From Fragile to Resilient:
ValidatingAdmissionPolicies Strengthen Kubernetes

Cloud Native Day Oslo

March 13th 2025

Hi 👋,

I'm **Marcus Noble!**

I'm a platform engineer at Giant Swarm working on release engineering, CI/CD and general Kubernetes development.

I run a monthly newsletter - [CloudNative.Now]

7+ years experience running Kubernetes in production environments.

💙☸️

🐘 @Marcus@k8s.social | 🌐 MarcusNoble.com | 🦋 @averagemarcus.bsky.social

Dynamic Admission Control

ValidatingAdmission Webhook

MutatingAdmission Webhook

Purpose / Use Cases

Defaulting

- Injecting `imagePullSecrets` dynamically when pods are created
- Injecting sidecars into pods
- Injecting proxy environment variables into pods

Policy Enforcement

- Prevent using `latest` image tag
- Require all pods to have resource limits set
- Block the use of deprecated Kubernetes APIs (e.g. `batch/v1beta1`)

Best Practices

- Require a `PodDisruptionBudget`
- Enforce a standard set of labels / annotations on all resources
- Replace all image registries with an in-house container image proxy / cache

Problem Mitigation

- Block nodes joining the cluster with known CVEs based on the kernel version (e.g. CVE-2022-0185)
- Inject Log4Shell mitigation env var into all pods (CVE-2021-44228)
- Block binding to the cluster-admin role

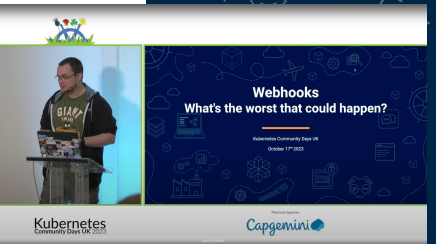
Webhooks in Kubernetes are

✨ **POWERFUL** ✨

But with that power comes

😱 **RISK** 😱

Taken from my talk about this at KCD UK



Wouldn’t it be great if we had a safer alternative?

Yes! *Yes, it would!*

Introducing ValidatingAdmissionPolicies

Status: Alpha in v1.26, Beta in v1.28, GA in v1.30

Introduced in: KEP-3488

A declarative, in-process alternative to validating admission webhooks.

Introducing ValidatingAdmissionPolicies

Status: Alpha in v1.26, Beta in v1.28, GA in v1.30

Introduced in: KEP-3488

A declarative, in-process alternative to validating admission webhooks.

Introducing ValidatingAdmissionPolicies

Status: Alpha in v1.26, Beta in v1.28, GA in v1.30

Introduced in: KEP-3488

api-server

A declarative, in-process alternative to validating admission webhooks.

Kubernetes manifests

Introducing ValidatingAdmissionPolicies

Status: Alpha in v1.26, Beta in v1.28, GA in v1.30

Introduced in: KEP-3488

api-server

A declarative, in-process alternative to validating admission webhooks.

Kubernetes manifests

More on this shortly

Uses the Common Expression Language (CEL) for the policy language.

Introducing ValidatingAdmissionPolicies

Status: Alpha in v1.26, Beta in v1.28, GA in v1.30

Introduced in: KEP-3488

api-server

A declarative, in-process alternative to validating admission webhooks.

Kubernetes manifests

More on this shortly

Uses the Common Expression Language (CEL) for the policy language.

Consists of two main resources:

- `ValidatingAdmissionPolicy` describes the policy logic
- `ValidatingAdmissionPolicyBinding` links the above policy to the resources it applies to

Brief introduction to CEL

- Uses a similar syntax to the expressions in C-based languages, e.g.
  `self.minReplicas <= self.replicas && self.replicas <= self.maxReplicas`
- Designed to be embedded into other applications with a focus on “one-liners” of code.
- Small number of built in functions (e.g. `split`, `has`)
- Functions expanded through custom libraries (Kubernetes includes several of these)
- Already used by Kyverno, Tekton, etc.
- Has a concept of “cost” per operation.
  This can be calculated ahead of running the expression and if needed, block its execution if too expensive.

