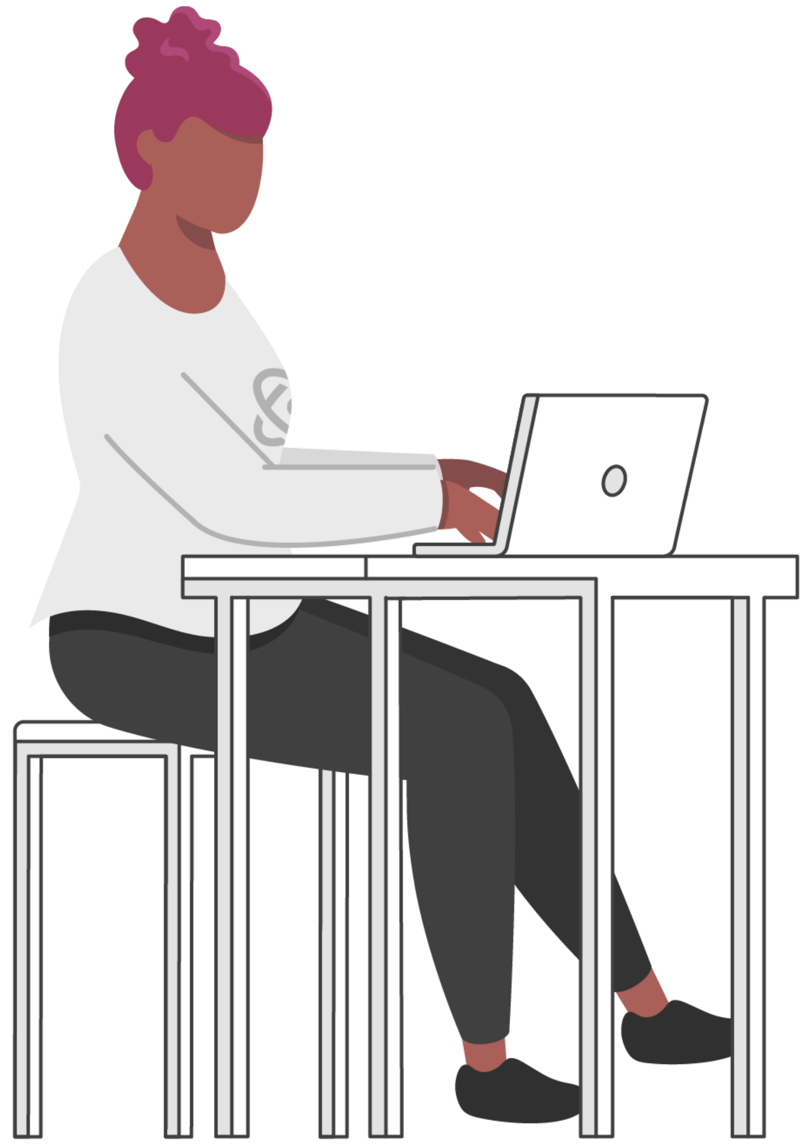
Best Practices for Optimizing Docker Images

Nigel Brown

@n_brownuk www.windsock.io



Mia is investigating the adoption of Docker

- Adept at using Dockerfiles for encapsulating apps
- Knows how to use containers to develop with different languages
- What are the best practices for authoring Dockerfiles?

Mia wants to stand on the shoulders of giants!

Module Outline

Coming up:

- Relationship between image layers and image size
- Dockerfile instruction sequencing for an efficient workflow
- Multi-stage Dockerfiles for optimizing image size
- Putting it all together

Anatomy of an Image

Understanding how Docker images are constructed is key to managing their size.

```
{
    “Config”: {
        .
        .

        “workingDir”: “/app
    }

    .
    .

    "RootFS": {
        "Type": "layers",
        "Layers": [
            "sha256:5a8512b2 ....",
            "sha256:de6a6a91 ....",
            “sha256:2470436b ....“
        ]
    }
}
```

