

# Class practice:

Introduction:	2
Steps:	2
Resources:	3
Create new docker registry in Nexus 3:	3
Maven in Jenkins file:	5
Git cheat sheet	6
Dockerfile instruction arguments (DockerFile)	6

## Introduction:

## Steps:

1. Git Operation:
  1. Create a new Git repo under your name:
  2. Paste 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 :)

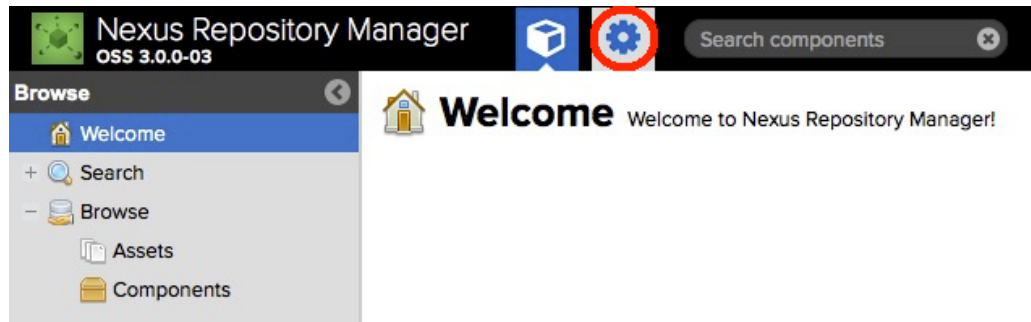
## Resources:

---

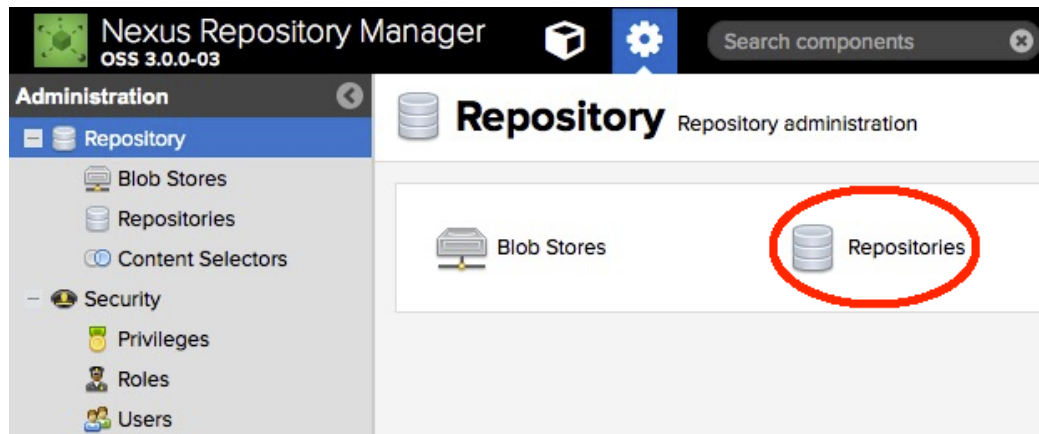
### Create new docker registry in Nexus 3:

- 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.

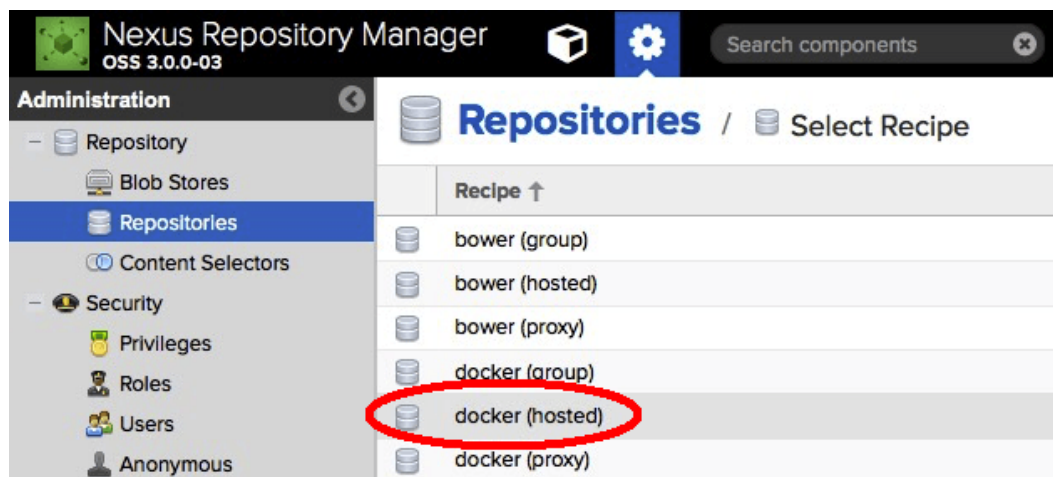


- Click Repositories.