References:

- https://kubernetes.io/docs/reference/using-api/cel/
- https://github.com/google/cel-go
- https://playcel.undistro.io/ (CEL playground)

Example

Blocking the use of the ‘latest’ image tag

prevent-latest-vap.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
 name: "prevent-latest-image-tag"
spec:
 failurePolicy: Fail
 matchConstraints:
   resourceRules:
   - apiGroups:   ["apps"]
     apiVersions: ["v1"]
     operations:  ["CREATE", "UPDATE"]
     resources:   ["deployments", "daemonsets", "statefulsets"]
 validations:
   - message: "The use of the 'latest' tag is not allowed"
     expression: |
       object.spec.template.spec.containers.all(container,
         !container.image.endsWith(":latest") && container.image.contains(":")
       ) &&
       (
         !has(object.spec.template.spec.initContainers) ||
         object.spec.template.spec.initContainers.all(container,
           !container.image.endsWith(":latest") && container.image.contains(":")
         )
       )
```

More examples:
github.com/AverageMarcus/common-admission-policies

Example

Blocking the use of the ‘latest’ image tag

prevent-latest-vap.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
 name: "prevent-latest-image-tag"
spec:
 failurePolicy: Fail
 matchConstraints:
   resourceRules:
   - apiGroups:   ["apps"]
     apiVersions: ["v1"]
     operations:  ["CREATE", "UPDATE"]
     resources:   ["deployments", "daemonsets", "statefulsets"]
 validations:
   - message: "The use of the 'latest' tag is not allowed"
     expression: |
       object.spec.template.spec.containers.all(container,
         !container.image.endsWith(":latest") && container.image.contains(":")
       ) &&
       (
         !has(object.spec.template.spec.initContainers) ||
         object.spec.template.spec.initContainers.all(container,
           !container.image.endsWith(":latest") && container.image.contains(":")
         )
       )
```

The name of our policy. We’ll reference this in our `ValidatingAdmissionPolicyBinding`

Example

Blocking the use of the ‘latest’ image tag

prevent-latest-vap.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
 name: "prevent-latest-image-tag"
spec:
 failurePolicy: Fail
 matchConstraints:
   resourceRules:
   - apiGroups:   ["apps"]
     apiVersions: ["v1"]
     operations:  ["CREATE", "UPDATE"]
     resources:   ["deployments", "daemonsets", "statefulsets"]
 validations:
   - message: "The use of the 'latest' tag is not allowed"
     expression: |
       object.spec.template.spec.containers.all(container,
         !container.image.endsWith(":latest") && container.image.contains(":")
       ) &&
       (
         !has(object.spec.template.spec.initContainers) ||
         object.spec.template.spec.initContainers.all(container,
           !container.image.endsWith(":latest") && container.image.contains(":")
         )
       )
```

We want our policy to block any incoming requests that don’t meet this policies requirements (default).

The alternative is `Ignore` if you want to disable enforcement of this policy.

Example

Blocking the use of the ‘latest’ image tag

prevent-latest-vap.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
 name: "prevent-latest-image-tag"
spec:
 failurePolicy: Fail
 matchConstraints:
   resourceRules:
   - apiGroups:   ["apps"]
     apiVersions: ["v1"]
     operations:  ["CREATE", "UPDATE"]
     resources:   ["deployments", "daemonsets", "statefulsets"]
 validations:
   - message: "The use of the 'latest' tag is not allowed"
     expression: |
       object.spec.template.spec.containers.all(container,
         !container.image.endsWith(":latest") && container.image.contains(":")
       ) &&
       (
         !has(object.spec.template.spec.initContainers) ||
         object.spec.template.spec.initContainers.all(container,
           !container.image.endsWith(":latest") && container.image.contains(":")
         )
       )
```

We define what resources this policy applies to.

Multiple resource types can be defined here but if they don’t have the same general API the CEL expression will quickly become very complex.

