isolieren von Pods und deren Berechtigungen, um das Prinzip der minimalen Rechte zu gewährleisten.
- Implementieren von Sicherheitsrichtlinien, um den Zugriff von Pods auf Cluster-Ressourcen zu steuern.

7.3.2 Best Practices für ServiceAccounts

- Spezifische ServiceAccounts für verschiedene Anwendungen oder Pods erstellen, anstatt den Standard-ServiceAccount zu verwenden.
- RoleBindings oder ClusterRoleBindings verwenden, um die Berechtigungen für ServiceAccounts zu steuern.
- Regelmäßig die vergebenen Berechtigungen überprüfen, um sicherzustellen, dass sie noch notwendig sind.
- Alle ServiceAccounts und deren Zweck dokumentieren, um die Verwaltung zu erleichtern.
- Namenskonventionen für ServiceAccounts nutzen, um ihre Zwecke klar zu kommunizieren.

7.3.3 Weitere nützliche Befehle für ServiceAccounts

Befehl	**Beschreibung**
`kubectl get serviceaccounts -o yaml`	YAML-Konfiguration aller ServiceAccounts im aktuellen Namespace anzeigen
`kubectl patch serviceaccount <serviceaccount-name> -p <patch-data>`	Einen Patch auf einen ServiceAccount anwenden
`kubectl get secret <secret-name> -o yaml`	Das Secret eines ServiceAccounts anzeigen
`kubectl describe secret <secret-name>`	Details zu einem Secret eines ServiceAccounts anzeigen
`kubectl create rolebinding <rolebinding-name> –role=<role-name> –serviceaccount=<namespace>:<serviceaccount-name>`	Eine RoleBinding für einen bestimmten ServiceAccount erstellen
`kubectl create clusterrolebinding <clusterrolebinding-name> –clusterrole=<clusterrole-name> –serviceaccount=<namespace>:<serviceaccount-name>`	Eine ClusterRoleBinding für einen bestimmten ServiceAccount erstellen

7.3.4 Komplettbeispiel

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: rbac-demo
```

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: demo-sa
  namespace: rbac-demo
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: rbac-demo
  name: pod-reader
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch"]
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: rbac-demo
subjects:
  - kind: ServiceAccount
    name: demo-sa
    namespace: rbac-demo
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-uses-sa
  namespace: rbac-demo
spec:
  serviceAccountName: demo-sa
  containers:
    - name: curl-container
      image: curlimages/curl:latest
      command: ["sleep", "3600"]
```

Nützliche Links und Ressourcen

- Kubernetes Task: Configuring Service Accounts for Pods
- Kubernetes ServiceAccounts Administration
- Kubernetes Blog: Security Best Practices for Kubernetes Deployment

7.4 Network Policies

Network Policies ermöglichen die Kontrolle des Datenverkehrs zwischen Pods und anderen Netzwerkendpunkten auf Basis von Label-Selektoren und Regeln. Sie bieten eine Möglichkeit, den ein- und ausgehenden Datenverkehr zu regulieren, um die Sicherheit und Isolation im Cluster zu erhöhen.

Befehl	Beschreibung
`kubectl get networkpolicies`	Alle Network Policies im aktuellen Namespace auflisten
`kubectl describe networkpolicy <policy-name>`	Details zu einer bestimmten Network Policy anzeigen
`kubectl create -f <networkpolicy.yaml>`	Eine neue Network Policy anhand einer YAML-Datei erstellen
`kubectl apply -f <networkpolicy.yaml>`	Eine Network Policy erstellen, oder aktualisieren
`kubectl delete networkpolicy <policy-name>`	Eine Network Policy löschen
`kubectl edit networkpolicy <policy-name>`	Eine Network Policy im Editor bearbeiten
`kubectl get networkpolicy <policy-name> -o yaml`	Eine Network Policy im YAML-Format anzeigen

Beispielkonfiguration für eine Network Policy

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: beispiel-networkpolicy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: frontend
      ports:
        - protocol: TCP
          port: 3306
  egress:
    - to:
        - podSelector:
            matchLabels:
              role: backend
      ports:
        - protocol: TCP
          port: 3306
```

7.4.1 Anwendungsfälle für Network Policies

- Isolieren von Pods, um sicherzustellen, dass nur autorisierter Datenverkehr zugelassen wird.
- Schutz sensibler Datenbanken, indem der Zugriff nur auf spezifische Pods beschränkt wird.
- Implementieren von Sicherheitsrichtlinien, um unautorisierten Datenverkehr zu verhindern.
- Erzwingen von Kommunikationsmustern zwischen verschiedenen Anwendungskomponenten.

7.4.2 Best Practices für Network Policies

- Beginne mit restriktiven Policies und erweitere schrittweise die erlaubten Verbindungen.
- Nutze Label-Selektoren, um granularere Kontrollen über den Datenverkehr zu erlangen.
- Überprüfe und teste regelmäßig die Network Policies, um sicherzustellen, dass sie die gewünschten Sicherheitsanforderungen erfüllen.
- Dokumentiere alle Network Policies und deren Zweck, um die Verwaltung zu erleichtern.
- Nutze Namespaces, um Network Policies auf bestimmte Bereiche des Clusters zu beschränken.

7.4.3 Weitere nützliche Befehle für Network Policies

Befehl	Beschreibung
`kubectl get networkpolicies -o yaml`	YAML-Konfiguration aller Network Policies im aktuellen Namespace anzeigen
`kubectl patch networkpolicy <policy-name> -p <patch-data>`	Einen Patch auf eine Network Policy anwenden
`kubectl label pod <pod-name> <label-key>=<label-value>`	Ein Label zu einem Pod hinzufügen, um die Network Policy anzuwenden
`kubectl annotate networkpolicy <policy-name> <annotation-key>=<annotation-value>`	Eine Annotation zu einer Network Policy hinzufügen
`kubectl get pods -selector=<label-selector>`	Alle Pods mit einem bestimmten Label-Selektor auflisten

7.4.4 Nützliche Links und Ressourcen

- Kubernetes Network Policies Dokumentation
- Kubernetes Task: Declare Network Policy
- Kubernetes Network Policy Recipes auf GitHub

8 Ressourcenverwaltung

8.1 Resource Quotas und LimitRanges

Ermöglicht es, Ressourcenlimits und -quoten für Namespaces festzulegen. Resource Quotas begrenzen die Gesamtmenge an Ressourcen, die von einem Namespace verbraucht werden können, während LimitRanges Mindest- und Höchstwerte für die Ressourcen festlegen, die einzelne Container oder Pods verwenden können.

Befehl	Beschreibung
`kubectl get resourcequotas`	Alle ResourceQuotas im aktuellen Namespace auflisten
`kubectl describe resourcequota <resourcequota-name>`	Details zu einer ResourceQuota anzeigen
`kubectl delete resourcequota <resourcequota-name>`	Eine ResourceQuota löschen
`kubectl get limitranges`	Alle LimitRanges im aktuellen Namespace auflisten
`kubectl create -f <resourcequota.yaml>`	eine Resource Quota erstellen
`kubectl create -f <limitrange.yaml>`	Eine LimitRange erstellen
`kubectl apply -f <resourcequota.yaml>`	eine Resource Quota erstellen oder updaten
`kubectl apply -f <limitrage.yaml>`	eine LimitRange erstellen oder updaten
`kubectl describe limitrange <limitrange-name>`	Details zu einer LimitRange anzeigen
`kubectl delete limitrange <limitrange-name>`	Eine LimitRange löschen

YAML-Datei für eine ResourceQuota

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: beispiel-resourcequota
  namespace: default
spec:
  hard:
    pods: "10"
    requests.cpu: "4"
    requests.memory: 8Gi
    limits.cpu: "8"
    limits.memory: 16Gi
```

YAML-Datei für eine LimitRange

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: beispiel-limitrange
  namespace: default
spec:
  limits:
    - max:
        cpu: "2"
        memory: 4Gi
      min:
        cpu: 200m
        memory: 6Mi
      default:
        cpu: "1"
        memory: 2Gi
      defaultRequest:
        cpu: 500m
        memory: 1Gi
      type: Container
```

8.1.1 Anwendungsfälle für Resource Quotas und LimitRanges

- Sicherstellen, dass ein Namespace nicht mehr Ressourcen als vorgesehen verbraucht.
- Verhindern, dass einzelne Pods oder Container zu viele Ressourcen beanspruchen.
- Ermöglichen einer fairen Ressourcenverteilung zwischen verschiedenen Teams oder Projekten.
- Schutz des Clusters vor Ressourcenengpässen durch übermäßige Nutzung.

8.1.2 Best Practices für Resource Quotas und LimitRanges

- Setze realistische Quoten und Limits basierend auf der tatsächlichen Nutzung und den Anforderungen der Anwendungen.
- Überwache regelmäßig die Ressourcennutzung, um sicherzustellen, dass die Quoten und Limits angemessen sind.
- Kommuniziere die Quoten und Limits klar an alle Teams, um Verständnis und Akzeptanz zu fördern.
- Nutze Namespaces, um verschiedene Quoten und Limits für unterschiedliche Teams oder Projekte festzulegen.
- Passe die Quoten und Limits bei Änderungen der Anforderungen oder der Clusterkapazität an.

8.1.3 Weitere nützliche Befehle für Resource Quotas und LimitRanges

Befehl	Beschreibung
`kubectl get resourcequotas -o yaml`	YAML-Konfiguration aller ResourceQuotas im aktuellen Namespace anzeigen
`kubectl get limitranges -o yaml`	YAML-Konfiguration aller LimitRanges im aktuellen Namespace anzeigen
`kubectl patch resourcequota <resourcequota-name> -p <patch-data>`	Einen Patch auf eine ResourceQuota anwenden
`kubectl patch limitrange <limitrange-name> -p <patch-data>`	Einen Patch auf eine LimitRange anwenden
`kubectl edit resourcequota <resourcequota-name>`	Eine ResourceQuota im Editor bearbeiten
`kubectl edit limitrange <limitrange-name>`	Eine LimitRange im Editor bearbeiten
`kubectl get pods –namespace=<namespace> –field-selector=status.phase=Failed`	Alle fehlgeschlagenen Pods in einem Namespace auflisten, um Ressourcenlecks zu identifizieren

Nützliche Links und Ressourcen

- Kubernetes Resource Quotas Dokumentation
- Kubernetes Task: Administering Resource Quotas
- Kubernetes Limit Range Dokumentation
- Kubernetes Task: Managing Compute Resources for Containers

8.2 PodDisruptionBudgets (PDB)

PodDisruptionBudgets (PDB) werden verwendet, um die Anzahl der gleichzeitigen Pod-Ausfälle zu begrenzen. Sie helfen dabei, die Verfügbarkeit von Anwendungen zu gewährleisten, indem sie sicherstellen, dass eine Mindestanzahl von Pods immer verfügbar bleibt, selbst während Wartungsarbeiten oder Upgrades.

Befehl	Beschreibung
`kubectl get pdb`	Alle PodDisruptionBudgets im aktuellen Namespace auflisten
`kubectl describe pdb <pdb-name>`	Details zu einem PodDisruptionBudget anzeigen
`kubectl create -f <pdb.yaml>`	PodDisruptionBudget erstellen
`kubectl apply -f <pdb.yaml>`	PodDisruptionBudget erstellen oder updaten
`kubectl delete pdb <pdb-name>`	Ein PodDisruptionBudget löschen

Beispielkonfiguration für ein PodDisruptionBudget

```yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: beispiel-pdb
  namespace: default
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: beispiel-app
```

8.2.1 Anwendungsfälle für PodDisruptionBudgets

- Sicherstellen, dass eine Mindestanzahl von Pods während geplanter Wartungsarbeiten verfügbar bleibt.
- Schutz kritischer Anwendungen vor zu vielen gleichzeitigen Pod-Ausfällen.
- Verbesserung der Anwendungsverfügbarkeit während automatisierter Upgrades und Rollouts.
- Unterstützung bei der Einhaltung von Service Level Agreements (SLAs) durch Gewährleistung der Verfügbarkeit.

8.2.2 Best Practices für PodDisruptionBudgets

- Setze realistische Werte für `minAvailable` oder `maxUnavailable`, um die Verfügbarkeit von Anwendungen zu gewährleisten.
- Überprüfe regelmäßig die Konfiguration der PDBs, um sicherzustellen, dass sie den aktuellen Anforderungen entsprechen.
- Kombiniere PDBs mit anderen Hochverfügbarkeitsstrategien wie ReplicaSets und Deployments.
- Dokumentiere alle PDBs und deren Zweck, um die Verwaltung zu erleichtern.
- Teste die Auswirkungen von PDBs in einer Staging-Umgebung, bevor sie in der Produktion verwendet werden.

8.2.3 Weitere nützliche Befehle für PodDisruptionBudgets

Befehl	Beschreibung
`kubectl get pdb -o yaml`	YAML-Konfiguration aller PodDisruptionBudgets im aktuellen Namespace anzeigen
`kubectl edit pdb <pdb-name>`	Ein PodDisruptionBudget im Editor bearbeiten
`kubectl patch pdb <pdb-name> -p <patch-data>`	Einen Patch auf ein PodDisruptionBudget anwenden
`kubectl get pods –selector=<label-selector>`	Alle Pods auflisten, die von einem bestimmten PDB betroffen sind

8.2.4 Nützliche Links und Ressourcen

- Kubernetes PodDisruptionBudgets Dokumentation
- Kubernetes Task: Configure PodDisruptionBudget

9 Erweiterungen und Anpassungen

9.1 Plugins für Kubectl

Kubectl-Plugins erweitern die Funktionalität des Standard-Kubectl-Befehlszeilenwerkzeugs. Sie ermöglichen benutzerdefinierte Befehle und Automatisierungen, die speziell auf die Bedürfnisse eines Clusters oder eines Teams zugeschnitten sind.

Befehl	Beschreibung
`kubectl plugin list`	Alle installierten Kubectl-Plugins auflisten
`kubectl plugin <plugin-name>`	Einen spezifischen Plugin-Befehl ausführen
`kubectl krew install <plugin-name>`	Ein Plugin mithilfe von Krew installieren
`kubectl krew uninstall <plugin-name>`	Ein Plugin mithilfe von Krew deinstallieren
`kubectl krew list`	Alle Plugins anzeigen, die mit Krew installiert wurden
`kubectl krew update`	Die Krew-Plugin-Datenbank aktualisieren
`kubectl krew search <plugin-name>`	Nach Plugins in der Krew-Datenbank suchen

9.1.1 Krew: Der Plugin-Manager für Kubectl

Krew ist ein Plugin-Manager für Kubectl, der die Installation und Verwaltung von Plugins vereinfacht. Mit Krew können Plugins aus einer zentralen Plugin-Datenbank durchsucht, installiert, aktualisiert und deinstalliert werden.

Installation von Krew

Um Krew zu installieren, kann das folgende Skript verwendet werden:

