Policy Based Security in Kubernetes usecases

Speaker.
Ravi Mishra

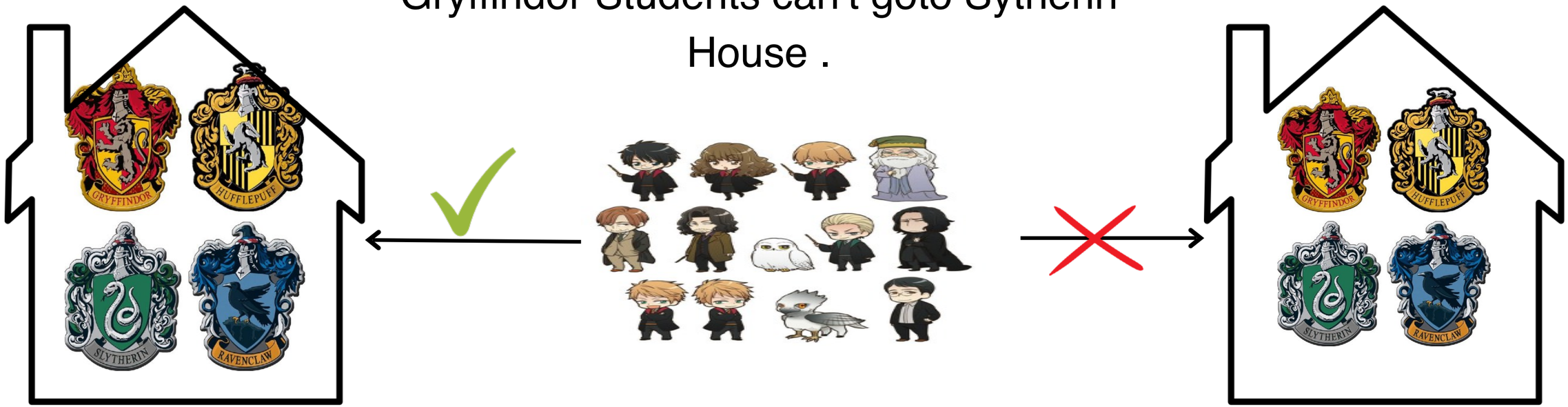
Kyverno

Policy Usecases

- Network Policy - Who can Go Where ?
- RBAC - Who can do what ?
- Pod Security Admission Controllers - Dumbeldore's Orders
  - Kyverno

Network Policy

- Gryffindor Students can't goto Sytherin House .

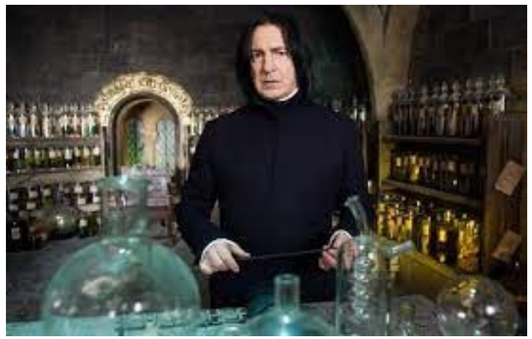


If we consider Grtffindor and Slytherin as two namepaces and Harry and his friends as Pods belonging to gryffindor, we know they cannot access Slytherin, as rule of Hogwarts.

Example: Suppose we have a Network Policy that do not allows traffic to Namespace A from Namespace B. This can be tested by trying to send a request from a pod of Namespace B to Pod of Namespace A. If the Network Policy is working correctly, Pod B's request should be denied but If we try to access Pod B from Pod A, it should be working perfectly fine.

RBAC: Role Based Access Control

ClusterRole: Dumbeldore is admin and can do anything in Hogwarts (Cluster).



Role: Severus Snape has role of Potion Master in the Classroom (Namespace)

Service Account -> Identity for Application

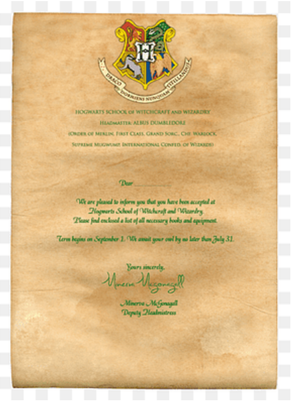
RoleBinding -> Mapping of Role to ServiceAccount/User

ClusterRoleBinding -> Mapping of ClusterRole with SA/User.

If we consider Hogwarts as a Cluster and Classrooms as a Namespaces. Then Dumbeldore is the Principal, he can perform any action on any classroom, whereas Professors can perform certain actions in there classrooms.

Example: Create a Role that only allows reading Pod information, and assign it to a User. Then, try to create a Pod as that User. The operation should be denied, indicating the RBAC is working correctly.

