

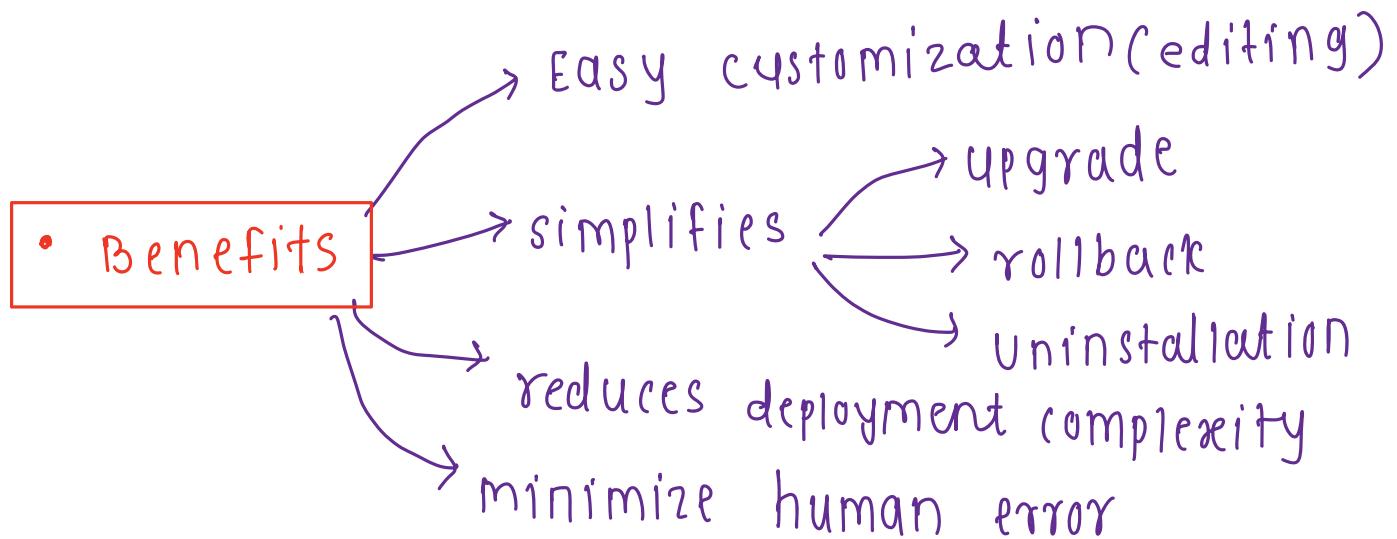
Helm Simplified



Handwritten By:
Vaibhav Matkar

Helm → package manager for kubernetes

- Treats collection of interrelated objects as a package



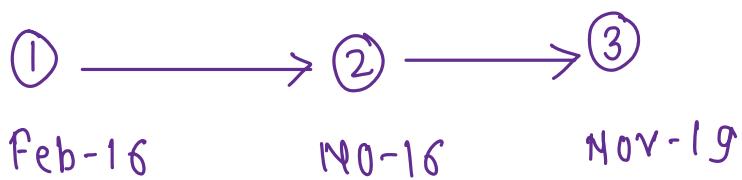
• prerequisites to installation

- functional k8 cluster
- Installed kubectl cli
- kubeconfig file → containing correct credentials to connect to k8 cluster.

