

Kubernetes im Einsatz

helm, Monitoring, Fehlersuche und Troubleshooting

Lernziele

- ★ Sie haben einen Überblick über Erweiterte Funktionen im Kubernetes Cluster.

Zeitlicher Ablauf

- ★ Continuous Integration (CI) und Delivery (CD)
- ★ Helm - der Paketmanager für Kubernetes*
- ★ Monitoring*
- ★ Logging
- ★ Kubernetes API - Zugriffssteuerung
- ★ Namespaces begrenzen
- ★ Fehlersuche und Troubleshooting*
- ★ Reflexion
- ★ Lernzielkontrolle

* jeweils mit anschliessender Übung.

Continuous Integration und Delivery - Definition

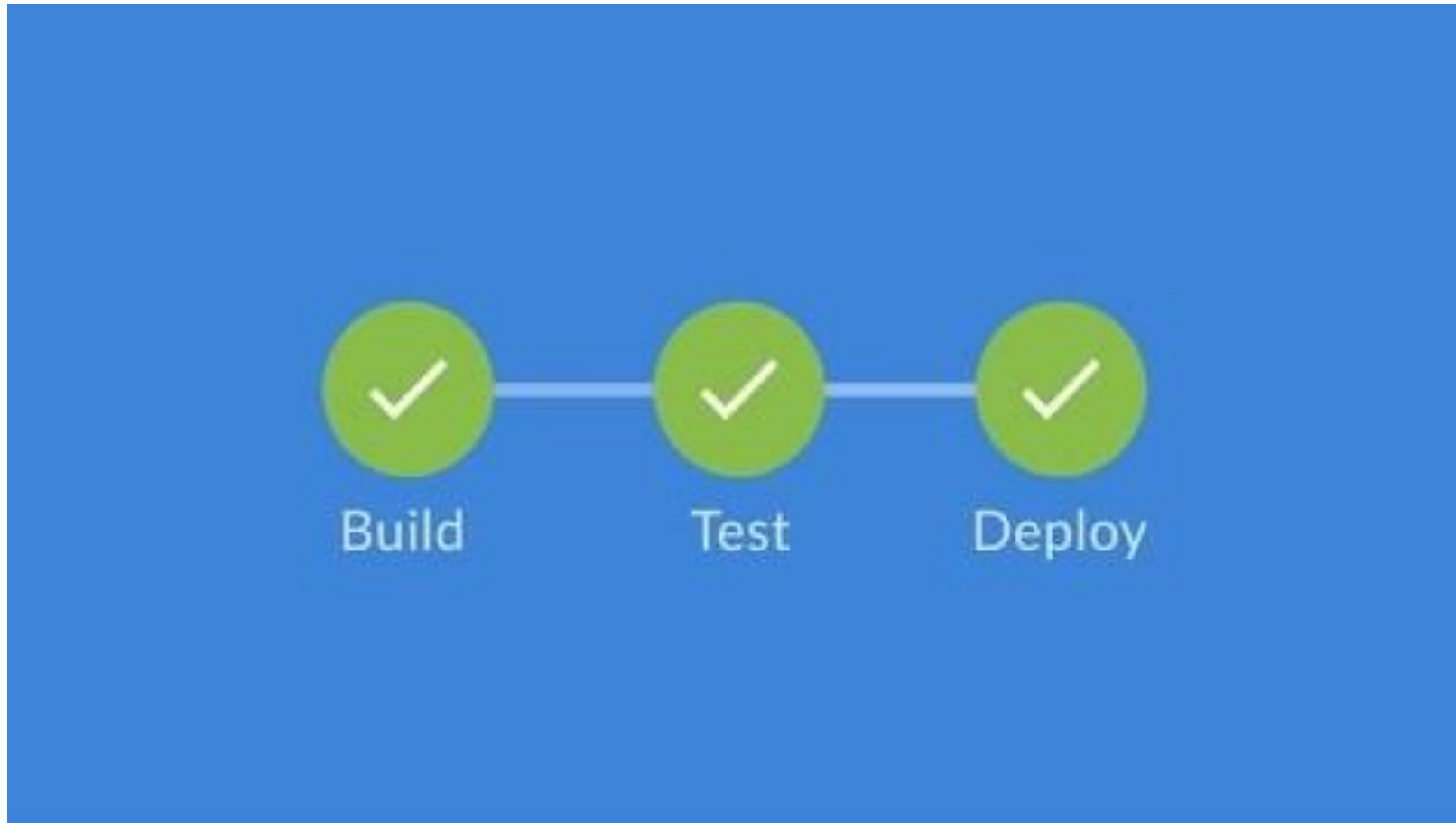
- ★ **Continuous Integration (CI)** (auch *fortlaufende* oder *permanente Integration*) ist ein Begriff aus der Software-Entwicklung, der den Prozess des fortlaufenden Zusammenfügens von Komponenten zu einer Anwendung beschreibt.
- ★ **Continuous Delivery (CD)** bezeichnet eine Sammlung von Techniken, Prozessen und Werkzeugen, die den Softwareauslieferungsprozess (englisch: Delivery) verbessern.

