DevOpsK8s WordPress

This is project 1 in course 1 of the CalTech DevOps post-graduate certificate program

System configuration

Environement

- 3 Ubuntu 20.04.3 LTS VMs running on HyperV hypervisor
    - devops (192.168.1.129)
    - devops2 (192.168.1.130)
    - k8sworker1 (192.168.1.133)
- K3s 2 node running on devops2 and k8sworker1
- Jenkins running on devops
- Personal Repository hosted at https://github.com/RedOneLima/DevOpsK8sWordPress

Deployment description

- This deployment contains wordpress and a mariadb database to back the wordpress instance. This contains a few different Kubernetes structures
    - Wordpress
        - Deployment
            - This defines the application configuration and all relivent controllers
        - Service
            - This defines the service bound to the deployment for cluster communication as well as access to the WP instance from outside the cluster
        - Persistant Volume Claim
            - This creates a volume to persist data from the WP container
    - MariaDB
        - Secret
            - Defines the password for the DB
        - Deployment
            - This defines the application configration
        - Service
            - Defines where the DB can be reached
        - Persistant Volume Claim
            - Persists the DB data

Create K3s cluster

- On the master node, run the Rancher K3s script (devops2)

```
curl -sfL https://get.k3s.io | sh -
```

- Once this is set up, we can get the token to join nodes to the new cluster by copying `/var/lib/rancher/k3s/server/node-token`
- From the worker node, we're going to run the same script, however we're going to add some additional options to make it join the cluster instead of starting a new control plane

```
#I've set the contents of /var/lib/rancher/k3s/server/node-token on the
master node as $mynodetoken

$ curl -sfL https://get.k3s.io | K3S_URL=https://devops2:6443
K3S_TOKEN=$mynodetoken sh -
```

- From the master node, we're going to move the kubectl configuration to the default location to be able to run kubectl without using the k3s command

```
$ mkdir ~/.kube
$ sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
```

- Now we can check if the nodes are connected and ready

```
$ kubectl get nodes

NAME          STATUS   ROLES                  AGE   VERSION
k8sworker01   Ready    <none>                 12d   v1.22.5+k3s1
devops2       Ready    control-plane,master   12d   v1.22.5+k3s1
```

- Finally, we want to copy the kubectl configuration over to devops so that we don't have to run kubectl on any of the cluster nodes. This will help identify connectivity issues that are masked by using a localhost configuration.

```
# From devops2
scp ~/.kube/config devops:~/.kube/config
```

- We need to point that configuration at the master node in the config file since by default its going to be localhost

```
# From devops change localhost to devops2 for server:
$ vim ~/.kube/config

...
server: https://devops2:6443
...
```

- Run get nodes from devops to make sure there's connectivity to cluster

```
$ kubectl get nodes

NAME          STATUS   ROLES                  AGE   VERSION
k8sworker01   Ready    <none>                 12d   v1.22.5+k3s1
devops2       Ready    control-plane,master   12d   v1.22.5+k3s1
```

