Helm Paketmanagement

Agenda

1. Kubernetes
    - Architektur Kubernetes
    - Aufbau von Browser zu Applikation - Schaubild
2. Helm Einfuehrung
    - Was ist helm ?
    - Was kann helm ?
    - Was ist in helm ein chart?
    - Warum Helm in Kubernetes verwenden ?
    - Überblick über den Ablauf bei der Nutzung von helm (Kommando: install)
    - Braucht helm das Programm kubectl ?
3. Helm Installation und Konfiguration (inkl. kubectl)
    - Installation von kubectl unter Linux
    - Konfiguration von kubectl mit namespaces
    - Installation von helm unter Linux
    - Installation bash completion
4. Helm - Spickzettel
    - Wichtig: Helm Spickzettel
5. Helm - Tipps & Tricks
    - Helm values.yaml autogenerieren
6. Arbeiten mit helm - charts (Basics)
    - Installation, Upgrade, Uninstall helm-Chart exercise
    - Nur fertiges manifest ausgeben ohne Installation
    - Informationen aus nicht installierten Helm-Charts bekommen
    - Chart runterladen und evtl. entpacken und bestimmte Version
    - Aufräumen von CRD's nach dem Deinstallieren
7. Arbeiten mit helm - charts (Debugging)
    - Nur fertiges manifest ausgeben ohne Installation
    - Chart trocken testen gegen api-server ohne Installation --dry-run
8. Helm Internals
    - Helm template - Rendering Prozess
    - helm vs. kubectl vs. oc
9. Helm - best practices
    - Wann quotes in yaml und in resources (Kubernetes/OCP)
    - Gute Struktur für Values und Charts
    - Best Practices for Chart Templating
10. Helm - Advanced

- Helm Dependencies Exercise

11. Helm Grundlagen
    - TopLevel Objekte
12. Helm Charts entwickeln
    - eigenes helm chart erstellen (Gruppe)
    - Wie starte ich am besten ganz einfach - Übung
13. Spezial: Umgang mit Einrückungen
    - Whitespaces meistern mit "-"
    - Exercise Whitespaces
14. Type - Conversions
    - Exercise toYaml
15. Flow Control
    - if
    - if - render only on condition (HPA)
    - with
    - range
16. Named Templates
    - named template
    - named template with dict
17. Helm - Fehlerhandling
    - Fehlerbehandlung mit require - url muss gesetzt werden
18. Helm mit gitlab ci/cd
    - Helm mit gitlab ci/cd ausrollen
19. Metrics - Server
    - Metrics - Server mit helm installieren und verwenden
20. helm - Dokumentation
    - Helm Documentation
    - Built in TopLevel - Objects like .Release

Backlog

1. Grundlagen
    - Feature / No-Features von Helm
2. Tipps & Tricks
    - kubernetes manifests mit privatem Repo
    - helm chart mit images auf privatem Repo
3. Helm-Befehle und -Funktionen

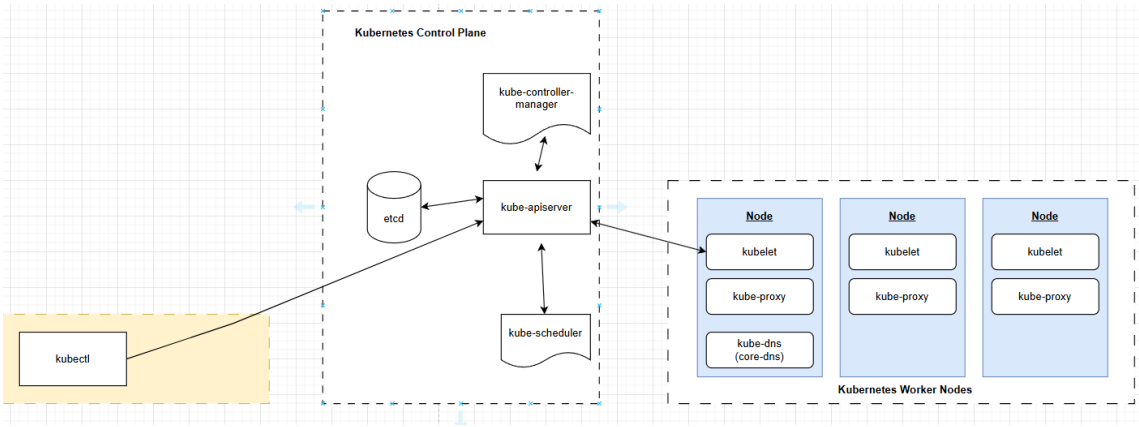
- Repo einrichten
- Suche in Repo und Artifacts Hub
- Anzeigen von Informationen aus dem Chart von Online
- Upgrade und auftretende Probleme

4. Helm Repository
    - Die wichtigsten Repo-Befehle
5. Struktur von Helm - Charts
    - Überblick
6. Grundlagen Helm-Charts
    - Testumgebung und Spaces (2 Themen)
7. Erstellen von Helm-Charts
    - Erstellen eines Guestbooks
    - Hooks für Guestbook erstellen
    - Dependencies/Abhängigkeiten herunterladen
    - Einfaches Testen
    - Input Validierung innerhalb von templates
    - Advanced Testing mit chart-testing
    - Chart auf github veröffentlichen
8. Sicherheit von helm-Chart
    - Grundlagen / Best Practices
    - Security Encrypted Passwords in helm
9. Testing in Helm-Charts
    - Testing in/von helm - charts
10. Durchführung von Upgrades und Rollbacks von Anwendungen
11. Helm in Continuous Integration / Continuous Deployment (CI/CD) Pipelines
12. Tipps & Tricks
    - Set namespace in config of kubectl
    - Create Ingress Redirect
    - Helm Charts - Development - Best practices
13. Integration mit anderen Tools
    - yamllint für Syntaxcheck von yaml - Dateien
14. Troubleshooting und Debugging
    - helm template --validate - gegen api-server testen

Kubernetes

Architektur Kubernetes

Schaubild



Komponenten / Grundbegriffe

Master (Control Plane)

Aufgaben

- Der Master koordiniert den Cluster
- Der Master koordiniert alle Aktivitäten in Ihrem Cluster
    - Planen von Anwendungen
    - Verwalten des gewünschten Status der Anwendungen
    - Skalieren von Anwendungen
    - Rollout neuer Updates.

Komponenten des Masters

ETCD

- Verwalten der Konfiguration des Clusters (key/value - pairs)

KUBE-CONTROLLER-MANAGER

- Zuständig für die Überwachung der Stati im Cluster mit Hilfe von endlos loops.
- kommuniziert mit dem Cluster über die kubernetes-api (bereitgestellt vom kube-api-server)

KUBE-API-SERVER

- provides api-frontend for administration (no gui)
- Exposes an HTTP API (users, parts of the cluster and external components communicate with it)
- REST API

KUBE-SCHEDULER

- assigns Pods to Nodes.
- scheduler determines which Nodes are valid placements for each Pod in the scheduling queue (according to constraints and available resources)
- The scheduler then ranks each valid Node and binds the Pod to a suitable Node.
- Reference implementation (other schedulers can be used)

Nodes

- Nodes (Knoten) sind die Arbeiter (Maschinen), die Anwendungen ausführen

- Ref: https://kubernetes.io/de/docs/concepts/architecture/nodes/

Pod/Pods

- Pods sind die kleinsten einsetzbaren Einheiten, die in Kubernetes erstellt und verwaltet werden können.
- Ein Pod (übersetzt Gruppe) ist eine Gruppe von einem oder mehreren Containern
  - gemeinsam genutzter Speicher- und Netzwerkressourcen
  - Befinden sich immer auf dem gleich virtuellen Server

Control Plane Node (former: master) - components

Node (Minion) - components

General

- On the nodes we will rollout the applications

kubelet

```
Node Agent that runs on every node (worker)
Er stellt sicher, dass Container in einem Pod ausgeführt werden.
```

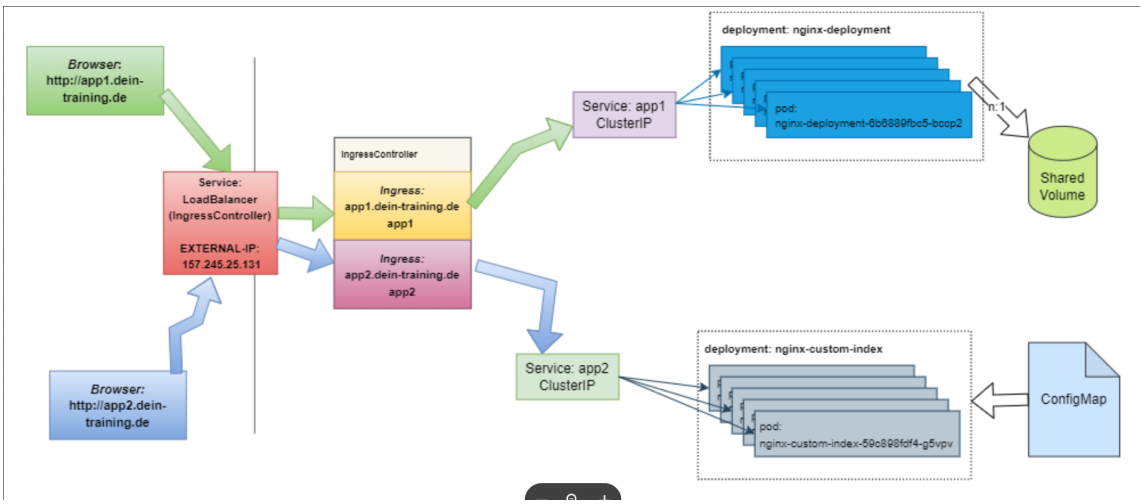
Kube-proxy

- Läuft auf jedem Node
- = Netzwerk-Proxy für die Kubernetes-Netzwerk-Services.
- Kube-proxy verwaltet die Netzwerkkommunikation innerhalb oder außerhalb Ihres Clusters.

