Containers & Kubernetes

Session #02

Tagging

Docker Commit

Dockerfile

Publish Images

Multi-stage Dockerfile

Lab

Tagging

Image Naming

Tagging

Image name is composed by

registry-server/repository:tag

registry-server

- Define where the image were pulled
- When omitted means "docker.io"

repository

- Define the name of the image
- Can be composed by several levels using '/' as divider

tag

- Define the version of the image
- When omitted gets the value "latest". So "latest" really means "no tag".

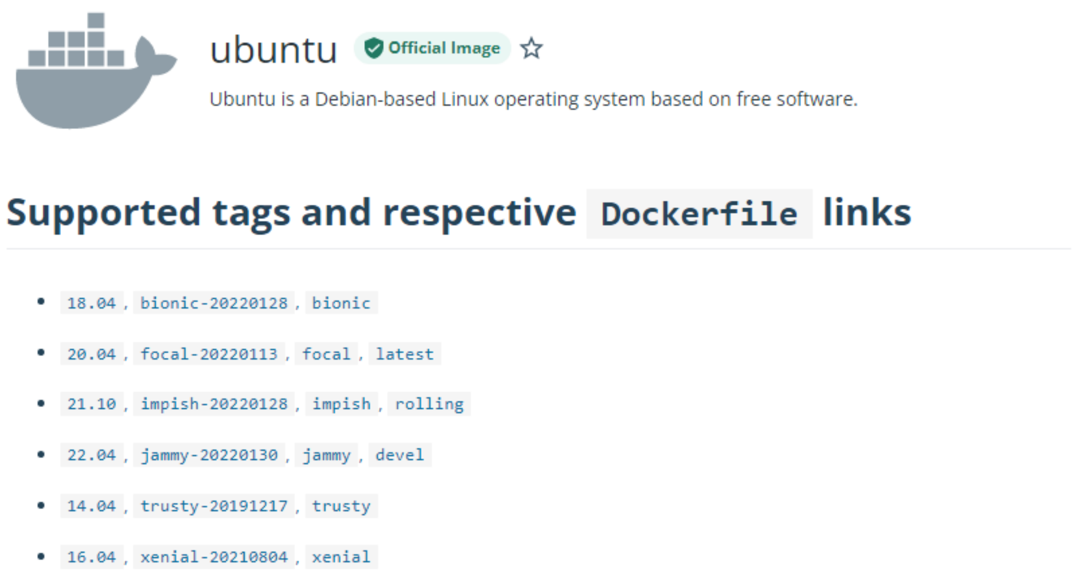
Image Tag

Tagging

- Tag value is textual and open
- Important to identify the image with unique value
- An image with a specific tag on a registry could be overwritten
- “latest” tag should be used very carefully
- Examples
  - 1.0
  - beta
  - 5.0-alpine
  - 20210506.1

