Helm Paketmanagement

Agenda

1. 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 ?
2. Helm Installation und Konfiguration (inkl. kubectl)
    - Installation von kubectl unter Linux
    - Konfiguration von kubectl mit namespaces
    - Installation von helm unter Linux
    - Installation bash completion
3. Helm - Spickzettel
    - Wichtig: Helm Spickzettel
4. Arbeiten mit helm - charts
    - Installation, Upgrade, Uninstall helm-Chart exercise
    - Informationen aus nicht installierten Helm-Charts bekommen
    - Chart runterladen und evtl. entpacken und bestimmte Version
5. Tipps & Tricks
    - kubernetes manifests mit privatem Repo
    - helm chart mit images auf privatem Repo
6. Spezial: Umgang mit Einrückungen
    - Whitespaces meistern mit "-"
    - Exercise Whitespaces
7. Type - Conversions
    - Exercise toYaml
8. Flow Control
    - if
    - with
    - range
9. Helm mit gitlab ci/cd
    - Helm mit gitlab ci/cd ausrollen

Backlog

1. Grundlagen

- Feature / No-Features von Helm
- TopLevel Objekte

2. Helm-Befehle und -Funktionen
   - Repo einrichten
   - Suche in Repo und Artifacts Hub
   - Anzeigen von Informationen aus dem Chart von Online
   - Upgrade und auftretende Probleme
3. Helm Repository
   - Die wichtigsten Repo-Befehle
4. Struktur von Helm - Charts
   - Überblick
5. Grundlagen Helm-Charts
   - Testumgebung und Spaces (2 Themen)
6. 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
7. Sicherheit von helm-Chart
   - Grundlagen / Best Practices
   - Security Encrypted Passwords in helm
8. Testing in Helm-Charts
   - Testing in/von helm - charts
9. Durchführung von Upgrades und Rollbacks von Anwendungen
10. Helm in Continuous Integration / Continuous Deployment (CI/CD) Pipelines
11. Tipps & Tricks
    - Set namespace in config of kubectl
    - Create Ingress Redirect
    - Helm Charts - Development - Best practices
12. Integration mit anderen Tools
    - yamllint für Syntaxcheck von yaml - Dateien
13. Troubleshooting und Debugging
    - helm template --validate - gegen api-server testen

Helm Einfuehrung

Was ist helm ?

- Paketmanager für Kubernetes
- Ermöglicht 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

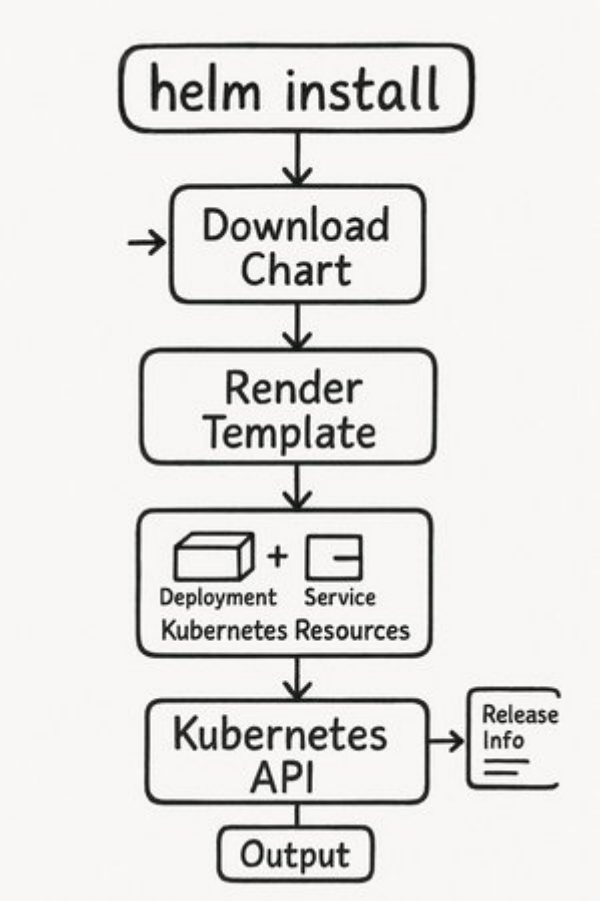
- Verzeichnis z.B. meine-app (und in dem Verzeichnis die bekannte Struktur von oben)
- tar.gz (Tape-Archive mit gnuzip komprimiert)
- URL

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 -a /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
```

Arbeiten mit helm - charts

Installation, Upgrade, Uninstall helm-Chart exercise

Install

```
## 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>
```

Exercise: Upgrade to new version

```
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 --namespace app-<nameskuerzel> -f
prod/values.yaml
```

Umschauen

```
kubectl -n app-<namenskuerzel> get pods
helm -n app-<namenskuerzel> status my-nginx
helm -n app-<namenskuerzel> list
helm -n app-<namenskuerzel> history
```

Uninstall

```
helm -n app-<namenskuerzel> uninstall my-nginx
## namespace wird nicht gelöscht
## 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 trriger

- 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

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
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 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
```

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

```
## 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

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)
```

helm create iftest cd iftest/templates rm -f *.yaml

```
### Step 2: values-file erweitern
```

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/

### 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 }}

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

nano cm.yaml

```

```

{{- 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)
```

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

```
#### Step 2c: Changing the scope
```

{{- with .Values.favorite }} drink: {{ .drink | default "tea" | quote }} food: {{ .food | upper | quote }} release: {{ $.Release.Name }} {{- end }}

```


### range


### Preparation
```

helm create testenv cd testenv/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 }}

```
## 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

![image](https://github.com/user-attachments/assets/5e88593b-5b31-4adf-a2bb-
e5e9a5129be5)

### 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

![image](https://github.com/user-attachments/assets/b5168cf3-dd74-4d86-becf-
e807985dd471)

### 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

## 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](/helm/internals.md)


### 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-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 abszuschneiden

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 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

```

```

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