Creating a base image

Most Dockerfiles start from a parent image. If you need to completely control the contents of your image, you might need to create a base image instead. Here's the difference:

- A parent image is the image that your image is based on. It refers to the contents of the `FROM` directive in the Dockerfile. Each subsequent declaration in the Dockerfile modifies this parent image. Most Dockerfiles start from a parent image, rather than a base image. However, the terms are sometimes used interchangeably.
- A base image either has no `FROM` line in its Dockerfile, or has `FROM scratch`.

This topic shows you several ways to create a base image. The specific process will depend heavily on the Linux distribution you want to package. We have some examples below, and you are encouraged to submit pull requests to contribute new ones.

Create a full image using tar

In general, start with a working machine that is running the distribution you'd like to package as a parent image, though that is not required for some tools like Debian's Debootstrap, which you can also use to build Ubuntu images.

It can be as simple as this to create an Ubuntu parent image:

```
$ sudo debootstrap xenial xenial > /dev/null
$ sudo tar -C xenial -c . | docker import - xenial

a29c15f1bf7a

$ docker run xenial cat /etc/lsb-release

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04 LTS"
```

There are more example scripts for creating parent images in the Docker GitHub Repo:

- BusyBox
- CentOS / Scientific Linux CERN (SLC) on Debian/Ubuntu or on CentOS/RHEL/SLC/etc.
- Debian / Ubuntu

Create a simple parent image using scratch

You can use Docker's reserved, minimal image, `scratch`, as a starting point for building containers. Using the `scratch` "image" signals to the build process that you want the next command in the `Dockerfile` to be the first filesystem layer in your image.

While `scratch` appears in Docker's repository on the hub, you can't pull it, run it, or tag any image with the name `scratch`. Instead, you can refer to it in your `Dockerfile`. For example, to create a minimal container using `scratch`:

```
FROM scratch
ADD hello /
CMD ["/hello"]
```

Assuming you built the "hello" executable example by following the instructions at https://github.com/docker-library/hello-world/, and you compiled it with the `-static` flag, you can build this Docker image using this `docker build` command:

```
docker build --tag hello .
```

Don't forget the `.` character at the end, which sets the build context to the current directory.

> **Note**: Because Docker for Mac and Docker for Windows use a Linux VM, you need a Linux binary, rather than a Mac or Windows binary. You can use a Docker container to build it:
>
> ```
> $ docker run --rm -it -v $PWD:/build ubuntu:16.04
>
> container# apt-get update && apt-get install build-essential
> container# cd /build
> container# gcc -o hello -static -nostartfiles hello.c
> ```

To run your new image, use the `docker run` command:

```
docker run --rm hello
```

This example creates the hello-world image used in the tutorials. If you want to test it out, you can clone the image repo.

More resources

There are lots more resources available to help you write your `Dockerfile`.

- There's a complete guide to all the instructions available for use in a `Dockerfile` in the reference section.
- To help you write a clear, readable, maintainable `Dockerfile`, we've also written a `Dockerfile` best practices guide.
- If your goal is to create a new Official Repository, be sure to read up on Docker's Official Repositories.