Referenzen

- https://www.redhat.com/de/topics/containers/kubernetes-architecture

Aufbau von Browser zu Applikation - Schaubild



Helm Einfuehrung

Was ist helm ?

- Paketmanager für Kubernetes
- Ermöglicht es Anwendungen in einem Kubernetes-Cluster zu definieren, zu installieren und zu verwalten

- ähnlich wie `apt` bei Debian oder `yum` bei CentOS, aber speziell für Kubernetes.

Was kann helm ?

- **Installieren** und **Deinstallieren** von Anwendungen in Kubernetes (`helm install / helm uninstall`)
- **Upgraden** von bestehenden Installationen (`helm upgrade`)
- **Rollbacks** durchführen, falls etwas schiefläuft (`helm rollback`)
- **Anpassen** von Anwendungen durch Konfigurationswerte (`values.yaml`)
- **Veröffentlichen** eigener Charts (z. B. in einem Helm-Repository)

Was ist in helm ein chart?

Definition

- Ein **Helm Chart** ist ein Paket, das alle nötigen Kubernetes-Ressourcen beschreibt, um eine Anwendung oder einen Dienst bereitzustellen.

Es enthält:

- /**templates**: Vorlagen in YAML-Format, die dynamisch Werte einsetzen
- **values.yaml**: Eine Datei mit Konfigurationswerten
- **Chart.yaml**: Metainformationen zum Chart (Name, Version, etc.)
- **Abhängigkeiten**: Optional können andere Charts mit eingebunden werden

Formate / Ort

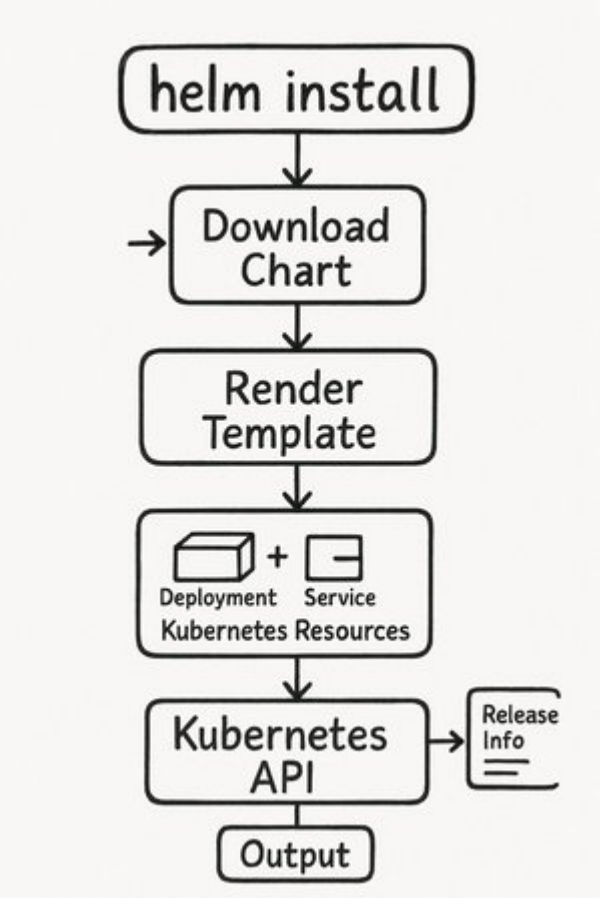
- Verzeichnis z.B. meine-app (und in dem Verzeichnis die bekannte Struktur von oben)
- tgz (Tape-Archive mit gnuzip komprimiert)
- URL

Warum Helm in Kubernetes verwenden ?

- **Wiederverwendbarkeit**: Ein Chart kann mehrfach und in unterschiedlichen Umgebungen genutzt werden.
- **Konfigurierbarkeit**: Anpassung an verschiedene Umgebungen wie Entwicklung, Test, Produktion.
- **Automatisierbarkeit**: Ideal für den Einsatz in CI/CD-Pipelines.
- **Große Community**: Viele fertige Charts für beliebte Software wie Prometheus, Grafana, nginx, etc.

Überblick über den Ablauf bei der Nutzung von helm (Kommando: install)

Grafik



Der Weg

Wenn der Befehl `helm install` ausgeführt wird, passiert intern Folgendes:

1. **Chart-Abfrage**:
   - Helm sucht Chart lokal oder im Repos und lädt es herunter.
2. **Chart-Templating**:
   - Helm rendert die Templates im Chart.
   - Variablen werden (wie in der `values.yaml` definiert) in die Templates eingefügt.
   - Dadurch werden manifeste für Kubernetes-Ressourcen (z. B. Deployments, Services) erstellt.
3. **Kubernetes API**:
   - Das gerenderte Kubernetes Manifest wird an den Kubernetes-API geschickt.
4. **Release-Verwaltung**:
   - Helm speichert die Chart- und Versionsinformationen in der Helm-Release-Datenbank (in Kubernetes als Secret)
   - Dies ermöglicht eine spätere Verwaltung und Aktualisierung des Releases.
5. **Ausgabe** (templates/NOTES.txt):
   - Helm gibt den Status des Installationsprozesses aus, einschließlich der erstellten Ressourcen und etwaiger Fehler.

Long story short

- Helm rendert Kubernetes-Ressourcen aus einem Chart und kommuniziert mit der Kubernetes-API, um diese Ressourcen zu erstellen und ein Release zu verwalten.

Braucht helm das Programm kubectl ?

- helm braucht zwar kubectl nicht, es verwendet aber auch die .kube/config - Datei per Default

Helm Installation und Konfiguration (inkl. kubectl)

Installation von kubectl unter Linux

Walkthrough (Start with unprivileged user like training or kurs)

```
sudo su -
```

```
## Get current version
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
## install the kubectl to the right directory
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Konfiguration von kubectl mit namespaces

config einrichten

```
cd
mkdir .kube
cd .kube
cp /tmp/config config
ls -la
## Alternative: nano config befüllen
## das bekommt ihr aus Eurem Cluster Management Tool
```

```
kubectl cluster-info
```

Arbeitsbereich konfigurieren

```
kubectl create ns jochen
kubectl get ns
kubectl config set-context --current --namespace jochen
kubectl get pods
```

Installation von helm unter Linux

Walkthrough (Start as unprivileged user, e.g. training or kurs)

```
sudo su -
```

```
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
```

```
chmod 700 get_helm.sh
./get_helm.sh
```

```
exit
```

Reference:

- https://helm.sh/docs/intro/install/

Installation bash completion

```
sudo su -
helm completion bash > /etc/bash_completion.d/helm
exit
## z.B.
su - tln11
```

Helm - Spickzettel

Wichtig: Helm Spickzettel

Alle helm-releases anzeigen

```
## im eigenen Namespace
helm list
## in allen Namespaces
helm list -A
## für einen speziellen
helm -n kube-system list
```

Helm - Chart installieren

```
## Empfehlung mit namespace
## Repo hinzufügen für Client
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install my-nginx bitnami/nginx --version 19.0.1 --create-namespace --
namespace=app-<namenskuerzel>
```

Helm - Suche

```
## welche Repos sind konfiguriert
helm repo list
helm search repo bitnami
helm search hub
```

Helm - template

```
## Rendern des Templates
helm repo add bitnami https://charts.bitnami.com/bitnami
```

```
helm template my-nginx bitnami/nginx
helm template bitnami/nginx
```

Helm - values (von installierter Release aus secrets)

```
helm -n app-jm get values my-nginx
helm -n app-jm get values my-nginx --revision 1
```

Helm - Hilfe

```
## Hilfe ist auf jeder Ebene möglich
helm --help
helm get --help
helm get values --help
```

Helm - Tipps & Tricks

Helm values.yaml autogenerieren

```
grep -Rho '{{[^}]*\.Values[^}]*}}' charts/mychart/templates/ \
  | sed -E 's/.*\.Values\.([a-zA-Z0-9_.-]+).*/\1/' \
  | sort -u \
  | awk -F. '{
      indent=""
      for (i=1; i<NF; i++) {
        indent=indent"  "
        printf "%s%s:\n", indent, $i
      }
      indent=indent"  "
      printf "%s%s: <TODO>\n", indent, $NF
    }' > values.generated.yaml
```

```
image:
  repository: <TODO>
  tag: <TODO>
service:
  port: <TODO>
  type: <TODO>
```

Arbeiten mit helm - charts (Basics)

Installation, Upgrade, Uninstall helm-Chart exercise

Install

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

```
## Installiert
helm install my-nginx bitnami/nginx --version 19.0.4 --create-namespace --namespace
```

```
app-<namenskuerzel>
## Zeigt an, was er ausrollen würde
helm install my-nginx bitnami/nginx --version 19.0.4 --dry-run # auch für uninstall,
upgrade
```

```
## noch besser
## Installiert
helm upgrade --install my-nginx bitnami/nginx --version 19.0.4 --create-namespace --
namespace app-<namenskuerzel>
```