Open Policy Agent (OPA)/Kyverno Policies



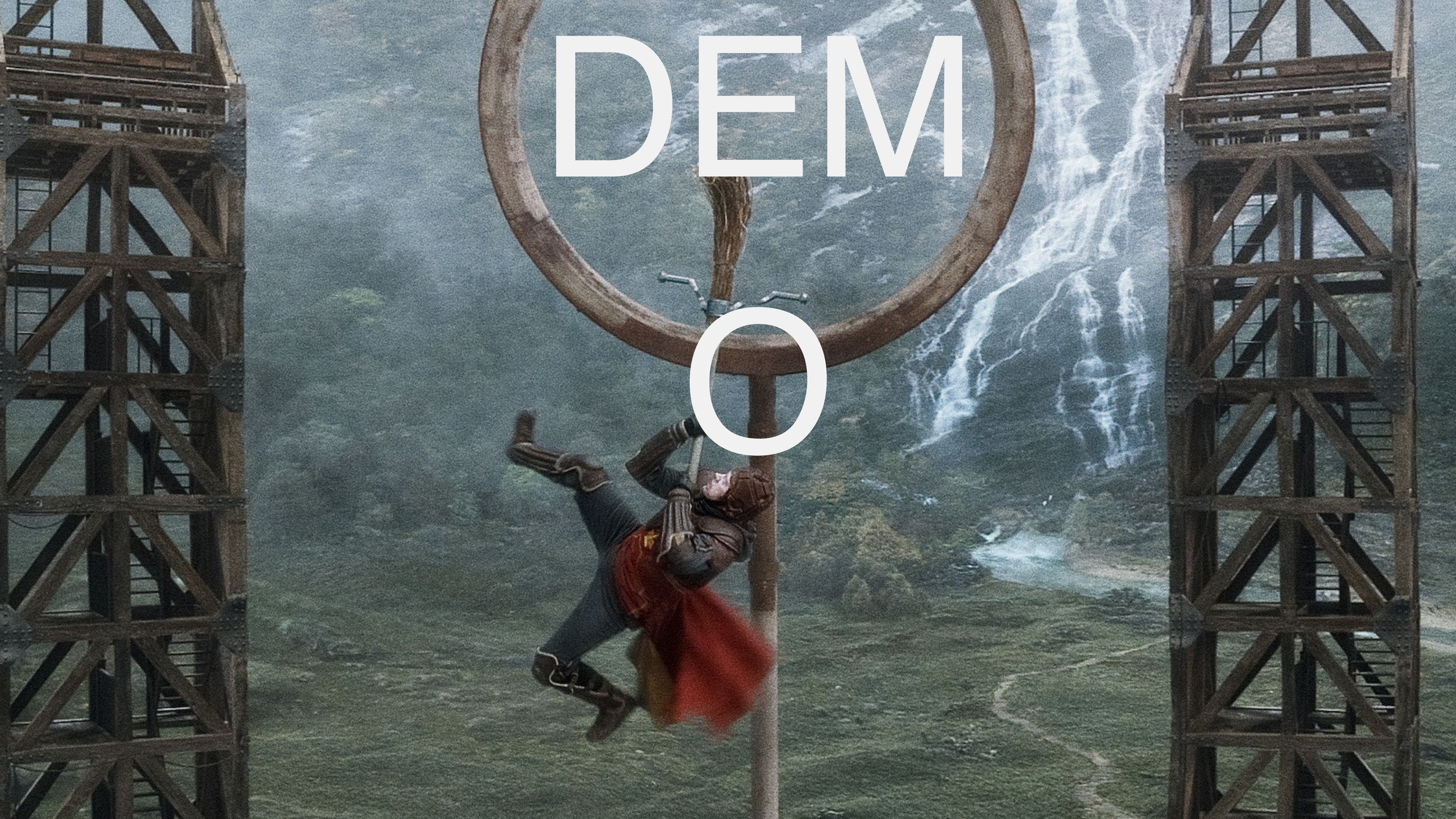
Validation: Hogwart's invite to join Hogwarts,

Mutation: Sorting into Houses by sorting hat.

Considering Hogwarts as a cluster, If we want to deploy a student into the campus they need to have an invitation from Hogwarts else deployment will fail. This was validation.

Now lets suppose our candidate is deployed into the Campus but we want them automatically set into a House, in that case we have Sorting Hat as Mutating Webhook, which here attaches a label of House (lets say Gryffindor) to our student.

Example: Create a Kyverno policy that requires all Pods to have a label "app". Then, try to create a Pod without this label. Kubernetes should prevent this Pod from being created if the policy is working correctly.


DEMO

Network Policy

Lets setup a small lab to test network policy.

