Docker for DevOps

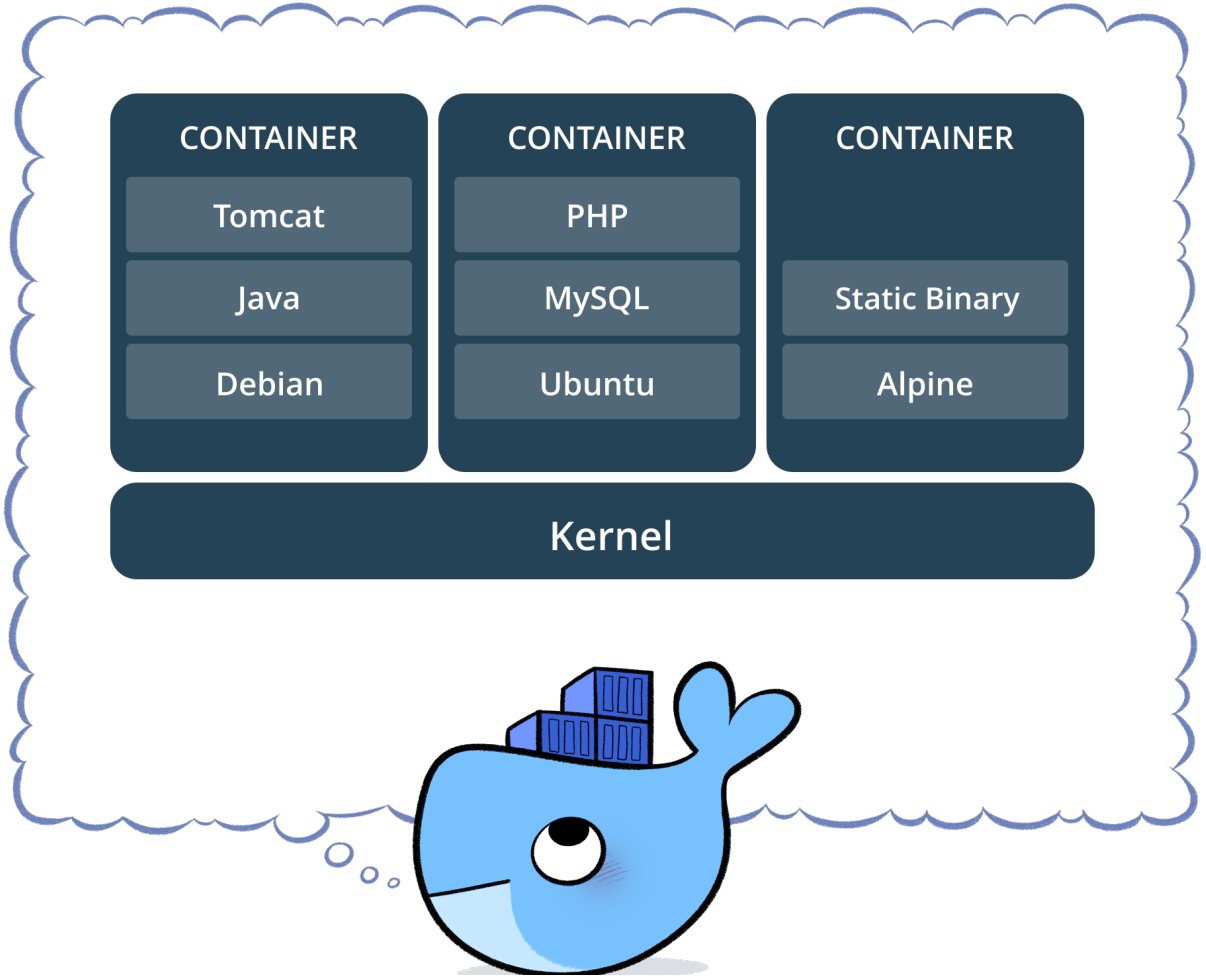
References

- Docker Documentation https://docs.docker.com/
- Mastering Docker Second Edition - Russ McKendrick, Scott Gallagher – Packt Publishing – July 2017

Containers & VMs

Containers

“a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.“



Why Containers? Lightweight

- Containers running on a single machine share that machine's operating system kernel; they start instantly and use less compute and RAM.
- Images are constructed from filesystem layers and share common files. This minimizes disk usage and image downloads are much faster.