Image Tag

Tagging

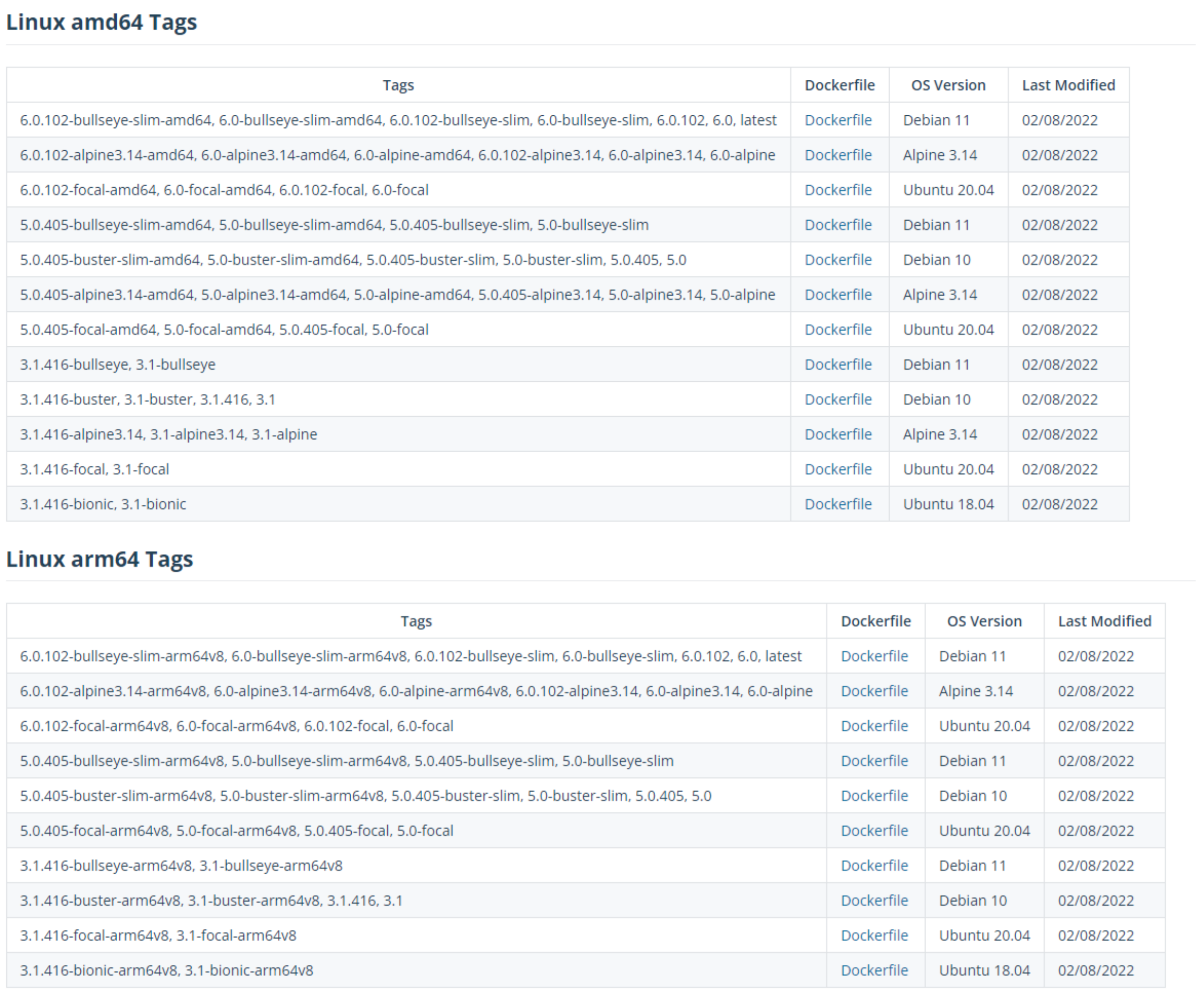


ubuntu Official Image ☆

Ubuntu is a Debian-based Linux operating system based on free software.

Supported tags and respective `Dockerfile` links

- `18.04`, `bionic-20220128`, `bionic`
- `20.04`, `focal-20220113`, `focal`, `latest`
- `21.10`, `impish-20220128`, `impish`, `rolling`
- `22.04`, `jammy-20220130`, `jammy`, `devel`
- `14.04`, `trusty-20191217`, `trusty`
- `16.04`, `xenial-20210804`, `xenial`



Linux amd64 Tags

Tags	Dockerfile	OS Version	Last Modified
6.0.102-bullseye-slim-amd64, 6.0-bullseye-slim-amd64, 6.0.102-bullseye-slim, 6.0-bullseye-slim, 6.0.102, 6.0, latest	Dockerfile	Debian 11	02/08/2022
6.0.102-alpine3.14-amd64, 6.0-alpine3.14-amd64, 6.0-alpine-amd64, 6.0.102-alpine3.14, 6.0-alpine3.14, 6.0-alpine	Dockerfile	Alpine 3.14	02/08/2022
6.0.102-focal-amd64, 6.0-focal-amd64, 6.0.102-focal, 6.0-focal	Dockerfile	Ubuntu 20.04	02/08/2022
5.0.405-bullseye-slim-amd64, 5.0-bullseye-slim-amd64, 5.0.405-bullseye-slim, 5.0-bullseye-slim	Dockerfile	Debian 11	02/08/2022
5.0.405-buster-slim-amd64, 5.0-buster-slim-amd64, 5.0.405-buster-slim, 5.0-buster-slim, 5.0.405, 5.0	Dockerfile	Debian 10	02/08/2022
5.0.405-alpine3.14-amd64, 5.0-alpine3.14-amd64, 5.0-alpine-amd64, 5.0.405-alpine3.14, 5.0-alpine3.14, 5.0-alpine	Dockerfile	Alpine 3.14	02/08/2022
5.0.405-focal-amd64, 5.0-focal-amd64, 5.0.405-focal, 5.0-focal	Dockerfile	Ubuntu 20.04	02/08/2022
3.1.416-bullseye, 3.1-bullseye	Dockerfile	Debian 11	02/08/2022
3.1.416-buster, 3.1-buster, 3.1.416, 3.1	Dockerfile	Debian 10	02/08/2022
3.1.416-alpine3.14, 3.1-alpine3.14, 3.1-alpine	Dockerfile	Alpine 3.14	02/08/2022
3.1.416-focal, 3.1-focal	Dockerfile	Ubuntu 20.04	02/08/2022
3.1.416-bionic, 3.1-bionic	Dockerfile	Ubuntu 18.04	02/08/2022