Example

Blocking the use of the ‘latest’ image tag

prevent-latest-vap.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
 name: "prevent-latest-image-tag"
spec:
 failurePolicy: Fail
 matchConstraints:
   resourceRules:
   - apiGroups:   ["apps"]
     apiVersions: ["v1"]
     operations:  ["CREATE", "UPDATE"]
     resources:   ["deployments", "daemonsets", "statefulsets"]
 validations:
   - message: "The use of the 'latest' tag is not allowed"
     expression: |
       object.spec.template.spec.containers.all(container,
         !container.image.endsWith(":latest") && container.image.contains(":")
       ) &&
       (
         !has(object.spec.template.spec.initContainers) ||
         object.spec.template.spec.initContainers.all(container,
           !container.image.endsWith(":latest") && container.image.contains(":")
         )
       )
```

We’ll add a human-friendly message to be shown when the API request has been blocked by this policy.

If we don’t include this, a generic message that include the whole expression is shown instead.

Example

Blocking the use of the ‘latest’ image tag

prevent-latest-vap.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
 name: "prevent-latest-image-tag"
spec:
 failurePolicy: Fail
 matchConstraints:
   resourceRules:
   - apiGroups:   ["apps"]
     apiVersions: ["v1"]
     operations:  ["CREATE", "UPDATE"]
     resources:   ["deployments", "daemonsets", "statefulsets"]
 validations:
   - message: "The use of the 'latest' tag is not allowed"
     expression: |
       object.spec.template.spec.containers.all(container,
         !container.image.endsWith(":latest") && container.image.contains(":")
       ) &&
       (
         !has(object.spec.template.spec.initContainers) ||
         object.spec.template.spec.initContainers.all(container,
           !container.image.endsWith(":latest") && container.image.contains(":")
         )
       )
```

Finally we have our expression.

This is the expression of ***allowed*** resources, not those to block.
(I keep getting caught out by this 😅)

Example

Blocking the use of the ‘latest’ image tag

prevent-latest-vap.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
 name: "prevent-latest-image-tag"
spec:
 failurePolicy: Fail
 matchConstraints:
   resourceRules:
   - apiGroups:   ["apps"]
     apiVersions: ["v1"]
     operations:  ["CREATE", "UPDATE"]
     resources:   ["deployments", "daemonsets", "statefulsets"]
 validations:
   - message: "The use of the 'latest' tag is not allowed"
     expression: |
       object.spec.template.spec.containers.all(container,
         !container.image.endsWith(":latest") && container.image.contains(":")
       ) &&
       (
         !has(object.spec.template.spec.initContainers) ||
         object.spec.template.spec.initContainers.all(container,
           !container.image.endsWith(":latest") && container.image.contains(":")
         )
       )
```

`object` is the incoming resource from the API call.

Example

Blocking the use of the ‘latest’ image tag

prevent-latest-vap.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
 name: "prevent-latest-image-tag"
spec:
 failurePolicy: Fail
 matchConstraints:
   resourceRules:
   - apiGroups:   ["apps"]
     apiVersions: ["v1"]
     operations:  ["CREATE", "UPDATE"]
     resources:   ["deployments", "daemonsets", "statefulsets"]
 validations:
   - message: "The use of the 'latest' tag is not allowed"
     expression: |
       object.spec.template.spec.containers.all(container,
         !container.image.endsWith(":latest") && container.image.contains(":")
       ) &&
       (
         !has(object.spec.template.spec.initContainers) ||
         object.spec.template.spec.initContainers.all(container,
           !container.image.endsWith(":latest") && container.image.contains(":")
         )
       )
```

First we check all containers don’t have the latest tag (or no tag specified).

Example

Blocking the use of the ‘latest’ image tag

prevent-latest-vap.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
 name: "prevent-latest-image-tag"
spec:
 failurePolicy: Fail
 matchConstraints:
   resourceRules:
   - apiGroups:   ["apps"]
     apiVersions: ["v1"]
     operations:  ["CREATE", "UPDATE"]
     resources:   ["deployments", "daemonsets", "statefulsets"]
 validations:
   - message: "The use of the 'latest' tag is not allowed"
     expression: |
       object.spec.template.spec.containers.all(container,
         !container.image.endsWith(":latest") && container.image.contains(":")
       ) &&
       (
         !has(object.spec.template.spec.initContainers) ||
         object.spec.template.spec.initContainers.all(container,
           !container.image.endsWith(":latest") && container.image.contains(":")
         )
       )
```

Then we check if this resource defines initContainers and if so we check they also don’t use latest.

Example

Blocking the use of the ‘latest’ image tag

Our policy does nothing until we bind it to some conditions.

prevent-latest-binding.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
    name: "prevent-latest-image-tag"
spec:
    policyName: "prevent-latest-image-tag"
    validationActions: [Deny]
    matchResources: {}
```

