HELM

Simplified Notes

Handwritten By:
Virendra Beniwal

https://www.linkedin.com/in/virendra-beniwal-893276177/

Helm → package manager for kubernetes

↓

- Treats collection of interrelated objects as a package

- Benefits
  - Easy customization (editing)
  - simplifies
    - upgrade
    - rollback
    - uninstallation
  - reduces deployment complexity
  - minimize human error

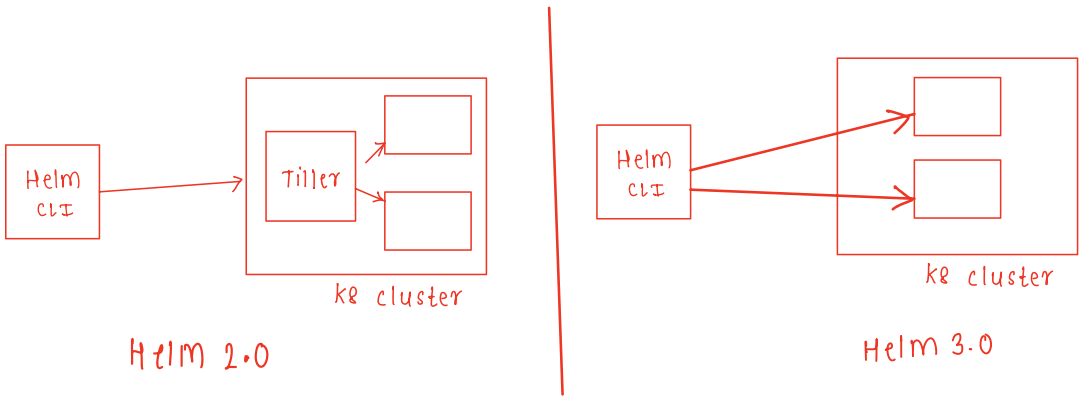
- prerequisites to installation
  - functional k8 cluster
  - Installed kubectl CLI
  - kubeconfig file → containing correct credentials to connect to k8 cluster.

- ways to install
  - using snap
  - using apt on debian/ubuntu
  - using pkg

- Helm History → ① → ② → ③

Feb-16 NO-16 NOV-19

- Helm Architecture



- Need of Tiller → due to absence of RBAC
- Need of Tiller → 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 → interaction with cluster
- Install charts
- upgrade/release
- rollback
- other operations

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.

→ After any action
- upgrade → New revision (snapshot) created
- config change

④ Metadata → stores release metadata
- chart details
- Revision History

↓

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

Helm templating

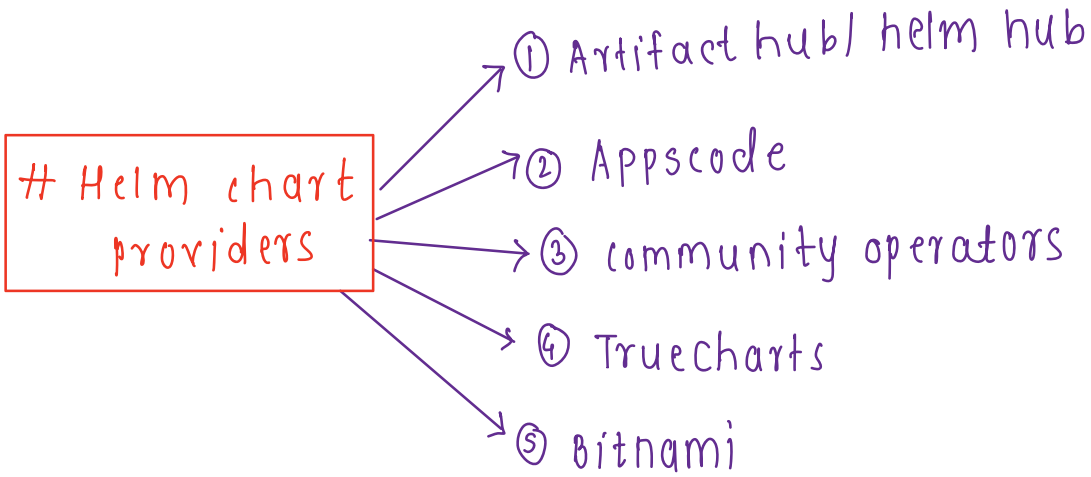
- Helm uses templating to generate K8 Yaml files dynamically instead of writing entire static yaml file.

① we add values in values.yaml → ② Helm uses templates from /template folder → ③ Generate final Yaml manifest file

Helm release management

Chart (Any app) → Development environment → own configuration & release history

Chart (Any app) → production environment → own configuration & release history



Helm chart directory structure

① /templates → contains templates for resources

- → NOTES.txt
- → _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 appln

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

① Add remote repo to local → ② Install chart → ③ view installed releases → ④ Uninstall release

customizing chart parameters

① By default → values.yaml

② using command line (few) → using multiple --set flag

③ custom-values.yaml
↳ overrides values.yaml using --values custom-values.yaml

④ Modify built in values.yaml
① pull → ② untar → ③ Edit → ④ Install
↓
Development → production

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

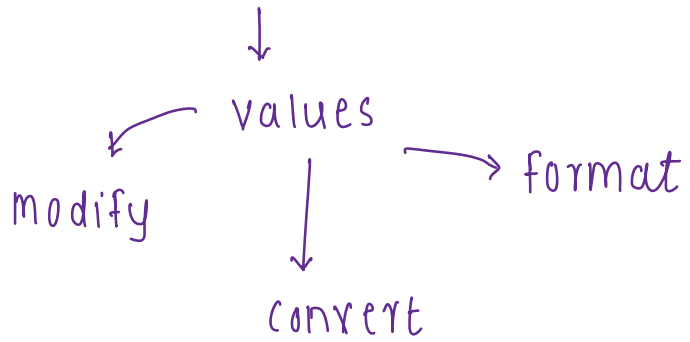
Helm chart validation

① Linting → Syntax/ Typo/ format/ mispells
⇒ helm lint ./path
• Icon → logo URL for chart

② Template rendering → checks manifest file is produced correctly according to logic
⇒ helm template release ./path

③ 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: {{ function 1 | function 2 }}

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.val1 }}
key2 : {{ argument.val2 }}
```

← Repetative traversing →

Nested With block

- syntax:

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