```
(
  set -x; cd "$(mktemp -d)" &&
  OS="$(uname | tr '[:upper:]' '[:lower:]')" &&
  ARCH="$(uname -m |

 sed -e 's/x86_64/amd64/' -e 's/arm.*$/arm/')" &&
  KREW="krew-${OS}_${ARCH}" &&
  curl -fsSLO "https://github.com/kubernetes-sigs/krew/releases/latest/ download/${KREW}.tar.gz" &&
  tar zxvf "${KREW}.tar.gz" &&
  ./"${KREW}" install krew
)
export PATH="${KREW_ROOT:-$HOME/.krew}/bin:$PATH"
```

Verwalten von Plugins mit Krew

Mit Krew können verschiedene Verwaltungsaufgaben für Plugins durchgeführt werden:

```
# Nach einem Plugin suchen
kubectl krew search <plugin-name>
# Ein Plugin installieren
kubectl krew install <plugin-name>
# Ein Plugin deinstallieren
kubectl krew uninstall <plugin-name>
# Alle installierten Plugins auflisten
kubectl krew list
# Die Krew-Plugin-Datenbank aktualisieren
kubectl krew update
```

9.1.2 Beliebte Kubectl-Plugins

Hier sind einige beliebte Kubectl-Plugins, die häufig verwendet werden:

- `kubectl neat`: Entfernt unnötige Felder aus der Ausgabe von Kubectl-Befehlen
- `kubectl ctx`: Ermöglicht das schnelle Wechseln zwischen Kubernetes-Kontexten
- `kubectl ns`: Ermöglicht das schnelle Wechseln zwischen Namespaces
- `kubectl tree`: Zeigt eine hierarchische Ansicht von Kubernetes-Ressourcen
- `kubectl view-secret`: Zeigt den Inhalt von Secrets in Klartext an

9.1.3 Erstellen eigener Kubectl-Plugins

Eigene Kubectl-Plugins können erstellt werden, um spezifische Anforderungen zu erfüllen. Ein Kubectl-Plugin ist im Grunde ein ausführbares Skript oder Programm, das im Pfad verfügbar ist und mit `kubectl-<plugin-name>` benannt ist.

Hier ist ein Beispiel für ein einfaches Kubectl-Plugin, das in Bash geschrieben ist und eine Liste der Nodes im Cluster ausgibt:

```
#!/bin/bash

# Datei als kubectl-nodes.sh speichern und ausführbar machen
kubectl get nodes -o custom-columns=NAME:.metadata.name,STATUS:.status.conditions [-1].type
```

Um dieses Skript als Kubectl-Plugin zu verwenden, sind folgende Schritte notwendig:

1. Das Skript als `kubectl-nodes` speichern.
2. Die Datei ausführbar machen:
   `chmod +x kubectl-nodes`
3. Das Skript in ein Verzeichnis legen, das im `PATH` enthalten ist, oder den `PATH` entsprechend anpassen:
   `export PATH=$PATH:/pfad/zum/skript`

Nach diesen Schritten kann das Plugin mit folgendem Befehl ausgeführt werden:
`kubectl nodes`

9.1.4 Sicherheitsaspekte bei der Nutzung von Plugins

Beim Einsatz von Kubectl-Plugins sind einige Sicherheitsaspekte zu beachten:

- Nur Plugins aus vertrauenswürdigen Quellen installieren, um das Risiko von Malware oder bösartigem Code zu minimieren.
- Die Skripte und ausführbaren Dateien der Plugins regelmäßig überprüfen, um sicherzustellen, dass sie keine unerwünschten Änderungen enthalten.
- Die Berechtigungen der Plugin-Skripte und -Dateien streng kontrollieren, um unbefugten Zugriff zu verhindern.
- Sicherheitsupdates für installierte Plugins beachten und diese regelmäßig aktualisieren.

9.2 Custom Resource Definitions (CRDs)

Custom Resource Definitions (CRDs) ermöglichen die Erweiterung von Kubernetes um benutzerdefinierte Ressourcen. Sie sind eine Möglichkeit, um eigene API-Objekte zu definieren und zu verwalten, die über die Standard-Ressourcen von Kubernetes hinausgehen.

Befehl	Beschreibung
`kubectl get crds`	Alle Custom Resource Definitions auflisten
`kubectl describe crd <crd-name>`	Details zu einer Custom Resource Definition anzeigen
`kubectl create -f <crd.yaml>`	Custom Resource Definition erstellen
`kubectl apply -f <crd.yaml>`	Custom Resource Definition erstellen, oder updaten
`kubectl delete crd <crd-name>`	Eine Custom Resource Definition löschen

YAML-Datei für eine Custom Resource Definition

```yaml
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: beispiel-crd.example.com
spec:
  group: example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                foo:
                  type: string
  scope: Namespaced
  names:
    plural: beispiel-crds
    singular: beispiel-crd
    kind: BeispielCRD
    shortNames:
      - bcrd
```

9.2.1 Anwendungsfälle für Custom Resource Definitions

- Definieren und Verwalten benutzerdefinierter Ressourcen, die spezifische Anforderungen erfüllen.
- Erweiterung der Kubernetes-API um projekt- oder domänenspezifische Funktionalitäten.
- Automatisierung und Verwaltung komplexer Anwendungslogik und -konfigurationen.
- Integration von Drittanbieter-Tools und -Services in Kubernetes.

9.2.2 Best Practices für Custom Resource Definitions

- Vermeide Namenskonflikte durch die Verwendung eindeutiger Gruppen- und Ressourcennamen.
- Verwende Versionskontrolle für CRDs, um die Abwärtskompatibilität sicherzustellen.
- Dokumentiere die Struktur und das Schema der CRDs klar und umfassend.
- Überprüfe und teste CRDs gründlich, bevor sie in der Produktion verwendet werden.
- Verwende Validierungs- und Konvertierungs-Webhook-Server, um die Konsistenz und Integrität der benutzerdefinierten Ressourcen sicherzustellen.

9.2.3 Weitere nützliche Befehle für Custom Resource Definitions

Befehl	Beschreibung
`kubectl get crd <crd-name> -o yaml`	YAML-Konfiguration einer Custom Resource Definition anzeigen
`kubectl edit crd <crd-name>`	Eine Custom Resource Definition im Editor bearbeiten
`kubectl patch crd <crd-name> -p <patch-data>`	Einen Patch auf eine Custom Resource Definition anwenden
`kubectl get <crd-name>`	Alle benutzerdefinierten Ressourcen für eine bestimmte CRD auflisten
`kubectl describe <crd-name> <resource-name>`	Details zu einer bestimmten benutzerdefinierten Ressource anzeigen
`kubectl delete <crd-name> <resource-name>`	Eine benutzerdefinierte Ressource löschen

Nützliche Links und Ressourcen

- Kubernetes Custom Resource Definitions Dokumentation
- Kubernetes Konzepte: Custom Resources
- Beispiel-Controller für Custom Resources auf GitHub

9.2.4 Erstellen und Verwalten einer benutzerdefinierten Ressource

Beispiel, bei dem eine benutzerdefinierte Ressource namens `BeispielResource` erstellt und verwaltet wird.

YAML-Datei für die benutzerdefinierte Ressource

```yaml
apiVersion: example.com/v1
kind: BeispielResource
metadata:
  name: beispiel-resource-instance
  namespace: default
spec:
  foo: bar
```

Erstellen einer benutzerdefinierten Ressource aus einer YAML-Datei

```
kubectl apply -f beispiel-resource-instance.yaml
```

Auflisten und Anzeigen der benutzerdefinierten Ressourcen

```
kubectl get beispielresources
kubectl describe beispielresource beispiel-resource-instance
```

Löschen einer benutzerdefinierten Ressource

```
kubectl delete beispielresource beispiel-resource-instance
```

9.2.5 Controller für Custom Resources

Um die Funktionalität von Custom Resources vollständig zu nutzen, kann ein benutzerdefinierter Controller entwickelt werden, der die Lebenszyklen der benutzerdefinierten Ressourcen verwaltet. Ein Controller überwacht die API-Server nach Änderungen an der benutzerdefinierten Ressource und führt entsprechende Aktionen aus.

Beispiel-Controller für `BeispielResource`

Ein Beispiel-Controller kann in Go geschrieben werden und verwendet das client-go Paket von Kubernetes. Der Grundlegende Code für einen solchen Controller würde wie folgt aussehen:

```go
// Reduziertes Beispiel eines Go-basierten Controllers mit client-go
package main

import (
    "context"
    "fmt"
    "time"

    "k8s.io/client-go/kubernetes"
    "k8s.io/client-go/rest"
    "k8s.io/client-go/tools/cache"
)

func main() {
    config, err := rest.InClusterConfig()
    if err != nil {
        panic(err.Error())
    }
    clientset, err := kubernetes.NewForConfig(config)
    if err != nil {
        panic(err.Error())
    }

    exampleResourceInformer := cache.NewSharedInformer(
        // Informer Konfiguration für BeispielResource
    )

    stopCh := make(chan struct{})
    defer close(stopCh)

    go exampleResourceInformer.Run(stopCh)

    for {
        select {
        case <-stopCh:
            fmt.Println("Stopping controller")
            return
        default:
            fmt.Println("Controller running")
            time.Sleep(10 * time.Second)
        }
    }
}
```

Der komplette Code für den funktionstüchtigen Controller sieht wie folgt aus:

```go
package main

import (
    "fmt"
    "os"
    "time"

    "k8s.io/client-go/informers"
    "k8s.io/client-go/kubernetes"
    "k8s.io/client-go/tools/cache"
    "k8s.io/client-go/tools/clientcmd"
)

func main() {
    // Laden der Kubernetes-Konfiguration (innerhalb oder außerhalb des Clusters)
    config, err := clientcmd.BuildConfigFromFlags("", clientcmd.RecommendedHomeFile)
    if err ≠ nil {
        fmt.Fprintf(os.Stderr, "Error building kubeconfig: %s\n", err.Error())
        os.Exit(1)
    }
    // Erstellen des Clientsets
    clientset, err := kubernetes.NewForConfig(config)
    if err ≠ nil {
        fmt.Fprintf(os.Stderr, "Error creating clientset: %s\n", err.Error())
        os.Exit(1)
    }
    // Erstellen eines Factory für SharedInformers
    factory := informers.NewSharedInformerFactory(clientset, 10*time.Minute)
    // Informer für Pods erstellen
    podInformer := factory.Core().V1().Pods().Informer()
    stopCh := make(chan struct{})
    defer close(stopCh)
    // Event-Handler für den Pod-Informer hinzufügen
    podInformer.AddEventHandler(cache.ResourceEventHandlerFuncs{
        AddFunc: func(obj interface{}) {
            fmt.Println("Pod added:", obj)
        },
        UpdateFunc: func(oldObj, newObj interface{}) {
            fmt.Println("Pod updated:", newObj)
        },
        DeleteFunc: func(obj interface{}) {
            fmt.Println("Pod deleted:", obj)
        },
    })
    // Informer starten
    go podInformer.Run(stopCh)
    // Warten auf Synchronisierung der Caches
    if !cache.WaitForCacheSync(stopCh, podInformer.HasSynced) {
        fmt.Fprintf(os.Stderr, "Error waiting for cache sync\n")
        os.Exit(1)
    }
    fmt.Println("Controller running")
    select {}
}

```

9.3 Helm

Helm ist ein Paketmanager für Kubernetes, der das Verwalten von Kubernetes-Anwendungen vereinfacht. Mit Helm können Anwendungen als Charts (Pakete mit vorkonfigurierten Kubernetes-Ressourcen) installiert, aktualisiert und verwaltet werden.

Befehl	Beschreibung
`helm list`	Alle installierten Helm-Releases auflisten
`helm install <release-name> <chart>`	Ein neues Helm-Chart installieren
`helm upgrade <release-name> <chart>`	Ein Helm-Release aktualisieren
`helm delete <release-name>`	Ein Helm-Release löschen
`helm repo add <repo-name> <repo-url>`	Ein Helm-Repository hinzufügen
`helm repo update`	Alle Helm-Repositories aktualisieren

Helm Installieren

```
sudo snap install helm –classic
```

Helm-Chart-Struktur

Ein Helm-Chart ist ein Paket, das alle notwendigen Konfigurationsdateien enthält, um eine Kubernetes-Anwendung zu deployen. Die typische Struktur eines Helm-Charts sieht wie folgt aus:

```
mychart
  ├── Chart.yaml        # Metadaten über das Chart
  ├── values.yaml       # Standardwerte für die Konfiguration
  ├── charts            # Abhängige Charts
  └── templates         # Kubernetes-Manifestdateien
      └── _helpers.tpl  # Hilfsfunktionen
```

Erstellen eines neuen Helm-Charts

Um ein neues Helm-Chart zu erstellen, verwendet man den folgenden Befehl:

```
helm create mychart
```

Anpassen von Werten in einem Helm-Chart

Die Werte in einem Helm-Chart können durch Erstellen einer eigenen `values.yaml`-Datei überschrieben werden. Diese Datei enthält die benutzerdefinierten Konfigurationen für das Chart.

