Getting Started with Docker

Mitch Dresdner

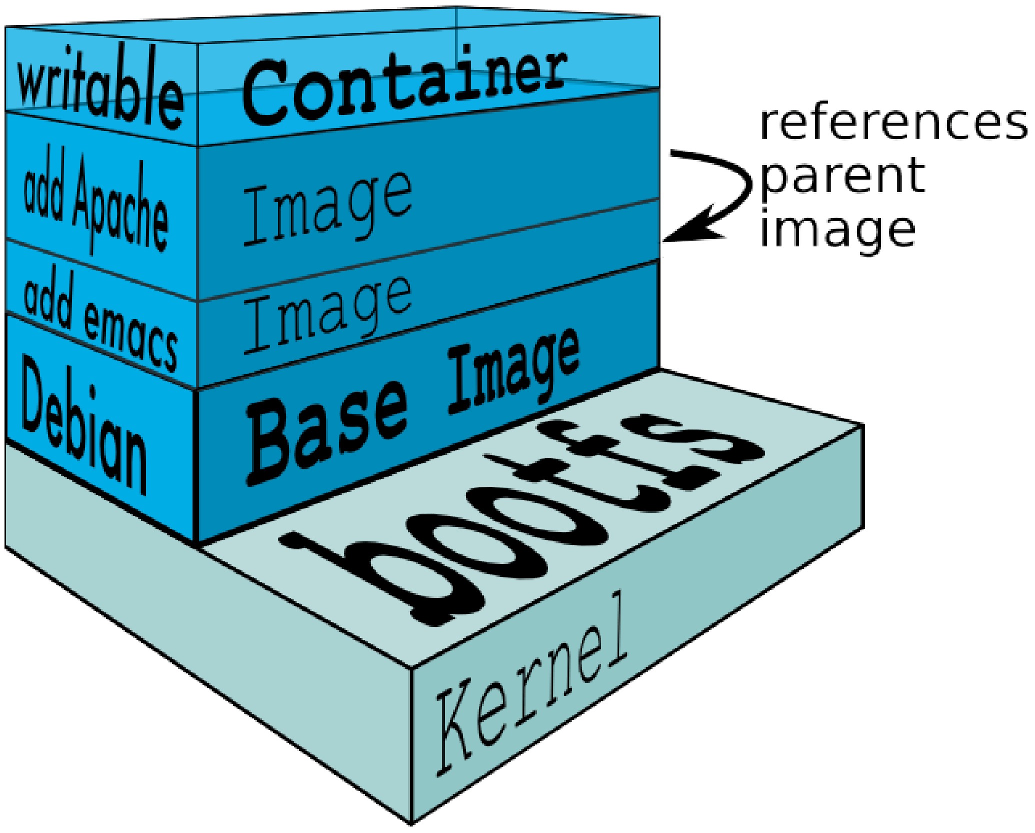
Table of Contents

A simple guide for getting Docker up and running fast on Windows to help *get started managing containers.*

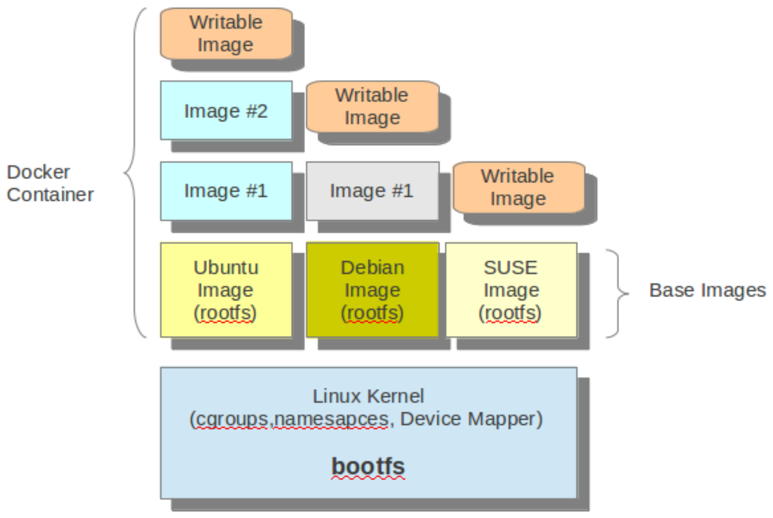
Terminology

Docker Image

Docker images are the basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it never changes.