```
## überprüfen // laufen die pods
kubectl -n app-<namenskuerzel> get all
```

Exercise: Upgrade to new version

```
## Recherchiere wie die Werte gesetzt werden (artifacthub.io) oder verwende die
folgenden Befehle:
helm show values bitnami/nginx
helm show values bitnami/nginx | less
```

```
cd
mkdir -p nginx-values
cd nginx-values
mkdir prod
cd prod
```

```
nano values.yaml
```

```
resources:
   requests:
     cpu: 0.1
     memory: 150Mi
   limits:
     cpu: 0.1
     memory: 150Mi
```

```
cd ..
helm upgrade --install my-nginx bitnami/nginx --create-namespace --namespace app-
<nameskuerzel> --version 21.0.6 -f prod/values.yaml
```

Umschauen

```
kubectl -n app-<namenskuerzel> get pods
helm -n app-<namenskuerzel> status my-nginx
helm -n app-<namenskuerzel> list
## alle helm charts anzeigen, die im gesamten Cluster installierst wurden
helm -n app-<namenskuerzel> list -A
helm -n app-<namenskuerzel> history my-nginx
```

Umschauen get

```
## Wo speichert er Information, die er später mit helm get abruft
kubectl -n app-<namenskuerzel> get secrets
```

```
helm -n app-tln1 get values my-nginx
helm -n app-tln1 get manifest my-nginx
helm -n app-tln1 get manifest my-nginx | grep "150Mi" -A4 -B4
## Can I see all values use -> YES
## Look for COMPUTED VALUES in get all ->
helm -n app-tln1 get all my-nginx
```

```
## Hack COMPUTED VALUES anzeigen lassen
## Welche Werte (values) hat er zur Installation verwendet
helm -n app-<namenskuerzel> get all my-nginx | grep -i computed -A 200
```

Tipp: values aus alter revision anzeigen

```
## Beispiel:
helm -n app-jm get values  my-nginx --revision 1
```

Uninstall

```
helm -n app-<namenskuerzel> uninstall my-nginx
## namespace wird nicht gelöscht
## händisch löschen
kubectl delete ns app-<namenskuerzel>
## crd's werden auch nicht gelöscht
```

Problem: OutOfMemory (OOM-Killer) if container passes limit in memory

- if memory of container is bigger than limit an OOM-Killer will be triggered
- How to fix. Use memory limit in the application too !
    - https://techcommunity.microsoft.com/blog/appsonazureblog/unleashing-javascript-applications-a-guide-to-boosting-memory-limits-in-node-js/4080857

Nur fertiges manifest ausgeben ohne Installation

template

Warum ?

- Ich will vorher sehen, wie mein Manifest ausschaut, bevor ich es zum Kube-API-Server schicke.

Was macht das ?

- Rendered das Template.

Was macht es nicht ?

- Da er erst nicht an den schickt,
- Überpüft er nicht, ob der Syntax korrekt ist, nur ob das yaml-format eingehalten wird

Beispiel:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
## Kann sehr lang sein
helm -n app-<namenskuerzel> template my-nginx bitnami/nginx --version 19.0.4 | less
helm -n app-<namenskuerzel> template my-nginx bitnami/nginx --version 19.0.4 | grep -A
4 -i ^Kind
```

template --debug

Warum ?

- Zeigt mein template auch an, wenn ein yaml-Einrückungsfehler oder Syntax - fehler da ist.

Beispiel

```
helm -n app-jm template my-nginx bitnami/nginx --version 19.0.4 --debug
```

Informationen aus nicht installierten Helm-Charts bekommen

```
helm show values bitnami/mariadb
helm show values bitnami/mariadb | grep -B 20 -i "image:"
## recommendation -> redirect to file
helm show values bitnami/mariadb > default-values.yaml
```

```
## Zeigt Chart-Definition, Readme usw. (=alles) an
helm show all bitnami/mariadb
```

```
helm show readme bitnami/mariadb
helm show chart bitnami/mariadb
```

```
helm show crds bitnami/mariadb
```

Chart runterladen und evtl. entpacken und bestimmte Version

```
cd
mkdir -p charts
cd charts
```

```
## Vorher müssen wir den Repo-Eintrag anlegen
helm repo add bitnami https://charts.bitnami.com/bitnami

## Lädt die letzte version herunter
helm pull bitnami/mariadb

## Lädt bestimmte chart-version runter
helm pull bitnami/mariadb --version 12.1.6
## evtl. entpacken wenn gewünscht
## tar xvf mariadb-12.1.6.tgz

## Schnelle Variante
helm pull bitnami/mariadb --version 12.1.6 --untar
```

Aufräumen von CRD's nach dem Deinstallieren

Schritt 1: repo hinzufügen

```
helm repo add jetstack https://charts.jetstack.io
```

Schritt 2: chart runterladen und entpacken (zum Gucken)

```
helm pull jetstack/cert-manager
ls -la
helm pull jetstack/cert-manager --untar
ls -la
cd cert-manager
ls -la
cd templates
ls -la crds.yaml
```

Schritt 3: Installieren

```
cd
mkdir cm-values
cd cm-values
nano values.yaml
```

```
crds:
  enabled: true
```

```
helm install cert-manager jetstack/cert-manager --namespace cert-manager-
<namenskuerzel> --create-namespace -f values.yaml
kubectl -n cert-manager-<namenskuerzel> get all
```

CRD's da ?

```
kubectl get crds | grep cert
```

Deinstallieren

```
helm -n cert-manager-<namenskuerzel> uninstall cert-manager
```

CRD's noch da ?

```
kubectl get crds | grep cert
```

CRD's händisch löschen

```
## Variante 1
kubectl delete crd certificaterequests.cert-manager.io certificates.cert-manager.io
```

```
challenges.acme.cert-manager.io  clusterissuers.cert-manager.io  issuers.cert-
manager.io orders.acme.cert-manager.io
```

Arbeiten mit helm - charts (Debugging)

Nur fertiges manifest ausgeben ohne Installation

template

Warum ?

- Ich will vorher sehen, wie mein Manifest ausschaut, bevor ich es zum Kube-API-Server schicke.

Was macht das ?

- Rendered das Template.

Was macht es nicht ?

- Da er erst nicht an den schickt,
- Überpüft er nicht, ob der Syntax korrekt ist, nur ob das yaml-format eingehalten wird

Beispiel:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
## Kann sehr lang sein
helm -n app-<namenskuerzel> template my-nginx bitnami/nginx --version 19.0.4 | less
helm -n app-<namenskuerzel> template my-nginx bitnami/nginx --version 19.0.4 | grep -A
4 -i ^Kind
```

template --debug

Warum ?

- Zeigt mein template auch an, wenn ein yaml-Einrückungsfehler oder Syntax - fehler da ist.

Beispiel

```
helm -n app-jm template my-nginx bitnami/nginx --version 19.0.4 --debug
```

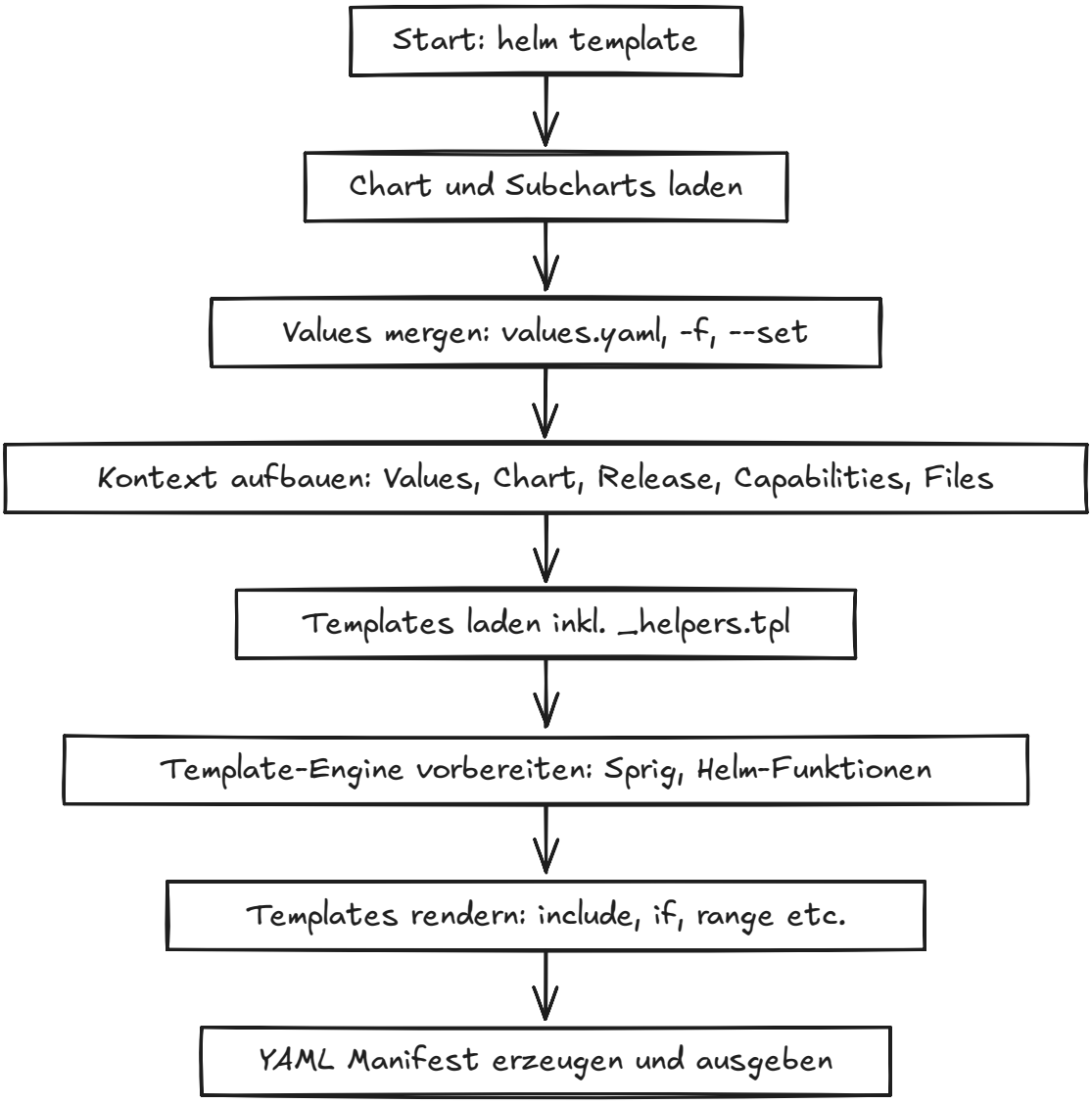
Chart trocken testen gegen api-server ohne Installation --dry-run