Jenkins

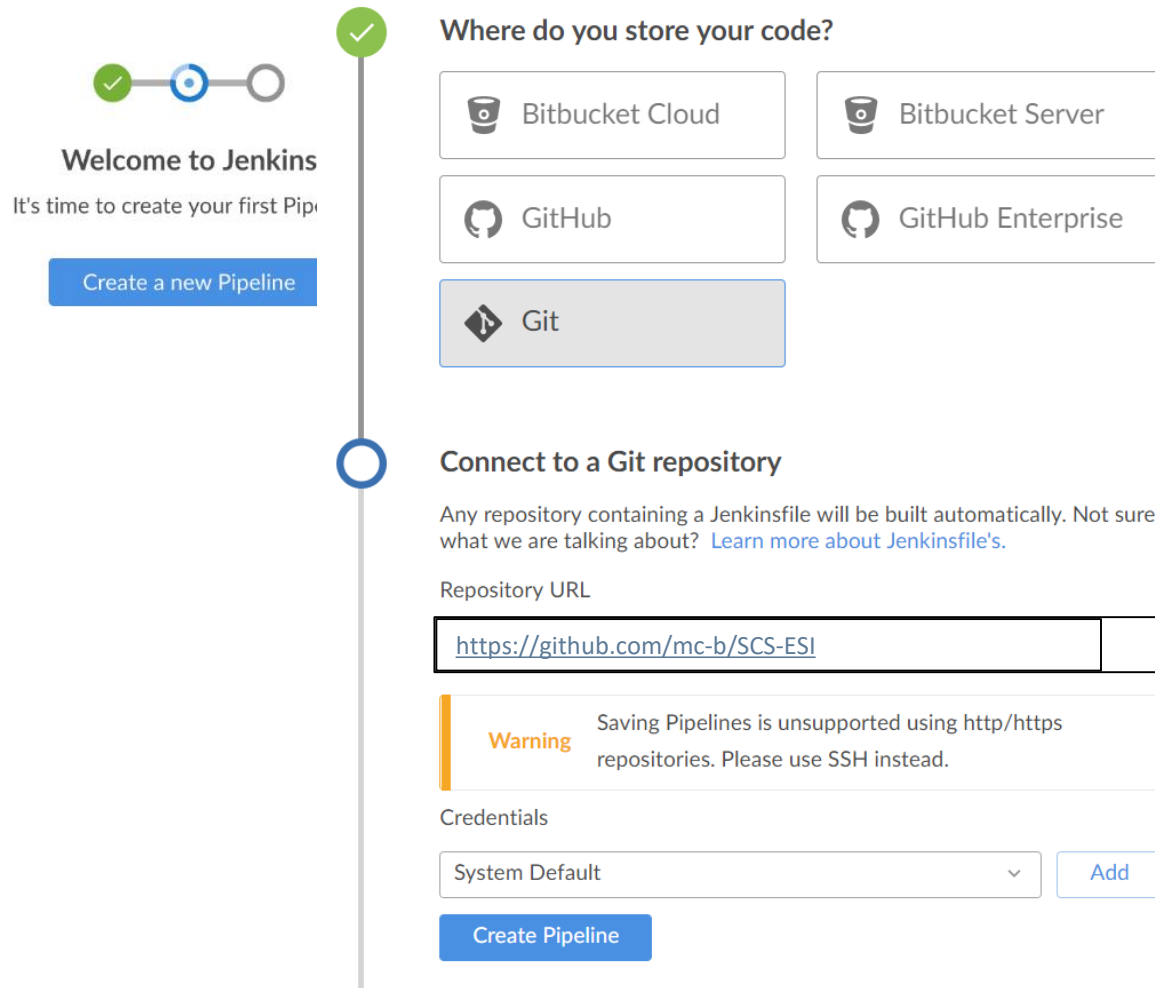
All +					
S	W	Name ↓	Last Success	Last Failure	
		Freestyle project	29 sec - #1	N/A	
		Github Org project	N/A	20 sec - log	
		Multibranch	N/A	N/A	
		Other	N/A	N/A	

- ★ Jenkins ist ein erweiterbares, [webbasiertes Software](#)-System zur [kontinuierlichen Integration](#) von Komponenten zu einem [Anwendungsprogramm](#).
- ★ Es wird als [Fork](#) der Software [Hudson](#) von Sun, heute Oracle, betrachtet.
- ★ Unterstützt werden verschiedene [Build-Tools](#) wie [Apache Ant](#), [Maven](#) oder [Gradle](#), [Versionsverwaltungssysteme](#) wie [CVS](#), [Subversion](#) oder [Git](#), automatische Testverfahren ("test tools") wie [JUnit](#) oder [Emma](#).
- ★ Durch verschiedene Zusatzmodule ("Plugins") können auch andere Compiler gesteuert werden, sodass neben Java- auch [PHP](#)-, [Ruby](#)- oder [.NET](#)-basierte Projekte verwaltet werden können.
- ★ Jenkins verfügt über eine [REST](#)-basierte [Programmierschnittstelle](#) zur Steuerung durch andere Programme.
- ★ Quelle: [Wikipedia](#)

Jenkins Blue Ocean



✎ Übung: Continuous Integration mit Jenkins (1)



The image shows the Jenkins 'Create a new Pipeline' wizard. It has a progress bar on the left with three steps: 'Welcome to Jenkins' (completed), 'Where do you store your code?' (current), and 'Connect to a Git repository' (next). The 'Where do you store your code?' section offers five options: Bitbucket Cloud, Bitbucket Server, GitHub, GitHub Enterprise, and Git (selected). Below this, the 'Connect to a Git repository' section provides instructions and a text field for the 'Repository URL' containing 'https://github.com/mc-b/SCS-ESI'. A warning message states: 'Warning: Saving Pipelines is unsupported using http/https repositories. Please use SSH instead.' At the bottom, there is a 'Credentials' dropdown set to 'System Default' and a 'Create Pipeline' button.