Images are built up from a base image on top of the operating system kernel. The image layers represent component packages which have been added to the base image. There are many choices for a base image.



Base images are generally pulled from the Docker Hub.

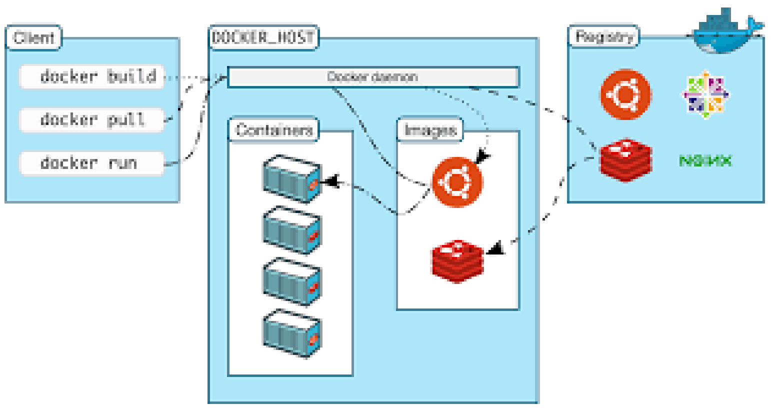
Docker Container

A container is a runtime instance of a Docker image.

A Docker container consists of

- A Docker image
- Execution environment
- A standard set of instructions

The concept is borrowed from Shipping Containers, which define a standard to ship goods globally. Docker defines a standard to ship software.



Containers are the *run time* instantiation of images. Docker runs processes in isolated containers. A container is a process which runs on a host. The host may be local or remote. When an operator executes docker run, the container process that runs is isolated in that it has its own file system, its own networking, and its own isolated process tree separate from the host.

The basic *docker run* command takes this form

```
docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

The docker run command must specify an IMAGE to derive the container from. An image developer can define image defaults related to

- detached or foreground running
- container identification
- network settings
- runtime constraints on CPU and memory

Why Docker?

Docker brings in an API for container management, an image format and a possibility to use a remote registry for sharing containers. This scheme benefits both developers and system administrators with advantages such as

- *Rapid application deployment* – containers include the minimal runtime requirements of the application, reducing their size and allowing them to be deployed quickly
- *Portability across machines* – an application and all its dependencies can be bundled into a single container that is independent from the host version of Linux kernel, platform distribution, or deployment model. This container can be transfered to another machine that runs Docker, and executed there without compatibility issues.
- *Version control and component reuse* – you can track successive versions of a container, inspect differences, or roll-back to previous versions. Containers reuse components from the preceding layers, which makes them noticeably lightweight.
- *Sharing* – you can use a remote repository to share your container with others. Red Hat provides a registry for this purpose, and it is also possible to configure your own private repository.
- *Lightweight footprint and minimal overhead* – Docker images are typically very small, which facilitates rapid delivery and reduces the time to deploy new application containers.

Install Docker

Requirements

A Windows installation should meet the following requirements

- 64bit Windows 10 Pro, Enterprise and Education (1511 November update, Build 10586 or later). In the future we will support more versions of Windows 10
- The Hyper-V package must be enabled. The Docker for Windows installer will enable it for you, if needed. (This requires a reboot).

Having met the installation requirements the Docker for Windows installation package can be obtained here -

NOTE https://download.docker.com/win/stable/InstallDocker.msi

Verify your installation

After the installation successfully completes, run the Docker *Hello World* program to ensure the installation was correct.

