NVISIA

Dockerfile Best Practices

September 17th, 2019

What is a Dockerfile?

Example Dockerfile

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

What does each step do?

- **FROM** creates a layer from the ubuntu:18.04 Docker image.
- **COPY** adds files from your Docker image's current directory.
- **RUN** builds your application with make.
- **CMD** specifies what command to run within the container

Build Context

Simple Build Context Example

Dockerfile

```
FROM alpine:3.10.1
COPY example.txt .
CMD cat example.txt
```

Current Directory

Dockerfile	59 bytes
example.txt	19 bytes
README.md	54 bytes

Build Command

```
docker build –t context-1 .
```

More Complex Build Context Example

Dockerfile

```
FROM alpine:3.10.1
COPY example.txt .
CMD cat example.txt
```

Build Command

```
docker build \
  -t context-2 \
  -f docker/Dockerfile
  data-1
```

Current Directory

.git	28,995 bytes
.gitignore	7 bytes
data-1	
--> example.txt	19 bytes
data-2	
--> additional.txt	26 bytes
data-3	
--> retropie-4.5 rpi2_rpi3.img.gz	813,960,280 bytes
docker	
--> Dockerfile	59 bytes
README.md	79 bytes

More Complex Build Context Example

Dockerfile

```
FROM alpine:3.10.1
COPY data-1/example.txt .
COPY data-2/additional.txt .
CMD cat example.txt additional.txt
```

Build Command

```
docker build \
  -t context-3 \
  -f docker/Dockerfile
  .
```

Current Directory

.git	28,995 bytes
.gitignore	7 bytes
data-1	
--> example.txt	19 bytes
data-2	
--> additional.txt	26 bytes
data-3	
--> retropie-4.5 rpi2_rpi3.img.gz	813,960,280 bytes
docker	
--> Dockerfile	59 bytes
README.md	74 bytes

More Complex Build Context Example

Dockerfile

```
FROM alpine:3.10.1
COPY data-1/example.txt .
COPY data-2/additional.txt .
CMD cat example.txt additional.txt
```

Build Command

```
docker build \
  -t context-4 \
  -f docker/Dockerfile
  .
```

Current Directory

.git	28,995 bytes
.gitignore	7 bytes
data-1	
--> example.txt	19 bytes
data-2	
--> additional.txt	26 bytes
data-3	
--> retropie-4.5 rpi2_rpi3.img.gz	813,960,280 bytes
docker	
--> Dockerfile	59 bytes
.dockerignore	13 bytes
README.md	74 bytes

Build Context

- Context is the given directory and down
  - Can be a url
  - Cannot COPY files from outside of the context
- Typically it's going to be the current directory, but it can be modified
- Does not have to be the same directory that has the Dockerfile
  - Use the -f option to specify the Dockerfile if it has a different name or is in a different directory
- Use a .dockerignore file to skip unneeded files from getting added to the context

Image Caching

Dockerfile Project

- Simple Spring Boot project
  - Follows the Maven project structure
- Built with Gradle & Docker
  - Takes roughly 110 seconds (150 seconds the first time) to build the jar

```
docker run \
  --rm \
  -w /home/gradle/project \
  -v $(pwd):/home/gradle/project \
  gradle:4.9.0-jdk8-alpine \
  gradle clean build
```

- `.dockerignore` file ignores (.git, .gradle, bin)
- Needs a runtime image
  - Preferably, something that can be built during CI

Dockerfile Caching Example

```
FROM ubuntu
COPY . /app
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk
RUN apt-get -y install ssh
RUN apt-get -y install vim
CMD ["java", "-jar", "/app/build/libs/docker-app-ds.jar"]
```

Copying in every file.
Any change breaks the cache.

