# ACIT 4850 – Lab 9 – Containerization

| | |
|---|---|
| **Instructor** | Mike Mulder (mmulder10@bcit.ca) |
| **Total Marks** | 10 |
| **Due Dates** | Demo due by the end of next class: • March 25th for Set C • March 26th for Set B • March 28th for Set A |

## Applicable Requirements

- **(Existing) REQ1180** – The Enterprise Development Environment shall support an Artifact Management capability. This Artifact Management capability will be DockerHub.
- **(Existing) REQ1190** – The Enterprise Development Environment Continuous Integration capability shall retrieve and publish artifacts from an Artifact Repository.
- **(New) REQ1200** – The Enterprise Development Environment shall use containerization to package applications for deployment to a target environment.
- **(New) REQ1230** – The Enterprise Development Environment shall hide and encrypt any passwords or secrets used in the pipeline builds.

## Group Work

You will be working on the same Azure cloud environment as the previous lab, shared with your Lab partner (if applicable). This lab will be done together with your partner.

You will need the following three/four applications running for this lab:

- Apache
- GitLab
- Jenkins
- SonarQube (Optional)

## Part 1 – Setup Jenkins to Build Docker Images

Login to your Jenkins VM using ssh.

If you haven't already done so, update your user so you can run docker without sudo. Here are some instructions: https://github.com/sindresorhus/guides/blob/master/docker-without-sudo.md

We want to be able to run docker commands on our Jenkins master which is running as a docker container. Add the following to the dockerfile for your Jenkins using USER ROOT:

```
RUN apt-get update && \
apt-get -y install apt-transport-https \
    ca-certificates \
    curl \
    gnupg2 \
    software-properties-common && \
    curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; \
    echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
    add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
    $(lsb_release -cs) \
    stable" && \
apt-get update && \
apt-get -y install docker-ce
RUN apt-get install -y docker-ce
RUN usermod -a -G docker jenkins
```

Stop and remove your running Jenkins container.

Rebuild the Docker image for Jenkins.

Run your Jenkins image as a container, but add the following (in bold) to the run command:

```
docker run -p 8080:8080 -p 50000:50000 --restart always --name=jenkins-master --mount
source=jenkins-log,target=/var/log/jenkins --mount source=jenkins-
data,target=/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock -d myjenkins
```

This will allow us to run Docker commands in our build pipelines on the host VM.

## Part 2 – Create Your DockerHub Account and Repositories

We could use SonaType Nexus as our Docker Artifact Respository. However, we will be using the cloud based DockerHub as our Artifact Repository for Docker images that we build in our pipelines just to diversify our experience.

Sign up for a DockerHub account at https://hub.docker.com/

Create three public repositories (Create Repository button):

- samplejava
- point
- carlot

Notes:
- Set the Visibility to Public (since Private repos are usually paid)
- Leave the optional Build Settings as-is (we are doing our own builds in Jenkins)

Add an access token for your account:

- Select your username in the top right corner
- Select Account Settings from the drop down menu
- Select the Security option on the left side
- Select New Access Token
    - Give the access token a name

    - Copy and store the generated token in a safe place.
    - When interacting with your DockerHub repos, your username is your login username and your password is the generated token.

**Part 3 – Add a Credential for DockerHub to Jenkins**

Make sure you have the Credentials Binding Plugin installed in Jenkins (Manage Jenkins -> Plugin Manager). If not, install it now.

In Jenkins navigate to:

- Manage Jenkins
- Manage Credentials
- Click on the (global) domain
    - You will get to the page titled: Global Credentials (unrestricted)
- Add Credentials (left side)
- Select:
    - Kind – Secret Text
    - Scope – Global
    - Secret – Your DockerHub Access Token
    - ID – DockerHub
- Select OK to save the credential

**Part 4 – Update the Java Sample Pipeline**

We won't use any Docker plugins for our pipeline, we will interact directly with Docker commands.

