Design Document for Kubernetes Admission Webhook

XiangYing Zhang

Contents

1 Introduction

This document describes the design of a Kubernetes Admission Webhook aimed at controlling the scheduling of pods in a cluster. The webhook enforces that the first pod is scheduled on an on-demand (OD) node, while the remaining pods are scheduled on spot instances.

2 Problem Statement

Kubernetes clusters that use spot instances are cost-effective but come with the risk of interruption. In critical systems, certain pods must run on stable on-demand nodes, while others can be scheduled on less expensive spot instances. The webhook provides a mechanism to control this scheduling behavior dynamically, based on the order in which pods are created.

3 System Architecture

3.1 Workflow

1. **Pod Creation**: When a pod creation request is sent to the Kubernetes API server, the webhook intercepts it.(I only intercept pod creation requests,So please make use of kubectl create command)
2. **Scheduling Decision**: Based on the current pod count, the webhook applies the following rules:
   - **First Pod**: Scheduled on an on-demand node(Required).
   - **Subsequent Pods**: Scheduled on spot nodes(Prefered, if not more spot nodes,then on-demand nodes).
3. **Patch Response**: The webhook responds to the API server with a JSON patch to adjust the pod's affinity settings according to the rules.
4. **Pod Scheduling**: The API server then schedules the pod based on the modified affinity settings.

4 Detailed Design

4.1 Webhook Implementation

The webhook is implemented in Go and uses the Kubernetes Admission API to handle admission requests. The webhook inspects the pod definition and modifies its node affinity based on the current count of scheduled pods.

Key components:

- **Counter (`replicaCounter`)**: Tracks the number of pods processed. It ensures that only the first pod is scheduled on an on-demand node, while others go to spot instances.

- **Mutex Lock**: Ensures thread safety for the counter, especially in a multi-threaded environment where multiple pod requests can be processed simultaneously.
- **Patches**: The webhook creates JSON patches that modify the affinity of the pods before they are scheduled.

4.2 Scheduling Rules

First Pod: The webhook assigns the pod to an on-demand node by modifying its node affinity using the following patch:

```
{
  "op": "add",
  "path": "/spec/affinity",
  "value": {
    "nodeAffinity": {
      "requiredDuringSchedulingIgnoredDuringExecution": {
        "nodeSelectorTerms": [
          {
            "matchExpressions": [
              {
                "key": "node.kubernetes.io/capacity",
                "operator": "In",
                "values": ["on-demand"]
              }
            ]
          }
        ]
      }
    }
  }
}
```

Subsequent Pods: For all other pods, the webhook assigns them to spot nodes.Similar to on-demand nodes, but the api method needs to be changed to PreferredSchedulingTerm, and you must explicitly specify a weight, which is used to indicate the weight that affinity gives to this label. If you give multiple label weights, the scheduler will schedule the pod on the node with the highest weight.

4.3 Webhook Server

The webhook is served over HTTPS to ensure secure communication between the Kubernetes API server and the webhook. The server listens on port `8443` and uses TLS certificates to secure the connection.

Key Functions:

- `handleMutate`: The api server will send a message of type admission review, which includes a request. Normally you need to add a response. The message body is of json type. So we can call json.unmarshal to convert it into a pod object.Then, we modify the node's affinity through JSONpatch, the modification method is introduced above. Finally, put the patch in the response and send it back to the API server

- `main`: Initializes the webhook server and listens for requests.

5 Certification

In Kubernetes, the API server calls the Admission Webhook service via HTTPS. To ensure the security of data transmission, the Webhook service must use a TLS certificate to encrypt communication. We generated the tls.key private key and tls.crt signing certificate. We follow the steps below to configure

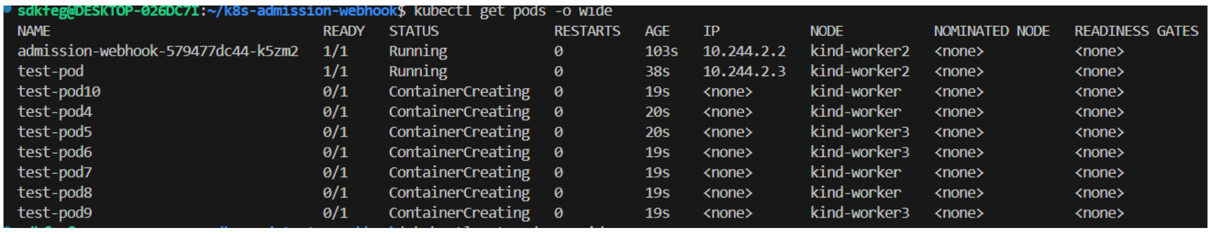
1. Store the generated .key and .crt files in Kubernetes
2. When the Webhook service starts, it loads the TLS certificate file from /etc/webhook/certs
3. The Kubernetes API server needs to verify that the webhook service's certificate is authentic, so we need to add the base64-encoded content of the CA certificate to the cabundle section of the webhook configuration file. The API server will use this CA to verify the TLS certificate of the webhook service.
4. Specify the certificate file path when the Webhook service starts the server and use it in the http.ListenAndServeTLS function

6 Configuration

It is worth noting that, as mentioned earlier, since the operation is Create and the resources is pod, we will only intercept the pod creation request.

