# Install Jenkins and docker using Dockerfile

**Build a Jenkins and Docker server**

To provide our Jenkins server, we're going to build an image from a Dockerfile that both installs Jenkins and Docker.

```
FROM jenkins/jenkins:lts
MAINTAINER madhusudan reddy
USER root
RUN apt-get -qqy update; apt-get install -qqy sudo
RUN echo "jenkins ALL=NOPASSWD: ALL" >> /etc/sudoers
RUN wget http://get.docker.com/builds/Linux/x86_64/docker-latest.tgz
RUN tar -xvzf docker-latest.tgz
RUN mv docker/* /usr/bin/
USER jenkins
```

We see that our Dockerfile inherits from the **Jenkins/Jenkins:lts** image. The j **Jenkins/Jenkins:lts** image is the official Jenkins image maintained by their community on the Docker Hub. The **Dockerfile** then does a lot of other stuff. Indeed, it is probably the most complex Dockerfile we've seen so far. Let's walk through what it does.

We've first set the **USER** to **root**, installed the **sudo** package and allowed the **jenkins** user to make use of **sudo**. We then installed the Docker binary. We'll use this to connect to our Docker host and run containers for our builds.

Next we switch back to the **jenkins** user. This user is the default for the **jenkins** image and is required for containers launched from the image to run Jenkins correctly.

Next, let's create a directory, **/var/jenkins_home**, to hold our Jenkin's configuration. This means every time we restart Jenkins we won't lose our configuration

```
$ sudo mkdir -p /var/jenkins_home
$ cd /var/jenkins_home
$ sudo chown -R 1000 /var/jenkins_home
```

We also set the ownership of the **jenkins_home** directory to **1000**, which is the UID of the **jenkins** user inside the image we're about to build. This will allow Jenkins to write into this directory and store our Jenkins configuration.

Now that we have our **Dockerfile** and our Jenkins home directory, let's build a new image using the **docker build** command.

```
$ sudo docker build -t ybmsr/Jenkins-docker .
```

We've called our new image, somewhat unoriginally, **ybmsr/jenkins**. We can now create a container from this image using the **docker run** command.

```
$ sudo docker run --restart=always -u root -d -p 8082:8080 -p 50000:50000 \
-v /var/jenkins_home:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
--name jenkins \
ybmsr/jenkins-docker
```

----------comment----------

```
$ sudo docker run --restart=always -u root -d -p 8082:8080 -p 50000:50000 \
-v /var/jenkins_home:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /home/ec2-user/.docker/config.json: /home/ec2-user/.docker/config.json \
--name jenkins \
ybmsr/jenkins-docker
```

---

We can see that we've used the **-p** flag to publish port **8082** on port **8080** on the local host, which would normally be poor practice, but we're only going to run one Jenkins server. We've also bound port **50000** on port **50000** which will be used by the Jenkins build API.

Next, we bind two volumes using the **-v** flag. The first mounts our **/var/jenkins_home** directory into the container at **/var/jenkins_home**. This will contain Jenkin's configuration data and allow us to perpetuate its state across container launches.

The second volume mounts **/var/run/docker.sock**, the socket for Docker's daemon into the Docker container. This will allow us to run Docker containers from inside our Jenkins container.

```
$ sudo docker logs Jenkins
```

You can keep checking the logs, or run **docker logs** with the -f flag, until you see a message similar to:

**INFO: Jenkins is fully up and running**

Take note of the initial admin password, in our case:
**e9eef9d4a4e44741b0368877a9efb17c**

