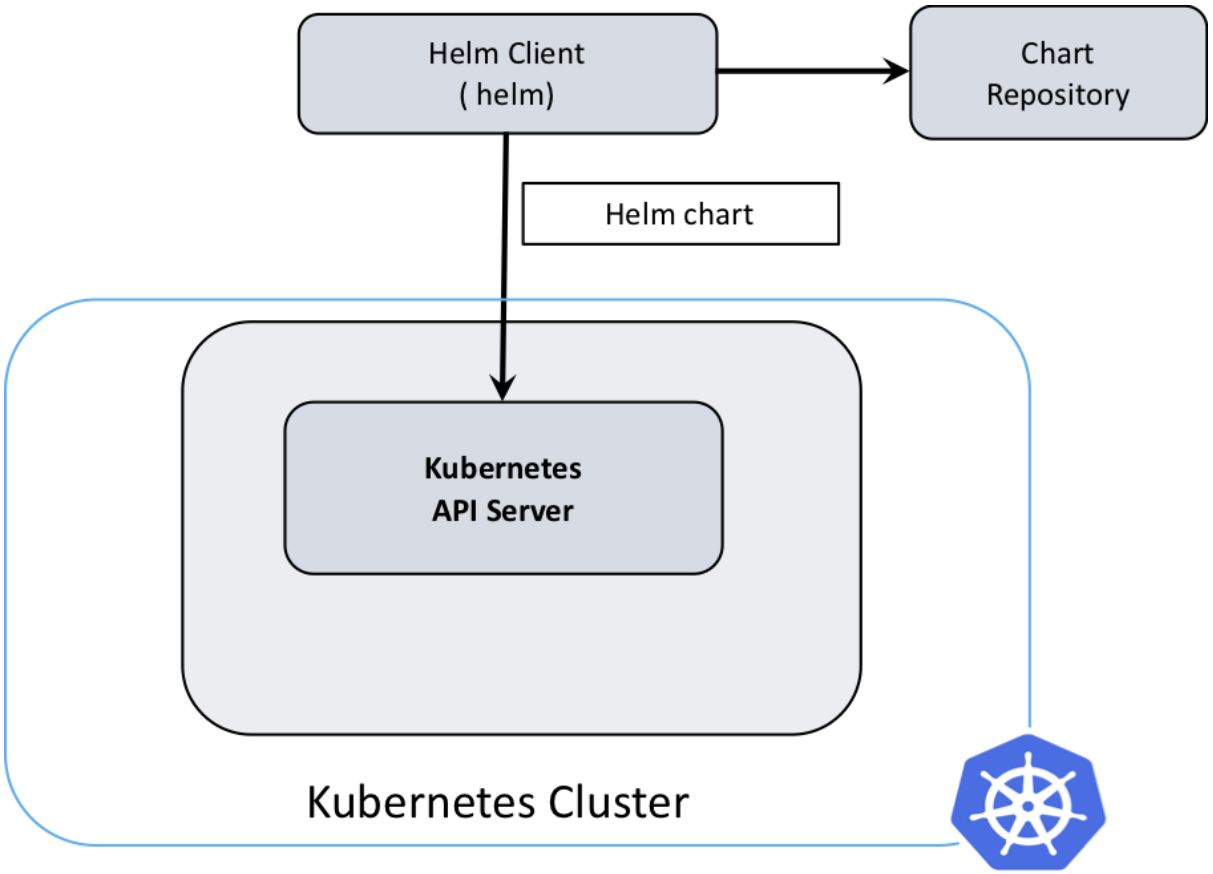
Module: Helm for Kubernetes — Detailed Notes

1. Introduction to Helm 3



1.1 What is Helm?

Helm is the **package manager for Kubernetes**, similar to:

- apt for Ubuntu
- yum for RedHat
- npm for Node.js

Helm helps you:

- Package Kubernetes YAMLs into **charts**
- Manage releases of applications
- Perform upgrades, rollbacks, and version control
- Reduce duplication using templates

1.2 Why Helm 3?

Helm 3 significantly improved security and architecture:

Feature	Helm 2	Helm 3
Server-side component	Tiller required	**No Tiller** (more secure)
RBAC issues	Complex	Simple, native K8s model
Release storage	ConfigMaps/Secrets	Secrets only
CRD handling	Manual	Built-in hooks

1.3 Key Concepts

Term	Meaning
Chart	A packaged app (template + values)
Release	A running instance of a chart
Repository	Storage for charts (ArtifactHub, S3, Git)
Values.yaml	Input parameters to templates
Templates	YAML + Go templating language

1.4 Helm CLI Basics

helm version

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

helm search repo nginx

helm install my-nginx bitnami/nginx

helm upgrade my-nginx bitnami/nginx

helm rollback my-nginx 1

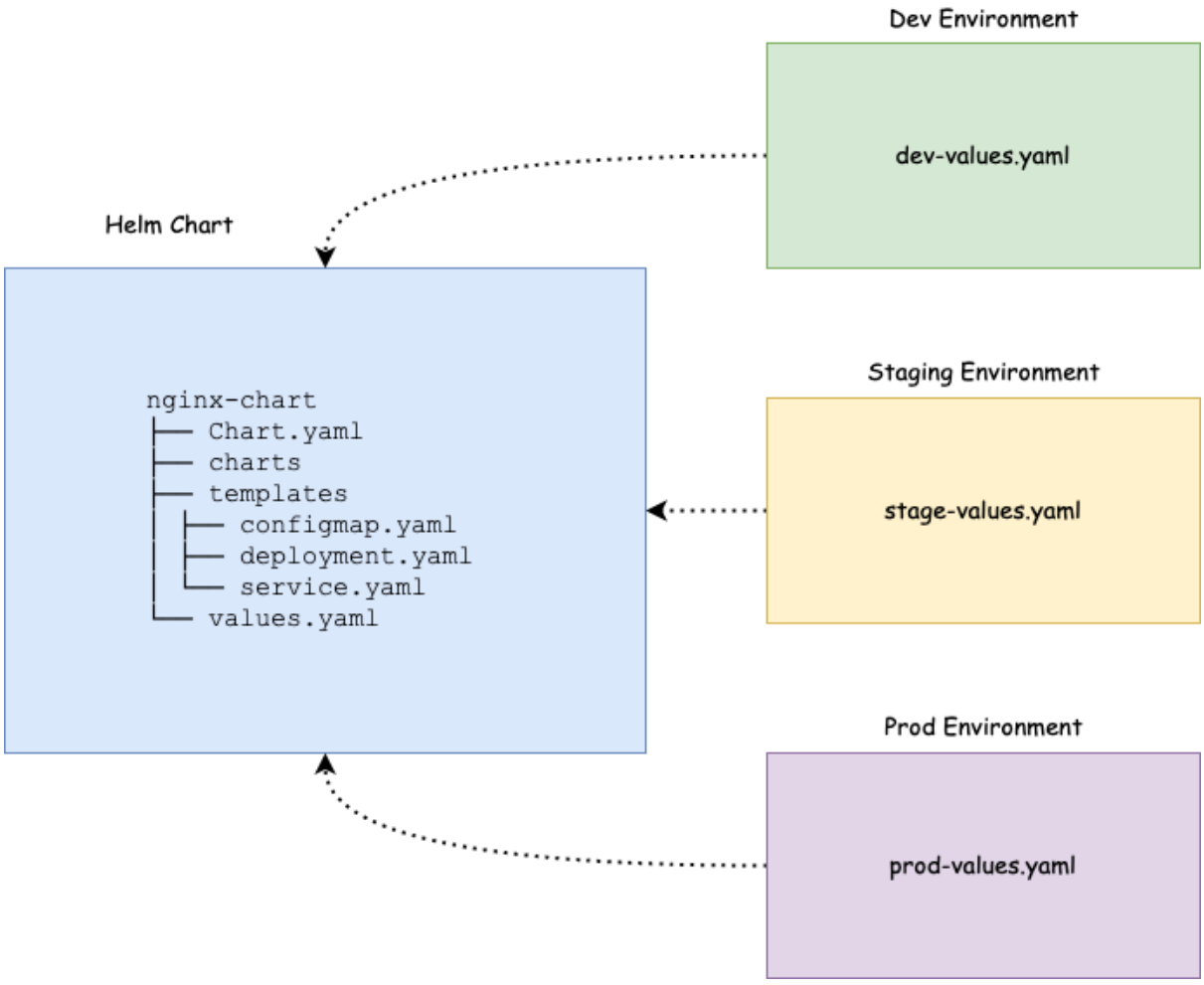
helm uninstall my-nginx

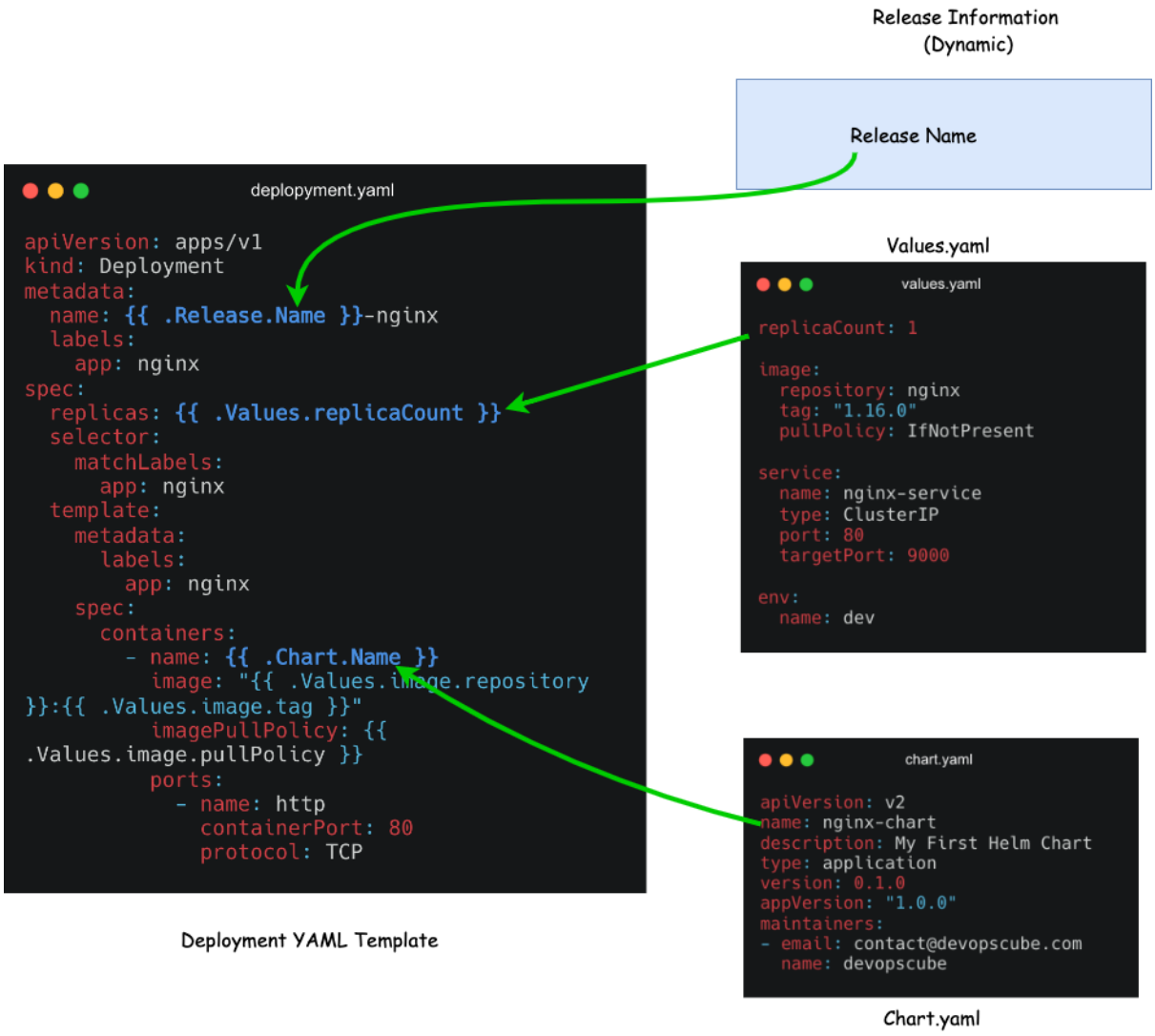
---

2. Creating Helm Charts

$ helm create mychart

```
mychart
├── Chart.yaml          # Information about your chart, metadata, version and dependency
├── charts              # Charts that this chart depends on
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml         # The default values for your templates
```





2.1 Create a New Chart

helm create myapp

The generated structure:

myapp/

Chart.yaml

values.yaml

templates/

deployment.yaml

service.yaml

ingress.yaml

_helpers.tpl

tests/

2.2 Important Files

Chart.yaml

Metadata about your application.

apiVersion: v2

name: myapp

description: Example Helm chart

version: 0.1.0

appVersion: "1.0.0"

values.yaml

Defines all configurable parameters.

replicaCount: 2

image:

repository: myrepo/myapp

tag: "1.0.0"

pullPolicy: IfNotPresent

Templates Directory

Contains Kubernetes manifest files using Go templating syntax.

2.3 Install Chart

helm install myapp-release myapp/

2.4 Package and Publish

helm package myapp/

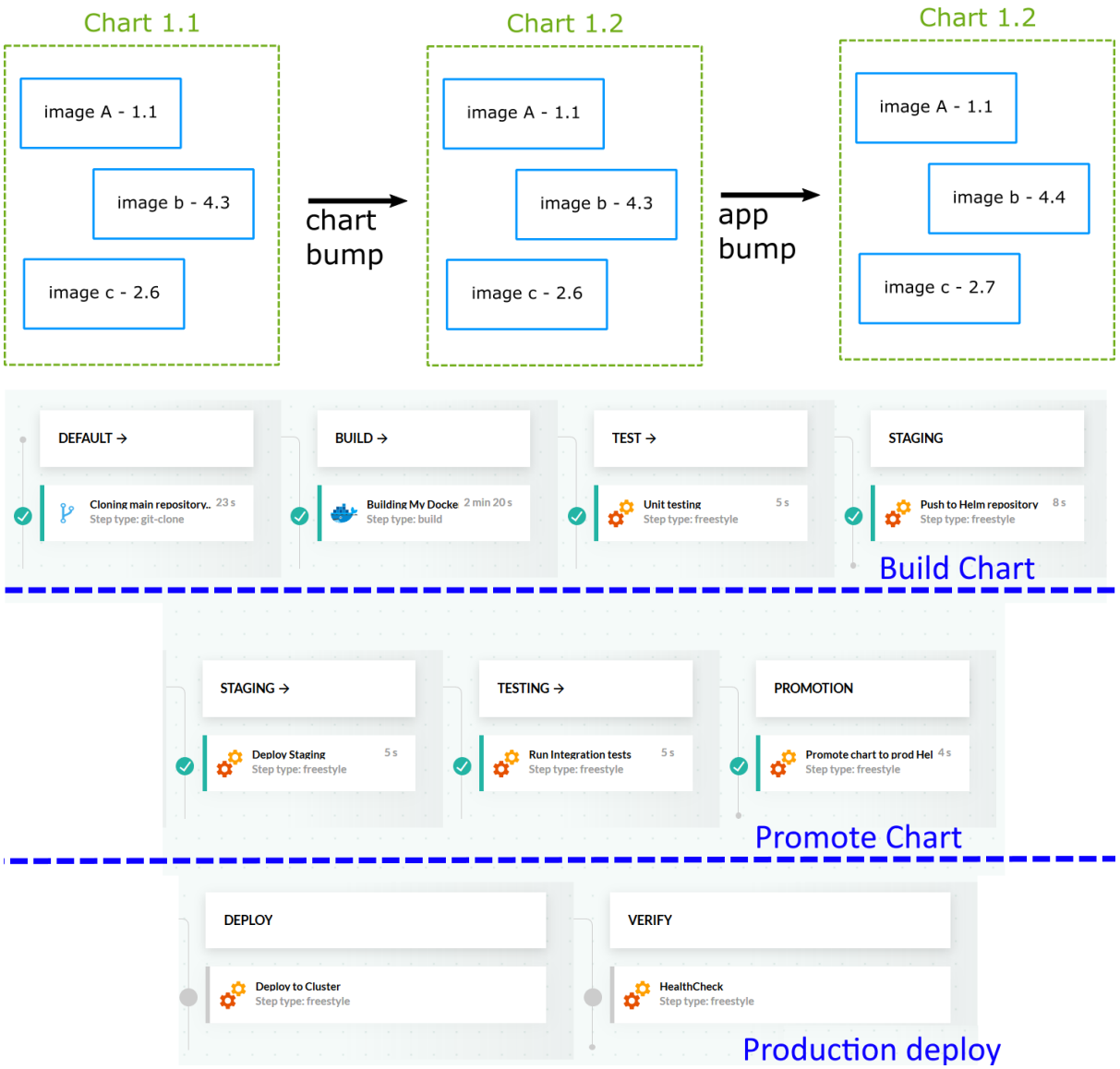
helm repo index .