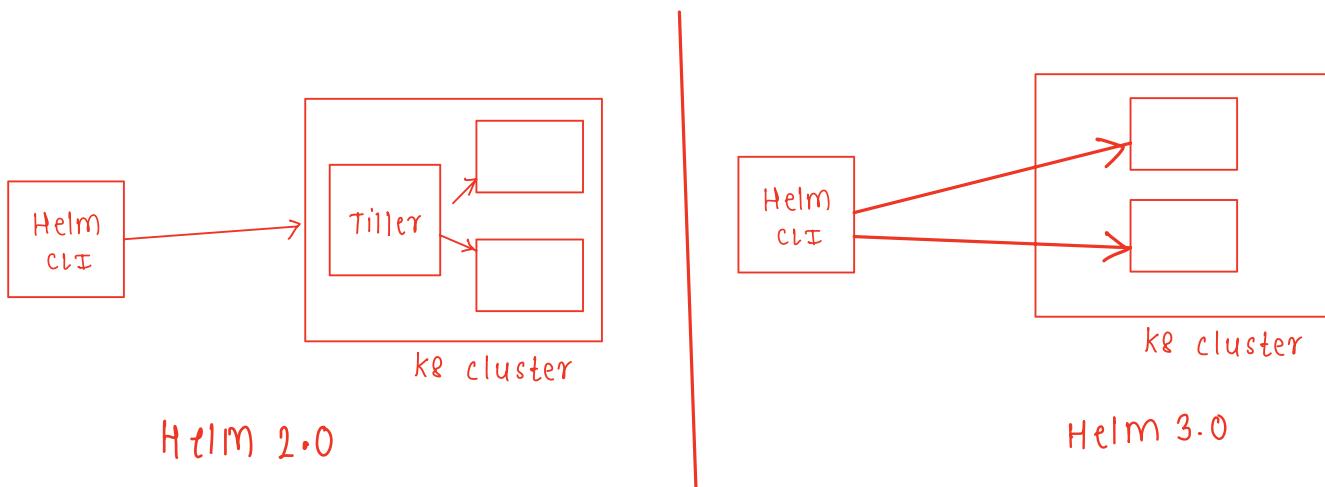
• Ways to install

- using snap
- using apt on debian/ubuntu
- using pkg

• Helm History →



• Helm Architecture



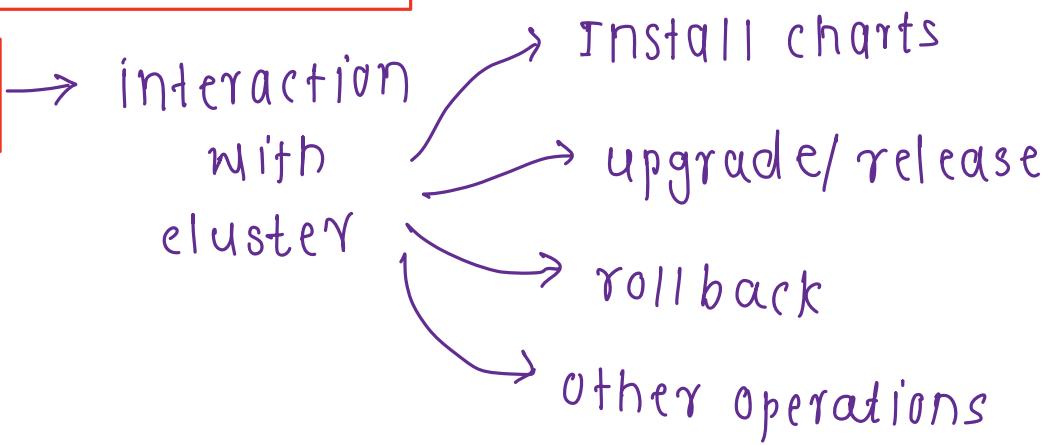
- Need of Tiller
 - due to absence of RBAC
 - due to absence of CRD

Three-way strategic merge patch - intelligent rollbacks & upgrades.

- Each revision is similar to snapshot
 - 1st stage → original → Last applied
 - 2nd stage → current → live
 - 3rd stage → New