Example

Blocking the use of the ‘latest’ image tag

The name of the policy we have just created.

prevent-latest-binding.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
    name: "prevent-latest-image-tag"
spec:
    policyName: "prevent-latest-image-tag"
    validationActions: [Deny]
    matchResources: {}
```

Example

Blocking the use of the ‘latest’ image tag

prevent-latest-binding.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
    name: "prevent-latest-image-tag"
spec:
    policyName: "prevent-latest-image-tag"
    validationActions: [Deny]
    matchResources: {}
```

Note: this is an array as you can specify both warn and audit together

The action to take when a policy isn’t met.

Available options are: `Deny`, `Warn` & `Audit`

Example

Blocking the use of the ‘latest’ image tag

We’re not defining any filtering so this policy applies cluster-wide.

We could limit our policy to specific namespaces or labels, for example.

prevent-latest-binding.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
    name: "prevent-latest-image-tag"
spec:
    policyName: "prevent-latest-image-tag"
    validationActions: [Deny]
    matchResources: {}
```

Example

Blocking the use of the ‘latest’ image tag

We’re not defining any filtering so this policy applies cluster-wide.

We could limit our policy to specific namespaces or labels, for example.

prevent-latest-binding.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
    name: "prevent-latest-image-tag"
spec:
    policyName: "prevent-latest-image-tag"
    validationActions: [Deny]
    matchResources:
       namespaceSelector:
           matchLabels:
              environment: prod
```

Example

Blocking the use of the ‘latest’ image tag

deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-blocked
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
```

```
# kubectl apply -f deployment.yaml

The deployments "nginx-blocked" is invalid: : ValidatingAdmissionPolicy
'prevent-latest-image-tag' with binding 'prevent-latest-image-tag' denied
request: The use of the 'latest' tag is not allowed
```

Our friendly message ↗

Blocked!

More advanced features

More advanced features

- **More context values** - Along with `object` you also have `oldObject`, `namespaceObject` & `request` you can use in your expressions

```
matchConditions:
  - name: "exclude-kubelet-requests"
    expression: "!("system:nodes" in request.userInfo.groups)"
```

More advanced features

- **More context values** - Along with `object` you also have `oldObject`, `namespaceObject` & `request` you can use in your expressions
- **Parameters** - make your policies configurable by allowing parameter resources to be applied by the binding

policy.yaml

```yaml
apiVersion: admissionregistration.k8s…
kind: ValidatingAdmissionPolicy
metadata:
  name: "param-example"
spec:
  paramKind:
    apiVersion: rules.example.com/v1
    kind: ReplicaLimit
  validations:
  - expression: object.spec.replicas <=
params.maxReplicas
```

policy-binding.yaml

```yaml
apiVersion: admissionregistration.k8s…
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: "param-example"
spec:
  policyName: "param-example"
  paramRef:
    name: "my-parameters"
    namespace: "default"
```

params.yaml

```yaml
apiVersion: rules.example.com/v1
kind: ReplicaLimit
metadata:
  name: "my-parameters"
  namespace: "default"
maxReplicas: 3
```

More advanced features

- **More context values** - Along with `object` you also have `oldObject`, `namespaceObject` & `request` you can use in your expressions
- **Parameters** - make your policies configurable by allowing parameter resources to be applied by the binding
- **Variables** - reusable CEL expressions to simplify your validation expressions.

