

DockerBuild

Overview

The **DockerBuild** native step performs a build to produce a Docker image from a Dockerfile in a [GitRepo](#) source repository resource.

In the step configuration, you must provide the name (`dockerFileName`) and directory (`dockerFileLocation`) of the [Dockerfile](#) that contains the command to be processed by a `docker build` command, as well as the name (`dockerImageName`) and tag (`dockerImageTag`) of the resulting image. The image is built on the build node, and information about that image is stored in the run state.

To build a Docker image that relies on a private base image:

1. Define the base image as an [Image](#) resource, with `autoPull` set to `true`.
2. Specify the Image resource as one of the `inputResources` of the DockerBuild step.

To include artifacts in the Docker image that are not part of the GitRepo source repository:

1. Define a [FileSpec](#) resource that specifies the files to include from Artifactory.
2. Specify the FileSpec resource as one of the `inputResources` of the DockerBuild step.



Proper usage of DockerBuild step

DockerBuild and [DockerPush](#) steps must be assigned to the same `affinityGroup` to share state. If this is not done, the output of DockerBuild will not be available for DockerPush. For more information on using `affinityGroup`, see [Running multiple steps on the same build node](#).



Docker Build and Push Quickstart

This [Docker Build and Push](#) quickstart demonstrates the definition of a pipeline that uses the DockerBuild and [DockerPush](#) native steps to build a single Docker Image, push it to Artifactory, and then publish the BuildInfo.

Page Contents

- [Overview](#)
- [YAML Schema](#)
- [Tags](#)
 - [name](#)
 - [type](#)
 - [configuration](#)
- [Examples](#)
 - [Resources](#)
 - [Build a Docker image from a source repository](#)
 - [Build a Docker image with dockerOptions](#)
 - [Build a Docker image with a private base image](#)
 - [Build a Docker image with files outside the current path](#)
- [How it Works](#)

YAML Schema

The YAML schema for DockerBuild native step is as follows:

DockerBuild

```
pipelines:
- name: <string>
  steps:
  - name: <string>
    type: DockerBuild
    configuration:
      #inherits all the tags from bash; https://www.jfrog.com/confluence/display/JFROG/Bash
      affinityGroup: <string>
      dockerFileLocation: <string>
      dockerFileName: <string>
      dockerImageName: <string>
      dockerImageTag: <string>
      dockerOptions: <string>

      integrations:
      - name: <artifactory or docker registry integration> # required

      inputResources:
      - name: <GitRepo resource> # required, git repository containing your
Dockerfile
      - name: <Image resource> # optional base image
      - name: <FileSpec resource> # optional

      execution:
      onStart:
      - echo "Preparing for work..."
      onSuccess:
      - echo "Job well done!"
      onFailure:
      - echo "uh oh, something went wrong"
      onComplete: #always
      - echo "Cleaning up some stuff"
```

Tags

name

An alphanumeric string (underscores are permitted) that identifies the step.

type

Must be `DockerBuild` for this step type.

configuration

Specifies all configuration selections for the step's execution environment. This step inherits the [Bash/PowerShell](#) step configuration tags, including these pertinent tags:

Tag	Description of usage	Required /Optional
<code>affinityGroup</code>	Must specify an affinity group string that is the same as specified in a subsequent DockerPush step.	Optional

inputResources	<p>Must specify:</p> <ul style="list-style-type: none"> a GitRepo resource (that contains the Dockerfile) <p>Optionally, you may also specify:</p> <ul style="list-style-type: none"> One or more Image resources to pull base images used in the build or to trigger this build. One or more FileSpec resources that specify what files to include in the build context. These files are automatically copied to <code>dockerFileLocation</code>. 	Required /Optional
----------------	---	--------------------

In addition, these tags can be defined to support the step's native operation:



Tags derived from Bash

All native steps derive from the [Bash](#) step. This means that all steps share the same base set of tags from Bash, while native steps have their own additional tags as well that support the step's particular function. So it's important to be familiar with the [Bash](#) step definition, since it's the core of the definition of all other steps.

Tag	Description of usage	Required /Optional
<code>dockerFileLocation</code>	Directory containing the Dockerfile, which is the file that has Docker build configuration. This file is also used as the context for the Docker build. The path provided should be relative to the root of the input GitRepo repository. If no location is provided, the default is the root of the GitRepo repository.	Required
<code>dockerFileName</code>	Name of the Dockerfile.	Required
<code>dockerImageName</code>	The name of the Docker image to create. This can be set using environment variables or triggering a run using parameters.	Required
<code>dockerImageTag</code>	The tag for the Docker image to create. This can be set using environment variables or triggering a run using parameters.	Required
<code>dockerOptions</code>	Additional options for the docker build command.	Optional

Examples

The following examples use a GoLang Git repository represented by a `GitRepo` resource named `gosvc_app` to create a Docker image that is published to Artifactory. They assume that an Artifactory integration named `MyArtifactory` has been created, and that the Artifactory instance has a Docker repository mapped to `docker.artprod.company`.