Welcome to Jenkins
It's time to create your first Pipeline

Create a new Pipeline

Where do you store your code?

- Bitbucket Cloud
- Bitbucket Server
- GitHub
- GitHub Enterprise
- Git**

Connect to a Git repository

Any repository containing a Jenkinsfile will be built automatically. Not sure what we are talking about? [Learn more about Jenkinsfile's.](#)

Repository URL

<https://github.com/mc-b/SCS-ESI>

Warning Saving Pipelines is unsupported using http/https repositories. Please use SSH instead.


Credentials

System Default

Add

Create Pipeline

★ Jenkins mit einer Pipeline aufsetzen

- Jenkins, im CLI, wie folgt starten:
- `kubectl create -f duk/devops/jenkins.yaml`
- Jenkins Oberfläche mittels <http://localhost:32100> anwählen.
- Username/Password: admin
- Auf  Open Blue Ocean wechseln
- Neue Pipeline mittels Git und Repository URL <https://github.com/mc-b/SCS-ESI> anlegen.
- Prüfen ob Container Images misegr/scsesi* erstellt wurden:

★ `docker image ls`

★ Weiteres Beispiel siehe: <https://github.com/mc-b/bpmn-tutorial/blob/master/Jenkinsfile>

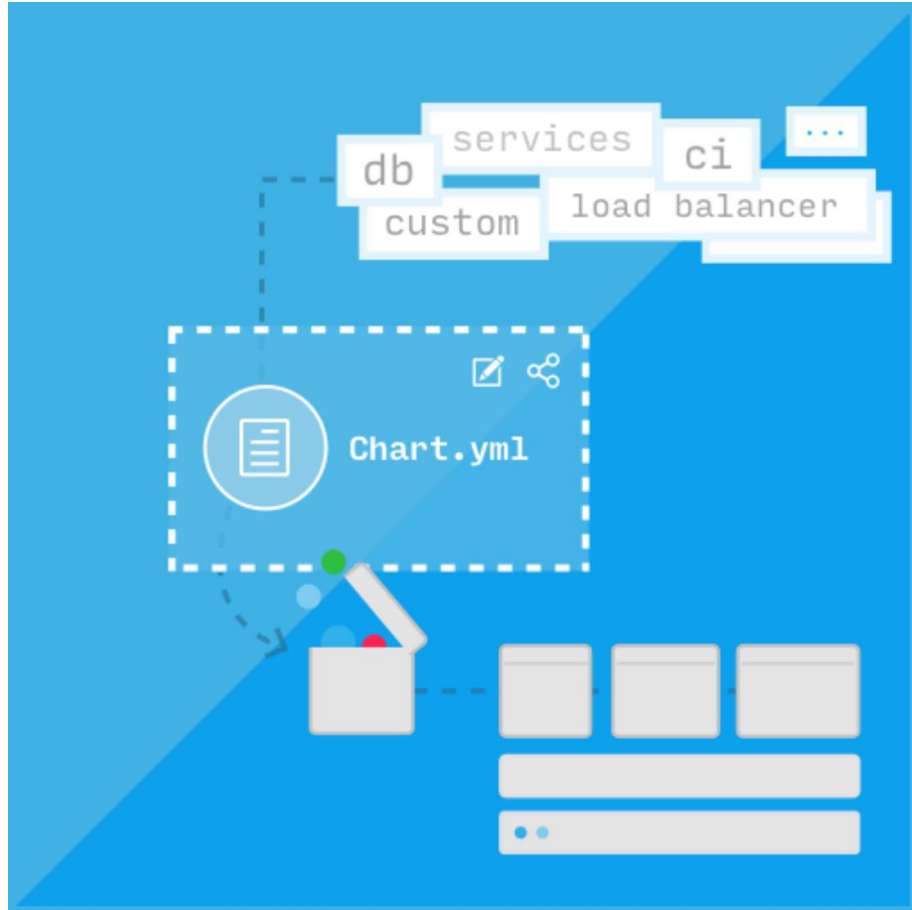
✎ Übung: Continuous Integration mit Jenkins (2)



```
pipeline {
  agent none
  stages {
    stage('Build') {
      agent {
        docker {
          image 'maven:3-alpine'
          args '-v /root/.m2:/root/.m2'
        }
      }
      steps {
        sh 'cd scs-demo-esi-order/ && mvn -B -DskipTests clean package'
        archiveArtifacts 'scs-demo-esi-order/target/*.jar'
      }
    }
    stage('Test') {
      agent {
        docker {
          image 'maven:3-alpine'
          args '-v /root/.m2:/root/.m2'
        }
      }
      steps {
        sh 'cd scs-demo-esi-order/ && mvn test'
      }
      post {
        always {
          junit 'scs-demo-esi-order/target/surefire-reports/*.xml'
        }
      }
    }
    stage('Build Images') {
      agent any
      steps {
        sh 'ls -l'
        sh 'cd docker/varnish && /usr/bin/docker build -t scsesi_varnish .'
        sh 'cd scs-demo-esi-common && /usr/bin/docker build -t scsesi_common .'
        sh 'cd scs-demo-esi-order && /usr/bin/docker build -t scsesi_order .'
      }
    }
  }
}
```