key components of helm

① Helm CLI



② charts

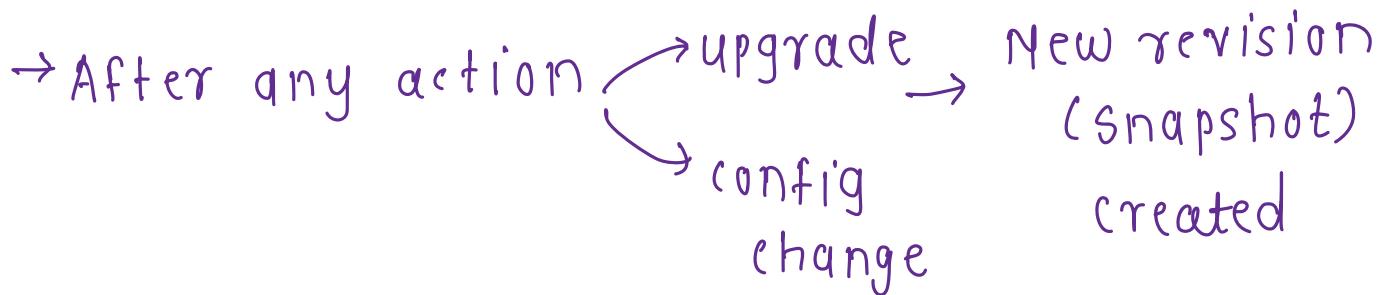
→ Reusable deployment packages



contains files that contains instructions
to create kubernetes objects

③ Releases

→ Release is created once package
is deployed to the cluster.

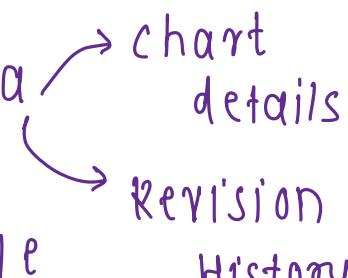


④ Metadata

→ stores release metadata



It makes deployment history available
to everyone working on the cluster.

