**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 shubham sangani  USER root  RUN apt-get -y update; apt-get install -y sudo; apt-get install -y git  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**

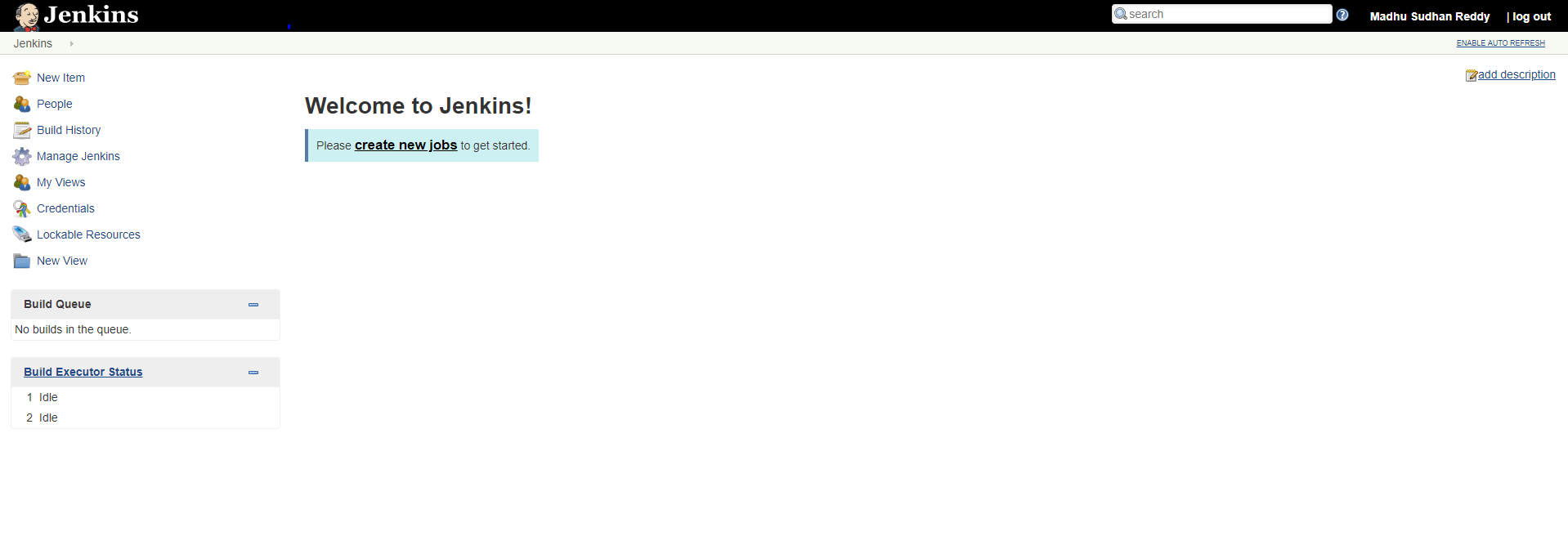
This is also stored in a file in the jenkins\_home directory at:

**/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:

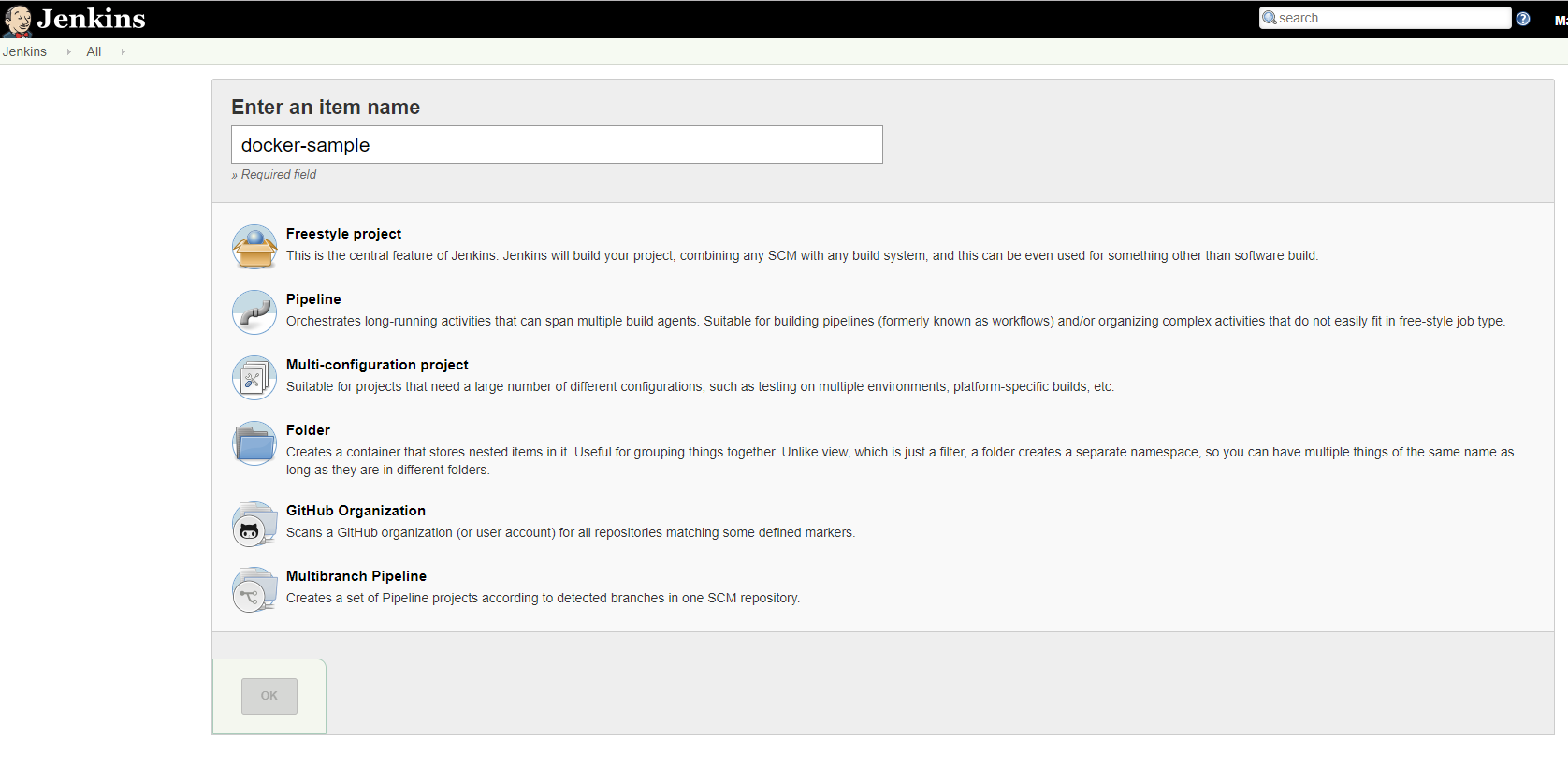


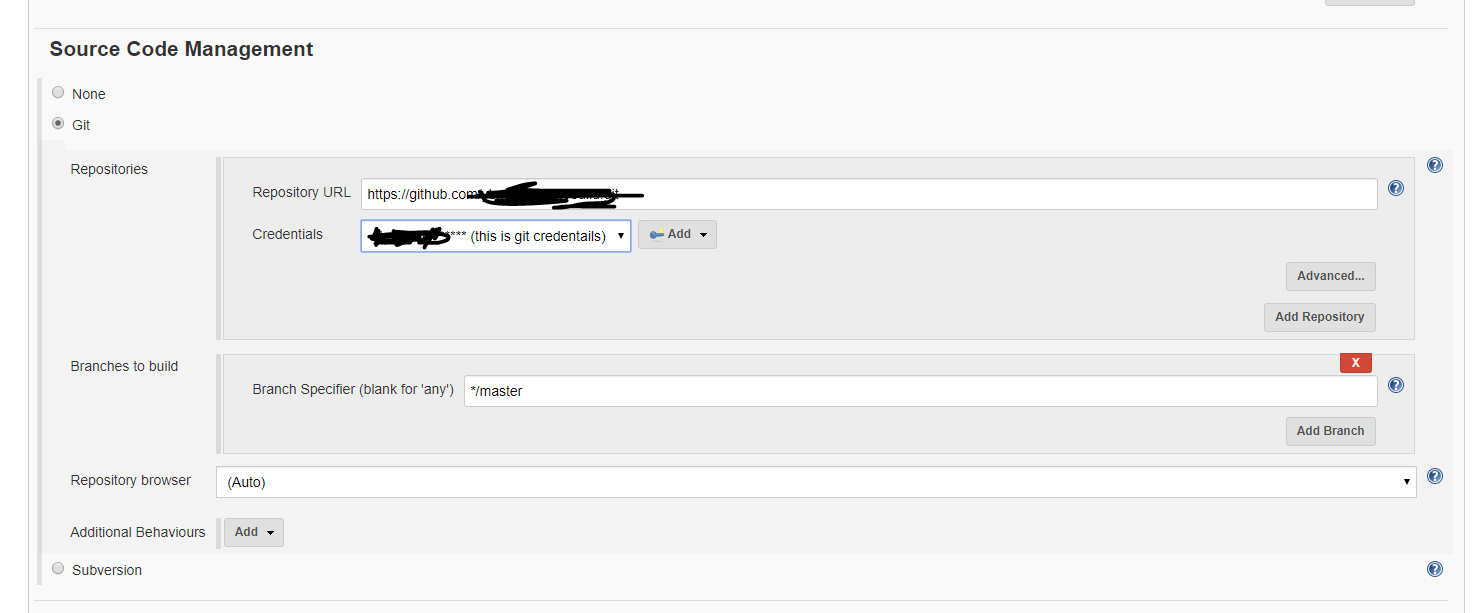
Select suggested plugins and go… this our brand new dash board of Jenkins.