K8s Configuration for Jenkins

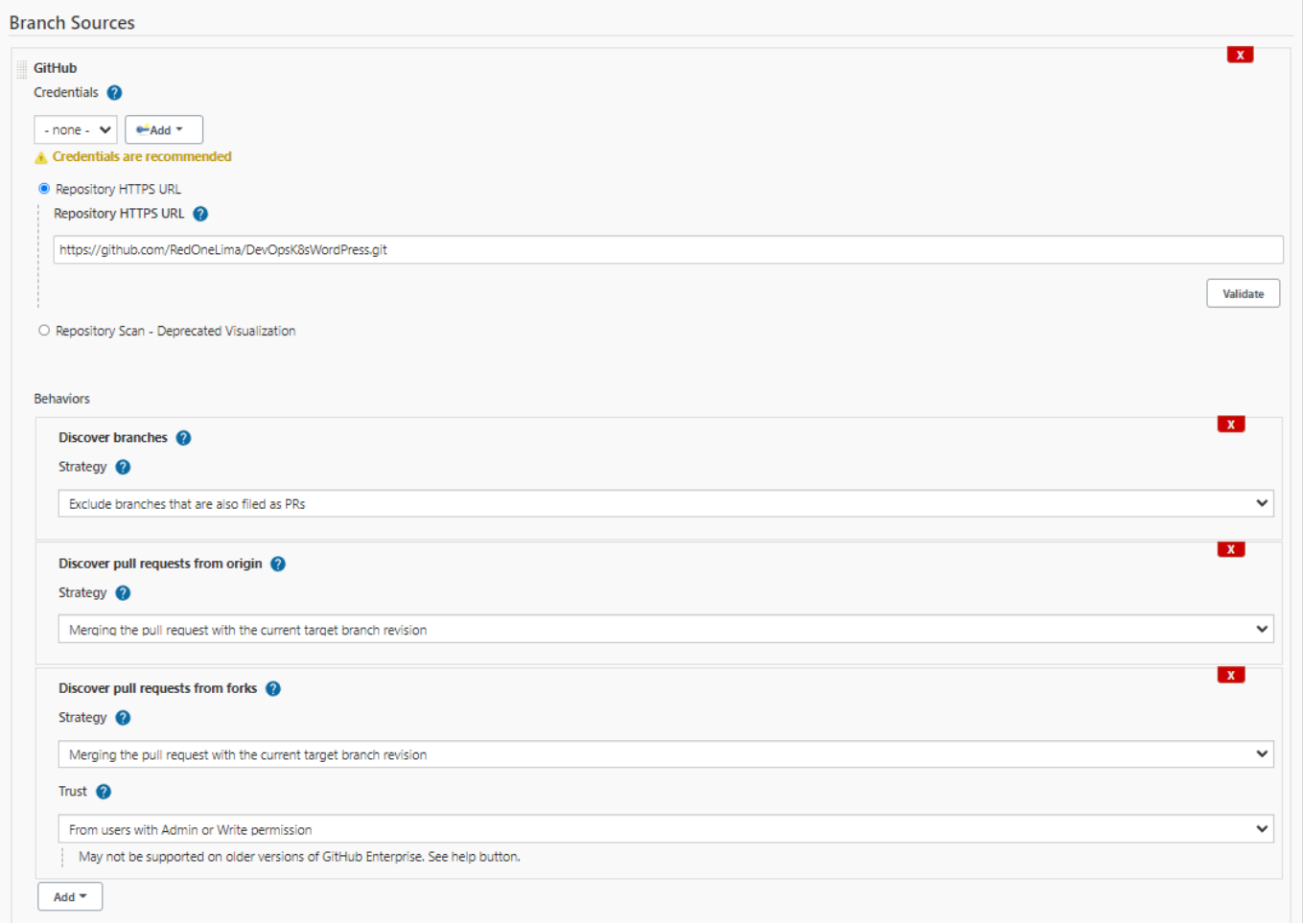
- For Jenkins to be able to send commands to the cluster, a service account and cluster role binding and generating secrets for the Jenkins host

```
$ kubectl -n <namespace> create serviceaccount jenkins-robot

$ kubectl -n <namespace> create rolebinding jenkins-robot-binding --
clusterrole=cluster-admin --serviceaccount=<namespace>:jenkins-robot

$ kubectl -n <namespace> get serviceaccount jenkins-robot -o go-template --
template='{{range .secrets}}{{.name}}{{"\n"}}{{end}}'

jenkins-robot-token-d6d8z

# Retrieve the token and decode it using base64.
$ kubectl -n <namespace> get secrets jenkins-robot-token-d6d8z -o go-
template --template '{{index .data "token"}}' | base64 -d

eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2V[...]
```

