The Wonders and Woes of Webhooks

Webinar

16 / 02 / 2023

Hi 👋,

I’m **Marcus Noble,** a *platform engineer* at ***Giant Swarm***

I’m found around the web as ✨***AverageMarcus***✨ in most places and **@Marcus@k8s.social** on Mastodon

~5 years experience running Kubernetes in production environments.

💙

My Relationship with Webhooks

- *a story in 3 acts*

Act #1

Introduction, backstory and the ✨wonders✨

Act #2

The conflicts, struggles and woes 😨

Act #3

The resolution and the future 🔮

Act #1

Webhooks in Kubernetes

Kubernetes has three main types of webhooks:

- `ValidatingWebhookConfiguration` - Introduced in **v1.9** Beta (replacing `GenericAdmissionWebhook` introduced in v1.7)
- `MutatingWebhookConfiguration` - Introduced in **v1.9** Beta
- `CustomResourceConversion` - Introduced in **v1.13**

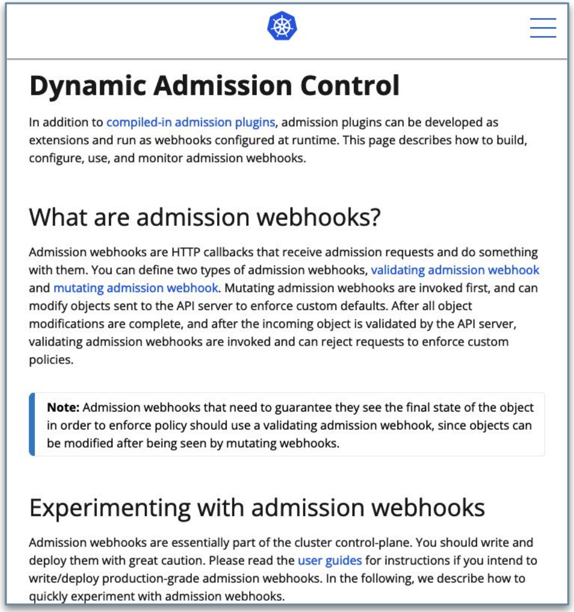
We're going to focus on the ***first two*** and ignore the `CustomResourceConversion` for the purpose of this talk.

Dynamic Admission Control

- Both Validating and Mutating admission webhooks come under the responsibility of the *Dynamic Admission* controller within apiserver.
- Can be triggered by (almost) all API operations against (almost) all Kubernetes resources.

CREATE, UPDATE, DELETE & CONNECT

- Part of the `admissionregistration.k8s.io/v1` API.
- Currently enabled by default by the default value of the `--enable-admission-plugins` apiserver flag.



Dynamic Admission Control

In addition to compiled-in admission plugins, admission plugins can be developed as extensions and run as webhooks configured at runtime. This page describes how to build, configure, use, and monitor admission webhooks.

What are admission webhooks?

Admission webhooks are HTTP callbacks that receive admission requests and do something with them. You can define two types of admission webhooks, validating admission webhook and mutating admission webhook. Mutating admission webhooks are invoked first, and can modify objects sent to the API server to enforce custom defaults. After all object modifications are complete, and after the incoming object is validated by the API server, validating admission webhooks are invoked and can reject requests to enforce custom policies.

> **Note:** Admission webhooks that need to guarantee they see the final state of the object in order to enforce policy should use a validating admission webhook, since objects can be modified after being seen by mutating webhooks.

Experimenting with admission webhooks

Admission webhooks are essentially part of the cluster control-plane. You should write and deploy them with great caution. Please read the user guides for instructions if you intend to write/deploy production-grade admission webhooks. In the following, we describe how to quickly experiment with admission webhooks.

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

Purpose / Use Cases

Defaulting	Policy Enforcement
Best Practices	Problem Mitigation

Defaulting

- Adding `imagePullSecrets` when images from private registries are used
- Generating the image registry secret when new namespaces are created

In the past

- Injecting a sidecar into pods (e.g. Istio)
- Setting default resource limits when not set (alternative to `LimitRange`)
- Inject proxy env vars into pods - e.g. `HTTP_PROXY`, `NO_PROXY`

Policy Enforcement

- Prevent using `latest` image tag or enforce the use of a SHA image tag
- Require resource limits to be set on all pods
- Block large container images (e.g. don’t pull container images >1Gb)
- Prevent use of deprecated Kubernetes APIs (e.g. `batch/v1beta1`)
- Block use of `hostPath`
- Replace old PSP functionality not supported by the new Pod Security Admission

Best Practices

- Enforce standard labels / annotations on all resources
- Require pod probes be set
- Restrict allowed namespaces
- Require a `PodDisruptionBudget` to be set
- Replace all pods image registries with an in-house image proxy / cache.

