How Webhook Authorization Works

Learn how webhook authorization works in Kubernetes.

We'll cover the following

Webhook authorization

The task during the authorization stage is to determine user privileges, in other words, if the user is allowed to perform the requested action. For example, the user `Bob` is trying to create a `Pod`. During the authorization stage, Kubernetes needs to verify if `Bob` is allowed to `POST` a `Pod` to the `kube-apiserver`.

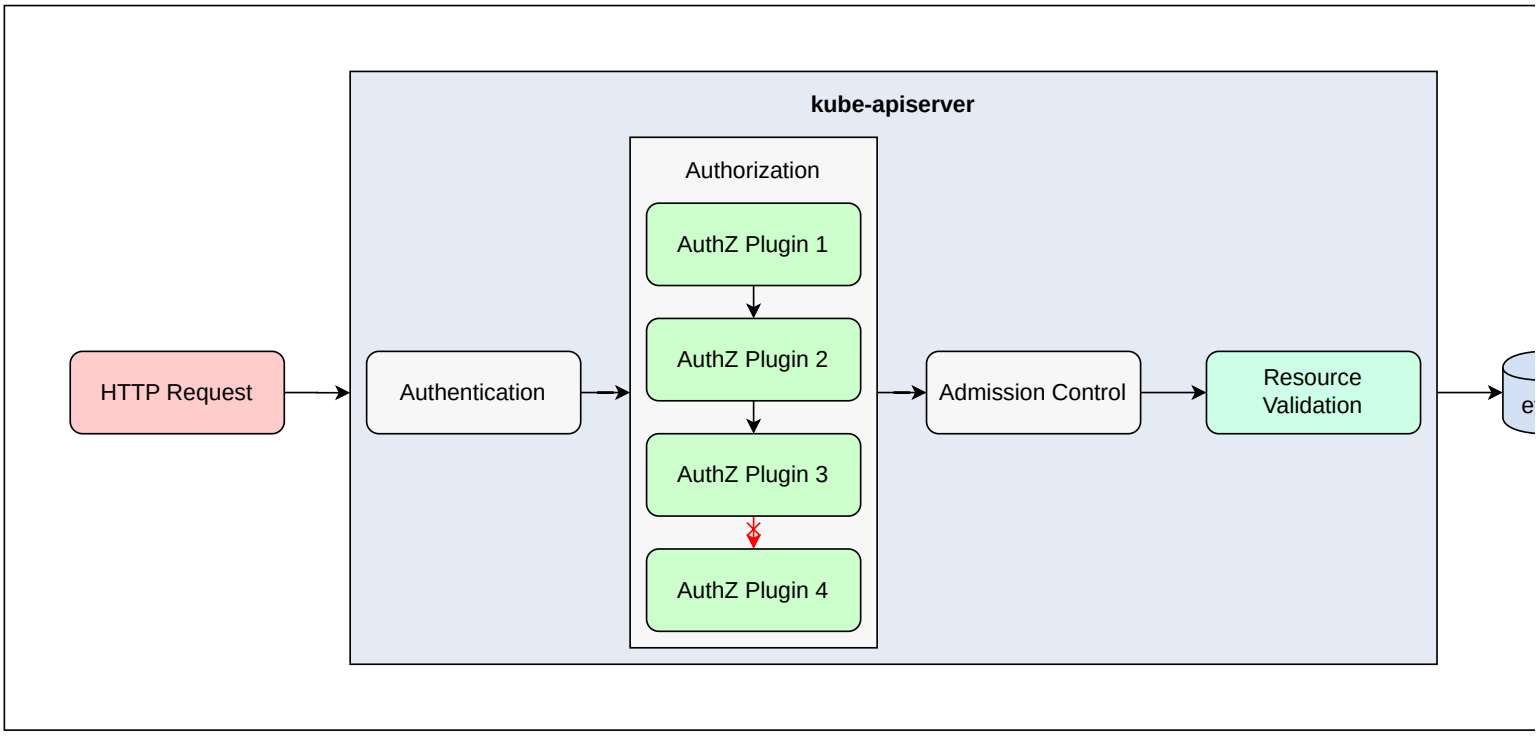
Kubernetes bundles a group of authorization plugins as a union authorization chain, just as the code snippet below shows:

```
// Code from https://github.com/kubernetes/kubernetes/blob/master/staging/src/k8s.io/apiserver/pkg/autho
// New() returns an authorizer that authorizes against a chain of authorizer.Authorizer objects
func New(authorizationHandlers ...authorizer.Authorizer) authorizer.Authorizer {
    return unionAuthzHandler(authorizationHandlers)
}

// Authorizes against a chain of authorizer.Authorizer objects and returns nil if successful and returns
func (authzHandler unionAuthzHandler) Authorize(ctx context.Context, a authorizer.Attributes) (authorize
    var (
        errlist    []error
        reasonlist []string
    )

    for _, currAuthzHandler := range authzHandler {
        decision, reason, err := currAuthzHandler.Authorize(ctx, a)

        if err != nil {
            errlist = append(errlist, err)
        }
        if len(reason) != 0 {
            reasonlist = append(reasonlist, reason)
        }
        switch decision {
        case authorizer.DecisionAllow, authorizer.DecisionDeny:
            return decision, reason, err
        case authorizer.DecisionNoOpinion:
            // continue to the next authorizer
        }
    }

    return authorizer.DecisionNoOpinion, strings.Join(reasonlist, "\n"), utilerrors.NewAggregate(errlist
```

Union authorizer in the kube-apiserver

Each plugin implements a specific authorization method, such as Node, RBAC, ABAC, etc. Any authenticated requests will be presented to each authorization plugin one by one, until one of them can successfully determine user privileges on the requested resource. Here, the `UserInfo` obtained from the previous authentication stage is used for decision making.