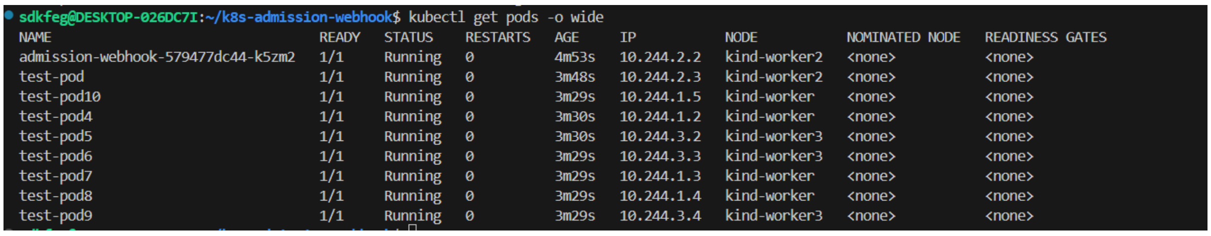
7 Results

For testing, I built 4 nodes through kind, 2 od and 2 spot. The following is the scheduling result of adding the first test node



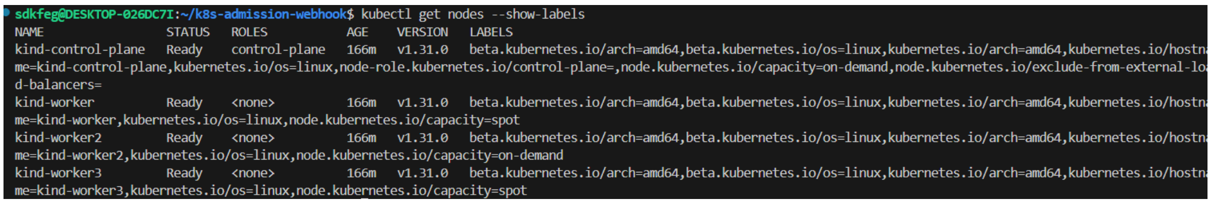
```
sdkfeg@DESKTOP-026DC7I:~/k8s-admission-webhook$ kubectl get pods -o wide
NAME                                 READY   STATUS              RESTARTS   AGE    IP           NODE           NOMINATED NODE   READINESS GATES
admission-webhook-579477dc44-k5zm2   1/1     Running             0          103s   10.244.2.2   kind-worker2   <none>           <none>
test-pod                             1/1     Running             0          38s    10.244.2.3   kind-worker2   <none>           <none>
test-pod10                           0/1     ContainerCreating   0          19s    <none>       kind-worker    <none>           <none>
test-pod4                            0/1     ContainerCreating   0          20s    <none>       kind-worker    <none>           <none>
test-pod5                            0/1     ContainerCreating   0          20s    <none>       kind-worker3   <none>           <none>
test-pod6                            0/1     ContainerCreating   0          19s    <none>       kind-worker3   <none>           <none>
test-pod7                            0/1     ContainerCreating   0          19s    <none>       kind-worker    <none>           <none>
test-pod8                            0/1     ContainerCreating   0          19s    <none>       kind-worker    <none>           <none>
test-pod9                            0/1     ContainerCreating   0          19s    <none>       kind-worker3   <none>           <none>
```

After that I added 7 new test pods, The following is the scheduling result



```
sdkfeg@DESKTOP-026DC7I:~/k8s-admission-webhook$ kubectl get pods -o wide
NAME                                 READY   STATUS    RESTARTS   AGE     IP            NODE           NOMINATED NODE   READINESS GATES
admission-webhook-579477dc44-k5zm2   1/1     Running   0          4m53s   10.244.2.2    kind-worker2   <none>           <none>
test-pod                             1/1     Running   0          3m48s   10.244.2.3    kind-worker2   <none>           <none>
test-pod10                           1/1     Running   0          3m29s   10.244.1.5    kind-worker    <none>           <none>
test-pod4                            1/1     Running   0          3m30s   10.244.1.2    kind-worker    <none>           <none>
test-pod5                            1/1     Running   0          3m30s   10.244.3.2    kind-worker3   <none>           <none>
test-pod6                            1/1     Running   0          3m29s   10.244.3.3    kind-worker3   <none>           <none>
test-pod7                            1/1     Running   0          3m29s   10.244.1.3    kind-worker    <none>           <none>
test-pod8                            1/1     Running   0          3m29s   10.244.1.4    kind-worker    <none>           <none>
test-pod9                            1/1     Running   0          3m29s   10.244.3.4    kind-worker3   <none>           <none>
```