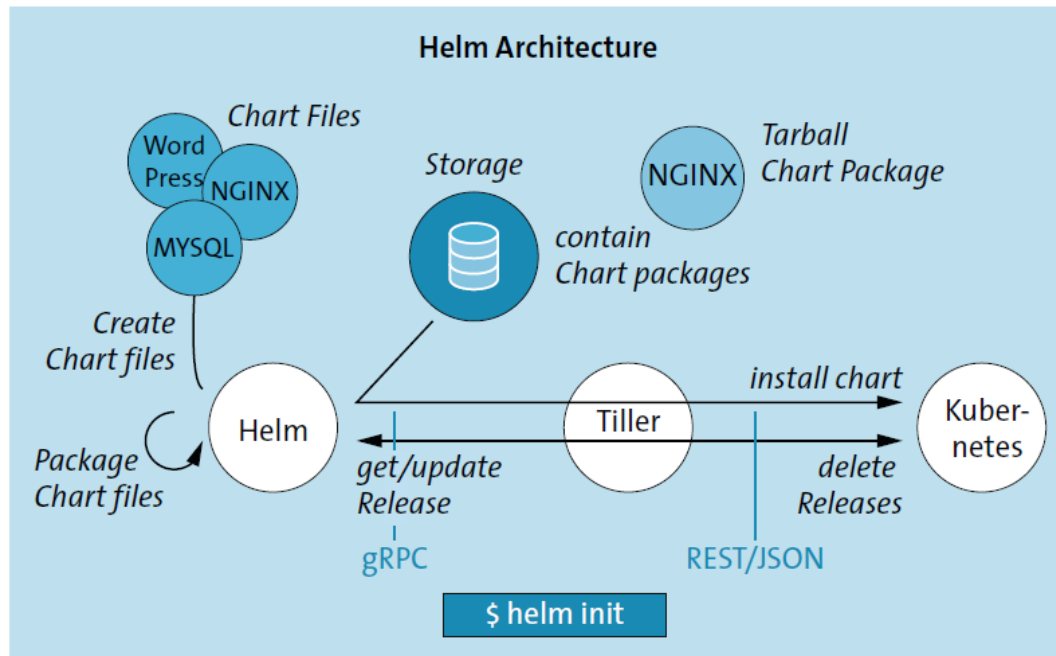
Jenkinsfile für Projekt <https://github.com/mc-b/SCS-ESI>

Helm - der Paketmanager? für Kubernetes



- ★ **Helm** hilft Ihnen bei der Verwaltung von Kubernetes-Anwendungen - mit Helm Charts können Sie selbst die komplexeste Kubernetes-Anwendung definieren, installieren und aktualisieren.
- ★ **Charts** sind einfach zu erstellen, zu versionieren, zu teilen und zu veröffentlichen - beginnen Sie also mit Helm und stoppen Sie den Copy-and-Paste-Wahnsinn.
- ★ Die neueste Version von Helm wird von der CNCF verwaltet - in Zusammenarbeit mit Microsoft , Google , Bitnami und der Helm-Community .

Helm - Funktionsweise



- ★ Helm unterteilt sich in zwei primäre Komponenten:
 - Helm-Client als lokales CLI Tools
 - Tiller-Server auf dem K8s Cluster
- ★ Tiller-Server, läuft in der Regel, innerhalb unseres K8s-Clusters und setzt als automatischer Deployer die Anforderungen der Charts in Workloads / Deployments um.
- ★ Standard Helm Repository:
 - <https://github.com/helm/charts>