◀ Image configuration object

◀ Working directory

◀ Filesystem definition for derived containers

◀ Content layers that make up the filesystem

Dockerfile Instruction Types

An image build processes a sequence of Dockerfile instructions

Instructions

Dockerfile instructions define the content and nature of images

Metadata

Instructions that define how derived containers will get executed

Content

Instructions that create files and directories for the image

Content Creating Dockerfile Instructions

COPY Instruction

Used to copy content from the build context into the image

ADD Instruction

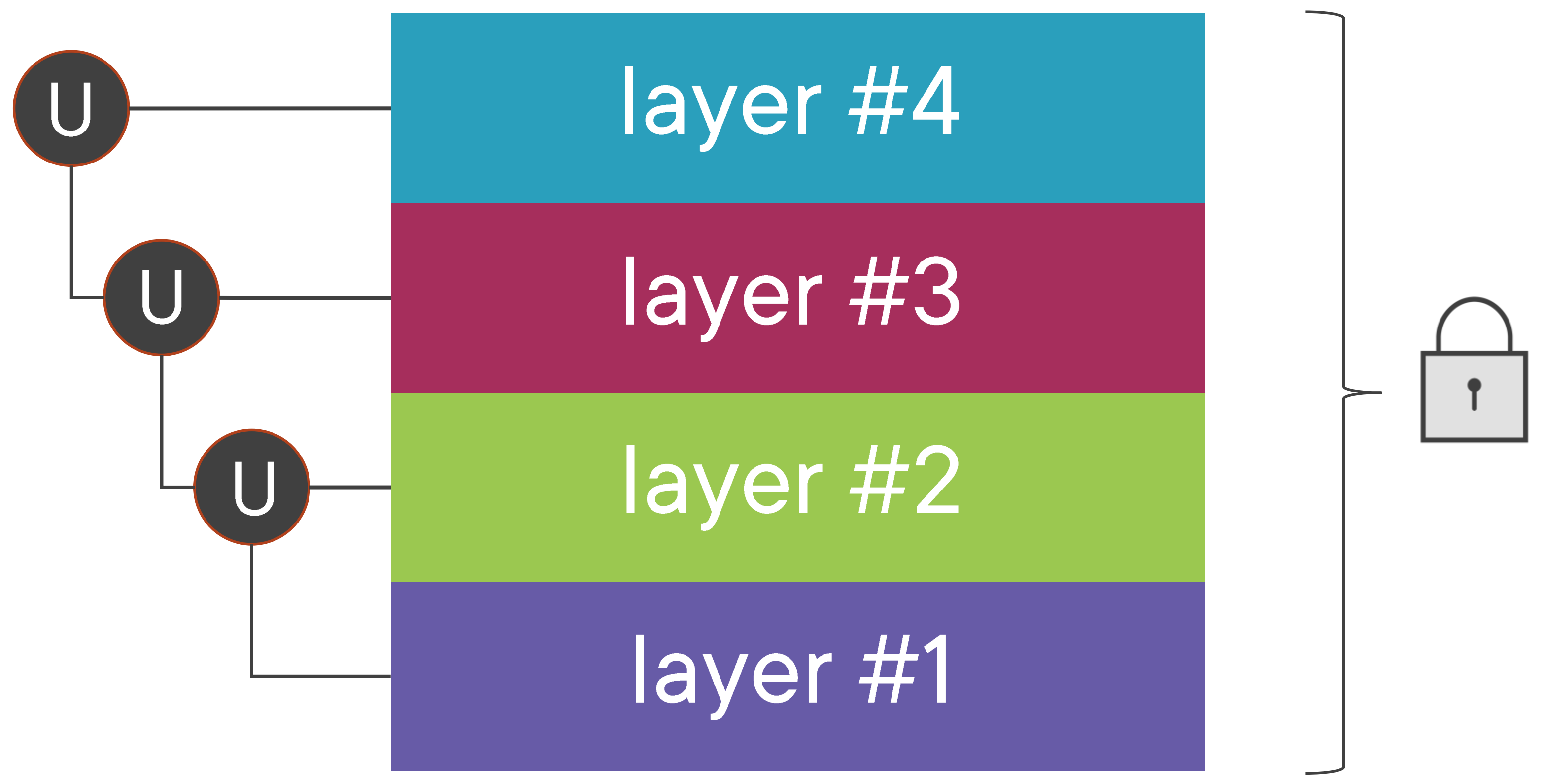
Like COPY instruction but can retrieve remote content

