# Developer Tooling for Cloud Native Applications

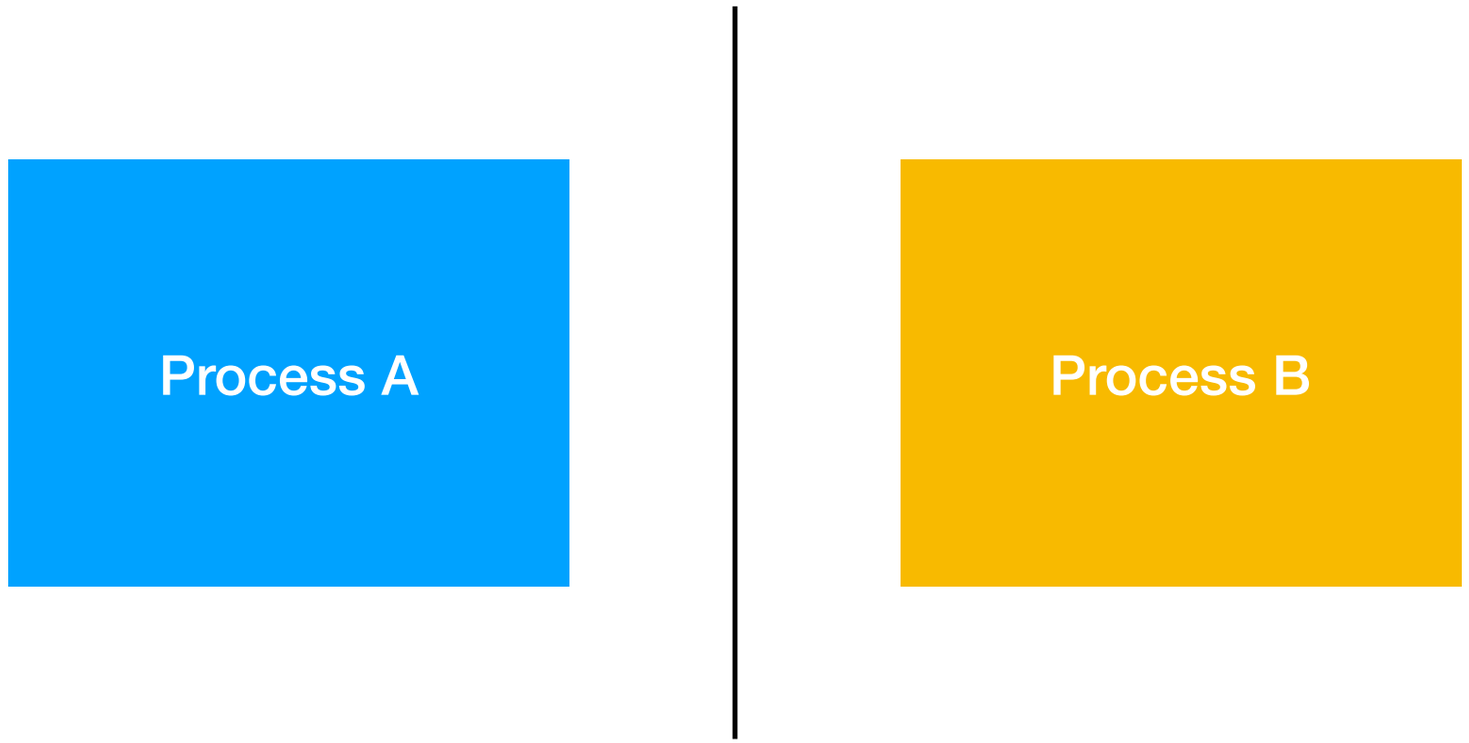
## Dive, multistage-builds, docker-squash

Raghavendra Sirigeri
Founder, Questodev
(https://www.linkedin.com/in/raghavendra-sirigeri/)

# Agenda

- Containers under the hood
- Overlay Filesystems
- Docker Image Layering
- Optimising Build Times
- Docker-dive for Image analysis
- Docker-squash for squashing layers

