# Kubernetes Security & Continuous Delivery

ytshih (2026, CC BY-SA)

# Outline

- Security
  - Control Plane Protection
  - Workload Protection
  - Policies
- Continuous Delivery
  - Helm
  - Kustomize
  - FluxCD

# Workload Scenario

- An Nginx server that serves user-uploaded websites
  - E.g. https://people.cs.nycu.edu.tw/~tsaimh/
- There are two instances of the app, one for testing, one for production
- There are three types of user: reporter, developer, and maintainer
  - Maintainer > Developer > Reporter
  - Reporter can view the resources in the app
  - Developer can edit resources in testing instance
  - Maintainer can edit all resources of the app
- Examples in this lecture will follow this scenario

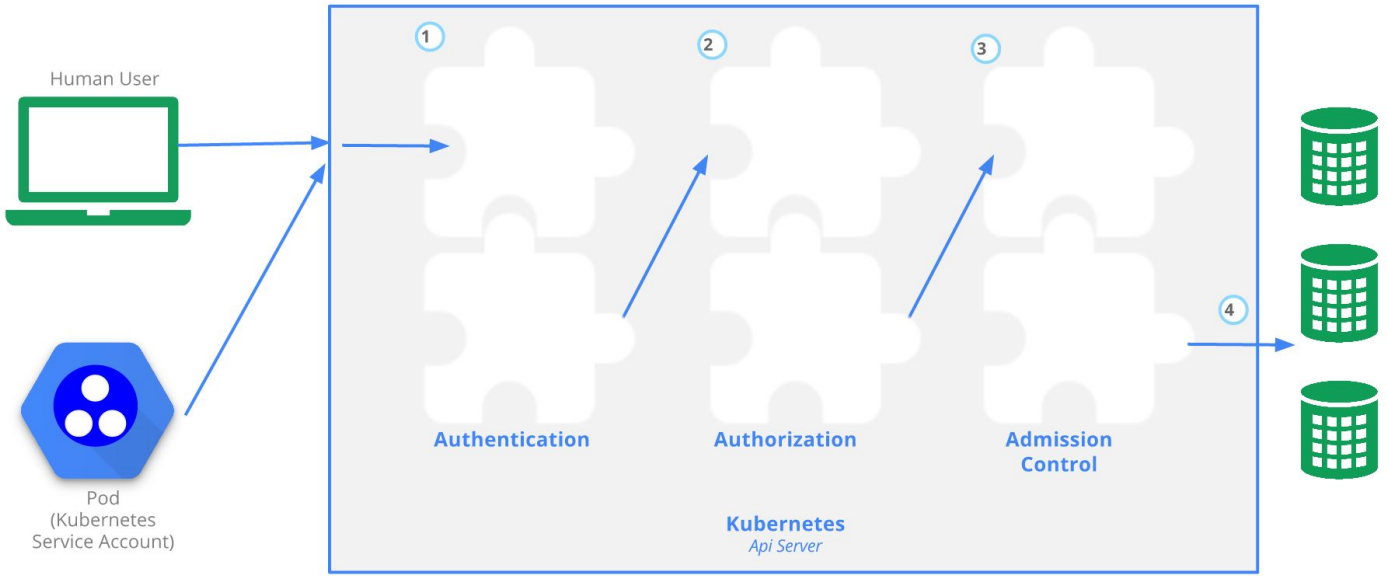
# Security

# Security

- Control Plane Protection
  - Authentication, Authorization, Admission Control
- Workload Protection
  - Pod Security Standards
  - Pod Security Admission
  - Container Runtimes / Runtime Class
- Policies
  - Limit Ranges / Resources Quotas
  - Network Policies
  - Admission Policies

# Control Plane Protection

# Access to Kubernetes API



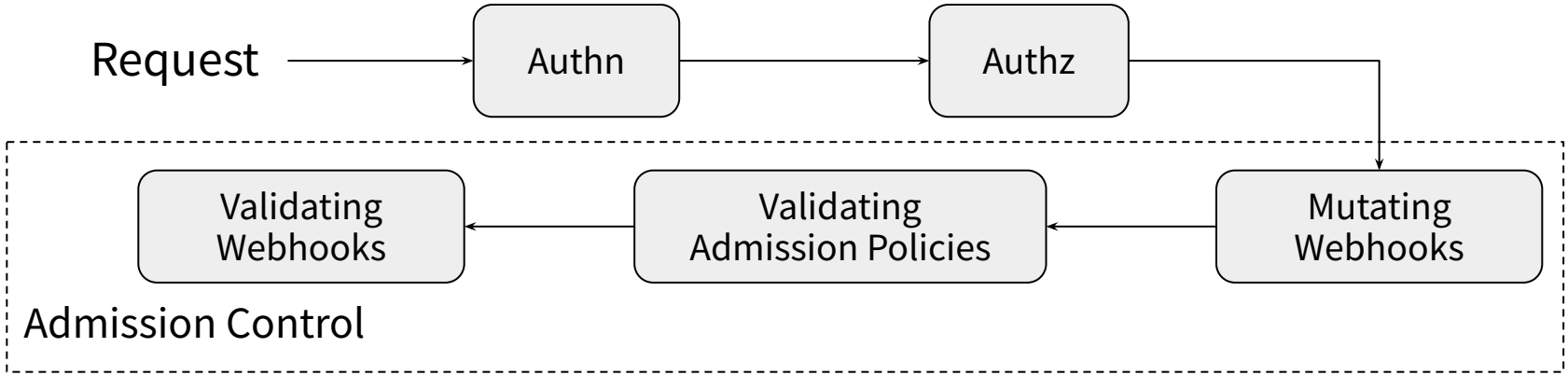

https://kubernetes.io/docs/concepts/security/controlling-access/

# Access to Kubernetes API - TLS, Authn, Authz

- Kubernetes API server is protected by TLS
- Authenticate with modules (Authentication, Authn)
  - Client certificates, password, plain tokens, bootstrap tokens, JSON Web Tokens(JWT)
  - Try in sequence, until one succeeds
  - Login as a specific `username` and `group`
- Authorize with modes (Authorization, Authz)
  - Attribute-based Access Control (ABAC)
  - Role-based Access Control (RBAC)
  - Node Authorization
  - Webhook