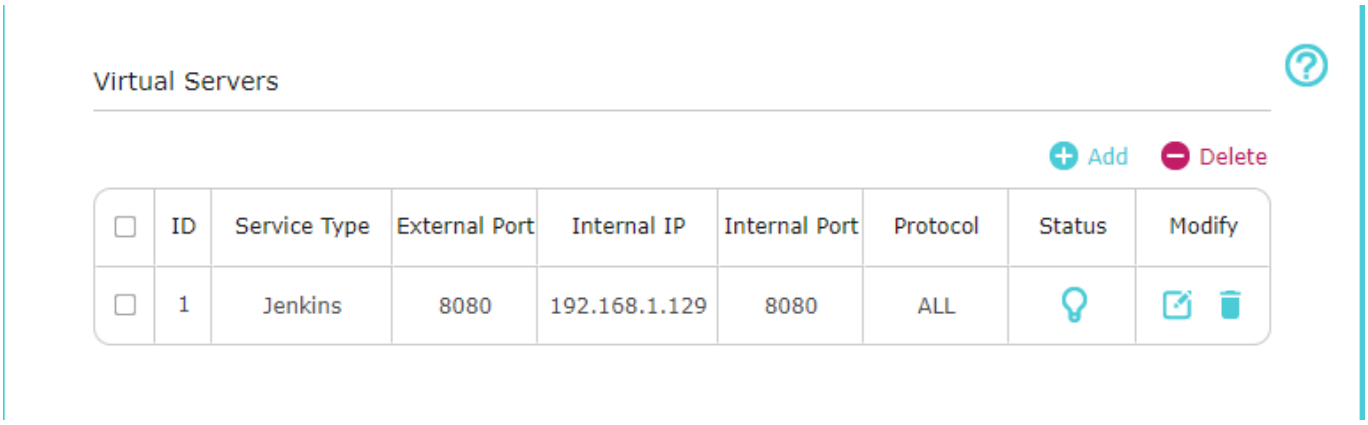
- Save this token for the Jenkins configuration

Jenkins configuration

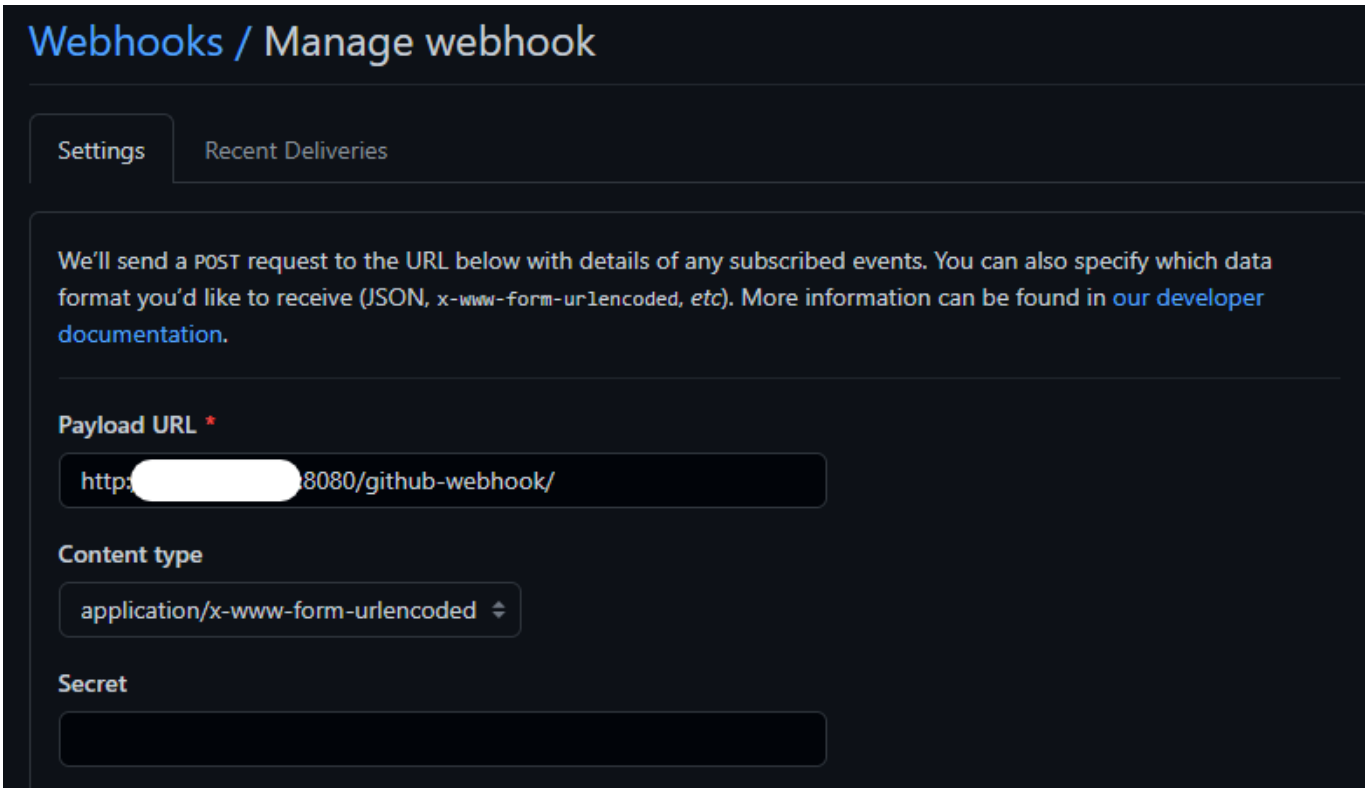
- Multi-stage pipeline project with GitHub source