- 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.

Nexus Repository Manager OSS 3.0.0-03

Administration

- Repository
- Blob Stores
- Repositories**
- Content Selectors
- Security
  - Privileges
  - Roles
  - Users
  - Anonymous
  - LDAP
  - Realms
  - SSL Certificates
- Support
- Analytics
- Logging
  - Log Viewer
- Metrics
- Support ZIP
- System Information
- System
  - Bundles
  - Capabilities
  - Email Server
  - HTTP
  - Tasks

Repositories / Select Recipe / Create Repository: docker (hosted)

**Name:** A unique identifier for this repository  
IvansDockerRepo

**Online:** ☒ If checked, the repository accepts incoming requests

**Repository Connectors**

Connectors allow Docker clients to connect directly to hosted registries, but are not always required. Consult our [documentation](#) for which connector is appropriate for your use case.

**HTTP:**  
Create an HTTP connector at specified port. Normally used if the server is behind a secure proxy.  
☒ 8123

**HTTPS:**  
Create an HTTPS connector at specified port. Normally used if the server is configured for https.  
☐

**Docker Registry API Support**

**Enable Docker V1 API:**  
☒ Allow clients to use the V1 API to interact with this Repository.

**Storage**

**Blob store:**  
Blob store used to store asset contents  
default

**Strict Content Type Validation:**  
☒ Validate that all content uploaded to this repository is of a MIME type appropriate for the repository format

**Hosted**

**Deployment policy:**  
Controls if deployments of and updates to artifacts are allowed  
Allow redeploy

Create repository Cancel

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



# GIT CHEAT SHEET

presented by Tower - the best Git client for Mac and Windows



### CREATE

Clone an existing repository

```
$ git clone ssh://user@domain.com:repo.git
```

Create a new local repository

```
$ git init
```

### LOCAL CHANGES

Changes files in your working directory

```
$ git status
```

Changes to tracked files

```
$ git diff
```

Add all current changes to the next commit

```
$ git add .
```

Add some changes in <file> to the next commit

```
$ git add -p <file>
```

Commit all local changes in tracked files

```
$ git commit -a
```

Commit previously staged changes

```
$ git commit
```

Change the last commit

*Don't amend published commits!*

```
$ git commit --amend
```

### COMMIT HISTORY

Show all commits, starting with newest

```
$ git log
```

Show changes over time for a specific file

```
$ git log -p <file>
```

Who changed what and when in <file>

```
$ git blame <file>
```

### BRANCHES & TAGS

List all existing branches

```
$ git branch -av
```

Switch HEAD branch

```
$ git checkout <branch>
```

Create a new branch based on your current HEAD

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch

```
$ git checkout -track <remote/branch>
```

Delete a local branch

```
$ git branch -d <branch>
```

Mark the current commit with a tag

```
$ git tag <tag-name>
```

### UPDATE & PUBLISH

List all currently configured remotes

```
$ git remote -v
```

Show information about a remote

```
$ git remote show <remote>
```

Add new remote repository, named <remote>

```
$ git remote add <shortname> <url>
```

Download all changes from <remote>, but don't integrate into HEAD

```
$ git fetch <remote>
```

Download changes and directly merge/integrate into HEAD

```
$ git pull <remote> <branch>
```

Publish local changes on a remote

```
$ git push <remote> <branch>
```

Delete a branch on the remote

```
$ git branch -dr <remote/branch>
```

Publish your tags

```
$ git push --tags
```

### MERGE & REBASE

Merge <branch> into your current HEAD

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>

*Don't rebase published commits!*

```
$ git rebase <branch>
```

Abort a rebase

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved

```
$ git add <resolved-file>
```

```
$ git rm <resolved-file>
```

### UNDO

Discard all local changes in your working directory

```
$ git reset --hard HEAD
```

Discard local changes in a specific file

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes)

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit ...and discard all changes since then

```
$ git reset --hard <commit>
```

...and preserve all changes as unstaged changes

```
$ git reset <commit>
```

...and preserve uncommitted local changes

```
$ git reset --keep <commit>
```

30-day free trial available at [www.git-tower.com](http://www.git-tower.com)

**TOWER**  
The best Git Client for Mac & Windows

## 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/)

<https://docs.docker.com/engine/reference/builder/>