Deployment in Kubernetes - Kustomize

starts at 9:05 pm

Agenda

1. Kustomize
    1. Understanding Kustomize
    2. Base and Overlay
2. Demo (Base and Overlay)
3. Advanced Kustomize Features
4. Demo (Advanced Kustomize Features)
5. Helm vs Kustomize

→ Kustomize

helm
- → templating
- → packaging
- → version control.

2018
↓ merged into kubectl in
2019 (March) Kubernetes v1.14

Pizza Restaurant (non veg, veg pizza)

→ Base Recipe

→ Customers customize pizza with unique toppings.

Kustomize.

① → Base Configuration.

→ default app. (never changes)

② → Customizations. (overlays)

dev /	(extra env variable)
prod	(more replicas)
qa	(different container image)

$\underbrace{\hspace{6em}}$

overlays.

→ Definition

Kustomize is a Kubernetes-native tool used to **customize Kubernetes manifests declaratively**.

Kustomize allows you to **overlay configurations** for different environments (dev, staging, production) without altering the original files.

kubectl apply -k

- It follows a **layered approach** where you define **a base configuration** and apply **environment-specific overlays** without duplication.

```
kustomize-demo/
 ├─── base/
 |    ├─── deployment.yaml
 |    ├─── service.yaml
 |    ├─── configmap.yaml
 |    ├─── kustomization.yaml
 ├─── overlays/
 |    ├─── dev/
 |    |    ├─── kustomization.yaml (modifies the base)
 |    |    ├───dev-configmap.yaml
```

Base

① Reusable

② unmodified

③ Independent.

Overlay

① Extends the Base.

```
|   ├── prod/
|   |   ├── kustomization.yaml
|   |   ├──prod-configmap.yaml
```

② Enviroment speific
③ Non destructive.

Demo → deploy nginx using Kustomize.

3 clusters
① Base
② Dev
③ Prod

```
kustomize-demo/
├── base/
|   ├── deployment.yaml
|   ├── service.yaml
|   ├── configmap.yaml
|   ├── kustomization.yaml
├── overlays/
|   ├── dev/
|   |   ├── kustomization.yaml (modifies the base)
|   |   ├──dev-configmap.yaml
|   ├── prod/
|   |   ├── kustomization.yaml

|   |   ├──prod-configmap.yaml
```

Kustomization.yaml

→ declare resources
→ apply patches
→ generating CM, Secrets
→ transform configuration

→ Create 3 kind clusters
→ Base
→ dev
→ prod

① Create Base directory

```
mkdir -p kustomize-demo/base

cd kustomize-demo/base
```

② Create service, deployment, configmap file

deployment.yaml

```
apiVersion: apps/v1

kind: Deployment
```

```
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx/conf.d
      volumes:
      - name: nginx-config
```

```
        configMap:
          name: nginx-config
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  default.conf: |
    server {
      listen 80;
      location / {
        return 200 "Welcome to Base Cluster!";
      }
    }
```

③ Create kustomization.yaml file

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - deployment.yaml
  - service.yaml
  - configmap.yaml
```

④ Create overlays for Dev and Prod

```
mkdir -p ../overlays/dev ../overlays/prod
```

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - ../../base
patchesStrategicMerge:
  - dev-configmap.yaml
commonLabels:
  environment: dev
```

dev kustomization.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  default.conf: |
    server {
      listen 80;
      location / {
        return 200 "Welcome to Dev Cluster!";
      }
    }
```

dev-configmap.yaml

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - ../../base
patchesStrategicMerge:
  - prod-configmap.yaml
commonLabels:
  environment: prod
```

prod kustomization.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  default.conf: |
    server {
      listen 80;
      location / {
        return 200 "Welcome to Prod Cluster!";
      }
    }
```

prod-configmap.yaml

(5) Apply Base Configuration to kind-base

```
kubectl config use-context kind-kind-base

kubectl apply -k kustomize-demo/base
```

Apply Dev Overlay to kind-dev

```
kubectl config use-context kind-kind-dev

kubectl apply -k kustomize-demo/overlays/dev
```

Dev - overlay

Apply Prod Overlay to kind-prod

```
kubectl config use-context kind-kind-prod

kubectl apply -k kustomize-demo/overlays/prod
```

Prod - overlay

Break 10:15 pm

→ Advanced Kustomize Feature

1. Vars

2. Patches

3. Cross Cutting Fields

4. Changing images

5. Generator Options.

① Vars

① definition (kustomization.yaml)

② substitution.

SERVICE_NAME

↓

my-service

deployment.yaml

↓

environment variable

SERVICE_URL = $(SERVICE_NAME)

SERVICE_URL = my-service

② PATCHES

In **Kustomize**, patches are used to **modify existing Kubernetes resources** (like Deployments, Services, ConfigMaps) without changing the original `base` manifests.

① Patches Strategic Merge.

- **Modifies fields in existing resources.**
- Used for **small, selective updates** to a resource.
- **Does not require full resource definition**—only specify fields to update.

→ deployment.yaml → <u>Base</u>:

→ patch-deployment.yaml. → (overlay)

apiVersion: apps/v1

kind: Deployment

```yaml
metadata:
  name: my-app
spec:
  replicas: 4  # Overriding the replica count from 2 to 4
```

→ kustomization.yaml

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - ../../base
patchesStrategicMerge:
  - patch-deployment.yaml
```

② → JSON Patch

Json 6902

→ add
→ remove
→ replace.

} Json files.

```yaml
- op: replace
  path: "/spec/template/spec/containers/0/image"
  value: "nginx:1.21"
```

} → overlays/prod/patch-image.json

```yaml
- op: add
  path: "/spec/template/spec/containers/0/env"
  value:
    - name: ENV
      value: "production"
```

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - ../../base
patchesJson6902:
  - target:
      group: apps
      version: v1
      kind: Deployment
      name: my-app
    path: patch-image.json
```

Kustomization.yaml.

$\boxed{\text{overlay / prod}}$

Kust.yaml   patch-img.

source

→ deployment
→ patch-json.

$\boxed{\text{base}}$

$\boxed{\text{deployment}}$

updated deployment
↓
applied

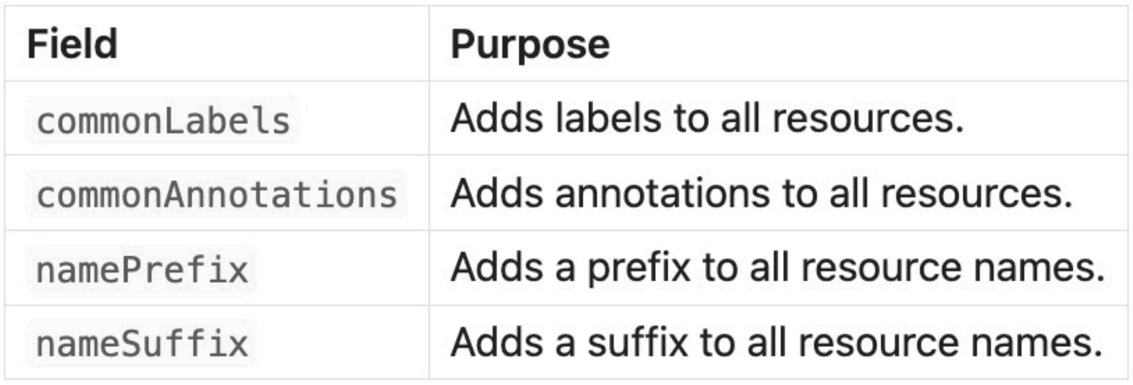
③ Patches.

Kustomize
v4.1.0

```
apiVersion: kustomize.config.k8s.io/v1beta1

kind: Kustomization

resources:

  - ../../base

patches:

  - target:

      kind: Deployment

      name: my-app
```

```yaml
patch: |  # Inline YAML Patch
  - op: replace
    path: "/spec/replicas"
    value: 4   # Changes replica count from 2 to 4

  - op: add
    path: "/spec/template/spec/containers/0/env"
    value:
      - name: ENV
        value: "production"   # Adds new environment variable

  - op: remove
    path: "/spec/template/spec/containers/0/ports"  # Removes ports
```

③ Cross Cutting Fields



Field	Purpose
`commonLabels`	Adds labels to all resources.
`commonAnnotations`	Adds annotations to all resources.
`namePrefix`	Adds a prefix to all resource names.
`nameSuffix`	Adds a suffix to all resource names.

④ Changing Images.

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - deployment.yaml
  - service.yaml
  - configmap.yaml

images:  # Image override (for dynamic image management)
  - name: nginx
    newTag: "1.23"
```

Kustomization.
yaml

nginx : 1.23

⑤ generator options.

→ CM, Secrets.

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

configMapGenerator:
```

Secret generator.

```
  - name: my-config
    literals:
      - key1=value1
      - key2=value2



apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config-7gfh4c7hkd  # <-- Hash suffix added



piVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
  - ../../base

configMapGenerator:
  - name: my-config
    literals:
      - key1=value1
    - key2=value2
  options:
    disableNameSuffixHash: true  # Prevents hash in the name
    labels:
      app: my-app
    annotations:
      owner: "dev-team@example.com"


apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config    # No hash suffix because of `disableNameSuffixHash`
  labels:
    app: my-app
  annotations:
    owner: "dev-team@example.com"
data:
  key1: value1
  key2: value2
```

→ Demo Advanced Kustomize Features

→ Create kind cluster

```
advance-kustomize-demo/
├── base/
|   ├── kustomization.yaml
|   ├── deployment.yaml
|   ├── service.yaml
|   ├── index.html
├── overlays/
|   ├── dev/
|   |   ├── kustomization.yaml
|   |   ├── patch-deployment.yaml
|   |   ├── index.html
```

→ Real World Scenario

→ Multi - Region Deployment

→ each region → own clusters

