CaseStudyNotes for Docker and Docker Compose:

<https://stackify.com/docker-tutorial/>

Docker is a platform for packaging, deploying, and running applications. Docker applications run in containers that can be used on any system: a developer’s laptop, systems on premises, or in the cloud.

[Containerization](https://searchitoperations.techtarget.com/definition/application-containerization-app-containerization) is a technology that’s been around for a long time, but it’s seen new life with [Docker](https://stackify.com/complete-docker-toolkit/). It packages applications as images that contain everything needed to run them: code, runtime environment, libraries, and configuration. Images run in containers, which are discrete processes that take up only as many resources as any other executable.

Compared to virtual machines, containers use less memory and less CPU.

[Containerization](https://stackify.com/microservices-containers/) has enjoyed widespread adoption because of its

* Consistent test environment for development and QA.
* Cross-platform packages called images.
* Isolation and encapsulation of application dependencies.
* Ability to scale efficiently, easily, and in real time.
* Enhances efficiency via easy reuse of images.

Common Docker Commands:

1. Docker run <image name>
2. Docker ps -a ( show all the containers )
3. Docker start -attach <container name> (reuse and attach tells docker to connect to the container output so we can see the results
4. Docker stop <container name or id>
5. Docker image ls
6. Docker can share system resources with a container: ###### Example ####

Images are created with a [Dockerfile,](https://docs.docker.com/engine/reference/builder/) which lists the components and commands that make up an image.

Create a Docker image

Now let’s build on this example to create an image of our own. We’ll package the Nginx image with our **html** file.

Images are created with a [Dockerfile,](https://docs.docker.com/engine/reference/builder/) which lists the components and commands that make up an image.

In **my-nginx**, create a Dockerfile:

FROM nginx

COPY html /usr/share/nginx/html

This Dockerfile contains two instructions:

1. First, create this image from an existing image, which is named **nginx.**The **FROM**instruction is a requirement for all Dockerfiles and establishes the base image. Subsequent instructions are executed on the base image.
2. The second instruction, **COPY,** tells Docker to copy our file tree into the base image, overriding the contents of **/usr/share/nginx/html** in the base image.
3. Build image: docker build -t mynginx . ; -t for tag name, “.” Tell docker to look for Dockerfile in the current directory.
4. Run the customer image: #docker run –name foo -d -p 8080:80 mynginx and our index.html is displayed. (Wow!)

Graphical user interface, text, application, chat or text message

Description automatically generated

1. Docker can check in local images to docker cloud or dockerhub “docker login”; username/password; docker tag mypyweb ericgoebel/stackify-tutorial:1.0.0; docker push <tag name> ; image uploaded to **hub.docker.com**
2. When you’re done with your build, you’re ready to look into [scanning your image with docker scan](https://docs.docker.com/engine/scan/), and [pushing your image to Docker Hub](https://docs.docker.com/docker-hub/repos/).
3. Build Docker image with Maven plugin.

After Spring Boot 2.3, the process got very easy to create docker containers.

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<image>

<name>in28min/mmv2-${project.artifactId}:${project.version}</name>

</image>

<pullPolicy>IF\_NOT\_PRESENT</pullPolicy>

</configuration>

</plugin>

</plugins>

</build>

DockerId; prefix mmv2-project artifactId; tag

<name>in28min/mmv2-${project.artifactId}:${project.version}</name>

Maven Install with goals: spring-boot:build-image -Dskiptests

Command Line: mvn spring-boot:build-image -DskipTests

# Dockerfile reference

<https://docs.docker.com/engine/reference/builder/>

Estimated reading time: 102 minutes

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

This page describes the commands you can use in a Dockerfile. When you are done reading this page, refer to the [Dockerfile Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/) for a tip-oriented guide.

## Usage

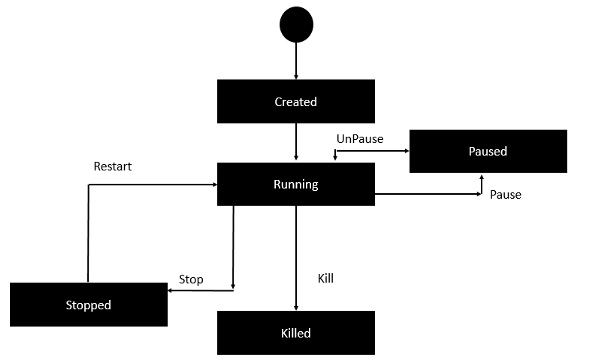
The [docker build](https://docs.docker.com/engine/reference/commandline/build/) command builds an image from a Dockerfile and a context. The build’s context is the set of files at a specified location PATH or URL. The PATH is a directory on your local filesystem. The URL is a Git repository location.

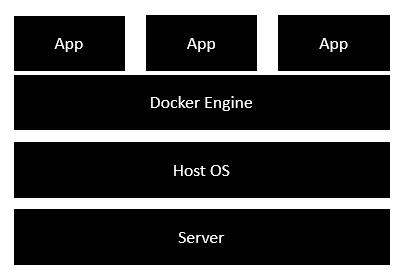
The build context is processed recursively. So, a PATH includes any subdirectories and the URL includes the repository and its submodules. This example shows a build command that uses the current directory (.) as build context:

$ docker build .

**Docker Compose** is used to run multiple containers as a single service.

Docker images lifecycle:





# **Difference between Docker Compose Vs Dockerfile**

A Dockerfile is a simple text file that contains the commands a user could call to assemble an image whereas Docker Compose is a tool for defining and running multi-container Docker applications.

Docker Compose define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment. It gets an app running in one command by just running docker-compose up. Docker compose uses the Dockerfile if you add the build command to your project’s docker-compose.yml. Your Docker workflow should be to build a suitable Dockerfile for each image you wish to create, then use compose to assemble the images using the build command.

Docker Command Cheat Sheet:

<https://collabnix.com/docker-cheatsheet/>

Docker Compose is a tool to launch multiple services with one command.

Use Master Microservices with Spring Boot and Spring Cloud as example:

// TODO: add notes for this.

# Document to follow: Debugging Guide - Microservices - V2 // TODO

https://github.com/in28minutes/spring-microservices-v2/blob/main/03.microservices/01-step-by-step-changes/microservices-v2-1.md#docker-section---connect-microservices-with-zipkin