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



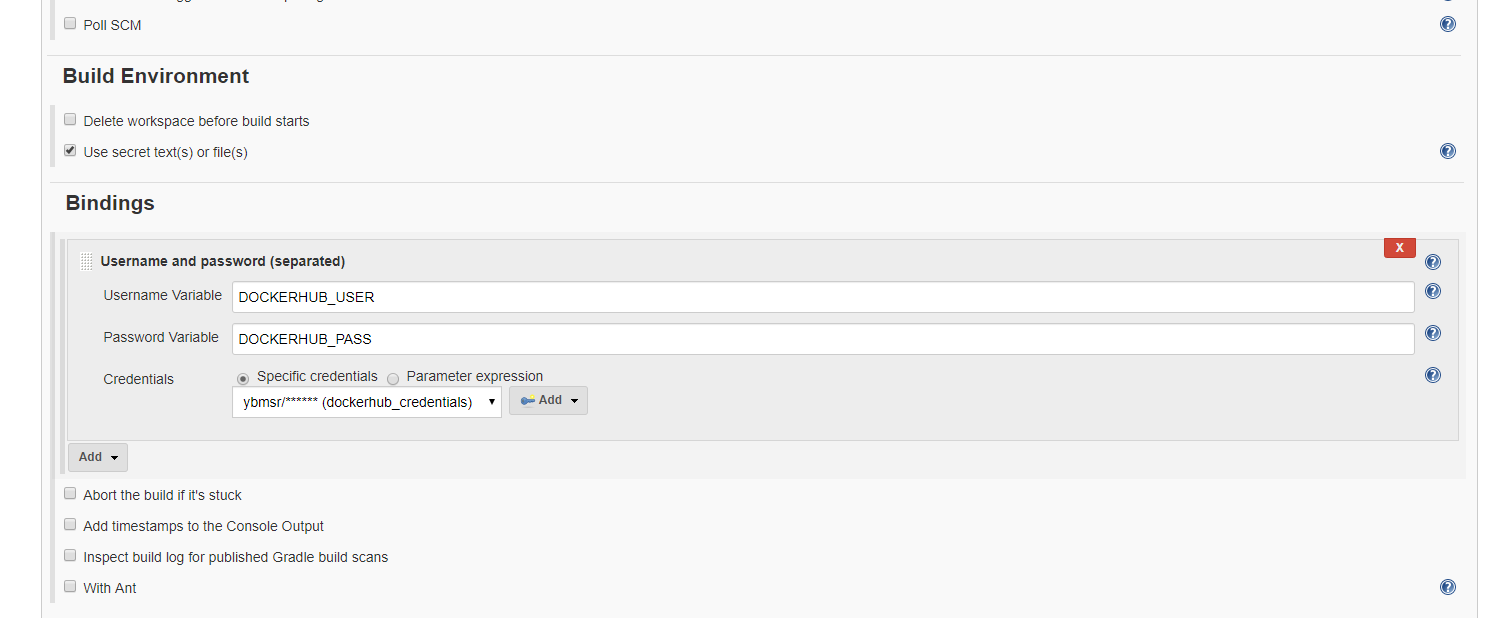
If you want to push images to docker hub use below steps

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.

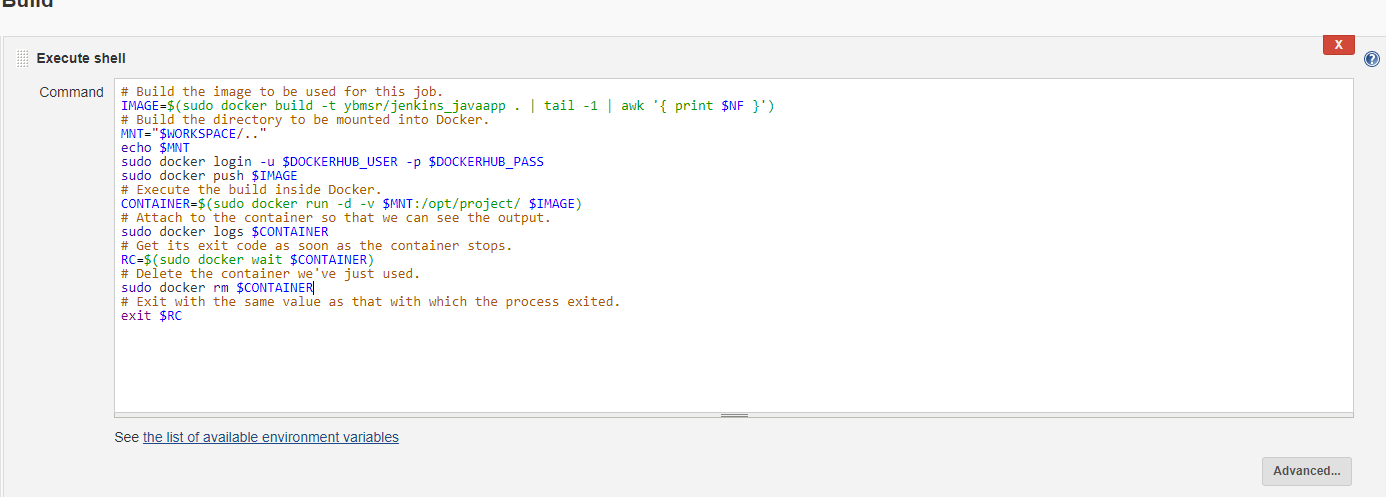
RC=$(sudo docker wait $CONTAINER)