```
spec:
 variables:
  - name: teamLabel
    expression: "'team' in object.spec.metadata.labels ? object.spec.metadata.labels['team'] : 'no-team'"
 validations:
  - expression: variables.teamLabel == 'rel-eng'
```

Must be a valid CEL identifier (e.g. no `-`)

More advanced features

- **More context values** - Along with `object` you also have `oldObject`, `namespaceObject` & `request` you can use in your expressions
- **Parameters** - make your policies configurable by allowing parameter resources to be applied by the binding
- **Variables** - reusable CEL expressions to simplify your validation expressions.
- **Audit annotations** - Add extra metadata to the audit logs, dynamic values using CEL

```
auditAnnotations:
 - key: "replica-count"
   valueExpression: |
    "'Deployment spec.replicas set to ' + string(object.spec.replicas)"
```

```
{
    "kind": "Event",
    "apiVersion": "audit.k8s.io/v1",
    "annotations": {
        "demo-policy.example.com/replica-count":
"Deployment spec.replicas set to 128"
        ...
    }
}
```

More advanced features

- **More context values** - Along with `object` you also have `oldObject`, `namespaceObject` & `request` you can use in your expressions
- **Parameters** - make your policies configurable by allowing parameter resources to be applied by the binding
- **Variables** - reusable CEL expressions to simplify your validation expressions.
- **Audit annotations** - Add extra metadata to the audit logs, dynamic values using CEL
- **Message expressions** - Leverage some CEL in your message to have dynamic validation messages

```
messageExpression: "'object.spec.replicas must be no greater than ' + string(params.maxReplicas)"
```

So that’s validation covered, what about mutating?

Well…

MutatingAdmissionPolicies

Status: Alpha in v1.32

Introduced in: KEP-3962

- Follows the same idea as ValidatingAdmissionPolicies.
- Introduces `MutatingAdmissionPolicy` and `MutatingAdmissionPolicyBinding`
- Supports two patch strategies: `ApplyConfiguration` and `JSONPatch`
- Not yet enabled by default, you must enable the following:
  - `MutatingAdmissionPolicy` feature gate
  - `admissionregistration.k8s.io/v1alpha1` API

kind-config.yaml

```yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
name: kind-cluster
featureGates:
 "MutatingAdmissionPolicy": true
runtimeConfig:
 "admissionregistration.k8s.io/v1alpha1": true
```

Create a Kind cluster with:

`kind create cluster --config kind-config.yaml`

Example

Injecting proxy values as env vars

proxy-vars-map.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1alpha1
kind: MutatingAdmissionPolicy
metadata:
    name: "proxy-values"
spec:
    failurePolicy: Fail
    reinvocationPolicy: IfNeeded
    matchConstraints:
        resourceRules:
            - apiGroups:   [""]
              apiVersions: ["v1"]
              operations:  ["CREATE"]
              resources:   ["pods"]
   mutations:
     - patchType: "ApplyConfiguration"
       applyConfiguration: >
           expression: >
             Object{
               spec: Object.spec{
                   containers: object.spec.containers.map(c,
                       Object.spec.containers.item{
                           name: c.name,
                           env: [
                             Object.spec.containers.env{
                               name: "HTTP_PROXY",
                               value: "http://proxy.proxy.svc:3128"
                             }
                           ]
                       }
                   )
               }
             }
```