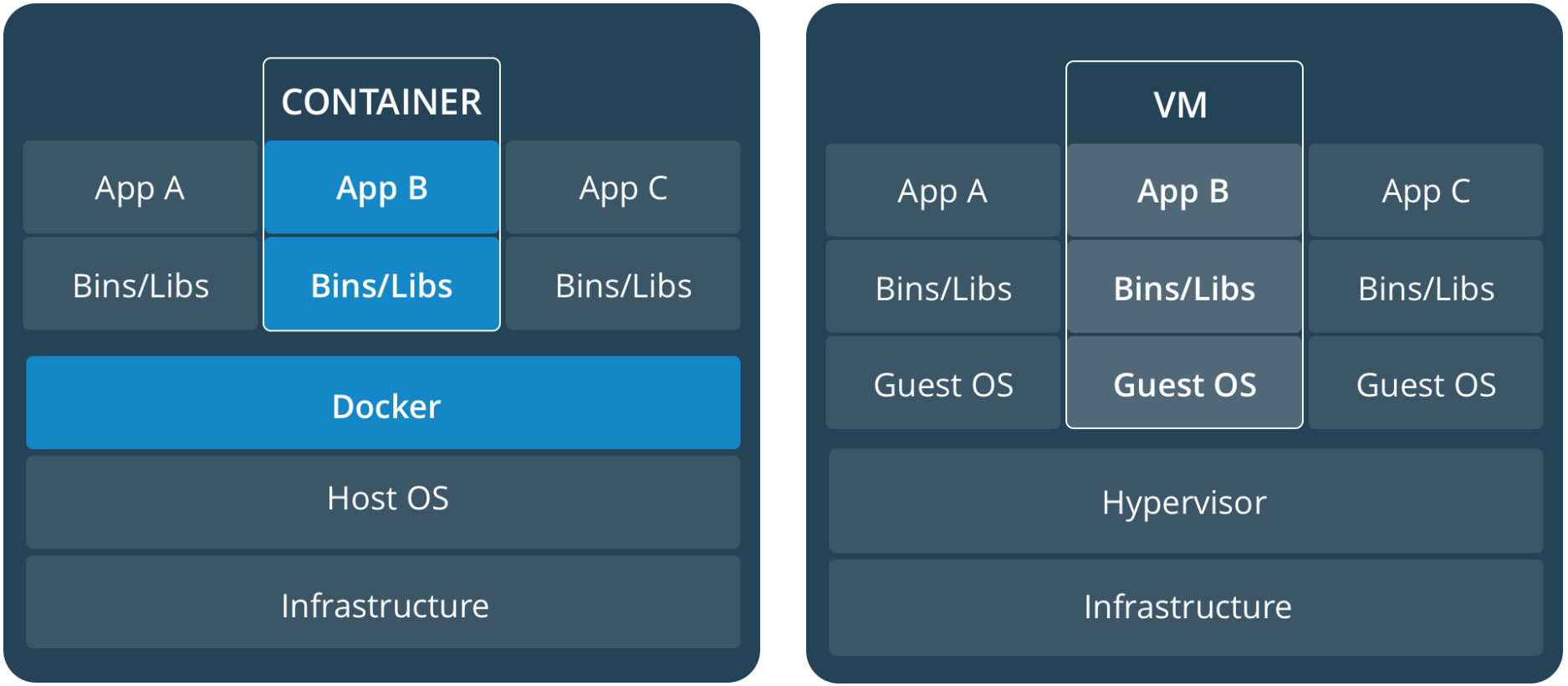
Why Containers? Standard

- Containers are based on open standards and run on all major Linux distributions, Microsoft Windows, and on any infrastructure including VMs, bare-metal and in the cloud.

Why Containers? Secure

- Docker containers isolate applications from one another and from the underlying infrastructure. Docker provides the strongest default isolation to limit app issues to a single container instead of the entire machine.