Problem Mitigation

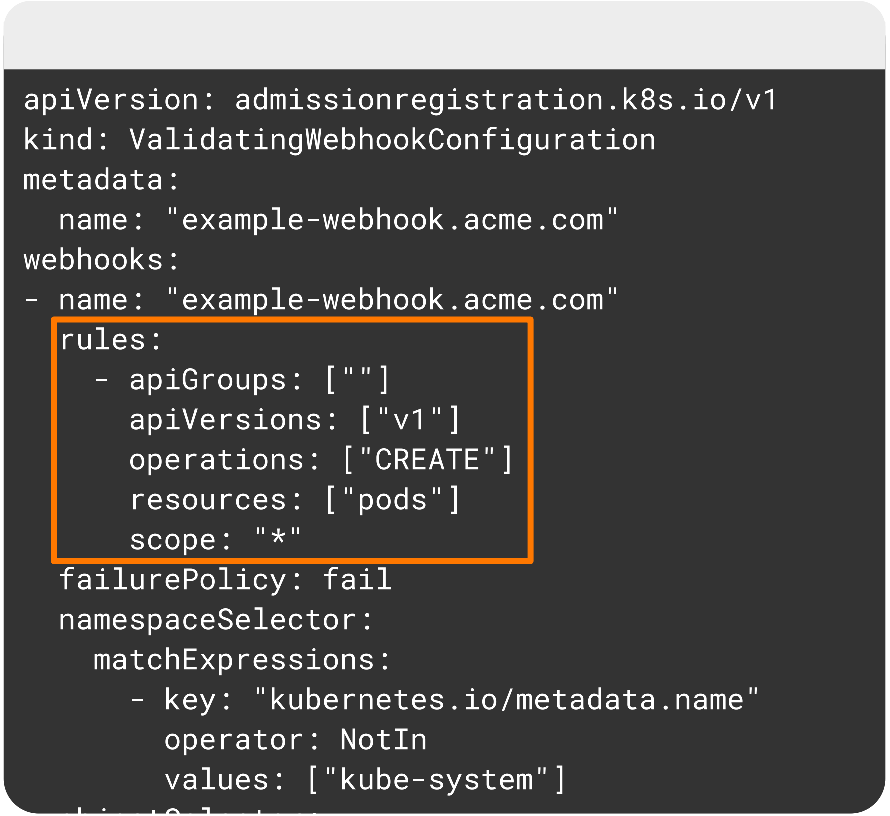
- Block nodes joining the cluster with known CVEs based on the kernel version (e.g. CVE-2022-0185)
- Prevent custom nginx snippets from being used (CVE-2021-25742)
- Inject Log4Shell mitigation env var, `LOG4J_FORMAT_MSG_NO_LOOKUPS`, into all pods (CVE-2021-44228)
- Block binding to the cluster-admin role
- Disallow privilege escalation

Example Webhook

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: "example-webhook.acme.com"
webhooks:
- name: "example-webhook.acme.com"
  rules:
    - apiGroups: [""]
      apiVersions: ["v1"]
      operations: ["CREATE"]
      resources: ["pods"]
      scope: "*"
  failurePolicy: fail
  namespaceSelector:
    matchExpressions:
      - key: "kubernetes.io/metadata.name"
        operator: NotIn
        values: ["kube-system"]
```

```yaml
  objectSelector:
    matchLabels:
      app.kubernetes.io/owned-by: my-team
  clientConfig:
    service:
      namespace: default
      name: example-webhook
      path: /validate-pods
      port: 443
```

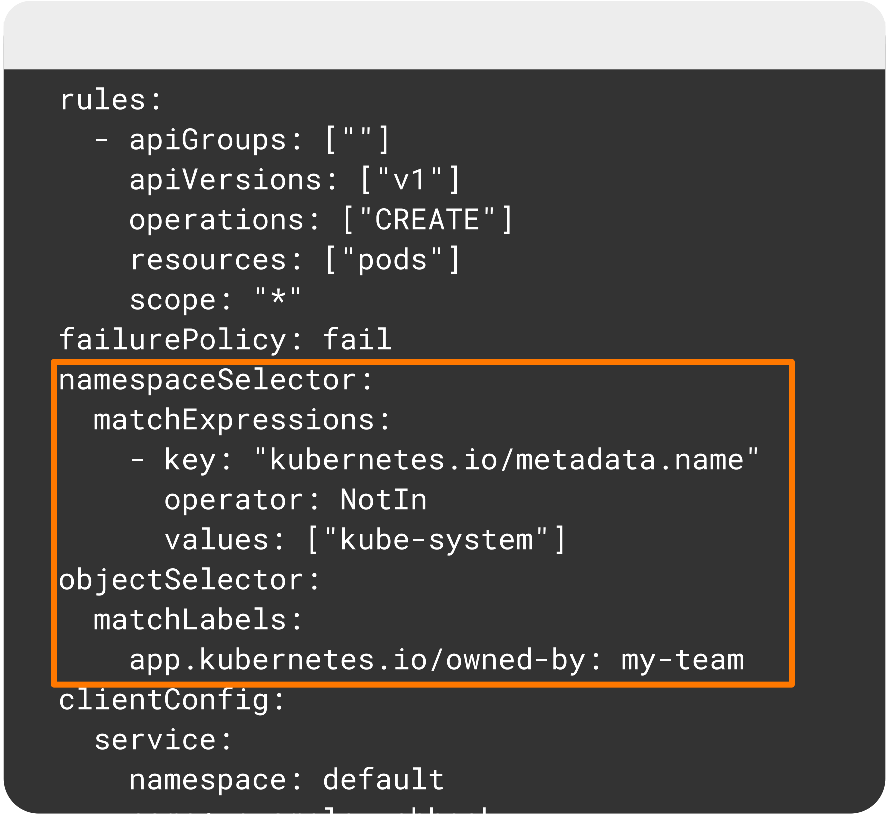
Example Webhook



```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: "example-webhook.acme.com"
webhooks:
- name: "example-webhook.acme.com"
  rules:
    - apiGroups: [""]
      apiVersions: ["v1"]
      operations: ["CREATE"]
      resources: ["pods"]
      scope: "*"
  failurePolicy: fail
  namespaceSelector:
    matchExpressions:
      - key: "kubernetes.io/metadata.name"
        operator: NotIn
        values: ["kube-system"]
  objectSelector:
```

For every resource created / modified / deleted in the cluster the Kubernetes apiserver checks for webhook configurations with a matching rule.

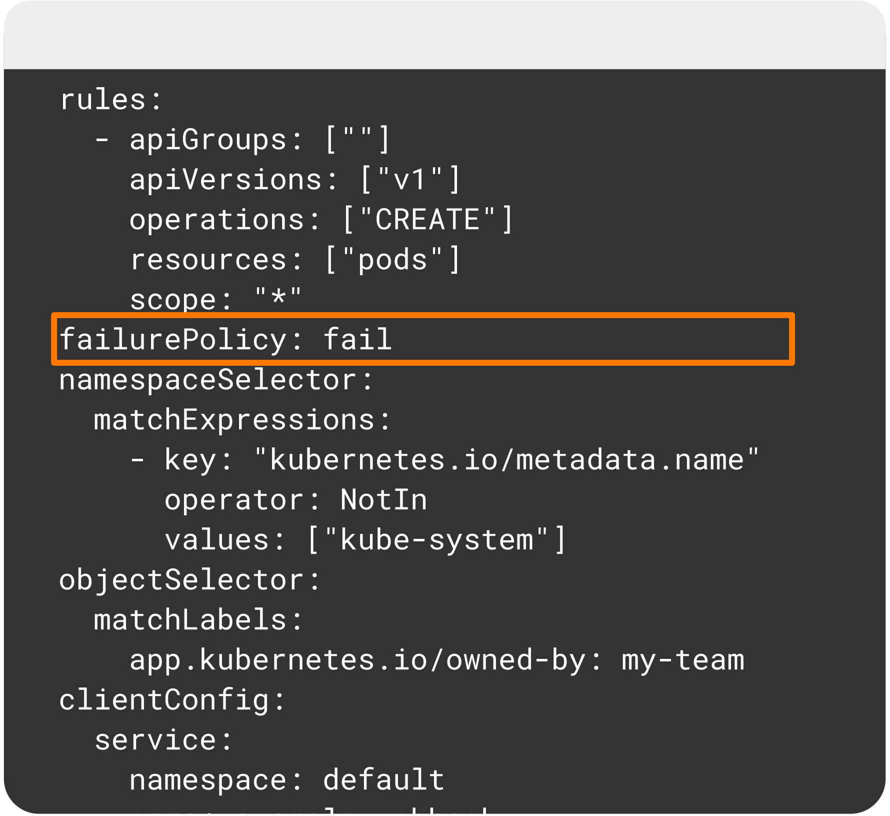
Example Webhook



```
rules:
  - apiGroups: [""]
    apiVersions: ["v1"]
    operations: ["CREATE"]
    resources: ["pods"]
    scope: "*"
failurePolicy: fail
namespaceSelector:
  matchExpressions:
    - key: "kubernetes.io/metadata.name"
      operator: NotIn
      values: ["kube-system"]
objectSelector:
  matchLabels:
    app.kubernetes.io/owned-by: my-team
clientConfig:
  service:
    namespace: default
```

The namespaceSelector and objectSelector are used to further filter what a webhook should apply to.

Example Webhook

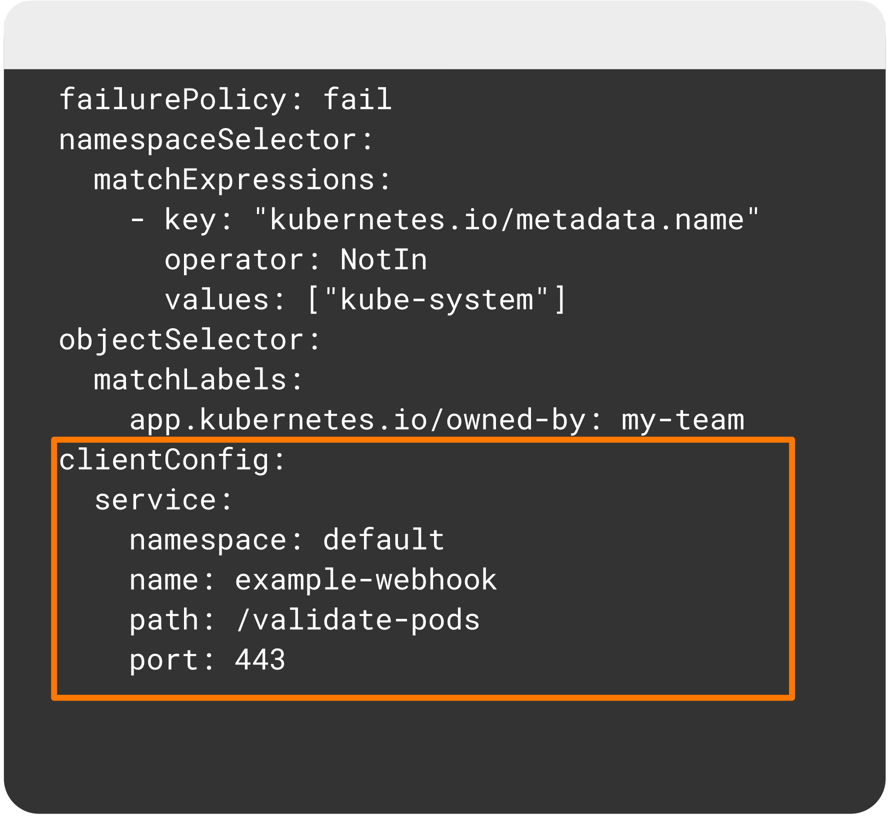


```
rules:
  - apiGroups: [""]
    apiVersions: ["v1"]
    operations: ["CREATE"]
    resources: ["pods"]
    scope: "*"
failurePolicy: fail
namespaceSelector:
  matchExpressions:
    - key: "kubernetes.io/metadata.name"
      operator: NotIn
      values: ["kube-system"]
objectSelector:
  matchLabels:
    app.kubernetes.io/owned-by: my-team
clientConfig:
  service:
    namespace: default
```

The failurePolicy property indicates how unexpected errors are handled. Valid options are `fail` and `ignore` with `fail` being the default.

Giant Swarm

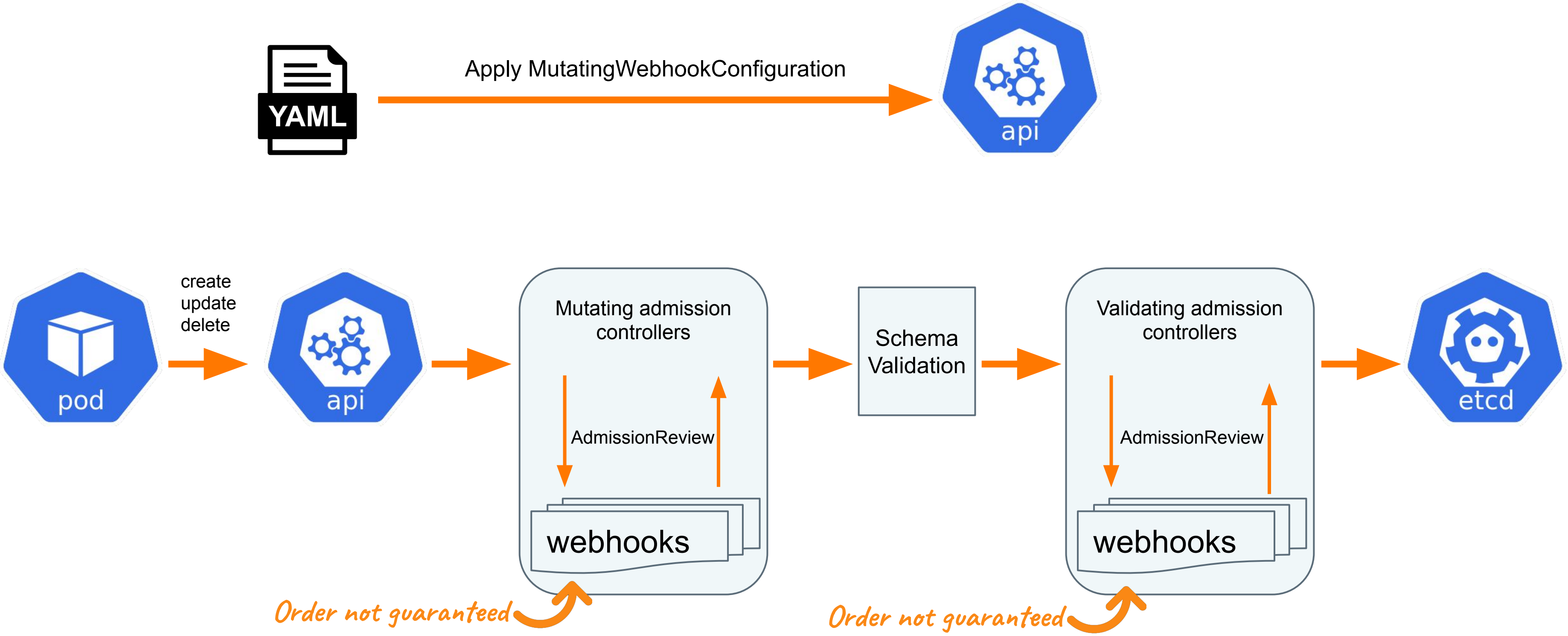
Example Webhook