Build Stats	
Build Time (Initial)	~ 180 Seconds
Build Time (Any file change)	~ 150 Seconds
Image Size	597 MB

Dockerfile Caching Example

Only copy in the jar. Setting Invalidated on jar change.

```
FROM ubuntu
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk
RUN apt-get -y install ssh
RUN apt-get -y install vim
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Build Stats	
Build Time (Initial)	~ 180 Seconds
Build Time (Any change)	~ 150 Seconds
Image Size	598 MB

Dockerfile Caching Example

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk
RUN apt-get -y install ssh
RUN apt-get -y install vim
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

The update gets cached in a separate layer.

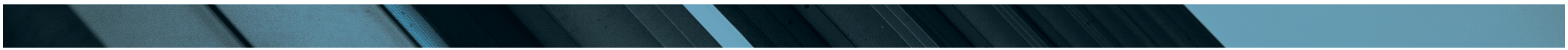
Moved to right before it's needed.

Build Stats	
Build Time (Initial)	~ 180 Seconds
Build Time (JAR change)	~ 150 Seconds
Image Size	596 MB

Dockerfile Caching Example

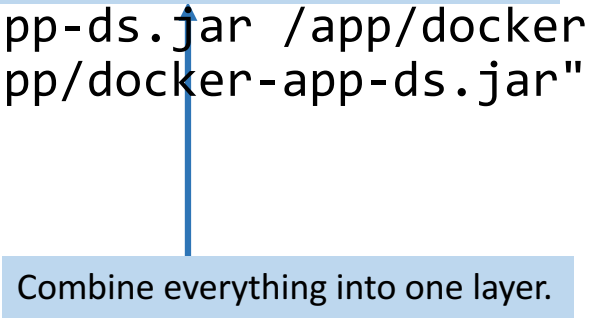
```
FROM ubuntu
RUN apt-get update && apt-get -y install openjdk-8-jdk
RUN apt-get update && apt-get -y install ssh
RUN apt-get update && apt-get -y install vim
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Every line gets its own update.



Dockerfile Caching Example

```
FROM ubuntu
RUN apt-get update && \
    apt-get -y install openjdk-8-jdk ssh vim
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```



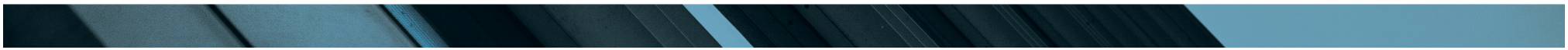
Caching

- Changing the file itself will invalidate that layer and down
  - Old versions of images may still exist in the cache and can be reused
- Only Copy what's needed
  - More focused copies limit the chances of the cache being invalidated
  - Also cuts down on the size of the image
- The order of execution matters
  - Frequently changing items are better off at the bottom of the files
  - Copy files in right before they are needed
- Bundle commands to prevent using old files

Reducing Size

Reducing Size

- Dump unnecessary files & dependencies
  - Don’t copy/add files that aren’t needed
  - .dockerignore can also help remove unnecessary files
  - Remove development dependencies from non-development images
  - Can also reduce security risk
- Cleanup after installations
  - Remove zip files after unzipping
  - Cleanup install/temp files
  - Remove applications only needed for installation (e.g. curl)



Dockerfile Sizing Example

Not needed for production.

```
FROM ubuntu
RUN apt-get update && \
    apt-get -y install openjdk-8-jdk ssh vim
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Build Stats	
Build Time (Initial)	~ 160 Seconds
Build Time (JAR change)	~ 5 Seconds
Image Size	596 MB

Dockerfile Sizing Example

```
FROM ubuntu
RUN apt-get update && \
    apt-get -y install openjdk-8-jdk
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Build Stats	
Build Time (Initial)	~ 120 Seconds
Build Time (JAR change)	~ 5 Seconds
Image Size	498 MB

Dockerfile Sizing Example

```
FROM ubuntu
RUN apt-get update && \
    apt-get -y install --no-install-recommends openjdk-8-jdk
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Only install required components.

Build Stats	
Build Time (Initial)	~ 100 Seconds
Build Time (JAR change)	~ 5 Seconds
Image Size	498 MB

Dockerfile Sizing Example

```
FROM ubuntu
RUN apt-get update && \
    apt-get -y install --no-install-recommends \
    openjdk-8-jdk && \
    rm -rf /var/lib/apt/lists/*
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Remove temp install files.

Build Stats	
Build Time (Initial)	~ 105 Seconds
Build Time (JAR change)	~ 5 Seconds
Image Size	447 MB

Parent Images

Parent Image

- Use Trusted Images
  - https://arstechnica.com/information-technology/2018/06/backdoored-images-downloaded-5-million-times-finally-removed-from-docker-hub/
- Don’t reinvent the wheel
- Make sure you select a version
  - ‘latest’ may work today, but may require changes tomorrow
- Find Smallest Image that does what you need to do
  - Quicker Downloads
  - Less Risks

Dockerfile Parent Example

```
FROM ubuntu
RUN apt-get update && \
    apt-get -y install --no-install-recommends \
    openjdk-8-jdk && \
    rm -rf /var/lib/apt/lists/*
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

There is already an official image for OpenJDK.