```yaml
replicaCount: 2
image:
  repository: myrepo/myimage
  tag: 1.0.0
  pullPolicy: IfNotPresent
service:
  type: LoadBalancer
  port: 80
```

Installieren eines Helm-Charts mit benutzerdefinierten Werten

Um ein Helm-Chart mit benutzerdefinierten Werten zu installieren, verwendet man den folgenden Befehl:

```
helm install <release-name> <chart> -f values.yaml
```

9.3.1 Anwendungsfälle für Helm

- Vereinfachung des Deployments und der Verwaltung von Kubernetes-Anwendungen.
- Bereitstellung und Aktualisierung von Anwendungen mit minimalem Aufwand.
- Verwaltung von Abhängigkeiten zwischen verschiedenen Kubernetes-Ressourcen.
- Wiederverwendbarkeit und gemeinsame Nutzung von Konfigurationen und Best Practices.
- Erleichterung der Zusammenarbeit in Teams durch standardisierte Charts.

9.3.2 Best Practices für die Nutzung von Helm

- Versioniere Helm-Charts sorgfältig, um die Rückverfolgbarkeit und Reproduzierbarkeit zu gewährleisten.
- Verwende `values.yaml`-Dateien, um Konfigurationen zu abstrahieren und flexibel zu gestalten.
- Nutze Helm-Repositorys, um Charts zu teilen und wiederzuverwenden.
- Teste Charts gründlich in einer Staging-Umgebung, bevor sie in der Produktion verwendet werden.
- Dokumentiere die Verwendung und Konfiguration von Charts klar und umfassend.

9.3.3 Weitere nützliche Befehle für Helm

Befehl	Beschreibung
`helm search hub <keyword>`	Suche nach Charts im Helm Hub
`helm show values <chart>`	Zeige die Standardwerte eines Charts an
`helm show chart <chart>`	Zeige die Metadaten eines Charts an
`helm show readme <chart>`	Zeige die README-Datei eines Charts an
`helm dependency update`	Aktualisiere die Abhängigkeiten eines Charts
`helm rollback <release-name> <revision>`	Mache eine bestimmte Revision eines Releases rückgängig
`helm history <release-name>`	Zeige die Versionshistorie eines Releases an

9.3.4 Nützliche Links und Ressourcen

- Offizielle Helm-Dokumentation
- Artifact Hub für Helm-Charts
- Helm Blog für Neuigkeiten und Updates
- Helm GitHub Repository
- Best Practices für Helm-Charts

9.3.5 Erstellen und Verwalten eines Helm-Charts

Erstellen eines neuen Helm-Charts

```
helm create mychart
```

Installieren des Helm-Charts

```
helm install my-release mychart
```

Überprüfen des Helm-Releases

```
helm list
helm status my-release
```

Anpassen der `values.yaml`-Datei

```yaml
replicaCount: 3
image:
  repository: nginx
  tag: 1.19.2
  pullPolicy: IfNotPresent
service:
  type: ClusterIP
  port: 80
```

Aktualisieren des Helm-Releases

Nach bearbeitung der `values.yaml`-Datei:

```
helm upgrade my-release mychart -f values.yaml
```

Löschen des Helm-Releases

```
helm delete my-release
```

9.4 Operators

Operators sind Kubernetes-Controller, die speziell dafür entwickelt wurden, komplexe Anwendungen und Zustandsmaschinen zu verwalten. Sie nutzen benutzerdefinierte Ressourcen (Custom Resources), um Anwendungen und deren Betriebsaufgaben zu automatisieren. Operators erweitern die Fähigkeiten von Kubernetes, indem sie Anwendungslogik und Betriebswissen in den Cluster einbringen.

Befehl	**Beschreibung**
`kubectl get operators`	Alle Operators im Cluster auflisten (wenn CRD für Operators vorhanden)
`kubectl get crds`	Alle Custom Resource Definitions (CRDs) im Cluster auflisten
`kubectl get <custom-resource>`	Alle Instanzen einer bestimmten benutzerdefinierten Ressource auflisten
`kubectl describe <custom-resource> <resource-name>`	Details zu einer bestimmten benutzerdefinierten Ressource anzeigen
`kubectl create -f <custom-resource.yaml>`	Eine neue benutzerdefinierte Ressource anhand einer YAML-Datei erstellen
`kubectl delete <custom-resource> <resource-name>`	Eine bestimmte benutzerdefinierte Ressource löschen
`kubectl edit <custom-resource> <resource-name>`	Eine benutzerdefinierte Ressource im Editor bearbeiten
`kubectl get <custom-resource> <resource-name> -o yaml`	Eine benutzerdefinierte Ressource im YAML-Format anzeigen
`kubectl logs <operator-pod-name>`	Logs eines Operator-Pods anzeigen, um den Status und Fehler zu überprüfen
`kubectl get pods -l name=<operator-name>`	Alle Pods eines bestimmten Operators auflisten
`kubectl describe pod <operator-pod-name>`	Details zu einem spezifischen Operator-Pod anzeigen
`kubectl apply -f <operator-deployment.yaml>`	Einen Operator aus einer YAML-Datei bereitstellen
`kubectl delete -f <operator-deployment.yaml>`	Einen Operator entfernen, der aus einer YAML-Datei bereitgestellt wurde

9.4.1 Anwendungsfälle für Operators

- Automatisierung des Lebenszyklusmanagements von stateful Anwendungen wie Datenbanken und Message Queues.
- Implementierung von komplexen Geschäftslogiken und Betriebsabläufen als Kubernetes-Ressourcen.
- Vereinfachung der Verwaltung und Skalierung von Anwendungen durch die Automatisierung wiederkehrender Aufgaben.
- Sicherstellung der Einhaltung von Best Practices und Unternehmensrichtlinien durch codierte Betriebslogik.

9.4.2 Best Practices für die Entwicklung und Nutzung von Operators

- Beginne mit klar definierten Anwendungsanforderungen und identifiziere, welche Betriebsaufgaben automatisiert werden können.
- Verwende geeignete Frameworks wie `Operator SDK` oder `Kubebuilder` zur Entwicklung von Operators.
- Implementiere umfassende Tests für den Operator, um sicherzustellen, dass er unter verschiedenen Szenarien korrekt funktioniert.
- Dokumentiere die benutzerdefinierten Ressourcen und deren Nutzung klar und umfassend.
- Überwache und logge die Aktivitäten des Operators, um Probleme frühzeitig zu erkennen und zu beheben.
- Verwalte die Versionierung des Operators sorgfältig, um Kompatibilitätsprobleme zu vermeiden.

9.4.3 Frameworks und Tools zur Entwicklung von Operators

Es gibt verschiedene Frameworks und Tools, die die Entwicklung von Operators erleichtern:

- Operator SDK: Ein Framework zur schnellen Entwicklung von Kubernetes Operators in Go, Ansible oder Helm.
- Kubebuilder: Ein Framework zur Entwicklung von Kubernetes APIs und Controllers mit CRDs.
- Metacontroller: Ein Controller, der es ermöglicht, benutzerdefinierte Controller durch Konfiguration und Scripts zu erstellen.

9.4.4 Erstellen eines einfachen Operators mit Operator SDK

Anleitung für neueste Version

Installation des Operator SDK

```
curl -Lo operator-sdk https://github.com/operator-framework/operator-sdk/releases/
↪  download/v1.9.0/operator-sdk_linux_amd64
chmod +x operator-sdk
mv operator-sdk /usr/local/bin/
```

Erstellen eines neuen Operator-Projekts

```
operator-sdk init --domain=example.com --repo=github.com/example/my-operator
```

Definieren der benutzerdefinierten Ressource (CRD)

```
# Eine neue API und einen Controller erstellen
operator-sdk create api --group cache --version v1alpha1 --kind Memcached --resource --controller
```

Die Datei `api/v1alpha1/memcached_types.go` bearbeiten, um das Schema der benutzerdefinierten Ressource zu definieren:

```
// MemcachedSpec defines the desired state of Memcached
type MemcachedSpec struct {
    Size int32 `json:"size"`
}

// MemcachedStatus defines the observed state of Memcached
type MemcachedStatus struct {
    Nodes []string `json:"nodes"`
}
```

Implementieren des Controllers

Die Datei `controllers/memcached_controller.go` bearbeiten, um die Geschäftslogik des Controllers zu implementieren:

```
// Reconcile-Methode implementieren, um gewünschte und tatsächliche Zustände abzugleichen
func (r *MemcachedReconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result, error) {
    // Logik zum Abgleich des Zustands
}
```

Erstellen und Anwenden der CRD

```
# Generiere die Manifeste
make manifests

# Wende die CRD auf den Cluster an
kubectl apply -f config/crd/bases
```

Bereitstellen des Operators

```
# Operator-Image erstellen und in ein Container-Registry pushen
make docker-build docker-push IMG=example/my-operator:v0.1.0

# Deployment Manifeste anwenden
make deploy IMG=example/my-operator:v0.1.0
```

Verwalten der benutzerdefinierten Ressource

Instanz der benutzerdefinierten Ressource erstellen, um den Operator zu testen:

```
apiVersion: cache.example.com/v1alpha1
kind: Memcached
metadata:
  name: example-memcached
spec:
  size: 3
```

```
# Benutzerdefinierte Ressource erstellen
kubectl apply -f config/samples/cache_v1alpha1_memcached.yaml

# Status des Operators und der Ressourcen überprüfen
kubectl get memcached
kubectl describe memcached example-memcached
```

10 Überwachung und Debugging

10.1 Events

Events ermöglichen die Überwachung von Ereignissen im Kubernetes-Cluster.

Befehl	Beschreibung
`kubectl get events`	Alle Events im aktuellen Namespace auflisten
`kubectl describe event <event-name>`	Details zu einem Event anzeigen
`kubectl get events –sort-by=.metadata.creationTimestamp`	Events nach Erstellungszeitpunkt sortieren
`kubectl get events -n <namespace>`	Alle Events in einem bestimmten Namespace auflisten
`kubectl get events –field-selector involvedObject.name=<resource-name>`	Events für eine bestimmte Ressource filtern

10.1.1 Anwendungsfälle für Events

- Überwachung des Zustands und Verhaltens von Ressourcen im Cluster.
- Fehlerbehebung und Diagnose von Problemen bei Deployments und Betriebsabläufen.
- Nachvollziehen von Änderungen und Aktivitäten im Cluster.
- Unterstützung bei der Einhaltung von Compliance- und Auditanforderungen.
- Integration in Monitoring- und Benachrichtigungssysteme für proaktive Überwachung.

10.1.2 Best Practices für die Nutzung von Events

- Regelmäßig die Events im Cluster überwachen, um frühzeitig auf Probleme reagieren zu können.
- Tools und Dashboards wie Kubernetes Dashboard oder Prometheus nutzen, um Events zu visualisieren und zu analysieren.
- Benachrichtigungssysteme implementieren, um bei kritischen Events automatisch alarmiert zu werden.
- Regelmäßige Audits der Events durchführen, um sicherzustellen, dass alle Änderungen und Aktivitäten nachvollziehbar sind.
- Events nach Relevanz sortieren und filtern, um die wichtigsten Informationen schnell zu identifizieren.

10.1.3 Integration von Events in Monitoring-Tools

Kubernetes-Events können in verschiedene Monitoring- und Logging-Tools integriert werden, um eine umfassende Überwachung und Analyse zu ermöglichen:

- **Prometheus und Grafana**: Den `kube-state-metrics` Exporter nutzen, um Kubernetes-Events in Prometheus zu importieren und mit Grafana zu visualisieren.
- **ELK Stack (Elasticsearch, Logstash, Kibana)**: Events an Elasticsearch weiterleiten und sie mit Kibana visualisieren.
- **Alertmanager**: Benachrichtigungen basierend auf bestimmten Event-Typen oder -Meldungen einrichten.

Automatisierte Benachrichtigungen bei kritischen Events

```yaml
# Beispielregel für Alertmanager
groups:
  - name: kubernetes-events
    rules:
      - alert: HighErrorRate
        expr: 'rate(kube_event_total{reason="Failed"}[5m]) > 5'
        for: 10m
        labels:
          severity: critical
        annotations:
          summary: "High error rate detected in Kubernetes events"
          description: "More than 5 failed events in 5 minutes."
```

10.1.4 Zusätzliche Ressourcen und Tools

- Kubernetes-Dokumentation zu Events: Offizielle Dokumentation zu Events in Kubernetes.
- Kubernetes Dashboard: Ein Web-UI, um Kubernetes-Clusterressourcen zu verwalten und zu überwachen.
- kube-state-metrics: Ein Prometheus-Exporter, der den Zustand der Kubernetes-Ressourcen überwacht.
- Prometheus: Ein Open-Source-Monitoring-System und Zeitreihen-Datenbank.
- ELK Stack: Eine Sammlung von Tools für das Suchen, Analysieren und Visualisieren von loggierten Daten in Echtzeit.
- Grafana: Ein Open-Source-Analysetool zur Visualisierung von Metriken.

10.1.5 Integration mit Prometheus und Grafana

```
# Installiere kube-state-metrics in den Kubernetes-Cluster
kubectl apply -f https://github.com/kubernetes/kube-state-metrics/blob/master/
↪  examples/standard/kube-state-metrics-service-account.yaml
kubectl apply -f https://github.com/kubernetes/kube-state-metrics/blob/master/
↪  examples/standard/kube-state-metrics-cluster-role.yaml
kubectl apply -f https://github.com/kubernetes/kube-state-metrics/blob/master/
↪  examples/standard/kube-state-metrics-cluster-role-binding.yaml
kubectl apply -f https://github.com/kubernetes/kube-state-metrics/blob/master/
↪  examples/standard/kube-state-metrics-deployment.yaml
kubectl apply -f https://github.com/kubernetes/kube-state-metrics/blob/master/
↪  examples/standard/kube-state-metrics-service.yaml
```

Installation von kube-state-metrics

Konfiguration von Prometheus

Die Prometheus-Konfigurationsdatei bearbeiten, um kube-state-metrics als Datenquelle hinzuzufügen:

```
scrape_configs:
  - job_name: kube-state-metrics
    static_configs:
      - targets:
          - <kube-state-metrics-service>:8080
```

Visualisierung in Grafana

Dashboard in Grafana erstellen und Panels hinzufügen, um Kubernetes-Events zu visualisieren. Prometheus-Abfragen verwenden, um relevante Event-Metriken darzustellen.