```
failurePolicy: fail
namespaceSelector:
  matchExpressions:
    - key: "kubernetes.io/metadata.name"
      operator: NotIn
      values: ["kube-system"]
objectSelector:
  matchLabels:
    app.kubernetes.io/owned-by: my-team
clientConfig:
  service:
    namespace: default
    name: example-webhook
    path: /validate-pods
    port: 443
```

clientConfig describes what endpoint the webhook should be called against.

Example API request



YAML

Apply MutatingWebhookConfiguration

api

pod

create
update
delete

api

Mutating admission controllers

AdmissionReview

webhooks

Schema Validation

Validating admission controllers

AdmissionReview

webhooks

etcd

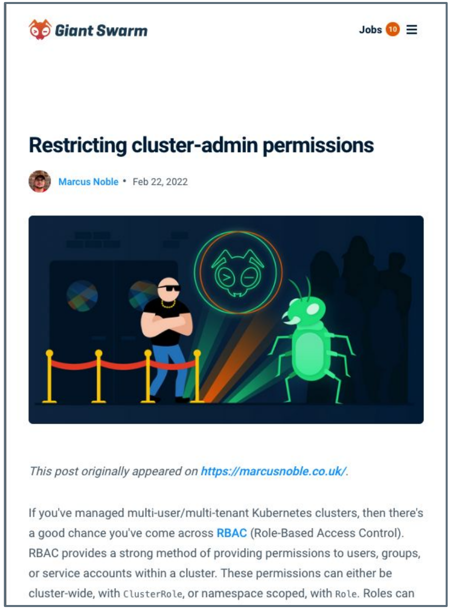
Order not guaranteed

Order not guaranteed

Salman Iqbal did a great ignite talk covering this at DevOpsDays London

Wonders in the Wild

Examples of webhooks solving real problems



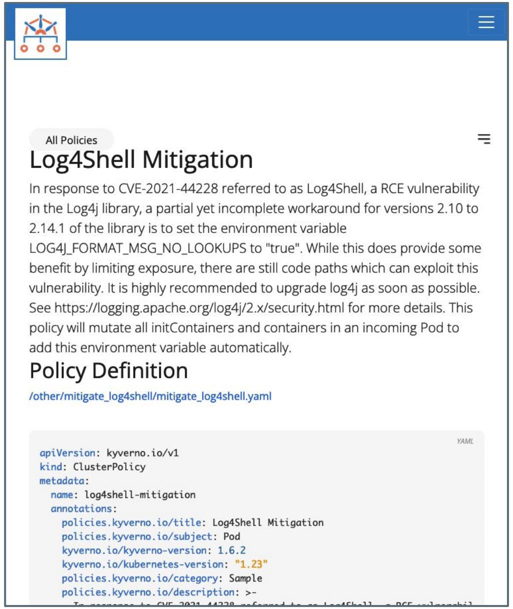
Restricting cluster-admin permissions

Marcus Noble • Feb 22, 2022

This post originally appeared on https://marcusnoble.co.uk/.

If you've managed multi-user/multi-tenant Kubernetes clusters, then there's a good chance you've come across RBAC (Role-Based Access Control). RBAC provides a strong method of providing permissions to users, groups, or service accounts within a cluster. These permissions can either be cluster-wide, with `ClusterRole`, or namespace scoped, with `Role`. Roles can

Leveraging validating webhooks to restrict the cluster-admin beyond what was possible by RBAC to block **a bug in our CLI tool**.



Log4Shell Mitigation

In response to CVE-2021-44228 referred to as Log4Shell, a RCE vulnerability in the Log4j library, a partial yet incomplete workaround for versions 2.10 to 2.14.1 of the library is to set the environment variable LOG4J_FORMAT_MSG_NO_LOOKUPS to "true". While this does provide some benefit by limiting exposure, there are still code paths which can exploit this vulnerability. It is highly recommended to upgrade log4j as soon as possible. See https://logging.apache.org/log4j/2.x/security.html for more details. This policy will mutate all initContainers and containers in an incoming Pod to add this environment variable automatically.

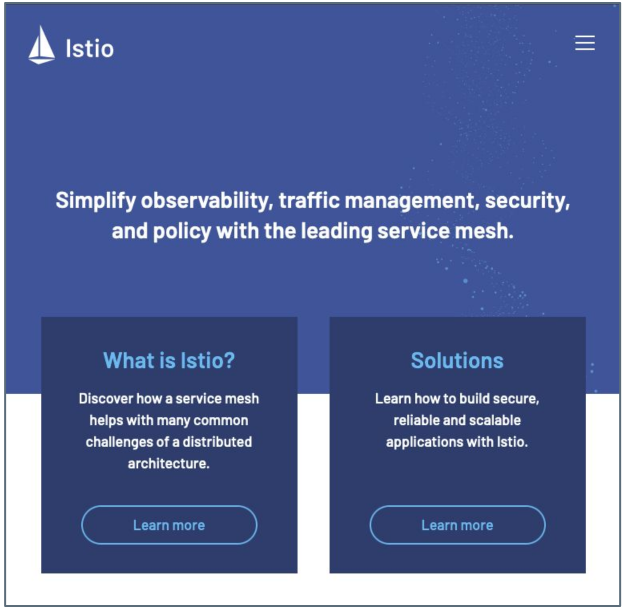
Policy Definition

/other/mitigate_log4shell/mitigate_log4shell.yaml

YAML

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: log4shell-mitigation
  annotations:
    policies.kyverno.io/title: Log4Shell Mitigation
    policies.kyverno.io/subject: Pod
    kyverno.io/kyverno-version: 1.6.2
    kyverno.io/kubernetes-version: "1.23"
    policies.kyverno.io/category: Sample
    policies.kyverno.io/description: >-
      In response to CVE-2021-44228 referred to as Log4Shell, a RCE vulnerabil
```

Mitigate the **Log4Shell vulnerability** cluster-wide by injecting an env var into all pods that disables the vulnerable code path, made possible by a mutating webhook.