- Added port forwarding on my router and pointed the github webhook at my public IP

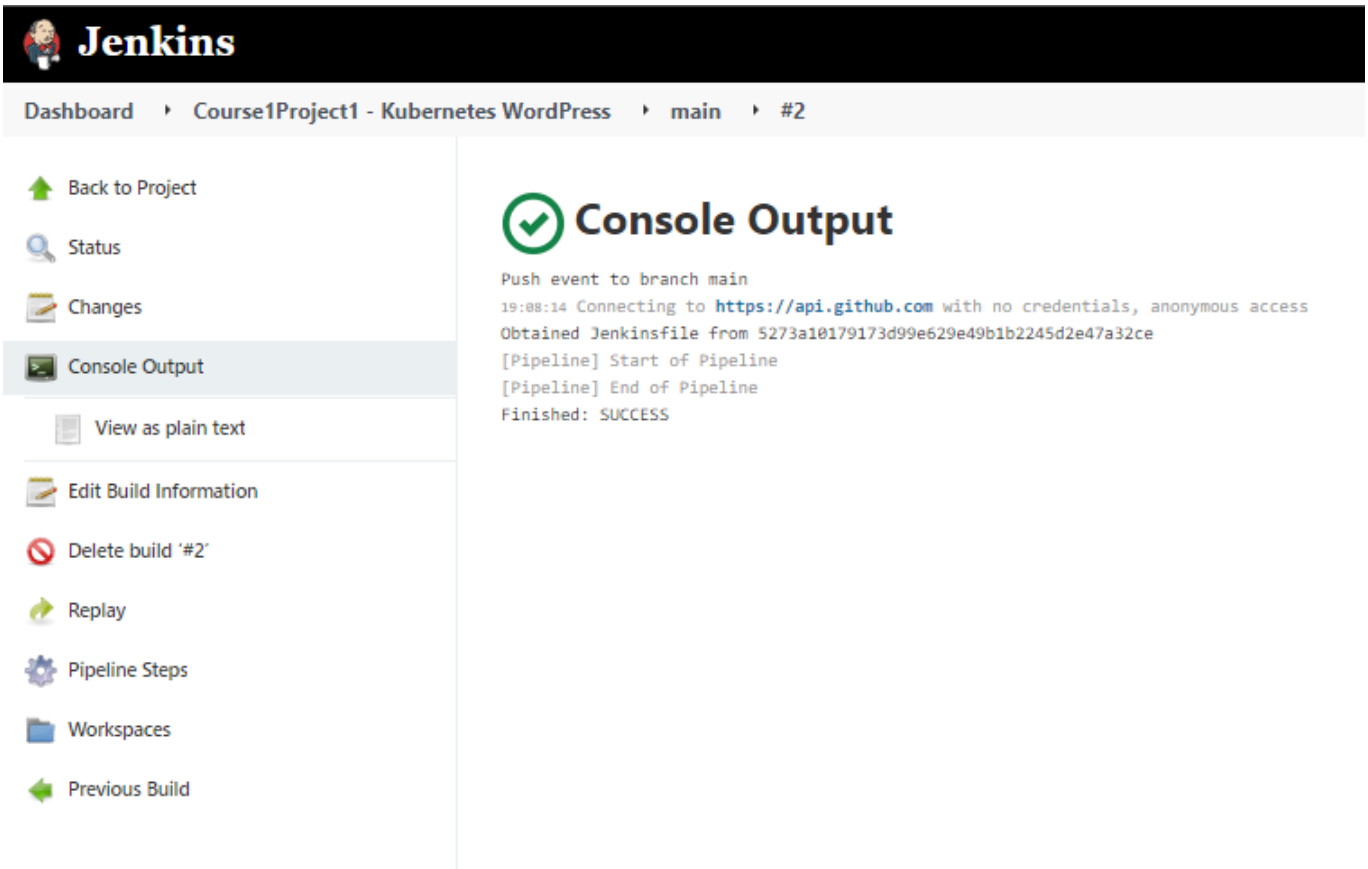


- Added webhook into my github repo to trigger remote build