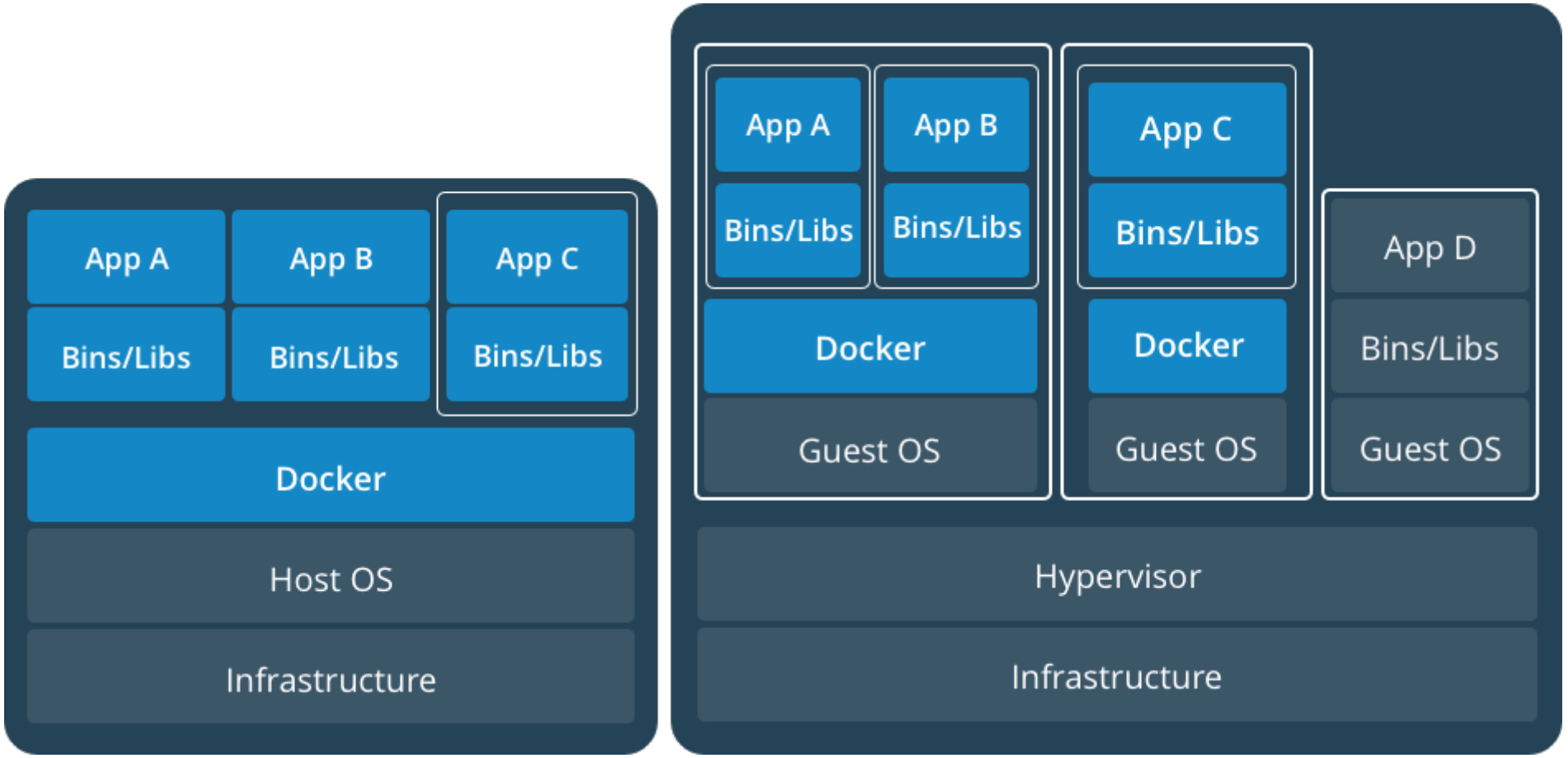
Comparing Containers & VMs (1)



Comparing Containers & VMs (2)

- Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.
- Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

Containers & VMs Together



Install Docker

Docker Release Schedule

Starting with Docker 17.03, Docker uses a time-based release schedule.

- Docker CE Stable releases generally happen quarterly, with patch releases as needed.
- Docker EE releases generally happen twice per year, with patch releases as needed.

Updates, and patches

- A given Docker EE release receives patches and updates for at least one year after it is released.
- A given Docker CE Stable release receives patches and updates for one month after the next Docker CE Stable release.