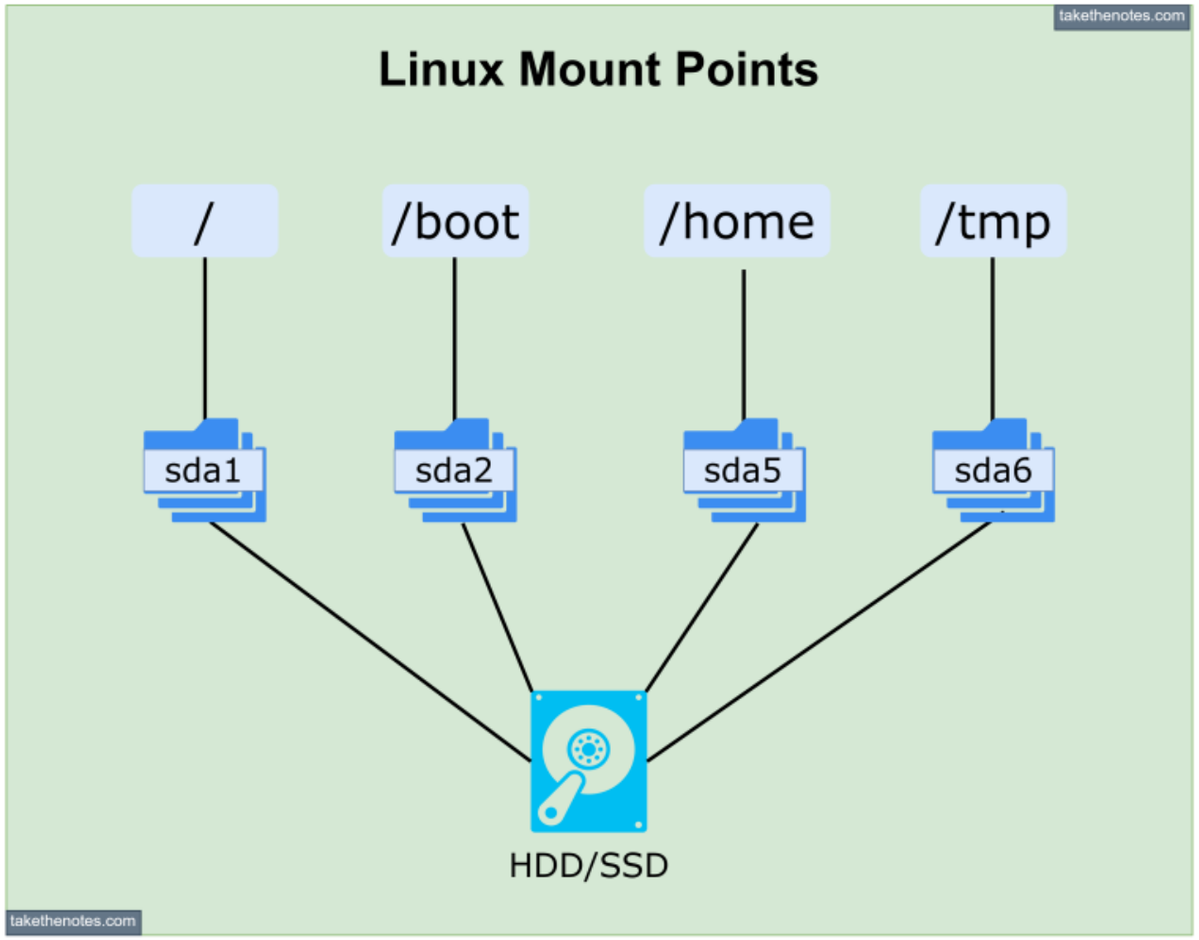
# How can we isolate two processes?




## Enter NAMESPACES..

DEMO

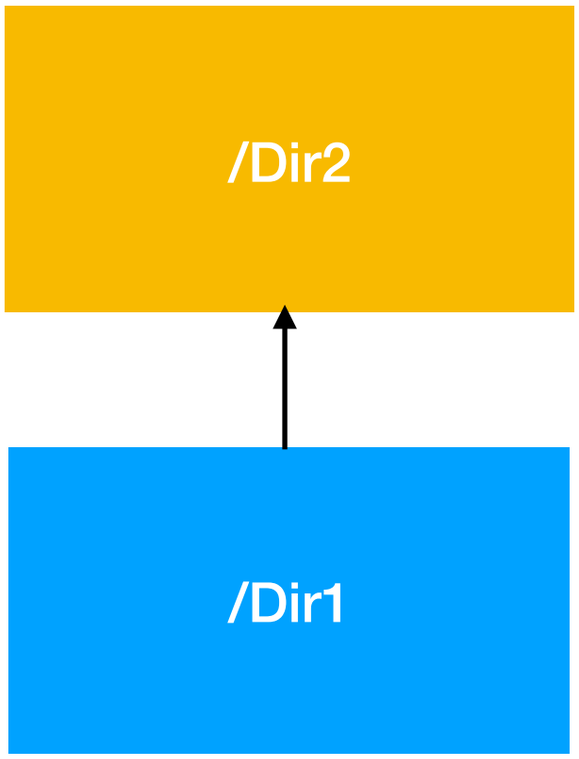
# Mounting and Mount Points




Mounting is the process of making a filesystem accessible at a certain point in the directory tree of a system.

A mount point is a directory in the filesystem where a mounted filesystem becomes accessible

