Tutorial

Jenkins

In this tutorial you will build on the calculator project you worked on in the last tutorial. You will add a Dockerfile and some more steps in the Jenkinsfile to make Jenkins automatically test the project and then deploy the application to a Docker image if it passes all the tests.

Setting up the Jenkins Continuous Delivery Environment

At the end of this Tutorial, Jenkins should:
- Automatically build your project when you push new code to the Github repository
- Automatically Clean, Build, Test short tests, Test long tests, and Package your program
- Automatically Build a Docker image from a Dockerfile containing your Java files
- Automatically Deploy Image to DockerHub and remove the docker image from server after it is pushed.
- Automatically send you an email if the build fails and provide a link to the broken build.

1. Create a docker hub repository which will contain the docker image for this java project
   a. Sign in : at the docker website https://hub.docker.com/
   b. Click : create repository -> give your repository a name that is descriptive of the calculator app
2. Start Jenkins Instance
   a. Navigate to the AWS EC2 Dashboard
   b. Right click your Jenkins Instance
   c. Click Instance State -> Start

Note: *There can be a cost associated with using EC2. AWS allows certain services to be run within a "free tier" for the first year of use of your AWS account, so most students can run an EC2 instance for free. However, if you have used up your free tier eligibility (by having created an AWS account more than a year ago, possibly for CS 260 or another class) you could be charged. However, the charges are small. If you create a EC2 t2.micro server for use in this lab, and leave it running for an entire week, you will be charged approximately $1.95 if you are not still eligible for the free tier. We recommend creating an EC2 t2.micro instance as you do this pre-class assignment and then stopping (but not terminating) your instance when you are done. You can then restart it when you need for the next two weeks when you do the two Jenkins tutorials for this class.*

3. Update Webhook
   a. If you stopped your instance from the pre-class assignment and restarted it , you will need to update the IP address that Github is using.

b. Follow the Webhook steps again in the pre-class assignment, but use the current IP address for your AWS EC2 instance.
4. Open up the calculator project you used for the previous Jenkins tutorial
   a. We will be building on this project
5. **Edit Jenkinsfile**
   a. **Environment** -
      We are going to add some variables that will be used in the pipeline.
      Add the following at the top of your existing Jenkinsfile but still within the pipeline{} scope:

```
environment {
        registry = "Your_Dockerhub_Username/Your_Dockerhub_Repository_Name"
        registryCredential = 'dockerhub'
        dockerImage=''
}
```

1. *registry* - Stores the location of the docker hub repository you created for this project.
2. *registryCredentials* - Stores the ID you gave the docker credential in the Jenkins Global credentials
3. *dockerImage* - Stores the identifier for the docker image that will be created by Jenkins.

   b. **Docker steps**
      We are going to add pipeline script to the existing Jenkinsfile to add steps that will build, deploy, and remove docker images.
      i. **Building a Docker image** - Add the following to your Jenkinsfile after the Package Step:

```
stage ('Package') {
    steps {
        sh 'mvn package'
        archiveArtifacts artifacts: 'src/**/*.java'
        archiveArtifacts artifacts: 'target/*.jar'
    }
}

stage ('Building image') {
    steps {
        script {
            dockerImage = docker.build registry +
":$BUILD_NUMBER"
        }
    }
}
```

1. *docker.build* - Uses the docker installed on your system to build an image based on the Dockerfile found in the home directory of your Jenkins workspace. It returns the image's identifier.

2. *registry* is the repository you defined at the top of the Jenkinsfile in the Jenkins environment
3. *":$BUILD_NUMBER"* - Gets the project's build number from Jenkins
4. *dockerImage* - Stores the build identifier, registry info, and the build number. We need this to be able to identify this specific docker image later on in the tutorial.

ii. **Deploy image** - We will now write the script to push the image we just created to the docker hub repository we created at the beginning of the tutorial. Add the following to your Jenkinsfile after the "Building image" stage:

```
stage ('Deploy Image') {
    steps {
        script {
            docker.withRegistry('', registryCredential) {
                dockerImage.push()
            }
        }
    }
}
```