Docker OS

DOCKER CE

Platform	x86_64 / amd64
CentOS	✔
Debian	✔
Fedora	✔
Ubuntu	✔

DOCKER EE

Platform	x86_64 / amd64
CentOS	✔
Oracle Linux	✔
Red Hat Enterprise Linux	✔
SUSE Linux Enterprise Server	✔
Ubuntu	✔
Microsoft Windows Server 2016	✔

Install Docker

- CentOS

  yum -y install docker
- Ubuntu

  apt -y install docker.io

First Docker Commands

```
## List Docker CLI commands
docker
docker container --help

## Display Docker version and info
docker --version
docker version
docker info

## Execute Docker image
docker run hello-world

## List Docker images
docker image ls

## List Docker containers (running, all, all in quiet mode)
docker container ls
docker container ls --all
docker container ls -aq
```

Images

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.

Dockerfile instructions (1)

- **FROM**, initializes a new build stage and sets the Base Image for subsequent instructions.
- **RUN**, execute any commands in a new layer on top of the current image and commit the results.
- **CMD**, provide defaults for an executing container.
- **LABEL**, adds metadata to an image.
- **EXPOSE**, informs Docker that the container listens on the specified network ports at runtime.
- **ENV**, sets the environment variable <key> to the value <value>.

Dockerfile instructions (2)

- **ADD**, copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.
- **COPY**, copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.
- **ENTRYPOINT**, configure a container that will run as an executable.
- **VOLUME**, creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.

Dockerfile instructions (3)

- **USER**, sets the user name (or UID) and optionally the user group (or GID) to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.
- **WORKDIR**, sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.
- **ARG**, defines a variable that users can pass at build-time to the builder with the docker build command using the --build-arg <varname>=<value> flag.
- **ONBUILD**, adds to the image a trigger instruction to be executed at a later time, when the image is used as the base for another build.

Dockerfile instructions (4)

- **STOPSIGNAL**. sets the system call signal that will be sent to the container to exit.
- **HEALTHCHECK**, tells Docker how to test a container to check that it is still working.
- **SHELL**, allows the default shell used for the shell form of commands to be overridden.

Dockerfile Example

```
# Firefox over VNC
#
# VERSION               0.3

FROM ubuntu

# Install vnc, xvfb in order to create a 'fake' display and firefox
RUN apt-get update && apt-get install -y x11vnc xvfb firefox
RUN mkdir ~/.vnc
# Setup a password
RUN x11vnc -storepasswd 1234 ~/.vnc/passwd
# Autostart firefox (might not be the best way, but it does the trick)
RUN bash -c 'echo "firefox" >> /.bashrc'

EXPOSE 5900
CMD    ["x11vnc", "-forever", "-usepw", "-create"]
```