# Access to Kubernetes API - Admission Control

- Intercepts requests to the Kubernetes API server
- Modify or reject requests.
- Does not works on read requests
- Requests are rejected if any admission controller module rejects



# Authentication with Client Certificate

- Authenticate with valid client certificate signed by cluster CA
- Username and groups are determined in the *subject* of the certificate

```
Certificate:
   Data:
       Version: 3 (0x2)
       Serial Number: 3912010528935946623 (0x364a40d96ed0a97f)
       Signature Algorithm: ecdsa-with-SHA256
       Issuer: CN=k3s-client-ca@1774164932
       Validity
           Not Before: Mar 22 07:35:32 2026 GMT
           Not After : Mar 22 07:35:32 2027 GMT
       Subject: O=system:masters, CN=system:admin
```

# Authorization - Attribute-based Access Control

- To enable, configure kube-apiserver with flags
  - `--authorization-mode=ABAC,RBAC`
  - `--authorization-policy-file=/etc/kubernetes/abac-policy.jsonl`
  - `.jsonl` file: each line is a self-contained JSON object
- Matching user / groups subject
- Matching API group, namespace, resource
- Allow read/write or readonly

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "bob",
"apiGroup": "*", "namespace": "projectCaribou", "resource": "pods", "readonly": true}}
```

# Authorization - Role-based Access Control

- Role / ClusterRole
  - Contains rules that represent **a set of permission**
- RoleBinding / ClusterRoleBinding
  - Grants the permissions defined in **a Role** to a user or **set of users**

https://kubernetes.io/docs/reference/access-authn-authz/rbac/

# RBAC - Role

- Sets permission within a particular namespace
- `rules` contains `apiGroups`, `resources`, `resourceNames`, `verbs`
  - `"*"` represents all
  - `""`(empty string) represents core API group in `apiGroups`
  - Common verbs: `get`, `watch`, `list`, `create`, `delete`, `patch`, `update`

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

# RBAC - ClusterRole

- Cluster-scoped roles
- No `namespace` in `metadata`

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  # at the HTTP level, the name of the resource for accessing Secret
  # objects is "secrets"
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

# RBAC - RoleBinding

- Grants the permissions defined in a Role to set of users or groups **in a specific namespace**
- Reference a **Role in the same namespace** or a **ClusterRole**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# You need to already have a Role named "pod-reader" in that namespace.
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  # "roleRef" specifies the binding to a Role / ClusterRole
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name of the Role or ClusterRole you wish to bind to
  apiGroup: rbac.authorization.k8s.io
```

# RBAC - ClusterRoleBinding

- Grant permissions across a **whole cluster**
- Reference only **ClusterRole**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This cluster role binding allows anyone in the "manager" group to read secrets in any namespace.
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

# RBAC Examples

- *tsaimh* is maintainer
  - Can edit all the resources in `people-test` and `people-prod` namespace
- *phkoan*, *bjhuang* are developers
  - Can edit resources in `people-test` namespace
  - Can view resources in `people-prod` namespace
- *ytshih* is reporter
  - Can view resources in `people-test` and `people-prod` namespace

# RBAC Examples - Role

- Create `editor`, `viewer` ClusterRole

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: editor
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["get", "list", "watch",
"create", "update", "patch", "delete"]
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: viewer
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["get", "list", "watch"]
```

# RBAC Examples - RoleBinding

- In `people-test` namespace
  - Create maintainer and developer RoleBinding for editor ClusterRole
  - Create reporter RoleBinding for viewer ClusterRole
- In `people-prod` namespace
  - Create maintainer RoleBinding for editor ClusterRole
  - Create developer and reporter RoleBinding for viewer ClusterRole

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: developer-binding
  namespace: people-test
subjects:
  - kind: User
    name: phkoan
    apiGroup: rbac.authorization.k8s.io
  - kind: User
    name: bjhuang
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: editor
  apiGroup: rbac.authorization.k8s.io
```

# RBAC Examples - RoleBinding

- Alternatively, use user group for reporter, developer, and maintainer
  - Provided by client certificate subjects rather than specify user explicitly in RoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: reporter-binding
  namespace: people-prod
subjects:
  - kind: Group
    name: reporter
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: viewer
  apiGroup: rbac.authorization.k8s.io
```

# Admission Controllers

- Run in admission control phases
- Required by several important features of Kubernetes
- Toggle by kube-apiserver parameters
  - `kube-apiserver --enable-admission-plugins=NamespaceLifecycle,LimitRanger`
  - `kube-apiserver --disable-admission-plugins=PodNodeSelector,AlwaysDeny`
  - Some plugins are enabled by default
    - In 1.36, i.e. NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PodGroupProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, ClusterTrustBundleAttest, CertificateSubjectRestriction, DefaultIngressClass, PodTopologyLabels, PodGroupWorkloadExists, NodeDeclaredFeatureValidator, JobValidation, PodResizeValidator, MutatingAdmissionPolicy, MutatingAdmissionWebhook, ValidatingAdmissionPolicy, ValidatingAdmissionWebhook, ResourceQuota

# FYR: Admission Controllers

| | |
|---|---|
| CertificateApproval,<br>CertificateSigning,<br>CertificateSubjectRestriction,<br>ClusterTrustBundleAttest | Observes requests to CertificateSigningRequest resources with additional authorization checks |
| MutatingAdmissionWebhook,<br>ValidatingAdmissionPolicy,<br>ValidatingAdmissionWebhook | Implements dynamic admission control and CEL validation |
| DefaultIngressClass | Adds a default ingress class to Ingress objects that do not request any specific ingress class |
| NamespaceLifecycle | Enforces that a Namespace that is undergoing termination cannot have new objects in it<br>Ensures that requests in a non-existent Namespace are rejected |

# FYR: Admission Controllers

| | |
|---|---|
| ServiceAccount | Implements automation for ServiceAccounts |
| RuntimeClass | Configures Pod overhead with RuntimeClass |
| TaintNodesByCondition | Taints newly created Nodes as `NotReady` and `Noschedule` |
| LimitRanger, ResourceQuota | Ensures incoming request that it does not violate any of the constraints enumerated in the LimitRange / ResourceQuota object |

# FYI: Admission Webhooks (Dynamic Admission Control)

- Admission plugins can be run as **webhooks**
- Enable by toggle plugins
- Validating admission webhooks
- Mutating admission webhooks

https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/

# Workload Protection

# Workload Protection

- Pod Security Standards
  - Three policies broadly cover the security specturm
- Pod Security Admission
  - Configure Pod security standards for namespace
- Container Runtimes / Runtime Class
  - Select the container runtime configuration by handler name

# Pod Security Standards (PSS)

- `privileged`
  - Unrestricted, allows privilege escalations
- `baseline`
  - Aims at ease of adoption, prevents known privilege escalations
  - e.g. no `privileged: true`, no host network / host path volume
- `restricted`
  - Enforce hardening best practices, at the expense of some compatibility
  - e.g. must `runAsNonRoot: true`, drop all capabilities

# Pod Security Admission

- Configure Pod security in each namespace
- `enforce`
  - Policy violations will cause the pod to be rejected
- `audit`
  - Allow, but will trigger audit annotation to the event
- `warn`
  - Allow, but will trigger a user-facing warning

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-baseline-namespace
  labels:
     pod-security.kubernetes.io/enforce: baseline
     pod-security.kubernetes.io/audit: restricted
     pod-security.kubernetes.io/warn: restricted
```

