

# Lab 6: Building & Publishing Docker Images

Author: Gourav Shah

Publisher: School of Devops

Version : v2024.05.01.01

---

In addition to helping the process of building and testing software, docker could also provide a standard packaging format i.e. docker images. Along with this packaging format, comes the distribution mechanism i.e. docker image registries.

In this lab you are going to learn,

- How to Build Docker Images using Dockerfiles
- How to automate Docker Image Build from Jenkins

## Automated Image build with Dockerfile

Since this stage involves adding a new file to the repository, begin by creating a new feature branch to add and test this code before merging to master.

Clone Repository for Java sysfoo app if you haven't already.

```
cd /root
git clone https://github.com/xxxxxxx/sysfoo.git
```

Replace the URL with your own GitHub repo above

Switch to sysfoo path and

```
cd sysfoo
git pull origin
```

If the branch exists, just switch to it

```
git checkout docker
```

create a new branch by name `docker` if it does not exist already.

```
git checkout -b docker
```

Now, let's build a Docker Image for sysfoo app using Dockerfile. To do this, create a file by name **Dockerfile** in the root of the source code.

```
file: sysfoo/Dockerfile
```

```
FROM maven:3.9.6-eclipse-temurin-17-alpine AS build
WORKDIR /app
COPY . .
RUN mvn package -DskipTests

FROM openjdk:17-jdk-alpine as package
WORKDIR /app
COPY --from=build /app/target/sysfoo-*.jar ./sysfoo.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "sysfoo.jar"]
```

Test build the image.

ensure you replace xxxxxx with the actual docker id of yours.

```
docker image build -t xxxxxx/sysfoo:v1 .

docker image ls
```

Try building again using the same command,

```
docker image build -t xxxxxx/sysfoo:v1 .
```

This time, observe it does not build everything, but uses cache.

Test run the image

```
docker container run -idt -p 8954:8080 xxxxxx/sysfoo:v1
```

And then browse to `http://IPADDRESS:8954/` to validate the application. Since its going to launch with tomcat, which takes time to initialise, expect it to take some time ( upto 10 minutes) for the application to show up.

Note: Expect a delay for the app to show up

[Sample Output]

# Welcome to Sysfoo!

## System Information

Hostname: fda781839ca1

IP Address: 172.17.0.4

Running in Docker: true

Running in Kubernetes: false

App Version: 1.0.0

## App Version

1.0.0

## Database Information

Status: Connected

Database Type: H2

## Add To-Do Item

Add To-Do

Once validated, tag the image as latest,

```
docker image tag xxxxxx/sysfoo:v1 xxxxxx/sysfoo:latest  
  
docker image ls
```

Finally, publish images to the registry,

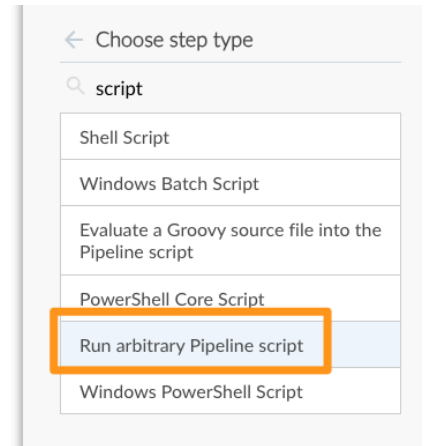
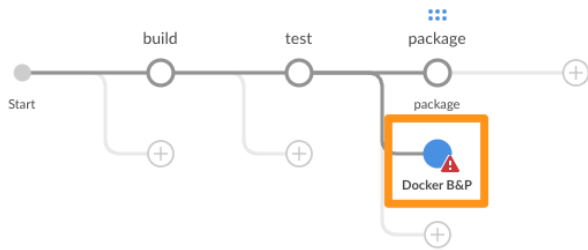
```
docker image push xxxxxx/sysfoo:v1  
docker login  
docker image push xxxxxx/sysfoo:v1  
docker image push xxxxxx/sysfoo:latest
```

Time to commit the Dockerfile to the repository

```
git add Dockerfile  
git commit -am "added multi stage Dockerfile to build sysfoo app"  
  
git push origin docker
```

Adding docker build stage to Jenkinsfile

Add a new stage parallel to package and name it as **Docker B&P**. Click on **Add Step** and search for *script*. Choose a step which reads **Run arbitrary Pipeline script**.



Provide the following code in the script box

```

docker.withRegistry('https://index.docker.io/v1/', 'dockerlogin') {
    def commitHash = env.GIT_COMMIT.take(7)
    def dockerImage = docker.build("xxxxxx/sysfoo:${commitHash}", "./")
    dockerImage.push()
    dockerImage.push("latest")
    dockerImage.push("dev")
}
  
```

Source: [Docker Build and Publish Code Snippet to add using Blue Ocean UI · GitHub](#)

Reference : [Using Docker with Pipeline](#)

Where,

- Replace xxxxxx with your docker registry namespace (username/org)
- `dockerlogin` is the reference to the docker registry credentials that you would add to jenkins
- `${commitHash}` is used to automatically provide unique and incremental versioning for the images built
- `GIT_COMMIT` is one of the environment variables made available by jenkins

**WORKSPACE**

The absolute path of the directory assigned to the build as a workspace

**WORKSPACE\_TMP**

A temporary directory near the workspace that will not be browsable if the workspace is a drive root.

**JENKINS\_HOME**

The absolute path of the directory assigned on the controller file system

**JENKINS\_URL**

Full URL of Jenkins, like `http://server:port/jenkins/` (note: only

**BUILD\_URL**

Full URL of this build, like `http://server:port/jenkins/job/foo/`

**JOB\_URL**

Full URL of this job, like `http://server:port/jenkins/job/foo/` (note: only

**GIT\_COMMIT**

The commit hash being checked out.

**GIT\_PREVIOUS\_COMMIT**

The hash of the commit last built on this branch, if any.

**GIT\_PREVIOUS\_SUCCESSFUL\_COMMIT**

The hash of the commit last successfully built on this branch, if any.

**GIT\_BRANCH**

The remote branch name, if any.

**GIT\_LOCAL\_BRANCH**

The local branch name being checked out, if applicable.