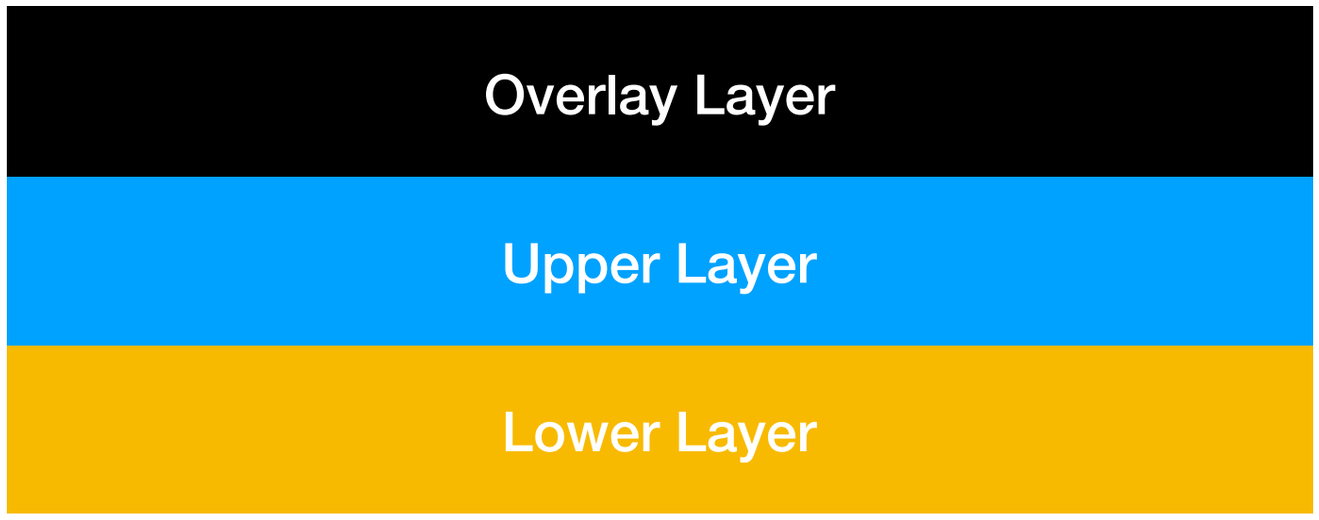
# Bind Mount Point



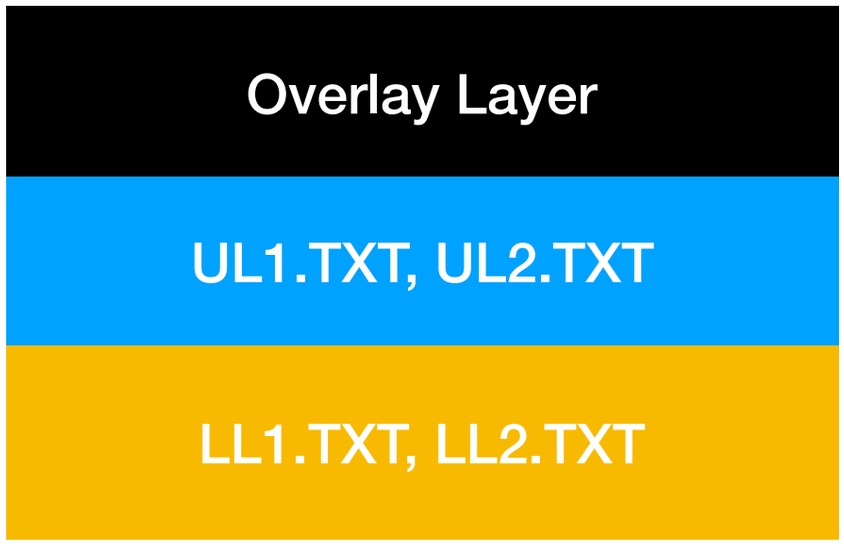

**A bind mount is a type of mount in Linux and Unix-like operating systems where a directory or file is mounted to another location in the filesystem. Unlike a traditional mount, which typically involves mounting a filesystem from a storage device, a bind mount simply creates an additional reference to an existing directory or file in a different location.**

# Overlay Filesystems - Quick Walkthrough

Essentially comprises of 3 layers



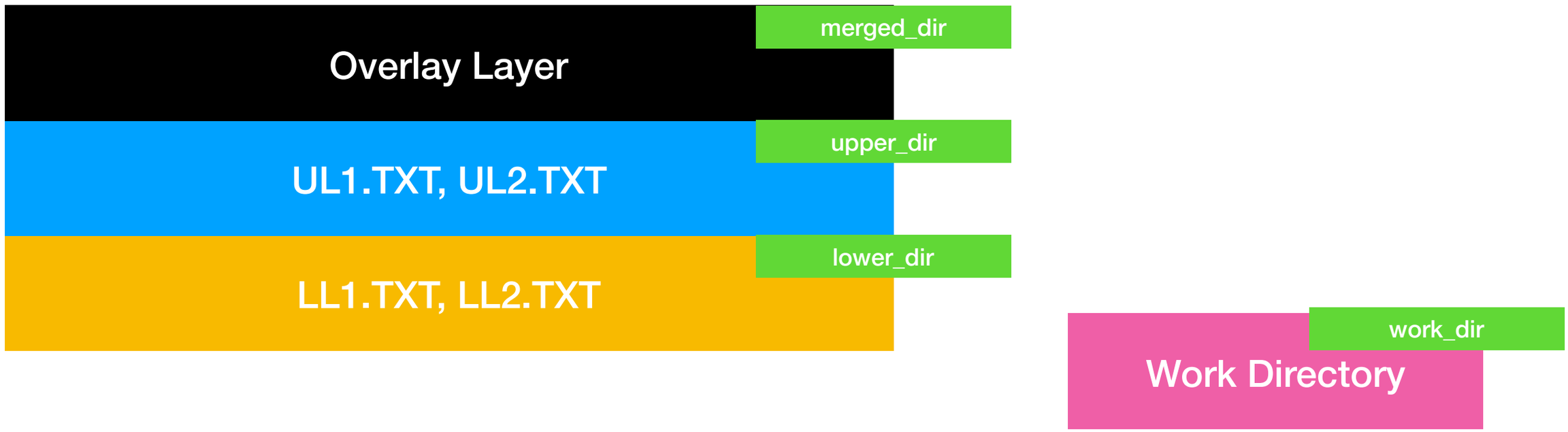
# Overlay Filesystems - Quick Walkthrough





- Overlay FS is a type of Union FS
- Union of all files and dirs from Lower and Upper Layer
- Lower Layer is READ-ONLY
- Any change done to files originally from Lower Layer in the Overlay Layer will create new COPY-ON-WRITE file of the the modified file in the upper layer as the Lower layer is READ-ONLY
- When you modify any of the files in the overlay layer which were originally from the lower layer, a copy (COPY-ON-WRITE) of that gets created in the upper directory and that is where the modifications go.