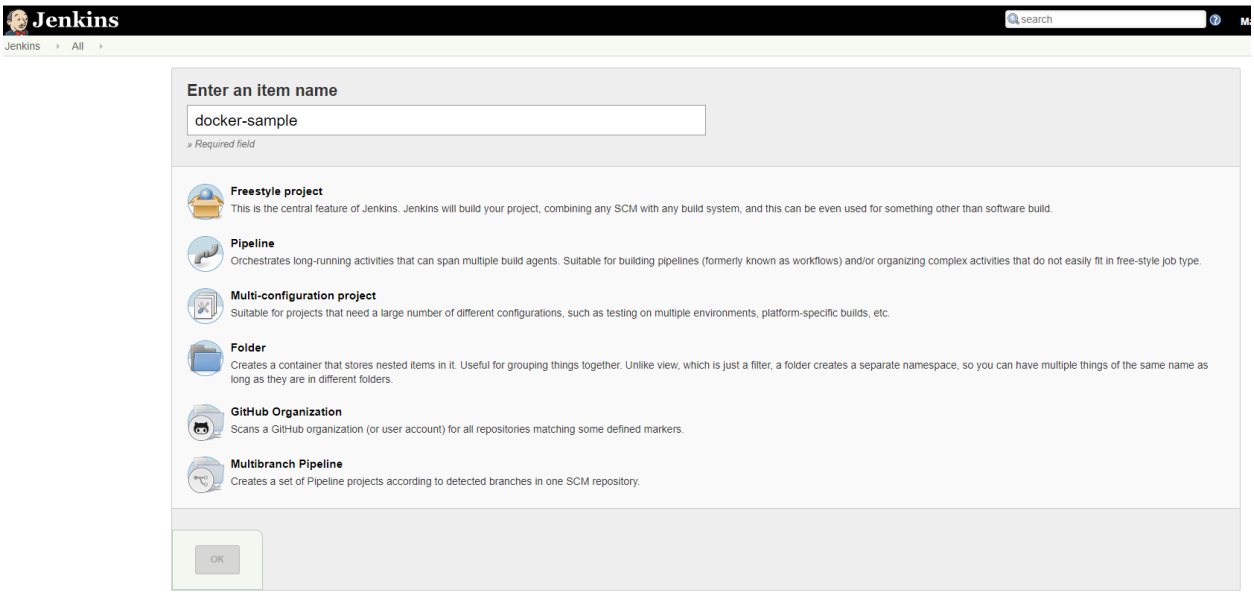
This is also stored in a file in the jenkins_home directory at:
**/var/jenkins_home/secrets/initialAdminPassword**

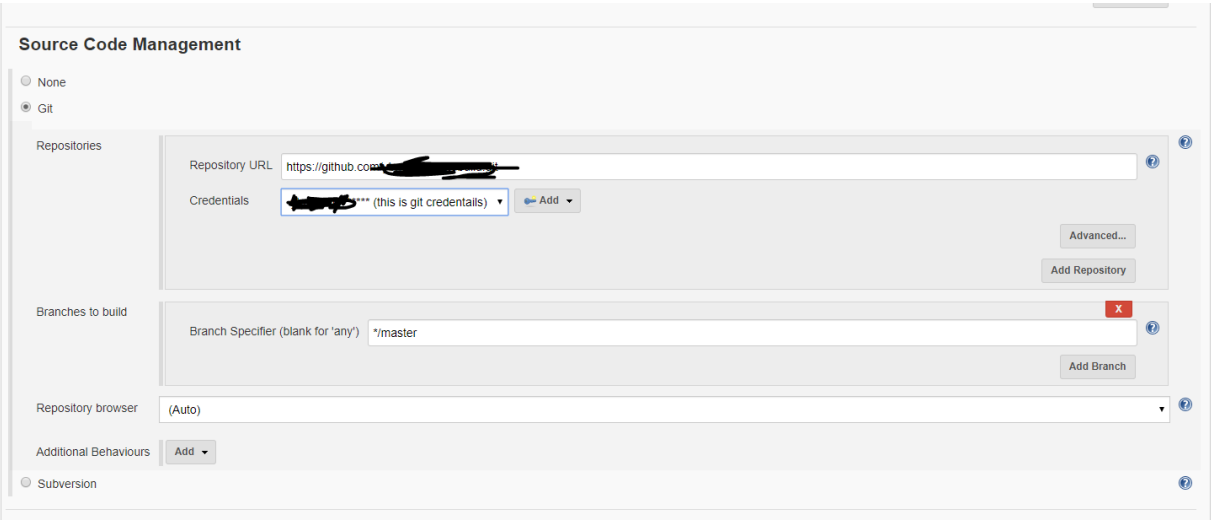
Finally, our Jenkins server should now be available in your browser with pulic ip on port 8082 (as you configure outside accessing port), as we see here:


Select suggested plugins and go… this our brand new dash board of Jenkins.


Lets create a  jon in Jenkins.