```
# Beispielhafte Prometheus-Abfrage zur Visualisierung von Event-Typen
count by(reason) (kube_event_total)
```

Einrichtung von Benachrichtigungen

Grafana Alarme einrichten, um bei kritischen Events automatisch Benachrichtigungen zu erhalten:

```
# Beispielhafte Alarmregel in Grafana
alert: HighErrorRate
expr: 'rate(kube_event_total{reason="Failed"}[5m]) > 5'
for: 10m
labels:
  severity: critical
annotations:
  summary: "High error rate detected in Kubernetes events"
  description: "More than 5 failed events in the last 5 minutes."
```

10.2 Logs und Debugging

Befehle zur Überwachung und Fehlerbehebung von Anwendungen.

Befehl	Beschreibung
`kubectl logs <pod-name>`	Logs eines Pods anzeigen
`kubectl logs <pod-name> -c <container-name>`	Logs eines bestimmten Containers in einem Pod anzeigen
`kubectl logs -f <pod-name>`	Live-Logs eines Pods verfolgen
`kubectl exec -it <pod-name> – /bin/sh`	Eine interaktive Shell in einem Pod starten
`kubectl exec <pod-name> – <command>`	Einen Befehl ausführen, ohne eine Shell zu starten
`kubectl describe pod <pod-name>`	Detaillierte Informationen über einen Pod anzeigen
`kubectl get pods -o wide`	Status aller Pods in einem Namespace anzeigen
`kubectl get pod <pod-name> -o yaml`	gibt den Status eines Pods im YAML-Format aus
`kubectl get events`	Cluster-Events anzeigen, um Probleme und Statusänderungen zu überwachen
`kubectl top pod <pod-name>`	Ressourcenverbrauch (CPU/Memory) eines Pods anzeigen
`kubectl port-forward <pod-name> <local-port>:<pod-port>`	Port Forwarding (Netzwerkzugriff auf einen Pod innerhalb des Clusters von außerhalb des Clusters) einrichten
`kubectl cp <pod-name>:/path/to/file /local/path`	Dateien von einem Pod auf den lokalen Rechner kopieren
`kubectl cp /local/path <pod-name>:/path/to/file`	Dateien vom lokalen Rechner in einen Pod kopieren

10.2.1 Fehlerbehebung bei Pod-Problemen

- Überprüfen der Logs des Pods oder der Container innerhalb des Pods.
- Überprüfen der Events
- Überprüfen der Ressourcenlimits
- Überprüfen der Netzwerkverbindungen
- Wiederherstellen eines Pods durch Löschen und automatische Wiederherstellung

10.2.2 Nützliche Tools und Erweiterungen

- **K9s**: Ein Terminal basiertes UI, um Kubernetes-Cluster zu verwalten und zu überwachen.
- **Lens**: Eine IDE für Kubernetes, die eine grafische Benutzeroberfläche bietet und es ermöglicht, Cluster-Ressourcen einfach zu überwachen und zu verwalten.
- **OpenLens**: Nur der offene, unter der MIT-Lizenz stehende, Teil von Lens
- **Stern**: Ein Tool, um Logs von mehreren Pods und Containern in Echtzeit zu verfolgen.
- **kubectx/kubens**: Werkzeuge zum schnellen Wechseln zwischen Kubernetes-Clustern (kubectx) und Namespaces (kubens).
- **Prometheus und Grafana**: Prometheus ist ein Monitoring- und Alerting-Toolkit, das speziell für die Überwachung von Kubernetes-Cluster entwickelt wurde. Grafana wird oft zusammen mit Prometheus verwendet, um visuelle Dashboards zu erstellen.
- **Jaeger**: Ein Tool zur verteilten Tracing-Überwachung, das hilft, Performance-Probleme und Latenz in verteilten Systemen zu analysieren.
- **Fluentd/Fluent Bit**: Logging-Agenten, die Logs in Echtzeit sammeln, transformieren und an verschiedene Backends weiterleiten können.

Verwendung von K9s

K9s ist ein beliebtes Tool, das eine benutzerfreundliche Oberfläche zur Verwaltung und Überwachung von Kubernetes-Clustern im Terminal bietet.

```
# Installation von K9s
brew install k9s
# Starten von K9s
k9s
```

Es kann unter Anderem genutzt werden um durch Kubernetes-Ressourcen zu navigieren, Logs anzuzeigen und Pods neu zu starten. Dies wird über eine Terminal-Oberfläche realisiert.

Verwendung von kubectx und kubens

Diese Tools erleichtern den Wechsel zwischen Kubernetes-Clustern und Namespaces.

```
# Installation von kubectx und kubens
brew install kubectx kubens

# Wechseln zu einem anderen Cluster
kubectx my-cluster

# Wechseln zu einem anderen Namespace
kubens my-namespace
```

Echtzeit-Log-Verfolgung mit Stern

Stern ermöglicht das Verfolgen von Logs mehrerer Pods gleichzeitig.

```
# Installation von Stern
brew install stern

# Logs aller Pods mit einem bestimmten Label in Echtzeit verfolgen
stern app=myapp
```

10.3 Metrics Server

Der Metrics Server sammelt und aggregiert Echtzeitmetriken von Pods und Nodes im Kubernetes-Cluster. Diese Metriken werden für Auto-Scaling verwendet. Für Monitoring soll der Kubelet Endpoint `/metrics/resource` verwendet werden.

Befehl	Beschreibung
`kubectl get –raw /apis/metrics.k8s.io/v1beta1/nodes”`	Metriken aller Nodes im Cluster anzeigen
`kubectl get –raw /apis/metrics.k8s.io/v1beta1/pods”`	Metriken aller Pods im Cluster anzeigen
`kubectl top nodes`	Ressourcenverbrauch (CPU/Memory) aller Nodes anzeigen
`kubectl top pods`	Ressourcenverbrauch (CPU/Memory) aller Pods im aktuellen Namespace anzeigen
`kubectl top pod <pod-name>`	Ressourcenverbrauch eines bestimmten Pods anzeigen
`kubectl top pod <pod-name> –containers`	Ressourcenverbrauch der Container innerhalb eines bestimmten Pods anzeigen
`kubectl top pod –all-namespaces`	Ressourcenverbrauch aller Pods in allen Namespaces anzeigen

10.3.1 Verwendungszwecke

Metrics Server haben folgende Anwendungszwecke:

- CPU-/Speicher-basiertes horizontales Autoscaling
- Automatisches Anpassen/Vorschlagen der von Containern benötigten Ressourcen (Vertikales Autoscaling)

Der Metric Server sollte nicht genutzt werden, wenn Folgendes benötigt wird:

- Nicht-Kubernetes-Cluster
- Eine genaue Quelle für Ressourcennutzungsmetriken
- Horizontales Autoscaling basierend auf anderen Ressourcen als CPU/Speicher

Für nicht unterstützte Anwendungsfälle sollten vollständige Überwachungslösungen wie Prometheus genutzt werden.

10.3.2 Installation und Konfiguration des Metrics Servers

```
# YAML-Manifeste des Metrics Servers herunterladen
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/
↪  latest/download/components.yaml
```

Nach der Installation kann der Status des Metrics Servers überprüft werden:

```
# Überprüfen, ob die Metrics Server Pods laufen
kubectl get pods -n kube-system | grep metrics-server
```

10.3.3 Konfiguration

Abhängig von der Cluster-Einrichtung müssen möglicherweise auch die an den Metrics Server-Container übergebenen Flags geändert werden. Die nützlichsten Flags:

- „texttt–kubelet-preferred-address-types“ - Die Priorität der Knotentypen, die verwendet werden, wenn eine Adresse für die Verbindung zu einem bestimmten Knoten ermittelt wird (Standard: [Hostname, InternalDNS, InternalIP, ExternalDNS, ExternalIP])
- „texttt–kubelet-insecure-tls“ - Die CA der von Kubelets präsentierten Servier-Zertifikate nicht überprüfen. Nur für Testzwecke.
- „texttt–requestheader-client-ca-file“ - Ein Root-Zertifikat-Bundle zur Überprüfung von Client-Zertifikaten bei eingehenden Anfragen angeben.
- „texttt–node-selector“ - Kann vervollständigen, um die Metriken von den angegebenen Knoten basierend auf Labels zu erfassen

Eine vollständige Liste der Konfigurations-Flags des Metrics Servers wird ausgegeben, indem folgender Befehl ausgeführt wird:

```
docker run --rm registry.k8s.io/metrics-server/metrics-server:v0.7.0 --help
```

10.3.4 Verwendung des Metrics Servers für Auto-Scaling

Der Metrics Server wird häufig in Kombination mit dem Horizontal Pod Autoscaler (HPA) verwendet, um die Anzahl der Replicas basierend auf den CPU- und Speicherauslastungen automatisch zu skalieren.

Erstellen eines Horizontal Pod Autoscalers

```
# Erstelle einen HPA für eine Deployment
kubectl autoscale deployment <deployment-name> --cpu-percent=50 --min=1 --max=10
```

Überprüfen des Status des Horizontal Pod Autoscalers

```
# Zeige den Status des HPA an
kubectl get hpa
```

10.3.5 Best Practices für die Nutzung des Metrics Servers

- Stelle sicher, dass der Metrics Server korrekt installiert und konfiguriert ist, um genaue Metriken zu erhalten.
- Überwache den Zustand und die Logs des Metrics Servers, um sicherzustellen, dass er ordnungsgemäß funktioniert.
- Verwende den Metrics Server in Kombination mit anderen Monitoring- und Logging-Tools, um eine umfassende Überwachung des Clusters zu gewährleisten.
- Skalieren Sie den Metrics Server entsprechend der Größe und den Anforderungen Ihres Clusters, um eine optimale Leistung zu gewährleisten.
- Aktualisieren Sie den Metrics Server regelmäßig, um von Verbesserungen und Fehlerbehebungen zu profitieren.

10.3.6 Troubleshooting des Metrics Servers

Falls Probleme mit dem Metrics Server auftreten, können folgende Schritte zur Fehlerbehebung helfen:

- Logs des Metrics Server Pods überprüfen:

```
kubectl logs -n kube-system <metrics-server-pod-name>
```

- Sicherstellen, dass der Metrics Server die erforderlichen Berechtigungen hat:

```
kubectl describe clusterrole system:metrics-server
```

- Überprüfen, ob der Metrics Server die Metriken korrekt sammelt und verarbeitet:

```
kubectl get --raw "/apis/metrics.k8s.io/v1beta1/nodes"
kubectl get --raw "/apis/metrics.k8s.io/v1beta1/pods"
```

- Vergewissern, dass der Metrics Server auf alle Nodes im Cluster zugreifen kann:

```
kubectl get nodes
```

- Die Netzwerkverbindung und Firewall-Einstellungen überprüfen, um sicherzustellen, dass der Metrics Server die Nodes erreichen kann:

```
kubectl get svc -n kube-system metrics-server
```

- Überprüfen, ob der Metrics Server die richtigen Ressourcenlimits und -anforderungen hat:

```
kubectl describe pod <metrics-server-pod-name> -n kube-system
```

- Sicherstellen, dass der Metrics Server korrekt konfiguriert ist, um Metriken von den Nodes zu sammeln. Die Konfigurationsdateien und Argumente des Metrics Servers prüfen:

```
kubectl describe deployment metrics-server -n kube-system
```

- Überprüfen, ob alle Abhängigkeiten des Metrics Server erfüllt sind, wie z.B. die API-Server- und Kubelet-Konfiguration:

```
kubectl get apiservices
```

10.4 Monitoring-Tools

10.4.1 Dashboard

Dashboard installieren und starten

1. `helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/`
2. `helm repo update`
3. `helm upgrade –install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard –create-namespace –namespace kubernetes-dashboard`
4. `kubectl -n kubernetes-dashboard port-forward svc/kubernetes-dashboard-kong-proxy 8443:443 # Pod muss laufen (Start dauert eventuell etwas)`

Das Dashboard ist in diesem Fall unter „`https://localhost:8443`“ erreichbar, es kann aber auch ein anderer Port verwendet werden.

Bearer Token erstellen

Um auf das Dashboard zugreifen zu können muss ein BearerToken erstellt werden

1. **ServiceAccount erstellen:**
   `kubectl create serviceaccount dashboard-admin-sa -n kubernetes-dashboard`
2. **ClusterRoleBindung erstellen:**
   `kubectl create clusterrolebinding dashboard-admin-sa –clusterrole=cluster-admin –serviceaccount=kubernetes-dashboard:dashboard-admin-sa`
3. **BearerToken generieren:**
   `kubectl -n kubernetes-dashboard create token dashboard-admin-sa`

Zum Login wird das BearerToken bei der Seite des Dashboards im Browser eingesetzt.

10.4.2 Prometheus

Prometheus ist ein Open-Source-Monitoring-System und eine Zeitreihen-Datenbank, die speziell für die Überwachung und Alarmierung von Cloud-nativen Anwendungen entwickelt wurde. Es ist bekannt für seine flexible Abfragesprache, PromQL, und seine Fähigkeit, Metriken in Echtzeit zu sammeln und zu speichern.

Hauptmerkmale von Prometheus:

- Multi-dimensionales Datenmodell
- Flexible und leistungsstarke Abfragesprache (PromQL)
- Autonomer Betrieb ohne Abhängigkeiten
- Push-basierte Metriksammlung über das Prometheus-Pushgateway

Prometheus installieren

1. `helm repo add prometheus-community https://prometheus-community.github.io/helm-charts`
2. `helm repo update`
3. `helm install prometheus prometheus-community/kube-prometheus-stack –namespace monitoring –create-namespace`
4. `kubectl port-forward -n monitoring svc/prometheus-kube-prometheus-prometheus 9090:9090 # Pod muss laufen`

Prometheus ist in diesem Fall im Browser unter `127.0.0.1:9090` erreichbar.

10.4.3 Grafana

Grafana ist ein Open-Source-Analyse- und Visualisierungstool, das zur Überwachung und Analyse von Metriken verwendet wird. Es bietet eine benutzerfreundliche Oberfläche zum Erstellen und Verwalten von Dashboards und unterstützt eine Vielzahl von Datenquellen, einschließlich Prometheus.

Hauptmerkmale von Grafana:

- Unterstützung für mehrere Datenquellen (z. B. Prometheus, Graphite, InfluxDB)
- Anpassbare Dashboards mit einer Vielzahl von Visualisierungsoptionen
- Alarmierungs- und Benachrichtigungsfunktionen
- Benutzer- und Rechteverwaltung

Grafana installieren

Für Grafana muss Prometheus, oder eine andere Datenquelle installiert sein.

1. `helm repo add grafana https://grafana.github.io/helm-charts`
2. `helm repo update`
3. `helm install grafana grafana/grafana –namespace monitoring`

Passwort generieren:

```
kubectl get secret --namespace monitoring grafana -o
jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

Umgebungsvariable setzen

```
export POD_NAME=$(kubectl get pods --namespace monitoring -l
"app.kubernetes.io/name=grafana,app.kubernetes.io/instance=grafana"
-o jsonpath="{.items[0].metadata.name}")
```

Auf Pod starten

```
kubectl --namespace monitoring port-forward $POD_NAME 3000
```

In diesem Fall ist Grafana im Browser unter „127.0.0.1:3000“ unter dem Nutzernamen „admin“ erreichbar.

10.4.4 ELK-Stack

Der ELK-Stack besteht aus Elasticsearch, Logstash und Kibana und bietet eine leistungsstarke Lösung für die Sammlung, Analyse und Visualisierung von Log-Daten.
Jedoch sind die Lizenzen restriktiver: Cloud-Service Anbieter dürfen Elasticsearch und Kibana nicht als gehostete Dienste anbieten, es sei denn sie haben eine komerzielle Vereinbarung mit Elastic. Zudem muss jeder, der den Dienst öffentlich zugänglich macht auch den gesamten Code der verwendeten Software, einschließlich der eigenen Anpassungen unter derselben Lizenz zur Verfügung stellen. Logstash und Beats sind unter der Apache 2.0 Lizenz verfügbar, was ihre Anwendung flexibler macht und es gibt proprietäre Plugins, die jeweils einer Lizenzgebühr unterliegen.

Elasticsearch:

Eine verteilte Such- und Analyse-Engine, die für ihre Skalierbarkeit und Echtzeitleistung bekannt ist.

Logstash:

Ein Datenverarbeitungspipeline-Tool, das Daten von einer Vielzahl von Quellen sammeln, transformieren und in Elasticsearch einfügen kann.

Kibana:

Ein Visualisierungs- und Dashboard-Tool, das speziell für die Arbeit mit Elasticsearch entwickelt wurde.

10.4.5 TICK-Stack

Der TICK-Stack besteht aus vier Hauptkomponenten: Telegraf, InfluxDB, Chronograf und Kapacitor. Zusammen bieten sie eine umfassende Lösung zur Sammlung, Speicherung, Visualisierung und Alarmierung von Zeitreihendaten.

Telegraf:

Telegraf ist ein serverseitiger Agent zum Sammeln und Senden von Metriken und Ereignissen aus Datenbanken, Systemen und IoT-Sensoren. Es unterstützt eine Vielzahl von Eingabe- und Ausgabe-Plugins, was es sehr flexibel macht.

InfluxDB:

InfluxDB ist eine leistungsstarke Zeitreihen-Datenbank, die speziell für hohe Schreiblasten und Echtzeitanalysen entwickelt wurde. Sie bietet eine SQL-ähnliche Abfragesprache (InfluxQL) sowie Unterstützung für die Datenkomprimierung.

Chronograf:

Chronograf ist das visuelle Dashboarding-Tool des TICK-Stacks. Es ermöglicht die einfache Visualisierung und Analyse der in InfluxDB gespeicherten Daten sowie das Erstellen und Verwalten von Dashboards.

Kapacitor:

Kapacitor ist die Echtzeit-Datenverarbeitungs- und Alarmierungskomponente des TICK-Stacks. Es ermöglicht die Erstellung von benutzerdefinierten Datenverarbeitungs-Pipelines und Alarmierungsregeln basierend auf den in InfluxDB gespeicherten Daten.

TICK-Stack installieren

1. **Telegraf installieren:**

```
wget -q https://repos.influxdata.com/influxdata-archive_compat.key
echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c influxdata-archive_compat.k
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg] https://repos.influxdat
sudo apt-get update && sudo apt-get install telegraf
```

2. **InfluxDB OSS installieren:**

```
wget -q https://repos.influxdata.com/influxdata-archive_compat.key
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg] https://repos.influxdat
sudo apt-get update && sudo apt-get install influxdb
sudo systemctl unmask influxdb.service
sudo systemctl start influxdb
```

3. **Chronograf installieren:**

```
wget https://download.influxdata.com/chronograf/releases/chronograf_1.10.5_amd64.deb
sudo dpkg -i chronograf_1.10.5_amd64.deb
```

4. **Kapacitor installieren:**

```
wget https://download.influxdata.com/kapacitor/releases/kapacitor_1.7.5-1_amd64.deb
sudo dpkg -i kapacitor_1.7.5-1_amd64.deb
```

Aktuelle Versionen (Abschnitt „Are you interested in InfluxDB 1.x Open Source?“
Die verschiedenen TICK-Stack-Komponenten sind auf den entsprechenden Ports erreichbar, die in ihren Konfigurationsdateien angegeben sind.

10.4.6 Integration von Prometheus, Grafana und ELK-Stack in Kubernetes

Die Integration dieser Tools in Kubernetes ermöglicht eine umfassende Überwachung und Analyse der Cluster-Ressourcen und Anwendungen.

Prometheus:

- Prometheus mithilfe von Helm installieren:

```
helm install prometheus stable/prometheus
```

- Prometheus wird konfiguriert, um Metriken von Kubernetes zu sammeln, indem der `kube-state-metrics` Exporter hinzugefügt wird.

Grafana:

- Grafana mithilfe von Helm installieren:

```
helm install grafana stable/grafana
```

- Prometheus als Datenquelle in Grafana hinzufügen und Dashboards zur Visualisierung der Metriken erstellen.

ELK-Stack:

- ELK-Stack mithilfe von Helm installieren:

```
helm install elasticsearch stable/elasticsearch
helm install logstash stable/logstash
helm install kibana stable/kibana
```

- Logstash konfigurieren, um Logs von Kubernetes zu sammeln und an Elasticsearch weiterzuleiten. Logstash-Konfigurationsdatei:

```
input {
  file {
    path ⇒ "/var/log/containers/*.log"
    type ⇒ "kubernetes"
  }
}

filter {
  json {
    source ⇒ "message"
  }
}

output {
  elasticsearch {
    hosts ⇒ ["http://elasticsearch:9200"]
  }
}
```

- In Kibana eine Indexvorlage hinzufügen, um die Logs von Elasticsearch zu visualisieren.

10.4.7 Best Practices für Monitoring in Kubernetes

Um sicherzustellen, dass Ihre Monitoring-Strategie effektiv ist, sollten die folgenden Best Practices befolgt werden:

- **Zentrale Überwachung:** Zentrale Monitoring-Tools nutzen, um alle Metriken, Logs und Events an einem Ort zu sammeln und zu analysieren.
- **Automatisierte Benachrichtigungen:** Alarme und Benachrichtigungen einrichten, um bei kritischen Ereignissen oder Metriken sofort informiert zu werden.
- **Ressourcensparende Konfiguration:** Sicherstellen, dass die Monitoring-Tools ressourcenschonend konfiguriert sind, um die Leistung des Clusters nicht zu beeinträchtigen.
- **Regelmäßige Audits:** Regelmäßige Audits der Monitoring-Konfiguration durchführen, um sicherzustellen, dass alle relevanten Metriken und Logs erfasst werden.
- **Sicherheitsaspekte beachten:** Auf die Sicherheit der Monitoring-Tools achten, insbesondere wenn diese Zugang zu sensiblen Daten haben.
- **Dokumentation und Schulung:** Die Monitoring-Strategie dokumentieren und das Team im Umgang mit den Tools und Dashboards schulen.

10.4.8 Zusätzliche Ressourcen

- Prometheus Website
- Grafana Website
- ELK-Stack Website

10.4.9 Verwendung von JSONPath

JSONPath ist eine Abfragesprache, die es ermöglicht, bestimmte Teile einer JSON-Struktur zu extrahieren. In Kubernetes wird JSONPath verwendet, um spezifische Felder aus den JSON-Antworten der Kubernetes-API zu extrahieren. Dies ist besonders nützlich, um präzise Informationen aus Ressourcen zu erhalten.

Grundlagen von JSONPath

JSONPath verwendet ähnliche Konzepte wie XPath, das für XML verwendet wird.
Grundlegende JSONPath-Ausdrücke:

- `.`: Wurzelselektor, der das gesamte Dokument darstellt.
- `.field`: Selektiert das angegebene Feld.
- `.field.subfield`: Selektiert ein Unterfeld.
- `.field[index]`: Selektiert ein Element in einem Array.
- `.field[*]`: Selektiert alle Elemente in einem Array.

Beispiele für JSONPath-Ausdrücke

Beispiele, wie JSONPath-Ausdrücke verwendet werden können:

```
# Anzahl der Replikate eines Deployments anzeigen
kubectl get deployment nginx-deployment -o=jsonpath='{.spec.replicas}'

# Image des Containers in einem Deployment anzeigen
kubectl get deployment nginx-deployment -o=jsonpath='{.spec.template.spec.containers[0].image}'

# Namen aller Pods in einem Namespace anzeigen
kubectl get pods -o=jsonpath='{.items[*].metadata.name}'

# Status eines bestimmten Pods anzeigen
kubectl get pod nginx-pod -o=jsonpath='{.status.phase}'

# Labels eines Deployments anzeigen
kubectl get deployment nginx-deployment -o=jsonpath='{.metadata.labels}'

# Ressourcenanforderungen eines Containers anzeigen
kubectl get pod nginx-pod -o=jsonpath='{.spec.containers[0].resources.requests}'

# Liste aller Namespaces anzeigen
kubectl get namespaces -o=jsonpath='{.items[*].metadata.name}'

# Alle Services in einem Namespace anzeigen
kubectl get services -o=jsonpath='{.items[*].metadata.name}'

# Knoteninformationen für einen bestimmten Pod anzeigen
kubectl get pod nginx-pod -o=jsonpath='{.spec.nodeName}'
```

JSON-Beispiel

Beispiel für die JSON-Ausgabe eines Deployments, das von der Kubernetes-API zurückgegeben wird:

```
{
  "apiVersion": "apps/v1",
  "kind": "Deployment",
  "metadata": {
    "name": "nginx-deployment",
    "namespace": "default",
    "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
              }
            ]
          }
        ]
      }
    }
  },
  "status": {
    "replicas": 3,
    "updatedReplicas": 3,
    "readyReplicas": 3,
    "availableReplicas": 3
  }
}
```

Mit dem JSONPath-Ausdruck `.spec.replicas` wird der Wert des Feldes `replicas` innerhalb des `spec`-Blocks extrahiert, was in diesem Fall `3` wäre.

Verwendung von JSONPath in kubectl

JSONPath kann in Kombination mit `kubectl` verwendet werden, um spezifische Informationen aus Kubernetes-Ressourcen zu extrahieren und anzuzeigen. Hier sind einige Beispiele, wie dies gemacht werden kann:

```
# Anzahl der Replikate eines Deployments anzeigen
kubectl get deployment nginx-deployment -o=jsonpath='{.spec.replicas}'

# Image des Containers in einem Deployment anzeigen
kubectl get deployment nginx-deployment -o=jsonpath='{.spec.template.spec.containers[0].image}'

# Namen aller Pods in einem Namespace anzeigen
kubectl get pods -o=jsonpath='{.items[*].metadata.name}'

# IP-Adressen aller Nodes anzeigen
kubectl get nodes -o=jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}'

# Namen und Images aller Container in einem Pod anzeigen
kubectl get pod nginx-pod -o=jsonpath='{.spec.containers[*].name}:{.spec.containers[*].image}'
```

Durch die Verwendung von JSONPath-Ausdrücken in `kubectl` können Administratoren und Entwickler präzise und gezielte Informationen aus der API-Antwort extrahieren, ohne die gesamte JSON-Antwort durchsehen zu müssen.