**GIT\_CHECKOUT\_DIR**

The directory that the repository will be checked out to. This contains

**GIT\_URL**

The remote URL. If there are multiple, will be `GIT_URL_1`, `GIT_URL_2`,

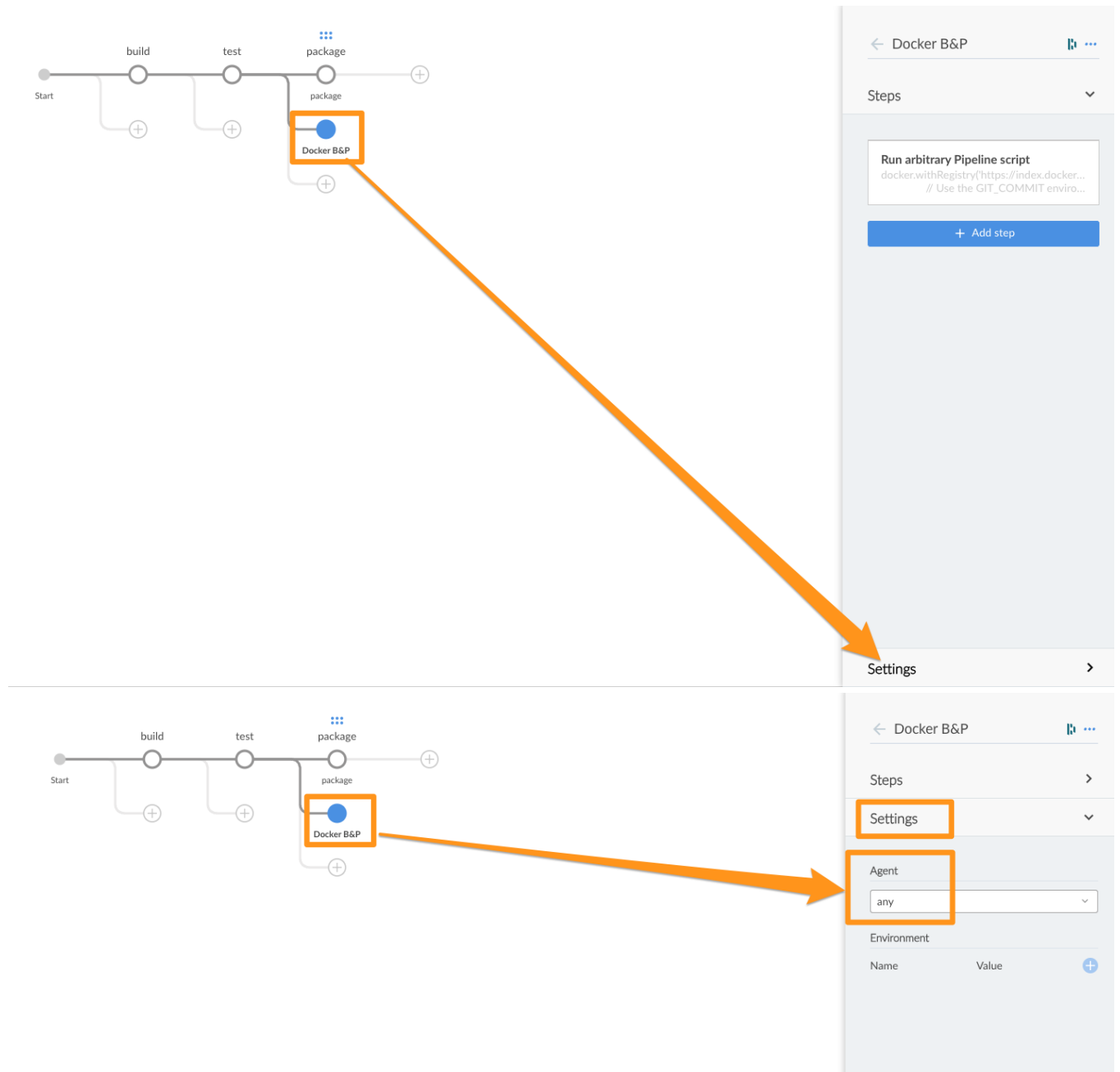
**GIT\_COMMITTER\_NAME**

The configured Git committer name, if any, that will be used for FUTURE

**GIT\_AUTHOR\_NAME**

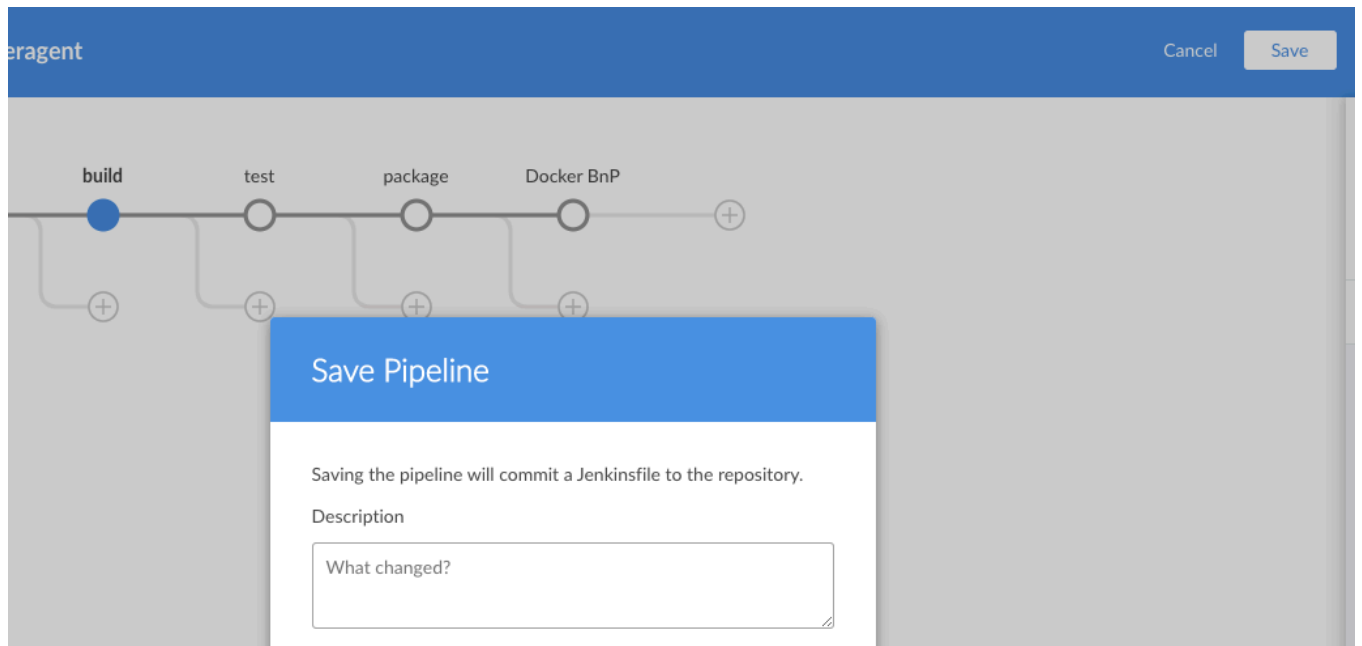
Ensure you replace the values for `xxxxx` and `yyyyy` with the actuals.