# Recap: Container Runtimes

- Responsible for running containers
- Requires to use a runtime that conforms with the **Container Runtime Interface (CRI)** after v1.35
- Common container runtimes
  - containerd
  - CRI-O
  - Docker Engine
  - Mirantis Container Runtime

# Runtime Class

- Select the container runtime configuration by handler name
- Provide a balance of performance versus security
  - e.g. run container in hardware virtualization
- Runtime handlers should be configured through CRI runtime config
  - e.g. containerd

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runsc]
```

```yaml
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: gvisor
handler: runsc
scheduling:
  nodeSelector: {}
```

# Workload Protection Examples

- Recap: An Nginx server that serves user-uploaded websites
  - The Pod is considered untrusted
- By the **Principle of Least Privilege**, we should
  - Use the highest level of Pod Security Standards
  - Run in an isolated environment

# Workload Protection Examples

- For `people-test` and `people-prod` namespace, use enforce: restricted
  - Use unprivileged image and runAsNonRoot as restricted PSS suggested
- Use gVisor RuntimeClass for additional isolation

```yaml
spec:
  runtimeClassName: gvisor
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: nginx
    image: docker.io/nginxinc/nginx-unprivileged:alpine3.23-perl
    ports:
      - containerPort: 8080
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
```

# Policies

# Policies

- Recap: Limit Ranges / Resources Quotas
  - Manage and limit resource consumption
- Network Policies
  - Restrict ingress and egress traffic for a Pod
- Admission Policies
  - Validate or mutate API requests

# Network Policies

- Control **East-West** traffic flow at IP address or port level (OSI layer 3, 4)
  - For North-South traffic, do it on Ingress or Gateway API level
- Implemented by the network plugin (CNI plugin)
  - Without a valid controller, network policies has no effect (e.g. flannel-io/flannel)
  - Calico and Cilium support it
- Network policies are **additive whitelist**
- By default, a pod is non-isolated for ingress(in) and egress(out)
- Implicit **deny all** applied if a network policy for the direction exists
  - I.e. **isolated** for that direction

https://kubernetes.io/docs/concepts/services-networking/network-policies/

# Network Policy Examples

- Recap: An Nginx server that serves user-uploaded websites
  - The Pod is considered untrusted
- Restrict traffic from the Pod to other critical services
  - Deny all egress traffic
  - Deny all cross namespace ingress traffic
- Allow ingress only from ingress controller namespace

# Network Policy Examples

- Restrict traffic from the Pod to other critical services
  - Deny all egress traffic
  - Deny all cross namespace ingress traffic

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-egress
  namespace: people-test
spec:
  podSelector:
    matchLabels:
      app: nginx-gvisor
  policyTypes:
    - Egress
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-cross-namespace-ingress
  namespace: people-test
spec:
  podSelector:
    matchLabels:
      app: nginx-gvisor
  policyTypes:
    - Ingress
  ingress:
    - from:
        - podSelector: {}
```

# Network Policy Examples

- Allow ingress only from ingress controller namespace

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: allow-traefik-ingress
 namespace: test
spec:
 podSelector:
   matchLabels:
     app: nginx-gvisor
 policyTypes:
 - Ingress
 ingress:
 - from:
   - namespaceSelector:
       matchLabels:
         kubernetes.io/metadata.name: traefik-test
 - from:
   - namespaceSelector:
       matchLabels:
         kubernetes.io/metadata.name: traefik-prod
```

# FYI: Common Expression Language (CEL)

- Declare validation rules, policy rules, and other constraints or conditions
- Evaluated directly in the API server
- A convenient alternative to out-of-process mechanisms (e.g. webhooks)
- E.g.
  - `self.minReplicas <= self.replicas && self.replicas <= self.maxReplicas`
  - `self.health.startsWith('ok')`
  - `self.envars.filter(e, e.name = 'MY_ENV').all(e, e.value.matches('^[a-zA-Z]*$'))`
- You can learn more about CEL at Kubernetes Documentation
  - https://kubernetes.io/docs/reference/using-api/cel/

# FYI: Admission Policies

- Offer a declarative, in-process alternative to admission webhooks
- Use the Common Expression Language (CEL)
- Mutating Admission Policy
- Validating Admission Policy

https://kubernetes.io/docs/reference/access-authn-authz/validating-admission-policy/
https://kubernetes.io/docs/reference/access-authn-authz/mutating-admission-policy/

# Continuous Delivery

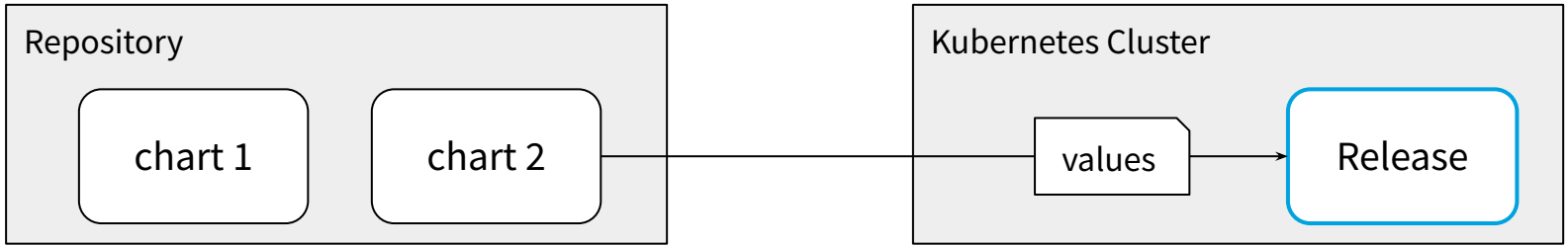
# Continuous Delivery

- Helm
- Kustomize
- Jsonnet
- GitOps
  - FluxCD
  - ArgoCD

# Helm

# Helm

- A tool for managing Kubernetes packages called **chart**s
- A **chart** is a bundle of information to create an instance of Kubernetes application
  - E.g. metadata, templates, configs, dependencies
- A **release** is a running instance of a **chart**, with a specific config
- A **repository** is the place where charts can be collected and shared



# Chart Structure

- A chart is a collection of files inside of a directory
- Templates are written in **Go template language**
- Value files (`values.yaml`) can be overridden while installing