RUN Instruction

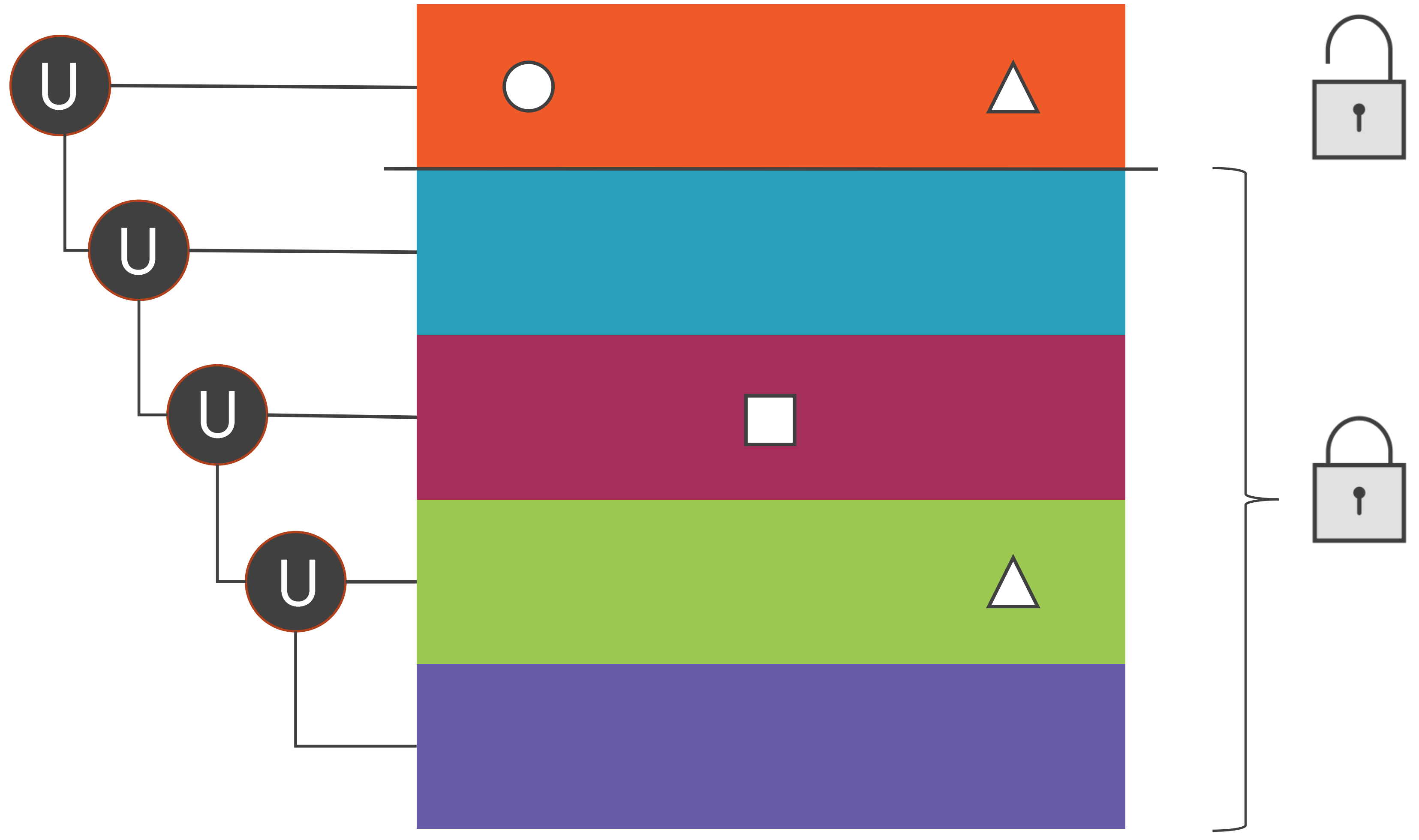
Executes commands to generate additional image content

Processing a COPY, ADD or RUN instruction adds a new content layer to the image.

Image Layers



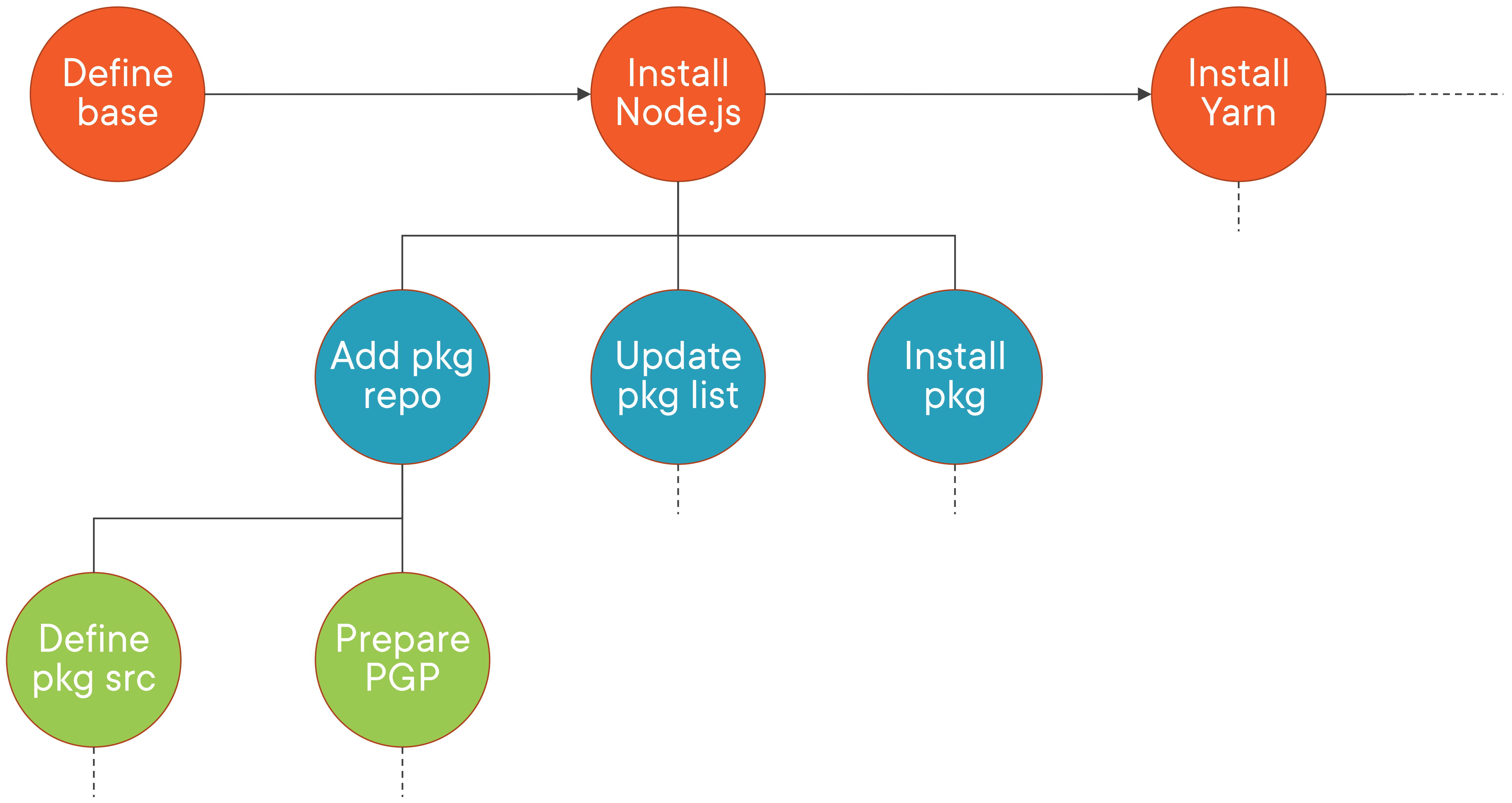
Container Filesystem



Dockerfile Scenario

The goal for our scenario is to install the Node.js runtime and the Yarn package manager into our image.