Step 1: Create 3 namespaces, gryffindor, slytherin, ravenclaw

kubectl create ns gryffindor

kubectl create ns slytherin

kubectl create ns ravenclaw

once done, lets put a pod/student to each namespace.

kubectl run draco -n slytherin --image=nginx

kubectl run harry -n gryffindor --image=nginx

kubectl run luna -n ravenclaw --image=nginx

Once the pods are up and running, save the below policy to a file, networkpolicy.yaml

Lets create a policy to block access of Students of Gryffindor to Slytherin

- kubectl apply -f cilium-slytherinHouse.yaml

Now test the connectivity by running below command.

- kubectl get po -n gryffindor -owide | awk 'NR>1' | awk '{print $6}'
- kubectl exec -it harry -- curl <IP>

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "deny-ingress"
  namespace: slytherin
spec:
  endpointSelector:
    matchLabels:
    "k8s:io.kubernetes.pod.namespace":
gryffindor
  egress:
  - toEndpoints:
    - matchLabels: {}
  ingress:
  - fromEndpoints:
    - matchLabels: {}
```

Role-Based Access Control (RBAC)

Lets create 2 users, snape and dumbeldore.

Snape being just a professor should be able to see all students/pods of first year only whereas Dumbeldore being the headmaster should be able to see students/pods of whole Hogwarts.

Creating User Snape and trying to access pods of first year.

- openssl genrsa -out snape.key 2048
- openssl req -new -key snape.key -out snape.csr -subj "/CN=snape/O=hogwarts"
- kubectl apply -f snape-csr.yaml
- kubectl certificate approve snape
- kubectl get csr snape -o jsonpath='{.status.certificate}' | base64 --decode > snape.crt
- kubectl apply -f snape-rolebinding.yaml
- kubectl auth can-i list pods --as user snape -n first-year

Creating User Dumbeldore and trying to access pods.

- openssl genrsa -out cluster-reader.key 2048
- openssl req -new -key dumbeldore.key -out dumbeldore.csr -subj "/CN=dumbeldore/O=hogwarts"
- kubectl apply -f dumbeldore-csr.yaml
- kubectl certificate approve dumbeldore
- kubectl get csr dumbeldore -o jsonpath='{.status.certificate}' | base64 --decode > snape.crt
- kubectl apply -f dumbeldore-rolebinding.yaml
- kubectl auth can-i list pods --as user dumbeldore -n first-year
- kubectl auth can-i list pods --as user dumbeldore

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: snape
spec:
  request: $(cat snape.csr | tr -d '\n' | base64 )
  signerName: kubernetes.io/kube-apiserver-client
  usages:
  - digital signature
  - key encipherment
  - client auth
```

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: dumbeldore
spec:
  request: $(cat dumbeldore.csr  | tr -d '\n' | base64)
  signerName: kubernetes.io/kube-apiserver-client
  usages:
  - digital signature
  - key encipherment
  - client auth
```

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: first-year
  name: potion-master
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: potion-master
  namespace: first-year
subjects:
- kind: User
  name: snape
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: potion-master
  apiGroup: rbac.authorization.k8s.io
```