# 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.

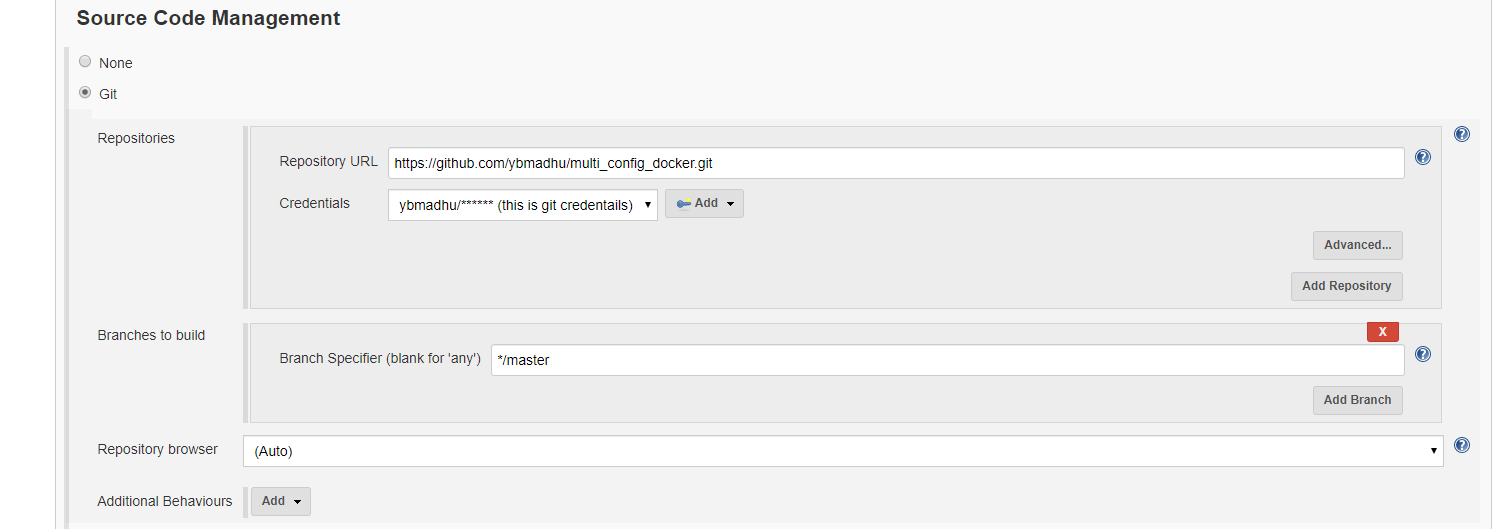


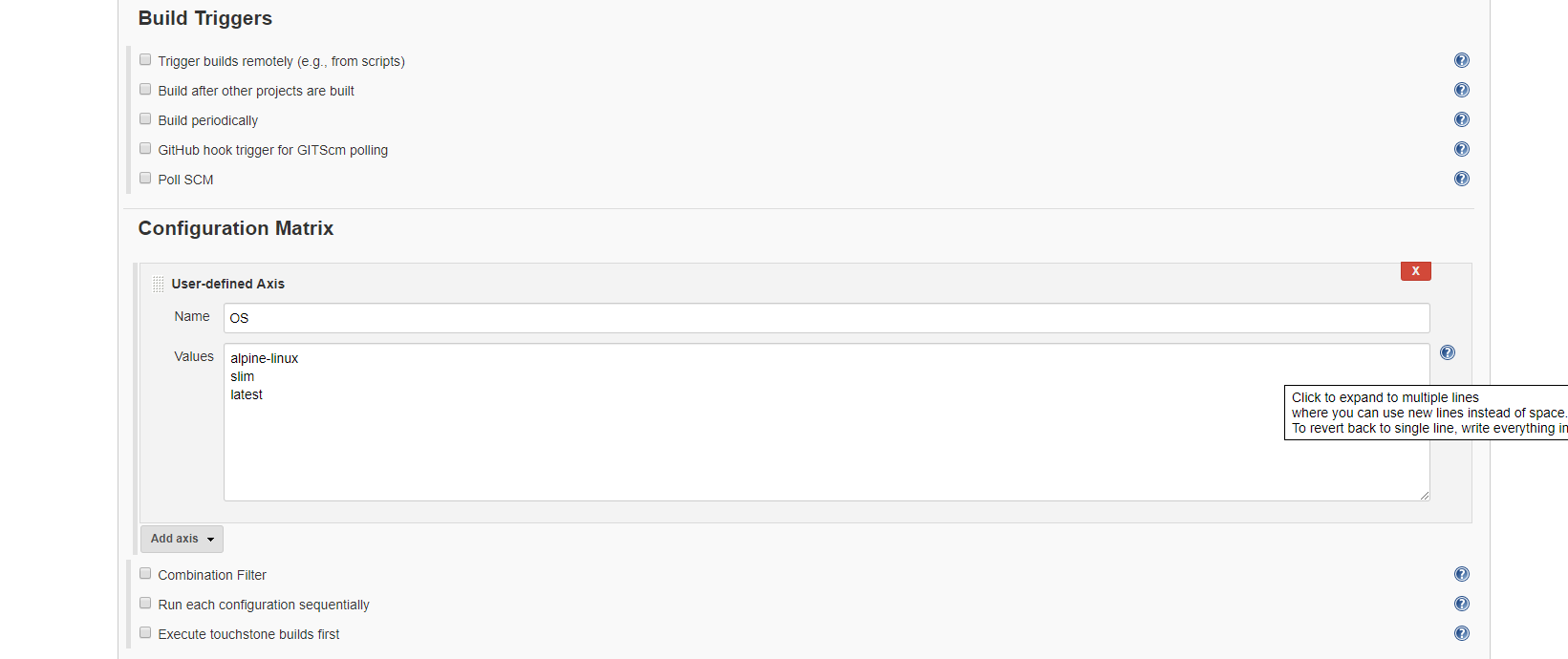
**Multi configuration job in Jenkins**

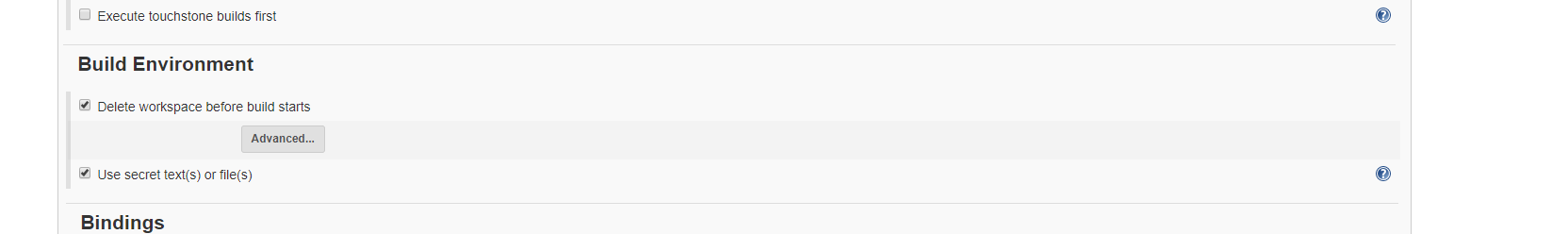