# Overlay Filesystems - Demo



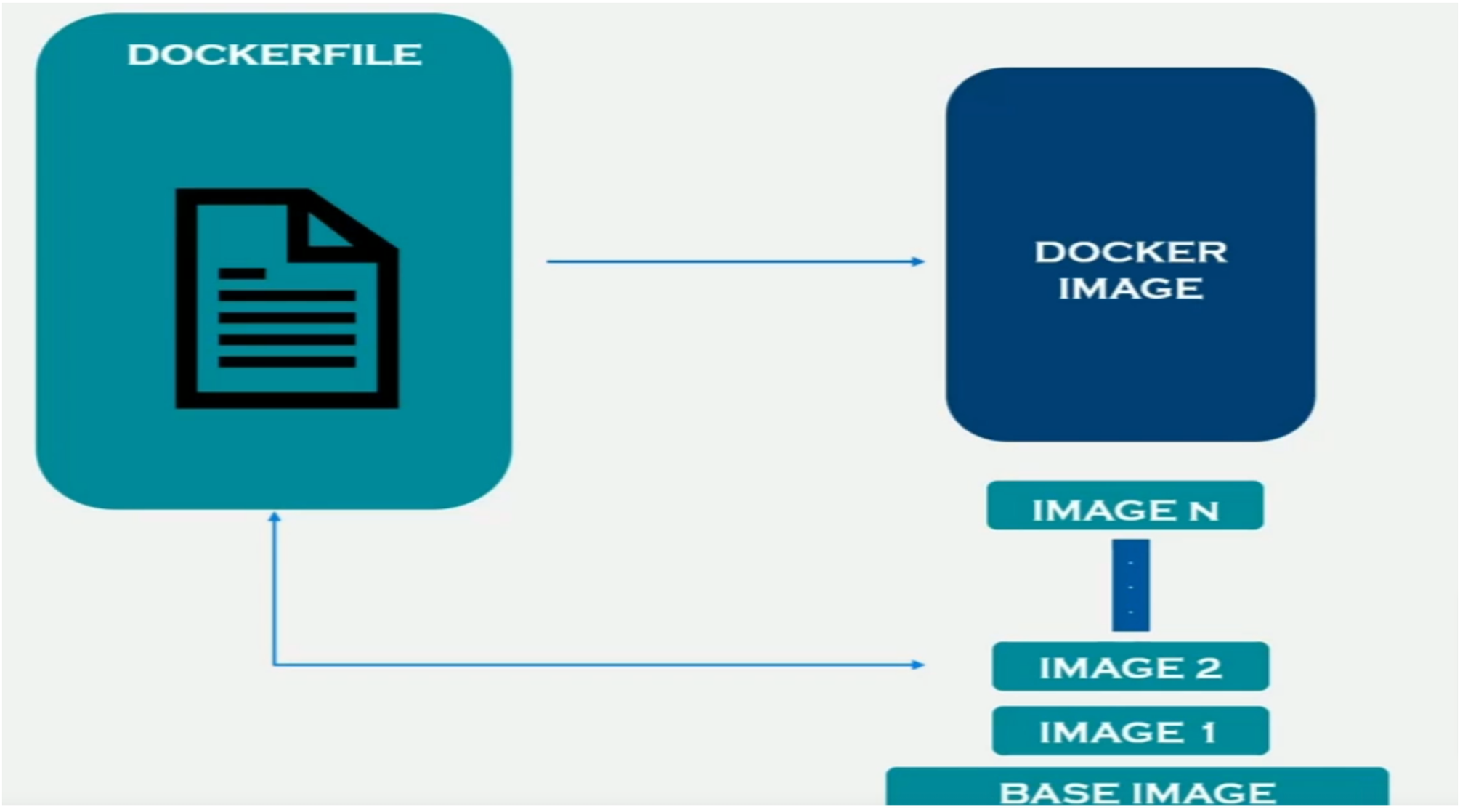

```
$ sudo mount -t overlay -o
lowerdir=lower_dir/,upperdir=upper_dir,workdir=work_dir/ none
merged_dir/
```

Work Directory is used by the system as a temporary work area for internal purposes

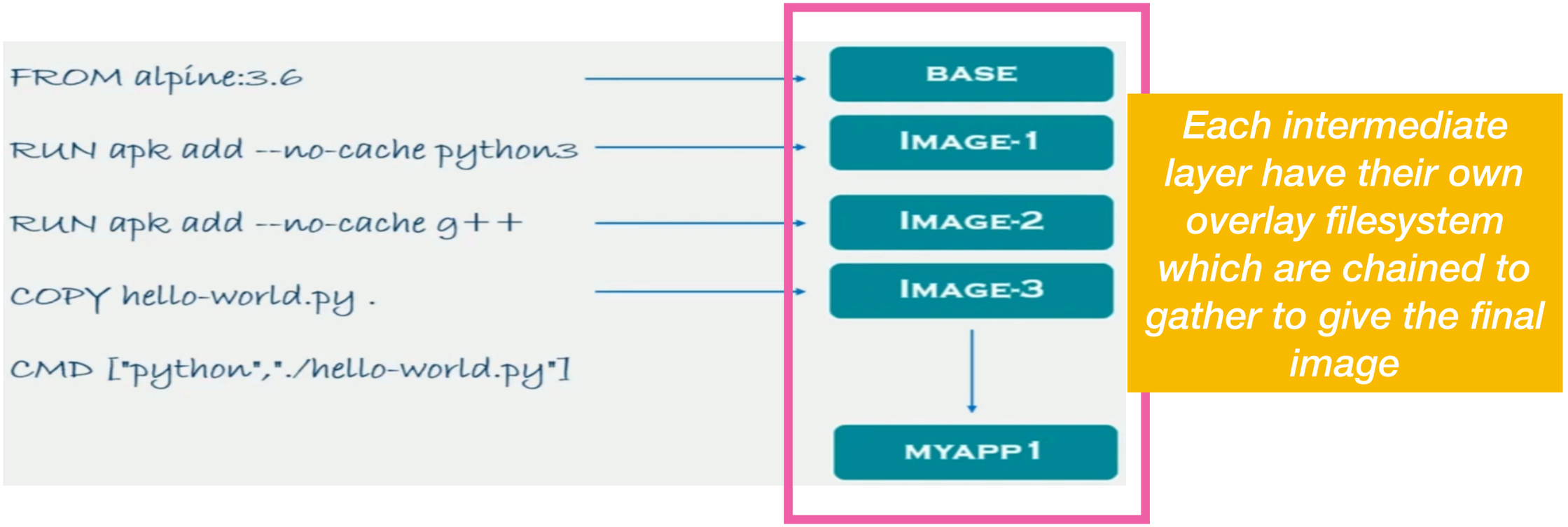
# Overlay Filesystems - Quick Walkthrough