Simplify observability, traffic management, security, and policy with the leading service mesh.

What is Istio?

Discover how a service mesh helps with many common challenges of a distributed architecture.

Solutions

Learn how to build secure, reliable and scalable applications with Istio.

Much of Istio's power in the earlier days came from its ability to have its own container running in every pod in the cluster as **a "sidecar"**, made possible with a mutating webhook.

Alternatives

Initializers

- Introduced in v1.7 providing a way of configuring out-of-tree code that can modify resources before they actually created.
- Each initializer relies on an operator performing a list/watch to catch resources that need to be processed.
- The apiserver adds pending initializers to objectMeta but that's all it handles.

Pod Presets

- Introduced in v1.6.
- Inject defaults into Pods at creation if a matching field isn't already set.
- Namespace scoped.

~~Alternatives~~

~~Initializers~~ - Removed in v1.16

- Introduced in v1.7 providing a way of configuring out-of-tree code that can modify resources before they actually created.
- Each initializer relies on an operator performing a list/watch to catch resources that need to be processed.
- The apiserver adds pending initializers to objectMeta but that's all it handles.

~~Pod Presets~~ - Removed in v1.20

- Introduced in v1.6.
- Inject defaults into Pods at creation if a matching field isn't already set.
- Namespace scoped.

Act #2

The woes

What follows next are incidents where webhooks have caused clusters to break, to varying degrees of severity, for myself, my team or others.

I mention specific tools for *context only* and **not to call any out for being at fault**.

The fault in all of these is the fragility of webhooks within Kubernetes and the lengths that must be taken to ensure some amount of resilience.

Incident #1 - Kyverno and the faulty AZ

Kyverno is a fantastic tool that makes it very easy to create policies to be applied to almost everything in a cluster. It does this by creating wide-catching Validating/Mutating webhooks.

Many of the policies are security-related (replacing old PSP functionality) and as such has a `failurePolicy` of **Fail**.

For resilience, the service behind the webhooks runs with at least 2 replicas and has some logic to de-register the webhook when the last replica is removed from the cluster. Pod anti-affinity is in place to ensure the replicas are scheduled onto different nodes.

Incident #1 - Kyverno and the faulty AZ

1. By chance, both pods were **scheduled onto nodes within the same Failure Domain**.
2. Something happened that **caused that failure domain to fail**. This could be an issue with the cloud provider, a manual error accidentally deleting an ASG or maybe some routing changes that left that subnet inaccessible.
3. Both Kyverno pods are suddenly **missing from the cluster**. The scheduler does its job and goes to **schedule two new pods**.
4. The apiserver receives the API call to create the new pods, checks the list of MutatingWebhookConfigurations and **sees the entry for the Kyverno webhook**.
5. A webhook request is made to the Kyverno service in the cluster but as no pods are running it returns an error and **blocks the new pod creation**.

Impact = Cluster at-risk. Autoscaling up not working. Recreating broken pods not possible.

Incident #2 - Cluster upgrade

Our cluster has several Mutating and Validating webhooks in place, many of them targeting Pods.

Some of the services behind the webhooks includes, but is not limited to, cert-manager, Instana, Kyverno and Linkerd.

Most were installed using 3rd party Helm charts with their default values.

Incident #2 - Cluster upgrade

1. An upgrade of the cluster to the latest Kubernetes version is triggered. The cluster has plenty of spare capacity so a strategy of **removing 25% of nodes** at a time is used.
2. The upgrade is performed by making changes to the **AWS Launch Template** used by the nodes and then an **Instance Refresh** is performed on the ASG.
3. The initial 25% of nodes includes 1 control plane and 2 worker nodes.
4. When the 3 new nodes are launched, they are **unable to schedule any pods** (including any for the control plane). Logs for controller-manager taken from the host node include several instances of `Internal error occurred: failed calling webhook`.
5. The instances in AWS were **reporting as running** so the Instance Refresh **continues cycling the rest of the cluster**.

Impact = Cluster completely taken down if not caught early enough! 😱

Incident #3 - Jetstack - OPA takes down GKE cluster

The following incident comes from Jetstack (source: https://blog.jetstack.io/blog/gke-webhook-outage/) and I have picked out the highlights to include here:

> We were in the process of upgrading the control plane for a development cluster used by many teams to test their apps during the working day.
>
> We began the upgrade via our GKE Terraform pipeline. When performing the control plane upgrade the operation did not complete before the Terraform timeout (which we had set to 20 minutes). This was the first sign that something was wrong though the cluster was still showing as upgrading in the GKE console.

Incident #3 - Jetstack - OPA takes down GKE cluster

1. GKE completed the upgrade of one control plane instance, and started to **receive all API server traffic** as the following control plane instance were upgraded.
2. During the upgrade of the second control plane instance, the **API server was unable to run [PostStartHook](#)** for [ca-registration](#).
3. While running this hook, the API server **attempted to update a ConfigMap** in kube-system. This operation timed out as the backend for the validating webhook, Open Policy Agent (OPA), **was not responding**.
4. This operation must complete for a control plane node to pass a health check, because it continuously failed the second control plane entered a crash loop and halted the upgrade.
5. Kubelet unable to report node health.
6. GKE node auto-repair continually recreated the nodes.