Let’s look at creating our new multi-configuration job. Click on the New Item link

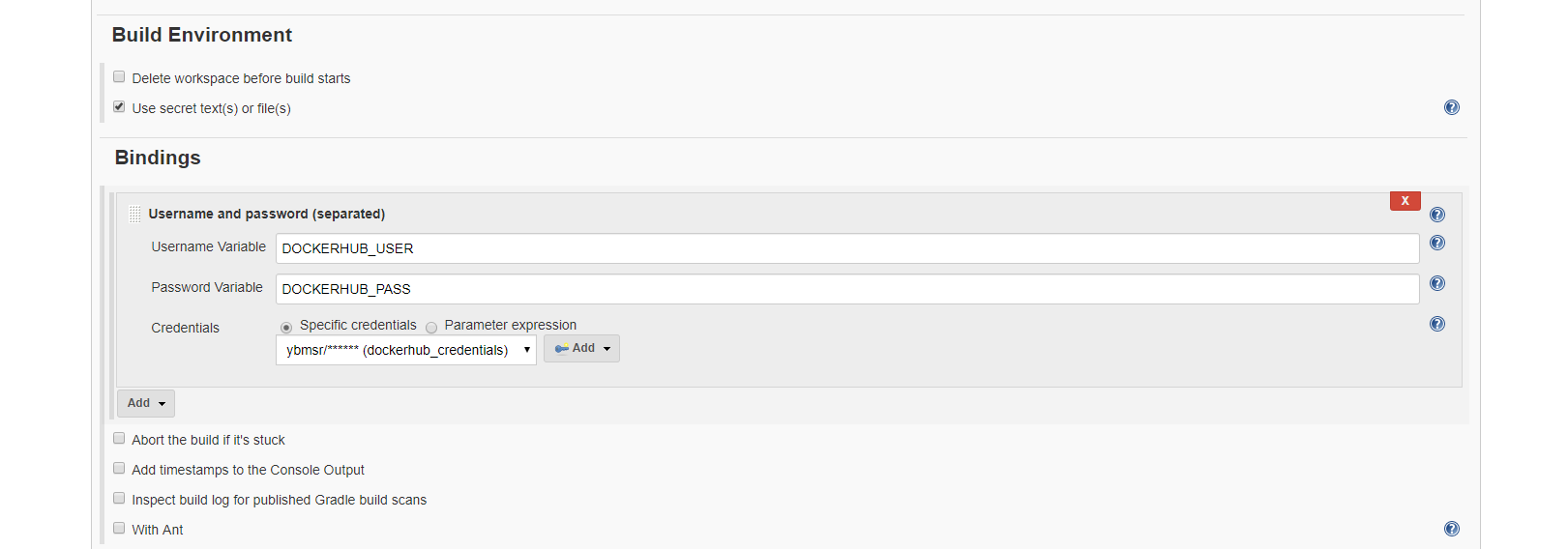
from the Jenkins console. We’re going to name our new job multi\_config\_docker,