Then, the authorization stage finishes and the request proceeds to the following admission control stage. If none of the authorization plugins can determine user privileges, the coming request is rejected with a `403 Forbidden` HTTP status code.



The kube-apiserver authorization flow

In this lesson, we'll elaborate on how authorization webhook works in Kubernetes.

How to configure the authorization webhook

We need to first configure the five flags below to trigger using the authorization webhook:

- `--authorization-mode`: We need to make sure that `Webhook` is present in this comma-delimited list.
- `--authorization-webhook-config-file`: This is the file that tells the `kube-apiserver` where to access the remote authorization webhook service. This file is in the same format as `kubeconfig`, as discussed in the previous lesson.
- `--authorization-webhook-cache-authorized-ttl`: This flag specifies how long it will take to cache authorization decisions. It's set to `5m0s` by default. This improves the performance of the `kube-apiserver` by caching authorization results, because every request needs passing authorization. In

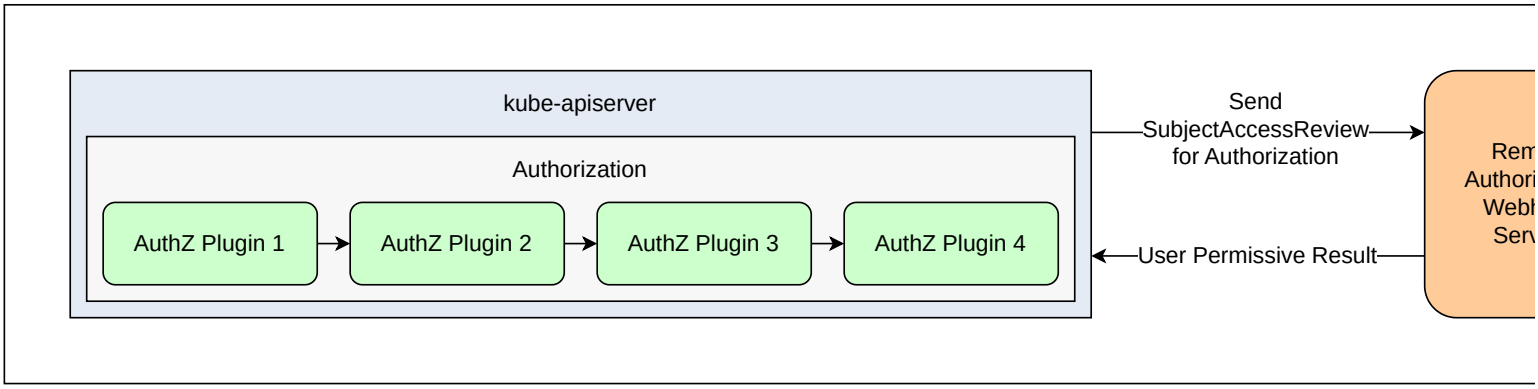
actual production environments, we can leave this to the default setting unless there's a need for adjustment.

- `--authorization-webhook-cache-unauthorized-ttl`: This flag, which is set to `30s` by default, specifies how long it will take to cache unauthorized decisions. In actual production environments, we can leave this to the default setting unless there's a need for adjustment.
- `--authorization-webhook-version`: This flag determines the API version of the `SubjectAccessReview` objects in group `authorization.k8s.io` to send to and expect from the remote webhook authorization service. Currently, in Kubernetes v1.23, it remains v1beta1 by default.

How it works

Now, let's see how this authorization webhook plugin works.

The authorization webhook is an HTTP callback that will be called in the authorization stage.



How the webhook authorization works

During the authorization stage, if the preceding plugins in the union authorization chain fail to determine user privileges, the webhook plugin will invoke an HTTP POST request to the remote authorization service specified in the flag `--authorization-webhook-config-file`.

```
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "kube-system",
      "verb": "get",
      "group": "",
      "resource": "pods"
    },
    "user": "Bob",
    "group": [
      "group1"
    ]
  }
}
```

The SubjectAccessReview request payload

> **Note:** By default, the `kube-apiserver` sends the `SubjectAccessReview` resources with version v1beta1. While the `SubjectAccessReview` of this version has a typo in the field `groups`, which is serialized to `group` (singular), this typo has already been fixed in the v1 version. If we want to use the v1 version, we can override the `kube-apiserver` flag `--authorization-webhook-version` to `v1`.

Here, the remote authorization webhook service is entirely independent of Kubernetes. It can run anywhere as long as the `kube-apiserver` can access it. The implementation of this service is completely up to us, and we can use any language we're familiar with. All the communications are through HTTP. In the next lesson, we'll demonstrate how to run it inside of a Kubernetes cluster by exposing a `Service`.

The response from the remote authorization service should also be a valid `SubjectAccessReview` object. The `status` field contains the authenticated result and detailed user info, as shown below:

```
1  {
2    "apiVersion": "authorization.k8s.io/v1beta1",
3    "kind": "SubjectAccessReview",
4    "status": {
5      "allowed": true
6    }
7  }
```

The SubjectAccessReview response

Summary

RBAC is the most commonly used authorization method in Kubernetes. If our cluster has some special requirements, the authorization webhook is a good option. As we've seen above, the authorization webhook provides a customized way through which we can easily define our own user privileges and build an authorization system to meet any requirements we have.

Back

Implementing a Webhook for Token Authentication

Next

Implementing a Webhook for Authorization