```
wordpress/
  Chart.yaml          # A YAML file containing information about the chart
  LICENSE             # OPTIONAL: A plain text file containing the license for the chart
  README.md           # OPTIONAL: A human-readable README file
  values.yaml         # The default configuration values for this chart
  values.schema.json  # OPTIONAL: A JSON Schema for imposing a structure on the values.yaml file
  charts/             # A directory containing any charts upon which this chart depends.
  crds/               # Custom Resource Definitions
  templates/          # A directory of templates that, when combined with values,
                      # will generate valid Kubernetes manifest files.
  templates/NOTES.txt # OPTIONAL: A plain text file containing short usage notes
```

https://github.com/bitnami/charts/tree/1e56a501272c018515f275ada23d84a28de35eff/bitnami/wordpress

# Using Helm - Install a Package

- Search repos / charts on Artifact Hub(artifacthub.io)
- Add the repository for your desired charts
  - `helm repo add [NAME] [URL]`
  - Repository can be either HTTP/S-based or OCI-based
- Customize the chart by override `values.yaml`
  - `helm show values [CHART]` - see what options are configurable
- Install with custom configs
  - `helm install -f values.yaml [NAME] [CHART]`
- Uninstall a release
  - `helm uninstall [NAME]`

# Using Helm - Upgrade and Rollback

- Upgrade to a new version or change with a new config
  - `helm upgrade -f [NAME] [chart]`
- See history revisions for a release
  - `helm history [RELEASE]`
- Rollback a release to a reversion
  - `helm rollback [RELEASE] [REVISION]`

# Kustomize

# JSON Patch (RFC 6902)

- A format for expressing a sequence of operations to apply to a target JSON document
  - Suitable for HTTP PATCH method with MIME type `application/json-patch+json`
- Operations are applied sequentially in order

```
[
   { "op": "test", "path": "/a/b/c", "value": "foo" },
   { "op": "remove", "path": "/a/b/c" },
   { "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ] },
   { "op": "replace", "path": "/a/b/c", "value": 42 },
   { "op": "move", "from": "/a/b/c", "path": "/a/b/d" },
   { "op": "copy", "from": "/a/b/d", "path": "/a/b/e" }
 ]
```

# JSON Merge Patch (RFC 7386, 7396)

- Describes changes to made using a syntax that closely mimics the document being modified
- Comparing the content of the patch against the current content of the target document

| Target \ Patch | `null` | contains member |
|---|---|---|
| **exist** | remove | replace |
| **not exist** | do nothing | add |

# JSON Merge Patch Examples

Target

```
{
  "a": "b",
  "c": {
    "d": "e",
    "f": "g"
  }
}
```

+

Patch

```
{
  "a": "z",
  "c": {
    "f": null
  }
}
```

=

Result

```
{
  "a": "z",
  "c": {
    "d": "e"
  }
}
```

# JSON Strategic Merge Patch

- In JSON Merge Patch(RFC 7386, 7396), lists are always replaced
- Strategic Merge Patch use directives for operations
  - e.g. replace, delete
- `kind` and `metadata.name` and must be specified
  - FYR: https://github.com/kubernetes-sigs/kustomize/blob/313aacedd3adcb6564fd362bcef52f5aa3abef85/api/resource/resource.go#L497
- Commonly used in K8s, notably kustomize and `kubectl patch`

https://github.com/kubernetes/community/blob/7a1c92e8cf5d566cbfa644865ebaf044cf6fb94a/contributors/devel/sig-api-machinery/strategic-merge-patch.md

# JSON Strategic Merge Patch - Merge

- Not only maps are merged, but also lists
  - Unlike JSON Merge Patch

Target

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: a
    image: busybox
```

+

Patch

```
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: b
    image: notbusybox
```

=

Result

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - image: notbusybox
    name: b
  - image: busybox
    name: a
```

# JSON Strategic Merge Patch - Replace

- Replace an element using syntax: `$patch: replace`
- Apply to all fields of the map / list it's in

Target

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: a
    image: busybox
```

+

Patch

```
kind: Pod
metadata:
  name: test
spec:
  $patch: replace
  containers:
  - name: b
    image: notbusybox
```

=

Result

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - image: notbusybox
    name: b
```

# JSON Strategic Merge Patch - Delete Maps

- Delete an element using syntax: `$patch: delete`
- The element that contains it should be deleted
- `null` in RFC 7396 can also be used

Target

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: a
    image: busybox
```

+

Patch

```
kind: Pod
metadata:
  name: test
spec:
  $patch: delete
  containers: {}
```

