# Mutating Admission Webhooks

Learn how the mutating admission webhook works in Kubernetes.

We'll cover the following



## Mutating admission webhooks

Kubernetes admission controllers provide us with ways to enforce rules or restrictions on the changes to the cluster, such as updating Pod labels, limiting resource quota, preventing unexpected operations on deleting objects, etc.

Sometimes, when built-in admission plugins don't suffice for what we need, we can use external webhooks for validating and mutating. In such an out-of-tree way, we can inject our custom logic into the Kubernetes admission control pipeline.

### Configuring the mutating admission webhook

Firstly, to use the mutating admission webhook in Kubernetes, we must ensure that the default enabled admission controller plugin `MutatingAdmissionWebhook` is actually enabled. This should be included if we have explicit settings on the flag `--enable-admission-plugins`, such as:

```
--enable-admission-plugins=...,MutatingAdmissionWebhook,...
```

Secondly, we need to create a configuration containing a stanza that looks just like the one below. This configuration can be configured dynamically.

```
apiVersion: admissionregistration.k8s.io/v1 # Could be v1beta1 as well
kind: MutatingWebhookConfiguration
metadata:
  name: demo-mutating-webhook
webhooks:
  - name: mutating-webhook.webhook-demo.svc #  Name must be fully qualified
    clientConfig: # Defines how the kube-apiserver communicates with the hook
      # URL:: "https://some-webhook.demo.com/validate"
      service: # If the webhook is running within the cluster, then we should use service
        name: mutating-webhook
        namespace: webhook-demo
        path: "/validate"
      caBundle: ${CA_PEM_B64}
    rules:
      - operations: [ "CREATE" ]
        apiGroups: [""]
```

```
        apiGroups: [""]
        apiVersions: ["v1"]
        resources: ["pods"]
    timeoutSeconds: 10
```

A sample MutatingWebhookConfiguration (on creation of a Pod)

There are several notable parts in the configuration above:

- We can use either use `admissionregistration.k8s.io/v1` or `admissionregistration.k8s.io/v1beta`, if the cluster version is v1.16+.
- We can define multiple webhooks in a `MutatingWebhookConfiguration` object.
- For every webhook, the name must be fully qualified, such as `mutating-webhook.webhook-demo.svc`.
- In `clientConfig`, we can define the endpoint that the `kube-apiserver` can communicate with. This is required. We can use either `url` or `service`, but not both. Normally, we would run our webhook within the cluster. We can specify the service namespace, name, and server path in `clientConfig.service`. However, the webhook service could be running outside of the cluster as well. Here, we can specify a valid `url` that follows the standard from (`scheme://host:port/path`). The `caBundle` is a base64-encoded CA certificate used to validate the webhook server's certificate. Here, we can't skip checking the validity of the webhook server at any cost.
- For every webhook, we can set a group of rules that describe the operations and the resources/subresources the webhook cares about. For example, the webhook above will only care about the creation of Pods.

When defining a `MutatingWebhookConfiguration` object, there are also some other fields we may pay attention to, like `FailurePolicy`, `MatchPolicy`, `NamespaceSelector`, `ObjectSelector`, `SideEffects`, etc. However, it's okay to leave them as their default values.

Lastly, if our webhook server requires authentication, we need an extra configuration to tell the `kube-apiserver` how to get authenticated with tokens or TLS certificates when talking to webhook servers. Normally, we don't enable authentication for webhook servers, but this will be imperative if we make our whole cluster more secure. We can append the flag `--admission-control-config-file` when starting the `kube-apiserver`. We can specify a series of kubeconfig files in the admission control configuration file for webhook servers. An example file is given below:

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: MutatingAdmissionWebhook
  configuration:
    apiVersion: apiserver.config.k8s.io/v1
    kind: WebhookAdmissionConfiguration
    kubeConfigFile: "<path-to-kubeconfig-file>"
- name: ValidatingAdmissionWebhook
```