Linux arm64 Tags

Tags	Dockerfile	OS Version	Last Modified
6.0.102-bullseye-slim-arm64v8, 6.0-bullseye-slim-arm64v8, 6.0.102-bullseye-slim, 6.0-bullseye-slim, 6.0.102, 6.0, latest	Dockerfile	Debian 11	02/08/2022
6.0.102-alpine3.14-arm64v8, 6.0-alpine3.14-arm64v8, 6.0-alpine-arm64v8, 6.0.102-alpine3.14, 6.0-alpine3.14, 6.0-alpine	Dockerfile	Alpine 3.14	02/08/2022
6.0.102-focal-arm64v8, 6.0-focal-arm64v8, 6.0.102-focal, 6.0-focal	Dockerfile	Ubuntu 20.04	02/08/2022
5.0.405-bullseye-slim-arm64v8, 5.0-bullseye-slim-arm64v8, 5.0.405-bullseye-slim, 5.0-bullseye-slim	Dockerfile	Debian 11	02/08/2022
5.0.405-buster-slim-arm64v8, 5.0-buster-slim-arm64v8, 5.0.405-buster-slim, 5.0-buster-slim, 5.0.405, 5.0	Dockerfile	Debian 10	02/08/2022
5.0.405-focal-arm64v8, 5.0-focal-arm64v8, 5.0.405-focal, 5.0-focal	Dockerfile	Ubuntu 20.04	02/08/2022
3.1.416-bullseye-arm64v8, 3.1-bullseye-arm64v8	Dockerfile	Debian 11	02/08/2022
3.1.416-buster-arm64v8, 3.1-buster-arm64v8, 3.1.416, 3.1	Dockerfile	Debian 10	02/08/2022
3.1.416-focal-arm64v8, 3.1-focal-arm64v8	Dockerfile	Ubuntu 20.04	02/08/2022
3.1.416-bionic-arm64v8, 3.1-bionic-arm64v8	Dockerfile	Ubuntu 18.04	02/08/2022

Docker Commit

How to create an Image?

Docker commit

- We want to package our application/solution/tool on an image to share
- “docker commit” allow to create an image based on an existing container
- Process consists in transforming container writable layer in a readonly layer
- New layer is added to the list of layers from the base image

How to create an Image?

Docker commit

- This process is not the most usual way to create new images
- Can be useful to have access to broken containers/processes
- Needs to be executed on a stopped container
- Can be a good (unique?) way to have access to files inside a stopped container

Dockerfile

How to create an Image?

Dockerfile

- Text file with Docker commands in it to create a new image
- Can be seen as a configuration file with set of instructions needed to assemble a new image.
- Docker has a docker build command that parses Dockerfile to build a new container image
- Each command creates a new layer
- Needs to be created to take the most from caching strategy
- Commands can be executed in parallel so grouping commands can be the way to guarantee sequence
- Can be used on automated image creation

Most used commands

Dockerfile

FROM instruction initializes a new build stage and sets the Base Image for subsequent instructions.

LABEL is a key-value pair, stored as a string. You can specify multiple labels for an object, but each key-value pair must be unique within an object.

RUN will execute any commands in a new layer on top of the current image and commit the results.

WORKDIR instruction sets the working directory for any RUN,CMD, ENTRYPOINT, COPY and ADD instructions that follow it.

ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.

CMD provide defaults for an executing container. These defaults can include an executable.

ENTRYPOINT allows you to configure a container that will run as an executable.

EXPOSE instruction informs Docker that the container listens on the specified network port(s).

Best practices

Dockerfile

- Minimize the number of steps in the Dockerfile
- Start your Dockerfile with the steps that are least likely to change
- Clean up your Dockerfile
- Use a .dockerignore file
- Containers should be ephemeral
- One container should have one concern
- More on docker official docs: Best practices for writing Dockerfiles | Docker Documentation

Demo: Dockerfile

Publish Images

How to share an Image?