=

Result

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec: {}
```

# JSON Strategic Merge Patch - Delete List of Maps

- Delete operation will delete all entries in the list that match the merge key

Target

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: a
    image: busybox
  - name: b
    image: notbusybox
```

+

Patch

```
kind: Pod
metadata:
  name: test
spec:
  containers:
  - $patch: delete
    name: a # merge key
  - name: c
    image: lazybox
```

=

Result

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: b
    image: notbusybox
  - name: c
    image: lazybox
```

# Kustomize

- A command line tool supporting template-free, structured customization of declarative configuration targeted to K8s-style objects
- Builds K8s resources according to `kustomization.yaml`
- A Kustomization file (`kustomization.yaml`) contains
  - Resources: what existing resources are to be customized
  - Generators: what new resources should be created
  - Transformers: what to do to the resources

# Kustomization - Resources

- Resources accept
  - Local files
  - Local directories
  - Remote URLs

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - ./docker/
  - ./docker-gcr/
  - ./ecr/
  - ./gcr/
  - ./ghcr/
  - ./k8s/
  - ./quay/
  - ./gitlab/
  - namespace.yaml
  - ingressroute.yaml
  - configMap.yaml
```

https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/resource/

# Kustomization - Generators

- ConfigMap generator
  - Generate ConfigMap resources
  - From file,  literals, or env file
  - Match arguments in `kubectl create configmap`
    - `--from-file`
    - `--from-literal`
    - `--from-env-file`
- Secret generator works the same

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

configMapGenerator:
- name: my-java-server-props
  behavior: merge
  files:
  - application.properties
  - more.properties
- name: my-java-server-env-file-vars
  envs:
  - my-server-env.properties
  - more-server-props.env
- name: my-java-server-env-vars
  literals:
  - JAVA_HOME=/opt/java/jdk
  - JAVA_TOOL_OPTIONS=-agentlib:hprof
  options:
    disableNameSuffixHash: true
    labels:
      pet: dog
```

# Kustomization - Transformers

- Annotation transformer
  - Add annotations (non-identifying metadata) to all resources and selectors
- Label transformer
  - Add labels to all resources and selectors
- Namespace transformer
  - Add namespace to all resources
- PrefixSuffix transformer
  - Prepends or postfixes the value to the names of all resources
- ReplicaCount transformer
  - Replicas modified the number of replicas for a resource

# Kustomization - Patch Transformer

- Add or override fields on resources
- Is either a **strategic merge patch**, or a **JSON6902 patch**
- Is either a file, or an inline string
- Use `target` to match the fields to apply
  - If a `target` is specified, the name contained in the metadata is required but not used

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

patches:
- path: patch.yaml
  target:
    group: apps
    version: v1
    kind: Deployment
    name: deploy.*
    labelSelector: "env=dev"
    annotationSelector: "zone=west"
- patch: |-
    - op: replace
      path: /some/existing/path
      value: new value
  target:
    kind: MyKind
    labelSelector: "env=dev"
```

# Kustomization - Overlays

- When the kustomization use a directory as resource
  - The encapsulating kustomization is called an **overlay**
  - What it refers to is called a **base**
- In this case, app1 and app2 are overlays
- Useful when deploying multiple instances with most of the resources are shared
  - E.g. testing and production, multi-tenant

```
overlays/prod/
  kustomization.yaml
    | resources:
    | - ../base
    | patches:
    | - patch1.yaml
  patch1.yaml

overlays/test/
  kustomization.yaml
    | resources:
    | - ../base
    | patches:
    | - patch2.yaml
  patch2.yaml

base/
  kustomization.yaml
    | resources:
    | - deployment.yaml
    | - configMap.yaml
  deployment.yaml
  configMap.yaml
```

# FluxCD

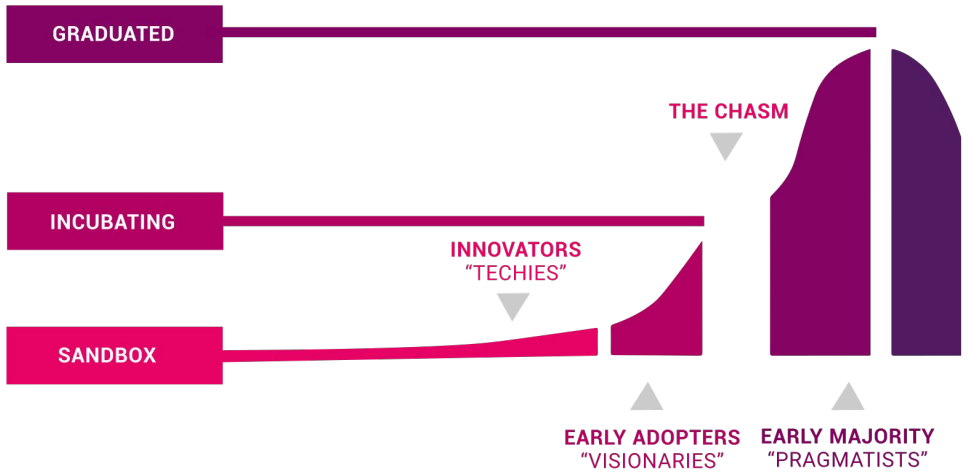
# GitOps

- Declarative
  - A system managed by GitOps must have its desired state expressed **declaratively**
- Versioned and Immutable
  - Desired state is stored in a way that enforces **immutablility**, and **versioning**
- Pulled Automatically
  - Software agents **automatically pull** the desired state declarations from the **source**
- Continuously Reconciled
  - Software agents continuously **observe actual system state** and attempt to **apply the desired state**

https://opengitops.dev/#principles

# FluxCD

- A tool for keeping Kubernetes clusters in sync with sources of configuration (e.g. Git repositories)
- GitOps for apps and infrastructure
- CNCF Graduated project