Build Stats	
Build Time (Initial)	~ 105 Seconds
Build Time (JAR change)	~ 5 Seconds
Image Size	417 MB

Dockerfile Parent Example

```
FROM openjdk
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Using the official image.
No more install steps.

Build Stats	
Build Time (Initial)	~~100~~ Seconds
Build Time (JAR change)	~ 5 Seconds
Image Size	~~501~~ MB

Dockerfile Parent Example

```
FROM openjdk:8
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Use a more specific version.

Build Stats	
Build Time (Initial)	~ 40 Seconds
Build Time (JAR change)	~ 5 Seconds
Image Size	509 MB

Dockerfile Parent Example

```
FROM openjdk:8-jre-alpine
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Using just the JRE on a smaller base image.

Build Stats	
Build Time (Initial)	~ 20 Seconds
Build Time (JAR change)	~ 5 Seconds
Image Size	510 MB

Compiling the Code

Current State

GOOD NEWS

- We have a good image for a runtime
  - No additional dependencies
  - Builds fast
  - Small size

BAD NEWS

- We have no image for compilation
  - Building the jar is external to the image
  - Can’t utilize the Docker cache

Compiling the Code Example

```
FROM openjdk:8-jre-alpine
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

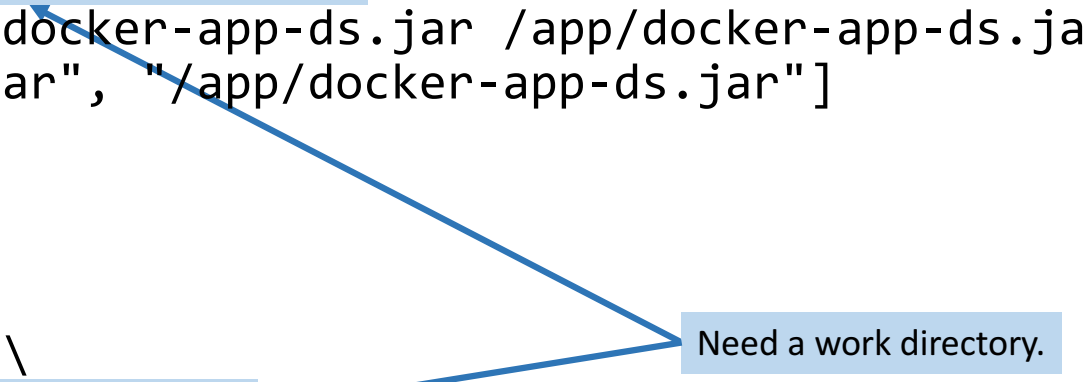
```
docker run --rm \
  -w /home/gradle/project \
  -v $(pwd):/home/gradle/project \
  gradle:4.9.0-jdk8-alpine \
  gradle clean build
```

New base image.