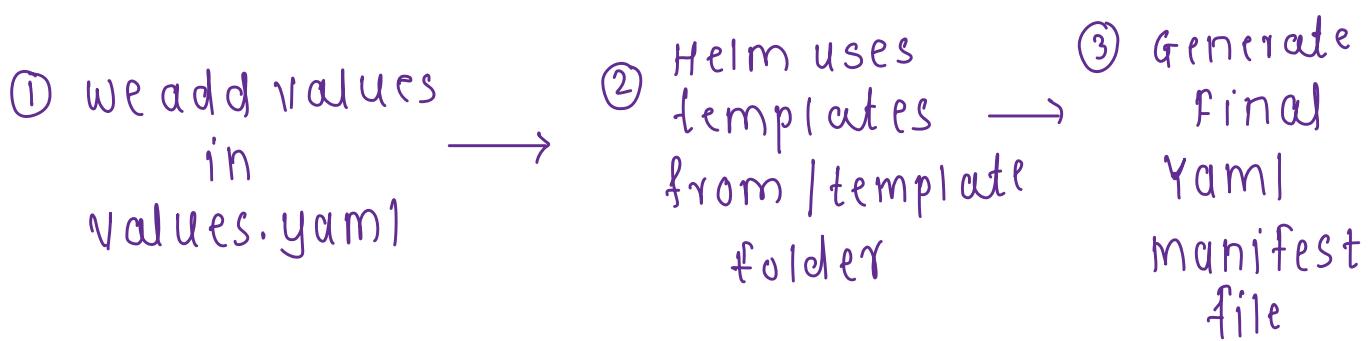


Helm templating

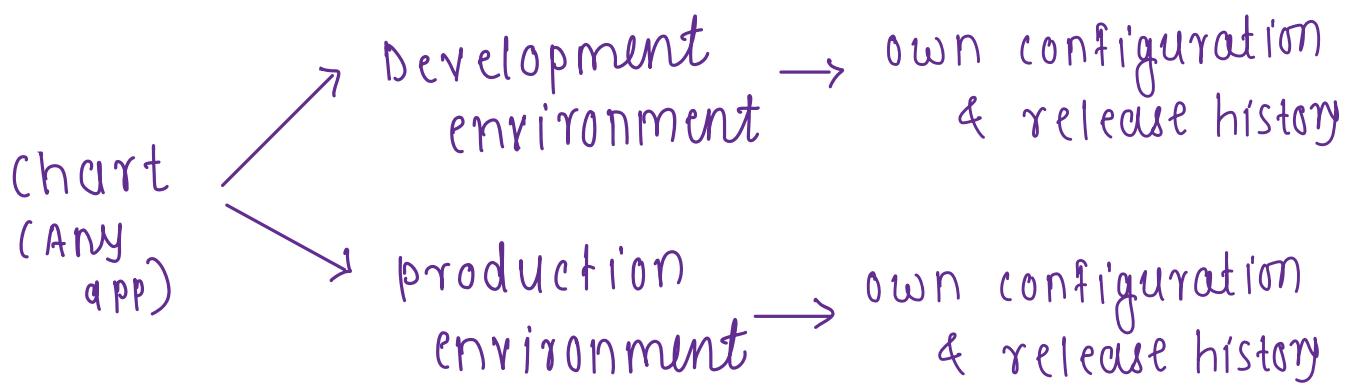


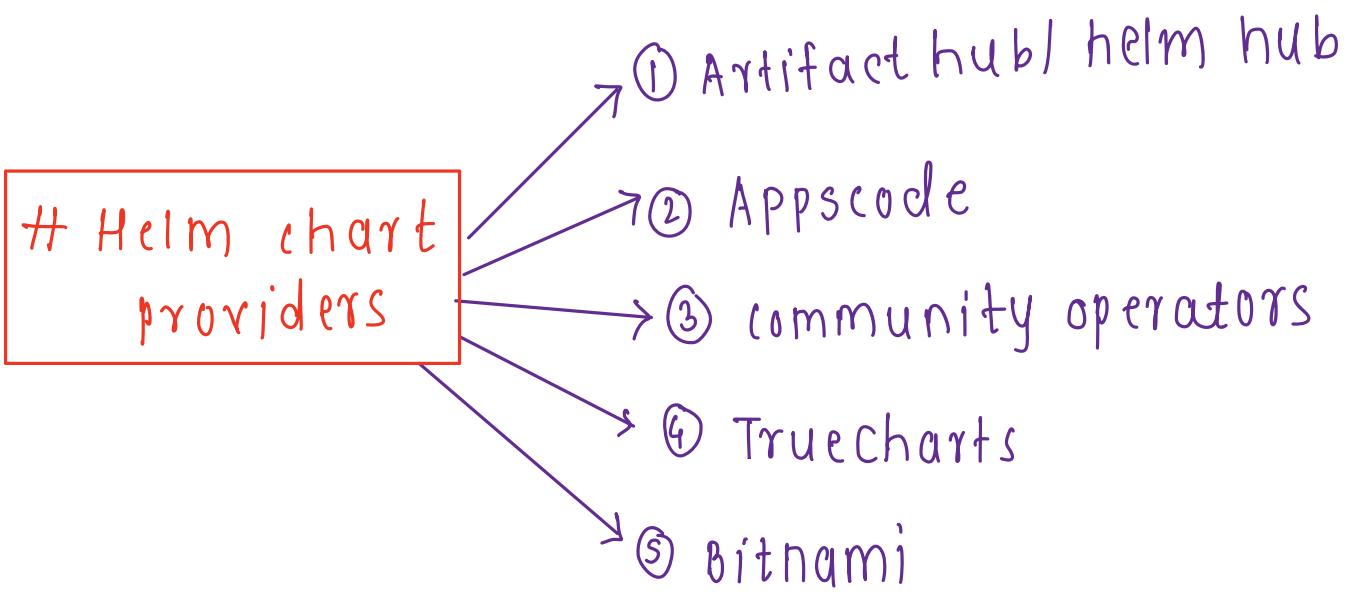
-

- Helm uses templating to generate K8 yaml files dynamically instead of writing entire static yaml file.



Helm release management





Helm chart directory structure

- ① /templates → contains templates for resources
- NOTES.yaml
 - -helpers.tpl → define reusable templates
 - different.yaml files
 - tests → contains tests that validate our chart
- ② values.yaml → defines configuration parameters
- ③ chart.yaml → holds chart metadata

- i) `apiVersion` → v2 for Helm 3 || v1 for Helm 2
- ii) `appVersion` → version of packaged appn
- iii) `version` → chart version
- iv) `name, description, type` → chart details
- v) `dependencies` → List dependent charts
- vi) `keywords, maintainers,`
home and icon

↓

provides additional metadata useful
for chart discovery & reference.

