Implementing a Validating Admission WebHook

Learn how to implement a validating admission webhook service.

We'll cover the following

Implement a validating admission webhook

A validating admission webhook service is a web server, because the `kube-apiserver` invokes it through HTTPS `POST` requests. Now, let's implement such a service step by step.

Step 1: Write a simple HTTP server

Let's write a simple HTTP server at the path `/validate` on port `443`. It checks the Pod name and rejects all Pods having `mock-app` in their names. The development environment that we can use to add and modify our programs is given below. We can hit the "Run" button to initialize it.

main.go

Search in directory...

/
- ca.pem
- go.sum
- go.mod
- webhook-deploy.yml
- webhook-config.yml
- Dockerfile
- main.go

```
1  package main
2
3  import (
4      "encoding/json"
5      "flag"
6      "fmt"
7      "io/ioutil"
8      "log"
9      "net/http"
10     "strings"
11
12     admissionapi "k8s.io/api/admission/v1"
13     corev1 "k8s.io/api/core/v1"
14     metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
15 )
16
17 var (
18     podResource = metav1.GroupVersionResource{Version: "v1", Reso
19 )
20
21 func validate(admissionReq *admissionapi.AdmissionRequest) (admis
22     admissionResp = &admissionapi.AdmissionResponse{}
23     // copy uid from tr.Request
24     admissionResp.UID = admissionReq.UID
```

```
25
26      var err error
27      defer func() {
28          // If the handler returned an error, incorporate the erro
29          if err != nil {
30              admissionResp.Allowed = false
31              admissionResp.Result = &metav1.Status{
```

Our development environment

In a real-world scenario, we only need to replace the mock code in the function `validate()` (**line 21** in `main.go`) with our actual business logic. In the demo above, we're handling Pods, and other resources can be handled too, only if we've set matching rules in `ValidatingAdmissionConfiguration`.

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 remediations, like above. We're rejecting all the Pods whose names contain `mock-app` and returning a hint message.

The rest of the code (**lines 68–117** in `main.go`) is a simple HTTP handler that handles requests from the `kube-apiserver` and deserializes the payloads, which are expected to be an `AdmissionReview` object in JSON format. This HTTP handler then sends the validating result back.

Another notable thing is that, after we finish handling `AdmissionReview`, we should not forget to set the `Response.UID`, which matches the `Request.UID`.

Step 2: Generate and use the certificates (optional)

This section can be skipped, because we've packed all the needed certificates into our prebuilt container image. However, if you want to know more about the detailed steps, follow the instructions below.

In production environments, the `kube-apiserver` uses the CA certificate to visit our admission webhooks, which are serving securely with HTTPS. Now, let's use `cfssl` to create some certificates for safe serving. Run the following commands in the terminal above:

```
1  echo '{"CA":{"expiry": "87600h","pathlen":0},"CN":"CA","key":{"algo":"rsa","size":2048}}' | cfssl gencer
2  echo '{"signing":{"default":{"expiry":"87600h","usages":["signing","key encipherment","server auth","cl
3  export ADDRESS=localhost,validating-admission-demo.kube-system.svc
4  export NAME=server
5  echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl gencert -config=ca-config
```

Generate keys and certificates for secure serving

Now, we should have `ca.pem`, `server.pem`, and `server-key.pem` in our directory.

Here, we generate a self-signed certificate with the `CN` field set to `validating-admission-demo.kube-system.svc`. We're going to run this service in Kubernetes, which is the endpoint that we want to expose. The DNS name `validating-admission-demo.kube-system.svc` indicates that there's a `Service` named `validating-admission-demo` running in the namespace `kube-system`.

Step 3: Build and deploy it

We've already built an image `dixudx/pwk:validating-admission-webhook`. If you want to build your own, run the following command in the terminal above:

```
1  docker build -t dixudx/pwk:validating-admission-webhook ./
```

Build a container image

After the image has been built successfully, we can deploy the webhook in Kubernetes. Now, let's run the following commands in the terminal above:

```
1  kubectl apply -f /usercode/webhook-deploy.yml
```

Deploy the webhook

Now, we run the following commands to make sure the Pods are running healthily:

```
1  kubectl get pod -n kube-system | grep validating-admission-demo
```

Check our demo Pods

The output will be as follows:

```
1  validating-admission-demo-6488cdcccc-8jzzz   1/1     Running   0          33s
2  validating-admission-demo-6488cdcccc-d9g4f   1/1     Running   0          33s
3  validating-admission-demo-6488cdcccc-xkx2v   1/1     Running   0          33s
```

Our running demo Pods

Then, we declare `ValidatingWebhookConfiguration` in the `kube-apiserver`.

```
1  CABUNDLE=`base64 -w 0 ca.pem` envsubst < /usercode/webhook-config.yml | kubectl apply -f -
```

Apply validating webhook configuration

In order to let our static `kube-apiserver` Pod access the webhook service with the DNS name `validating-admission-demo.kube-system.svc`, we need to export the DNS record as follows. Run it in the terminal above.

```
1  echo "$(kubectl get svc -n kube-system validating-admission-demo | grep -v NAME | awk '{print $3}') val
```

Add a DNS record

In this lesson, our `kube-apiserver` is running as a static Pod, whose lifecycle is managed by the `kubelet`. We only need to update the `kube-apiserver` manifest file, and then the `kubelet` will be notified of the change and restart it later. Then, the `kube-apiserver` can take effect on the DNS updates. In the terminal above, we can take the steps shown below:

```
1  docker ps -a | grep apiserver | grep -v "pause" | awk '{print $1}' | xargs docker rm -f
```

Enable webhook authentication in the kube-apiserver

Step 4: Test it

Let's do some tests to see whether our validating admission webhook `validating-admission-demo.kube-system.svc` takes effect for the creation of Pods.

```
1  kubectl run a-mock-app --image=nginx:1.23
```

Construct a Pod for testing

The command above will create a Pod with the name `a-mock-app-pod`, which contains `mock-app`. Let's run it in the terminal above to see what happens.

```
Error from server: admission webhook "validating-admission-demo.kube-system.svc" denied the request: K
eep calm and this is a webhook demo in the cluster!
```

We can see that the Pod `a-mock-app` is rejected correctly by our webhook, which is working as expected.

Ta-da! The validating admission webhook works!

Back

Validating Admission Webhooks

Next

Mutating Admission Webhooks