DEMO

# How to build a container image

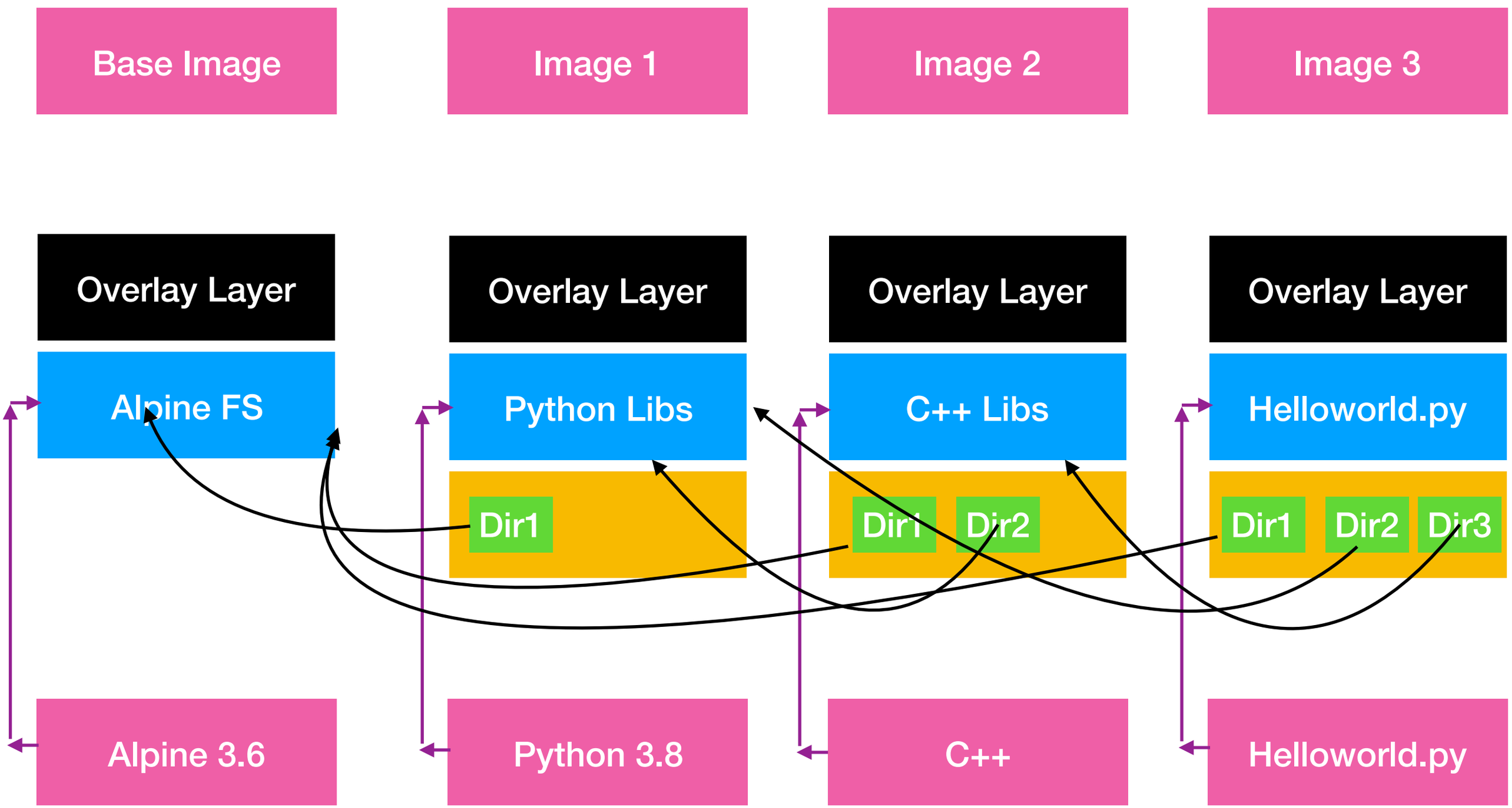


# Dissecting Container Images




DEMO

# Image Layering in Depth




**Legend** Upper Layer | Lower Layer

# Scenario 1 - Simple Node.js App

```
FROM node:22
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
CMD node server.js
```

**vs**

```
FROM node:22
WORKDIR /app
COPY . .
RUN npm install
CMD node server.js
```

***Hint: Think in terms of subsequent build times when source code changes***

# Scenario 1 - Simple Node.js App

```
FROM node:22
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
CMD node server.js
```