- ④ `/charts` → contains dependent charts
- ⑤ `.helmignore` → files to ignore when packaging helm charts
- ⑥ `LICENSE` → open source published license info
- ⑦ `README.MD` → Helm chart documentation

Helm basic commands

① helm --help → commands info & usage

```
$ helm --help

The Kubernetes package manager

Common actions for Helm:
- helm search: search for charts
- helm pull: download a chart to your local directory for inspection
- helm install: deploy a chart onto Kubernetes
- helm list: list chart releases

Usage:
  helm [command]

Available Commands:
  completion  generate autocompletion scripts for the specified shell
  create      create a new chart with the given name
  dependency  manage a chart's dependencies
  env         helm client environment information
  get         download extended information of a named release
  help        Help about any command
  history     fetch release history
```

② helm repository mgmt

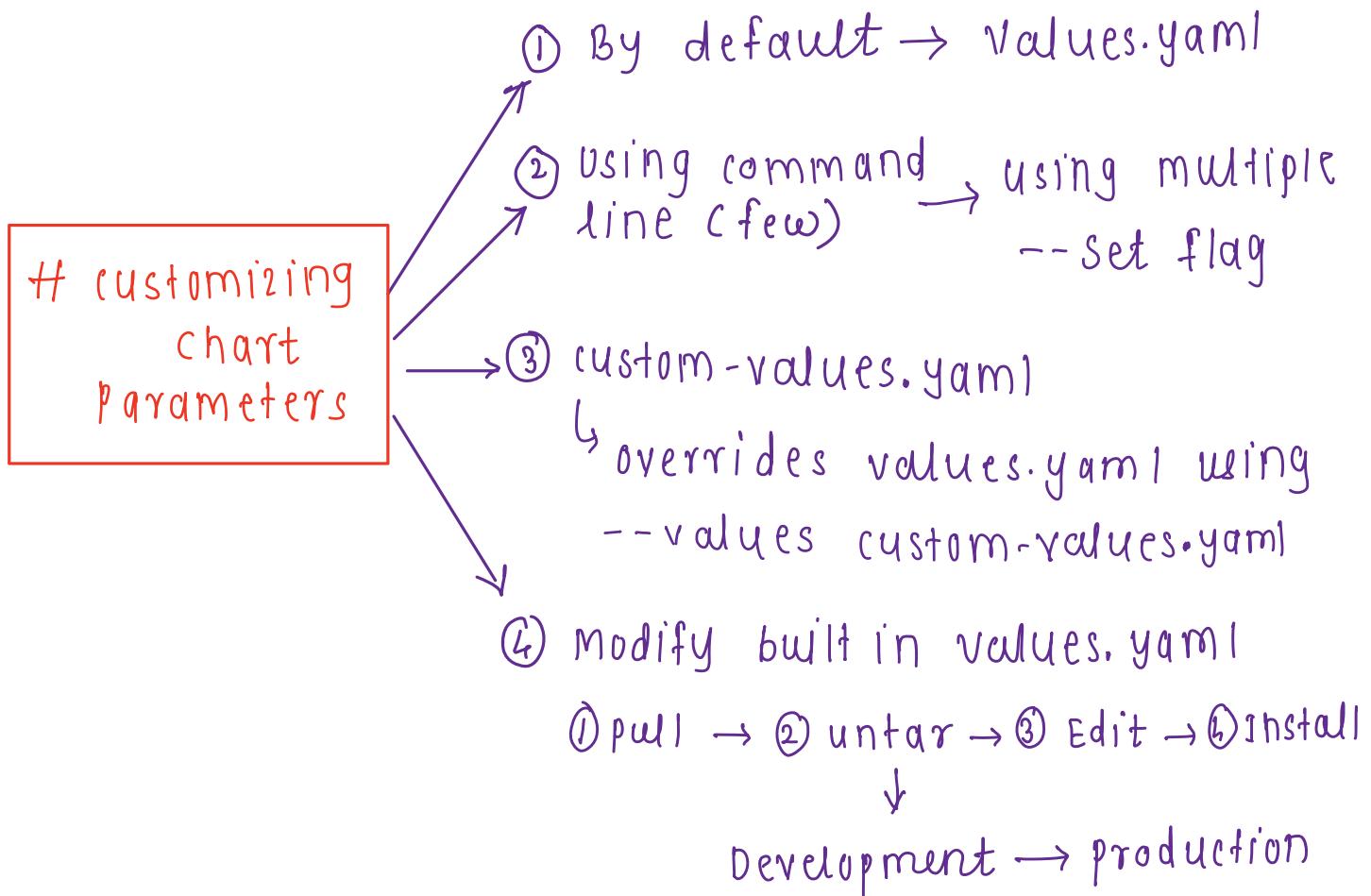
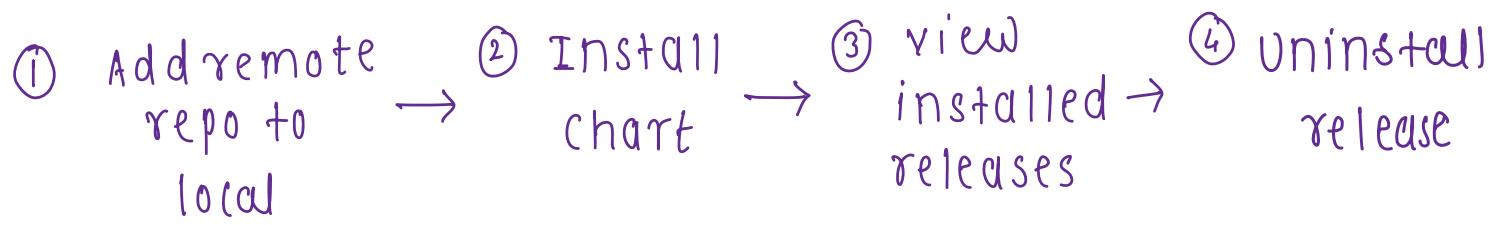
```
$ helm repo --help

This command consists of multiple subcommands to interact with chart repositor
It can be used to add, remove, list, and index chart repositories.

Usage:
  helm repo [command]

Available Commands:
  add      add a chart repository
  index    generate an index file given a directory containing packaged chart
  list     list chart repositories
  remove   remove one or more chart repositories
  update   update information of available charts locally from chart reposito
```

Application deployment using helm chart



Lifecycle management with helm

- Release → collection of k8 objects



Install → upgrade → Roll back
(Rel-1) (Rel-2) (Rel-3)

- Note ⇒ For external data upgrade
 - chart hooks
 - Automations

Names

Static Names

- resource name is fixed
- metadata:
name: myapp
- may cause conflict

Templated Names

- resource name is generated dynamically
- metadata:
name: {{ release.Name }}
--service
- No conflict

Helm chart validation

① Linting → Syntax/Typo
format/misspellings

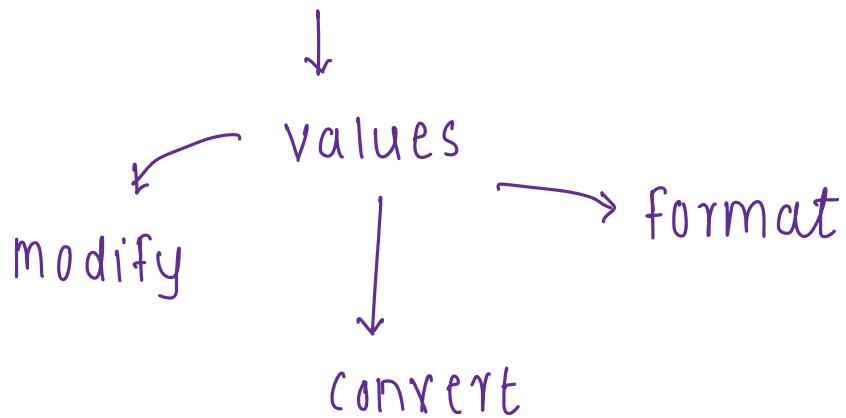
⇒ helm lint .
/path

② Icon → logo URL for chart

③ Template rendering → checks
manifest file is produced
correctly according to logic
⇒ helm template release.
/path

