Validating Admission Webhooks

Learn about the validating admission webhooks in Kubernetes.

We'll cover the following

Validating admission webhooks

Kubernetes provides advanced dynamic mechanisms to gate and govern changes on the configurations and workloads. With admission controllers, we can have granular control over things such as vulnerable container images, containers injections, labels checking, etc. Kubernetes provides not only built-in admission plugins, but also supports external webhooks, such as `ImagePolicyWebhook`, `Validating AdmissionWebhook`, and `MutatingAdmissionWebhook`.

With these pluggable webhooks, we can easily hook our own business logic into the admission control phase in the `kube-apiserver`. In this lesson, we're going to dive into the validating admission webhooks.

Configuration

First, to use the validating admission webhook in Kubernetes, we must ensure that the default enabled admission controller plugin `ValidatingAdmissionWebhook` is in fact enabled, which should be included if we have explicit settings on the flag `--enable-admission-plugins`, such as:

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

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

```
apiVersion: admissionregistration.k8s.io/v1 # Could be v1beta1 as well
kind: ValidatingWebhookConfiguration
metadata:
  name: demo-validating-webhook
webhooks:
  - name: validating-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: validating-webhook
        namespace: webhook-demo
        path: "/validate"
      caBundle: ${CA_PEM_B64}
    rules:
      - operations: [ "CREATE" ]
        apiGroups: [""]
        apiVersions: ["v1"]
        resources: ["pods"]
    timeoutSeconds: 10
```

A sample ValidatingWebhookConfiguration (upon creation of a Pod)

There are several notable parts in the configuration above:

- We can use either `admissionregistration.k8s.io/v1` or `admissionregistration.k8s.io/v1beta`, if the cluster version is v1.16+.
- We can define multiple webhooks in a `ValidatingWebhookConfiguration` object.
- For every webhook, the name must be fully qualified, such as `validating-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 serving 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 resources/subresources that the webhook cares about. For example, the webhook above will only care about the creation of Pods.
- When defining a `ValidatingWebhookConfiguration` object, there are also some other fields we may pay attention to, like `FailurePolicy`, `MatchPolicy`, `NamespaceSelector`, `ObjectSelector`, `SideEffects` etc. But 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: ValidatingAdmissionWebhook
```

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

A sample 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 `ValidatingAdmissionWebhook`. In the kubeconfig file, we only need to specify the credentials for every webhook server, as shown below:

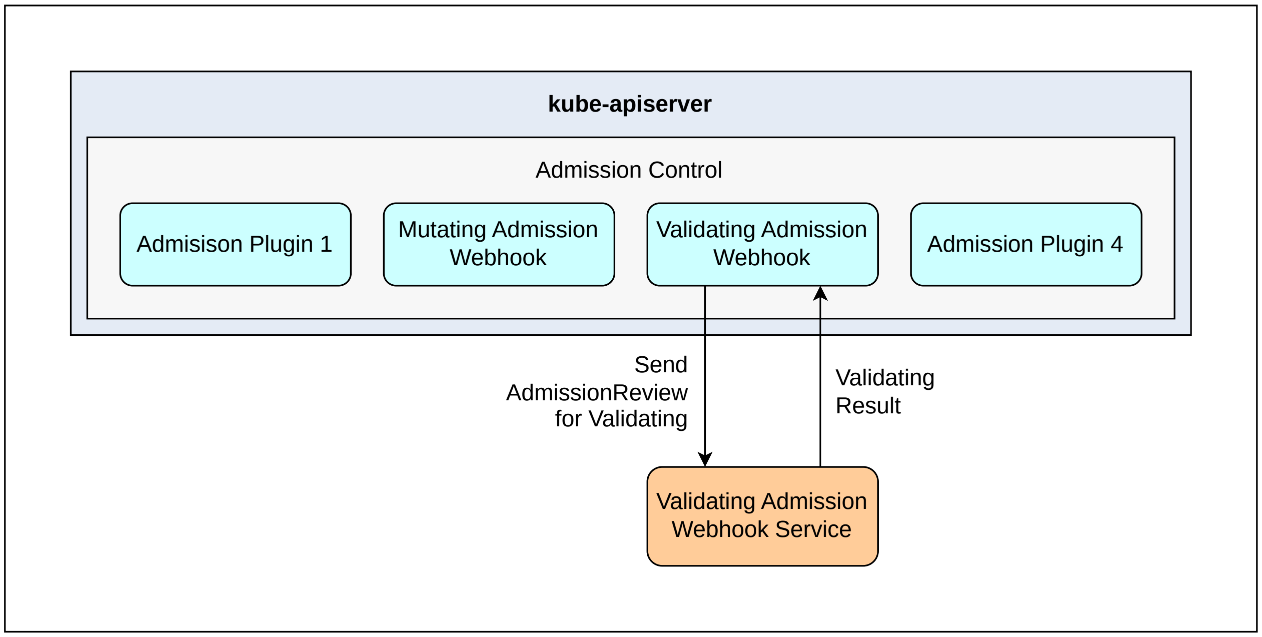
```
apiVersion: v1
kind: Config
users:
- name: 'webhook-demo.ns1.svc:8443'
  user:
    client-certificate-data: "<pem encoded certificate>"
    client-key-data: "<pem encoded key>"
- name: 'external-webhook-1.webhook-company.org'
  user:
    password: "<password>"
    username: "<name>"
- name: 'external-webhook-2.webhook-company.org'
  user:
    token: "<token>"
```

A sample kubeconfig file for admission control

How it works

Now, let's see how validating admission webhooks work.

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


kube-apiserver
Admission Control
Admisison Plugin 1
Mutating Admission Webhook
Validating Admission Webhook
Admission Plugin 4
Send AdmissionReview for Validating
Validating Result
Validating Admission Webhook Service

How validating admission webhooks works

The payload of the request is a serialized JSON string of the object `AdmissionReview` in the API group `admission.k8s.io`. Below is an example of a `AdmissionReview` object for a request to scale a `Deployment` in the 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 autoscaling.v1.Scale or v1.Pod"
  kind:
    group: autoscaling
    version: v1
    kind: Scale

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

  subResource: scale

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

  # Fully-qualified GVK of the original resource being modified
```

The AdmissionReview request payload

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

```yaml
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
```

The AdmissionReview response

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 on the denial reasons or suggested remediations, like in the code snippet below:

```yaml
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 presented here to the user.