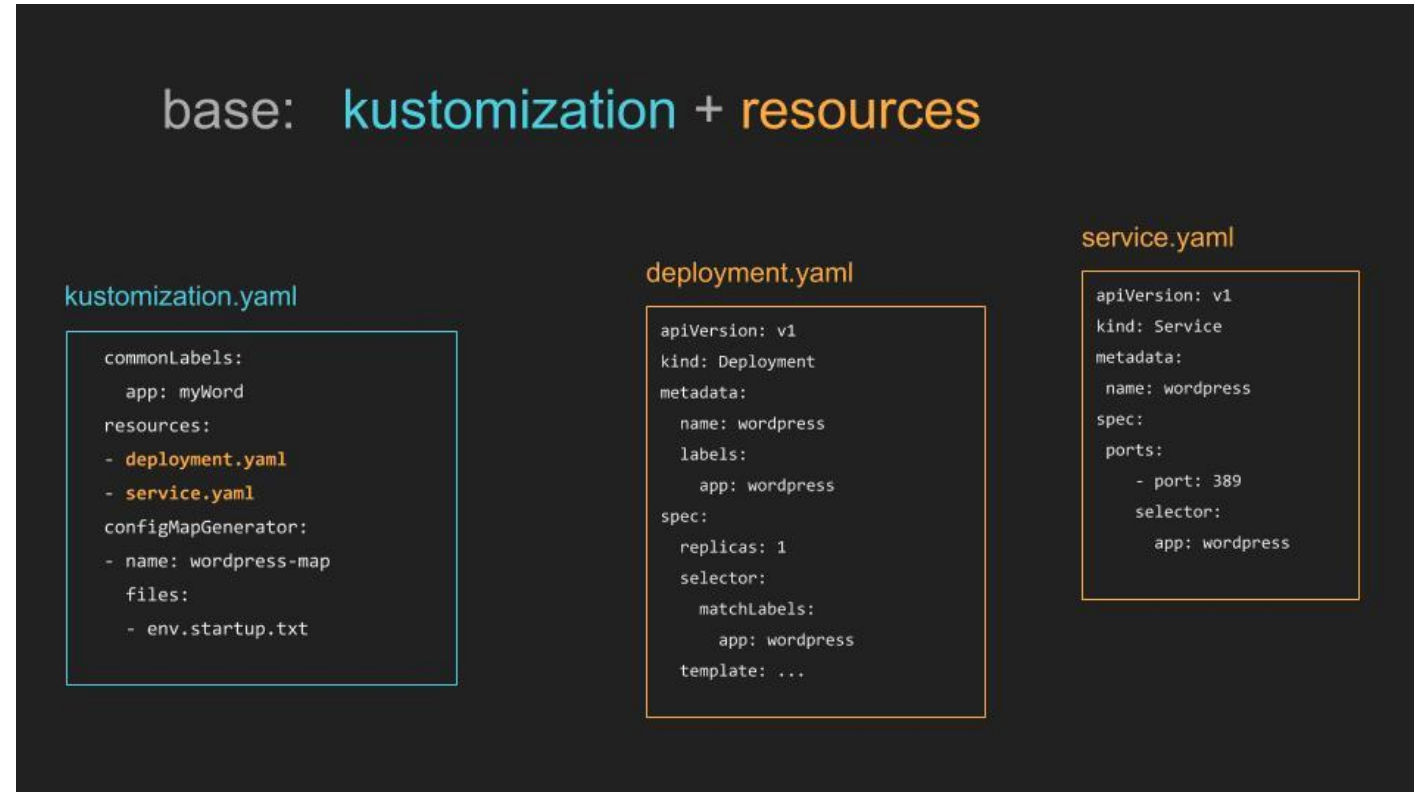


Übung: Helm

- ★ Erstellt die nötigen Services Accounts
 - <https://github.com/mc-b/duk/tree/master/helm> (1. Teil Installation)
- ★ Spielt das Beispiel für die Installation der MySQL Datenbank, mit angepassten Werten, durch:
 - <https://github.com/mc-b/duk/tree/master/mysql/helm>
- ★ **Anmerkungen:**
 - Das helm CLI Tools wurde beim der Installation des K8s Cluster bereits installiert und befindet sich im Verzeichnis lernkube/bin.
 - Ab helm V3.0 braucht es Tiller nicht mehr.