```
kustomize/
    ├── base/
    |    ├── deployment.yaml
    |    ├── kustomization.yaml
    |    └── service.yaml
    ├── regions/
    |    ├── us-east/
    |    |    ├── kustomization.yaml
    |    |    └── patch-us-east.yaml
    |    ├── eu-west/
    |    |    ├── kustomization.yaml
    |    |    └── patch-eu-west.yaml
    |    └── ap-south/
    |         ├── kustomization.yaml
    |         └── patch-ap-south.yaml
    └── global/
         ├── kustomization.yaml
         └── common.yaml
```

resource
→ base
– global.

} cluster 1

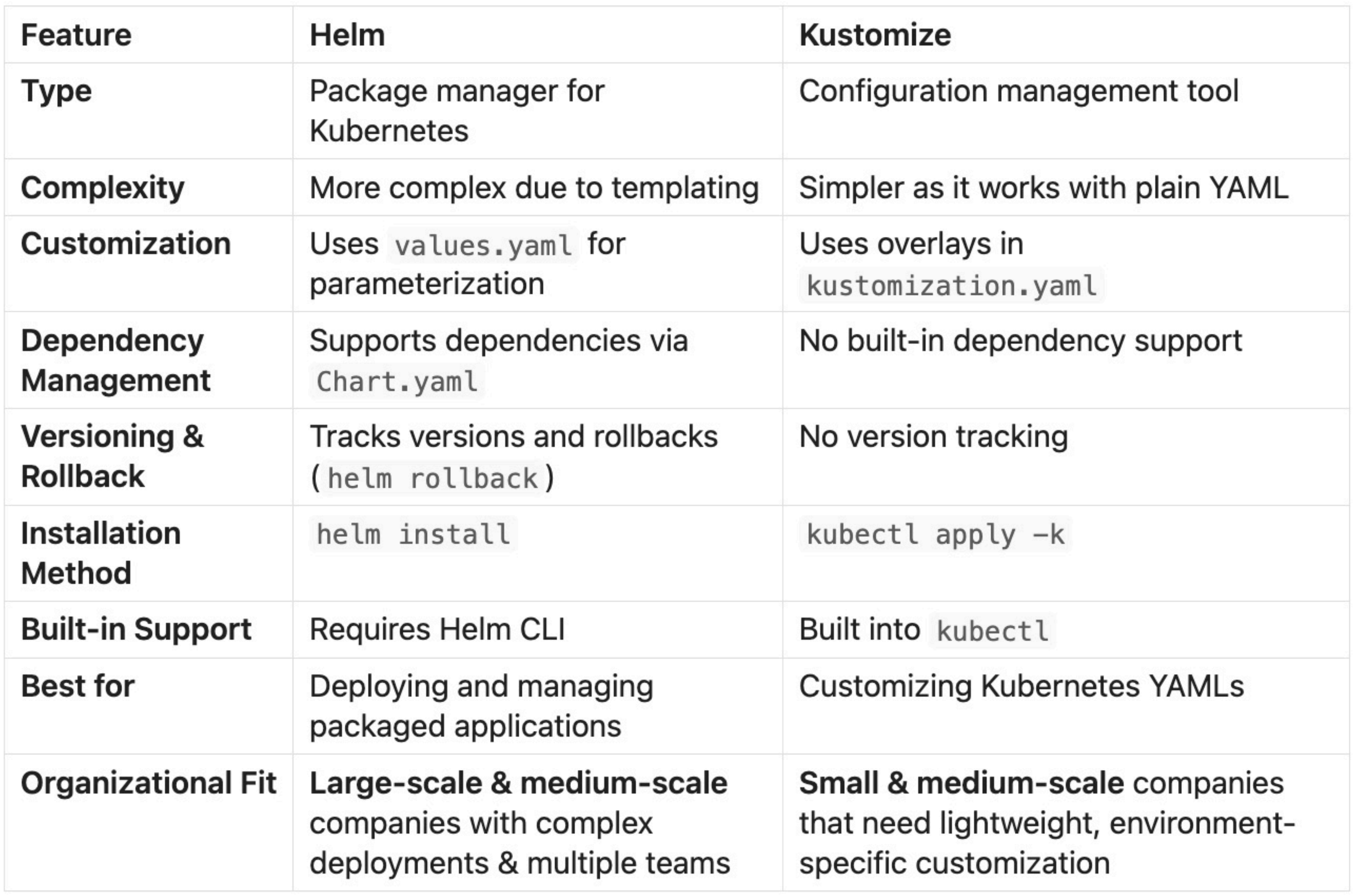
} cluster 2

} cluster 3

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
```

```yaml
resources:
  - ../../base
  - ../../global
patchesStrategicMerge:
  - patch-us-east.yaml
```

→ Helm vs Kustomize



Feature	**Helm**	**Kustomize**
Type	Package manager for Kubernetes	Configuration management tool
Complexity	More complex due to templating	Simpler as it works with plain YAML
Customization	Uses `values.yaml` for parameterization	Uses overlays in `kustomization.yaml`
Dependency Management	Supports dependencies via `Chart.yaml`	No built-in dependency support
Versioning & Rollback	Tracks versions and rollbacks (`helm rollback`)	No version tracking
Installation Method	`helm install`	`kubectl apply -k`
Built-in Support	Requires Helm CLI	Built into `kubectl`
Best for	Deploying and managing packaged applications	Customizing Kubernetes YAMLs
Organizational Fit	**Large-scale & medium-scale** companies with complex deployments & multiple teams	**Small & medium-scale** companies that need lightweight, environment-specific customization

→ Hybrid