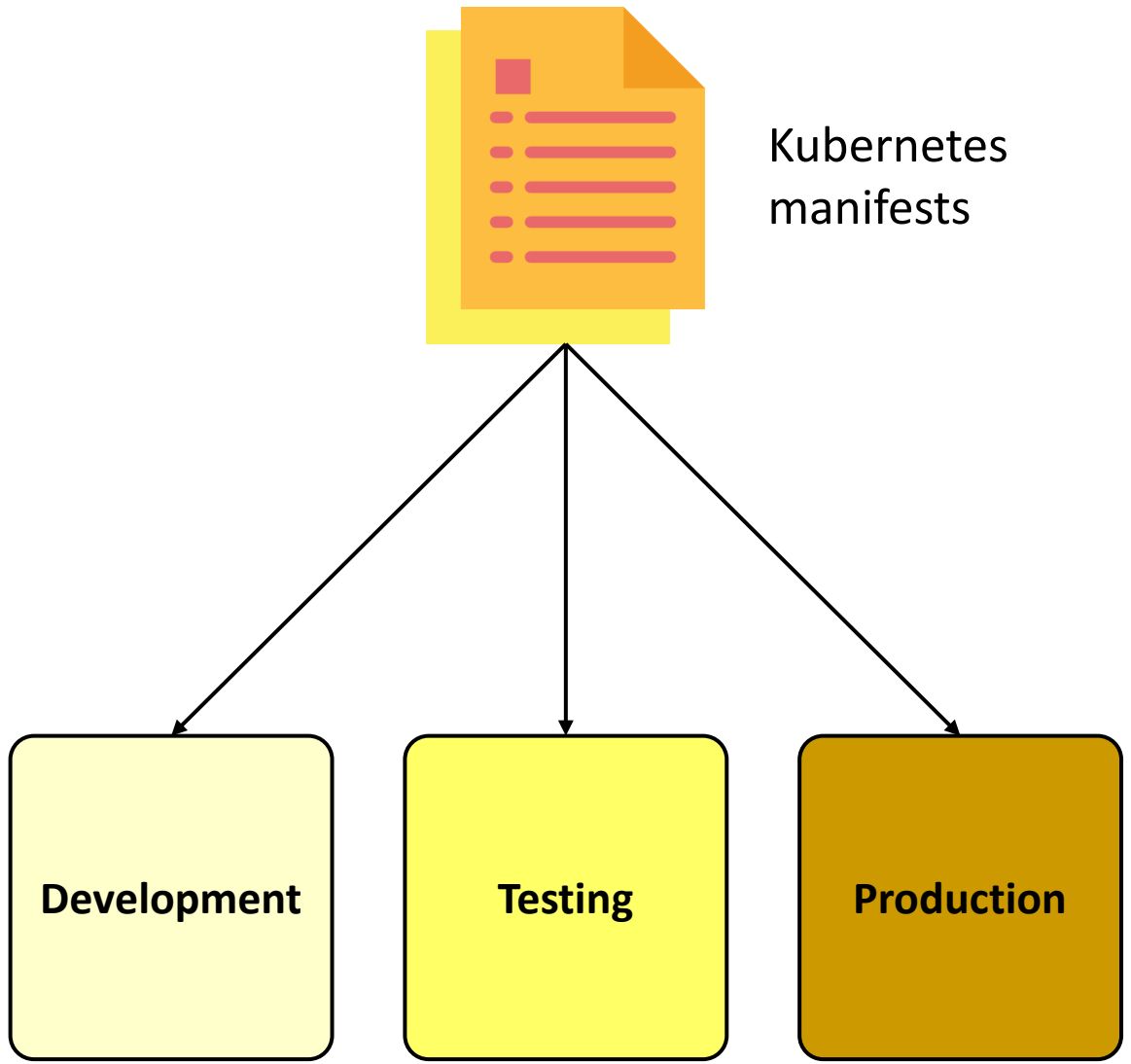

# Tools for Working with Kubernetes


# Kustomize

# Multiple Environments

- Different environments will require different tweaks to the manifests
  - Different configurations
    - Some can be configured with `ConfigMaps` or `Secrets`
    - Some configurations cannot eg. replicas, image name
  - Different attributes
    - Require resource limits in production
    - Additional resources like cache in production



# Using Kubernetes In Multiple Environments

- DRY – Don't Repeat Yourself
- Develop a Helm chart
- Nontrivial
  - Need to develop templates that cover all possible scenarios
  - Learn Go templates, template functions, conventions, etc
  - Debugging!
- Significant time investment
- Helm provides
  - Structured way of customizing deployments
  - Deployment management - install, delete , upgrades

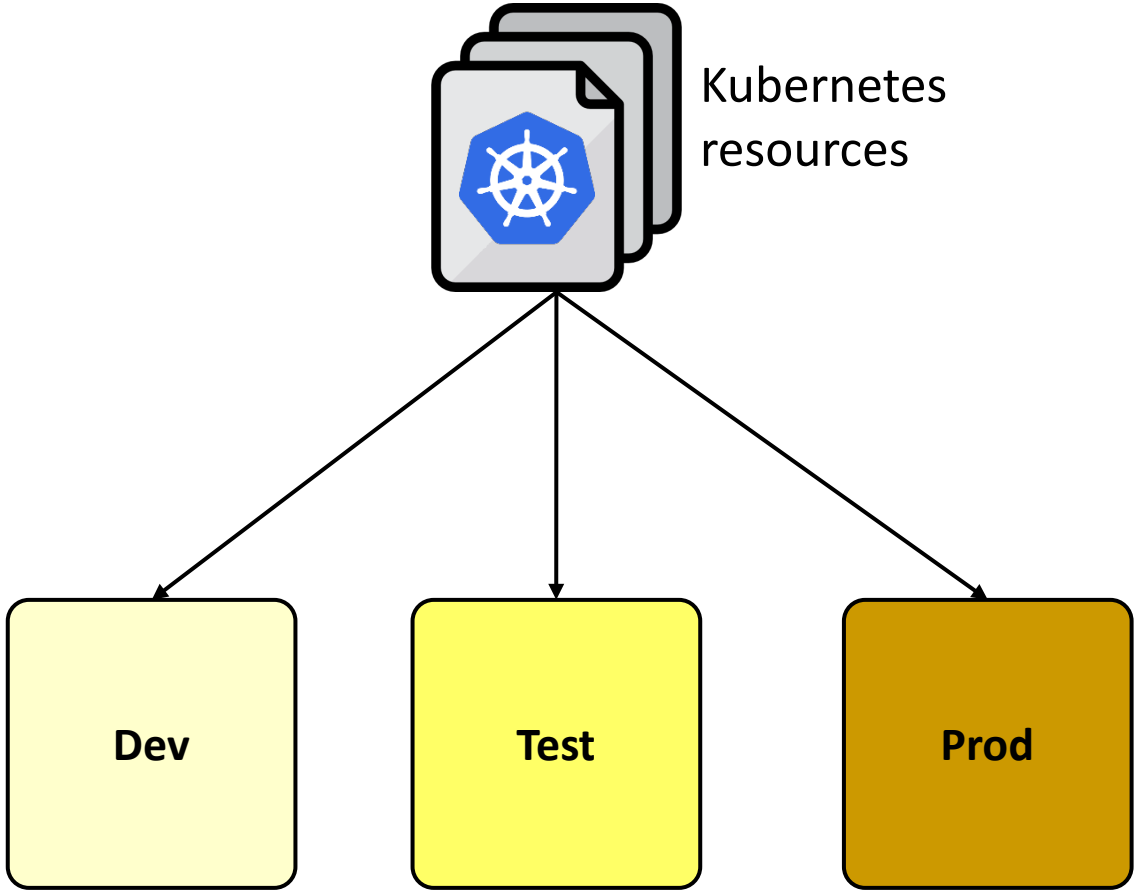
- WET – Write Everything Twice
- Duplicate and modify the original Kubernetes manifest
- Need to actively update downstream manifest when upstream changes
  - Eg. bug fixes, improvement, etc.
- Copy and paste nature is
  - Easy to use
  - No new concepts to learn
  - Fast


