Kustomize Template-free manifests

START HERE

`kustomize create --autodetect`
Generate kustomization.yaml

`kustomize build <dir>`
Render the manifests

`kubectl apply -k <dir>`
Build and apply

`kubectl diff -k <dir>`
Preview the change

`kustomize edit add resource <file>`
Add a resource

`kustomize edit set image <name>=<image>:<tag>`
Pin an image

`kustomize edit set namespace <ns>`
Set the namespace

`kustomize build <dir> | kubectl apply -f -`
Pipe a build

Base plain manifests → `resources` → **Overlay** patches, images → `build` → **Rendered YAML** stdout → `apply -k` → **Cluster**

CONCEPTS

Kustomize never edits your YAML: it reads a base, applies declared transformations and prints the result.

Layout convention: base/ holds shared manifests, overlays/<env>/ each hold a kustomization.yaml that lists ../../base under resources.

kubectl embeds a kustomize that is often older than the standalone binary, so output can differ between kubectl -k and kustomize build.

BUILD AND APPLY

`kustomize build <dir>`
Render a kustomization to stdout

`kustomize build <dir> -o <path>`
Write output to a file or directory

`kubectl apply -k <dir>`
Build and apply to the cluster

`kubectl kustomize <dir>`
Build using kubectl's bundled kustomize

`kubectl diff -k <dir>`
Show what apply would change

`kubectl delete -k <dir>`
Delete everything the build produces

`kubectl get -k <dir>`
List live objects of the build

`kustomize build --enable-helm <dir>`
Inflate helmCharts entries
Needs the helm binary on PATH.

`kustomize build --load-restrictor LoadRestrictionsNone <dir>`
Allow files outside the root

`kustomize build <git-url>//<path>?ref=<tag>`
Build straight from a git repo

`kustomize version`
Print the version

CREATE AND EDIT

`kustomize create`
Write an empty kustomization.yaml

`kustomize create --autodetect`
Create it listing YAML files found

CREATE AND EDIT (cont.)

`kustomize edit add resource <path>`
Add a file or directory to resources

`kustomize edit add patch --path <file> --kind <kind>`
Add a patch entry

`kustomize edit add configmap <name> --from-literal=<k>=<v>`
Add a generated ConfigMap

`kustomize edit add label <k>:<v>`
Add a label to all resources

`kustomize edit set image <name>=<newname>:<tag>`
Change an image name and tag

`kustomize edit set namespace <ns>`
Set the namespace

`kustomize edit set nameprefix <prefix>`
Set the name prefix

`kustomize edit set replicas <name>=<n>`
Set a replica count

`kustomize edit fix`
Migrate deprecated fields
Rewrites patchesStrategicMerge, commonLabels and bases to current fields.

KUSTOMIZATION.YAML

Skeleton

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

`resources:`
Files, directories or URLs to include

`namespace: <ns>`
Set namespace on every resource

`namePrefix: <p>` / `nameSuffix: <s>`
Prefix or suffix all names
References between resources are updated too.

`labels:`
Add labels, optionally to selectors
Prefer over commonLabels: selectors are only touched with includeSelectors: true.

`commonAnnotations:`
Add annotations to all resources

KUSTOMIZATION.YAML (cont.)

`replicas:`
Override replica counts

`components:`
Include reusable Component directories

PATCHES

Strategic merge from a file

```
patches:
- path: patch.yaml
  target:
    kind: Deployment
    name: web
```

Inline JSON6902

```
patches:
- target: {kind: Deployment,
name: web}
  patch: |-
    - op: replace
      path: /spec/replicas
      value: 3
```

`target: {kind, name, labelSelector}`
Select what a patch applies to
name and namespace are regexes, anchored automatically.

`op: add / replace / remove`
JSON6902 operations

`path: /metadata/labels/a~1b`
Escape / as ~1 in JSON paths

Without a target, a strategic merge patch matches by the apiVersion, kind and name written in the patch itself. JSON6902 always needs a target.

IMAGES AND GENERATORS

Images

```
images:
- name: nginx
  newName: registry.example/
nginx
  newTag: "1.27"
```

`digest: sha256:<hash>`
Pin an image by digest instead of tag

ConfigMap from literals and files

```
configMapGenerator:
- name: app-config
  literals:
  - LOG_LEVEL=debug
  files:
  - app.properties
```

`envs: [<file>.env]`
Load KEY=value lines as entries

`secretGenerator:`
Same fields as configMapGenerator, makes Secrets

IMAGES AND GENERATORS (cont.)

`behavior: create / replace / merge`
How an overlay treats a base's generator

`options: {disableNameSuffixHash: true}`
Keep the plain name without a hash
The hash suffix triggers rollouts when config changes; disabling it loses that.

REPLACEMENTS AND COMPONENTS

Copy a value into other fields

```
replacements:
- source:
    kind: ConfigMap
    name: app-config
    fieldPath: data.HOST
  targets:
  - select:
      kind: Deployment
    fieldPaths:
    - spec.template.spec.
containers.[name=web].env.
[name=HOST].value
```

A component

```
apiVersion: kustomize.config.
k8s.io/v1alpha1
kind: Component
patches:
- path: add-sidecar.yaml
```

RECIPES

Base plus overlay layout

```
base/
  kustomization.yaml   #
resources: [deployment.yaml,
service.yaml]
overlays/prod/
  kustomization.yaml   #
resources: [../../base]
kubectl apply -k overlays/prod
```

Prod overlay

```
apiVersion:
kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: prod
resources:
- ../../base
replicas:
- name: web
  count: 5
images:
- name: web
  newTag: v2.3.1
```

Preview before applying

```
kustomize build overlays/prod
> /tmp/prod.yaml
kubectl diff -k overlays/prod
kubectl apply -k overlays/prod
```

Bump an image in CI

```
cd overlays/prod
kustomize edit set image
web=registry.example/web:$TAG
kustomize build . | kubectl
apply -f -
```