Docker Hub

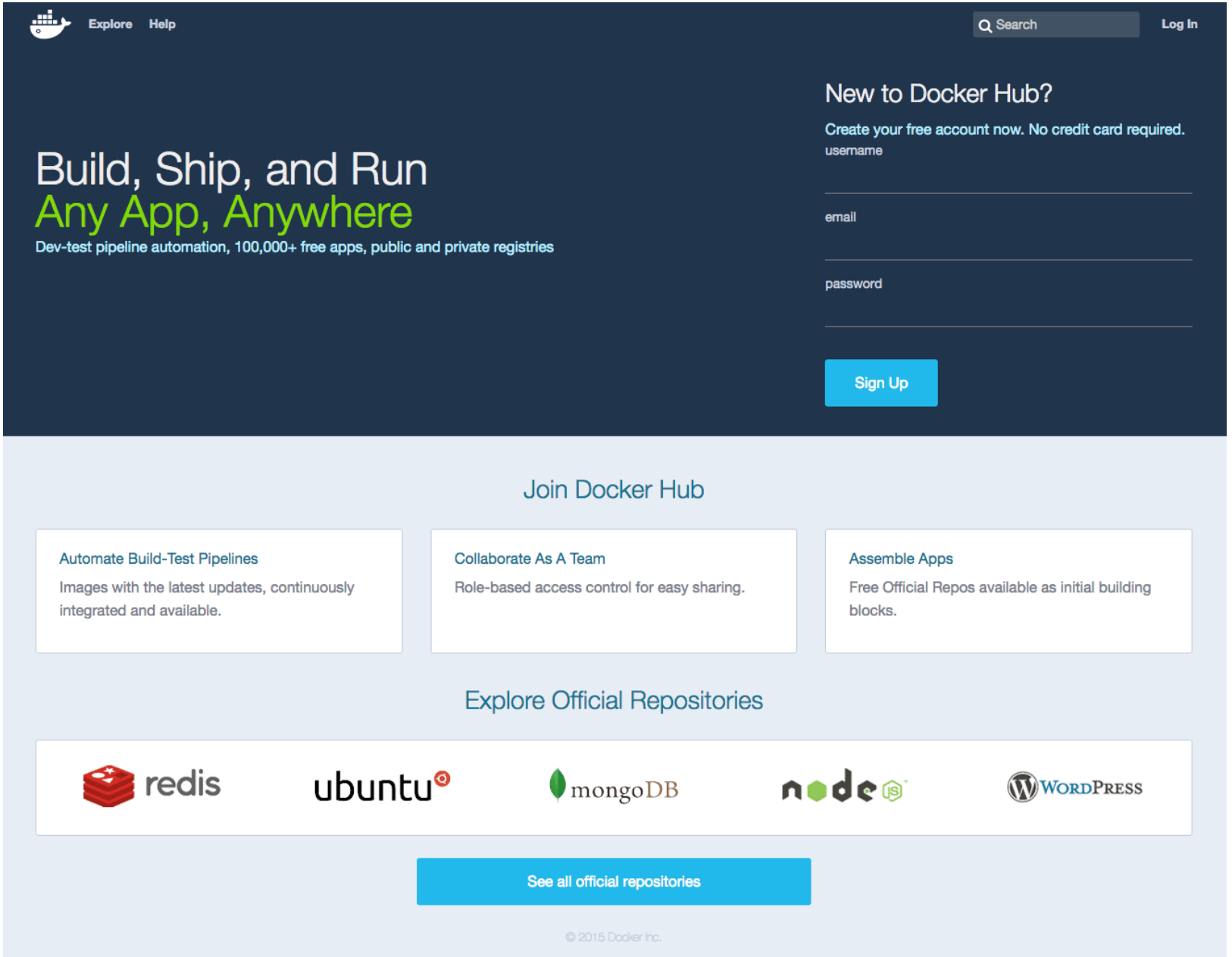