1. **docker.withRegistry('',registryCredential)** will use the Jenkins docker credentials you defined to log into Docker.
2. **dockerImage.push()** uses the image identifier you got in the"Build Image:" stage to push the built image to your Docker hub repository

iii. **Remove unused docker image** - We will now write a script to remove the docker image we just created inorder to keep our AWS instance files clean. Add the following to your Jenkinsfile after the 'Deploy Image' stage:

```
stage ('Remove unused docker image') {
    steps {
        sh "docker rmi $registry:$BUILD_NUMBER"
    }
}
```

1. This stage removes the image you just created and pushed from your local system. It identifies the image with the registry you defined in the pipeline environment and the BUILD_NUMBER that Jenkins automatically keeps track of it.

c. Email Notification steps -
   i. **Navigate to the bottom of the Jenkinsfile**
   ii. Add a post step **AFTER the STAGES** are defined. In this post step send a notification when the build fails.

```
post {
        failure{
          mail to: 'youremail@gmail.com',
          subject: "Failed Pipeline: ${currentBuild.fullDisplayName}",
          body: "Something is wrong with ${env.BUILD_URL}"
          }
   }
```

1. To understand this step, follow the documentation here : https://jenkins.io/doc/pipeline/tour/post/
2. The pipeline script in the Jenkinsfile should look like the screenshot below.

```
stage ('Remove unused docker image') {
    steps {
        sh "docker rmi $registry:$BUILD_NUMBER"
    }
}
}

post {
    mail to: 'johndoe@gmail.com',
        subject: "Failed Pipeline: ${currentBuild.fullDisplayName}",
        body: "Something is wrong with ${env.BUILD_URL}"
}

}
```

6. You can double check your Jenkinsfile with mine using this text file or just use mine and update the email, and docker credentials : https://docs.google.com/document/d/14xoR0F8h0F1zZYuspi-Fpf8UodCZyp5NPQ9Qj4NFm8s/edit?usp=sharing
7. Set up Dockerfile
   a. Create a new file named: "Dockerfile" in your project directory
   b. Place the following text into your Dockerfile:

```
from openjdk
COPY ./src/main/java/*.java /
RUN javac /Calculator.java
ENTRYPOINT ["java"]
```

   1. **From openjdk** - Creates an image based on the openjdk image that already exists. It allows the image to run java.

2. **COPY <src> <dest> -** Copies all the .java files found in the project's srs/main/java folder into the home directory of the image
3. **RUN javac** - Compiles all .java files that you specify
4. **ENTRYPOINT** - Invokes the jvm, passing Main as the argument.

8. Start and Log into Docker
Unless you specifically set Docker to start when your ec2 instance starts, it will be off .
    a. Start docker by using the following command in your AWS Ec2 instance:
       ```
       sudo service docker start
       ```
    b. In order to push an image into docker, you need to sign into your Docker Account on your terminal. Use the following command to log into Docker before continuing;
       ```
       docker login -u=<your docker hub username> -p <your docker hubpassword>
       ```

9. Navigate to Jenkins in your browser, click on the calculator Jenkins job, and navigate to "configure"
    a. Make sure that the **build trigger is set to github hook trigger**
        i. This allows Github to trigger a Jenkins build.
    b. Make sure the pipeline definition is set to pipeline script from SCM.
    c. Make sure the github repository url is correct
    d. The configuration dashboard should look similar to the screenshot below