# Kustomize

- Kustomize is a tool for customizing Kubernetes deployments
  - Apply a set of changes to a base Kubernetes manifest
  - Produces a new set of manifest
  - Original manifest (base) is untouched/unchanged
  - Template free viz. logic less
- Kustomize allow you to consume and customize upstream Kubernetes manifest without forking them
- Open source
  - Available as standalone - https://kubectl.docs.kubernetes.io/installation/kustomize/
  - Integrated with kubectl (post 1.14) with '`apply -k`'

# Directory Layout

- Create a structure to support customizing a set of Kubernetes manifest
  - Support multiple environments (eg dev, test, prod) without interfering with another environment
  - Any changes will be picked up by all upstreams viz. dev, test, prod, etc
- Base manifest
  - Generic Kubernetes manifest applicable to all environments
- Overlays
  - Kustomize configurations to be applied to the base manifest
  - One per environment



# Directory Structure

**base/kustomization.yaml**

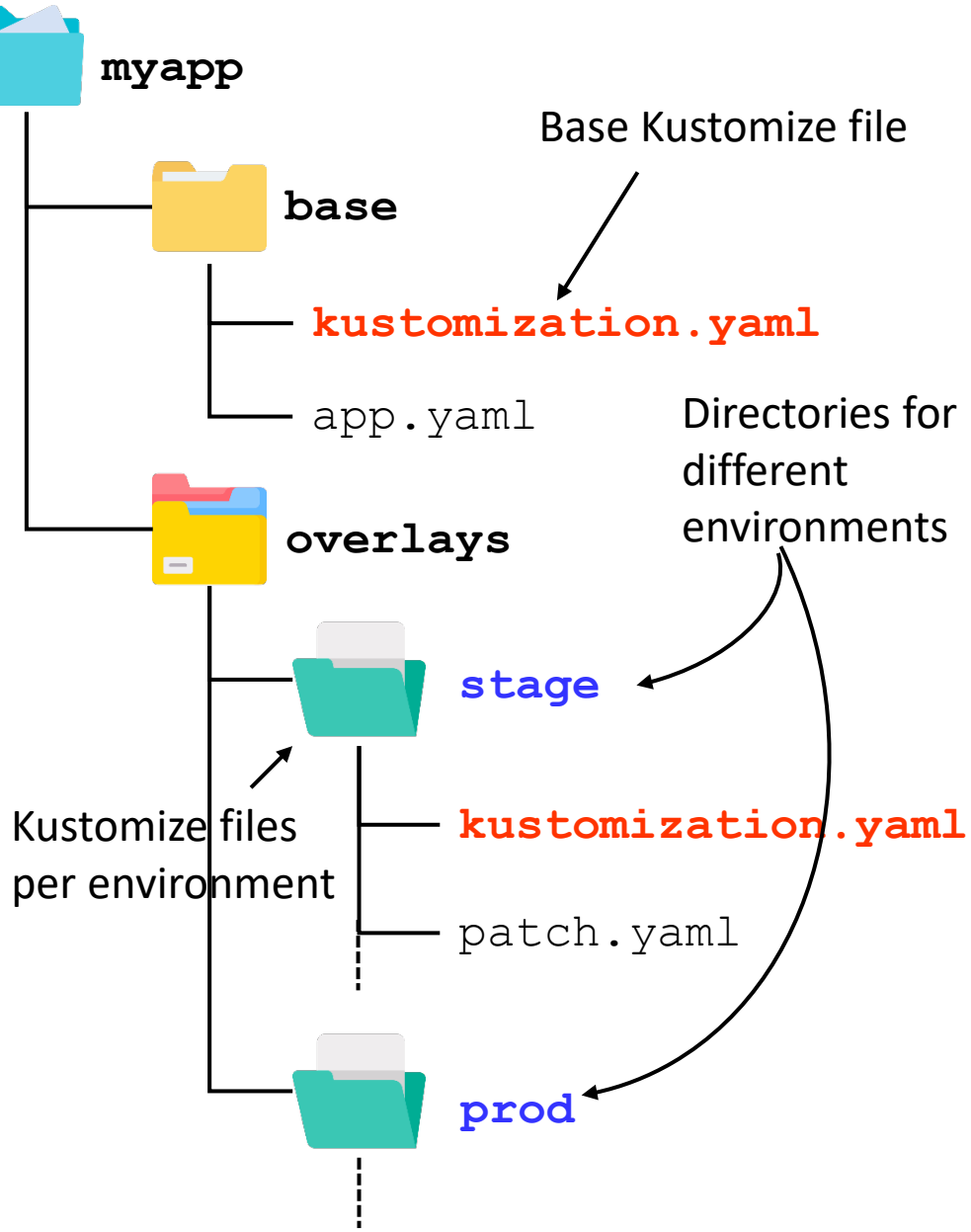
```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- app.yaml
```