```
helm repo add bitnami https://charts.bitnami.com/bitnami
## auch die gerenderten Manifeste werden angezeit auch ohne "--debug"
helm -n app-jm install my-nginx bitnami/nginx --version 19.0.4 --dry-run
```

Helm Internals

Helm template - Rendering Prozess

Ablauf



Aufbau des Kontext:

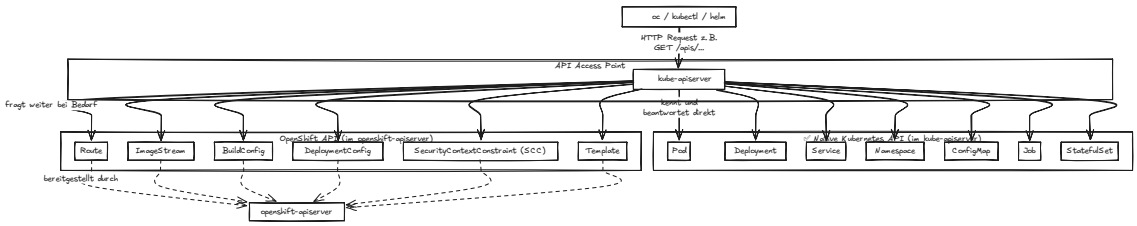
- Helm baut einen Kontext auf:
- bedeutet bei Helm, dass für jedes Chart (und Subchart) eine Datenstruktur erstellt wird, die die Templates beim Rendern als Eingabe verwenden.

Struktur des Kontext

```
├── Chart:
│   ├── Name: mychart
│   └── Version: 0.1.0
├── Release:
│   ├── Name: myrelease
│   └── Namespace: default
├── Values:
│   ├── image:
│   │   └── tag: 1.2.3
```

```
│   └── global:
│       └── commonLabels: {...}
├── Capabilities:
│   └── APIVersions: [...]
├── Files:
│   └── get: function to read files/
└── Template:
    └── Name: templates/deployment.yaml
```

helm vs. kubectl vs. oc



Helm - best practices

Wann quotes in yaml und in resources (Kubernetes/OCP)

In YAML sind Anführungszeichen *optional*, aber es gibt einige klare Regeln, **wann du sie brauchst** und **wann du sie besser weglässt**. Ich erkläre es dir allgemein und dann mit Blick auf Kubernetes-Ressourcen.

✅ Wann du in YAML Anführungszeichen verwenden solltest

Fall	Beispiel	Erklärung
Sonderzeichen wie `:`, `#`, `@`, `{`, `}`	`"abc:123"`	Ohne Quotes würde `:` als Trennzeichen interpretiert.
Strings, die mit `~`, `null`, `yes`, `no`, `on`, `off`, `true`, `false` beginnen	`"no"`	YAML interpretiert unquoted `no` als `false`.
Strings mit Leerzeichen oder Zeilenumbrüchen	`"Hello World"`	Ohne Quotes wäre das ein Syntaxfehler.
Zahlen, die nicht als Zahl interpretiert werden sollen (z. B. Postleitzahlen mit führender Null)	`"01234"`	Ohne Quotes würde `01234` als Oktalzahl gelesen.
Strings mit Unicode oder Escape-Sequenzen	`"Line\nBreak"`	Nur mit doppelten Quotes wird `\n` als Linebreak erkannt.
Variablen oder Helm-Templates in Helm Charts	`value: "{{ .Values.foo }}"`	Hier brauchst du doppelte Quotes, damit die Klammern als String erkannt werden.

❌ Wann du in YAML keine Anführungszeichen brauchst

Fall	Beispiel	Erklärung
Einfache Strings ohne Sonderzeichen	`name: nginx`	Klarer, einfacher Name – kein Risiko.
Zahlen, die wirklich Zahlen sind	`replicas: 3`	Wird korrekt als Zahl erkannt.
Booleans, die wirklich booleans sein sollen	`enabled: true`	Wird korrekt als boolescher Wert erkannt.
Listen und Dictionaries	`ports: [80, 443]`	Quotes wären unnötig.

Bei Kubernetes-Ressourcen (manifeste YAMLs)

In Kubernetes-YAML-Dateien wie `Deployment`, `Service`, `ConfigMap`, usw. gelten die gleichen YAML-Regeln, aber einige typische Beispiele:

Feld	Beispiel	Mit/ohne Quotes
`kind`, `apiVersion`	`kind: Deployment`	Ohne Quotes
Container-Befehle	`command: ["sh", "-c", "echo Hello"]`	Strings in Arrays **mit** Quotes
`env`-Variablen	`value: "123"`	Quotes empfehlenswert, da es sonst als Zahl gedeutet werden könnte
`args` mit Template	`args: ["--url={{ .Values.url }}"]`	Quotes zwingend wegen Helm-Template

Faustregel

- **Immer Quotes**, wenn es Zweifel geben könnte.
- **Keine Quotes nötig**, wenn es ein klarer einfacher Wert ist (String ohne Sonderzeichen, Zahl, Bool).
- **Bei Helm immer vorsichtig sein** – lieber doppelte Quotes `"{{ }}"` statt einfache `'{{ }}'` oder ganz ohne.

Wenn du willst, zeige ich dir gerne ein konkretes Kubernetes-Beispiel mit und ohne Quotes.

Gute Struktur für Values und Charts

- e.g. in git repo

Beispiel 1

```
helm-exercises/
├── helm-values
│   ├── iftest
│   │   └── values.yaml
│   └── my-dep
│       └── values.yaml
└── iftest
    ├── Chart.yaml
```

```
    ├── charts
    ├── templates
    │   ├── _helpers.tpl
    │   └── cm.yaml
    └── values.yaml
```

Beispiel 2: Struktur für gitlab ci/cd

```
helm-repo-app1/
├── charts
│   └── app
│       ├── Chart.yaml
│       ├── templates
│       └── values.yaml
└── helm-values
    ├── prod
    │   └── values.yaml
    ├── staging
    │   └── values.yaml
    └── testing
        └── values.yaml
```

Best Practices for Chart Templating

- https://helm.sh/docs/chart_best_practices/

Helm - Advanced

Helm Dependencies Exercise

Exercise 1: Create chart with Dependency

```
cd
mkdir -p helm-exercises
cd helm-exercises
helm create my-dep
cd my-dep
nano Chart.yaml
```

```
## Add dependencies
dependencies:
  - name: redis
    version: "18.0.0"
    repository: "https://charts.bitnami.com/bitnami"
```

```
## Das 1. Mal - dann wird Chart.lock angelegt
helm dependency update
ls -la Chart.lock
```

```
rm -fR charts
helm dependency build
```

```
helm dependency --help
### what is the difference
```

```
helm template .
```

Exercise 2: Create chart with condition

```
nano Chart.yaml
```

```
## change dependency block
## adding condition
dependencies:
  - name: redis
    version: "18.3.2"
    repository: "https://charts.bitnami.com/bitnami"
    condition: redis.enabled
```

```
nano values.yaml
```

```
## unten anfügen
redis:
  enabled: false
```

```
helm template .
```

```
## values-file anlegen
cd
mkdir -p helm-values
cd helm-values
mkdir my-dep
cd my-dep
```

```
nano values.yaml
```

```
redis:
  enabled: true
```

```
cd
cd helm-exercises
helm template my-dep -f ../helm-values/my-dep/values.yaml
helm template my-dep -f ../helm-values/my-dep/values.yaml | grep kind -A 2
```

Helm Grundlagen

TopLevel Objekte

.Chart

- Zieht alle Infomationen aus der Chart.yaml
- Alle Eigenschaften fangen mit einem grossen Buchstaben, statt klein wie im Chart, z.B. .Chart.Name

.Values

- Ansprechen der Values bzw. Default Values

.Release

- Ansprechen aller Eigenschaften aus der Release z.B. .Release.Name

Helm Charts entwickeln

eigenes helm chart erstellen (Gruppe)

Chart erstellen

```
cd
mkdir my-charts
cd my-charts
```

```
helm create my-app
```

Chart testen

```
## nur template rendern
helm template my-app-release my-app
## chart trockenlauf (--dry-run) rendern und an den Server (kube-api-server) zur
Überprüfung schickt
helm -n my-app-<namenskuerzel> upgrade --install my-app-release my-app --create-
namespace --dry-run
```

Install helm - chart