Kustomize

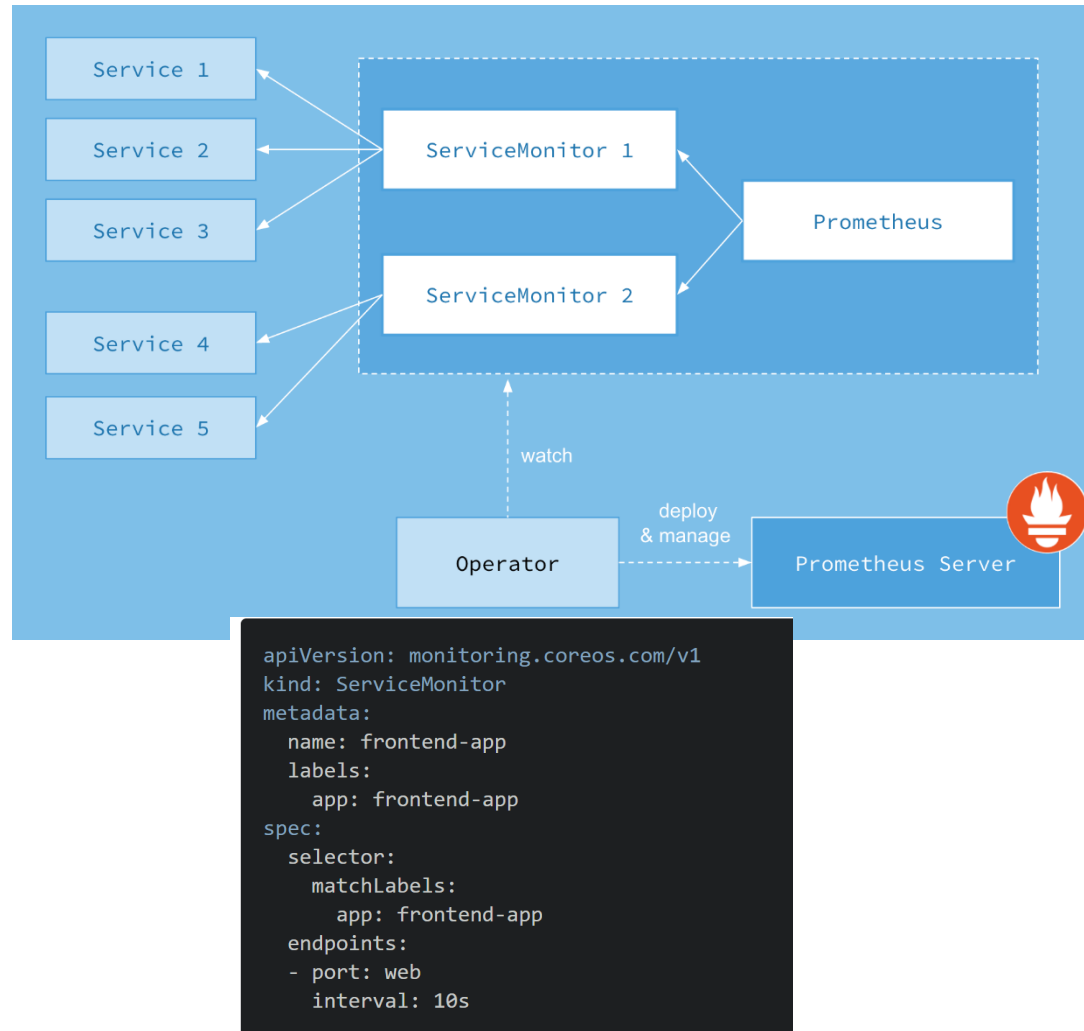
- ★ Mit kustomize kann man YAML-Dateien für mehrere Zwecke anpassen.
- ★ Die ursprüngliche YAML bleibt dabei unverändert.
- ★ `kubectl apply -k .` – integriert in Kubectl
- ★ Dokumentation: <https://github.com/kubernetes-sigs/kustomize>
- ★ Beispiele: <https://github.com/kubernetes-sigs/kustomize/tree/master/examples>



Monitoring - Vorbetrachtungen

- ★ Manuelles Überwachen ständig sterbender, neu gestarteter sowie veränderter Services, Container und sonstiger Ressourcen im Container-Cluster stellt komplett neue Anforderungen an das Monitoring.
- ★ Setups wie in konventionellen Sysadmin-Umgebungen auf Blech oder VMs sind hier weder praktikabel oder sinnvoll noch in irgendeiner Form realisierbar.

Monitoring -Prometheus



- ★ Prometheus
 - Monitoring Lösung für K8s
 - CNCF Projekt
- ★ Prometheus Aufgaben
 - Proaktive Überwachung
 - Sichtbarkeit und Kapazitätsplanung von Clustern
 - Alarme und Benachrichtigung auslösen
 - Metrics-Dashboards
- ★ Prometheus Service Monitor
 - Erweiterung K8s
 - Legt, mittels Selektoren, fest was überwacht wird.



Übung: Prometheus

- ★ Installation Prometheus: <https://github.com/mc-b/duk/tree/master/prometheus#installation>
- ★ Startet das Dashboard, wechselt auf Namespace «monitoring» und Ändert in der Konfiguration des Service «kube-prometheus» den Eintrag «spec.type» auf «NodePort». Dito für Service «kube-prometheus-grafana».
- ★ Öffnet einen Browser und gibt als URL die K8s Master IP und den Port (Beides im Dashboard Ersichtlich) an.
- ★ Die Informationen zum K8s Cluster findet ihr unter «Status -> Targets».
- ★ Eine Ausgabe der neu Erstellten Ressourcen erhält man mittels:
 - `kubectl get CustomResourceDefinition`
- ★ Beispielanwendung laut <https://coreos.com/blog/the-prometheus-operator.html> starten
 - `kubectl create -f duk/prometheus`
- ★ Wechseln auf <http://localhost:30100>