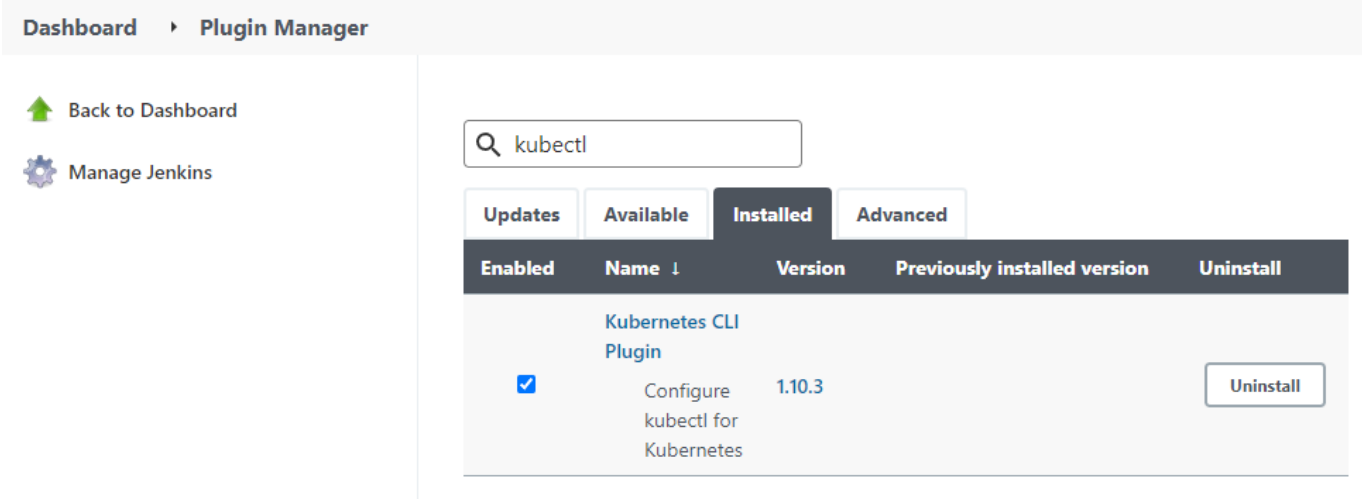
Test configuration

- We know this was successful because the console output shows `Push event to branch main` showing that it was a webhook triggered from a push that triggered the build