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: dumbeldore
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: dumbeldore
subjects:
- kind: User
  name: dumbeldore
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: dumbeldore
  apiGroup: rbac.authorization.k8s.io
```

Open Policy Agent (OPA)/Kyverno Policies

Kyverno is a policy engine designed for Kubernetes. It can validate, mutate, and generate configurations using policies.

Setup Kyverno

- helm repo add kyverno https://kyverno.github.io/kyverno/
- helm repo update
- helm install kyverno kyverno/kyverno -n kyverno --create-namespace --devel

Lets try to send harry to Hogwarts.

- kubectl apply -f wizardinvite.yaml

Add house automatically.

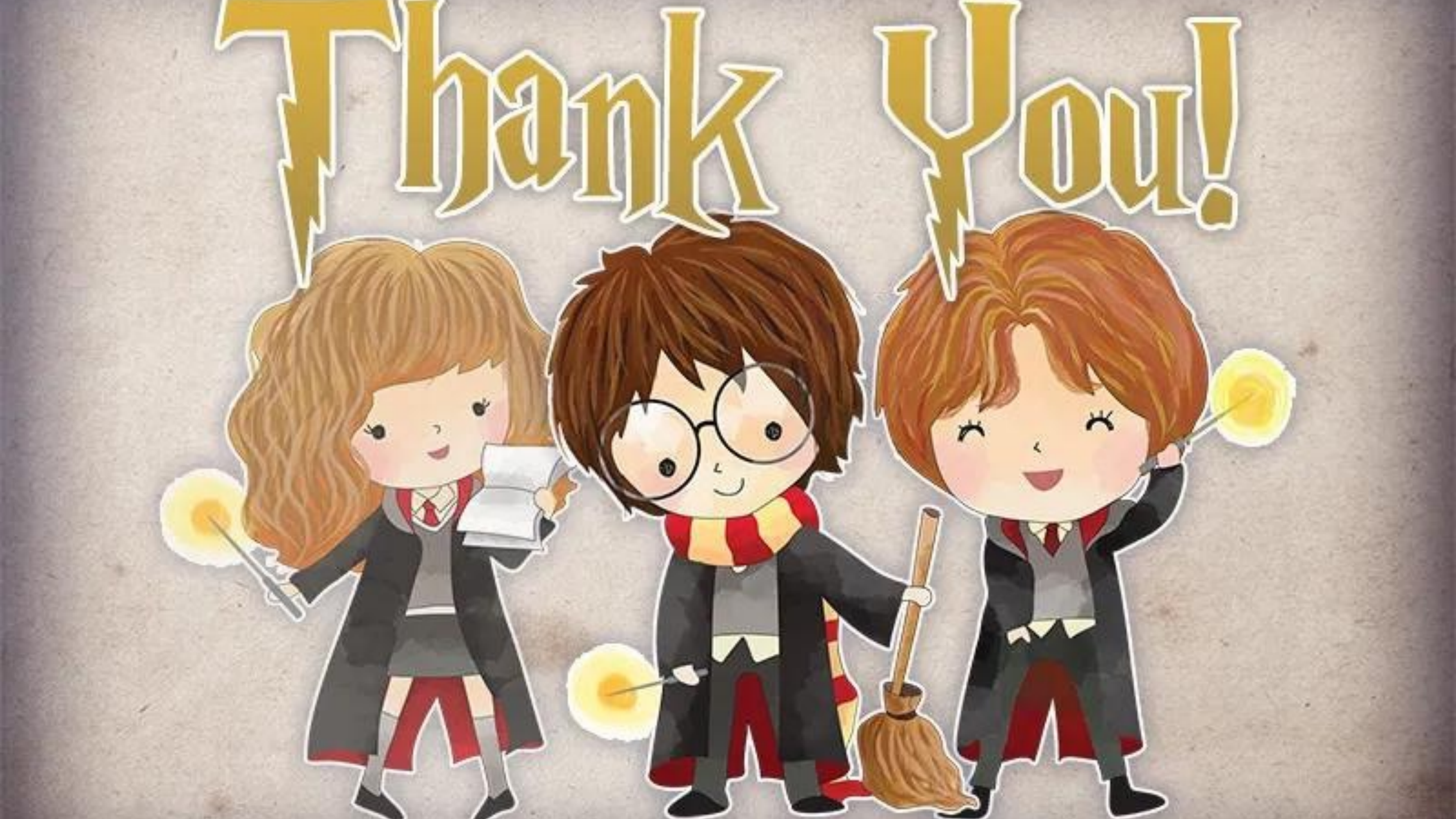
- kubectl apply -f add-house.yaml
- kubectl delete -f harry.yaml
- kubectl apply -f harry.yaml

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-wizard-invite
spec:
  validationFailureAction: Enforce
  rules:
  - name: check-wizard-invite
    match:
      resources:
        kinds:
        - Pod
    validate:
      message: "The invitation from Hogwarts 'invite' is required."
      pattern:
        metadata:
          labels:
            invite: "?*"
```

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-house
spec:
  background: false
  rules:
  - name: add-house-to-student
    match:
      resources:
        kinds:
        - Pod
    mutate:
      patchStrategicMerge:
        metadata:
          labels:
            house: gryffindor
```

Why did we chose Kyverno Policy Reporter ?

- Policy Language:
  - Kubewarden: Supports multiple languages. You can write policies using languages such as Rust, Go, AssemblyScript.
  - Kyverno: Uses YAML-based Kubernetes native policy management, which is a simpler approach for those already familiar with Kubernetes.
  - Gatekeeper: Uses Rego from Open Policy Agent (OPA) as its policy language.
- Complexity:
  - Kubewarden: Being language agnostic, the complexity depends on the policy author’s chosen language.
  - Kyverno: As the policies are written in YAML, it might be easier for users already familiar with Kubernetes and YAML.
  - Gatekeeper: The learning curve for Rego can be steep.
- Policy Execution:
  - Kubewarden: Policies are compiled to WebAssembly and executed in a sandboxed environment, providing an added layer of security.
  - Kyverno: Policies are executed inside the Kyverno controller in the Kubernetes cluster.
  - Gatekeeper: Policies are executed inside the Gatekeeper controller.
- Mutating Policies:
  - Kubewarden: Supports mutating admission policies.
  - Kyverno: Also supports mutating admission policies, as well as generating policies.
  - Gatekeeper: As of my knowledge cutoff in September 2021, it did not support mutating policies.
- Policy Distribution:
  - Kubewarden: Policies can be distributed using regular OCI registries.
  - Kyverno: Policies are typically defined as Kubernetes resources and applied directly to the cluster.
  - Gatekeeper: Policies are distributed as Kubernetes Custom Resource Definitions (CRDs).


Thank You!