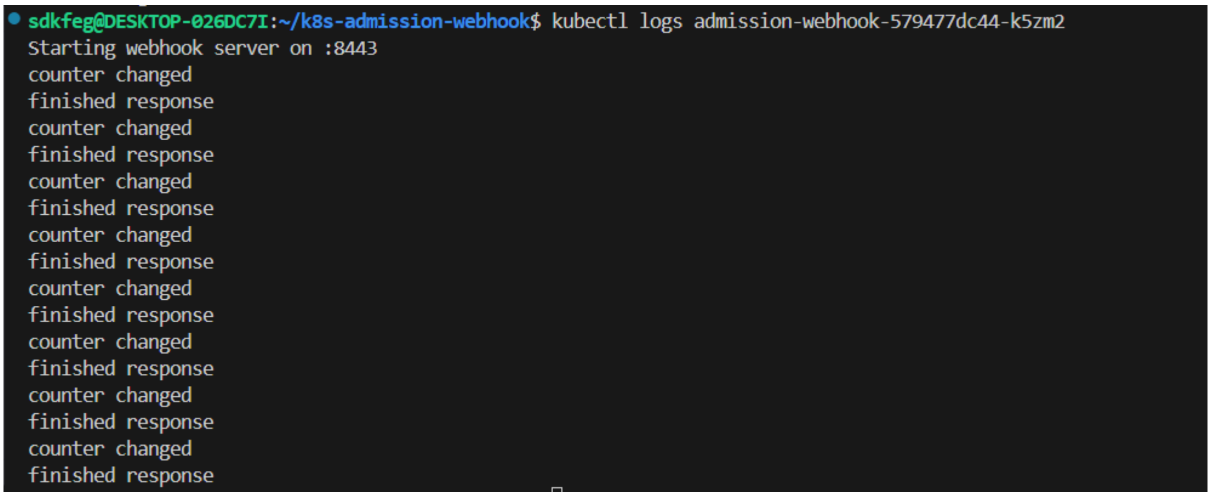
The following figure shows the labels of each node Worker and worker3 are spot nodes,



```
sdkfeg@DESKTOP-026DC7I:~/k8s-admission-webhook$ kubectl get nodes --show-labels
NAME                 STATUS   ROLES           AGE    VERSION   LABELS
kind-control-plane   Ready    control-plane   166m   v1.31.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostna
me=kind-control-plane,kubernetes.io/os=linux,node-role.kubernetes.io/control-plane=,node.kubernetes.io/capacity=on-demand,node.kubernetes.io/exclude-from-external-loa
d-balancers=
kind-worker          Ready    <none>          166m   v1.31.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostna
me=kind-worker,kubernetes.io/os=linux,node.kubernetes.io/capacity=spot
kind-worker2         Ready    <none>          166m   v1.31.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostna
me=kind-worker2,kubernetes.io/os=linux,node.kubernetes.io/capacity=on-demand
kind-worker3         Ready    <none>          166m   v1.31.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostna
me=kind-worker3,kubernetes.io/os=linux,node.kubernetes.io/capacity=spot
```

and subsequent pods are indeed scheduled on these nodes.

Check webhook's log,we can see contents below



```
sdkfeg@DESKTOP-026DC7I:~/k8s-admission-webhook$ kubectl logs admission-webhook-579477dc44-k5zm2
Starting webhook server on :8443
counter changed
finished response
counter changed
finished response
counter changed
finished response
counter changed
finished response
counter changed
finished response
counter changed
finished response
counter changed
finished response
counter changed
finished response
```

8 Validation

```
kind create cluster --config kind-config.yaml
kubectl create secret tls webhook-tls --cert=tls.crt --key=tls.
    key
# can be ignored,docker image construction
docker build -t sdkfeg/admission-webhook:latest .
docker push sdkfeg/admission-webhook:latest
#
kubectl apply -f admission-webhook-deployment.yaml
kubectl apply -f MutatingWebhookConfiguration.yaml
kubectl create -f test-pod.yaml
kubectl create -f test-pod4.yaml
kubectl create -f test-pod5.yaml
kubectl create -f test-pod6.yaml
kubectl create -f test-pod7.yaml
kubectl create -f test-pod8.yaml
kubectl create -f test-pod9.yaml
kubectl create -f test-pod10.yaml
```

9 Future Improvements

- **Load Balancing**: Deploy admission webhook over multiple nodes to reduce the loading.And we need to replace the global variable to a redis instance to avoid inconsistency.

- **Error Handling**: Improve error handling for edge cases, such as failures in pod creation or network issues between the API server and the webhook.