```
C:\docker run hello-world
```

Run docker command *ps -a* to show all containers on the system

```
C:\docker ps -a
 CONTAINER ID    IMAGE           COMMAND     CREATED         STATUS      NAMES
 592376ff3eb8    hello-world     "/hello"    25 sec ago      Exited
prickly_wozniak
```

NOTE https://docs.docker.com/engine/getstarted/step_one/#/docker-for-windows

Docker CLI

Commands

attach
: Attach to a running container

build
: Build an image from a Dockerfile

commit
: Create a new image from a container’s changes

cp
: Copy files/folders between a container and the local filesystem

create
: Create a new container

deploy
: Deploy a new stack or update an existing stack

diff
: Inspect changes on a container’s filesystem

events
: Get real time events from the server

exec
: Run a command in a running container

export
: Export a container’s filesystem as a tar archive

history
: Show the history of an image

images
: List images

import
: Import the contents from a tarball to create a filesystem image

info
: Display system-wide information

inspect
: Return low-level information on Docker objects

kill
: Kill one or more running containers

load
: Load an image from a tar archive or STDIN

login
: Log in to a Docker registry

logout
: Log out from a Docker registry

logs
: Fetch the logs of a container

pause
: Pause all processes within one or more containers

port
: List port mappings or a specific mapping for the container

ps
: List containers

pull
: Pull an image or a repository from a registry

push
: Push an image or a repository to a registry

rename
: Rename a container

restart
: Restart one or more containers

rm
: Remove one or more containers

rmi
: Remove one or more images

run
: Run a command in a new container

save
: Save one or more images to a tar archive (streamed to STDOUT by default)

search
: Search the Docker Hub for images

start
: Start one or more stopped containers

stats
: Display a live stream of container(s) resource usage statistics

stop
: Stop one or more running containers

tag
: Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

top
: Display the running processes of a container

unpause
: Unpause all processes within one or more containers

update
: Update configuration of one or more containers

version
: Show the Docker version information

wait

Block until one or more containers stop, then print their exit codes

Run *docker COMMAND --help* for more information on a command.

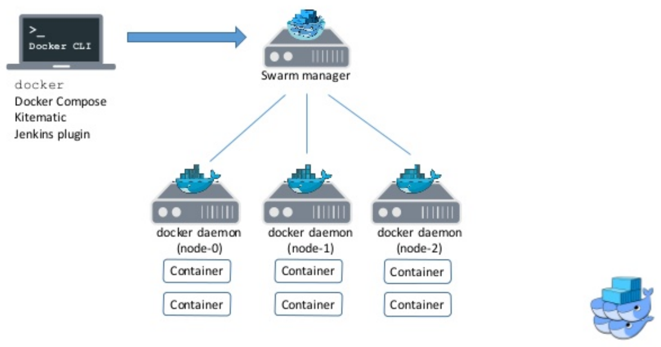
Docker Hub

The Docker Hub is a centralized resource for working with Docker and its components. It provides the following services:

- Docker image hosting
- User authentication
- Automated image builds and work-flow tools such as build triggers and web hooks