e. Jenkins should now run every time you push changes to the repository
f. **Commit and push your project,** make sure the changes you made to your Jenkinsfile and Dockerfile are committed and pushed.
g. If your Github webhook was set-up correctly, your project should have been built automatically. It should look like the screenshot below
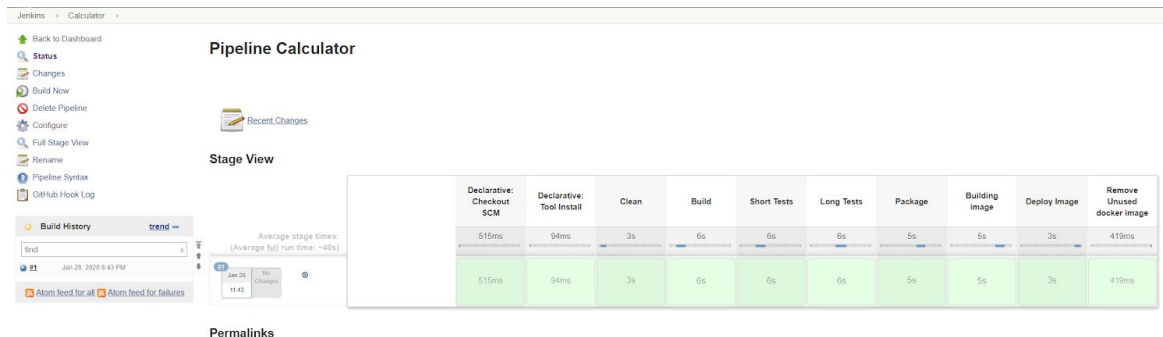
h. If other errors appear , check the build's console output to find the error
   i. The console output can be found on the left menu , as seen by the Screenshot below
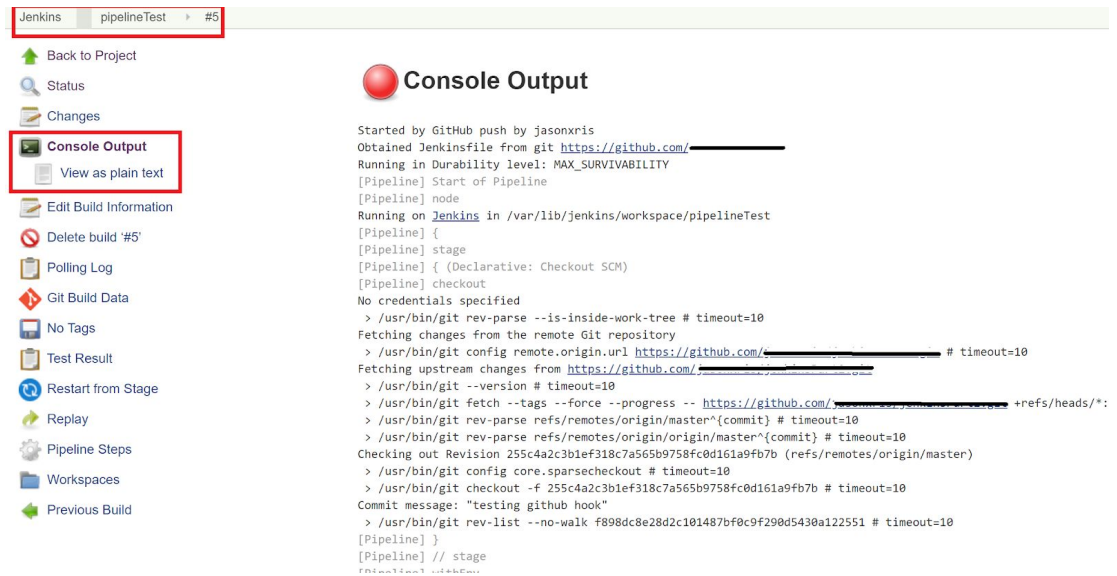
10. Purposely fail a pipeline step in order to test the email notification
   a. In the Calculator class **add anything that will break the program.**
      i. **Commit and push the code to Github**
      ii. Jenkins should automatically start building the code and send an email notifying you that the project failed.
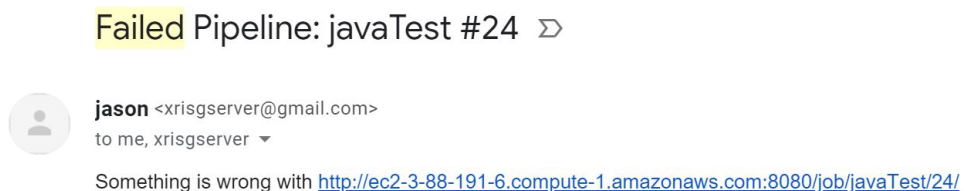   iii. I broke the project by adding random text above the constructor

   b. **Take a screenshot of the email that Jenkins sends you, it should look like the screenshot below**

      i. Remove the code you added to break the build. **Commit and Push**

Main.java Specifications

Jenkins should now be set up to run as a Continuous Deployment Environment. Everytime you push code to your github repository, Jenkins will build the project, test it, and deploy it as a docker image if it passes all the tests. As long as you have tests written for the program, there will always be a functional and updated docker image of your application.

We will now **Create and implement a Main.java** to practice developing code in a CD environment. The Main.java file will contain a simple text interface for the Calculator.