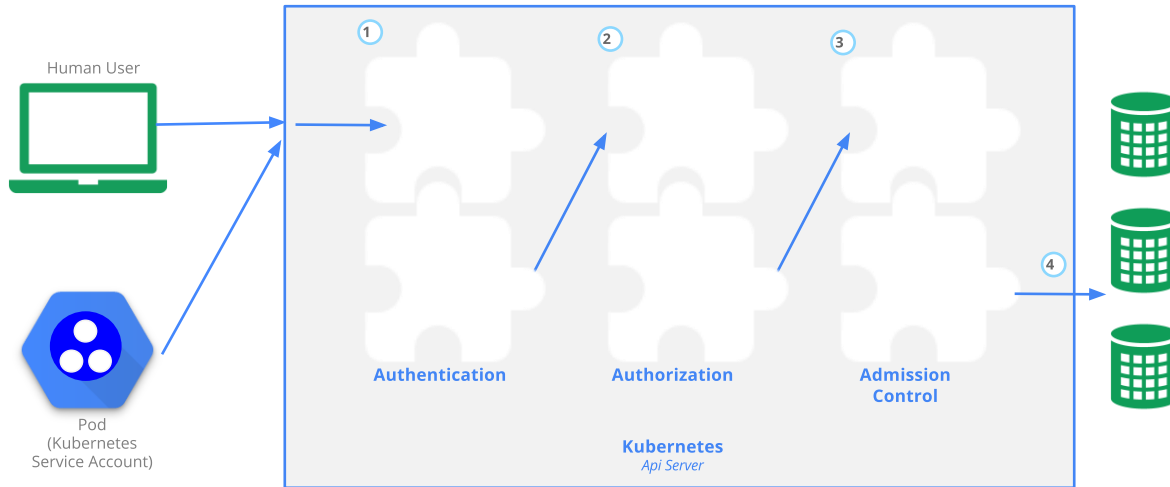
Logging: fluentd und fluentbit



- ★ CNCF Projekt(s)
- ★ <https://www.fluentd.org/>, Hauptprojekt
- ★ <https://fluentbit.io/>, Teil des fluentd Ökosystems
 - Kubernetes Logging with Fluent Bit, <https://github.com/fluent/fluent-bit-kubernetes-logging>

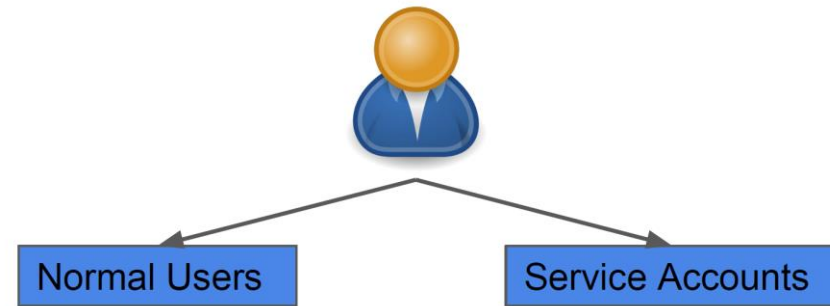


Kubernetes API - Zugriffssteuerung



1. Authentifizierung
 - Ist der User berechtigt einzuloggen?
2. Autorisierung
 - Darf der User die gewünschte Aktion durchführen?
3. Admission Control
 - Handelt es sich um eine gültige Anforderung?
4. Zugriff auf die Ressource

Kubernetes-»User«



- ★ Alle Kubernetes-Cluster haben zwei Kategorien von Usern: von Kubernetes verwaltete Dienstkonten und normale User.
- ★ Es wird davon ausgegangen, dass normale User von einem externen, unabhängigen Dienst verwaltet werden. Ein Administrator, der private Schlüssel verteilt, ein Userspeicher wie OpenStacks Keystone, Google-Konten, ein Identity Framework wie [spiffe](#) oder eine Datei mit einer Liste von Usernamen und Kennwörtern.
- ★ **Kubernetes verfügt nicht über Objekte, die normale Userkonten darstellen. Normale User können nicht über einen API-Aufruf zu einem Cluster hinzugefügt werden.**
- ★ Kubernetes unterstützt folgende **Authentifizierungsstrategien**
 - X509-Client-Zertifikate
 - Inhaber-Token
 - einen Authentifizierungs-Proxy
 - HTTP-Basisauthentifizierung, um API-Anforderungen über Authentifizierungs-Plug-ins zu authentifizieren.

Ressourcen

- ★ Ressourcen von Kubernetes werden über das API angesprochen.
- ★ Der URL für das Ansprechen eines Pods lautet z.B.:
 - `GET /api/v1/namespaces/{namespace}/pods/{name}`
- ★ Einige Ressourcen enthalten jedoch Subressourcen. Der Name der Subresource wird hinten angehängt, z.B.:
 - `GET /api/v1/namespaces/{namespace}/pods/{name}/log`
- ★ Die Ressourcen bzw. Subressourcen sind in der Rollen Beschreibung anzugeben.
- ★ Eine Liste aller Ressourcen kann wie folgt angezeigt werden:
 - `kubectl api-resources`

Verbs

- ★ Die Kubernetes-API ist eine ressourcenbasierte (RESTful) Programmierschnittstelle, die über HTTP bereitgestellt wird. Es unterstützt das Abrufen, Erstellen, Aktualisieren und Löschen von Ressourcen über die Standard-HTTP-Verben (POST, PUT, PATCH, DELETE, GET).
- ★ Die Verben sind in der Rollen Beschreibung anzugeben und steuern die Aktionen die auf Ressourcen angewandt werden können.

Rollen und RBAC-Autorisierung

- ★ Die rollenbasierte Zugriffssteuerung (RBAC) ist eine Methode zur Regulierung des Zugriffs auf Computer- oder Netzwerkressourcen basierend auf den Rollen einzelner Benutzer in einem Unternehmen.

Role	ClusterRole
<i>“Applicable to a given namespace only.”</i>	<i>“Applicable Cluster Wide.”</i>
kind: Role apiVersion: rbac.authorization.k8s.io/v1 metadata: namespace: cloudyuga name: deployment-manager rules: - apiGroups: ["" , "apps"] resources: ["deployments", "replicasets", "pods"] verbs: ["get", "list", "watch", "create", "update"]	kind: ClusterRole apiVersion: rbac.authorization.k8s.io/v1 metadata: name: deployment-manager-cluster rules: - apiGroups: ["" , "apps"] resources: ["deployments", "replicasets", "pods"] verbs: ["get", "list", "watch", "create", "update"]

Einfaches RBAC-Beispiel Kubernetes-»User«

★ Ziel

- User `snoopy` mit Begrenzten Zugriffsrechten auf Namespace `default`.