Compiling the Code Example

```
FROM gradle:4.9.0-jdk8-alpine
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

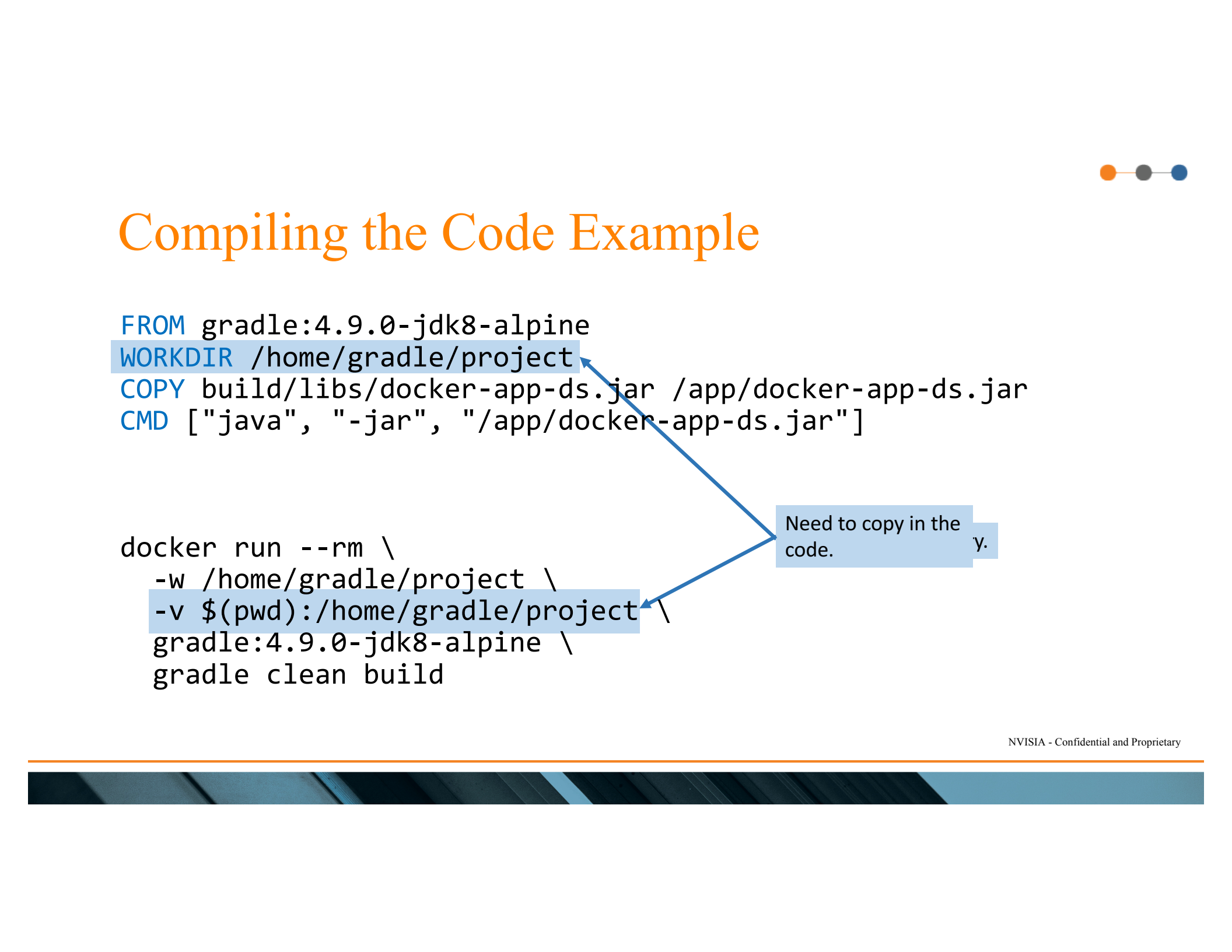
```
docker run --rm \
  -w /home/gradle/project \
  -v $(pwd):/home/gradle/project \
  gradle:4.9.0-jdk8-alpine \
  gradle clean build
```



Compiling the Code Example

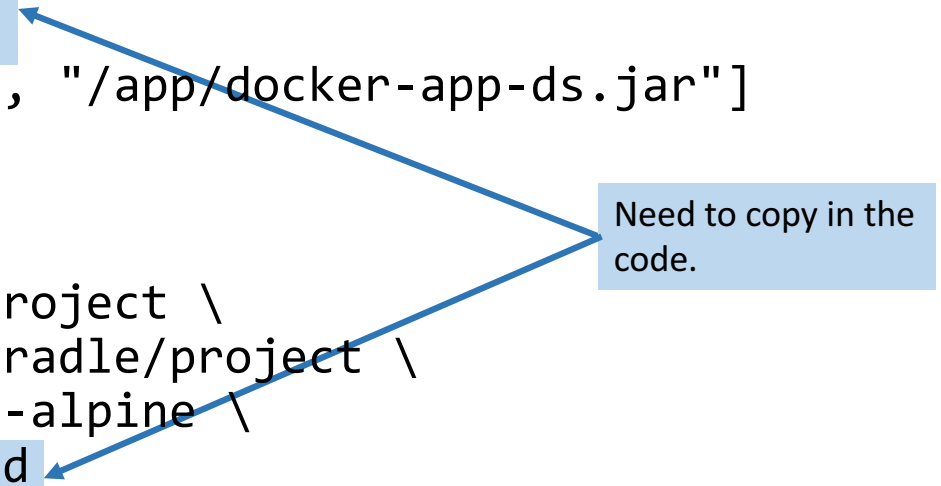
```
FROM gradle:4.9.0-jdk8-alpine
WORKDIR /home/gradle/project
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

```
docker run --rm \
  -w /home/gradle/project \
  -v $(pwd):/home/gradle/project \
  gradle:4.9.0-jdk8-alpine \
  gradle clean build
```