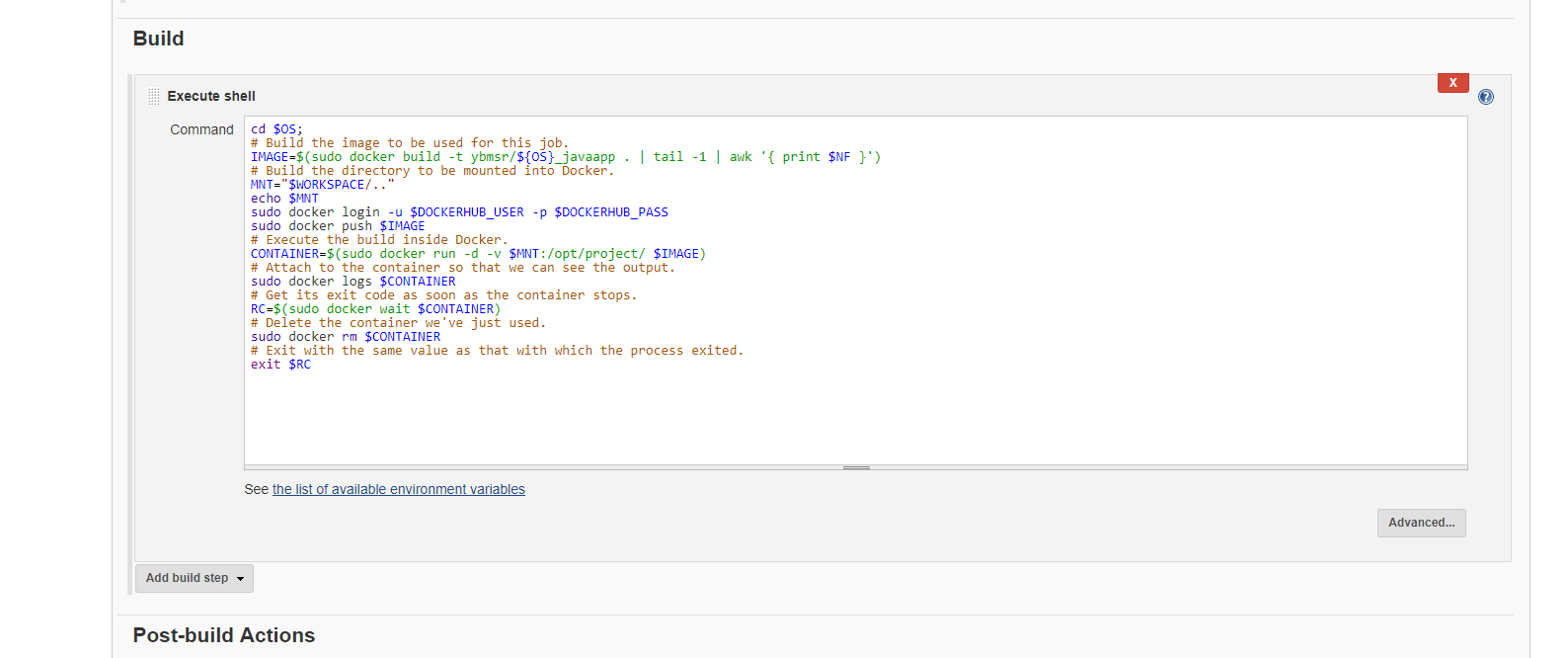
select Multi-configuration project, and click OK.











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.

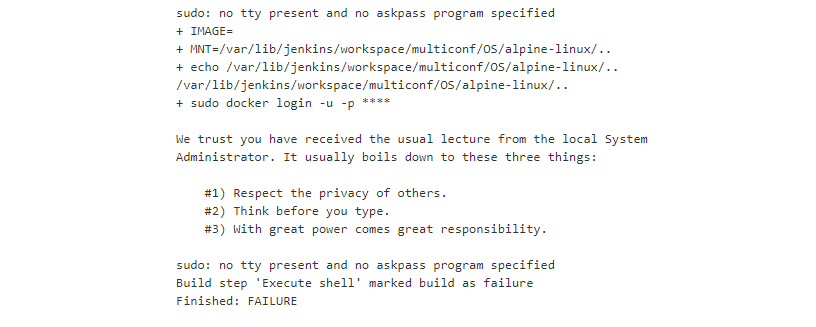
exit $RC

------------

erros

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

jenkins ALL=(ALL) NOPASSWD: ALL



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.

