CloudBees Docker Build and Publish plugin

Plugin Information
View CloudBees Docker Build and Publish on the plugin site for more information.

This plugin provides the ability to build projects with a Dockerfile, and publish the resultant tagged image (repo) to the docker registry. Docs are here.

If you want to build and push your Docker based project to the docker registry (including private repos), then you are in luck!

This is an early version - build @michaelneale of @cloudbees - ask me if you have questions.

Features:

- Only a Dockerfile needed to build your project
- Publish to docker index/registry - INCLUDING PRIVATE REPOS/IMAGES
- nocache option (for rebuild of all Dockerfile steps)
- publish option
- manage registry credentials for private and public repos
- tag the image built - use any Jenkins env. variables.

Dockerfile as buildfile

A Dockerfile is a convenient way to express build instructions. This plugin will use the Dockerfile in the workspace (possibly previously checked out from git) and will invoke docker build to create the Docker image. The result can be automatically uploaded to the Docker Registry or a private registry.

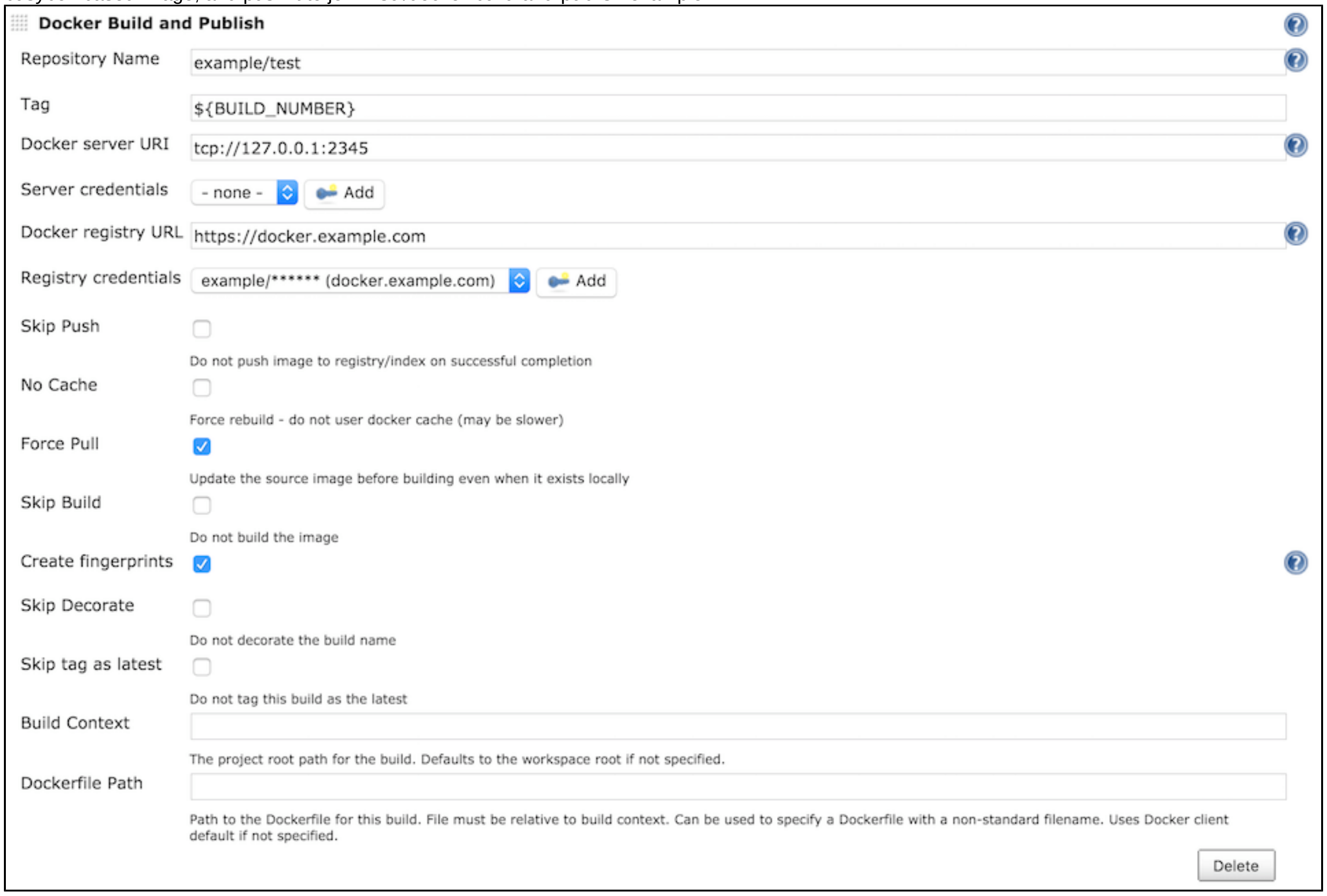
As the Beatles song, all you need is Dockerfile, and love. If you have a Dockerfile in the root of your project, then no further configuration is needed.