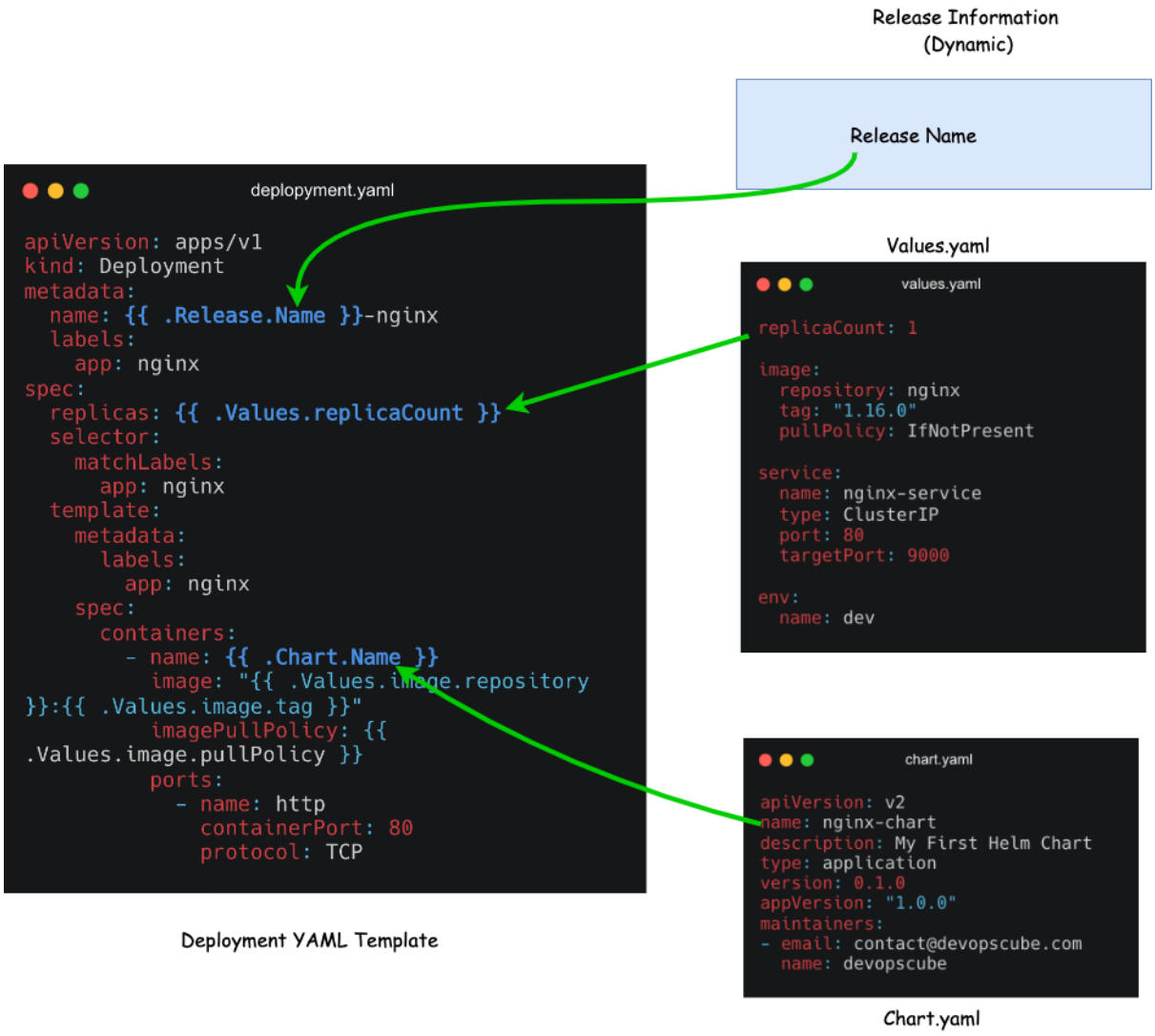
Upload to:

- GitHub Pages
- S3 bucket
- ArtifactHub

---

3. Templating Best Practices





Helm templates use **Go template syntax**, enabling:

- Variables
- Conditionals
- Loops
- Functions
- Named templates

3.1 Use _helpers.tpl for reusable templates

{{- define "myapp.fullname" -}}

{{ .Release.Name }}-{{ .Chart.Name }}

{{- end -}}

Use it in templates:

```
metadata:
  name: {{ include "myapp.fullname" . }}
```

3.2 Avoid Hardcoding — Always Use Values.yaml

Bad:

```
replicas: 3
```

Good:

```
replicas: {{ .Values.replicaCount }}
```

3.3 Use Conditionals

Example: enable Ingress only if turned on:

```
{{ if .Values.ingress.enabled }}
# ingress manifest
{{ end }}
```

3.4 Loops for ConfigMaps/Secrets

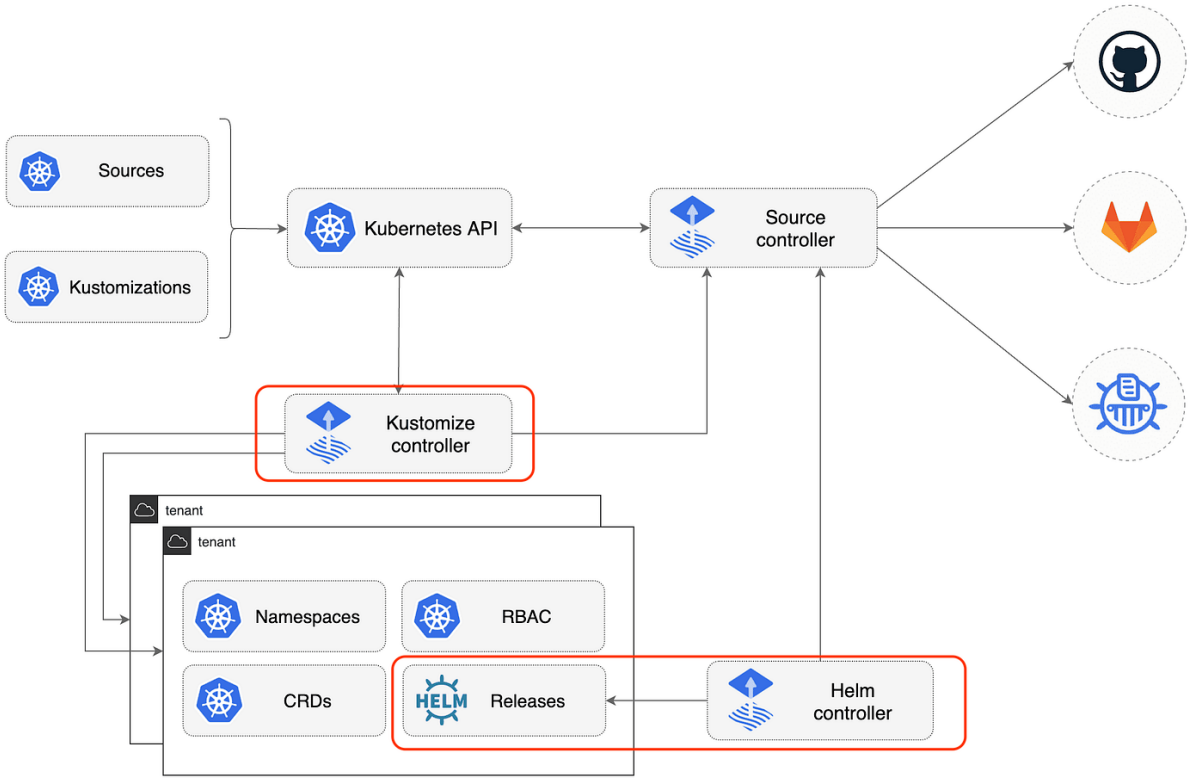
```
data:
{{- range $key, $value := .Values.config }}
  {{ $key }}: {{ $value | quote }}
{{- end }}
```

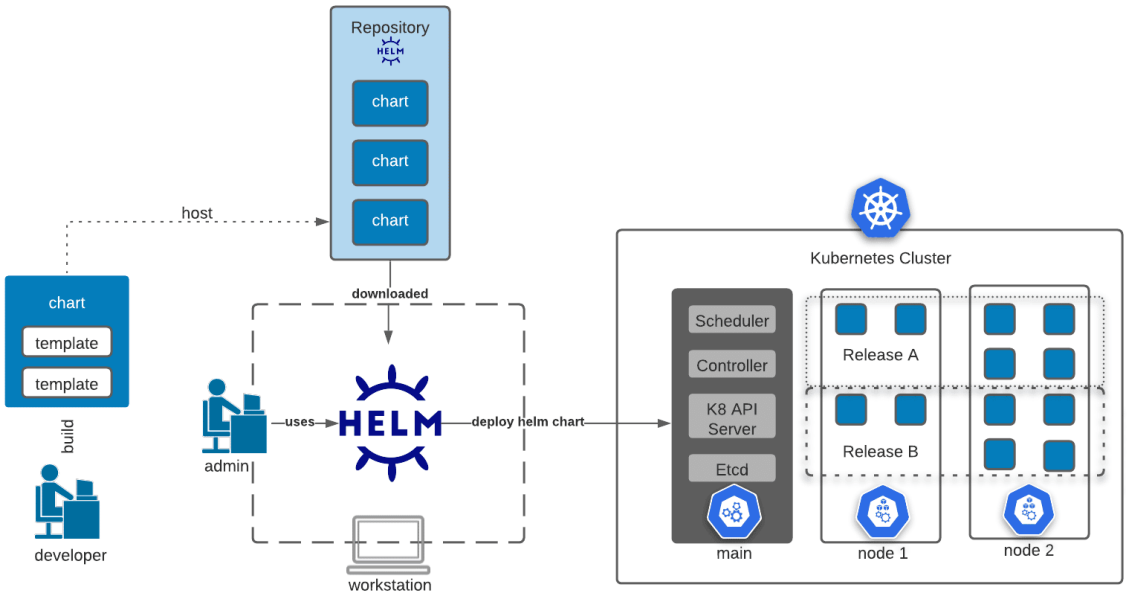
3.5 Best Practices (Summary)

Best Practice	**Benefit**
Use _helpers.tpl	Reusability
Use values.yaml extensively	Flexibility
Keep templates minimal	Maintainability
Provide defaults but allow overrides	Better UX
Template only what is needed	Avoid complexity
Use schema validation (values.schema.json)	Avoid invalid values