---

**overlays/stage/kustomization.yaml**

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

# site specific customization
namePrefix: ...
```




# Customizing Kubernetes Deployments

- Kustomize transform a set of related Kubernetes manifest by adding or replacing fields and objects
  - Can delete fields but not common
- Modifications are updates can be applied to
  - All resources - eg adding a common label
  - Specific resource by providing patterns that match the resource
- Input Kubernetes manifest are specified with the `resources` field

# Example - Common Fields

**kustomization.yaml**

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- pod.yaml
labels:
  - pairs:
      env: stage
    includeSelectors: true
    includeTemplates: true
namePrefix: eng-
namespace: stage-ns
```

**pod.yaml**

```
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
  labels:
    name: app
spec:
  containers:
  - name: app-container
    image: nwapp:v1
```

```
apiVersion: v1
kind: Pod
metadata:
  name: eng-app-pod
  labels:
    name: app
    env: eng
  namespace: stage-ns
spec:
  containers:
  - name: app-container
    image: nwapp:v1
```

With kustomize

```
kustomize build | kubectl apply -f -
```

With kubectl

```
kubectl apply -k .
```

Directory with kustomization.yaml

# Example - Deployments

```
kustomization.yaml
apiVersion: ...
kind: Kustomization

replicas:
- name: web-deploy
  count: 1

images:
- name: resilio/sync
  newName: eeacms/rsync
  newTag:2.3
- name: nginx
  newTag: 1.21.1-perl
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deploy
spec:
  replicas: 2
  selector:
    ...
  template:
    ...
    spec:
      containers:
       - name: file-sync
         image: resilio/sync:2.7.2
       ...
       - name: nginx
         image: nginx:1.20.1
       ...
```

1

eeacms/rsync:2.3

nginx:1.21.1-perl

# `ConfigMap` and `Secrets`

- Can customize an existing configuration or secret specific to an environment
  - Modify an existing configuration by merging with new keys or replacing existing keys
  - Generate new configurations and secrets
- **`ConfigMap`** and **`Secrets`** behaviour
  - Pods that references values in these 2 resources are not recreated if the content changes eg. value of a secret key is replaced
    - Referenced with either `envFrom` or `valueFrom`
  - Pods that mount these 2 resources will be recreated

# Example - Override Existing Configurations

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

configMapGenerator:
- name: app-cm
  namespace: app-ns
  behavior: merge
  literals:
  - DB_USER=barney
  - TRACE_ENABLE="1"

secretGenerator:
- name: app-secret
  namespace: app-ns
  behavior: merge
  type: Opaque
  literals:
  - DB_PASSWORD=mypassword
```

Add or update the values in the base

Replaced and added to the base

Replace the base

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-cm
  namespace: app-ns
data:
  DB_HOST: localhost
  DB_PORT: 3306
  DB_USER: fred
  DB_NAME: inventory
```

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
  namespace: app-ns
type: Opaque
data:
  DB_PASSWORD: bXlzZWNyZXQ=
```

# Example - Generating `ConfigMap` and `Secret`

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

resources:
- app.yaml

generatorOptions:
  labels:
    env: stage

configMapGenerator:
- name: app-cm
  namespace: app-ns
  behavior: create
  literals:
  - DB_HOST=dbserver
    DB_USER=barney
```

Add additional labels and annotations to all the generated `ConfigMap` and `Secret`