- These examples require an [Artifactory Integration](#) and a [GitHub Integration](#).
- The Pipelines DSL for a similar example is available in [this repository](#) in the [JFrog](#) GitHub account.
- For a full tutorial, see [Pipeline Example: Docker Build and Push](#).

The following resources declarations support these examples. Not all of these resources are used in all examples.

Resources

```

resources:
# Application source repository
- name: gosvc_app
  type: GitRepo
  configuration:
    gitProvider: myGithub
    path: myuser/myrepo          # replace with your repository name
    branches:
      include: master

# Docker image in an Artifactory repository
- name: base_image
  type: Image
  configuration:
    registry: myArtifactory
    sourceRepository: docker-local    # replace with your repository name
    imageName: docker.artprod.mycompany.com/baseimage
    imageTag: latest
    autoPull: true

# Files in an Artifactory repository
- name: icon_files
  type: FileSpec
  configuration:
    sourceArtifactory: myArtifactory
    pattern: my-local-repo/all-my-images/
    target: icons/

```

Build a Docker image from a source repository

This example builds a Docker image to a Docker registry in Artifactory. The tag for the image is set to the pipeline's run number.

```

pipelines:
- name: demo_pipeline
  steps:
    - name: bld_image
      type: DockerBuild
      configuration:
        dockerFileLocation: .
        dockerFileName: Dockerfile
        dockerImageName: docker.artprod.mycompany.com/gosvc    # replace with your fully qualified Docker
        registry/image name
        dockerImageTag: ${run_number}
        inputResources:
          - name: gosvc_app
        integrations:
          - name: MyArtifactory

```

Build a Docker image with dockerOptions

This example demonstrates use of the `dockerOptions` tag to set the `build-arg` option for the Docker command. An environment variable named `build_number_env_variable` is dynamically set to the pipeline's run number. The example assumes the environment variable is used in the Dockerfile commands.

```

pipelines:
  - name: demo_pipeline
    steps:
      - name: bld_image
        type: DockerBuild
        configuration:
          dockerFileLocation: .
          dockerFileName: Dockerfile
          dockerImageName: docker.artprod.mycompany.com/gosvc # replace with your fully qualified Docker
registry/image name
          dockerImageTag: ${run_number}
          dockerOptions: --build-arg build_number_env_variable=${run_number}
          inputResources:
            - name: gosvc_app
          integrations:
            - name: MyArtifactory

```

Build a Docker image with a private base image

This example builds a Docker image that relies on a private base image stored in an Artifactory Docker repository.

```

pipelines:
  - name: demo_pipeline
    steps:
      - name: bld_image
        type: DockerBuild
        configuration:
          dockerFileLocation: .
          dockerFileName: Dockerfile
          dockerImageName: docker.artprod.mycompany.com/gosvc # replace with your fully qualified
Docker registry/image name
          dockerImageTag: ${run_number}
          inputResources:
            - name: gosvc_app
            - name: base_image
          integrations:
            - name: MyArtifactory

```

Build a Docker image with files outside the current path

This example demonstrates building a Docker image that includes files outside of the current path. It pulls icon files stored in an Artifactory repository for integration art named `my-local-repo`. It is assumed that the Dockerfile has a command that will include the files in `/icons` into the image.

```

pipelines:
  - name: demo_pipeline
    steps:
      - name: bld_image
        type: DockerBuild
        configuration:
          dockerFileLocation: .
          dockerFileName: Dockerfile
          dockerImageName: docker.artprod.mycompany.com/gosvc # replace with your fully qualified
Docker registry/image name
          dockerImageTag: ${run_number}
          inputResources:
            - name: gosvc_app
            - name: icon_files
          integrations:
            - name: MyArtifactory

```

When you use the **DockerBuild** native step in a pipeline, it performs the following functions in the background:

- cp (if there is a FileSpec input, copy those files to the root of the cloned GitRepo input)
- docker build
- add_run_variables (add several variables that are later used when pushing the Docker image or publishing build info)
- jfrog rt build-collect-env (collect environment information to be later published as part of build info)
- add_run_files (save information collected for build info)