```
## Variante 1:
helm -n my-app-<namenskuerzel> upgrade --install my-app-release my-app --create-
namespace
```

```
## Variante 2:
cd my-app
helm -n my-app-<namenskuerzel> upgrade --install my-app-release . --create-namespace
```

```
kubectl -n my-app-<namenskuerzel> get all
kubectl -n my-app-<namenskuerzel> get pods
```

Fehler bei ocp debuggen

```
kubectl -n my-app-<namenskuerzel> get pods
```

```
tln1@client:~/my-charts$ kubectl -n my-app-jm2 get pods
NAME                                READY   STATUS             RESTARTS        AGE
my-app-release-7d9bd79cb7-9gbbd     0/1     CrashLoopBackOff   7 (2m48s ago)   13m
```

```
## Wie debuggen -> Schritt 1:
kubectl -n my-app-<namenskuerzel> describe po my-app-release-7d9bd79cb7-9gbbd
```

```
11m  Normal   Pulled     2m27s (x74 over 17m)  kubelet            Container image "my-app:1.1" already present on machine
  Warning  BackOff    2m27s (x74 over 17m)  kubelet            Back-off restarting failed container my-app in pod my-app-release-7d9bd79cb7-9gbbd_my-app-jm2(fbffd33a-d3cd-454d-9ec6-457196262ea9)
```

```
## Wenn Schritt 1 kein gesichertes Ergebnis liefert.
## Wie debuggen -> Schritt 2: Logs
kubectl -n my-app-jm2 logs my-app-release-7d9bd79cb7-9gbbd
```

```
privileges, ignored in /etc/nginx/nginx.conf:2
2025/07/11 08:07:29 [emerg] 1#1: mkdir() "/var/cache/nginx/client_temp" failed (13: Permission denied)
```

```
## Schritt 3: yaml von pod anschauen, warum tritt der Fehler auf
kubectl -n my-app-<namenskuerzel> get pods -o yaml
```

```
## Dieser Block ist dafür verantwortlich, dass keine Pods als root ausgeführt werden, können. nginx will aber unter root laufen (bzw. muss)
```

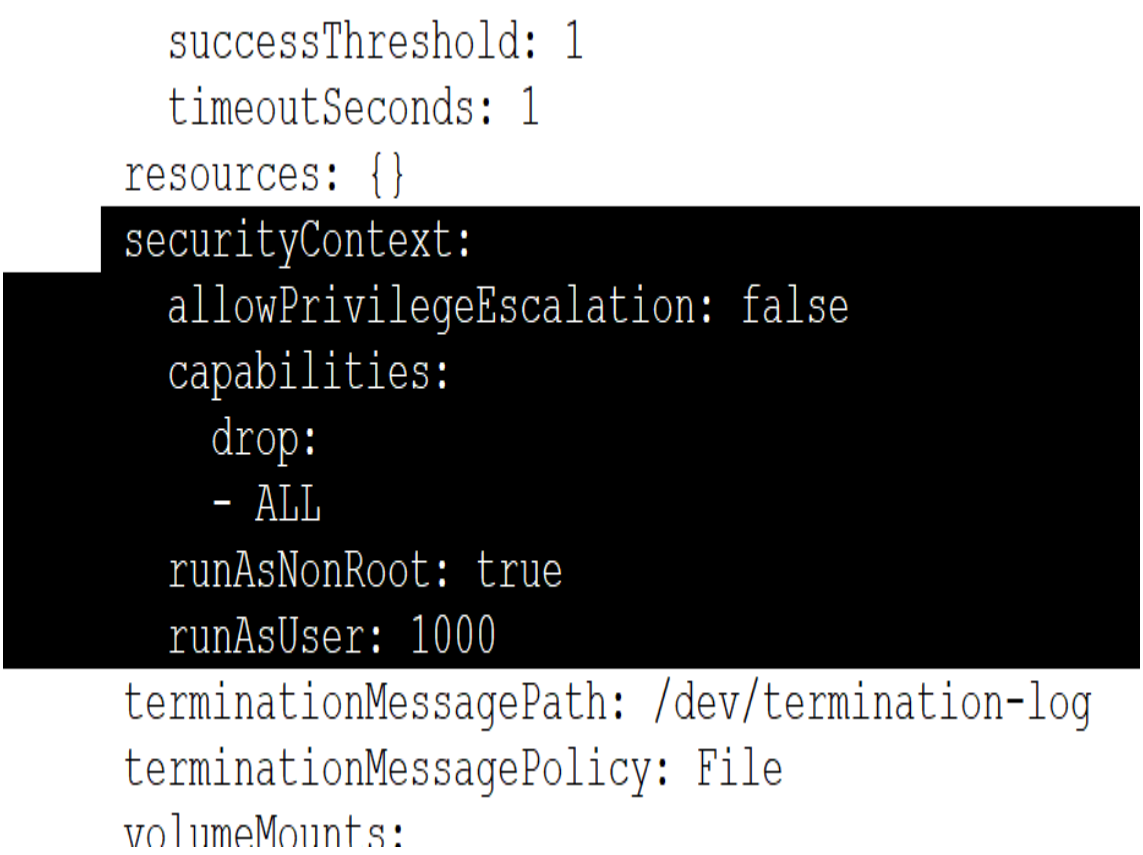


Image verwenden was auch als nicht-root läuft

```
cd
cd my-charts
nano my-app/values.yaml
```

```
## image Zeile ändern
## von ->
image:
  repository: nginx

## in ->
image:
  repository: bitnami/nginx
```

```
## Auch wichtig version in Chart.yaml um 1 erhöhen z.B. 0.1.0 -> 0.1.1
```

```
helm -n my-app-<namenskuerzel> upgrade --install my-app-release my-app --create-
namespace
```

```
kubectl -n my-app-<namenskuerzel> get all
kubectl -n my-app-<namenskuerzel> get pods
```

```
## Schlägt fehl, weil readiness auf 80 abfragt, aber dort nichts läuft
```

Readiness-Probe port anpassen und version (Chart-Version) hochziehen

```
cd my-app
nano Chart.yaml
```

```
version: 0.1.2
```

```
nano values.yaml
```

```
##### von --_>
livenessProbe:
  httpGet:
    path: /
    port: http
readinessProbe:
  httpGet:
    path: /
    port: http

#### auf --_>

livenessProbe:
  httpGet:
    path: /
    port: 8080
readinessProbe:
  httpGet:
    path: /
    port: 8080
```

```
helm -n my-app-<namenskuerzel> upgrade --install my-app-release . --create-namespace
```

```
kubectl -n my-app-<namenskuerzel> get all
kubectl -n my-app-<namenskuerzel> get pods
```

Wie starte ich am besten ganz einfach - Übung

- Really simple version to start

Step 1: Create sample chart

```
cd
mkdir -p my-charts
cd my-charts
helm create app
cd app
```

Step 2: Cleanup

```
cd templates
rm -fR tests
rm -fR *.yaml
rm NOTES.txt
echo "meine app ist ausgerollt" > NOTES.txt
cd ..
rm values.yaml
## leere datei wird erzeugt
touch values.yaml
```

Step 3: Create Deployment manifest

```
nano templates/deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 8 # tells deployment to run 8 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: bitnami/nginx:1.22
        ports:
        - containerPort: 8080
```

Step 4: Testen des Charts

```
helm template .
helm lint .
## Akzeptiert der API das so, wie ich es ihm schicke
helm -n app-<namenskuerzel> install app . --dry-run
helm -n app-<namenskuerzel> upgrade --install app . --create-namespace
kubectl -n app-<namenskuerzel> get all
```

Spezial: Umgang mit Einrückungen

Whitespaces meistern mit "-"

Grundlagen

- In Helm (bzw. in Go-Templates) hast du verschiedene Möglichkeiten, den Umgang mit Whitespace (z. B. Leerzeichen, Zeilenumbrüche) zu steuern:
- `{{ ... }}`:
  Standardvariante. Lässt den Whitespace außerhalb der geschweiften Klammern unverändert.
- `{{- ... }}`:
  Entfernt den Whitespace links (vor) dem Ausdruck.
- `{{ ... -}}`:
  Entfernt den Whitespace rechts (nach) dem Ausdruck, aber AUCH Zeilenumbrüche
- `{{- ... -}}`:
  Entfernt Whitespace sowohl links als auch rechts des Ausdrucks, aber AUCH Zeilenumbrüche

Exercise Whitespaces

Explanation

- {{- -> trim on left side
- -}} -> trim on right side / ALSO: new lines
- trim tabs, whitespaces a.s.o. (see ref)

Walkthrough

```
cd
mkdir -p helm-exercises
cd helm-exercises
```

```
## When ever we encounter error while parsing yaml, we can use comment !!!
helm create testenv
cd testenv/templates
rm -fR *.yaml
rm -fR tests
```

```
nano test.yaml
```

```
## "{{23 -}} < {{- 45}}"
```

```
helm template ..
helm template --debug ..
```

```
## now with new lines
nano test2.yaml
```

```
## {{23 -}}
newline here
```

```
helm template ..
helm template --debug ..
```

Reference:

- https://pkg.go.dev/text/template#hdr-Text_and_spaces

Type - Conversions

Exercise toYaml

Exercise

```
cd
mkdir -p helm-exercises
cd helm-exercises
helm create example-toyaml
cd example-toyaml
rm -fR values.yaml
rm -fR templates/*
```