Weitere nützliche JSONPath-Ausdrücke

Hier sind einige zusätzliche nützliche JSONPath-Ausdrücke, die häufig in Kubernetes verwendet werden:

- `.items[*].metadata.name`: Selektiert die Namen aller Elemente in einer Liste (z.B. alle Pod-Namen).
- `.items[?(@.status.phase=="Running")].metadata.name`: Selektiert die Namen aller Pods, die im Zustand "Runningßind.
- `.spec.template.spec.containers[?(@.name=="nginx")].image`: Selektiert das Image des Containers mit dem Namen "nginx".
- `.items[*].status.containerStatuses[?(@.ready=true)].name`: Selektiert die Namen aller Container, die bereit sind.

Nützliche Links und Ressourcen

Um mehr über JSONPath und seine Verwendung in Kubernetes zu erfahren, sind hier einige nützliche Links und Ressourcen:

- Offizielle JSONPath-Spezifikation: https://goessner.net/articles/JsonPath/
- `kubectl` Dokumentation: https://kubernetes.io/docs/reference/kubectl/overview/
- Kubernetes JSONPath Support: https://kubernetes.io/docs/reference/kubectl/jsonpath/

11 Taints und Affinitäten

11.1 Taints und Tolerations

Taints und Tolerations werden verwendet, um die Zuweisung von Pods zu Nodes zu steuern. Ein „Taint“ ist eine Eigenschaft, die einer Node zugewiesen wird und verhindert, dass Pods auf dieser Node geplant werden, es sei denn, der Pod hat eine entsprechende Toleration. Dies hilft, bestimmte Nodes für spezielle Workloads zu reservieren oder bestimmte Pods von bestimmten Nodes fernzuhalten.

Befehl	Beschreibung
`kubectl taint nodes <node-name> key=value:taint-effect`	Einen Taint zu einer Node hinzufügen
`kubectl taint nodes <node-name> key:NoSchedule-`	Einen Taint von einer Node entfernen
`kubectl get nodes –show-labels`	Alle Nodes mit ihren Labels anzeigen
`kubectl label nodes <node-name> <label-key>=<label-value>`	Ein Label zu einer Node hinzufügen
`kubectl label nodes <node-name> <label-key>–`	Ein Label von einer Node entfernen
`kubectl describe node <node-name>`	Details zu einer Node anzeigen, einschließlich Taints und Labels
`kubectl get pods –selector=<label-key>=<label-value>`	Alle Pods anzeigen, die einem bestimmten Label entsprechen

11.1.1 Taint-Effekte

Es gibt drei Haupttypen von Taint-Effekten, die verwendet werden können:

- `NoSchedule`: Verhindert, dass der Scheduler einen Pod auf einer Node plant, es sei denn, der Pod hat eine entsprechende Toleration.
- `PreferNoSchedule`: Vermeidet im Allgemeinen, dass der Scheduler einen Pod auf einer Node plant, ist jedoch nicht strikt zwingend.
- `NoExecute`: Entfernt bereits laufende Pods von der Node und verhindert, dass neue Pods geplant werden, es sei denn, sie haben eine entsprechende Toleration.

11.1.2 Anwendung von Taints

```
# Füge einen NoSchedule Taint zu einer Node hinzu
kubectl taint nodes node1 key=value:NoSchedule

# Füge einen PreferNoSchedule Taint zu einer Node hinzu
kubectl taint nodes node1 key=value:PreferNoSchedule

# Füge einen NoExecute Taint zu einer Node hinzu
kubectl taint nodes node1 key=value:NoExecute

# Entferne einen Taint von einer Node
kubectl taint nodes node1 key:NoSchedule-
```

11.1.3 Tolerations in Pod-Spezifikationen

Toleration in einem Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: tolerant-pod
spec:
  tolerations:
    - key: key
      operator: Equal
      value: value
      effect: NoSchedule
  containers:
    - name: nginx
      image: nginx
```

Erklärung der Felder in Tolerations

- `key`: Der Schlüssel des Taints, den der Pod toleriert.
- `operator`: Der Operator, der den Vergleichsmodus bestimmt (z.B. `Equal`).
- `value`: Der Wert des Taints, den der Pod toleriert.
- `effect`: Der Effekt des Taints, den der Pod toleriert (z.B. `NoSchedule`).

11.1.4 Anwendungsfälle für Taints und Tolerations

- Reservieren von Nodes für spezielle Workloads, beispielsweise für hochverfügbare oder ressourcenintensive Anwendungen.
- Fernhalten bestimmter Pods von Nodes, die für spezielle Zwecke verwendet werden, wie z.B. Datenbankknoten oder spezielle Hardware.
- Isolierung von Workloads, um sicherzustellen, dass bestimmte Pods nicht auf denselben Nodes ausgeführt werden.
- Implementierung von Node-Maintenance-Strategien, bei denen Nodes temporär für Wartungsarbeiten gesperrt werden.

11.1.5 Best Practices für die Verwendung von Taints und Tolerations

- `NoExecute`-Taints vorsichtig verwenden, da sie laufende Pods von Nodes entfernen können.
- Die Verwendung von Taints und Tolerations in dem Cluster dokumentieren, um Missverständnisse und Fehlkonfigurationen zu vermeiden.
- Die Auswirkungen von Taints und Tolerations auf die Pod-Platzierung und -Verfügbarkeit überwachen, um sicherzustellen, dass Workloads wie erwartet ausgeführt werden.
- Taints und Tolerations mit anderen Kubernetes-Mechanismen wie Node-Selectors und Affinity/Anti-Affinity-Regeln kombinieren, um eine feinere Steuerung der Pod-Platzierung zu erreichen.
- Spezifische Schlüssel und Werte für Taints und Tolerations verwenden, um eine gezielte Steuerung zu ermöglichen und Kollisionen zu vermeiden.
- Die Konfiguration von Taints und Tolerations in einer Staging-Umgebung testen, bevor sie in die Produktion übernommen wird.

11.1.6 Weitere nützliche Befehle für Taints und Tolerations

Neben den grundlegenden Befehlen gibt es weitere nützliche Kommandos, die bei der Verwaltung von Taints und Tolerations hilfreich sein können:

Befehl	Beschreibung
`kubectl get nodes -o json | jq '.items[].spec.taints'`	Liste alle Taints auf allen Nodes im Cluster auf
`kubectl describe node <node-name>`	Zeige detaillierte Informationen, einschließlich Taints, zu einer bestimmten Node an
`kubectl describe pod <pod-name>`	Zeige Informationen zu einem Pod, einschließlich seiner Tolerations, an
`kubectl get pod -o json | jq '.items[].spec.tolerations'`	Liste alle Tolerations von allen Pods im Cluster auf

11.1.7 Node für spezialisierte Workloads reservieren

Einen Taint zur Node hinzufügen

```
kubectl taint nodes database-node dedicated=db:NoSchedule
```

Pod-Spezifikation mit Toleration

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
spec:
  tolerations:
    - key: dedicated
      operator: Equal
      value: db
      effect: NoSchedule
  containers:
    - name: postgres
      image: postgres:latest
```

11.1.8 Temporäre Node-Sperrung für Wartungsarbeiten

Wenn eine Node temporär für Wartungsarbeiten gesperrt werden soll, kann ein `NoExecute`-Taint verwendet werden, um alle Pods von der Node zu entfernen und neue Pods daran zu hindern, auf dieser Node geplant zu werden:

Einen `NoExecute`-Taint hinzufügen

```
kubectl taint nodes maintenance-node maintenance=true:NoExecute
```

Taint nach Wartungsarbeiten entfernen

```
kubectl taint nodes maintenance-node maintenance:NoExecute-
```

11.2 Affinity und Anti-Affinity

Affinity und Anti-Affinity werden verwendet, um die Platzierung von Pods auf Nodes zu steuern. Sie sind Mechanismen, die es ermöglichen, Pods auf spezifischen Nodes oder in der Nähe (oder fern) von anderen Pods zu platzieren. Diese Mechanismen bieten eine feinere Steuerung der Pod-Platzierung, um verschiedene Anforderungen wie Performance-Optimierung, Ressourcenauslastung oder Redundanz zu erfüllen.

Attribut	Beschreibung
`nodeAffinity`	Bestimmt, auf welchen Nodes ein Pod bevorzugt oder zwingend ausgeführt werden soll
`podAffinity`	Bestimmt, dass Pods in der Nähe von bestimmten anderen Pods ausgeführt werden sollen
`podAntiAffinity`	Bestimmt, dass Pods nicht in der Nähe von bestimmten anderen Pods ausgeführt werden sollen

11.2.1 Befehle für Affinity und Anti-Affinity

Befehl	Beschreibung
`kubectl apply -f <pod-affinity.yaml>`	Ein Pod mit Affinity-Regeln anhand einer YAML-Datei erstellen
`kubectl describe pod <pod-name>`	Details zu einem Pod anzeigen, einschließlich Affinity und Anti-Affinity Regeln
`kubectl get pods --selector=<label-key>=<label-value>`	Alle Pods anzeigen, die einem bestimmten Label entsprechen, um Affinity zu überprüfen
`kubectl edit pod <pod-name>`	Einen Pod im Editor bearbeiten, um Affinity oder Anti-Affinity Regeln hinzuzufügen oder zu ändern
`kubectl delete pod <pod-name>`	Einen Pod löschen, um Affinity oder Anti-Affinity Regeln zu entfernen
`kubectl get nodes --show-labels`	Alle Nodes mit ihren Labels anzeigen, um Affinity-Regeln zu planen
`kubectl describe node <node-name>`	Details zu einer Node anzeigen, um zu überprüfen, ob sie den Affinity-Bedingungen entspricht
`kubectl logs <pod-name>`	Logs eines Pods anzeigen, um Probleme im Zusammenhang mit Affinity oder Anti-Affinity zu debuggen

11.2.2 Node Affinity

Node Affinity ermöglicht es, Pods auf spezifischen Nodes zu platzieren, basierend auf Node-Labels. Es gibt zwei Typen von Node Affinity:

- `requiredDuringSchedulingIgnoredDuringExecution`:
  Strikte Anforderungen, die während der Planung erfüllt sein müssen, aber nach der Platzierung ignoriert werden.
- `preferredDuringSchedulingIgnoredDuringExecution`:
  Bevorzugte, aber nicht zwingende Anforderungen, die während der Planung berücksichtigt werden, aber nach der Platzierung ignoriert werden.

Konfigurationsbeispiel für Node Affinity

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: disktype
                operator: In
                values:
                  - ssd
  containers:
    - name: nginx
      image: nginx
```

11.2.3 Pod Affinity

Pod Affinity wird verwendet, um Pods in der Nähe von anderen Pods zu platzieren, die bestimmte Kriterien erfüllen. Dies kann nützlich sein, um Pods zusammen zu gruppieren, die miteinander kommunizieren müssen.

Konfigurationsbeispiel für Pod Affinity

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-pod-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        labelSelector:
          matchLabels:
            app: frontend
        topologyKey: 'kubernetes.io/hostname'
  containers:
    - name: nginx
      image: nginx
```

11.2.4 Pod Anti-Affinity

Pod Anti-Affinity wird verwendet, um Pods von anderen Pods fernzuhalten, die bestimmte Kriterien erfüllen. Dies kann nützlich sein, um Redundanz zu gewährleisten oder Ressourcenkonflikte zu vermeiden.