Goto new item → free style → name of project and click  ok.




If didn't add the credentials for github add your credentials.

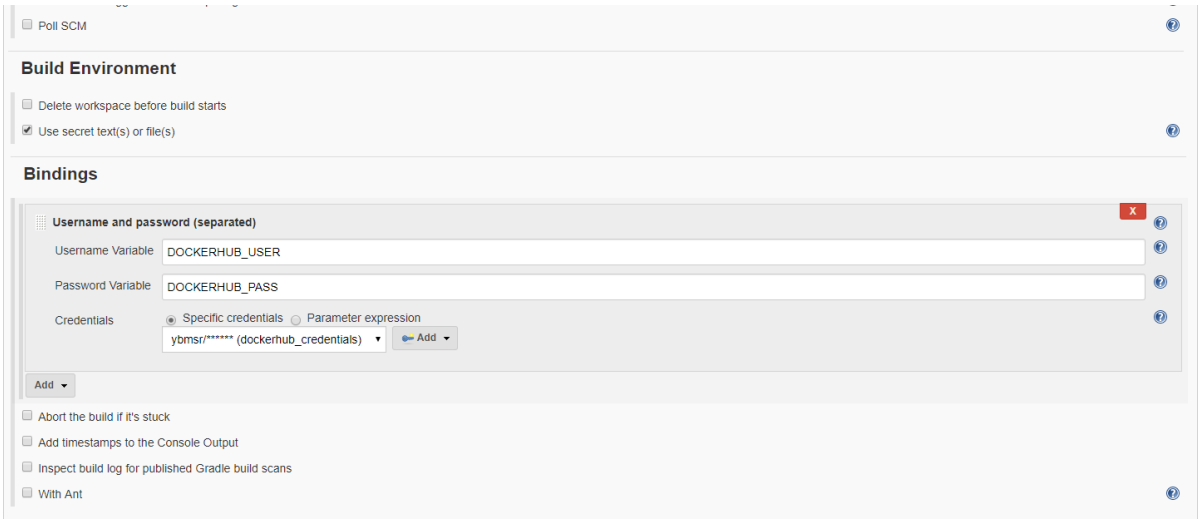
Use execute shell to build and run the container..

```
# Build the image to be used for this job.
IMAGE=$(sudo docker build -t ybmsr/jenkins_javaapp . | tail -1 | awk '{ print $NF }')
# Build the directory to be mounted into Docker.
MNT="$WORKSPACE/.."
echo $MNT
# Execute the build inside Docker.
CONTAINER=$(sudo docker run -d -v $MNT:/opt/project/ $IMAGE)
```

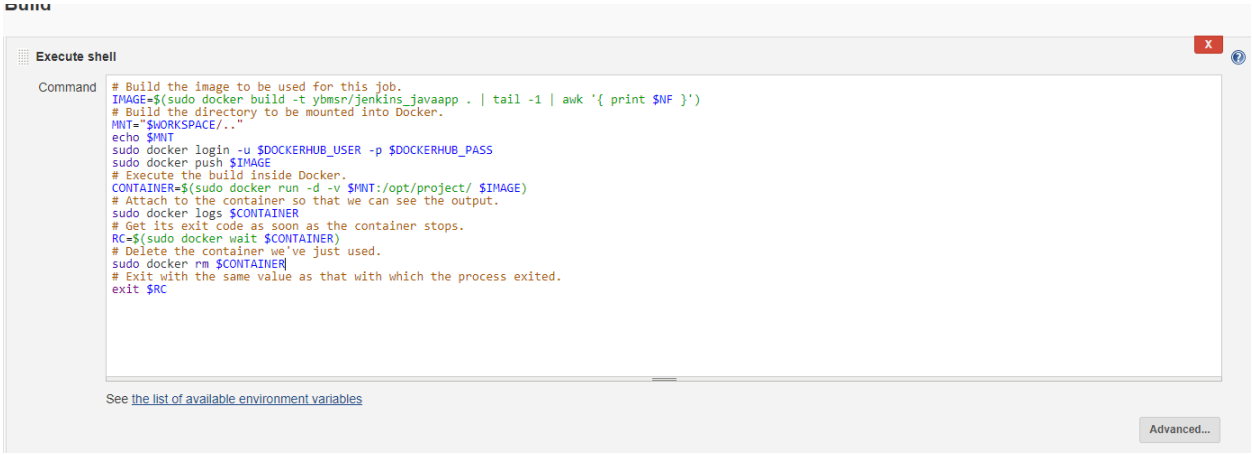
# Attach to the container so that we can see the output.
sudo docker logs $CONTAINER
# Get its exit code as soon as the container stops.
RC=$(sudo docker wait $CONTAINER)
# Delete the container we've just used.
sudo docker rm $CONTAINER
# Exit with the same value as that with which the process exited.
exit $RC