Build Steps


Define base
Install Node.js
Install Yarn
Add pkg repo
Update pkg list
Install pkg
Define pkg src
Prepare PGP

Side Effects

Dockerfile

```
FROM debian:buster

RUN apt-get update

RUN apt-get install -y  \
        curl            \
        ca-certificates \
        gnupg

<snip>

RUN apt-get update

RUN apt-get install -y  \
        nodejs          \
        yarn

<snip>
```

Temporary Content

The additional content is required temporarily

It makes the image larger than it needs to be

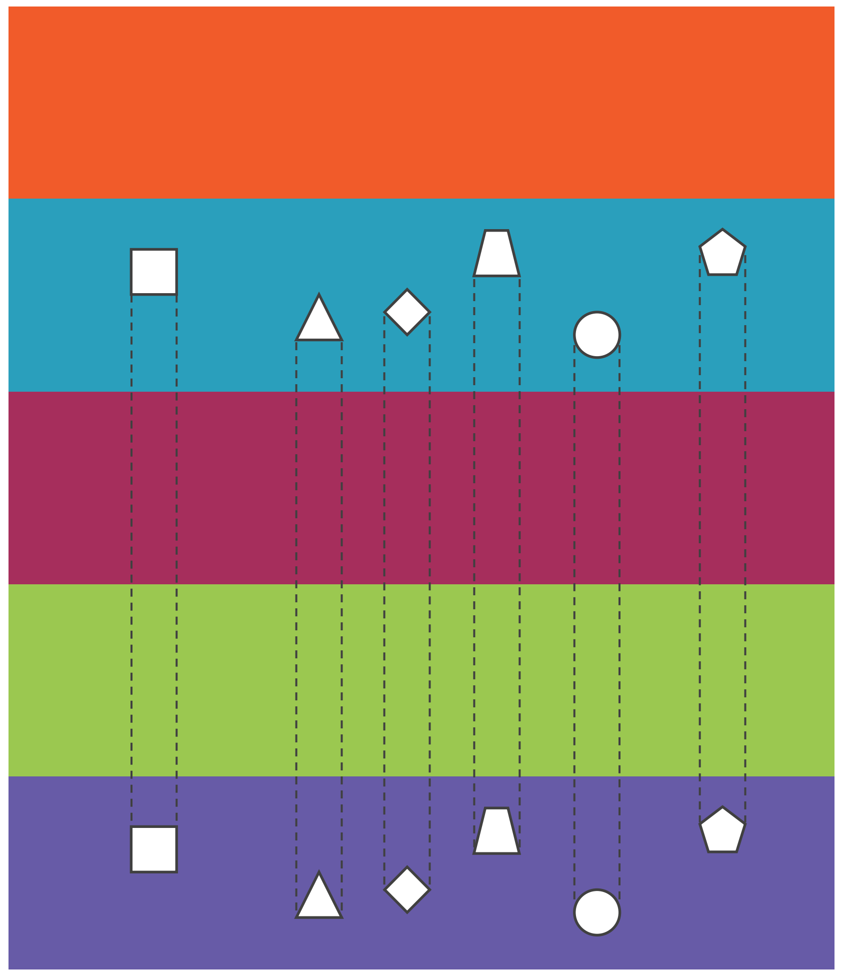
The content needs to be removed after use

Content Removal

Dockerfile

```
FROM debian:buster

RUN apt-get update

RUN apt-get install -y   \
        curl             \
        ca-certificates \
        gnupg

<snip>

RUN apt-get update

RUN apt-get install -y   \
        nodejs           \
        yarn

RUN apt-get purge -y curl ca-certificates gnupg

<snip>
```

Hidden Content in Layers



RUN instruction to remove content

RUN instruction to add content

Logical AND Operator Use

Dockerfile

```
FROM debian:buster

RUN apt-get update    && \

RUN apt-get install -y  \
        curl            \
        ca-certificates \
        gnupg         && \

<snip>                && \

RUN apt-get update    && \

RUN apt-get install -y  \
        nodejs          \
        yarn          && \

RUN apt-get purge -y curl gnupg

<snip>
```

Build Cache

Docker uses a local cache of image build steps. Careful placement of Dockerfile instructions can maximize cache hits.

