Week 6

This week we shall make our Jenkins configuration more realistic by implementing triggers and branches.

Thus far, we have run Jenkins pipelines manually on our own. A more realistic scenario is for them to be **triggered** when something happens. For example, when someone proposes a change (a **pull request** in github), or when someone makes a change (**push**es new code to the repository.)

Triggering can be in three forms.

External: Github notifies our Jenkins process when such a change occurs. It will have to somehow communicate with our jenkins instance, which is running on a kubernetes pod on our laptop. This will have to be done in a completely secure way. In Lab 1, we will set up a "webhook" to that end.

Polling: Jenkins periodically checks for changes in github.

Scheduled: Jenkins periodically wakes up and runs.This is good for nightly builds with random parameters.

Generally, developers work in their own feature branch. They test changes in their own branch. This keeps the number of changes to the master under control. When their feature is tested and ready, they merge their branch with the master branch.

In the intervening time, they keep their branch up to date with the master branch, so it does not fall too far behind. They periodically **merge** or **rebase** their branch with the master branch (you should understand the differences between those terms.)

Lab 1

Push

Setting up webhook to trigger jobs on push and pull requests

Follow instructions from:

https://webhookrelay.com/blog/2018/12/18/webhooks-to-jenkins-on-kubernetes/

1. Create a webhook relay account
2. Create a token and secret
3. Use the token and secret to create a "kubernetes" secret
4. Make a jenkins bucket on the webhook relay
5. Configure github to use the webhook
    a. Define when you want to be invoked by github (push, pull request)
    b. Point the webhook to the relay
6. Change your kubernetes deployment to include the sidecar (webhook relay)
7. Create a pipeline that uses a webhook
8. Test by pushing to the repository

Steps 1-5 are the same as in the blog. On step 5, configure notifications for push and pull requests.

Change your kubernetes deployment

Modify your deployment file to include the new webhook "sidecar" container (in red). This file can also be downloaded from the umlF21 repository. Delete your old deployment. Doing so will delete your jenkins pod, but will not affect what you have saved in your pvc. So when you restart the Jenkins pod, it should read what you saved on your pvc.

```
kubectl delete deployment jenkins-deployment -n devops-tools
```

```
kubectl apply -f jenkins-full-deployment.yaml  -n devops-tools
```

When you look at the deployment, you should now see the pod contains two containers. It contains the jenkins container and it also now contains the webhook relay container.

```
kubectl describe deployment jenkins-deployment -n devops-tools
```

Periodic or Poll

What if you do not have the webhook mechanism?

What if you want to run nightly tests, because they are random, and the random generator creates different test scenarios each time?

You can configure Jenkins to periodically poll github for changes. If it detects a change, it will compile the code. Or you can have it periodically run the pipeline regardless of changes.

In the tab "build triggers", click on "build periodically". Then enter the cron job notation. The syntax is the same as cron jobs.

For example:

H/20 * * * *

Would build every 20 minutes.

Troubleshooting

If you have trouble getting **webhooks** working, you can look at logs at each point the webhook is sent:

- on github, settings->webhooks->"edit" webhook->Recent delivery, to show every webhook sent by github.
- on webhook relay: go to https://my.webhookrelay.com/logs .
- on the new "sidecar" container in the pod: `kubectl logs <jenkins pod> webhookrelayd -n devops-tools --follow`
- on jenkins: `kubectl logs <jenkins pod> jenkins -n devops-tools --follow`
- More Jenkins logs are in the jenkins container.
    - Get into `kubectl exec -it  <jenkins pod>  jenkins -n devops-tools-test -- /bin/bash`
    - Then search for "webhook" in directory `/var/jenkins_home/logs`

Lab 2

You should understand how to make a **pull request**. This is how new code is put up for review from your peers. Git can be configured to deny the ability to merge code until a successful review.