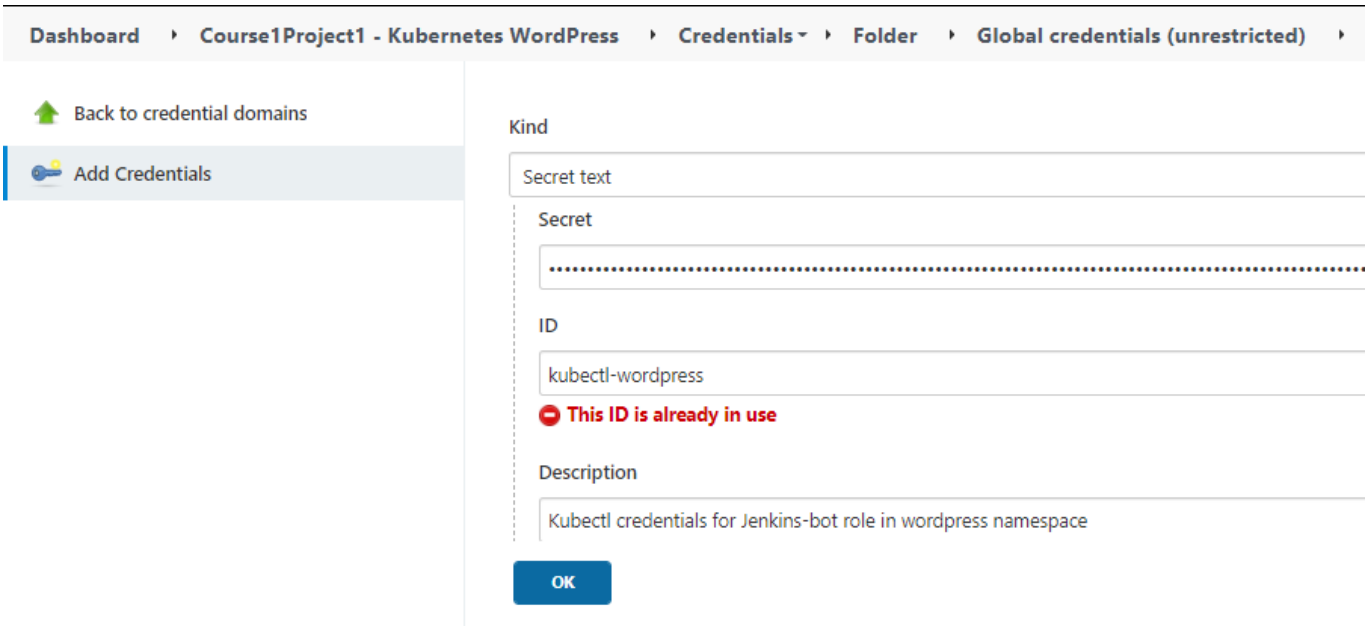


Kubectl Plugin Configuration

- Install the kubernetes plugin into Jenkins from the marketplace



- Configure the credential needed to access cluster
    - This is the secret from the [K8s secret](#k8s-configuration-for jenkins)



Create the Jenkinsfile

- The objective of the Jenkinsfile is to deploy the kubernetes manifests to the cluster on build

```
pipeline {
   agent any

   stages {

      // Execute when branch = 'main'
      stage("Main Branch deployment") {
        when {
              branch 'main'
        }
         steps {
            withKubeConfig([
               credentialsId: 'kubectl-wordpress',
               serverUrl: 'https://devops2:6443',
```

```
                    namespace: 'wordpress'
                    ]) {
                    sh 'kubectl apply -f ./kubernetes'
                }
            }
        }

    }
}
```

- There's a single stage with the name `"Main Branch Deployment"`
    - This only runs when on the main branch becaues of the block

    ```
    when {
      branch 'main'
    }
    ```

- This uses the kubectl plugin which allow us to use `withKubeConfig`
    - We pass in the server's URL, the namespace (this is important because our binding is only within the wordpress namespace) and the credential for our Jenkins' service secret.
    - This sets the context that would normally be set in ~/.kube/config to allow us to run kubectl commands the same way we've been doing from the cli.
- Finally, we call apply on everything within the `kubernetes` directory which is where our `wordpress` and `mariadb` workload manifests are.