Create a `ConfigMap` call `app-cm`

One or me configurations

**app.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  containers:
  - name: app-container
    image: nwapp:v1
    envFrom:
     configMapRef
       name: app-cm
```

Reference a non existence `ConfigMap`

Similar for `secretGenerator`

# Example - Generating `ConfigMap` and `Secret`

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-cm-bfA9Ekh0c5
  labels:
    env: stage
data:
  DB_HOST: dbserver
  DB_USER: barney
```

Generated `ConfigMap` with a unique suffix.
A new suffix is generated everytime kustomize runs
Uses this `ConfigMap` when it is referenced from containers
Suffix hash forces Kubernetes to recreate the pods when configurations and secrets are updated

**app.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  containers:
  - name: app-container
    image: nwapp:v1
    envFrom:
    - app-cm-bfA9Ekh0c5
```

# Patching

- Overrides or add fields to the target resource
- A list of patch files are provided
  - Order of patches applied is the order in which they are listed
- Patch file is similar to the resource to be patched
  - Should have `apiVersion`, `kind` and `name`; used by Kustomize to select the resource to be patched
  - The remainder of the patch file used to overlay on the resource viz. metadata and spec sections
  - Attributes with the same name are replaced
  - Additional attributes are added
- Patches are described with JSON patch

# JSON Patch Example

Original JSON document

```
{
  "baz": "qux",
  "foo": "bar"
}
```

JSON patch operations

```
[
  { "op": "replace", "path": "/baz", "value": "boo" },
  { "op": "add", "path": "/hello", "value": [ "world" ] },
  { "op": "remove", "path": "/foo" }
]
```

Patched JSON document

```
{
  "baz": "boo",
  "hello": [ "world" ]
}
```

Example from https://jsonpatch.com/

# JSON Patch Example

Original JSON document

```
{
  "biscuits": [
    { "name": "Digestive" },
    { "name": "Jacbos" }
  ]
}
```

JSON patch operations

```
[
  { "op": "add", "path": "/biscuits/-",
      "value": { "name": "Khong Guan" } },
  { "op": "remove", "path": "/biscuits/1" }
]
```

Patched JSON document

```
{
  "biscuits": [
    { "name": "Digestive" },
    { "name": "Khong Guan" }
  ]
}
```

Example from https://jsonpatch.com/

# Example - Patching

**app.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 2
  selector:
    ...
  template:
    ...
    spec:
      containers:
      - name: nwapp
        image: nwapp:v2
        ports:
        - containerPort: 3000
```

**app.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 2
  selector:
    ...
  template:
    ...
    spec:
      containers:
      - name: nwapp
        image: nwapp:v2
        ports:
        - containerPort: 3000
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
```

Customized resource

Add these

# Example - Patching

**kustomization.yaml**

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

resources:
- app.yaml

patches:
- target:
    group: apps
    version: v1
    kind: Deployment
    name: myapp
  path: patch.yaml
```

**patch.yaml**

```
- op: "add"
  path: /spec/template/spec/containers/0/resources
  value: |
   { "requests": {
        "cpu": "100m",
        "memory": "128Mi" }
   }
```


# Replacements

- Need to know certain info from resources generated by Kustomize
  - Eg. generated a service for a database with a prefix. Need to pass this service name to a deployment that uses the database
- Used to copy fields from one source to one or more targets
  - Eg copying the service name to environment variables
- Source is the resource that will provide the value; must be a single resource
- Target(s) are the resources that the fields will be replaced
  - Include and reject sets
- Use `fieldPath` to select specific fields as source and targets
  - See https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/replacements/#delimiter

# Example - Replacement

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

replacements:
- source:
    kind: Service
    name: app-svc
    fieldPath: metadata.name
  targets:
  - select:
      kind: Ingress
      name: app-ing
    fieldPaths:
    - spec.rules.[host=acme.com].http.paths.[path=/].backend.service.name
```

Get the service name

Selects this resource

```yaml
apiVersion: apps/v1
kind: Ingress
metadata:
  name: app-ing
spec:
  rules:
  - host: acme.com
    http:
      paths:
      - path: /
        backend:
          service:
            name: REPLACE
```

Update the service name in the rules