Image Chain

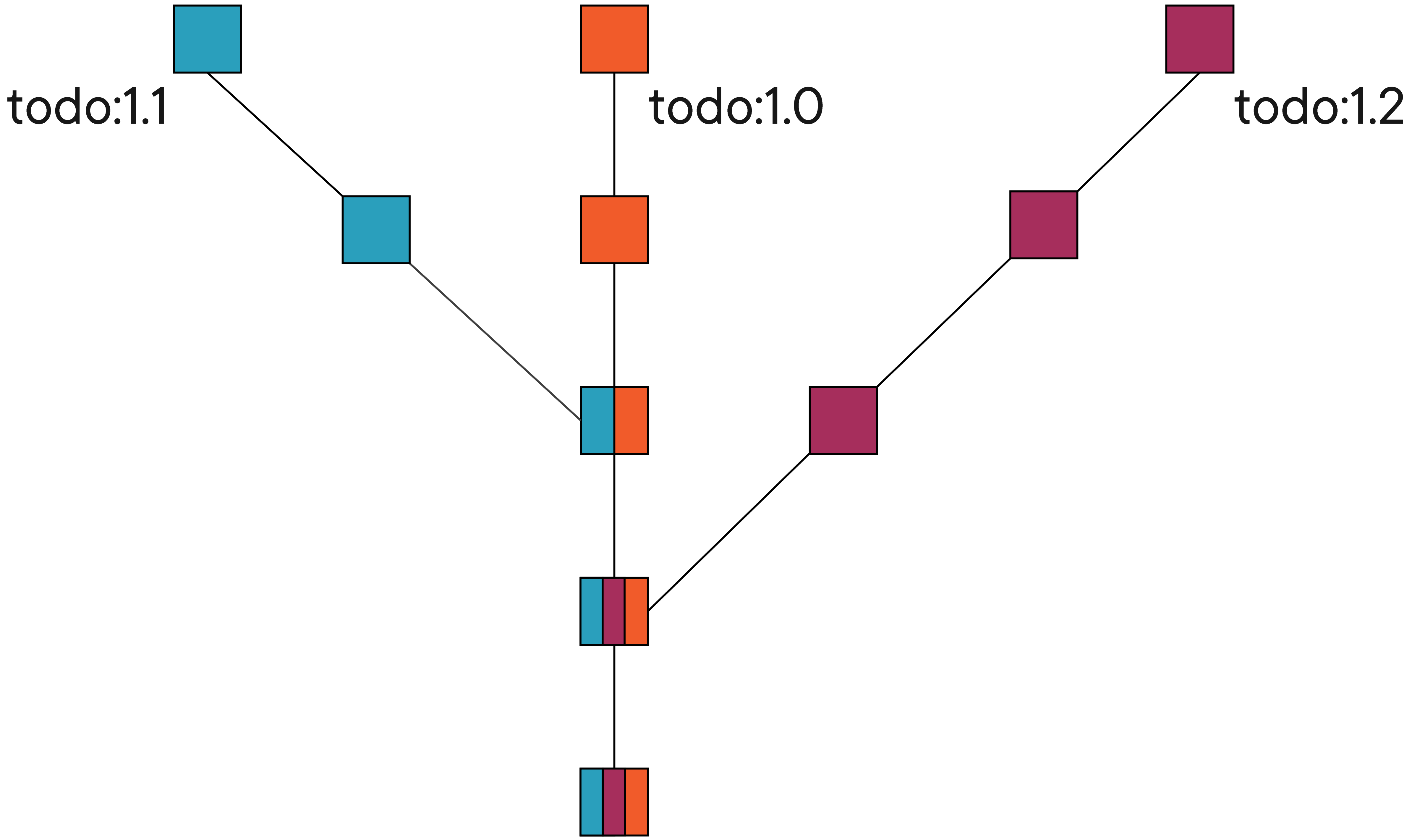
Each Dockerfile instruction processed during a build results in the creation of an intermediary image that is part of the build cache

These images are created by ‘committing’ containers created from the image associated with the preceding Dockerfile instruction

Images reference their parent image and thereby create an implicit chain of images that represent a sequence of instructions

Docker will pass over instructions that form a sequence that already exists in the build cache.