Also from Settings (almost hidden in the bottom right corner) set Agent to **any**. This is important as the Docker B&P job needs to run on the Jenkins instance where Docker client is configured.



Now save and commit to the *docker* branch created/checked out earlier.



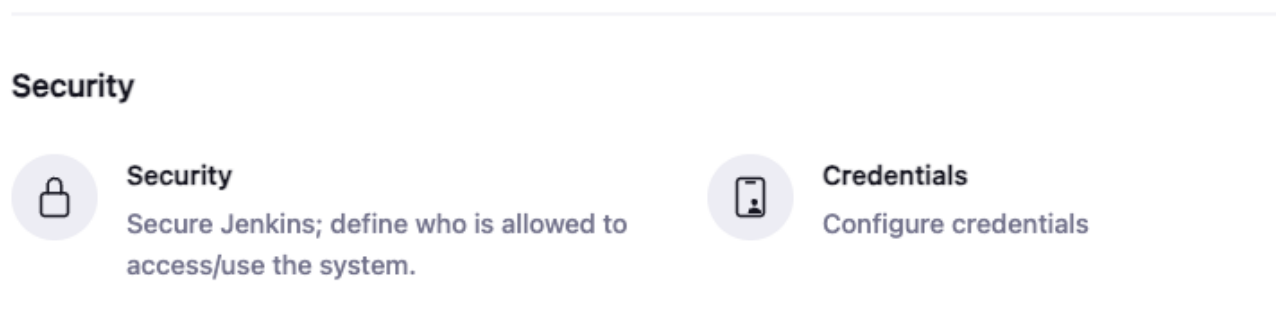


You would see the job failing due to missing credentials *dockerlogin*. For example, this is what I see in my logs

```
ERROR: Could not find credentials matching dockerlogin
```


## Adding Credentials to connect Jenkins with Docker Hub

From Classic Jenkins UI, browse to Manage Jenkins and select **Security** => **Credentials** this time.




Select **System** global credentials.


## Stores scoped to Jenkins


P	Store ↓	Domains
	<b>System</b>	(global)


Click on **Global Credentials**


**Jenkins** ▶ **Credentials** ▶ **System** ▶


 New Item


 People

 Build History

 Project Relationship

 Check File Fingerprint

 **System**

Domain	Description
 <b>Global credentials (unrestricted)</b>	Credentials requirem



Icon: [S](#) [M](#) [L](#)

From the top right corner, select **Add Credentials**

### Global credentials (unrestricted)

[+ Add Credentials](#)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
 <a href="#">github-initcron</a>	initcron/***** (Gourav's GitHub Creds)	Username with password	Gourav's GitHub Creds 

And provide your docker hub username and password to publish the images as.



# New credentials

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

[REDACTED]

☐ Treat username as secret ?

Password ?

.....

ID ?

dockerlogin

Description ?

DockerHub Creds

Create

Ensure the id is set to dockerlogin

Once added, re run the pipeline and validate it runs successfully.

Merge Changes: Time to bring in the changes to the master branch by running a pull request followed by a Code Review process.

## Challenge

Now that you have added all the relevant stages, its time to run some of those conditionally. You are going to update the Jenkinsfile in such a way that,

- For main branch, you run the full pipeline with all stages i.e. Build, Test, Package, Docker BnP.
- For all other branches, only run Build and Test, skip over remaining stages.

You could achieve this by adding conditional stage execution based on the branch.

Your challenge is to modify Jenkinsfile to incorporate conditional execution as specified above.

#cicd/labsv3