Pull request - branch flow

1. Make a branch
2. Push a change to the branch: `git push origin <branchname>`
3. Go to github, and create a pull request from <branch> <main>, the webhook should be triggered.
4. Wait for test to complete, then merge it

There are two ways to bring a git branch up to date with change in master. You should understand the difference between them: **merge** and **rebase**. Merge interleaves by time all the changes from the main branch. Rebase takes the changes in the branch and puts them over all the changes on master.

Run the following twice:

```
mkdir week6; cd week6
git init
echo 'first commit to main' > README.md
git add README.md
git branch -M main
git commit -m "first commit to main"
git checkout -b feature
echo 'my feature' > feature
git add feature
git commit -m "first change to feature"
git checkout main
echo 'second change to main' >> README.md
git add README.md
git commit -m "second change to main"
git checkout feature
echo 'second change to feature' >> feature
git add feature
git commit -m "second change to feature"
git merge main
# the second time, issue: git rebase main
git log
```

Look at the log. Then rerun the test, instead of git merge, issue git rebase main. Observe how the logs are different.

Multibranch

The multibranch pipeline has the ability to discover all the branches within a repository. Once it discovers branches, it can then perform different work depending on the branch. For example, a development branch may only do unit tests, and the production branch may do both unit tests and deployment.

Multibranch is the only pipeline type that proactively "seeks" new branches. Some good documentation is here. There is also a required video to watch.

You may run into difficulties using our class's configuration. Multibranch sends a lot of requests to github, which may in turn use up your "quota" of requests. This will start a ratelimiter, which blocks traffic to github.

The when statement (declarative syntax)

You can use the **when** clause to run steps depending on the branch name. Below is an example in "declarative" format. This works in both a multibranch or non-multibranch pipeline.

```
pipeline {
  agent {
    kubernetes {
      yaml '''
      spec:
        containers:
        - name: gradle
          image: gradle:6.3-jdk14
      '''
    }
  }
  stages {
        stage('debug') {
               steps {
                              echo env.GIT_BRANCH
                       echo env.GIT_LOCAL_BRANCH
               }
        }
        stage('feature') {
            when {
               expression {
                         return env.GIT_BRANCH == "origin/feature"
               }
             }
            steps {
               echo "I am a feature branch"
            }
        }
        stage('main') {
            when {
              expression {
                return env.GIT_BRANCH == "origin/main"
              }
            }
            steps {
               echo "I am a main branch"
            }
        }
    }
        .
```

```
# Deployment Config
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jenkins
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      securityContext:
            fsGroup: 1000
            runAsUser: 1000
      containers:
        - name: jenkins
          image: jenkins/jenkins:lts
          resources:
            limits:
              memory: "2Gi"
              cpu: "1000m"
            requests:
              memory: "500Mi"
              cpu: "500m"
          ports:
            - name: httpport
              containerPort: 8080
            - name: jnlpport
              containerPort: 50000
          livenessProbe:
            httpGet:
              path: "/login"
              port: 8080
            initialDelaySeconds: 90
            periodSeconds: 10
            timeoutSeconds: 5
            failureThreshold: 5
          readinessProbe:
            httpGet:
              path: "/login"
              port: 8080
            initialDelaySeconds: 60
            periodSeconds: 10
            timeoutSeconds: 5
            failureThreshold: 3
          volumeMounts:
            - name: jenkins-data
              mountPath: /var/jenkins_home
        - name: webhookrelayd
          image: "webhookrelay/webhookrelayd:latest"
          imagePullPolicy: IfNotPresent
          command: ["/relayd"]
          env:
            - name: KEY
              valueFrom:
                secretKeyRef:
                  name: whr-credentials
```

```
            key: key
      - name: SECRET
        valueFrom:
          secretKeyRef:
            name: whr-credentials
            key: secret
      - name: BUCKET
        value: "jenkins"
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
      requests:
        cpu: 50m
        memory: 64Mi
volumes:
  - name: jenkins-data
    persistentVolumeClaim:
        claimName: jenkins-pv-claim
```