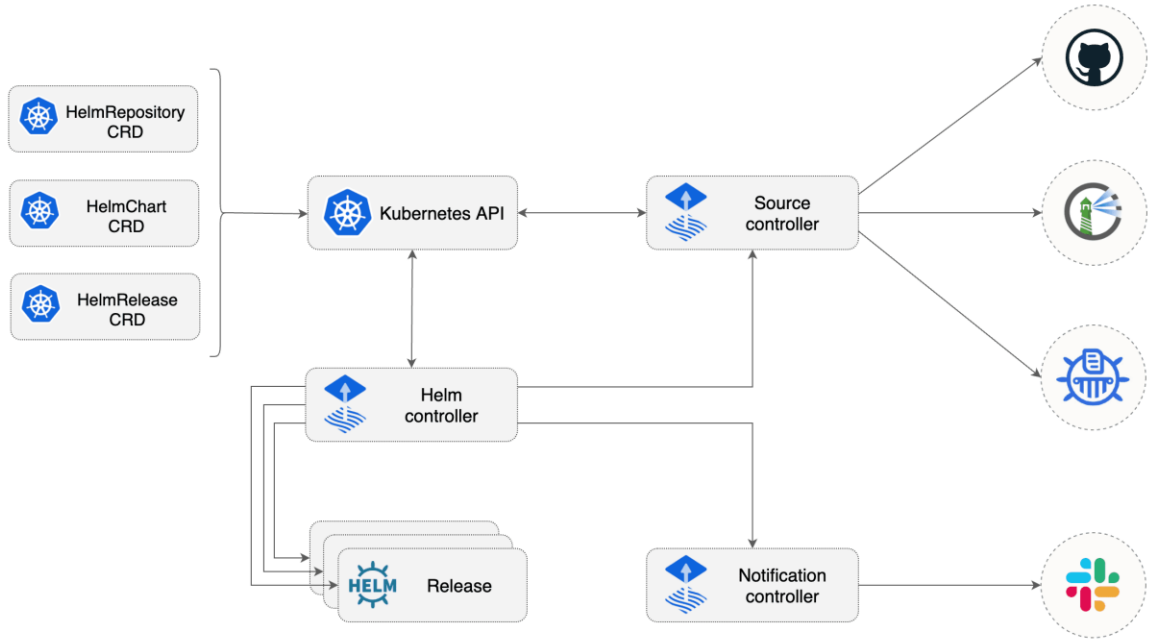
---

4. Helm with GitOps Workflows





Helm Workflow



Helm integrates seamlessly with **GitOps tools** like:

- **ArgoCD**
- **FluxCD**
- Jenkins-X
- GitHub Actions → Kubernetes

GitOps ensures:

- Declarative deployments
- Fully version-controlled releases
- Automated synchronization

4.1 GitOps with ArgoCD + Helm

ArgoCD supports three patterns:

1. **Helm chart stored inside Git**
2. **Helm values.yaml overridden by Git**
3. **Pulling charts from external Helm repositories**

Example ArgoCD App:

apiVersion: argoproj.io/v1alpha1

kind: Application

```
metadata:
  name: myapp
spec:
  project: default
  source:
    repoURL: https://github.com/vivek-org/helm-charts
    targetRevision: main
    path: myapp
    helm:
      values: |
        replicaCount: 3
        image:
          tag: "2.0.0"
  destination:
    server: https://kubernetes.default.svc
    namespace: prod
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

4.2 GitOps with Flux + Helm

FluxCD has a dedicated component: **Helm Controller**

You define:

HelmRepository

Points to chart source.

```
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: HelmRepository
metadata:
  name: bitnami
spec:
```

```
  url: https://charts.bitnami.com/bitnami
```

HelmRelease

Defines the release of a chart.

```
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: my-nginx
spec:
  chart:
    spec:
      chart: nginx
      version: "13.x"
      sourceRef:
        kind: HelmRepository
        name: bitnami
  values:
    replicaCount: 4
```

4.3 GitOps Best Practices

Practice	**Why Important**
Store all Helm values in Git	Full audit trail
Enable auto-sync (ArgoCD/Flux)	Immediate recovery
Use chart versioning	Safe upgrades/rollbacks
Split prod & dev values files	Environment separation
Use pipelines to test charts	Quality & security

Final Summary — Helm Module

Topic	Summary
Helm 3 Intro	Package manager for Kubernetes; no Tiller; secure
Creating Charts	helm create, templates, Chart.yaml, values.yaml
Templating Best Practices	Helpers, loops, conditionals, schema, defaults
Helm + GitOps	Automations with ArgoCD, FluxCD, version-controlled infra