Using the Build Cache



Hitting and Missing

Instruction change
Adding, removing or altering an instruction invalidates the cache

Checksum check
Content change in build context will invalidate the cache

Command output
Consequences of command execution are not checked

Sequencing Dockerfile Instructions

Before

```
<snip>

WORKDIR /app

COPY . .

RUN yarn install

<snip>
```

After

```
<snip>

WORKDIR /app

COPY package.json yarn.lock ./

RUN yarn install

COPY spec src ./

<snip>
```

The Practicalities

Analyze the dependencies between Dockerfile instructions to determine ordering constraints

Order Dockerfile instructions according to the frequency of change; less frequent first, more frequent last

Where it's beneficial, split COPY Dockerfile instructions that copy content from the build context

Multi-stage Dockerfiles can play a bit part in optimizing the size of images.

Dockerfile

```
FROM debian:buster

RUN apt-get update                                                          && \
    apt-get install -y --no-install-recommends curl ca-certificates gnupg && \
    curl -s https://deb.nodesource.com/gpgkey/nodesource.gpg.key |            \
        apt-key add -                                                       && \
    echo 'deb https://deb.nodesource.com/node_14.x buster main' |             \
        tee /etc/apt/sources.list.d/nodesource.list                         && \
    curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | apt-key add -      && \
    echo "deb https://dl.yarnpkg.com/debian/ stable main" |                   \
        tee /etc/apt/sources.list.d/yarn.list                               && \
    apt-get update                                                          && \
    apt-get install -y --no-install-recommends nodejs yarn                  && \
    apt-get purge -y curl gnupg                                             && \
    rm -rf /var/lib/apt/lists/*
```

Dockerfile

```
FROM debian:buster

RUN apt-get update                                                          && \
    apt-get install -y --no-install-recommends curl ca-certificates gnupg && \
    curl -s https://deb.nodesource.com/gpgkey/nodesource.gpg.key |            \
        apt-key add -                                                       && \
    echo 'deb https://deb.nodesource.com/node_15.x buster main' |             \
        tee /etc/apt/sources.list.d/nodesource.list                         && \
    curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | apt-key add -      && \
    echo "deb https://dl.yarnpkg.com/debian/ stable main" |                   \
        tee /etc/apt/sources.list.d/yarn.list                               && \
    apt-get update                                                          && \
    apt-get install -y --no-install-recommends nodejs yarn                  && \
    apt-get purge -y curl gnupg                                             && \
    rm -rf /var/lib/apt/lists/*
```

Dockerfile

```
FROM debian:buster AS base

RUN apt-get update
RUN apt-get install -y --no-install-recommends curl ca-certificates gnupg
RUN curl -s https://deb.nodesource.com/gpgkey/nodesource.gpg.key |           \
        apt-key add -
RUN echo 'deb https://deb.nodesource.com/node_14.x buster main' |            \
        tee /etc/apt/sources.list.d/nodesource.list
RUN curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | apt-key add -
RUN echo "deb https://dl.yarnpkg.com/debian/ stable main" |                  \
        tee /etc/apt/sources.list.d/yarn.list
RUN apt-get update
RUN apt-get install -y --no-install-recommends nodejs yarn
RUN apt-get purge -y curl gnupg
RUN rm -rf /var/lib/apt/lists/*
```

Profiting from Multi-stage Dockerfiles

Return to a RUN instruction for each command

Maximizes the use of the build cache

Temporary content resides in a previous stage

Choice is a trade-off

- **Image size vs build speed**
- **Be wary when sharing the build cache**

Demo

Creating an optimal image build for an application

- Start with a sub-optimal Dockerfile
- Minimize content using a single layer
- See the effect of careful Dockerfile instruction sequencing
- Optimize image size using multiple stages

Up Next:

Making Configuration Data Available to Containerized Applications

Module Summary

What we covered:

- Relationship between layers and size
- Concatenating commands using the AND operator
- Docker instruction sequencing
- Enhanced builds with multi-stage Dockerfiles