- Integration with GitHub and Bitbucket

Docker Compose

Compose is a tool for defining and running complex applications with Docker. With compose, you define a multi-container application in a single file, then spin your application up in a single command which does everything that needs to be done to get it running.



Docker Swarm

Docker Swarm is the name of a standalone native clustering tool for Docker. Docker Swarm pools together several Docker hosts and exposes them as a single virtual Docker host. It serves the standard Docker API, so any tool that already works with Docker can now transparently scale up to multiple hosts.

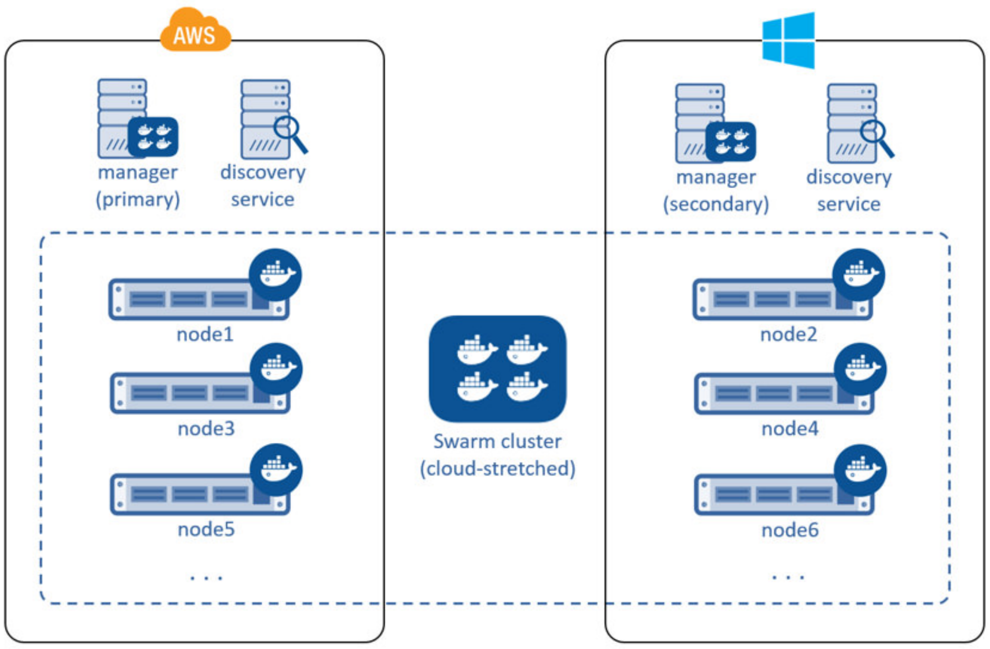


Figure 1. Docker Swarm

Some benefits of Docker Swarm

- *High scalability and performance* - Swarm is production ready and tested to scale up to one thousand (1,000) nodes and fifty thousand (50,000) containers with no performance degradation in spinning up incremental containers onto the node cluster.
- *Failover and high availability* - Ensure a highly available Swarm Manager. Create multiple Swarm masters and specify policies on leader election in case the primary master experiences a failure. Provides experimental support for rescheduling containers when a node fails. Also includes error alerts when a node fails to join a cluster.
- *Integrated networking and volumes* - As a Docker native solution, you can use Docker Networking, Volumes and plugins through their respective Docker commands via Swarm.

CICD and Docker

Continuous integration and continuous deployment has become one of the most common use cases of Docker early adopters. CI/CD merges development with testing, allowing developers to build code collaboratively, submit it the master branch, and checked for issues. This allows developers to not only build their code, but also test their code in any environment type and as often as possible to catch bugs early in the applications development lifecycle.

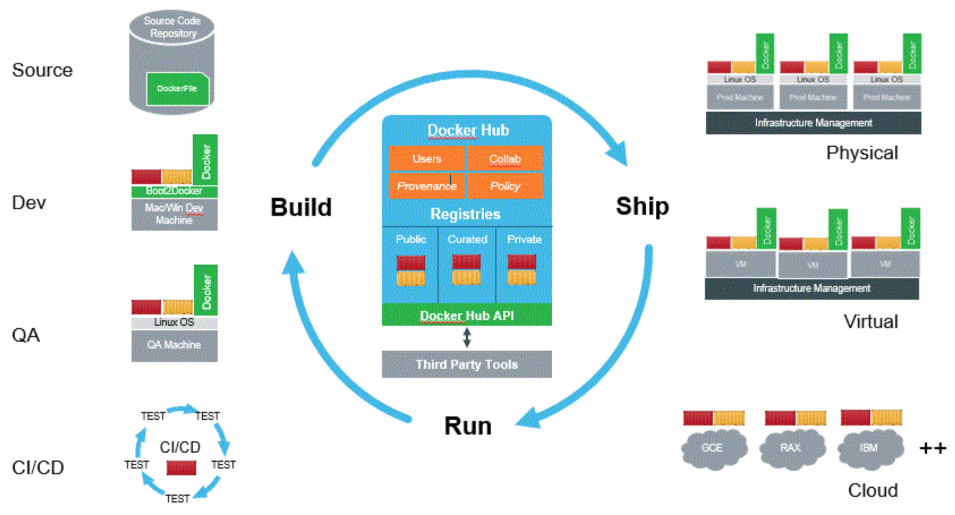


Figure 2. Docker Continuous intergration, continuous deployment

Since Docker can integrate with tools like Jenkins and GitHub, developers can submit code in GitHub, test the code and automatically trigger a build using Jenkins, and once the image is complete, images can be added to Docker registries. This streamlines the process, saves time on build and set up processes, all while allowing developers to run tests in parallel and automate them so that they can continue to work on other projects while tests are being run. Since Docker works on prem, in the cloud or virtual environment and supports both Linux and Windows, enterprises no longer have to deal with inconsistencies between different environments types.