After creating your Main.java, **Update your Dockerfile** by Adding Main.java to the RUN command and ENTRYPOINT of your docker file, as see below.

```
from openjdk
COPY ./src/main/java/*.java /
RUN javac /Main.java /Calculator.java
ENTRYPOINT ["java", "Main"]
```

Requirements

Main.java
- Must be able to run the add, subtract, multiply, divide, fibonacciNumberFinder, and intToBinaryNumber in the terminal.
- You can assume that all input will be perfectly formatted. No need to account for null input or bad formatting.
- You do **NOT** need to support the createUniqueID method
- Inputs do **NOT** need to be identical to my examples

Commit and Push your project when you are finished.

Example commands:

Input: add 1 3   *Output: 4*
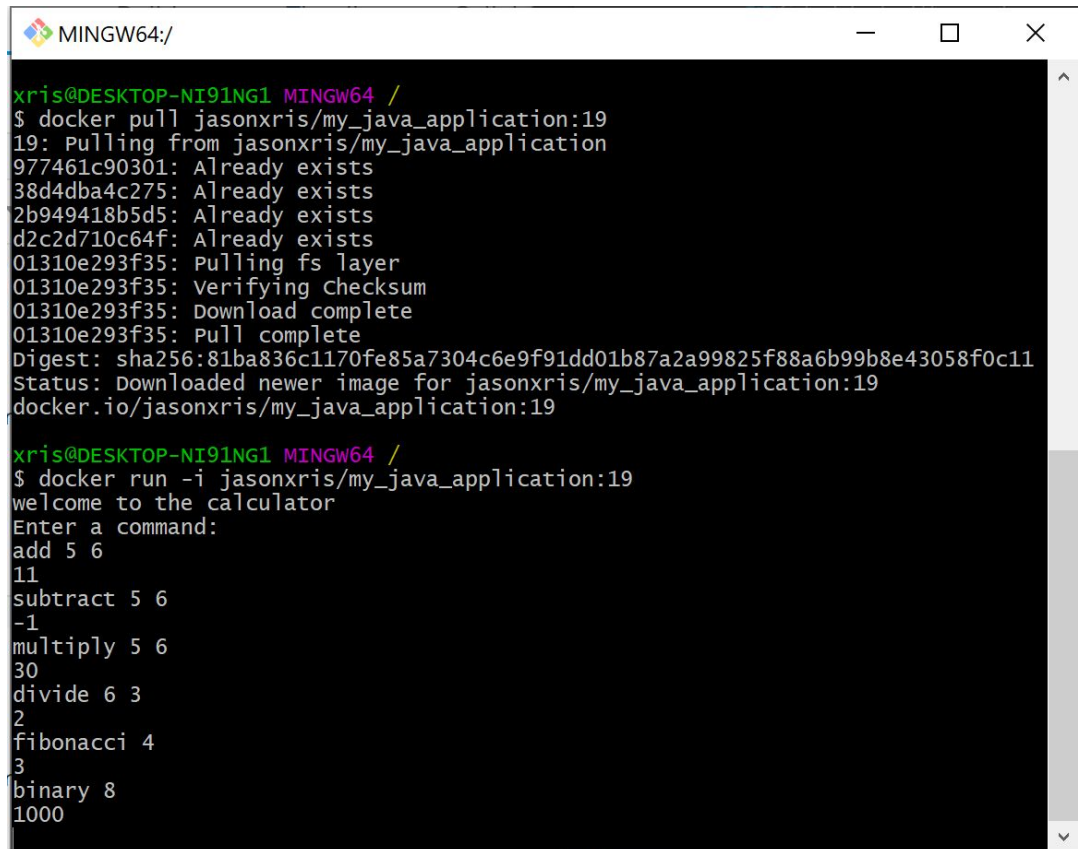
Input: binary 8   *Output: 1000*

An example of what the program should look like in the terminal **is shown below in the Test Docker Image section**.

Test Docker Image

By packaging your application in a docker image, your application should be able and ready to run on any system that has docker installed. We will test this by running the docker image that Jenkins created on your own computer.