Publish Image

- Based on image tagging
- Uses repository server on image name to define where to publish
- Can be publish to public or private registry
- Private registry usually needs authn/authz

How to authenticate

```
docker login <server> -u <username>
```

How to publish

```
docker push <image_name>
```

Demo: Tagging & Publish

Multi-stage Dockerfile

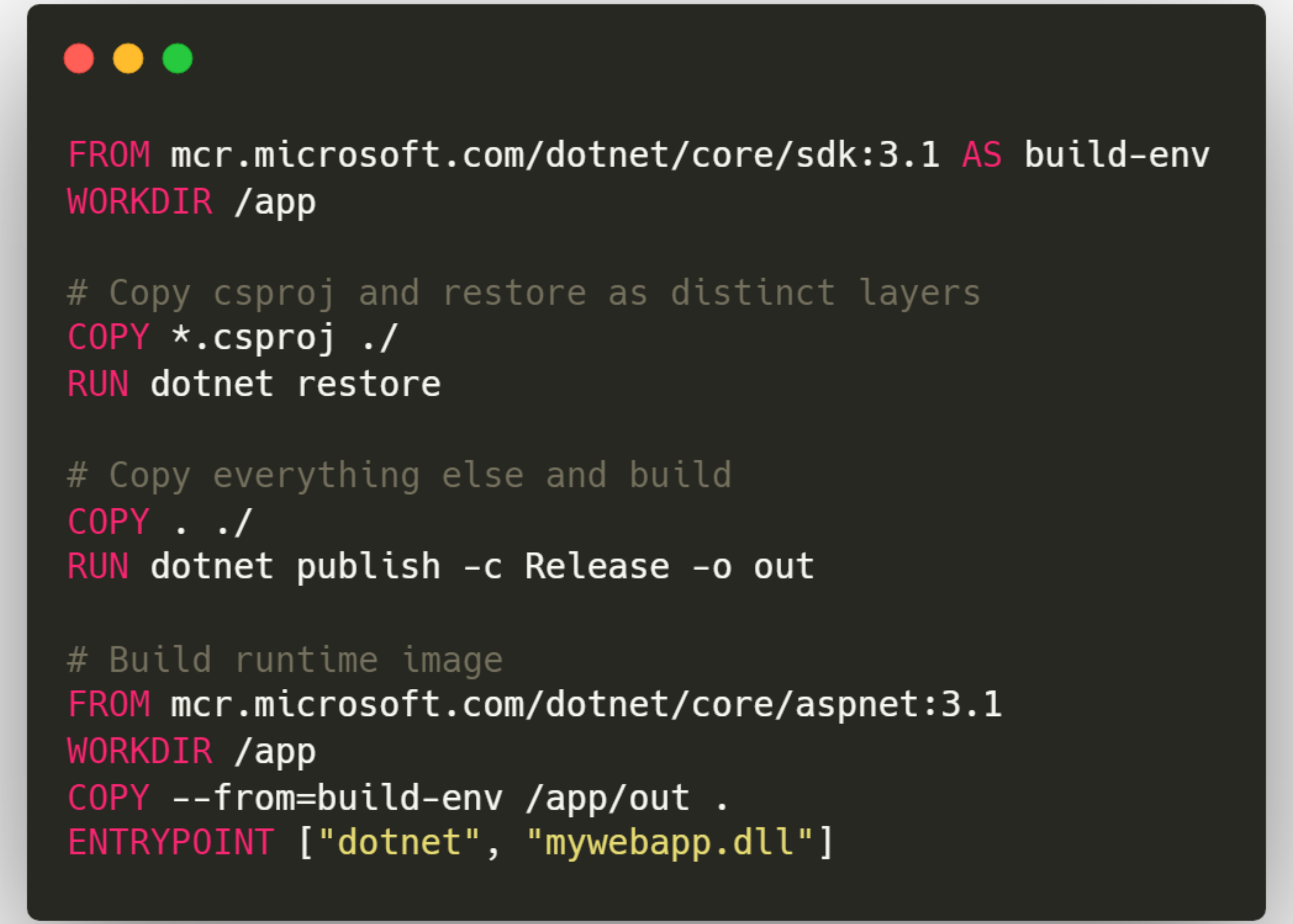
Integrate all phases on build

Multi-stage Dockerfile

- Containers bring total autonomy on execution
- To create images we need to have access to all needed binaries
- Define all phases of build process inside a container
- Before several Dockerfiles was used
- Benefits
  - Autonomy
  - All needed libraries/frameworks available independently on the host
  - Allow partial execution (docker build –target=<stage_name>
  - Multiple tooling for application creation
  - Full images to build phase, slim images to runtime images

Integrate all phases on build

Multi-stage Dockerfile



```
FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build-env
WORKDIR /app

# Copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore

# Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out

# Build runtime image
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "mywebapp.dll"]
```

Demo: Multi-stage Dockerfile