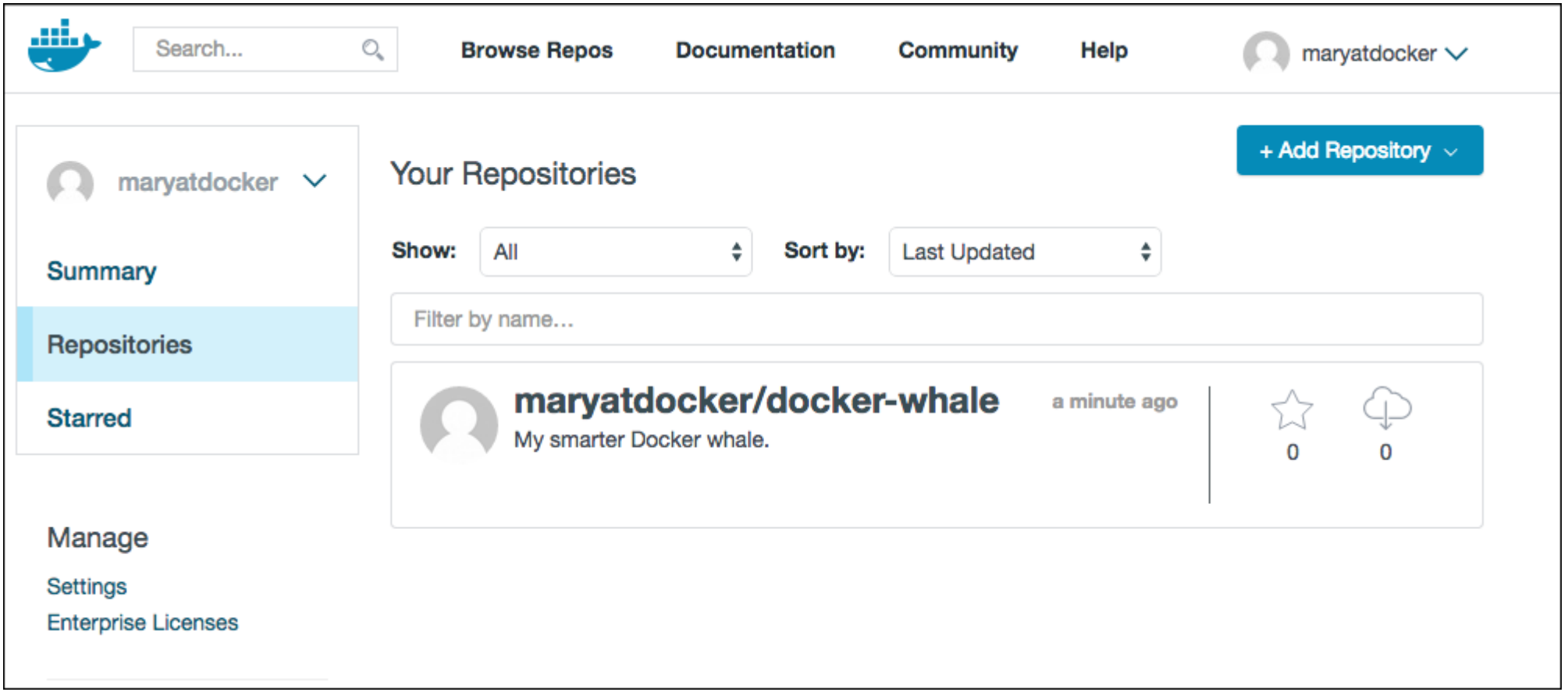
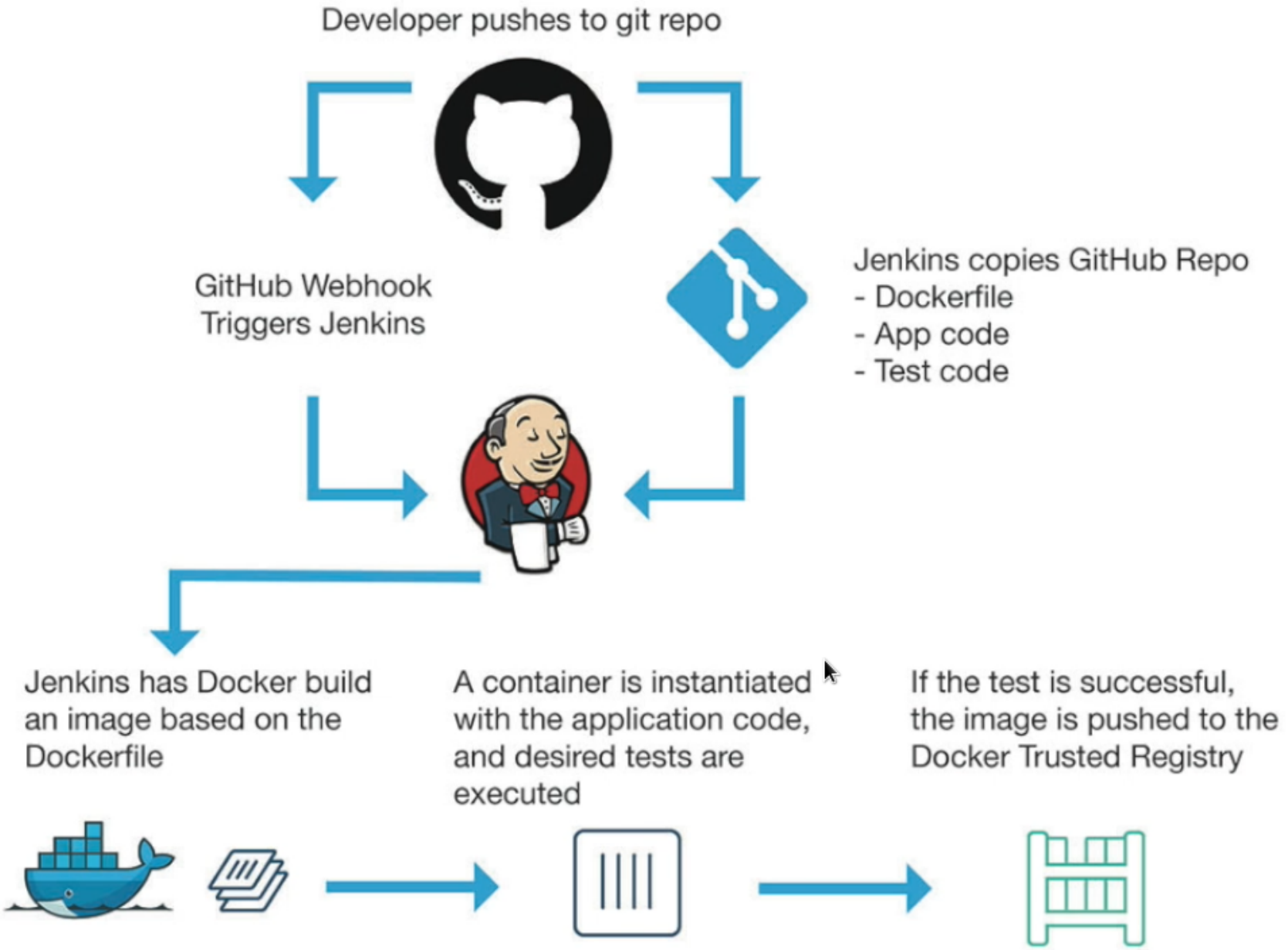

Explore
Help
Search
Log In
Build, Ship, and Run
Any App, Anywhere
Dev-test pipeline automation, 100,000+ free apps, public and private registries
New to Docker Hub?
Create your free account now. No credit card required.
username
email
password
Sign Up
Join Docker Hub
Automate Build-Test Pipelines
Images with the latest updates, continuously integrated and available.
Collaborate As A Team
Role-based access control for easy sharing.
Assemble Apps
Free Official Repos available as initial building blocks.
Explore Official Repositories
redis
ubuntu
mongoDB
node
WordPress
See all official repositories
© 2015 Docker Inc.

Image Repositories



Docker Continous Integration (CI)

CI Using Docker



Docker Hub Automated Build

Build images automatically from a build context stored in a repository. A build context is a Dockerfile and any files at a specific location.

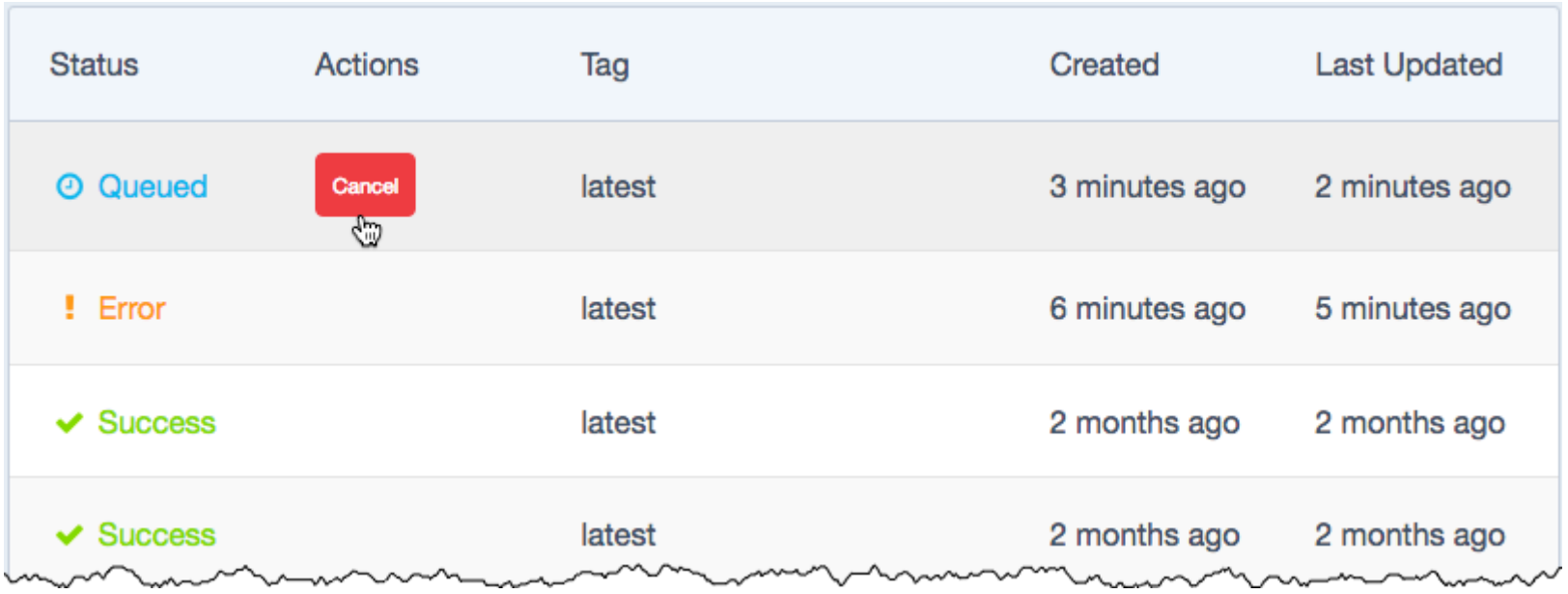
Automated Builds have several advantages:

- Images built in this way are built exactly as specified.
- The Dockerfile is available to anyone with access to your Docker Hub repository.
- Your repository is kept up-to-date with code changes automatically.

Automated Builds are supported for both public and private repositories on both **GitHub** and **Bitbucket**.

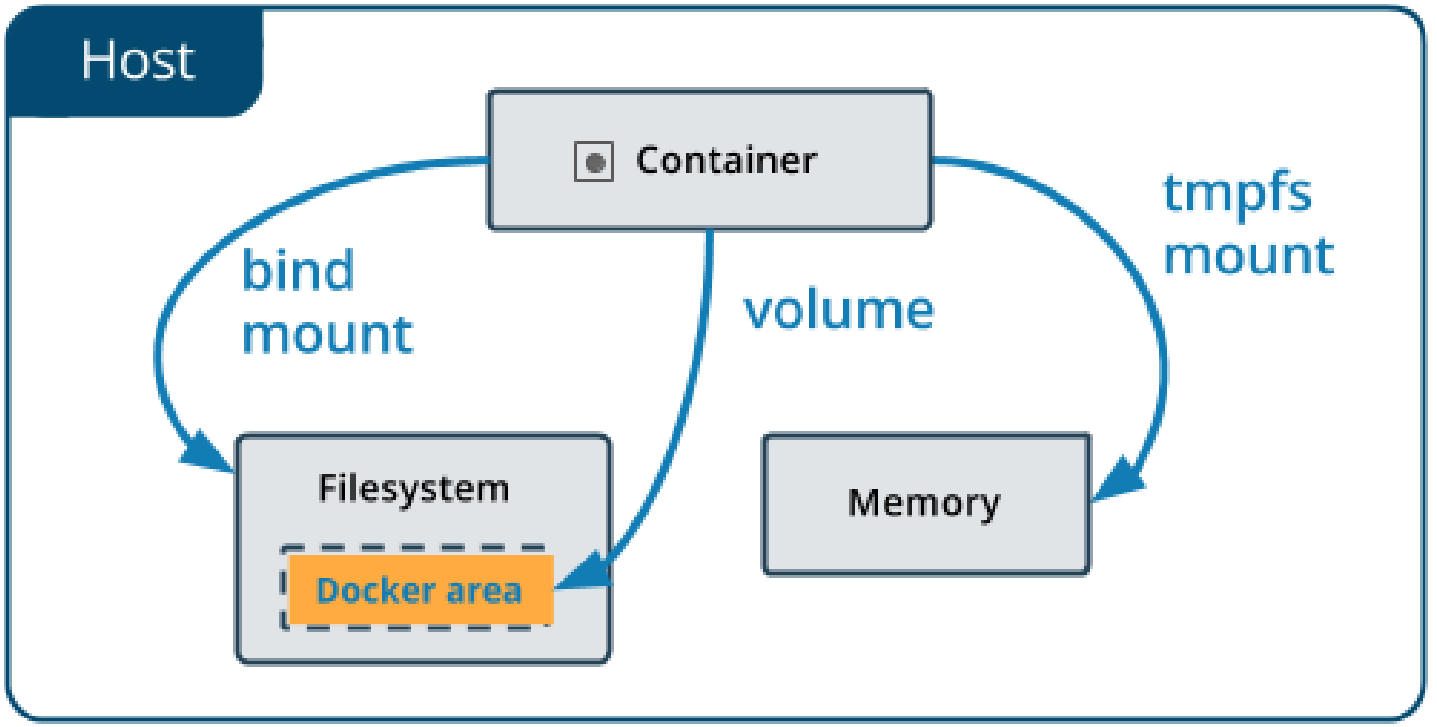
Build Statuses

- **Queued**: in line for image to be built.
- **Building**: The image is building.
- **Success**: The image has been built with no issues.
- **Error**: There was an issue with your image.



Volumes

Bind Mounts & Volumes



Share Data Among Machines