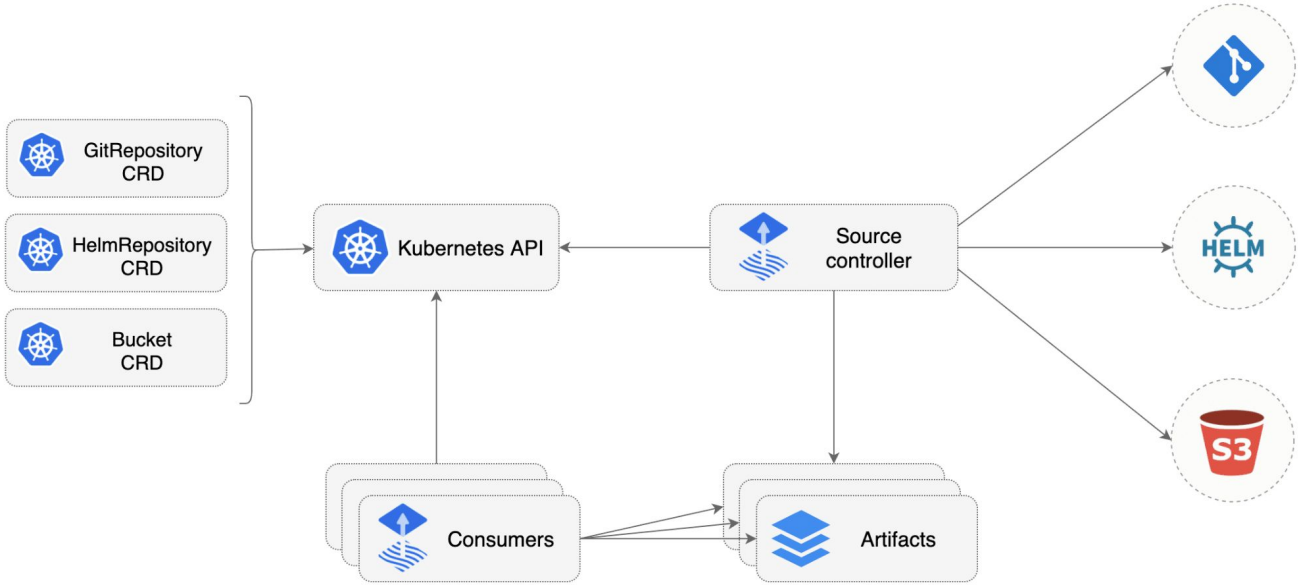


# FluxCD - Concepts

- Sources
  - Define the origin of repositories containing the desired state of the system
  - Consumed by other Flux components
  - E.g. GitRepository, OCIRepository, HelmRepository
- Kustomization
  - Represents a set of Kubernetes resources
- Reconciliation
  - Ensure the running state matches the desired state
    - Detect states and correct
  - E.g. GitRepository, HelmRelease, Kustomization reconciliation

# FluxCD - Source Controllers

- Provide a common interface for artifacts(archived resources) acquisition
- Fetch resources on-demand and on-a-schedule

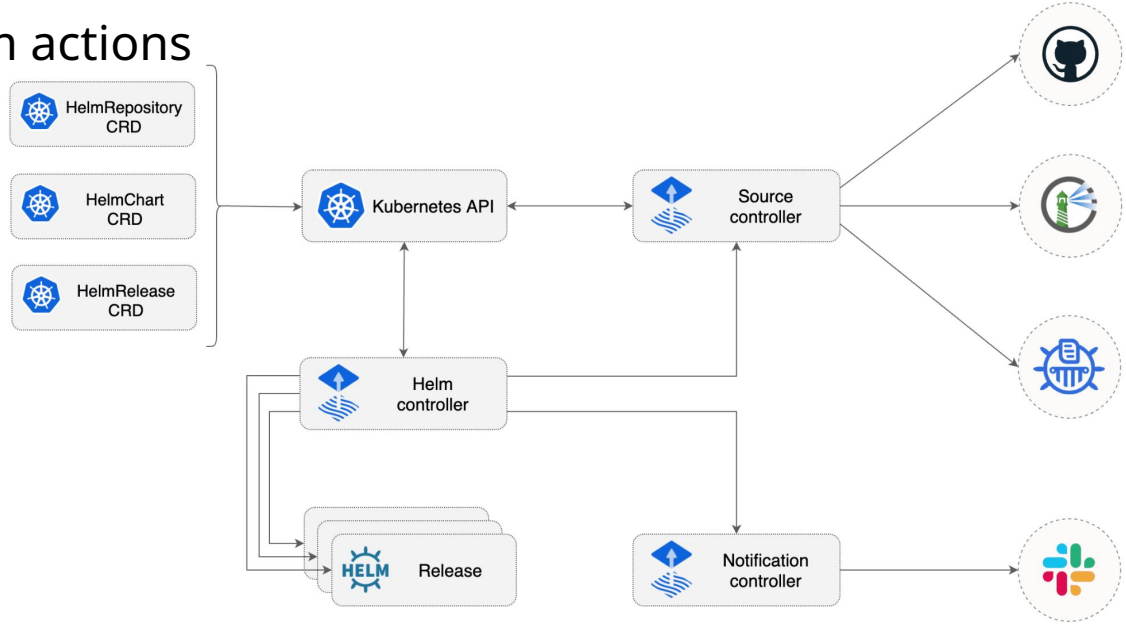


# FluxCD - Source Resources

- Git Repositories
  - Produce an Artifact for a Git repository revision
- OCI Repositories
  - Produce an Artifact for an OCI repository
- Helm Repositories
  - For HTTP/S-based repository, produce an Artifact for a Helm repository **index** YAML
  - For OCI-based repository, store the information about the repository
- Helm Charts
  - Produce an Artifact for a Helm chart archive
  - Usually managed by the helm controller, based on the HelmRelease configuration

# FluxCD - Helm Controller

- Watches for HelmRelease objects and generates HelmChart objects
- Watches HelmChart objects for revision changes
- Performs automated Helm actions