How to use

⚠ ***Warning**: the instructions below seem to cover the 0.x version of the plugin. Current documentation is here.*

Firstly, ensure you have docker running (if you are running with a slave, ensure the slave can run docker) - and that Jenkins can run docker commands.

Setup a build of any type - with a CloudBees Docker Build and Publish build step. You can use the example under src/test/example to build a very simple busybox based image, and push it to jenkinsci/docker-build-and-publish-example.

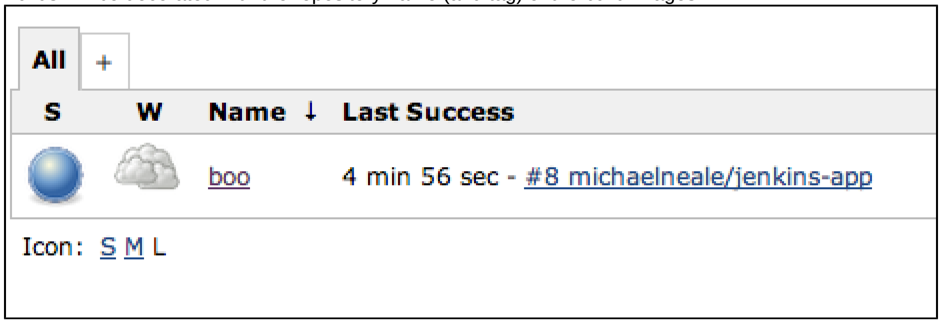


Docker Build and Publish

Repository Name	example/test
Tag	${BUILD_NUMBER}
Docker server URI	tcp://127.0.0.1:2345
Server credentials	- none -
Docker registry URL	https://docker.example.com
Registry credentials	example/****** (docker.example.com)
Skip Push	☐ Do not push image to registry/index on successful completion
No Cache	☐ Force rebuild - do not user docker cache (may be slower)
Force Pull	✓ Update the source image before building even when it exists locally
Skip Build	☐ Do not build the image
Create fingerprints	✓
Skip Decorate	☐ Do not decorate the build name
Skip tag as latest	☐ Do not tag this build as the latest
Build Context	The project root path for the build. Defaults to the workspace root if not specified.
Dockerfile Path	Path to the Dockerfile for this build. File must be relative to build context. Can be used to specify a Dockerfile with a non-standard filename. Uses Docker client default if not specified.

The usual Docker build caching mechanism applies - and you can choose to publish, or not, the resultant image, configured under Advanced options.

Builds will be decorated with the repository name (and tag) of the build images:



S	W	Name ↓	Last Success
●		boo	4 min 56 sec - #8 michaelneale/jenkins-app

You can supply multiple tags for an image separated by commas. The latest tag is automatically applied to image - if you do not want this check the Do not tag this build as latest checkbox.

Why use a Dockerfile

Defining your build as a Dockerfile means that it will run in a consistent linux environment, no matter where the build is run. You also end up with an image (in a repository, possibly pushed to a registry) that can then be deployed - the exact same image you built.

Dockerfiles can also help speed up builds. If there has been no change relative to a build instruction in the Dockerfile - then a cached version of that portion of the build can be used (this is a fundamental feature of docker)

More info

See more on the GitHub readme: