Class practice:

Introduction:

Steps:

1. Git Operation:
    1. Create a new Git repo under your name:
    2. Pace into it the content of the following git repo: https://github.com/zivkashtan/course.git
    3. Commit your changes and push
2. In Jenkins (http://localhost:8080), create a new pipeline job called “class-work”
3. Create a simple pipeline job flow constructed of following stages:
    1. “Pull from scm”
    2. “build artifact”
    3. “Deploy artifact” please note, install the artifact into local tomcat (you may install tomcat locally). The deployment will basically copy the output artifact and restart the tomcat server. Please pay attention, before starting the tomcat make sure to modify the default port from 8080 to 9595.
4. Check & test, if working, continue,
5. Push to nexus
    1. Add the following to your pom.xml

```
<plugin>
   <groupId>org.sonatype.plugins</groupId>
   <artifactId>nexus-staging-maven-plugin</artifactId>
   <version>1.5.1</version>
   <executions>
      <execution>
         <id>default-deploy</id>
         <phase>deploy</phase>
         <goals>
            <goal>deploy</goal>
         </goals>
      </execution>
   </executions>
   <configuration>
      <serverId>nexus</serverId>
      <nexusUrl>http://localhost:8081/nexus/</nexusUrl>
      <skipStaging>true</skipStaging>
   </configuration>
</plugin>
```

    2. Add the following to setting.xml

```
<servers>

   <server>

      <id>nexus</id>

      <username>admin</username>

      <password>admin123</password>

   </server>

</servers>
```

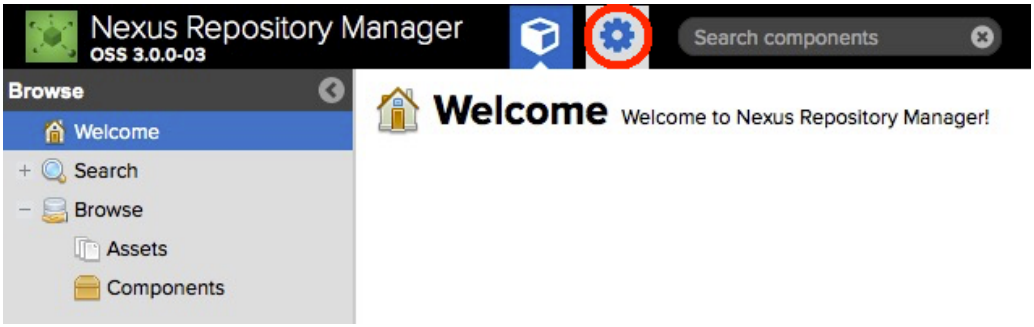
   3. Test it by running “mvn deploy”
   4. Commit your change and push
   5. Add to your pipeline a new stage, “push to nexus”, and test
6. Place your Jenkinsfile in your git repo, what modifications you need to perform in your job definition?
7. Dockerfile:
   1. Create a docker file definition, base your docker image of following image: **tomcat:8.0.20-jre8**
   2. Your artifact should be placed on following path: **/usr/local/tomcat/webapps/**
   3. Run docker build using: **“docker build -t shykes/myapp .”**
   4. Test your docker image by running it, note to mount it to the correct port (expose to the OS in 7575 port)
   5. Add the Docker file you created to your git repo, commit and push.
8. Pushing the docker image to artifactory (Nexus)
   1. Logging to the docker repo using: **“docker login -u admin -p admin123 172.16.4.101:8123”**
   2. Tagging the new docker image, example: docker tag tomcat:latest 192.168.1.72:8123/ivans_tomcat:latest
   3. Push the new docker image using”**docker tag tomcat:latest 172.16.4.101:8123/ivans_tomcat:latest**”
   4. Add the appropriate stage into your pipeline file.
9. Enjoy :)

Resources:

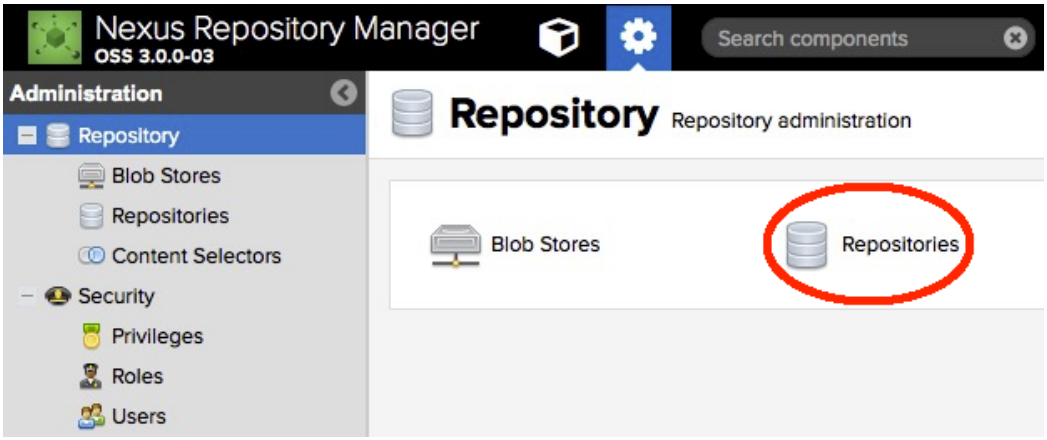
Create new docker registry in Nexus 3:

- Open the URL [Docker host IP address]:8081 in a web browser.
- Sign in to Nexus 3.
  Click the Sign In button in the upper right corner and use the username “admin” and the password “admin123”. If these credentials have changed, please check the Nexus 3 Docker image page in DockerHub.

- Click the cogwheel to go to the server administration and configuration section.

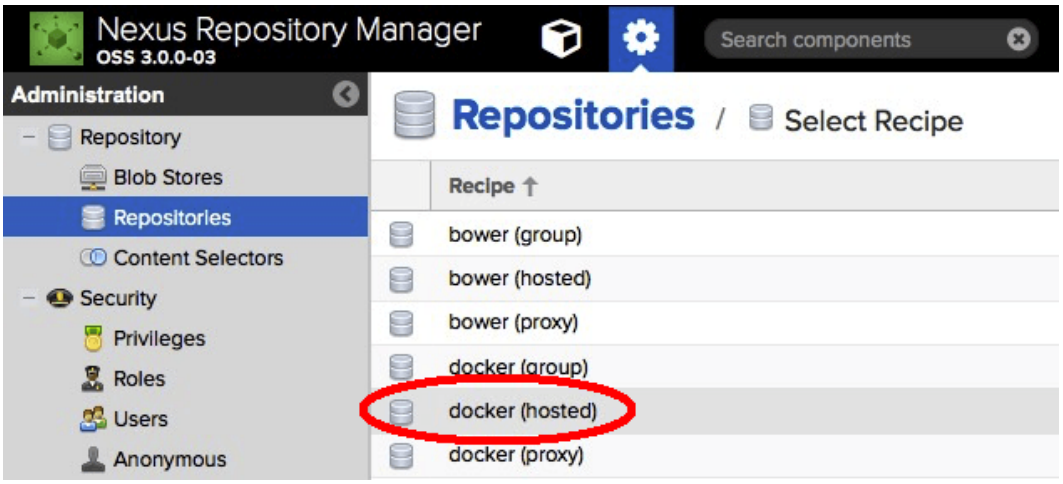


- Click Repositories.



- Click the Create repository button.

- In the list of repository types, select “docker (hosted)” as the type of the new registry.



- Configure the Docker repository.

  Give the repository a name – in my case it is “IvansDockerRepo”.