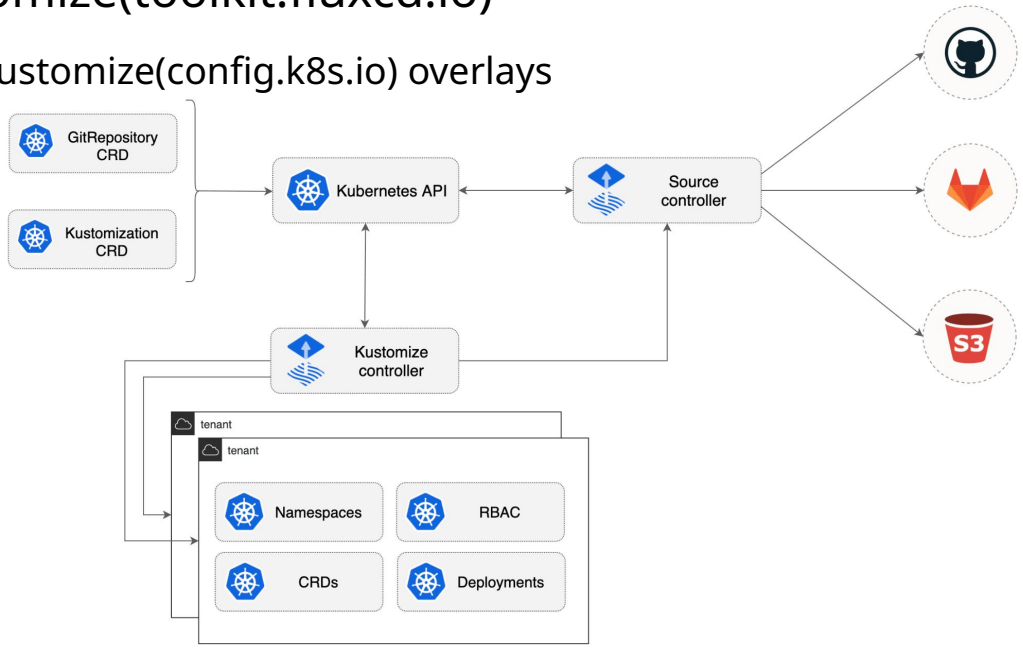
# FluxCD - Helm Releases

- Controller-driven reconciliation of Helm releases
  - E.g. install, upgrade, uninstall
- Detects and corrects state drift
- Create Helm Chart object from `.spec.chart` and watch it for Artifact changes

| | **HelmChart** | **HelmRelease** |
|---|---|---|
| **Purpose** | Sync Chart Artifacts | Apply Chart with custom values |

# FluxCD - Kustomize Controller

- Reconcile cluster states from multiple sources
- Generates manifests with Kustomize(toolkit.fluxcd.io)
  - From plain Kubernetes YAMLs or Kustomize(config.k8s.io) overlays



# Two Kustomizations

| | | |
|---|---|---|
| **API version** | kustomize.toolkit.fluxcd.io | kustomize.config.k8s.io |
| **Used by** | FluxCD Kustomize Controller | kubectl / kustomize |
| **Purpose** | Apply kustomize(.config.k8s.io) overlays or plain manifests | Resources with generators and transformers |

```
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
 name: flux-applications
 namespace: flux-system
spec:
 interval: 10m0s
 path: ./flux-applications
 prune: true
 force: false
 sourceRef:
   kind: GitRepository
   name: flux-applications
```

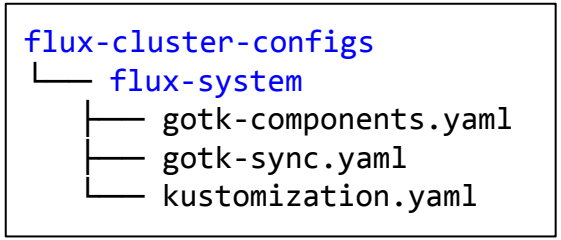
```
apiVersion:
kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: nginx-prod
resources:
 - ../../base
configMapGenerator:
 - name: nginx-html
   files:
     - index.html=index.html
patches:
 - path: patch.deployment.yaml
```

# Simplified Workload Scenario

- An Nginx server that serves user-uploaded websites
  - E.g. https://people.cs.nycu.edu.tw/~tsaimh/
- There are two instances of the app, one for testing, one for production
  - Two instances provide different content
- There are three types of user: reporter, developer, and maintainer
  - Maintainer > Developer > Reporter
  - Reporter can view the resources in the app
  - Developer can edit resources in testing instance
  - Maintainer can edit all resources of the app