```
nano templates/configmap.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-config
data:
  app-config.yaml: |
    {{- toYaml .Values.appConfig | nindent 4 }}
```

```
nano values.yaml
```

```
appConfig:
  server:
    port: 8080
    host: "0.0.0.0"
  features:
    auth: true
    metrics: true
  database:
    user: "admin"
    password: "secret"
    hosts:
      - db1.example.com
      - db2.example.com
```

```
helm template .
```

Ref:

- https://helm.sh/docs/chart_template_guide/function_list/#type-conversion-functions

Flow Control

if

Prepare (if not done yet)

```
cd
mkdir -p helm-exercises
cd helm-exercises
helm create iftest
cd iftest/templates
rm -fR *.yaml
```

Step 2: values-file erweitern

```
rm ../values.yaml
rm -fR tests
rm -fR NOTES.txt
nano ../values.yaml
```

```
## Adjust values.yaml file accordingly
favorite:
  food: PIZZA
  drink: coffee
```

Step 3: Probably the best solution

```
nano cm.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- if eq .Values.favorite.drink "coffee"}}
  {{ "mug: true" }}
  {{- end }}
```

```
helm template ..
```

Step 4: change favorite drin

```
nano ../values.yaml
```

```
## Adjust values.yaml file accordingly
favorite:
  food: PIZZA
  drink: tea
```

```
helm template ..
```

Reference

- https://helm.sh/docs/chart_template_guide/control_structures/

if - render only on condition (HPA)

```
cd
mkdir -p helm-exercises

cd helm-exercises
helm create app2
cd app2/templates
less hpa.yaml
```

```
## find out the correct value to enable hpa
cd ..
less values.yaml
```

```
## Seperate values from from helm-charts
cd
mkdir -p helm-values
cd helm-values
mkdir prod
cd prod
nano values.yaml
```

```
autoscaling:
    enabled: true
```

```
cd
cd helm-exercises
helm template app2 -f ../helm-values/prod/values.yaml
```

with

Walkthrough

Preparation

```
cd
mkdir -p helm-exercises
cd helm-exercises
helm create with-example
cd with-example/templates
rm -fR *.yaml
```

```
nano ../values.yaml
```

```
## Adjust values.yaml file accordingly
favorite:
  food: PIZZA
  drink: coffee
```

Step 1:

```
nano cm.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  {{- end }}
```

```
helm template ..
```

Step 2a: Does not work because scope does not fit

```
nano cm.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  release: {{ .Release.Name }}
  {{- end }}
```

```
helm template --debug ..
```

Step 2b: Solution 1: (Outside with)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- with .Values.favorite }}
```

```
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  {{- end }}
  release: {{ .Release.Name }}
```

```
helm template --debug ..
```

Step 2c: Changing the scope

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  release: {{ $.Release.Name }}
  {{- end }}
```

```
helm template --debug ..
```

range

Preparation

```
cd
mkdir -p helm-exercises
cd helm-exercises
helm create range
cd range/templates
rm -f *.yaml
```

Step 1: Values.yaml

```
favorite:
  drink: coffee
  food: pizza
pizzaToppings:
  - mushrooms
  - cheese
  - peppers
  - onions
```

Step 2 (Version 1):

```
## nano cm.yaml
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  {{- end }}
  toppings: |-
    {{- range .Values.pizzaToppings }}
    - {{ . | title | quote }}
    {{- end }}
```

Step 3 (Version 2 - works as well)

- Accessing the parent scope

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  toppings: |-
    {{- range $.Values.pizzaToppings }}
    - {{ . | title | quote }}
    {{- end }}
  {{- end }}
```

Named Templates

Helm - Fehlerhandling

Fehlerbehandlung mit require - url muss gesetzt werden

Walkthrough

```
cd
mkdir -p helm-exercises
cd helm-exercises
helm create urlexample
cd urlexample
nano values.yaml
```

```
## ganz am Ende anhängen
deployment:
  url: ''
```

```
cd templates
nano deployment.yaml
```

```
## labels ergänzen (4 spaces eingerueckt)
   url: {{ required "Please set url" .Values.deployment.url | quote }}
```

```
cd
mkdir -p helm-values
cd helm-values
mkdir -p prod
mkdir -p dev
nano prod/values.yaml
```

```
deployment:
  url: 'https://someprod.com'
```

```
nano dev/values.yaml
```

```
deployment:
  url: 'https://somedev.com'
```

Testing

```
cd
cd helm-exercises
```

```
## ohne values
helm template urlexample
```

```
## mit prod  values
helm template urlexample -f ../helm-values/prod/values.yaml
```

```
helm template urlexample -f ../helm-values/dev/values.yaml
```

Helm mit gitlab ci/cd

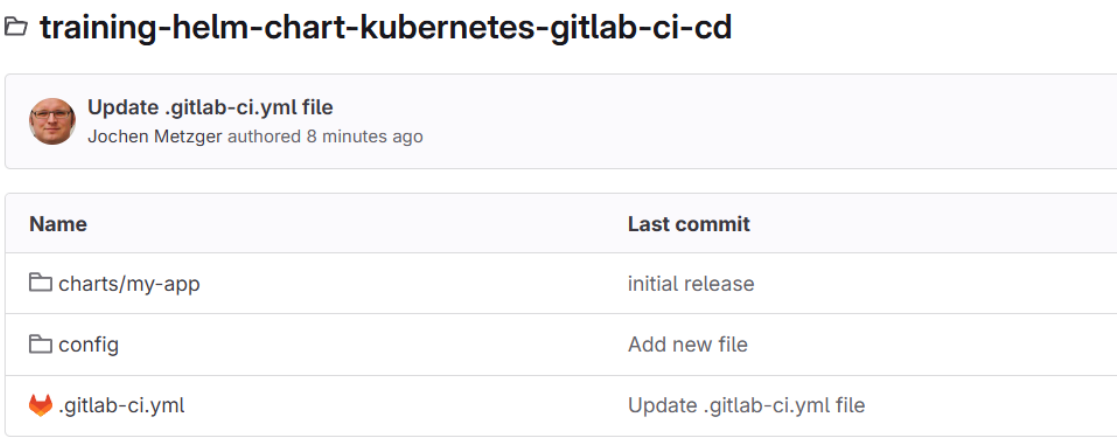
Helm mit gitlab ci/cd ausrollen

Step 1: Create gitlab - repo and pipeline

```
1. Create new repo on gitlab
2. Click on pipeline Editor and creat .gitlab-ci.yml with Button
```

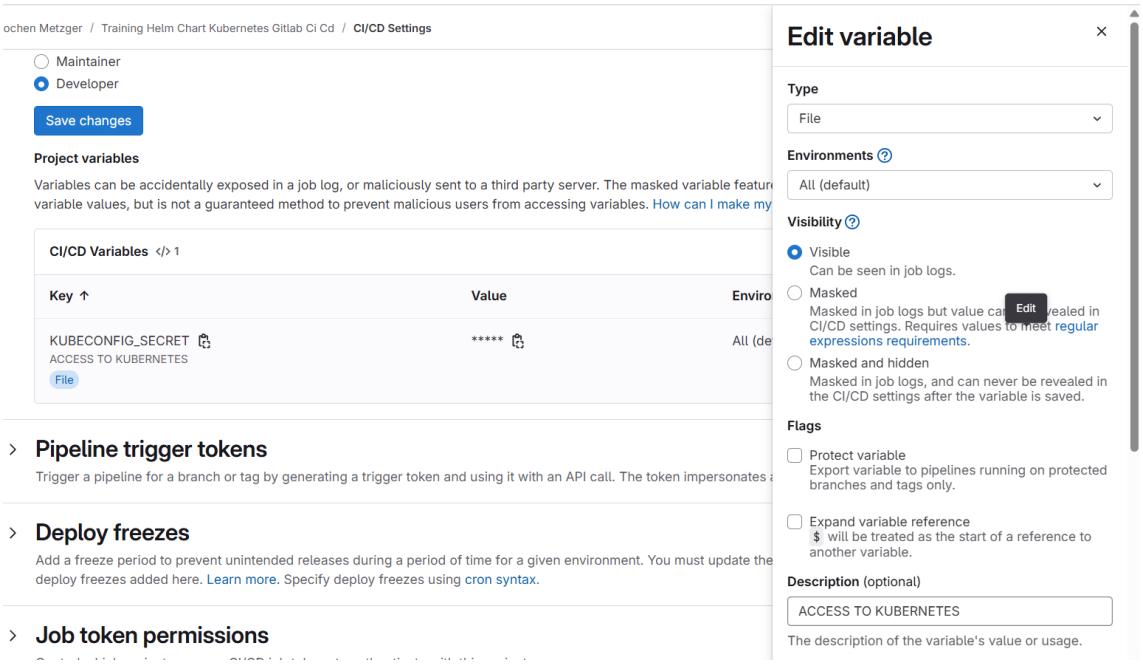
Step 2: Push your helm chart files to repo

- Now looks like this



Step 3: Add your KUBECONFIG as Variable (type: File) to Variables

- https://gitlab.com/jmetzger/training-helm-chart-kubernetes-gitlab-ci-cd/-/settings/ci_cd#js-cicd-variables-settings



Step 4: Create a pipeline for deployment

```
stages:          # List of stages for jobs, and their order of execution
  - deploy

variables:
  APP_NAME: my-first-app

deploy:
  stage: deploy
  image:
    name: alpine/helm:3.2.1
## Important to unset entrypoint
    entrypoint: [""]
  script:
    - ls -la
    - cd; mkdir .kube; cd .kube; cat $KUBECONFIG_SECRET > config; ls -la;
    - cd $CI_PROJECT_DIR; helm upgrade ${APP_NAME} ./charts/my-app --install --
namespace ${APP_NAME} --create-namespace -f ./config/values.yaml
  rules:
    - if: $CI_COMMIT_BRANCH == 'master'
      when: always
```

Reference: Example Project (Public)

- https://gitlab.com/jmetzger/training-helm-chart-kubernetes-gitlab-ci-cd

Metrics - Server

Metrics - Server mit helm installieren und verwenden

Warum ?

- Es wird ein API bereitgestellt, die Informationen zu den Auslastung von Pods und Nodes sammelt

Installation

```
helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/
helm -n kube-system upgrade --install my-metrics-server metrics-server/metrics-server
--version 3.12.2
```

- Achtung, danach geht es nicht sofort, es dauert einen Momeent bis ich es verwenden kann (geschätzt 5 Minuten)

Verwendung

```
kubectl top pods
## Pods in allen Namespaces
kubectl top pods -A
kubectl top nodes
```

helm - Dokumentation

Helm Documentation

- https://helm.sh/docs/

Built in TopLevel - Objects like .Release

- https://helm.sh/docs/chart_template_guide/builtin_objects/

Grundlagen

Feature / No-Features von Helm

- Sortiert, die Manifeste bzw. Objekte bereits automatisch in der richtigen Reihenfolge für das Anwenden (apply) gegen den Server (Kube-Api-Server)

Which order is it ?

- see also Internals Helm Sorting Objects

Tipps & Tricks

kubernetes manifests mit privatem Repo

Exercise

```
mkdir -p manifests
cd manifests
mkdir private-repo
cd private-repo
```

```
kubectl create secret docker-registry regcred --docker-server=registry.do.t3isp.de \
--docker-username=11trainingdo --docker-password=<sehr-geheim> --dry-run=client -o
yaml > 01-secret.yaml
```

```
kubectl create secret generic mariadb-secret --from-
literal=MARIADB_ROOT_PASSWORD=11abc432 --dry-run=client -o yaml > 02-secret.yml
```

```
nano 02-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: private-reg
spec:
  containers:
  - name: private-reg-container
    image: registry.do.t3isp.de/mariadb:11.4.5
    envFrom:
      - secretRef:
          name: mariadb-secret
  imagePullSecrets:
  - name: regcred
```

```
kubectl apply -f .
kubectl get pods -o wide private-reg
```

```
kubectl describe pods private-reg

### helm chart mit images auf privatem Repo


### Walkthrough
```

cd mkdir -p manifests cd manifests mkdir nginx-values cd nginx-values mkdir prod cd prod nano values.yaml

```

```

global: security: allowInsecureImages: true

image: registry: "registry.do.t3isp.de" repository: nginx

tag: 1.27.4

pullSecrets: - regcred-do

extraDeploy:

- apiVersion: v1 data: .dockerconfigjson: kind: Secret metadata: name: regcred-do type: kubernetes.io/dockerconfigjson

```

```

cd cd manifests/nginx-values helm upgrade --install my-nginx bitnami/nginx -f prod/values.yaml

```
## Helm-Befehle und -Funktionen

### Repo einrichten
```

helm repo list helm repo add bitnami https://charts.bitnami.com/bitnami helm repo remove bitnami helm repo update

```

### Suche in Repo und Artifacts Hub


### Suche im hub
```

helm search hub mariadb

Zeige kompletten Zeilen an ohne abzuschneiden

helm search hub mariadb --max-col-width=0

```

### Suche im Repo
```

Suche nach allen Charts, die mariadb im Namen oder der Beschreibung tragen

helm search repo mariadb

Zeige alle Version von charts an, die mit bitnami/mariadb beginnen

helm search repo bitnami/mariadb --versions

```
### Anzeigen von Informationen aus dem Chart von Online
```

helm show values bitnami/mariadb helm show values bitnami/mariadb | grep -B 20 -i "image:"

recommendation -> redirect to file

helm show values bitnami/mariadb > default-values.yaml

```

```

Zeigt Chart-Definition, Readme usw. (=alles) an

helm show all bitnami/mariadb

```

```

helm show readme bitnami/mariadb helm show chart bitnami/mariadb

```

```

helm show crds bitnami/mariadb

```


### Upgrade und auftretende Probleme

### Die wichtigsten Repo-Befehle
```

helm repo list helm repo add bitnami https://charts.bitnami.com/bitnami helm repo remove bitnami helm repo update

```


## Struktur von Helm - Charts

### Überblick


### Komponenten von Helm-Charts
```

```
#### Chart.yml

#### Chart.lock (wird automatisch generiert)

#### templates/

##### _helper.tpl

  * Enthält snippet die mit include oder templates inkludiert werden können
  * Konvention der Snippets mit define ChartName.Eigenschaft z.B. botti.fullname

##### NOTES.txt

  * Wird ausgegeben, nachdem das Chart installiert wurde
    * oder:
```

after installation

helm install my-botti -n my-application --create-namespace botti

helm get -n my-application notes my-botti

```
#### charts/

  * Hier werden die abhängigen charts runtergeladen und als .tgz



## Grundlagen Helm-Charts

### Testumgebung und Spaces (2 Themen)


### Explanation

  * {{- -> trim on left side
  * -}} -> trim on right side / ALSO: new lines
  * trim tabs, whitespaces a.s.o. (see ref)

### Walkthrough
```

cd mkdir -p helm-exercises cd helm-exercises

```

```

When ever we encounter error while parsing yaml, we can use comment !!!

helm create testenv cd testenv/templates rm -fR *.yaml rm -fR tests

```

```

nano test.yaml

```

```

"{{23 -}} < {{- 45}}"

```

```

helm template .. helm template --debug ..

```

```

now with new lines

nano test2.yaml

```

```

{{23 -}}

newline here

```

```

helm template .. helm template --debug ..

```


### Reference:

  * https://pkg.go.dev/text/template#hdr-Text_and_spaces

## Erstellen von Helm-Charts

### Erstellen eines Guestbooks


### Step 1: Create namespace and structure of helm chart
```

cd

```

```

helm create guestbook

now we have in folder "guestbook"

charts/

Chart.yaml

templates

values.yaml

```
### Step 2: Explore templates folder and cleanup
```

cd templates ls -la rm -fR tests

```
### Step 3: Explore the Chart.yaml
```

cd .. cat Chart.yaml

```

```

type: Application or Library # please explain !

dependencies - what other charts are needed - we will download them by helm command and they will be put in the charts - folder

```
### Step 4: Add redis as dependency
```

find the redis chart

helm search hub --max-col-width=0 redis | grep bitnami

adding the repo for bitnami

helm repo add bitnami https://charts.bitnami.com/bitnami

```

```

now find the availabe versions (these are the chart versions

helm search repo redis --versions

```

```

nano Chart.yaml

```

```

now add the dependency-block at the end of the file

dependencies:

- name: redis version: "17.14.x" # quotes are important here repository: https://charts.bitnami.com/bitnami

Save the file and leave nano:

STRG + o + RETURN -> then -> STRG + x

cd .. helm dependency update guestbook

explore the newly populated folder

cd guestbook/charts ls -la cd ../..

```
### Step 5: Modifying the values.yaml file
```

the version might have changed since i wrote this / adjust

helm show values charts/redis-17.14.5.tgz

what are the service name of the redis leader and the redis follower

helm show values charts/redis-17.14.5.tgz | grep -B 4 -i fullnameoverride

the service names need to be adjusted, add the following to the values.yaml

The guestbook - application needs the redis - services called. redis-leader and redis-follower

cd cd guestbook nano values.yaml

add at the end of the file

redis: fullnameOverride: redis

enable unauthorized access to redis

usePassword: false

Disable AOF persistence

configmap: |- appendonly no

save file and exit

STRG + o + ENTER -> then -> STRG + x

now check, if this really worked

cd cd guestbook helm template . | grep -A 20 master/service

```
### Setting the right repo and the right version
```

cd cd guestbook cat templates/deployment.yaml

Welche Version brauche ich ? https://kubernetes.io/docs/tutorials/stateless-application/guestbook/#creating-the-guestbook-frontend-deployment