***These layers are cached as no changes are done here and can be easily retrieved in subsequent builds***

**VS**

```
FROM node:22
WORKDIR /app
COPY . .
RUN npm install
CMD node server.js
```

# Scenario 1 - Simple Node.js App

```
FROM node:22
RUN apt-get update
RUN apt-get install sqlite3
WORKDIR /app
COPY package.json package-lock.
RUN npm install
COPY . .
CMD node server.js
```

*If a new package has to be installed the cached layer corresponding to apt-get update will contain old package cache and the newly installed package may be outdated.*

**vs**

```
FROM node:22
RUN apt-get update && apt-get install sqlite3 redis
WORKDIR /app
COPY . .
RUN npm install
CMD node server.js
```

# Optimising Build Times

- During incremental build we want to ensure that we can use build cache effectively to avoid high build times. For example COPY . /app should not be done early in the Dockerfile as any changes to the code will invalidate the build cache.
  **ORDER FROM LEAST TO MOST FREQUENTLY CHANGING CONTENT.**

- Only copy whats needed. Avoid COPY . if possible because any changes to the files will bust the cache

- Line buddies - apt-get update and apt-get install should be done together rather than in separate lines. If a new package has to be installed the cached layer corresponding to apt-get update will contain old package cache and the newly installed package may be outdated.

# Scenario 2 - Reducing Docker Image Size

```
FROM node:22
WORKDIR /app
RUN apt-get update && apt-
get install libcairo2-dev
libjpeg-dev libgif-dev
libpango1.0-dev -y
COPY package.json package-
lock.json ./
RUN npm install
COPY . .
CMD node server.js
```

**Results in an image of size 1.3GB**

```
{
  "name": "scenario-1-simple-node-
app",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test
specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^4.19.2"
  },
  "devDependencies": {
    "eslint": "^9.5.0",
    "mocha": "^10.4.0",
    "nodemon": "^3.1.4"
  }
}
```

***Do we need these during runtime? Not really***

# Scenario 2 - Reducing Docker Image Size

Node:22 base image layers and size is more than 1 GB

```
● Layers
Cmp   Size  Command
    117 MB  FROM b3b31c0586cb09b
     48 MB  set -eux;     apt-get update;     apt-get install -y --no-insta
    177 MB  set -eux;     apt-get update;     apt-get install -y --no-insta
    587 MB  set -ex;     apt-get update;     apt-get install -y --no-instal
    8.9 kB  groupadd --gid 1000 node   && useradd --uid 1000 --gid node --s
    174 MB  ARCH= && dpkgArch="$(dpkg --print-architecture)"   && case "${d
    5.3 MB  set -ex   && export GNUPGHOME="$(mktemp -d)"   && for key in
    388 B   #(nop) COPY file:4d192565a7220e135cab6c77fbc1c73211b69f3d9fb37e
      0 B   WORKDIR /app
     34 MB  RUN /bin/sh -c apt-get update && apt-get install libcairo2-dev
    234 kB  COPY package.json package-lock.json ./ # buildkit
    113 MB  RUN /bin/sh -c npm install # buildkit

Layer Details

Tags:   (unavailable)
Id:     dc030281fbe958a4420719b49b9495a533634698b5496875ce2e4d456bb1f875
Digest: sha256:060f5da969763c7b24cfcfbfc2d5607741e427e8397f17fab88cdd3c5aa7
2518
Command:
#(nop) COPY file:4d192565a7220e135cab6c77fbc1c73211b69f3d9fb37e62857b2c6eb9
363d51 in /usr/local/bin/

Image Details

Image name: node-sample-without-multi
Total Image size: 1.3 GB
Potential wasted space: 10 MB
Image efficiency score: 99 %

Count   Total Space  Path
    4        3.2 MB  /var/cache/debconf/templates.dat
    4        3.1 MB  /var/cache/debconf/templates.dat-old
    5        1.1 MB  /var/lib/dpkg/status
    5        1.1 MB  /var/lib/dpkg/status-old
    3        644 kB  /app/package-lock.json
    4        382 kB  /var/log/dpkg.log
```

```
Current Layer Contents
Permission     UID:GID       Size  Filetree
-rwxrwxrwx         0:0        0 B  ├── bin → usr/bin
drwxr-xr-x         0:0        0 B  ├── boot
drwxr-xr-x         0:0        0 B  ├── dev
drwxr-xr-x         0:0     851 kB  ├── etc
-rw-------         0:0        0 B  │   ├── .pwd.lock
drwxr-xr-x         0:0     244 kB  │   ├── ImageMagick-6
-rw-r--r--         0:0      899 B  │   │   ├── coder.xml
-rw-r--r--         0:0     1.4 kB  │   │   ├── colors.xml
-rw-r--r--         0:0      14 kB  │   │   ├── delegates.xml
-rw-r--r--         0:0     1.6 kB  │   │   ├── log.xml
-rw-r--r--         0:0      888 B  │   │   ├── magic.xml
-rw-r--r--         0:0      134 kB  │   │   ├── mime.xml
-rw-r--r--         0:0     4.7 kB  │   │   ├── policy.xml
-rw-r--r--         0:0     2.4 kB  │   │   ├── quantization-table.xml
-rw-r--r--         0:0      12 kB  │   │   ├── thresholds.xml
-rw-r--r--         0:0      29 kB  │   │   ├── type-apple.xml
-rw-r--r--         0:0     8.5 kB  │   │   ├── type-dejavu.xml
-rw-r--r--         0:0     9.7 kB  │   │   ├── type-ghostscript.xml
-rw-r--r--         0:0      10 kB  │   │   ├── type-urw-base35.xml
-rw-r--r--         0:0      14 kB  │   │   ├── type-windows.xml
-rw-r--r--         0:0      651 B  │   │   └── type.xml
drwxr-xr-x         0:0      29 kB  │   ├── X11
-rwxr-xr-x         0:0      709 B  │   │   ├── Xreset
drwxr-xr-x         0:0      205 B  │   │   ├── Xreset.d
-rw-r--r--         0:0      205 B  │   │   │   └── README
drwxr-xr-x         0:0      319 B  │   │   ├── Xresources
-rw-r--r--         0:0      319 B  │   │   │   └── x11-common
-rwxr-xr-x         0:0     3.9 kB  │   │   ├── Xsession
drwxr-xr-x         0:0     6.3 kB  │   │   ├── Xsession.d
-rw-r--r--         0:0     1.9 kB  │   │   │   ├── 20x11-common_process-arg
-rw-r--r--         0:0      878 B  │   │   │   ├── 30x11-common_xresources
-rw-r--r--         0:0      389 B  │   │   │   ├── 35x11-common_xhost-local
-rw-r--r--         0:0      187 B  │   │   │   ├── 40x11-common_xsessionrc
-rw-r--r--         0:0     1.6 kB  │   │   │   ├── 50x11-common_determine-s
-rw-r--r--         0:0      880 B  │   │   │   ├── 90gpg-agent
-rw-r--r--         0:0      385 B  │   │   │   ├── 90x11-common_ssh-agent
-rw-r--r--         0:0      166 B  │   │   │   └── 99x11-common_start
-rw-r--r--         0:0      265 B  │   │   ├── Xsession.options
-rw-r--r--         0:0       17 kB  │   │   ├── rgb.txt
drwxr-xr-x         0:0        0 B  │   │   └── xorg.conf.d
-rw-r--r--         0:0     3.0 kB  │   ├── adduser.conf
drwxr-xr-x         0:0      100 B  │   ├── alternatives
-rw-r--r--         0:0      100 B  │   │   ├── README
-rwxrwxrwx         0:0        0 B  │   │   ├── aclocal → /usr/bin/aclocal-1
-rwxrwxrwx         0:0        0 B  │   │   ├── aclocal.1.gz → /usr/share/ma
```