# FluxCD Examples - Bootstrap

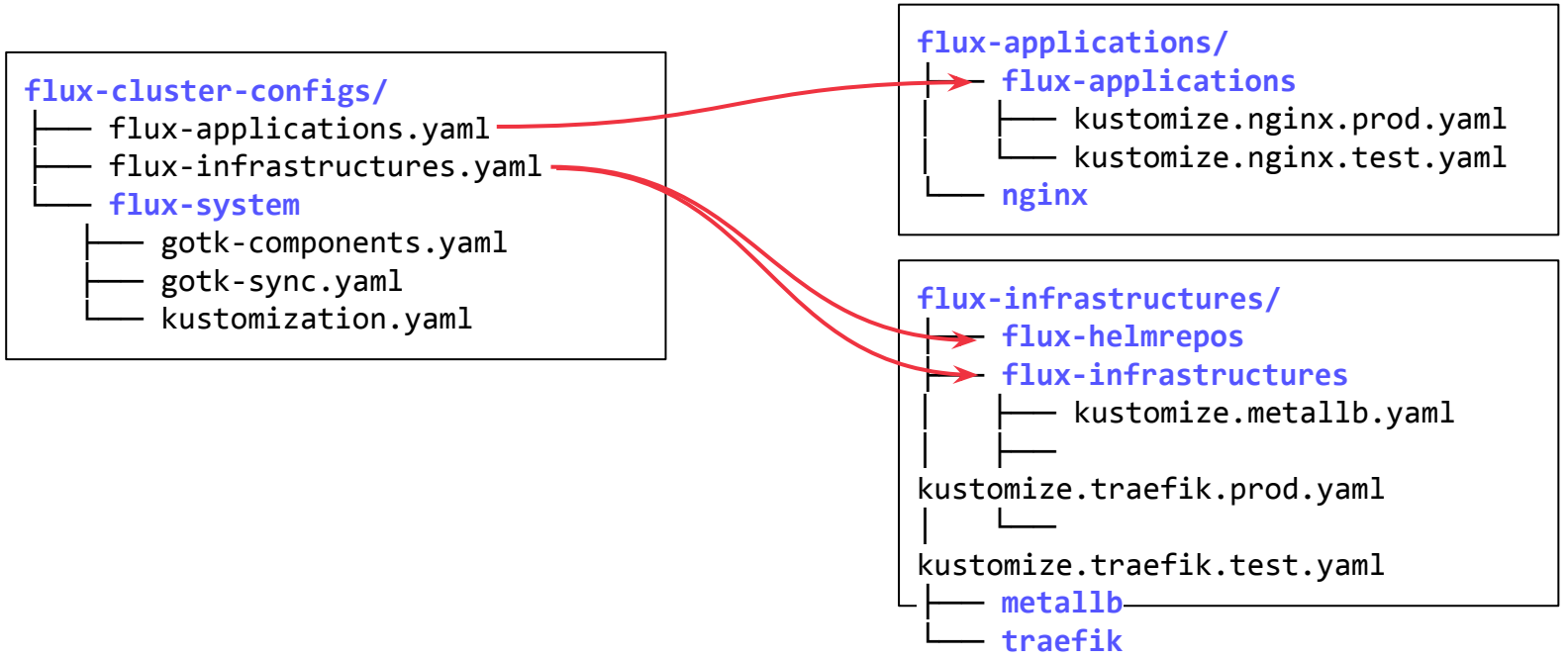
- `flux bootstrap <method>`
  - FluxCD supports various methods, e.g. Gitea, GitHub, GitLab, or generic Git server
  - We use generic Git server with SSH agent
  - `flux bootstrap git --url ssh://gitea@gitea.localhost/fluxcd/flux-cluster-configs.git --branch=main --private-key-file ./flux`
  - FluxCD will create a commit in the repo with FluxCD resources (i.e. CRDs, sources)



# FluxCD Examples - Repositories

- In this examples, the FluxCD kustomizations are placed in three different repositories
  - `flux-cluster-configs`: Resources created by FluxCD bootstrap
  - `flux-infrastructures`: Infrastructures that are less likely to change
  - `flux-applications`: Applications that change more frequently; the main workload
  - `flux-cluster-configs` should contain the GitRepository sources and Kustomization to `flux-infrastructures` and `flux-applications`
    - In the VM, files are placed in `~/repos/sources`. Copy them to `flux-cluster-configs`

# FluxCD Examples - Cluster Configs



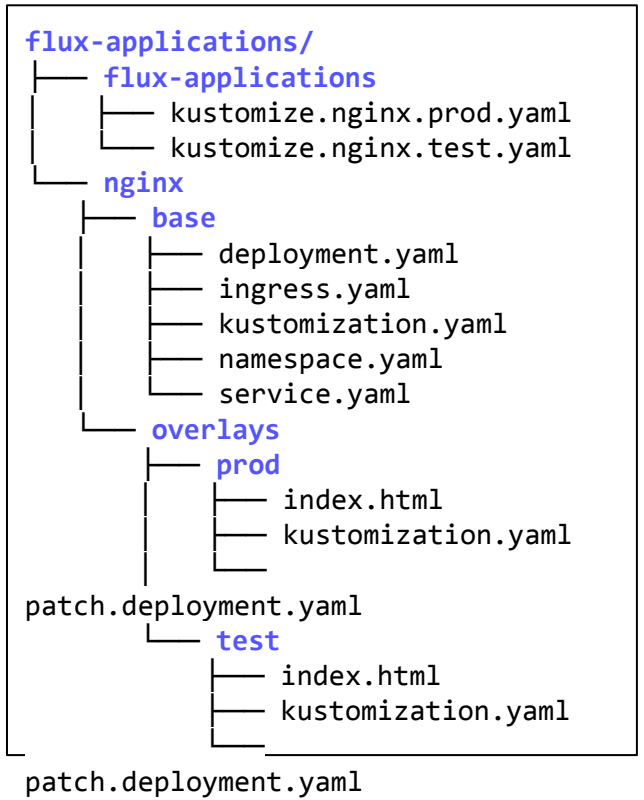
# FluxCD Examples - Infrastructures

- MetalLB for LoadBalancer
- Traefik for Ingress controller
  - Two instances for test and prod using overlays
- Both applications are deployed by HelmRelease

```
flux-infrastructures/
├── flux-helmrepos
│   ├── metallb.yaml
│   └── traefik.yaml
├── flux-infrastructures
│   ├── kustomize.metallb.yaml
│   ├── kustomize.traefik.prod.yaml
│   └── kustomize.traefik.test.yaml
├── metallb
│   ├── custom-resources
│   │   ├── kustomization.yaml
│   │   ├── l2advertisement.yaml
│   │   ├── pool.prod.yaml
│   │   └── pool.test.yaml
│   └── helmrelease
│       ├── helmrelease.yaml
│       ├── kustomization.yaml
│       ├── namespace.yaml
│       └── values.yaml
└── traefik
    ├── base
    │   ├── helmrelease.yaml
    │   ├── kustomization.yaml
    │   └── namespace.yaml
    └── overlays
        ├── prod
        │   ├── kustomization.yaml
        │   └── values.yaml
        └── test
            ├── kustomization.yaml
            └── values.yaml
```

# FluxCD Examples - Applications

- Nginx server
  - Two instances with different contents



# FluxCD Examples - Results

- After FluxCD reconciliation, there should be
  - Two Traefik instances with different IP address
  - Two Nginx instances with different IngressClass
  - Two Nginx instances have different contents

# References


- Kubernetes - https://kubernetes.io
- gVisor documentation - https://gvisor.dev/docs/
- Helm documentation - https://helm.sh/docs/
- Kubectl Kustomize documentation - https://kubectl.docs.kubernetes.io/references/kustomize/
- Flux documentation - https://fluxcd.io/flux/
- Kubernetes community content strategy merge patch - https://github.com/kubernetes/community/blob/main/contributors/devel/sig-api-machinery/strategic-merge-patch.md
- RFC 6902, 7386, 7396