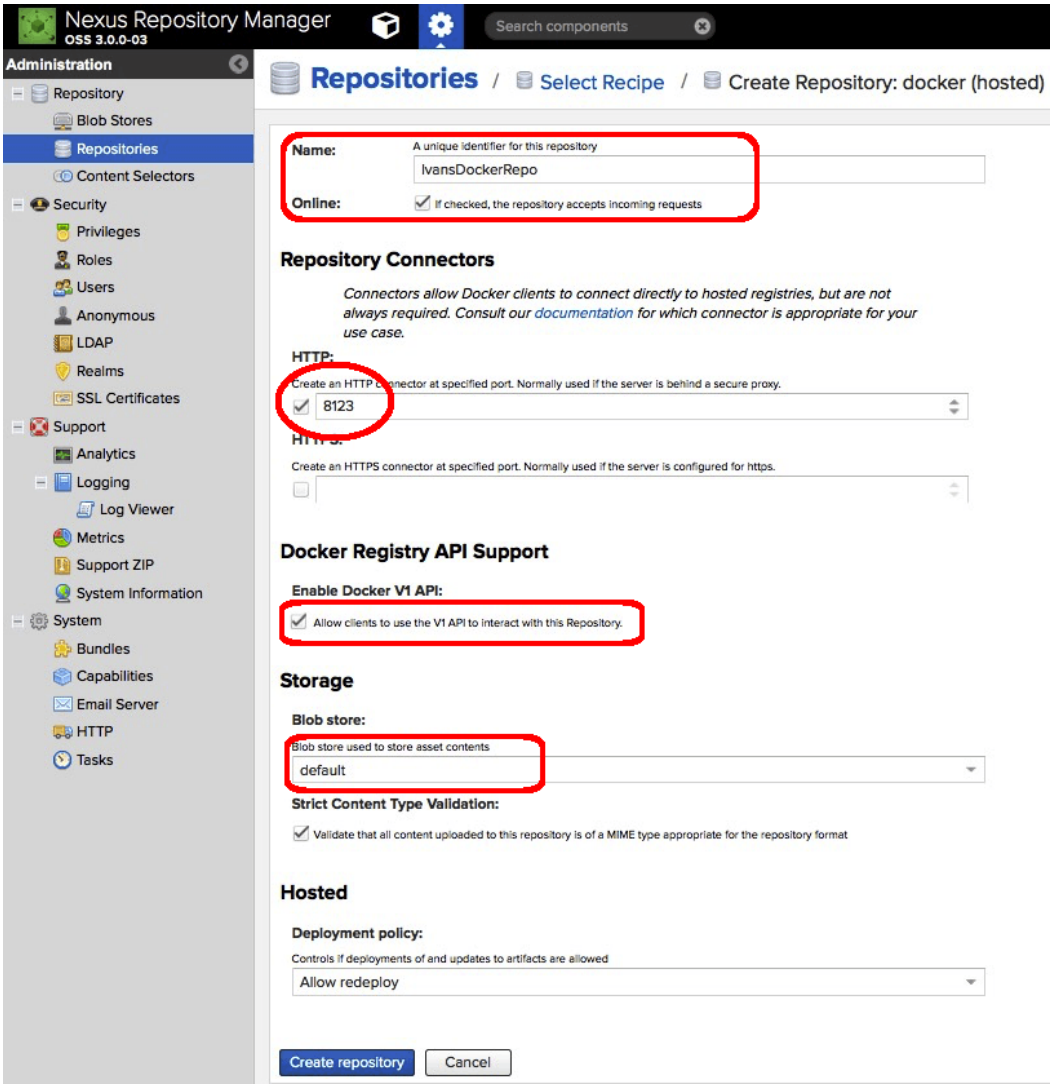
  Make sure that the Online checkbox is checked.

  Check the HTTP checkbox under Repository Connectors and enter the port number 8123.

  Check the Enable Docker V1 API checkbox.

  Select default under Blob store.

- Click the button Create repository.



The private Docker registry is now ready to be used.

Maven in Jenkins file:

https://jenkins.io/blog/2017/02/07/declarative-maven-project/

Git cheat sheet

Dockerfile instruction arguments (DockerFile)

Command	Desc
FROM	Sets the base image for subsequent
MAINTAINER	Sets the author field of the generated images
RUN	Execute commands in a new layer on top of the current image and commit the results

Command	Desc
CMD	Allowed only once (if many then last one takes effect)
LABEL	Adds metadata to an image
EXPOSE	Informs container runtime that the container listens on the specified network ports at runtime
ENV	Sets an environment variable
ADD	Copy new files, directories, or remote file URLs from into the filesystem of the container
COPY	Copy new files or directories into the filesystem of the container
ENTRYPOINT	Allows you to configure a container that will run as an executable
VOLUME	Creates a mount point and marks it as holding externally mounted volumes from native host or other containers
USER	Sets the username or UID to use when running the image
WORKDIR	Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD commands
ARG	Defines a variable that users can pass at build-time to the builder using --build-arg
ONBUILD	Adds an instruction to be executed later, when the image is used as the base for another build
STOPSIGNAL	Sets the system call signal that will be sent to the container to exit

More info:
https://docs.docker.com/engine/reference/builder/