^C Quit | Tab Switch view | ^F Filter | ^L Show layer changes | ^A Show aggregated changes

# Scenario 2 - Multi-stage builds

```
FROM node:22 as builder
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
```

**BUILD Stage**

```
FROM node:22-slim
WORKDIR /app
COPY --from=builder /app .
RUN npm install --only=production

CMD node server.js
```

**RUNTIME Stage**

**Huge Reduction in image size from 1.2 GB to < 300MB**

# Scenario 3 - Installing awscli in our Docker Image

Image size of 777 MB

```
FROM node:22 as builder
WORKDIR /app
RUN apt-get update && apt-get install libcairo2-dev
libjpeg-dev libgif-dev libpango1.0-dev -y
COPY package.json package-lock.json ./
RUN npm install
COPY . .

#Runtime
FROM node:22-slim
WORKDIR /app
COPY --from=builder /app .
RUN npm install --only=production

# Installing awscliv2
RUN apt-get update && apt-get install curl unzip -y
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-
x86_64.zip" -o "awscliv2.zip"
RUN unzip awscliv2.zip && ./aws/install

CMD node server.js
```

# Scenario 3 - Installing awscli in our Docker Image

```
| Layers ├──────────────────────────────────────────────────────────────────────
Cmp   Size  Command
    75 MB  FROM 0417392ea6b973f
   8.9 kB  groupadd --gid 1000 node   && useradd --uid 1000 --gid node --s
   130 MB  ARCH= OPENSSL_ARCH= && dpkgArch="$(dpkg --print-architecture)"
   7.2 MB  set -ex   && savedAptMark="$(apt-mark showmanual)"   && apt-get
    388 B  #(nop) COPY file:4d192565a7220e135cab6c77fbc1c73211b69f3d9fb37e
      0 B  WORKDIR /app
    48 MB  COPY /app . # buildkit
   320 kB  RUN /bin/sh -c npm install --only=production # buildkit
    35 MB  RUN /bin/sh -c apt-get update && apt-get install curl unzip -y
    61 MB  RUN /bin/sh -c curl "https://awscli.amazonaws.com/awscli-exe-li
   420 MB  RUN /bin/sh -c unzip awscliv2.zip && ./aws/install # buildkit
```

```
| ● Current Layer Contents ├──────────────────────────────────────────────────
Permission     UID:GID       Size  Filetree
drwxr-xr-x         0:0      95 MB  ├── app
-rw-r--r--         0:0       26 B  │   ├── .dockerignore
-rw-r--r--         0:0       519 B │   ├── Dockerfile-v1
-rw-r--r--         0:0       61 MB │   ├── awscliv2.zip
drwxr-xr-x         0:0       34 MB │   ├⊕ node_modules
-rw-r--r--         0:0      176 kB │   ├── package-lock.json
-rw-r--r--         0:0      482 B  │   ├── package.json
-rw-r--r--         0:0      232 B  │   └── server.js
-rwxrwxrwx         0:0        0 B  ├── bin → usr/bin
drwxr-xr-x         0:0        0 B  ├── boot
drwxr-xr-x         0:0        0 B  ├── dev
drwxr-xr-x         0:0      406 kB ├⊕ etc
```

```
| Layers ├──────────────────────────────────────────────────────────────────────
Cmp   Size  Command
    75 MB  FROM 0417392ea6b973f
   8.9 kB  groupadd --gid 1000 node   && useradd --uid 1000 --gid node --s
   130 MB  ARCH= OPENSSL_ARCH= && dpkgArch="$(dpkg --print-architecture)"
   7.2 MB  set -ex   && savedAptMark="$(apt-mark showmanual)"   && apt-get
    388 B  #(nop) COPY file:4d192565a7220e135cab6c77fbc1c73211b69f3d9fb37e
      0 B  WORKDIR /app
    48 MB  COPY /app . # buildkit
   320 kB  RUN /bin/sh -c npm install --only=production # buildkit
    35 MB  RUN /bin/sh -c apt-get update && apt-get install curl unzip -y
    61 MB  RUN /bin/sh -c curl "https://awscli.amazonaws.com/awscli-exe-li
   420 MB  RUN /bin/sh -c unzip awscliv2.zip && ./aws/install # buildkit
```

```
| ● Current Layer Contents ├──────────────────────────────────────────────────
Permission     UID:GID       Size  Filetree
drwxr-xr-x         0:0      305 MB  ├── app
-rw-r--r--         0:0       26 B  │   ├── .dockerignore
-rw-r--r--         0:0       519 B │   ├── Dockerfile-v1
drwxr-xr-x         0:0      210 MB │   ├── aws
-rw-r--r--         0:0      1.5 kB │   │   ├── README.md