If you want to push images to docker hub use below steps

Need to add credentials in Jenkins Build environment section select **user secret texts or files**
And separate username password variables and add the dockerhub credentials.
Call this variable inside execute shell.
Note: you must have install credentials binding plugin before to use this option. And this plugin is a suggested plugin.




```
# Build the image to be used for this job.
IMAGE=$(sudo docker build -t ybmsr/jenkins_javaapp . | tail -1 | awk '{ print $NF }')
# Build the directory to be mounted into Docker.
MNT="$WORKSPACE/.."
echo $MNT
sudo docker login -u $DOCKERHUB_USER -p $DOCKERHUB_PASS
sudo docker push $IMAGE
# Execute the build inside Docker.
CONTAINER=$(sudo docker run -d -v $MNT:/opt/project/ $IMAGE)
# Attach to the container so that we can see the output.
sudo docker logs $CONTAINER
# Get its exit code as soon as the container stops.
RC=$(sudo docker wait $CONTAINER)
# Delete the container we've just used.
sudo docker rm $CONTAINER
# Exit with the same value as that with which the process exited.
exit $RC
```

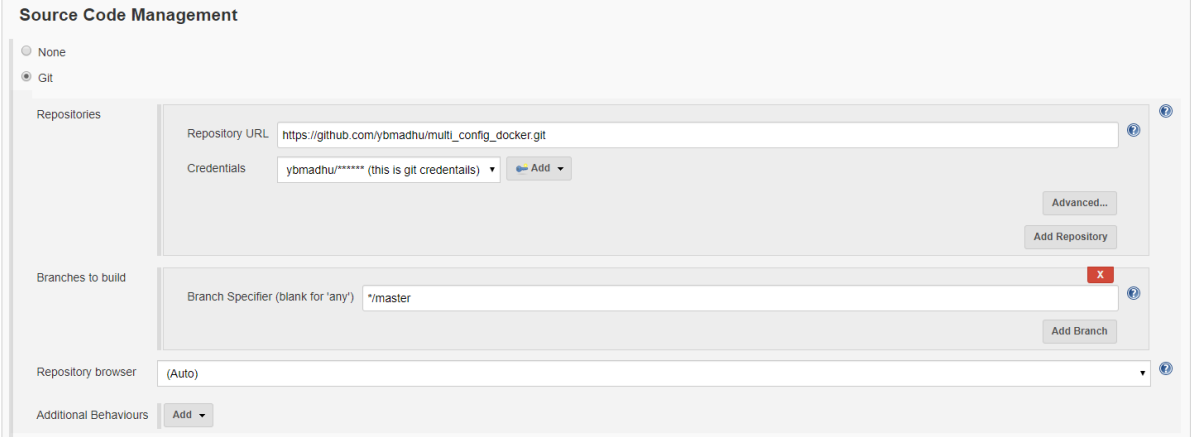


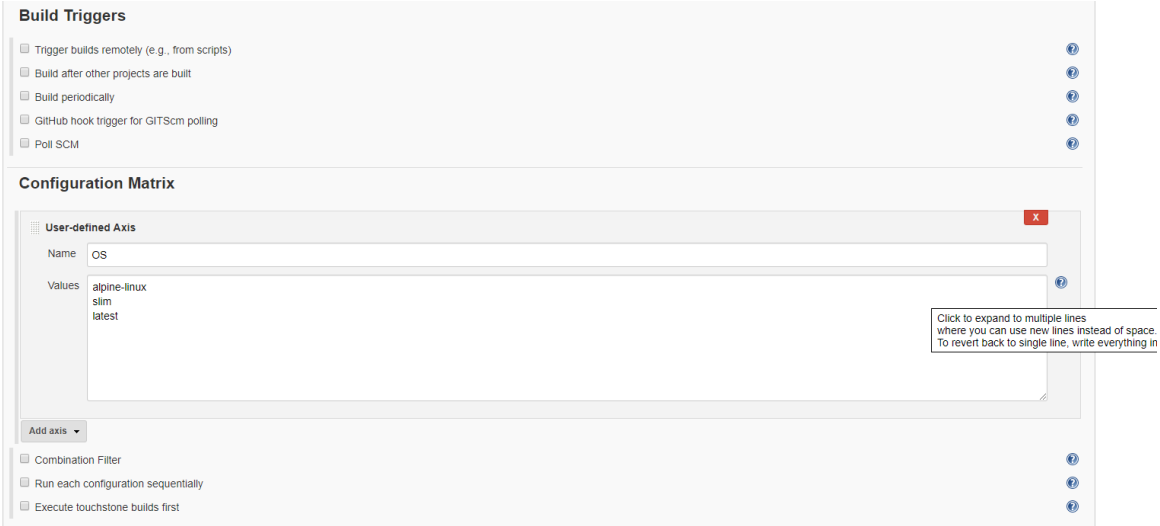
And run build now and check the console output.

```
 > git rev-list --no-walk a282376b8ef6a81b14b078211a7efb9d44e9022e # timeout=10
[docker-sample] $ /bin/sh -xe /tmp/jenkins6748457922522742252.sh
+ awk { print $NF }
+ tail -1
+ sudo docker build -t ****/jenkins_javaapp .
+ IMAGE=****/jenkins_javaapp:latest
+ MNT=/var/jenkins_home/workspace/docker-sample/..
+ echo /var/jenkins_home/workspace/docker-sample/..
/var/jenkins_home/workspace/docker-sample/..
+ sudo docker login -u **** -p ****
Login Succeeded
+ sudo docker push ****/jenkins_javaapp:latest
The push refers to repository [docker.io/****/jenkins_javaapp]
c448c52637be: Preparing
4124688a9b7d: Preparing
869de33b0256: Preparing
61427d9501e8: Preparing
ce6c8756685b: Preparing
30339f20ced0: Preparing
0eb22bfb707d: Preparing
a2ae92ffcd29: Preparing
30339f20ced0: Waiting
0eb22bfb707d: Waiting
a2ae92ffcd29: Waiting
869de33b0256: Layer already exists
4124688a9b7d: Layer already exists
c448c52637be: Layer already exists
ce6c8756685b: Layer already exists
61427d9501e8: Layer already exists
0eb22bfb707d: Layer already exists
30339f20ced0: Layer already exists
a2ae92ffcd29: Layer already exists
latest: digest: sha256:f98709c71cc6db1086ce3b7e8437c414015e4f51b615fe08059cd186c6ebf756 size: 1997
+ sudo docker run -d -v /var/jenkins_home/workspace/docker-sample/..:/opt/project/ ****/jenkins_javaapp:latest
+ CONTAINER=1a54c64301d76edbabadbf57326ec3f09363393269287a7f949731c347c32235
+ sudo docker logs 1a54c64301d76edbabadbf57326ec3f09363393269287a7f949731c347c32235
Hello World .. madhu JMS technologies.
+ sudo docker wait 1a54c64301d76edbabadbf57326ec3f09363393269287a7f949731c347c32235
+ RC=0
+ sudo docker rm 1a54c64301d76edbabadbf57326ec3f09363393269287a7f949731c347c32235
1a54c64301d76edbabadbf57326ec3f09363393269287a7f949731c347c32235
+ exit 0
Finished: SUCCESS
```