Stand 2023-08-08

gcr.io/google_samples/gb-frontend:v5

nano Chart.yaml

korrigieren

appVersion: "v5"

nano values.yaml

image: repository: gcr.io/google_samples/gb-frontend

```
### Step 6: Changing LoadBalancer to NodePort
```

nano values.yaml

service: type: NodePort port: 80

```
### Step 7: Installing helm chart
```

helm install my-guestbook guestbook -n jochen --create-namespace kubectl -n jochen get all

```


### Reference:

  * https://kubernetes.io/docs/tutorials/stateless-application/guestbook/

### Hooks für Guestbook erstellen


### Step 1:
```

cd mkdir guestbook/templates/backup touch guestbook/templates/backup/persistentVolume-claim.yaml touch guestbook/templates/backup/job.yaml

```
### Step 2: persistentvolumeclaim.yaml und job bevölkern
```

nano guestbook/templates/backup/persistentVolume-claim.yaml

{{- if .Values.redis.master.persistence.enabled }} apiVersion: v1 kind: PersistentVolumeClaim metadata: name: redis-data-{{ .Values.redis.fullnameOverride }}-master-0-backup-{{ sub .Release.Revision 1 }} labels: {{- include "guestbook.labels" . | nindent 4 }} annotations: "helm.sh/hook": pre-upgrade "helm.sh/hook-weight": "0" spec: accessModes: - ReadWriteOnce resources: requests: storage: {{ .Values.redis.master.persistence.size }} {{- end }}

```

## nano guestbook/templates/backup/job.yaml

{{- if .Values.redis.master.persistence.enabled }} apiVersion: batch/v1 kind: Job metadata: name: {{ include "guestbook.fullname" . }}-backup labels: {{- include "guestbook.labels" . | nindent 4 }} annotations: "helm.sh/hook": pre-upgrade "helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded "helm.sh/hook-weight": "1" spec: template: spec: containers: - name: backup image: redis:alpine3.11 command: ["/bin/sh", "-c"] args: ["redis-cli -h {{ .Values.redis.fullnameOverride }}-master save && cp /data/dump.rdb /backup/dump.rdb"] volumeMounts: - name: redis-data mountPath: /data - name: backup mountPath: /backup restartPolicy: Never volumes: - name: redis-data persistentVolumeClaim: claimName: redis-data-{{ .Values.redis.fullnameOverride }}-master-0 - name: backup persistentVolumeClaim: claimName: redis-data-{{ .Values.redis.fullnameOverride }}-master-0-backup-{{ sub .Release.Revision 1 }} {{- end }}
```

```
### Step 3: pre-rollback hook erstellen
```

mkdir guestbook/templates/restore touch guestbook/templates/restore/job.yaml

```

```

nano guestbook/templates/restore/job.yaml

{{- if .Values.redis.master.persistence.enabled }} apiVersion: batch/v1 kind: Job metadata: name: {{ include "guestbook.fullname" . }}-restore labels: {{- include "guestbook.labels" . | nindent 4 }} annotations: "helm.sh/hook": pre-rollback "helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded spec: template: spec: containers: - name: restore image: redis:alpine3.11 command: ["/bin/sh", "-c"] args: ["cp /backup/dump.rdb /data/dump.rdb && redis-cli -h {{ .Values.redis.fullnameOverride }}-master debug restart || true"] volumeMounts: - name: redis-data mountPath: /data - name: backup mountPath: /backup restartPolicy: Never volumes: - name: redis-data persistentVolumeClaim: claimName: redis-data-{{ .Values.redis.fullnameOverride }}-master-0 - name: backup persistentVolumeClaim: claimName: redis-data-{{ .Values.redis.fullnameOverride }}-master-0-backup-{{ .Release.Revision }} {{- end }}

```
### Reference

  * https://helm.sh/docs/topics/charts_hooks/

### Dependencies/Abhängigkeiten herunterladen


### Voraussetzung:

  * Dependencies sind in Chart.yml eingetragen
  * Achtung: Version ist die Version des Charts nicht der App !!!

### Das 1. Mal
```

1. Alle Abhängigkeiten werden in Form von .tgz - Archiven heruntergeladen

```
 -> in das charts - Verzeichnis
```

2. Eine Chart.lock - datei wird erstellt. (hält den aktuellen Stand fest)

helm dependancy update $CHART_PATH

botti erklärt sich gleich unten im Walkthrough

helm dependancy update botti

```
### Das 2. Mal (wenn Chart.lock vorhanden, aber charts/ muss nicht da sein
```

helm dependancy build botti

```
### List all dependencies
```

helm dependancy list botti

```
### Walkthrough
```

cd helm create botti

```

```

cd botti

add dependency

nano Chart.yml

```

```

at the end of the file add

After that save and exit STRG + O + ENTER , STRG + X

```

```

Update to download depdendancies

cd .. helm dependency update botti cd botti/charts ls -la cd ../../

```

```

Add repo to be able to do helm dependency build

rm -fR botti/charts

Chart.lock needs to be there

ls -la botti/Chart.lock

Add repo / needs to be there, otherwice

helm repo add bitnami https://charts.bitnami.com/bitnami helm dependency build botti

```
### Einfaches Testen
```

```
### Input Validierung innerhalb von templates


### Walkthrough
```

cd helm create inputtest cd inputtest cd templates/ rm d* h* i* servicea* rm -fR tests

nano service.yaml mit folgendem Inhalt

apiVersion: v1 kind: Service metadata: name: {{ include "inputtest.fullname" . }} labels: {{- include "inputtest.labels" . | nindent 4 }} spec: {{- $serviceType := list "ClusterIP" "NodePort" }} {{- if has .Values.service.type $serviceType }} type: {{ .Values.service.type }} {{- else }} {{- fail "value 'service.type' must be either 'ClusterIP' or 'NodePort'" }} {{- end }} ports: - port: {{ .Values.service.port }} targetPort: http protocol: TCP name: http selector: {{- include "inputtest.selectorLabels" . | nindent 4 }}

cd cd inputtest nano values.yaml

service: type: nodePorty # written wrong port: 80

cd helm template --debug inputtest

and eventually also test against server

helm template inputtest --validate

```
### Advanced Testing mit chart-testing


### Reference

  * https://github.com/helm/chart-testing/
  * https://github.com/helm/chart-testing/blob/main/doc/ct_install.md

### Chart auf github veröffentlichen


### Prep
```

Create new public repo with README.md Go to Settings -> Pages -> an enable for branch "main" git clone the repo locally

```
### Locally pack, index and upload it.
```

git clone https://github.com/jmetzger/chart-test.git

guestbook must be present as folder with charts

helm package guestbook cp guestbook-0.1.0.tgz chart-test/ helm repo index chart-test/ git add . git commit -m "initial release" git push -u origin main

```
### Work with it
```

helm repo add githubrepo https://jmetzger.github.io/chart-test/ helm search repo guestbook helm repo list helm pull githubrepo/guestbook

```
## Sicherheit von helm-Chart

### Grundlagen / Best Practices



* https://sysdig.com/blog/how-to-secure-helm/

### Security Encrypted Passwords in helm




### Reference:

  * https://www.thorsten-hans.com/encrypted-secrets-in-helm-charts/
  * https://github.com/jkroepke/helm-secrets

### Alternative: SealedSecrets

  * https://dev.to/timtsoitt/argo-cd-and-sealed-secrets-is-a-perfect-match-1dbf

## Testing in Helm-Charts

### Testing in/von helm - charts


### Walkthrough
```

helm create demo helm install demo demo helm test demo

```
### Reference

  * https://helm.sh/docs/topics/chart_tests/
```

```
## Durchführung von Upgrades und Rollbacks von Anwendungen

## Helm in Continuous Integration / Continuous Deployment (CI/CD) Pipelines

## Tipps & Tricks

### Set namespace in config of kubectl
```

kubectl create ns mynamespace kubectl config set-context --current --namespace=mynamespace

```
### Create Ingress Redirect
```

cd helm create testprojekt cd testprojekt cd templates

```

mkdir routes/ cd routes nano 01-redirect.yaml

```
Schritt 1: Mit der Basis anfangen
```

apiVersion: networking.k8s.io/v1 kind: Ingress metadata: annotations: nginx.ingress.kubernetes.io/permanent-redirect: https://www.google.de nginx.ingress.kubernetes.io/permanent-redirect-code: "308" creationTimestamp: null name: destination-home namespace: my-namespace spec: rules:

- host: web.training.local http: paths:
  - backend: service: name: http-svc port: number: 80 path: /source pathType: ImplementationSpecific

```
### Schritt 2: values - file mit eigenen Werten ergänzen (Default - Werte)
```

cd ../..

nano values.yaml

Zeilen ergänzt.

Achtung: Eigenschaft UNBEDINGT ! ohne "-"

myRedirect: url: "http://www.google.de" code: 302

```
### Schritt 3: Variablen aus values in template einbauen
```

cd templates/routes

```

## nano 01-redirect.yaml

## Neue Fassung: Alle Änderungen beginnen mit Platzhalter - Zeichen {{

apiVersion: networking.k8s.io/v1 kind: Ingress metadata: annotations: nginx.ingress.kubernetes.io/permanent-redirect: {{ .Values.myRedirect.url }} nginx.ingress.kubernetes.io/permanent-redirect-code: {{ .Values.myRedirect.code | quote }} creationTimestamp: null name: destination-home namespace: my-namespace spec: rules:

- host: web.training.local http: paths:
  - backend: service: name: http-svc port: number: 80 path: /source pathType: ImplementationSpecific

```
Schritt 4: Test mit Default – Werten aus values.yaml
```

helm template ../..

achten auf ausgaben von Ingress

helm template ../.. | grep -A 40 "kind: Ingress"

```
### Schritt 5: Default – Werte überschreibung für Produktion mit speziellen prod-values.yaml (Name beliebig)
```

Empfehlung: ausserhalb des Charts anlegen

cd nano prod-values.yaml

```

```

myRedirect: url: "http://www.stiftung-warentest.de"

```

```

Testen wie folgt

helm template -f prod-values.yaml testprojekt

oder aber auch testen mit validate

helm template --validate -f prod-values.yaml testprojekt

oder aber direkt release installation

helm install --dry-run -f prod-values.yaml testprojekt

```
### Helm Charts – Development – Best practices
```

```
  * https://helm.sh/docs/howto/charts_tips_and_tricks/

## Integration mit anderen Tools

### yamllint für Syntaxcheck von yaml - Dateien
```

apt install -y yamllint

```


## Troubleshooting und Debugging

### helm template --validate - gegen api-server testen


### How ?
```

helm template guestbook --validate

```

```
```
```