④ simulating installation with
dry run → validate manifest
against K8 standards
⇒ helm install release-name
. /path --dry-run

Function in helm



- syntax: {{ Funct-Name Arg1 Arg2 }}

- commonly used functions:

- ① quote
- ② upper
- ③ lower
- ④ title
- ⑤ default
- ⑥ indent
- ⑦ replace
- ⑧ shuffle

pipeline/ pipe

→ Allows to chain multiple functions together

- syntax: {{ function1 | function2 }}

conditional in helm

① if

```
 {{- if argument }}
```

condition

```
 {{ -end }}
```

→ Here (-) at begining removes whitespaces

② if else if

```
 {{- if argument }}
```

condition

```
 {{- else if eq argument "value" }}
```

condition

```
 {{ -end }}
```

→ Here, eq is helper function for equality

with block

- syntax :

```
 {{- with argument }}  
   key1: {{ value1 }}  
   key2: {{ value2 }}  
 {{- end }}
```

← optimized →
traversing

```
key1: {{ argument.value1 }}  
key2: {{ argument.value2 }}
```

← Repetitive →
traversing

Nested with block

- syntax :

```
 {{- with argument }}  
   {{- with argument }}  
     key1: {{ value1 }}  
     key2: {{ value2 }}  
   {{- end }}  
   {{- with argument }}  
     {{- with argument }}  
       key3: {{ value3 }}  
       key4: {{ value4 }}  
     {{- end }}  
   {{- end }}
```

range in helm

- syntax:

```
 {{-range argument?} }  
 -{{.}}  
 {{-end}}
```

→ useful for configmap

Named templates

- define directive can be used to separate duplicate codebase into (-helpers.tpl) file where (-) ignores this file when generating k8 manifest.
- syntax:

```
 {{-define "name"} }  
 key1: {{val1}}  
 key2: {{val2}}  
 {{-end}}
```

This is updated
in
-helpers.tpl

- {{ - template "name". }}

} updated in
resource manifest
file

- {{ - include "key". | indent (n) }}

↳ correct indentation / readability

Hooks in helm



Kubernetes resource
used to execute events



- ① After installation
- ② Before deletion
- ③ Before installation

- eg.
- DB migration
 - config generation
 - cleanup job
 - initialize scripts

→ Hooks are defined in metadata annotations

- syntax: metadata:

annotations:

"helm.sh/hook": hook event

common hook events:

- i) pre-install
- ii) post-install
- iii) pre-delete
- iv) post-delete
- v) pre-upgrade
- vi) post-upgrade
- vii) pre-rollback
- viii) post-rollback
- ix) test

• **Hook weight** → Used to decide order of execution of multiple hooks.

• Less weight → Execute 1st

• **Hook policies** → Decides to keep hook or not after it's work.

- i) before-hook-creation → Delete old hook before new hook creation
- ii) hook-succeeded
- iii) hook-failed

packaging and signing charts

① Generate GPG [GNU Privacy Guard] keys

```
gpg --quick-generate-key "signer name"
```

```
gpg --full-generate-key "signer name" → prod env
```

② convert this key (new format) to older format preferred by helm

```
gpg --export-secret-keys > ~/.gnupg/secring.gpg
```

③ packaging chart with signature

```
helm package --sign --key 'signer-name'  
--keyring ~/.gnupg/secring.gpg . /chart-directory
```

↓
This creates 2 outputs

• tgz archive

• tgz.prov file

↓

- SHA256 hash of chart
- PGP signature verifying archives integrity

④ verifying signed helm chart

- gpg --export 'key-ID' > output-file-name
- helm verify --keyring ./public-key-file
 ./package-directory

↑ uploading charts

- chart repo contains
 - packaged chart (.tgz)
 - Index file (index.yaml)
 - ↓
 - ↳ checksum / description / URL
 - provenance file (.prov)
 - ↳ cryptographic signature signed by private key

• creation of index.yaml

→ helm repo index chart-directory --url chart-repo

↳ Helm see index.yaml file → see chart URL in it and download .tgz chart file

- chart upload options
 - Google cloud storage
 - Amazon S3
 - digitalocean object store
 - github pages