

corewire

Helm

Helm Values

Folien-Hinweis

- Space, Page down: Nächste Folie
- Page up: Vorherige Folie
- ESC, o: Übersicht

[Zur Kapitelübersicht](#)

Lernziele

- Vorhandene Helm Charts gezielt über Values konfigurieren
- Default Values, Release Values und User-Supplied Values unterscheiden
- Kleine Anpassungen mit `--set`, größere mit `values.yaml` vornehmen
- Gerenderte Manifeste mit `helm template` analysieren, ohne den Cluster zu ändern

Woher kommen Helm Values?

1. Default Values aus dem Chart
2. Values-Dateien mit `-f` oder `--values`
3. Einzelne Overrides mit `--set`

Später angegebene Werte überschreiben frühere Werte.

Default Values anzeigen

```
helm show values corewire/docker-demoapp
```

Typische Bereiche im Chart:

- replicaCount
- image
- service
- ingress
- database

Chart Values vs. Release Values

Chart Defaults anzeigen

```
helm show values corewire/docker-demoapp
```

Explizit gesetzte Release Values anzeigen

```
helm get values docker-demoapp --namespace helm-demo
```

Alle aktiven Values des Releases anzeigen

```
helm get values docker-demoapp --namespace helm-demo --all
```

Values mit `--set`

Für kleine, punktuelle Änderungen:

```
helm upgrade docker-demoapp corewire/docker-demoapp \  
  --namespace helm-demo \  
  --set replicaCount=5
```

Auch verschachtelte Keys sind möglich:

```
--set database.enabled=true
```

Wann ist `--set` sinnvoll?

Gut geeignet für:

- einzelne Zahlen, Booleans oder Strings
- schnelle Tests in Shell oder CI
- einmalige kleine Overrides

Wann ist `--set` sinnvoll?

Weniger geeignet für:

- viele Werte gleichzeitig
- komplexe Strukturen
- versionierte Konfiguration

Values über Datei pflegen

custom-values.yaml

```
replicaCount: 7

service:
  type: ClusterIP

database:
  enabled: true

ingress:
  enabled: true
  host: demoapp.cluster.example.test
```

Upgrade mit Values-Datei

```
helm upgrade docker-demoapp corewire/docker-demoapp \  
  --namespace helm-demo \  
  --values custom-values.yaml
```

Vorteile:

- lesbar
- versionierbar
- wiederverwendbar

Priorität der Werte

Beispiel:

```
helm upgrade docker-demoapp corewire/docker-demoapp \  
  --namespace helm-demo \  
  --values base.yaml \  
  --values prod.yaml \  
  --set replicaCount=5
```

Reihenfolge der Priorität:

1. Chart Defaults
2. `base.yaml`
3. `prod.yaml`
4. `--set replicaCount=5`

helm template

helm template: Lokales Rendern

```
helm template docker-demoapp corewire/docker-demoapp \
--namespace helm-demo
```

Eigenschaften:

- erzeugt Kubernetes YAML lokal
- erstellt kein Release
- verändert keinen Cluster

helm template mit Overrides

Mit `--set`

```
helm template docker-demoapp corewire/docker-demoapp \  
  --namespace helm-demo \  
  --set replicaCount=4
```

Mit Values-Datei

```
helm template docker-demoapp corewire/docker-demoapp \  
  --namespace helm-demo \  
  --values custom-values.yaml
```

Template-Ausgabe analysieren

In Datei schreiben:

```
helm template docker-demoapp corewire/docker-demoapp \  
  --namespace helm-demo \  
  --values custom-values.yaml > rendered.yaml
```

helm upgrade vs. helm template

helm upgrade

- speichert eine neue Release-Revision
- ändert Ressourcen im Cluster

helm template

- rendert nur lokal
- eignet sich für Review, Debugging und CI Checks

Typische Stolperfallen

- falscher Key-Pfad in `--set`
- Typkonflikte zwischen String, Zahl und Boolean
- alte Release Values bleiben erhalten, obwohl neue Defaults erwartet werden
- Template-Ausgabe wird nicht geprüft, bevor ein Upgrade gegen den Cluster läuft

Empfehlenswerter Workflow

1. `helm show values` für den Ausgangspunkt
2. kleine Tests mit `--set`
3. stabile Konfiguration in `custom-values.yaml`
4. Ergebnis mit `helm template` rendern
5. erst danach `helm upgrade` ausführen

Demo: Helm Values und `helm template`

Lab

- Gehen Sie auf: <https://learn.corewire.de>
- Navigieren Sie nach:
 - Helm
 - Aufgaben
 - Helm Values

Zur Kapitelübersicht

- Vorheriges Kapitel: [Helm Grundlagen](#)
- Nächstes Kapitel: [Helm Charts erstellen](#)