★ Vorgehen

- Anlegen des User Zertifikates und beglaubigen durch Kubernetes mittels `openssl` und `kubectl`.
- Erstellen einer K8s-Konfiguration
- Role und RoleBinding erzeugen
- Details siehe: <https://github.com/mc-b/duk/tree/master/rbac>

★ Resultat

- K8s-Konfiguration mit User Informationen und gewünschtem Zugriff.
- Verwenden mittels:
 - ★ `KUBECONFIG=./kube/config-snoopy`
 - ★ `kubectl get pods`
- Siehe auch: [Using kubeconfig Files](#)

```
apiVersion: v1
clusters:
- cluster:
    insecure-skip-tls-verify: true
    server: https://192.168.137.100:6443
    name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: snoopy
    name: snoopy
current-context: snoopy
kind: Config
preferences: {}
users:
- name: snoopy
  user:
    client-certificate-data: LS0tLS1CRUdJ
```

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: default
  name: pod-reader-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: pod-reader-rolebinding
  namespace: default
subjects:
- kind: User
  name: snoopy
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader-role
  apiGroup: rbac.authorization.k8s.io
```

Namespaces begrenzen

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
spec:
  limits:
  - default:
      memory: 512Mi
    defaultRequest:
      memory: 256Mi
    type: Container
```

- ★ Standardspeicheranforderungen und -beschränkungen für einen Namespace
 - Wenn ein Container in einem Namespace mit einem Standardspeicherlimit erstellt wird und der Container kein eigenes Speicherlimit angibt, wird dem Container das Standardspeicherlimit zugewiesen.
 - Gleiche Möglichkeiten bestehen für CPU Beschränkungen und Memory und CPU Kontingente (Quotas).
 - Details: <https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/memory-default-namespace/>
- ★ Ebenfalls können Pod Kontingente (Quotas) für Namespaces konfiguriert werden.
 - Details: <https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-pod-namespace/>

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pod-demo
spec:
  hard:
    pods: "2"
```

Debugging/Troubleshooting (1)

- ★ Troubleshooting und Debugging allein würde ein weiteres komplettes Buch füllen.
- ★ Zudem wäre selbst ein umfangreicher Debugging-Leitfaden immer nur so aktuell wie die vorliegende K8s-Version.
- ★ Daher ist einer der flexibelsten Ansätze, die relativ gut dokumentierten Debugging und Troubleshooting-Seiten des K8s Projekts aufzurufen:
 - **Allgemeine Troubleshooting- und Debugging-FAQs und generelle Verweise**
 - ★ <https://github.com/kubernetes/kubernetes/wiki/Debugging-FAQ>
 - ★ <http://kubernetes.io/docs/troubleshooting/>
 - ★ <https://kubernetes.io/docs/tasks/debug-application-cluster/>
 - **Cluster-Troubleshooting**
 - ★ <http://kubernetes.io/docs/admin/cluster-troubleshooting/>
 - **Pods in Produktivumgebungen debuggen**
 - ★ <http://kubernetes.io/docs/user-guide/production-pods/>

Debugging/Troubleshooting (2)

★ Pods und ReplicaController debuggen

- <http://kubernetes.io/docs/user-guide/debugging-pods-and-replication-controllers/>

★ Services debuggen

- <https://kubernetes.io/docs/tasks/debug-application-cluster/local-debugging/>

★ Applikationsspezifisches Debugging in den Pods

- <https://kubernetes.io/docs/tasks/debug-application-cluster/debug-application-introspection/>

★ Audits

- <https://kubernetes.io/docs/tasks/debug-application-cluster/audit/>

★ Zustand der Ressourcen und des Cluster als Dump (inkl. Log pro Pod)

- `kubectl cluster-info dump --output-directory=k8sdump`



Reflexion

- ★ **CI/CD** ist, mit den entsprechenden Erweiterungen (z.B. Jenkins und Jenkinsfile), einfach in Kubernetes zu integrieren.
- ★ **Helm** hilft Ihnen bei der Verwaltung von Kubernetes-Anwendungen - mit Helm Charts können Sie selbst die komplexeste Kubernetes-Anwendung definieren, installieren und aktualisieren.
- ★ **Monitoring** unterscheidet sich bei Kubernetes, u.a. wegen der kurz lebigkeit, vom klassischen Ansatz. Prometheus und andere Tools füllen langsam die Lücke.
- ★ Der Zugriff auf das **Kubernetes API** und **Namespaces** können begrenzt werden.
- ★ **Troubleshooting und Debugging** allein würde ein weiteres komplettes Buch füllen, besser die Debugging und Troubleshooting-Seiten des K8s Projekts aufzurufen.
- ★ Für alle weiteren Anforderungen, wie **Logging, Workflow, Serverless** etc., entstehen fast täglich neue Produkte rund um Kubernetes.

? Lernzielkontrolle

- ★ Sie haben einen Überblick über Erweiterte Funktionen im Kubernetes Cluster.