```
10    configuration:
11      apiVersion: apiserver.config.k8s.io/v1
12      kind: WebhookAdmissionConfiguration
13      kubeConfigFile: "<another-path-to-kubeconfig-file>"
```

An example of AdmissionConfiguration

The configuration file can be used to configure multiple admission plugins. The name of every plugin in `AdmissionConfiguration` must match the registered admission plugin. Here, we're using `MutatingAdmissionWebhook`. In the kubeconfig file, we only need to specify the credentials for every webhook server, as shown below:

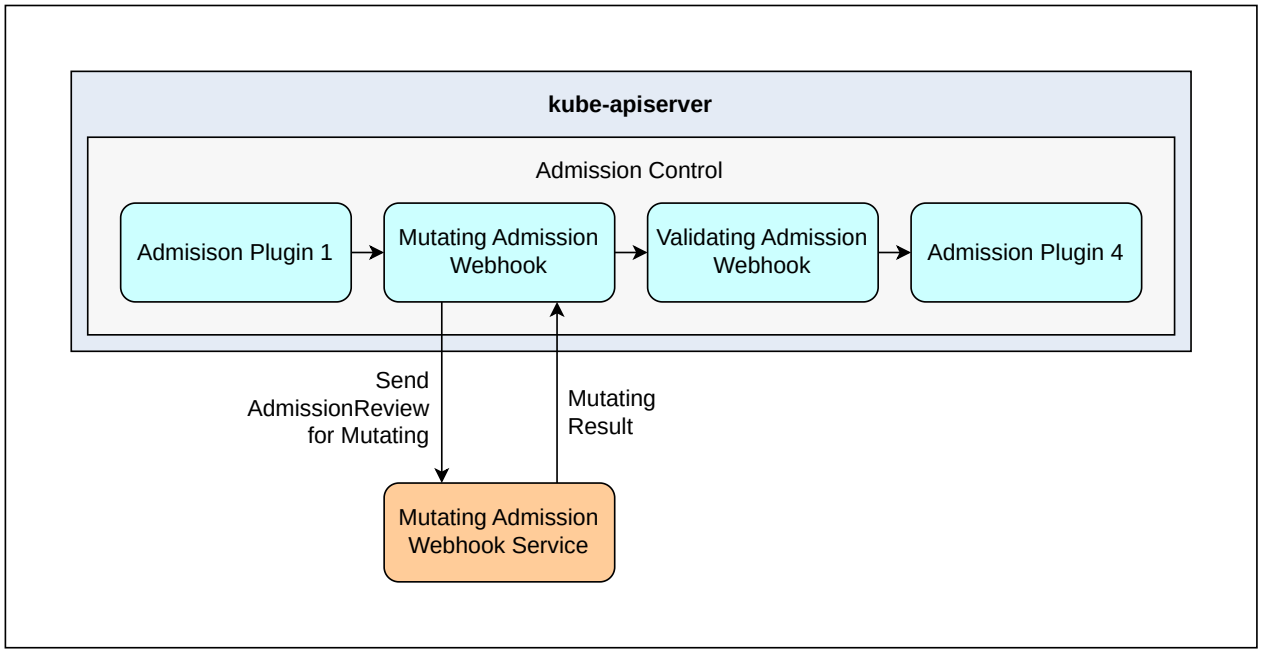
```
 1  apiVersion: v1
 2  kind: Config
 3  users:
 4  - name: 'webhook-demo.ns1.svc:8443'
 5    user:
 6      client-certificate-data: "<pem encoded certificate>"
 7      client-key-data: "<pem encoded key>"
 8  - name: 'external-webhook-1.webhook-company.org'
 9    user:
10      password: "<password>"
11      username: "<name>"
12  - name: 'external-webhook-2.webhook-company.org'
13    user:
14      token: "<token>"
```

A sample kubeconfig file for admission control

## How it works

Now, let's see how the validating admission webhook works.

When the `kube-apiserver` receives a request matching one of the `rules` defined in `MutatingWebhookConfiguration`, it sends out a `POST` request to the admission webhook server specified in `clientConfig`.




How the mutating admission webhook works

The payload of the request is a serialized JSON string of `AdmissionReview` objects in the API group `admission.k8s.io`. Below is an example `AdmissionReview` object for a request to scale a `Deployment` in group `apps/v1`:

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
request:
  # An auto-generated uid, which can uniquely identify this admission call
  uid: dc91021d-2361-4f6d-a404-7c33b9e01118

  # Fully qualified GVK of object being submitted,
  # such as v1.Pod
  kind:
    group: apps
    version: v1
    kind: Deployment

  # Fully qualified GVK of the resource being modified
  resource:
    group: apps
    version: v1
    resource: deployments

  # Fully qualified GVK of the original incoming object
  # This field is used to indicate whether the incoming object
  # needs a conversion.
  requestKind:
    group: apps
    version: v1
    kind: Deployment

  # Fully-qualified GVK of the original resource being modified
  # This field is used to indicate whether the incoming object
  # needs a conversion.
  requestResource:
```

The AdmissionReview request payload

When the admission webhook server receives the `POST` request, it makes decisions based on the payload, in other words, allowing or rejecting such a request. Below is a stanza response from an admission webhook server to allow a request:

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
response:
  # Value from request.uid
  # Here, we use the example above
  uid: dc91021d-2361-4f6d-a404-7c33b9e01118
  allowed: true
  patchType: JSONPatch
  patch: W3sib3AiOiAiYWRkIiwgInBhdGgiOiAiL3NwZWMvcmVwbGljYXMiLCAidmFsdWUiOiAzfV0=
```

The AdmissionReview response

When admitting a request, the incoming object can be optionally modified by the mutating admission webhook server. We can specify the `patchType` and `patch` data in the field `response`. Notice that `response.patch` accepts only base64-encoded data.

One of the notable qualities of good admission controllers is their capability for easy diagnostics. So, when rejecting a request, we should return an error with precise or detailed information about the denial reasons or suggested solutions, such as in the code snippet below:

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
response:
  # Value from request.uid
  # Here, we use the example above
  uid: dc91021d-2361-4f6d-a404-7c33b9e01118
  allowed: false
  status:
    code: 403
    message: You cannot do this because it is Friday. No operations are allowed on Friday.
```

The AdmissionReview response with an error message

When the `kube-apiserver` receives the response, it will return the HTTP status code and message