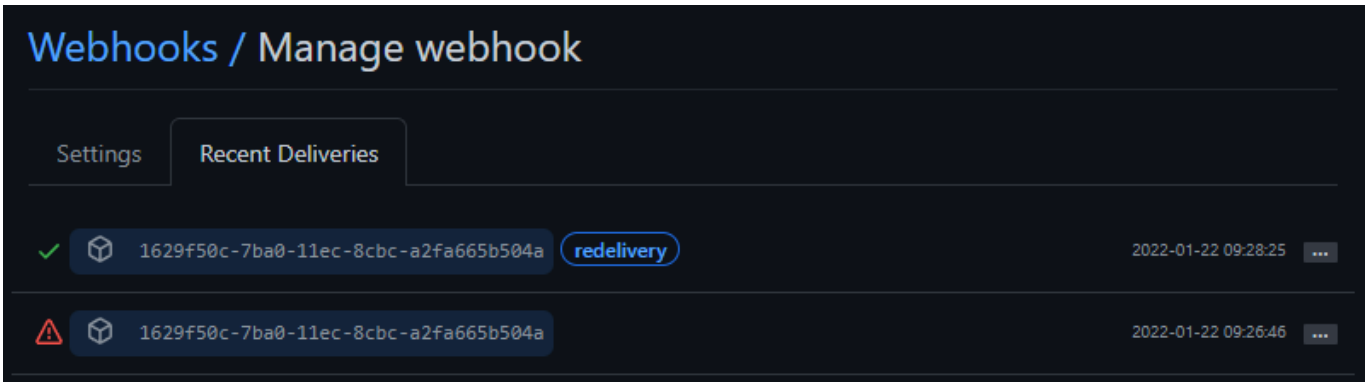
Testing workflow

- Commit add kubernetes files and commit changes

```
[khewitt@DevOps DevOpsK8sWordPress (main)]:: git push origin HEAD
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 6 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 44.18 KiB | 44.18 MiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To github.com:RedOneLima/DevOpsK8sWordPress.git
   68fab62..8792b29  HEAD -> main
```

- Check my GitHub to make sure the webhook triggered

    - Note, that my public IP changed since I originally set up my webhook, so that's why there was a failure and a successful retry



- Next, we'll verify that the Jenkins job triggered and sent our kubectl command to our cluster

Console Output

```
Push event to branch main
09:28:37 Connecting to https://api.github.com with no credentials, anonymous access
Obtained Jenkinsfile from 8792b299aa8790fc057e4413fadd429c4db82efb
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/ect1_-_Kubernetes_WordPress_main
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: NONE
No credentials specified
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/ect1_-_Kubernetes_WordPress_main/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/RedOneLima/DevOpsK8sWordPress.git # timeout=10
Fetching without tags
Fetching upstream changes from https://github.com/RedOneLima/DevOpsK8sWordPress.git
 > git --version # timeout=10
 > git --version # 'git version 2.25.1'
 > git fetch --no-tags --force --progress -- https://github.com/RedOneLima/DevOpsK8sWordPress.git +refs/heads/main:refs/remotes/origin/main # timeout=10
Checking out Revision 8792b299aa8790fc057e4413fadd429c4db82efb (main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 8792b299aa8790fc057e4413fadd429c4db82efb # timeout=10
Commit message: "Updating documentation"
 > git rev-list --no-walk 68fab62288965d8527238b7aeec7ccf47424e80a # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Main Branch deployment)
[Pipeline] withKubeConfig
[Pipeline] {
[Pipeline] sh
+ kubectl apply -f ./kubernetes
secret/mariadb-pass created
persistentvolumeclaim/mariadb-pv-claim created
deployment.apps/wordpress-mariadb created
service/wordpress-mariadb created
persistentvolumeclaim/wp-pv-claim created
deployment.apps/wordpress created
service/wordpress created
[Pipeline] }
[kubernetes-cli] kubectl configuration cleaned up
[Pipeline] // withKubeConfig
```

- Finally, we'll check our cluster status and go to our deployed wordpress
    - Note, that our worpress service is on nodeport on 30000 which is where we'll point our browser to.

```
[khewitt@DevOps DevOpsK8sWordPress (main)]:: kubectl get all -n wordpress
NAME                                    READY   STATUS    RESTARTS   AGE
pod/wordpress-668b6f77c6-hbp82          1/1     Running   0          16m
pod/wordpress-mariadb-66484947bb-k5g66  1/1     Running   0          16m

NAME                        TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
service/wordpress-mariadb   ClusterIP   None          <none>        3306/TCP       16m
service/wordpress           NodePort    10.43.72.85   <none>        80:30000/TCP   16m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wordpress           1/1     1            1           16m
deployment.apps/wordpress-mariadb   1/1     1            1           16m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/wordpress-668b6f77c6           1         1         1       16m
replicaset.apps/wordpress-mariadb-66484947bb   1         1         1       16m
```