Pod Anti-Affinity

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-pod-anti-affinity
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        labelSelector:
          matchLabels:
            app: frontend
        topologyKey: 'kubernetes.io/hostname'
  containers:
    - name: nginx
      image: nginx
```

11.2.5 Best Practices für die Verwendung von Affinity und Anti-Affinity

- `requiredDuringSchedulingIgnoredDuringExecution` für zwingende Platzierungsanforderungen und `preferredDuringSchedulingIgnoredDuringExecution` für bevorzugte, aber nicht zwingende Anforderungen verwenden.
- Die Verwendung von Affinity und Anti-Affinity dokumentieren, um Missverständnisse und Fehlkonfigurationen zu vermeiden.
- Die Auswirkungen von Affinity und Anti-Affinity auf die Pod-Platzierung und -Verfügbarkeit überwachen, um sicherzustellen, dass Workloads wie erwartet ausgeführt werden.
- Affinity und Anti-Affinity mit anderen Kubernetes-Mechanismen wie Taints und Tolerations kombinieren, um eine feinere Steuerung der Pod-Platzierung zu erreichen.
- Affinity- und Anti-Affinity-Konfigurationen in einer Staging-Umgebung testen, bevor du sie in die Produktion übernimmst.
- Spezifische Schlüssel und Werte für Affinity- und Anti-Affinity-Regeln verwenden, um eine gezielte Steuerung zu ermöglichen und Kollisionen zu vermeiden.
- Die Topologie deines Clusters, z.B. durch die Verwendung von `topologyKey` berücksichtigen, um Pods auf unterschiedliche Nodes, Zonen oder Regionen zu verteilen.

11.2.6 Verteilung von Pods über verschiedene Zonen

Wenn sichergestellt werden soll, dass Pods über verschiedene Zonen verteilt werden, kann Pod Anti-Affinity verwendet werden, um sicherzustellen, dass Pods nicht auf derselben Zone geplant werden:

Konfigurationsbeispiel für Pod Anti-Affinity über Zonen

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-zone-anti-affinity
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        labelSelector:
          matchLabels:
            app: my-app
        topologyKey: 'failure-domain.beta.kubernetes.io/zone'
  containers:
    - name: my-container
      image: my-image
```

11.2.7 Gruppierung von Pods auf derselben Node

Wenn sichergestellt werden soll, dass bestimmte Pods auf derselben Node ausgeführt werden, kann Pod Affinity verwendet werden:

Konfigurationsbeispiel für Pod Affinity auf derselben Node

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-node-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        labelSelector:
          matchLabels:
            app: my-app
        topologyKey: 'kubernetes.io/hostname'
  containers:
    - name: my-container
      image: my-image
```

11.3 Node Selectors

Node Selectors ermöglichen es, Pods auf bestimmte Nodes im Kubernetes-Cluster zu platzieren, indem Labels verwendet werden. Dies hilft, Workloads auf spezifizierte Hardware oder geografische Standorte zu beschränken. Node Selectors sind eine einfache und effektive Methode, um Pods auf Nodes basierend auf bestimmten Kriterien zu planen, und sind besonders nützlich in Umgebungen, in denen bestimmte Workloads spezielle Ressourcen oder Konfigurationen benötigen.

Befehl	Beschreibung
`kubectl get nodes –show-labels`	Alle Nodes mit ihren Labels auflisten
`kubectl label node <node-name> <label-key>=<label-value>`	Ein Label zu einem Node hinzufügen
`kubectl label node <node-name> <label-key>–`	Ein Label von einem Node entfernen
`kubectl get pods -o wide –field-selector spec.nodeName=<node-name>`	Alle Pods auf einem bestimmten Node auflisten
`kubectl describe node <node-name>`	Details zu einem bestimmten Node, einschließlich seiner Labels, anzeigen
`kubectl get nodes -l <label-key>=<label-value>`	Nodes mit einem bestimmten Label selektieren

11.3.1 Verwendung von Node Selectors

Um einen Pod auf Nodes mit einem bestimmten Label zu planen, muss der Node Selector in der Pod-Spezifikation definiert werden. Hier ist ein Beispiel, wie man einen Pod auf Nodes mit dem Label `disktype=ssd` plant:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-node-selector
spec:
  nodeSelector:
    disktype: ssd
  containers:
    - name: nginx
      image: nginx
```

11.3.2 Best Practices für die Verwendung von Node Selectors

- Verwendung von spezifischen und gut definierten Labels, um eine gezielte Steuerung der Pod-Platzierung zu ermöglichen.
- Dokumentierung der Verwendung von Node Selectors und Labels, um Missverständnisse und Fehlkonfigurationen zu vermeiden.
- Überwachung der Auswirkungen von Node Selectors auf die Pod-Platzierung und -Verfügbarkeit, um sicherzustellen, dass Workloads wie erwartet ausgeführt werden.
- Kombinierung von Node Selectors mit anderen Kubernetes-Mechanismen wie Taints, Tolerations, und Affinity/Anti-Affinity-Regeln, um eine noch feinere Steuerung der Pod-Platzierung zu erreichen.
- Testen von Node Selector-Konfigurationen in einer Staging-Umgebung, bevor sie in die Produktion übernommen werden.
- Berücksichtigung der Ressourcenkapazität und -auslastung der Nodes, um sicherzustellen, dass die Platzierung von Pods die vorhandenen Ressourcen optimal nutzt.

11.3.3 Weitere nützliche Befehle und Informationen

Befehl	Beschreibung
`kubectl get pods –selector=<label-key>=<label-value>`	Listet alle Pods auf, die einem bestimmten Label-Selector entsprechen
`kubectl get nodes -o=jsonpath='{.items[?(@.metadata.labels.<label-key>==«label-value>")].metadata.name}'`	Listet Nodes anhand eines spezifischen Labels

11.3.4 Workloads auf spezifizierte Hardware beschränken

Soll ein Pod auf Nodes mit einer speziellen GPU-Hardware geplant werden kann den Nodes ein Label hinzugefügt werden und der Node-Selector in der Pod-Spezifikation definiert werden.

Ein Label zu Nodes hinzufügen

```
kubectl label node <node-name> gpu=true
```

Pod-Spezifikation mit Node Selector

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-gpu
spec:
  nodeSelector:
    gpu: "true"
  containers:
    - name: gpu-container
      image: nvidia/cuda:10.0-base
      resources:
        limits:
          nvidia.com/gpu: 1
```

11.3.5 Geografische Platzierung von Workloads

Um sicherzustellen, dass bestimmte Workloads in einer bestimmten geografischen Region ausgeführt werden, können Nodes entsprechend gelabelt und der Node Selector in der Pod-Spezifikation verwendet werden.

Ein Label zu Nodes hinzufügen

```
kubectl label node <node-name> region=us-west1
```

Pod-Spezifikation mit geografischem Node Selector

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-in-us-west1
spec:
  nodeSelector:
    region: us-west1
  containers:
    - name: nginx
      image: nginx
```

12 Bereitstellungsstrategien

12.1 Blue-Green Deployment

Blue-Green Deployment ist eine Methode, bei der zwei nahezu identische Umgebungen (Blue und Green) verwendet werden. Eine Umgebung dient dem aktuellen Produktionsverkehr, während die andere für die neue Version der Anwendung verwendet wird. Nach erfolgreicher Bereitstellung und Tests wird der Verkehr auf die neue Umgebung umgeschaltet.

Befehl	Beschreibung
`kubectl apply -f <blue-deployment.yaml>`	Die Blue-Umgebung bereitstellen
`kubectl apply -f <green-deployment.yaml>`	Die Green-Umgebung bereitstellen
`kubectl get services`	Services anzeigen, um den aktuellen Traffic zu überprüfen
`kubectl edit service <service-name>`	Den Service bearbeiten, um den Traffic auf die Green-Umgebung umzuleiten
`kubectl delete -f <blue-deployment.yaml>`	Die Blue-Umgebung nach erfolgreicher Migration löschen

12.2 Canary Releases

Canary Releases sind eine Methode, bei der eine neue Version der Anwendung nur für einen kleinen Teil der Benutzer bereitgestellt wird, um die neue Version unter realen Bedingungen zu testen. Nach und nach wird der Traffic auf die neue Version erhöht, bis sie vollständig übernommen wird.

Befehl	Beschreibung
`kubectl apply -f <canary-deployment.yaml>`	Die Canary-Version der Anwendung bereitstellen
`kubectl get pods -l app=<app-name>`	Alle Pods der Anwendung anzeigen
`kubectl scale deployment <canary-deployment> -replicas=<number>`	Die Anzahl der Replikate für die Canary-Version erhöhen oder verringern
`kubectl describe service <service-name>`	Details des Services anzeigen, um die Traffic-Verteilung zu überprüfen
`kubectl edit service <service-name>`	Den Service bearbeiten, um den Traffic auf die Canary-Version anzupassen
`kubectl delete -f <old-deployment.yaml>`	Die alte Version der Anwendung nach erfolgreicher Migration löschen

12.3 Rolling Updates

Rolling Updates ermöglichen es, eine neue Version einer Anwendung schrittweise bereitzustellen, indem Pods der alten Version durch Pods der neuen Version ersetzt werden. Dies stellt sicher, dass zu jeder Zeit eine minimale Anzahl von Pods verfügbar ist.

Befehl	Beschreibung
`kubectl set image deployment/<deployment-name> <container-name>=<new-image>`	Ein Rolling Update für ein Deployment initiieren
`kubectl rollout status deployment/<deployment-name>`	Den Status des Rollouts anzeigen
`kubectl rollout history deployment/<deployment-name>`	Die Rollout-Historie eines Deployments anzeigen
`kubectl rollout undo deployment/<deployment-name>`	Den letzten Rollout rückgängig machen

12.4 A/B Testing

A/B Testing ist eine Methode, bei der zwei oder mehr Versionen einer Anwendung parallel bereitgestellt werden, um deren Leistung zu vergleichen. Benutzer werden zufällig auf die verschiedenen Versionen verteilt, und die Ergebnisse werden analysiert, um die beste Version zu bestimmen.

Befehl	Beschreibung
`kubectl apply -f <version-a-deployment.yaml>`	Version A der Anwendung bereitstellen
`kubectl apply -f <version-b-deployment.yaml>`	Version B der Anwendung bereitstellen
`kubectl get services`	Services anzeigen, um die Traffic-Verteilung zu überprüfen
`kubectl edit service <service-name>`	Den Service bearbeiten, um den Traffic zwischen Version A und Version B zu verteilen
`kubectl get pods -l app=<app-name>`	Alle Pods der verschiedenen Versionen der Anwendung anzeigen

12.5 Strategiewahl und Best Practices

Die Wahl der richtigen Bereitstellungsstrategie hängt von verschiedenen Faktoren ab, darunter die Anforderungen an die Verfügbarkeit, das Risiko der neuen Version und die Fähigkeit des Teams, schnell auf Probleme zu reagieren. Hier sind einige Best Practices:

- **Blue-Green Deployment**: Gut geeignet für Anwendungen, die eine nahezu sofortige Umschaltung auf die neue Version erfordern und bei denen eine vollständige Parallelumgebung verfügbar ist.
- **Canary Releases**: Ideal, um neue Versionen schrittweise und kontrolliert auszurollen, insbesondere wenn das Risiko von Fehlern minimiert werden muss.
- **Rolling Updates**: Standardmethode für eine kontinuierliche Bereitstellung ohne Ausfallzeiten, geeignet für die meisten Anwendungen.
- **A/B Testing**: Nützlich, um verschiedene Versionen einer Anwendung unter realen Bedingungen zu vergleichen und datenbasierte Entscheidungen zu treffen.

12.6 Monitoring und Rollback

Unabhängig von der gewählten Strategie ist es wichtig, die neue Version kontinuierlich zu überwachen und bei Bedarf schnell zurückrollen zu können. Hier sind einige zusätzliche Befehle und Hinweise für das Monitoring und Rollback:

Befehl	Beschreibung
`kubectl logs <pod-name>`	Logs eines bestimmten Pods anzeigen, um Fehler zu diagnostizieren
`kubectl describe pod <pod-name>`	Detailinformationen zu einem bestimmten Pod anzeigen
`kubectl get events`	Aktuelle Ereignisse im Cluster anzeigen
`kubectl rollout undo deployment/<deployment-name>`	Den letzten Rollout einer neuen Version rückgängig machen
`kubectl delete pod <pod-name>`	Einen fehlerhaften Pod löschen, um ihn neu zu starten

13 Backup und Wiederherstellung

13.1 Backup-Strategien

Es gibt verschiedene Strategien, um Backups in Kubernetes durchzuführen:

- **ETCD-Backup**: Sicherung der ETCD-Datenbank, die den Zustand des gesamten Kubernetes-Clusters speichert.
- **Persistent Volume (PV) Backup**: Sicherung der Daten, die in Persistent Volumes gespeichert sind.
- **Anwendungs-Backup**: Sicherung der Anwendungsdaten und Konfigurationen.

13.2 ETCD-Backup

ETCD ist das Herzstück eines Kubernetes-Clusters. Ein Backup der ETCD-Datenbank stellt sicher, dass der Clusterzustand und die Konfigurationen wiederhergestellt werden können.

Befehl	Beschreibung
`ETCDCTL_API=3 etcdctl snapshot save <snapshot.db>`	Ein ETCD-Snapshot erstellen
`ETCDCTL_API=3 etcdctl snapshot restore <snapshot.db>`	Ein ETCD-Snapshot wiederherstellen
`ETCDCTL_API=3 etcdctl –write-out=table snapshot status <snapshot.db>`	Den Status eines ETCD-Snapshots überprüfen

13.3 Persistent Volume (PV) Backup

Daten, die in Persistent Volumes gespeichert sind, müssen regelmäßig gesichert werden. Die Methoden können je nach Storage-Provider variieren.

Befehl	Beschreibung
`kubectl get pv`	Alle Persistent Volumes im Cluster auflisten
`kubectl describe pv <pv-name>`	Details eines bestimmten Persistent Volumes anzeigen
`<storage-provider-specific-backup-command>`	Backup eines Persistent Volumes durchführen

13.4 Anwendungs-Backup

Anwendungsdaten und -konfigurationen können mit Tools wie Velero gesichert werden. Velero ist ein beliebtes Open-Source-Tool für Backup und Wiederherstellung von Kubernetes-Clustern.

Befehl	Beschreibung
`velero install –provider <provider> –bucket <bucket> –secret-file <credentials>`	Velero im Cluster installieren
`velero backup create <backup-name>`	Ein Backup des gesamten Clusters erstellen
`velero backup describe <backup-name>`	Details eines bestimmten Backups anzeigen
`velero backup delete <backup-name>`	Ein Backup löschen
`velero restore create –from-backup <backup-name>`	Ein Backup wiederherstellen
`velero restore describe <restore-name>`	Details einer Wiederherstellung anzeigen

13.5 Best Practices für Backup und Wiederherstellung

- **Regelmäßige Backups**: Planung und Automatisierung von regelmäßigen Backups
- **Backup-Überprüfung**: Regelmäßiges Testen der Backups
- **Offsite-Backups**: Speichern der Backups an Externen Orten
- **Sicherheitsmaßnahmen**: Schutz von Backups durch Verschlüsselung und Zugriffskontrollen, um Datenmissbrauch zu verhindern
- **Dokumentation und Schulung**: Backup- und Wiederherstellungsverfahren dokumentieren und Team regelmäßig schulen

14 CI/CD Integration

14.1 Einführung in CI/CD

CI/CD umfasst eine Reihe von Prozessen und Tools, die das Bauen, Testen und Bereitstellen von Anwendungen automatisieren. Diese Praktiken helfen, die Qualität des Codes zu verbessern, Fehler frühzeitig zu erkennen und die Bereitstellungsgeschwindigkeit zu erhöhen.
Es steht für „Continuous Integration“ und „Continuous Delivery“ oder „Continuous Deployment“.