Example

Injecting proxy values as env vars

Same as ValidatingAdmissionPolicies

proxy-vars-map.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1alpha1
kind: MutatingAdmissionPolicy
metadata:
    name: "proxy-values"
spec:
    failurePolicy: Fail
    reinvocationPolicy: IfNeeded
    matchConstraints:
        resourceRules:
            - apiGroups:   [""]
              apiVersions: ["v1"]
              operations:  ["CREATE"]
              resources:   ["pods"]
   mutations:
     - patchType: "ApplyConfiguration"
       applyConfiguration: >
           expression: >
             Object{
               spec: Object.spec{
                   containers: object.spec.containers.map(c,
                       Object.spec.containers.item{
                           name: c.name,
                           env: [
                             Object.spec.containers.env{
                               name: "HTTP_PROXY",
                               value: "http://proxy.proxy.svc:3128"
                             }
                           ]
                       }
                   )
               }
             }
```

Example

Injecting proxy values as env vars

Should the policy be re-applied if other mutations are made to the resource?

Allowed values are "Never" and "IfNeeded"

proxy-vars-map.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1alpha1
kind: MutatingAdmissionPolicy
metadata:
    name: "proxy-values"
spec:
    failurePolicy: Fail
    reinvocationPolicy: IfNeeded
    matchConstraints:
        resourceRules:
            - apiGroups:   [""]
              apiVersions: ["v1"]
              operations:  ["CREATE"]
              resources:   ["pods"]
   mutations:
     - patchType: "ApplyConfiguration"
       applyConfiguration: >
           expression: >
             Object{
               spec: Object.spec{
                   containers: object.spec.containers.map(c,
                       Object.spec.containers.item{
                           name: c.name,
                           env: [
                             Object.spec.containers.env{
                               name: "HTTP_PROXY",
                               value: "http://proxy.proxy.svc:3128"
                             }
                           ]
                       }
                   )
               }
             }
```

Example

Injecting proxy values as env vars

Replace `validations` with `mutations`

proxy-vars-map.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1alpha1
kind: MutatingAdmissionPolicy
metadata:
    name: "proxy-values"
spec:
    failurePolicy: Fail
    reinvocationPolicy: IfNeeded
    matchConstraints:
        resourceRules:
            - apiGroups:   [""]
              apiVersions: ["v1"]
              operations:  ["CREATE"]
              resources:   ["pods"]
    mutations:
     - patchType: "ApplyConfiguration"
       applyConfiguration: >
           expression: >
             Object{
               spec: Object.spec{
                   containers: object.spec.containers.map(c,
                       Object.spec.containers.item{
                           name: c.name,
                           env: [
                             Object.spec.containers.env{
                               name: "HTTP_PROXY",
                               value: "http://proxy.proxy.svc:3128"
                             }
                           ]
                       }
                   )
               }
             }
```

Example

Injecting proxy values as env vars

Indicates the patch strategy
Possible values:

- `ApplyConfiguration`
- `JSONPatch`

proxy-vars-map.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1alpha1
kind: MutatingAdmissionPolicy
metadata:
    name: "proxy-values"
spec:
    failurePolicy: Fail
    reinvocationPolicy: IfNeeded
    matchConstraints:
        resourceRules:
            - apiGroups:   [""]
              apiVersions: ["v1"]
              operations:  ["CREATE"]
              resources:   ["pods"]
   mutations:
     - patchType: "ApplyConfiguration"
       applyConfiguration: >
           expression: >
             Object{
               spec: Object.spec{
                   containers: object.spec.containers.map(c,
                       Object.spec.containers.item{
                           name: c.name,
                           env: [
                             Object.spec.containers.env{
                               name: "HTTP_PROXY",
                               value: "http://proxy.proxy.svc:3128"
                             }
                           ]
                       }
                   )
               }
             }
```

Example

Injecting proxy values as env vars

Either `applyConfiguration` or `jsonPatch` depending on the patchType.

proxy-vars-map.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1alpha1
kind: MutatingAdmissionPolicy
metadata:
    name: "proxy-values"
spec:
    failurePolicy: Fail
    reinvocationPolicy: IfNeeded
    matchConstraints:
        resourceRules:
            - apiGroups:   [""]
              apiVersions: ["v1"]
              operations:  ["CREATE"]
              resources:   ["pods"]
   mutations:
     - patchType: "ApplyConfiguration"
       applyConfiguration: >
           expression: >
             Object{
               spec: Object.spec{
                   containers: object.spec.containers.map(c,
                       Object.spec.containers.item{
                           name: c.name,
                           env: [
                             Object.spec.containers.env{
                               name: "HTTP_PROXY",
                               value: "http://proxy.proxy.svc:3128"
                             }
                           ]
                       }
                   )
               }
             }
```

Example

Injecting proxy values as env vars

Introduction of named types

The Object refers to the type of the incoming resource (Pod in this example).

Child fields can be used by using `Object.[fieldname]` and can go multiple levels down as see here with the `Object.spec.containers.env`

proxy-vars-map.yaml