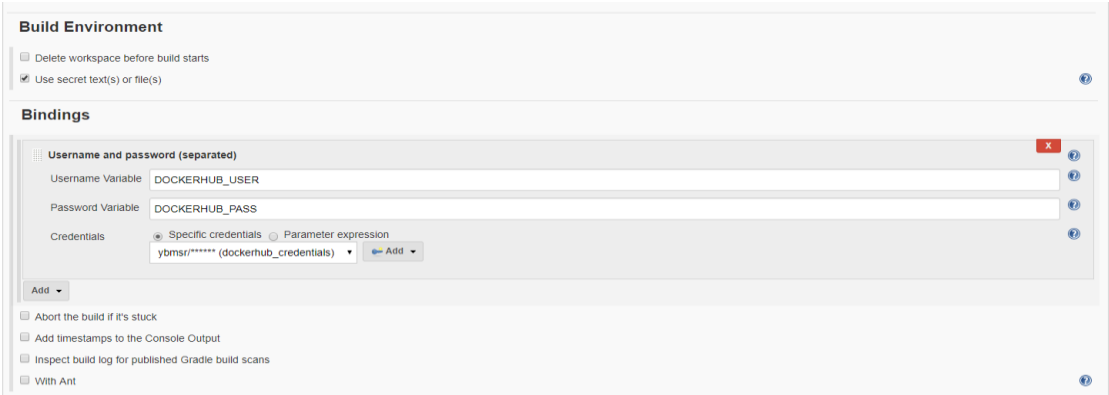
## Multi configuration job in Jenkins

Let’s look at creating our new multi-configuration job. Click on the New Item link from the Jenkins console. We’re going to name our new job multi_config_docker, select Multi-configuration project, and click OK.


Source Code Management
None
Git
Repositories
Repository URL
https://github.com/ybmadhu/multi_config_docker.git
Credentials
ybmadhu/****** (this is git credentails)
Add
Advanced...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/master
Add Branch
Repository browser
(Auto)
Additional Behaviours
Add


Build Triggers
Trigger builds remotely (e.g., from scripts)
Build after other projects are built
Build periodically
GitHub hook trigger for GITScm polling
Poll SCM
Configuration Matrix
User-defined Axis
Name
OS
Values
alpine-linux
slim
latest
Click to expand to multiple lines
where you can use new lines instead of space.
To revert back to single line, write everything in
Add axis
Combination Filter
Run each configuration sequentially
Execute touchstone builds first


Execute touchstone builds first
Build Environment
Delete workspace before build starts
Advanced...
Use secret text(s) or file(s)
Bindings


Build Environment
Delete workspace before build starts
Use secret text(s) or file(s)
Bindings
Username and password (separated)
Username Variable
DOCKERHUB_USER
Password Variable
DOCKERHUB_PASS
Credentials
Specific credentials
Parameter expression
ybmsr/****** (dockerhub_credentials)
Add
Add
Abort the build if it's stuck
Add timestamps to the Console Output
Inspect build log for published Gradle build scans
With Ant


```
cd $OS;
# Build the image to be used for this job.
IMAGE=$(sudo docker build -t ybmsr/${OS}_javaapp . | tail -1 | awk '{ print $NF }')
# Build the directory to be mounted into Docker.
MNT="$WORKSPACE/.."
echo $MNT
sudo docker login -u $DOCKERHUB_USER -p $DOCKERHUB_PASS
sudo docker push $IMAGE
# Execute the build inside Docker.
CONTAINER=$(sudo docker run -d -v $MNT:/opt/project/ $IMAGE)
# Attach to the container so that we can see the output.
sudo docker logs $CONTAINER
# Get its exit code as soon as the container stops.
RC=$(sudo docker wait $CONTAINER)
# Delete the container we've just used.
sudo docker rm $CONTAINER
# Exit with the same value as that with which the process exited.
exit $RC
```

---

erros

if you see below error .you need to add below in visudo file

```
jenkins ALL=(ALL) NOPASSWD: ALL
```

```
sudo: no tty present and no askpass program specified
+ IMAGE=
+ MNT=/var/lib/jenkins/workspace/multiconf/OS/alpine-linux/..
+ echo /var/lib/jenkins/workspace/multiconf/OS/alpine-linux/..
/var/lib/jenkins/workspace/multiconf/OS/alpine-linux/..
+ sudo docker login -u -p ****

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

sudo: no tty present and no askpass program specified
Build step 'Execute shell' marked build as failure
Finished: FAILURE
```

Note: if you are doing freestyle job with docker cloudbees docker build and publish plugin.

You should login docker login inside container.

If not  you will see below error.

```
3fcb4803ca2d: Layer already exists
errors:
denied: requested access to the resource is denied
unauthorized: authentication required

Build step 'Docker Build and Publish' marked build as failure
Finished: FAILURE
```