14.1.1 Continuous Integration (CI)

Continuous Integration (CI) ist eine Softwareentwicklungspraxis, bei der Entwickler regelmäßig ihre Codeänderungen in ein zentrales Repository integrieren. Dies geschieht in der Regel mehrmals täglich. Jede Integration wird automatisch durch Builds und automatisierte Tests überprüft. Ziel ist es, Integrationsprobleme frühzeitig zu erkennen und zu beheben.
Wichtige Aspekte von CI:

- **Regelmäßige Code-Commits**: Entwickler committen ihren Code häufig, um Integration und Testen zu erleichtern.
- **Automatisierte Builds**: Jeder Commit löst einen automatisierten Build-Prozess aus, um sicherzustellen, dass der neue Code kompiliert und integrierbar ist.
- **Automatisierte Tests**: Nach dem Build werden automatisierte Tests ausgeführt, um sicherzustellen, dass der neue Code keine bestehenden Funktionalitäten bricht.
- **Schnelles Feedback**: Entwickler erhalten schnell Feedback zu ihren Änderungen, was die Fehlerbehebung beschleunigt.

14.1.2 Continuous Delivery (CD)

Continuous Delivery (CD) baut auf CI auf und stellt sicher, dass die Software jederzeit in einem zustellbaren Zustand ist. Nach jedem erfolgreichen Build und Test wird der Code automatisch in eine Staging-Umgebung bereitgestellt, wo er weiter getestet werden kann. Dies ermöglicht eine schnelle und zuverlässige Bereitstellung von Software in der Produktionsumgebung, sobald diese freigegeben wird.
Wichtige Aspekte von CD:

- **Automatisierte Bereitstellung**: Der Code wird automatisch in verschiedene Umgebungen (z.B. Staging, Testing) bereitgestellt.
- **Manuelle Freigabe**: Bevor der Code in die Produktion geht, kann eine manuelle Freigabe erforderlich sein, um zusätzliche Überprüfungen durchzuführen.
- **Wiederholbarkeit und Zuverlässigkeit**: Der Bereitstellungsprozess ist standardisiert und kann jederzeit reproduziert werden.
- **Kontinuierliches Testen**: Nach jeder Bereitstellung werden weitere Tests durchgeführt, um sicherzustellen, dass der Code in der neuen Umgebung funktioniert.

14.1.3 Continuous Deployment (CD)

Continuous Deployment (CD) geht einen Schritt weiter als Continuous Delivery. Hier wird der Code nach jedem erfolgreichen Build und Test automatisch in die Produktionsumgebung bereitgestellt, ohne dass eine manuelle Freigabe erforderlich ist. Dies ermöglicht eine sehr schnelle Bereitstellung von neuen Funktionen und Bugfixes, erfordert jedoch ein hohes Maß an Testautomatisierung und Überwachungsmechanismen.
Wichtige Aspekte von Continuous Deployment:

- **Automatisierte Produktionsbereitstellung**: Jede Codeänderung, die alle Tests besteht, wird automatisch in die Produktion übernommen.
- **Umfassende Testabdeckung**: Alle Tests, einschließlich Unit-, Integrations- und End-to-End-Tests, müssen automatisiert und zuverlässig sein.
- **Überwachung und Rollback**: Ein robustes Monitoring-System ist erforderlich, um Probleme in der Produktion schnell zu erkennen und bei Bedarf Rollbacks durchzuführen.
- **Feature Toggles**: Feature-Toggles können verwendet werden, um neue Funktionen bei Bedarf zu aktivieren oder zu deaktivieren, ohne dass ein erneuter Deployment erforderlich ist.

14.2 Beliebte CI/CD-Tools

Es gibt mehrere Tools, die sich nahtlos in Kubernetes integrieren lassen:

- **Jenkins**: Ein weit verbreitetes Open-Source CI/CD-Tool, das sich gut in Kubernetes integrieren lässt.
- **GitLab CI**: Ein integriertes CI/CD-Tool, das direkt in GitLab verfügbar ist.
- **Argo CD**: Ein deklaratives GitOps-Tool für Kubernetes.
- **Tekton**: Ein Kubernetes-natives CI/CD-Tool.

14.3 Einrichtung einer CI/CD-Pipeline mit Jenkins

Jenkins kann verwendet werden, um eine vollständige CI/CD-Pipeline für Kubernetes einzurichten.

14.3.1 Jenkins-Installation auf Kubernetes

Befehl	Beschreibung
`kubectl apply -f jenkins-deployment.yaml`	Jenkins im Kubernetes-Cluster bereitstellen
`kubectl get pods -l app=jenkins`	Jenkins-Pods anzeigen
`kubectl port-forward <jenkins-pod> 8080:8080`	Jenkins-Dashboard zugänglich machen

14.3.2 Erstellung einer Jenkins-Pipeline

Schritt	Beschreibung
`pipeline {`	Beginn der Pipeline-Definition
`agent any`	Verwenden eines beliebigen verfügbaren Agents
`stages {`	Beginn der Stufen-Definition
`stage('Build') {`	Build-Stufe definieren
`steps {`	Schritte der Build-Stufe
`sh 'kubectl apply -f deployment.yaml'`	Kubernetes-Deployment anwenden
`}`	Ende der Build-Stufe
`stage('Test') {`	Test-Stufe definieren
`steps {`	Schritte der Test-Stufe
`sh 'kubectl get pods'`	Pods im Cluster anzeigen
`}`	Ende der Test-Stufe
`}`	Ende der Stufen-Definition
`}`	Ende der Pipeline-Definition

14.4 Einrichtung einer CI/CD-Pipeline mit GitLab CI

GitLab CI bietet eine integrierte Lösung für CI/CD.

14.4.1 GitLab Runner Installation

Befehl	Beschreibung
`kubectl apply -f gitlab-runner.yaml`	GitLab Runner im Kubernetes-Cluster bereitstellen
`kubectl get pods -l app=gitlab-runner`	GitLab Runner-Pods anzeigen

14.4.2 Erstellung einer GitLab CI/CD-Pipeline

```yaml
stages:
  - build
  - test
  - deploy

build:
  stage: build
  script:
    - kubectl apply -f deployment.yaml

test:
  stage: test
  script:
    - kubectl get pods

deploy:
  stage: deploy
  script:
    - kubectl apply -f service.yaml
```

14.5 Einrichtung einer CI/CD-Pipeline mit Argo CD

Argo CD verwendet eine deklarative Methode für CI/CD, indem es den Zustand des Clusters mit dem Zustand im Git-Repository synchronisiert.

14.5.1 Argo CD-Installation

Befehl	Beschreibung
`kubectl create namespace argocd`	Namespace für Argo CD erstellen
`kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml`	Argo CD im Cluster installieren
`kubectl get pods -n argocd`	Argo CD-Pods anzeigen
`kubectl port-forward svc/argocd-server -n argocd 8080:443`	Argo CD-Dashboard zugänglich machen

14.5.2 Erstellung einer Argo CD-Applikation

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  project: default
  source:
    repoURL: 'https://github.com/my-repo/my-app.git'
    targetRevision: HEAD
    path: 'path/to/manifests'
  destination:
    server: 'https://kubernetes.default.svc'
    namespace: default
  syncPolicy:
    automated:
      selfHeal: true
      prune: true
```

14.6 Einrichtung einer CI/CD-Pipeline mit Tekton

Tekton ist ein Kubernetes-natives CI/CD-Tool, das es ermöglicht, CI/CD-Pipelines direkt in Kubernetes zu definieren und auszuführen.

14.6.1 Tekton-Installation

Befehl	Beschreibung
`kubectl apply –filename https://storage.googleapis.com/tekton-releases/pipeline/latest/release.yaml`	Tekton Pipelines im Cluster installieren
`kubectl get pods -n tekton-pipelines`	Tekton Pipelines-Pods anzeigen

14.6.2 Erstellung einer Tekton-Pipeline

```yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: my-pipeline
  namespace: default
spec:
  tasks:
    - name: build
      taskRef:
        name: build-task
    - name: deploy
      runAfter:
        - build
      taskRef:
        name: deploy-task
```

14.6.3 Definieren der Tekton-Tasks

Tekton Build Task:

```yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: build-task
spec:
  steps:
    - name: build
      image: golang:1.13
      script: |
        go build -o my-app .
```

Tekton Deploy Task:

```yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: deploy-task
spec:
  steps:
    - name: deploy
      image: bitnami/kubectl
      script: |
        kubectl apply -f deployment.yaml
```

14.7 Best Practices für CI/CD in Kubernetes

- **Automatisierte Tests**: Unit-Tests, Integrationstests und End-to-End-Tests in CI/CD-Pipeline integrieren, um die Codequalität sicherzustellen
- **Sicherheitsüberprüfungen**: Sicherheitsüberprüfungen durchführen, um Schwachstellen und Sicherheitsrisiken frühzeitig zu erkennen
- **Ressourcenoptimierung**: Sicherstellen dass die CI/CD-Pipeline Ressourcen effizient nutzt, um Kosten zu minimieren und die Leistung zu maximieren
- **Monitoring und Alerts**: Monitoring und Alerts implementieren, um Probleme in der Pipeline frühzeitig zu erkennen und schnell darauf reagieren zu können.
- **Rollback-Strategien**: Rollback-Strategien einplanen, für den Fall, dass ein Deployment fehlschlägt, um die Auswirkungen auf die Produktion zu minimieren
- **Dokumentation und Schulung**: CI/CD-Pipelines dokumentieren und Team regelmäßig schulen, um sicherzustellen, dass alle Mitglieder die Prozesse und Tools verstehen und effektiv nutzen können

15 Kompendium

Dieses Kapitel enthält eine kompakte Übersicht nützlicher Befehle, Werkzeuge und Konfigurationen für den Kubernetes-Alltag. Die Tabellen sind so aufgebaut, dass Befehle direkt verwendet oder angepasst werden können.

Hinweis

Ziel ist es, typische Aufgaben schnell umzusetzen.

15.1 Grundbefehle

Befehl	Beschreibung
`kubectl run <name>`	Erstellt und startet einen neuen Pod
`kubectl get <resource>`	Zeigt eine Liste der Ressourcen an
`kubectl describe <resource> <name>`	Zeigt detaillierte Informationen über eine Ressource
`kubectl create -f <file.yaml>`	Erstellt Ressourcen aus einer Konfigurationsdatei
`kubectl apply -f <file.yaml>`	Aktualisiert eine Ressource aus einer Konfigurationsdatei
`kubectl delete <resource> <name>`	Löscht eine Ressource
`kubectl logs <pod-name>`	Zeigt die Logs eines Pods
`kubectl exec -it <pod-name> – <command>`	Führt einen Befehl in einem laufenden Pod aus
`kubectl expose <resource> –port=<port>`	Erstellt einen neuen Service
`kubectl proxy`	Startet einen lokalen Proxy zum API-Server

15.2 Allgemeine Befehle

Befehl	Beschreibung
`kubectl version`	Version von kubectl anzeigen
`kubectl get nodes`	Alle Knoten im Cluster auflisten
`kubectl cluster-info`	Cluster-Informationen anzeigen
`kubectl config view`	Konfiguration anzeigen
`kubectl create`	Erstellt eine Ressource
`kubectl delete`	Löscht eine Ressource

15.3 Pods

Befehl	Beschreibung
`kubectl get pods`	Alle Pods im Standard-Namespace auflisten
`kubectl get pods -n <namespace>`	Alle Pods in einem bestimmten Namespace auflisten
`kubectl get pods -o wide`	Alle Pods mit Details auflisten
`kubectl describe pod <pod-name>`	Details zu einem bestimmten Pod anzeigen
`kubectl logs <pod-name>`	Logs eines Pods anzeigen
`kubectl delete pod <pod-name>`	Einen Pod löschen

15.4 Deployments

Befehl	Beschreibung
`kubectl get deployments`	Alle Deployments auflisten
`kubectl describe deployment <deployment-name>`	Details zu einem Deployment anzeigen
`kubectl delete deployment <deployment-name>`	Ein Deployment löschen
`kubectl rollout restart deployment <deployment-name>`	Deployment neu starten

15.5 Services

Befehl	Beschreibung
`kubectl get services`	Alle Services auflisten
`kubectl describe service <service-name>`	Details zu einem Service anzeigen
`kubectl delete service <service-name>`	Einen Service löschen

15.6 Namespaces

Befehl	Beschreibung
`kubectl get namespaces`	Alle Namespaces auflisten
`kubectl describe namespace <namespace-name>`	Details zu einem Namespace anzeigen
`kubectl create namespace <namespace-name>`	Neuen Namespace erstellen
`kubectl delete namespace <namespace-name>`	Einen Namespace löschen

15.7 Konfiguration

Befehl	Beschreibung
`kubectl config get-contexts`	Alle Kontexte auflisten
`kubectl config current-context`	Aktuellen Kontext anzeigen
`kubectl config use-context <context-name>`	Kontext wechseln
`kubectl config get-clusters`	Alle in der kubeconfig definierten Cluster anzeigen
`kubectl config get-contexts`	Alle Kontexte anzeigen