```
apiVersion: admissionregistration.k8s.io/v1alpha1
kind: MutatingAdmissionPolicy
metadata:
    name: "proxy-values"
spec:
    failurePolicy: Fail
    reinvocationPolicy: IfNeeded
    matchConstraints:
        resourceRules:
            - apiGroups:   [""]
              apiVersions: ["v1"]
              operations:  ["CREATE"]
              resources:   ["pods"]
   mutations:
     - patchType: "ApplyConfiguration"
       applyConfiguration: >
          expression: >
            Object{
              spec: Object.spec{
                  containers: object.spec.containers.map(c,
                      Object.spec.containers.item{
                          name: c.name,
                          env: [
                            Object.spec.containers.env{
                              name: "HTTP_PROXY",
                              value: "http://proxy.proxy.svc:3128"
                            }
                          ]
                      }
                  )
              }
            }
```

Example

Injecting proxy values as env vars

Merge our proxy vars with any existing vars in all containers.

As this is an ApplyConfiguration patch type the existing env vars on the object remain untouched.

Note: We're missing `initContainers` and `ephermeralContainers` here due to limited space.

proxy-vars-map.yaml

```
apiVersion: admissionregistration.k8s.io/v1alpha1
kind: MutatingAdmissionPolicy
metadata:
    name: "proxy-values"
spec:
    failurePolicy: Fail
    reinvocationPolicy: IfNeeded
    matchConstraints:
        resourceRules:
            - apiGroups:   [""]
              apiVersions: ["v1"]
              operations:  ["CREATE"]
              resources:   ["pods"]
   mutations:
     - patchType: "ApplyConfiguration"
       applyConfiguration: >
           expression: >
             Object{
               spec: Object.spec{
                   containers: object.spec.containers.map(c,
                       Object.spec.containers.item{
                           name: c.name,
                           env: [
                             Object.spec.containers.env{
                               name: "HTTP_PROXY",
                               value: "http://proxy.proxy.svc:3128"
                             }
                           ]
                       }
                   )
               }
             }
```

Example

Injecting proxy values as env vars

proxy-vars-binding.yaml

```yaml
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingAdmissionPolicyBinding
metadata:
    name: "proxy-values"
spec:
    policyName: "proxy-values"
    matchResources: {}
    paramRef: {}
```

Nothing too surprising about our binding resource. All properties are the same as Validating except for the lack of a `validationActions` property.

Advanced features

- **More context values** - Along with `object` you also have `oldObject`, `namespaceObject` & `request` you can use in your expressions
- **Parameters** - make your policies configurable by allowing parameter resources to be applied by the binding
- **Variables** - reusable CEL expressions to simplify your validation expressions.
- ~~**Audit annotations** - Add extra metadata to the audit logs, dynamic values using CEL~~
- ~~**Message expressions** - Leverage some CEL in your message to have dynamic validation messages~~

What about the future beyond that?

* All these are speculative / wishful thinking right now

- Generative policies - create new resources based on API requests
- Resource lookup - get the current state of other resources within the cluster
- Policy exceptions - disable policies under certain circumstances
- Policy reports - user friendly view of validation failures, etc.
- More abstractions - e.g. Kyverno already working on this with VAP

🐘 @Marcus@k8s.social | 🌐 MarcusNoble.com | 🦋 @averagemarcus.bsky.social

Summary

- Time to start replacing those risky webhooks with in-process policies.
- ValidatingAdmissionPolicies generally available for use from **v1.30** for safe validation logic.
- **v1.32** for *alpha release* of MutatingAdmissionPolicies. (Must be enabled)
- More abstractions, generative policies and API lookups hopefully coming in the future.
- Examples of common use-case policies:

  github.com/AverageMarcus/common-admission-policies

  Contributions welcome ↗

Wrap-up

Slides and resources available at:

https://go-get.link/cndoslo

Thoughts, comments and feedback:

feedback@marcusnoble.co.uk

https://k8s.social/@Marcus

https://bsky.app/profile/averagemarcus.bsky.social

Thank you