Do the following commands on your OWN computer, NOT THE AWS EC2 server
1. Pull the image from your repository with the following command :
    a. *docker pull Docker-Username/Docker-Repository:BUILD_NUMBER*
        i. Make sure that the BUILD_NUMBER is equivalent to the most recent successful Jenkins build. You can double check this by logging into Docker Hub and checking for the most recent BuildTag.
2. Run the image with the following command :
    a. *docker run -i Docker-Username/Docker-Repository:BUILD_NUMBER*
        i. This command will run the image and allow your terminal to interact with the image
3. Verify that your application works
    a. Use the add, subtract , multiply, divide, fibonacci, and binary function
        i. **Take a screenshot of your terminal containing the docker pull , docker run, and the method commands.**
   b. It should look like the screenshot below

```
MINGW64:/                                                      —  □  ×

xris@DESKTOP-NI91NG1 MINGW64 /
$ docker pull jasonxris/my_java_application:19
19: Pulling from jasonxris/my_java_application
977461c90301: Already exists
38d4dba4c275: Already exists
2b949418b5d5: Already exists
d2c2d710c64f: Already exists
01310e293f35: Pulling fs layer
01310e293f35: Verifying Checksum
01310e293f35: Download complete
01310e293f35: Pull complete
Digest: sha256:81ba836c1170fe85a7304c6e9f91dd01b87a2a99825f88a6b99b8e43058f0c11
Status: Downloaded newer image for jasonxris/my_java_application:19
docker.io/jasonxris/my_java_application:19

xris@DESKTOP-NI91NG1 MINGW64 /
$ docker run -i jasonxris/my_java_application:19
welcome to the calculator
Enter a command:
add 5 6
11
subtract 5 6
-1
multiply 5 6
30
divide 6 3
2
fibonacci 4
3
binary 8
1000
```

What to turn in

- **1 screenshot** of the failure email from Jenkins
- **1 screenshot** of the created docker image being pulled, run, and using the add, subtract, multiply, divide, fibonacci, and binary methods
- **1 screenshot** of the Jenkins pipeline running on the AWS EC2 server
    - It must include AWS EC2 server ip in the URL field
    - It should look like the screenshot below:
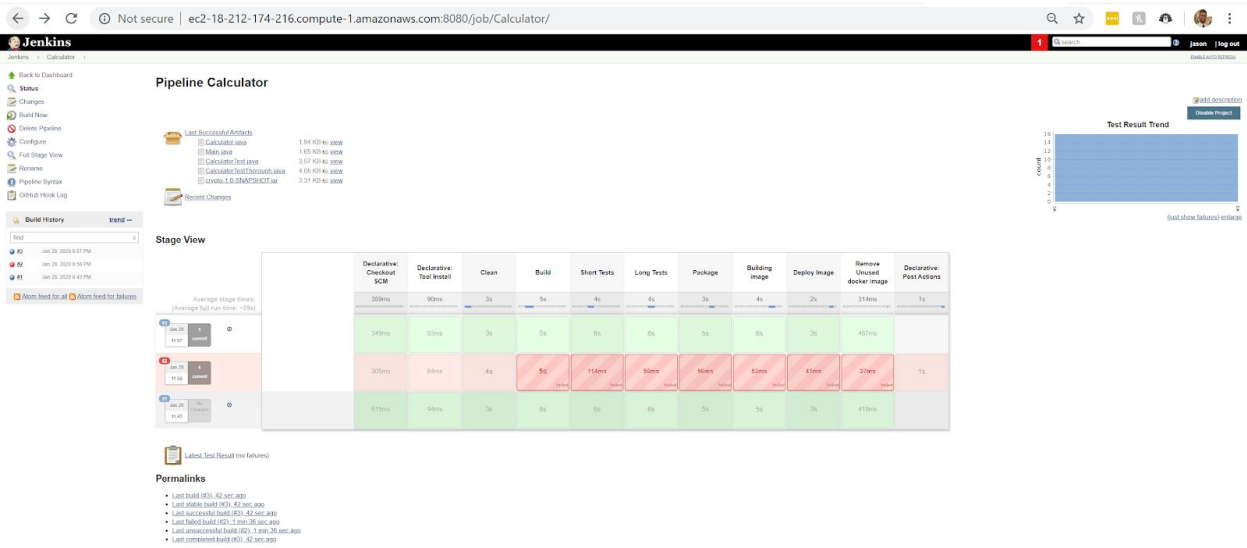
Finish the Lab

1. Navigate to your AWS EC2 dashboard
2. Stop or terminate your Jenkins Instance.