-rw-r--r--         0:0       68 kB │   │   ├── THIRD_PARTY_LICENSES
drwxr-xr-x         0:0      210 MB │   │   ├⊕ dist
-rwxr-xr-x         0:0      4.0 kB │   │   └── install
-rw-r--r--         0:0       61 MB │   ├── awscliv2.zip
drwxr-xr-x         0:0       34 MB │   ├⊕ node_modules
-rw-r--r--         0:0      176 kB │   ├── package-lock.json
-rw-r--r--         0:0      482 B  │   ├── package.json
-rw-r--r--         0:0      232 B  │   └── server.js
```

# Scenario 3 - Installing awscli in our Docker Image

```
FROM node:22 as builder
WORKDIR /app
RUN apt-get update && apt-get install libcairo2-dev
libjpeg-dev libgif-dev libpango1.0-dev -y
COPY package.json package-lock.json ./
RUN npm install
COPY . .

#Runtime
FROM node:22-slim
WORKDIR /app
COPY --from=builder /app .
RUN npm install --only=production

# Installing awscliv2
RUN apt-get update && apt-get install curl unzip -y
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-
x86_64.zip" -o "awscliv2.zip"
RUN unzip awscliv2.zip && ./aws/install
RUN rm -rf awscliv2.zip ./aws

CMD node server.js
```

Image size is still 777 MB
Whats happening here?

Hint: Overlay filesystems Lower Layer Characteristic

# Scenario 3 - Installing awscli in our Docker Image

```
FROM node:22 as builder
WORKDIR /app
RUN apt-get update && apt-get install libcairo2-dev
libjpeg-dev libgif-dev libpango1.0-dev -y
COPY package.json package-lock.json ./
RUN npm install
COPY . .

#Runtime
FROM node:22-slim
WORKDIR /app
COPY --from=builder /app .
RUN npm install --only=production

# Installing awscliv2
RUN apt-get update && apt-get install curl unzip -y
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-
x86_64.zip" -o “awscliv2.zip" && unzip awscliv2.zip
&& ./aws/install && rm -rf awscliv2.zip ./aws

CMD node server.js
```

Image size is now 506 MB

# Scenario 4 - Docker squash

Once the image is built, docker-squash combines the new layers into a new image with a single new layer. Squashing doesn't destroy any existing image, rather it creates a new image with the content of the squashed layers. This effectively makes it look like all `Dockerfile` commands were created with a single layer.

Squashing layers can be beneficial if your Dockerfile produces multiple layers modifying the same files. For example, files created in one step and removed in another step.

We saw this with our awscli package where we wanted to get rid of the download files and bloatware but using the rm command had no effect.

## LETS NOW TRY TO SQUASH THE NODE-SAMPLE-AWSCLI-IMAGE!

**$ docker-squash -t node-sample-after-squash node-sample-awscli-rm-v1**

# Scenario 4 - Docker squash

LETS NOW TRY TO SQUASH THE NODE-SAMPLE-AWSCLI-IMAGE!

**$ docker-squash -t node-sample-after-squash node-sample-awscli-rm-v1**

NEW IMAGE IS close to 480 MB compare to the original 770 MB!

# Scenario 4 - Docker squash

## Limitations -

- When squashing layers, the resulting image can't take advantage of layer sharing with other images, and may use significantly more space. Sharing the base image is still supported.

- While squashing layers may produce smaller images, it may have a negative impact on performance, as a single layer takes longer to extract, and you can't parallelize downloading a single layer.

# Resources

- Udemy course - https://www.udemy.com/course/containers-under-the-hood/
- https://github.com/wagoodman/dive
- https://github.com/goldmann/docker-squash
- Dockerfile Best Practices - https://www.youtube.com/watch?v=JofsaZ3H1qM&t=1945s

# About Questodev

mindsdb Product Solutions Resources Company Stars 21k Join our Slack LOG IN GET STARTED

## Classify and label text in your database with Hugging Face and MindsDB integration

Martyna Slawinska · Dec 12, 2022 · Launch on Questodev