Add a file called dockerfile to the root level of your SampleJava repository with the following content:

```
FROM openjdk:11

WORKDIR /

COPY target/my-app-1.0-SNAPSHOT.jar /

CMD java -jar my-app-1.0-SNAPSHOT.jar
```

Edit your Jenkinsfile for your Java Sample repository. Add the Package stage after the Deploy stage:

```
stage('Package') {
    steps {
        withCredentials([string(credentialsId: 'DockerHub', variable: 'TOKEN')]) {
            sh "docker login -u '<username>' -p '$TOKEN' docker.io"
            sh "docker build -t myapp:latest --tag <username>/samplejava:myapp ."
            sh "docker push <username>/samplejava:myapp"
        }
    }
}
```

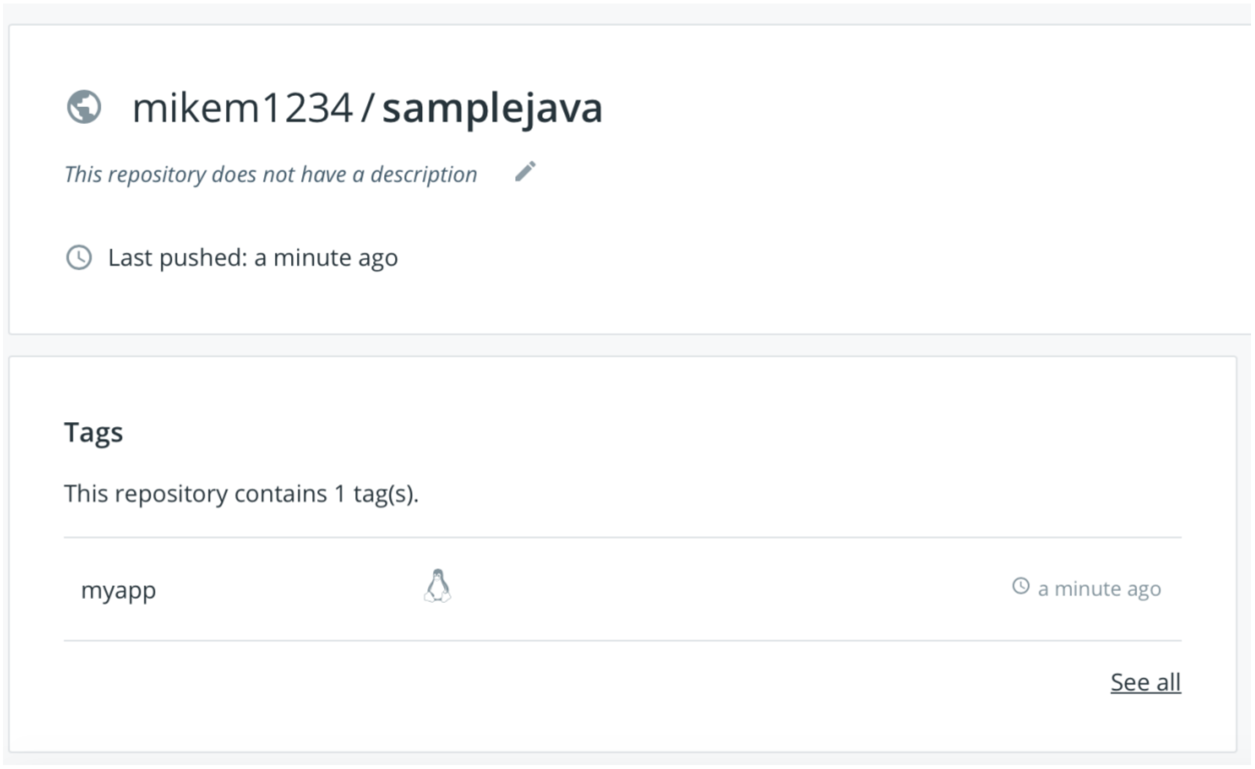
Here is a summary of what this Package stage is doing:

- withCredentials – retrieves the secret text from the credential called DockerHub and assigns it to the variable TOKEN, which you can reference as $TOKEN inside this block
- docker login – logs us in to our DockerHub account so we can push images
- docker build – Builds our image from our Dockerfile
- docker push – Pushes the built image (named myapp) to our DockerHub repository called samplejava

Effectively this stage is creating a Docker image for our application that can be used to run our application using Docker, Docker Compose or a Container Orchestration Engine.

**Note that using the withCredentials block hides our DockerHub access token in the Console Output for the build.**

Run your Java Sample pipeline from Jenkins and make sure the Docker image is pushed to your DockerHub repository. It should look something like this:


mikem1234 / samplejava

This repository does not have a description

Last pushed: a minute ago

Tags

This repository contains 1 tag(s).

myapp    a minute ago

See all

Take a screenshots of your DockerHub samplejava repository with your username and the tag showing.