Impact = Intermittent API downtime.

Incident #3 - Scale-to-zero

A non-production cluster uses cluster-autoscaler to scale down worker nodes to 0 outside of working hours to save on costs.

The control plane nodes remain (either as a single node or a HA cluster of 3).

Cluster-autoscaler is set to evict DaemonSets and a daily CronJob is run to scale down all Deployments to 0 replicas (and back up again in the morning).

The CronJob has a toleration for control plane nodes to ensure it can run again in the morning with no workers.

Incident #3 - Scale-to-zero

1. For weeks the cluster scaling operated *as expected*, scaling to 0 and back up based on the CronJob.
2. A team member deploys a new application that includes a `ValidatingWebhookConfiguration` with a `failurePolicy` set to **Fail**.
3. The next time to CronJob runs, the cluster scales down all worker nodes, terminating all pods of the newly installed application.
4. The following morning **no worker nodes** are created and all deployments are still set to **0 replicas**.

Impact = No worker nodes and no deployments running.

Act #3

Lessons Learned

- **All** webhook services should have *at least* 2 replicas, a PodDisruptionBudget, anti-affinity ensuring the pods end up in different failure domains and health probes in place.
- Where possible, ensure that `namespaceSelector` is set to **ignore kube-system**.
- Where possible, make use of `objectSelector` to only **target what is required**.
- Be careful when cycling nodes and not relying on the cloud providers health checks alone.
- Avoid cluster-autoscaler scale-to-0 when using webhooks without a `failurePolicy` set to ignore.

Regardless of the "importance" of the functionality the app provides

So what can we,
as **cluster operators**,
do to avoid this?

So what can we, as **cluster operators**, do to avoid this?

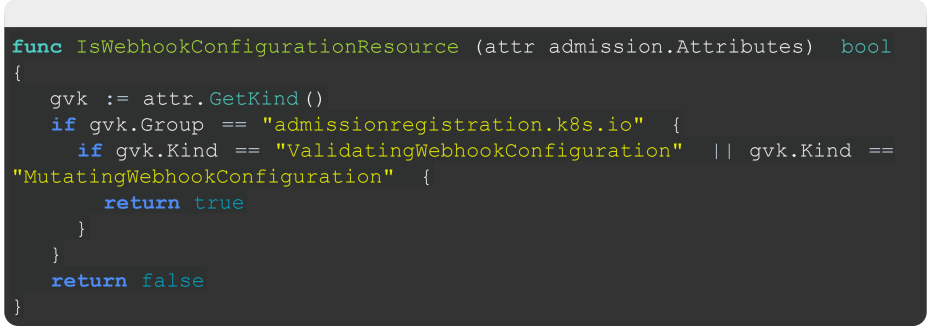
Unfortunately not a whole lot. 😞

A webhook to enforce resilient webhooks

A webhook to enforce resilient webhooks

NOPE!

It’s not possible to have webhooks with rules targeting webhooks. They’re the only resources explicitly excluded in the code.



```
func IsWebhookConfigurationResource (attr admission.Attributes)  bool
{
   gvk := attr.GetKind()
   if gvk.Group == "admissionregistration.k8s.io"  {
     if gvk.Kind == "ValidatingWebhookConfiguration"  || gvk.Kind ==
"MutatingWebhookConfiguration"  {
       return true
     }
   }
   return false
}
```

Enforce best practices

While we can’t have a webhook watching other *webhooks*, we can trigger based on the creation of **Services** and **Deployments** and then check for associated webhooks pointing at them.

BUT… this is only works if the webhook has already been created in the cluster. Not much use to us on first install as the deployments and services need to exist first.

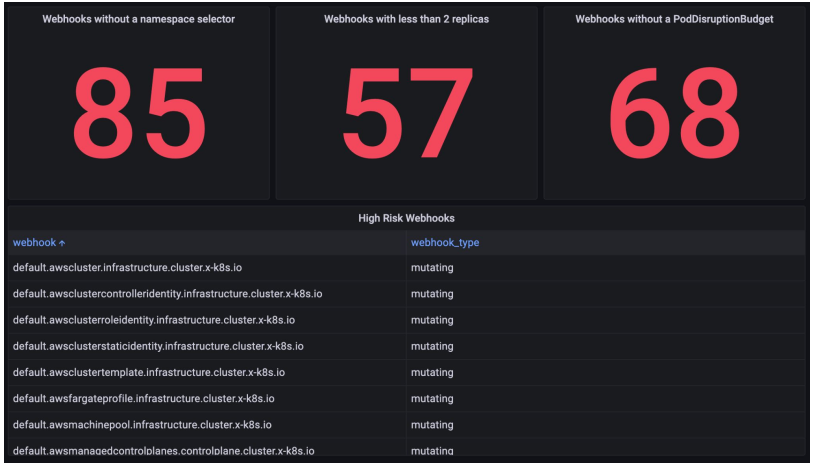
Instead, we must enforce best practices (min replicas, PDB, anti-affinity, etc.) **on all deployments** to ensure we catch all webhook services.

