Week 7

This week we will begin "acceptance testing". This involves running the software in a black box form to check if the business requirements have been met. So instead of testing a small part of the code as done in unit tests, the entire body of software itself is tested as a whole. Such tests simulate what actual users would be doing.

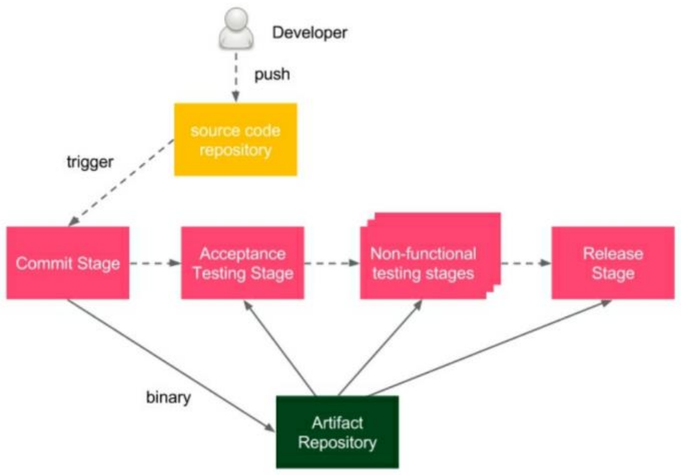
To do this, the software is run in a staging environment that resembles where the software will actually run.

We would like consistency: the software should be built exactly once and carry all its dependencies with it. Containers are the technology to accomplish this.

The containers need to be built and stored. To that end, we would use a *registry*. We will first use one in the cloud. You will need to set up a docker account for that (docker.com). Alternatively, you can create your own private repository that runs in kubernetes.

The repository works with big bulky files (containers). Each container should have a version. A convention to name containers is:

```
registry-name/image-name:<version>
```



Lab

You will need to create a docker account and then create a secret. Insert your username and password in the command below.

```
kubectl create secret docker-registry dockercred     --docker-server=https://index.docker.io/v1/
--docker-username=<your user>     --docker-password=<your pass>--docker-email=dlambrig@gmail.com
```

The pipeline will be creating a container and storing it in a repository. The problem here is running a "container in a container" (aka docker in docker or DinD) is very difficult. See links below to explore why.

https://medium.com/hootsuite-engineering/building-docker-images-inside-kubernetes-42c6af855f25

https://joostvdg.github.io/jenkins-pipeline/podtemplate-dind/

This lab takes the approach described here:

https://devopscube.com/build-docker-image-kubernetes-pod/

The approach uses a tool called Kaniko. It builds docker images inside kubernetes. Exactly what we need. Below is a way to quickly test it.

```
apiVersion: v1
kind: Pod
metadata:
  name: kaniko
spec:
  containers:
  - name: kaniko
    image: gcr.io/kaniko-project/executor:latest
    args:
    - "--context=git://github.com/scriptcamp/kubernetes-kaniko"
    - "--destination=<dockerhub-username>/kaniko-demo-image:1.0"
    volumeMounts:
    - name: kaniko-secret
      mountPath: /kaniko/.docker
  restartPolicy: Never
  volumes:
  - name: kaniko-secret
    secret:
      secretName: dockercred
      items:
        - key: .dockerconfigjson
          path: config.json
```

Some notes about the pipeline

The pipeline on the next page builds a gradle project, builds a container, and then pushes it to the docker repository.

In the yaml file, there is a shared pvc between the kaniko and the gradle container. You will need to create a pvc, in the same manner as in earlier labs.

In step 1, the build command will create a jar file.

```
./gradlew build
```

This creates a jar file. Copy it to the pvc shared with kaniko.

```
cp ./build/libs/calculator-0.0.1-SNAPSHOT.jar /mnt
```

In step 2, the jar file will then be turned into a container. This is done using a dockerfile. This dockerfile is created on the fly in the pipeline. It will look like this:

```
FROM openjdk:8-jre
COPY ./calculator-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Run your container

At this point, your container should be in your docker repository. We can actually run the code that we have built. The two yaml files can be found in the Chapter08 github folder.

```
kubectl apply -f hazelcast.yaml
kubectl apply -f calculator.yaml
```

Then go to your browser and have the program add 3 to 4. Enter

```
http://127.0.0.1:30499/sum?a=3&b=4
```

Next week we will run the container in kubernetes and perform tests against it.

```
podTemplate(yaml: '''
    apiVersion: v1
    kind: Pod
    spec:
      containers:
      - name: gradle
        image: gradle:6.3-jdk14
        command:
        - sleep
        args:
        - 99d
        volumeMounts:
        - name: shared-storage
          mountPath: /mnt
      - name: kaniko
        image: gcr.io/kaniko-project/executor:debug
        command:
        - sleep
        args:
        - 9999999
        volumeMounts:
        - name: shared-storage
          mountPath: /mnt
        - name: kaniko-secret
          mountPath: /kaniko/.docker
      restartPolicy: Never
      volumes:
      - name: shared-storage
        persistentVolumeClaim:
          claimName: jenkins-pv-claim
      - name: kaniko-secret
        secret:
            secretName: dockercred
            items:
            - key: .dockerconfigjson
              path: config.json
''') {
  node(POD_LABEL) {
    stage('Build a gradle project') {
      git 'https://github.com/dlambrig/Continuous-Delivery-with-Docker-and-Jenkins-Second-
Edition.git'
      container('gradle') {
        stage('Build a gradle project') {
          sh '''
          ./gradlew build
          mv ./build/libs/calculator-0.0.1-SNAPSHOT.jar /mnt
          '''
        }
      }
    }

    stage('Build Java Image') {
      container('kaniko') {
        stage('Build a gradle project') {
          sh '''
          echo 'FROM openjdk:8-jre' > Dockerfile
          echo 'COPY ./calculator-0.0.1-SNAPSHOT.jar app.jar' >> Dockerfile
          echo 'ENTRYPOINT ["java", "-jar", "app.jar"]' >> Dockerfile
          mv /mnt/calculator-0.0.1-SNAPSHOT.jar app.jar
          /kaniko/executor --context `pwd` --destination dlambrig/hello-kaniko:1.0
          '''
        }
```

```
            }
        }

    }
}
```