**Part 5 – Update the Point and CarLot Pipelines**

Add a file called dockerfile to the root level of each of the Point and CarLot repositories with the following content:

```
FROM ubuntu:20.04

LABEL maintainer="<your email address>"
```

```
RUN apt-get update -y && \
    apt-get install -y python3-pip python-dev

# We copy just the requirements.txt first to leverage Docker cache
COPY ./requirements.txt /app/requirements.txt

WORKDIR /app

RUN pip install -r requirements.txt

COPY . /app

RUN cd /app && python create_tables.py

ENTRYPOINT [ "python3" ]

CMD [ "points_api.py" ]
```

Make sure to set the maintainer information in the dockerfile above.

Edit the python_build.groovy file in your CI functions repository. Add a parameter to the function defined in this file.

Change:

```
def call() {
```

To:

```
def call(dockerRepoName, imageName) {
```

Now when you call python_build as a function, you need to pass in the name of the DockerHub repository you want to push the image to and the name of the image you want to build.

Also add the Package stage just prior to the stage that zips the Python file. We are also adding a condition to this stage so it ONLY runs if the current branch is main.

```
stage('Package') {
    when {
        expression { env.GIT_BRANCH == 'origin/main' }
    }
    steps {
        withCredentials([string(credentialsId: 'DockerHub', variable: 'TOKEN')]) {
           sh "docker login -u '<username>' -p '$TOKEN' docker.io"
           sh "docker build -t ${dockerRepoName}:latest --tag
<username>/${dockerRepoName}:${imageName} ."
           sh "docker push <username>/${dockerRepoName}:${imageName}"
        }
    }
}
```

Notes:

- The when block causes this stage to be skipped unless the expression evaluation to true. In this case, this stage is only run if the pipeline is being run for the master branch.
- This stage will not be run for other branches because they are likely feature branches and, for our purposes, we do not want interim docker images pushed to DockerHub.

Update the Jenkinsfile for both the Point and CarLot repos as follows so that the correct arguments are passed to the pipeline.

Point:

@Library(‘ci_functions@master’) _

**python_build(‘point’, ‘pointapp’)**

CarLot:

@Library(‘ci_functions@master’) _

**python_build(‘carlot, ‘carlotapp’)**

Run your Point and CarLot pipelines from Jenkins and make sure the Docker images are pushed to your DockerHub repositories. Take a screenshots of your DockerHub point and carlot repositories with your username and the tag showing.

**<u>Part 6 – Testing Your Images</u>**

Login to your Jenkins VM using ssh if you aren’t already logged in.

Run the “docker images” command. This is the list of Docker images built on this VM.

You should see the images built by your Jenkins build in this local Docker repository.

Test this out by running them.

```
docker run myapp:latest
```

- This should output “Hello World!” to the console

```
docker run point:latest
```

- This should start a Flask server. Press Ctrl-C to stop it.

```
docker run carlot:latest
```

- This should start a Flask server. Press Ctrl-C to stop it.

**<u>Make sure you shutdown any Azure resources (i.e., the VM) to conserve your credits and free tier usage.</u>**

**Demo, Grading and Submission**

A demo of your lab against the applicable requirements which will determine your grade on the lab. All mandatory requirements must be met otherwise you will receive zero on the lab. You can re-demo the lab if you haven't met the mandatory requirements but you will lose 20% every week late.

| **Reqt.** | **Mandatory** | **Demo** | **Marks** |
|---|---|---|---|
| REQ1180 | Yes | • Show your three DockerHub repositories with published artifacts | 3 |
| REQ1190 | Yes | • Show your three CI Pipelines building and publishing Docker images<br>o Point Pipeline<br>o CarLot Pipeline<br>o Java Sample Pipeline<br>Verify that credentials are hidden in the console output. | 3 |
| REQ1200 | Yes | • Show that you can run each of the three Docker images built from your CI Pipelines on your Jenkins VM | 4 |

## <u>Submit the following screenshots to the Lab 9 dropbox on D2L to receive the marks for your demo.</u>

- A screenshot showing each of the three DockerHub repositories