We all follow best practices all the time anyway, right?

Watchdog

Rather than *preventing* the potential issues from being created, we can instead **monitor for their existence**.

An operator running in our cluster watching all webhooks, reporting **metrics** and **alerting** on ones that don’t meet our minimum requirements.



Webhooks without a namespace selector	Webhooks with less than 2 replicas	Webhooks without a PodDisruptionBudget
85	57	68

High Risk Webhooks

webhook ↑	webhook_type
default.awscluster.infrastructure.cluster.x-k8s.io	mutating
default.awsclustercontrolleridentity.infrastructure.cluster.x-k8s.io	mutating
default.awsclusterroleidentity.infrastructure.cluster.x-k8s.io	mutating
default.awsclusterstaticidentity.infrastructure.cluster.x-k8s.io	mutating
default.awsclustertemplate.infrastructure.cluster.x-k8s.io	mutating
default.awsfargateprofile.infrastructure.cluster.x-k8s.io	mutating
default.awsmachinepool.infrastructure.cluster.x-k8s.io	mutating
default.awsmanagedcontrolplanes.controlplane.cluster.x-k8s.io	mutating

Yikes! I'm glad this is a test cluster.

Out of cluster services

It’s possible to point a webhook configuration at an **external endpoint (URL)** instead of a Kubernetes Service resource.

This avoids the issues of the **webhook blocking its own creation** as it’s no longer managed as a Pod.

Needs some other system to ensure the application remains running, stays accessible from the cluster and responds quickly.

The (possible) future

KEP-1872 - Manifest based registration of Admission webhooks

- No gap in enforcement between when apiserver is started and webhook configuration is created
- Prevent deletion of these webhook configurations similar to how static pods are handled

Introduced: 2020-04-21 | **Status:** Dropped

Ok, I said we wouldn't talk about CRD webhooks but this is worth a mention

KEP-2876 - CRD Validation Expression Language

- Implement expression language support (CEL) into current validation mechanism, avoiding some cases where webhooks would be needed
- Make CRD validation more self-contained

Introduced: 2021-05-26 | **Status:** Alpha in v1.23, Beta in v1.25

There was a good blog post about this recently

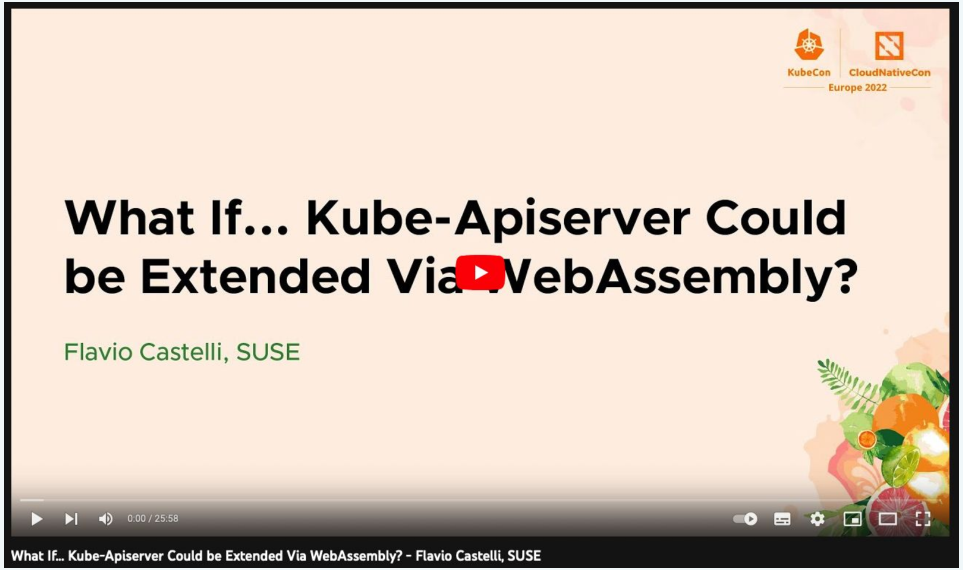
The (possible) future

Proposed idea:

A new apiserver admission plugin that makes use of **WebAssembly** modules to run admission requests against, instead of calling a webhook.

Benefits:

- Less uncertainty from not relying on network
- Less resource usage - no need for multiple controllers, all handled by the apiserver



What If... Kube-Apiserver Could be Extended Via WebAssembly?

Flavio Castelli, SUSE

What If... Kube-Apiserver Could be Extended Via WebAssembly? - Flavio Castelli, SUSE

Wrap-up

The wonders:

- Defaulting
- Policy enforcement
- Best practices
- Issue mitigation

The woes:

- Webhook services need to be resilient
- Cluster can be taken down if not careful
- Very little can be done at a cluster level to ensure foolproof webhooks are used

The future:

- Less reliance on webhooks for things like schema validation
- Admission plugins offering alternative methods to webhooks - e.g. WebAssembly

Wrap-up

Slides and resources available at:

https://go-get.link/wonders-and-woes-webinar

Thoughts, comments and feedback:

✉ feedback@marcusnoble.co.uk

https://k8s.social/@Marcus

Thank you