Compiling the Code Example

```
FROM gradle:4.9.0-jdk8-alpine
WORKDIR /home/gradle/project
COPY build.gradle .
COPY src src
CMD ["java", "-jar", "/app/docker-app-ds.jar"]


docker run --rm \
  -w /home/gradle/project \
  -v $(pwd):/home/gradle/project \
  gradle:4.9.0-jdk8-alpine \
  gradle clean build
```



Compiling the Code Example

```
FROM gradle:4.9.0-jdk8-alpine
WORKDIR /home/gradle/project
COPY build.gradle .
COPY src src
RUN gradle build --no-daemon
CMD ["java", "-jar", "build/libs/docker-app-ds.jar"]

docker run --rm \
  -w /home/gradle/project \
  -v $(pwd):/home/gradle/project \
  gradle:4.9.0-jdk8-alpine \
  gradle clean build
```

Adjust the jar location.

Run the build.

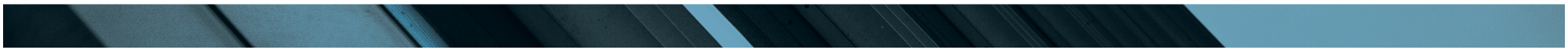
Compiling the Code Example

```
FROM gradle:4.9.0-jdk8-alpine
WORKDIR /home/gradle/project
COPY build.gradle .
COPY src src
RUN gradle build --no-daemon
CMD ["java", "-jar", "build/libs/docker-app-ds.jar"]
```

Creates the directory as root user.

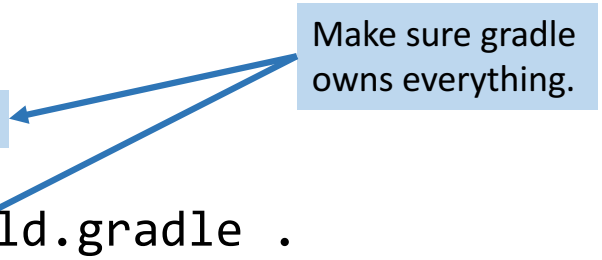
DOESN'T WORK!

Gradle throws a permission error!



Compiling the Code Example

```
FROM gradle:4.9.0-jdk8-alpine
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon
CMD ["java", "-jar", "build/libs/docker-app-ds.jar"]
```



Compiling the Code Example

```
FROM gradle:4.9.0-jdk8-alpine
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon
CMD ["java", "-jar", "build/libs/docker-app-ds.jar"]
```

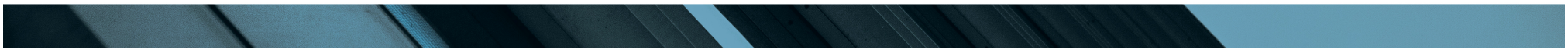
Build Stats	
Build Time (Initial)	~~105 Seconds
Build Time (Code change)	~~75 Seconds
Image Size	220 MB

Compiling the Code Example

PROBLEM: The image is bigger than it needs to be and now contains the source code and a compiler.

QUESTION: How can we add the build steps without increasing the size of the runtime?

ANSWER: Multi-Stage Build



Compiling the Code Example

```
FROM gradle:4.9.0-jdk8-alpine
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon
CMD ["java", "-jar", "build/libs/docker-app-ds.jar"]

FROM openjdk:8-jre-alpine
COPY build/libs/docker-app-ds.jar /app
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Not needed for the build.

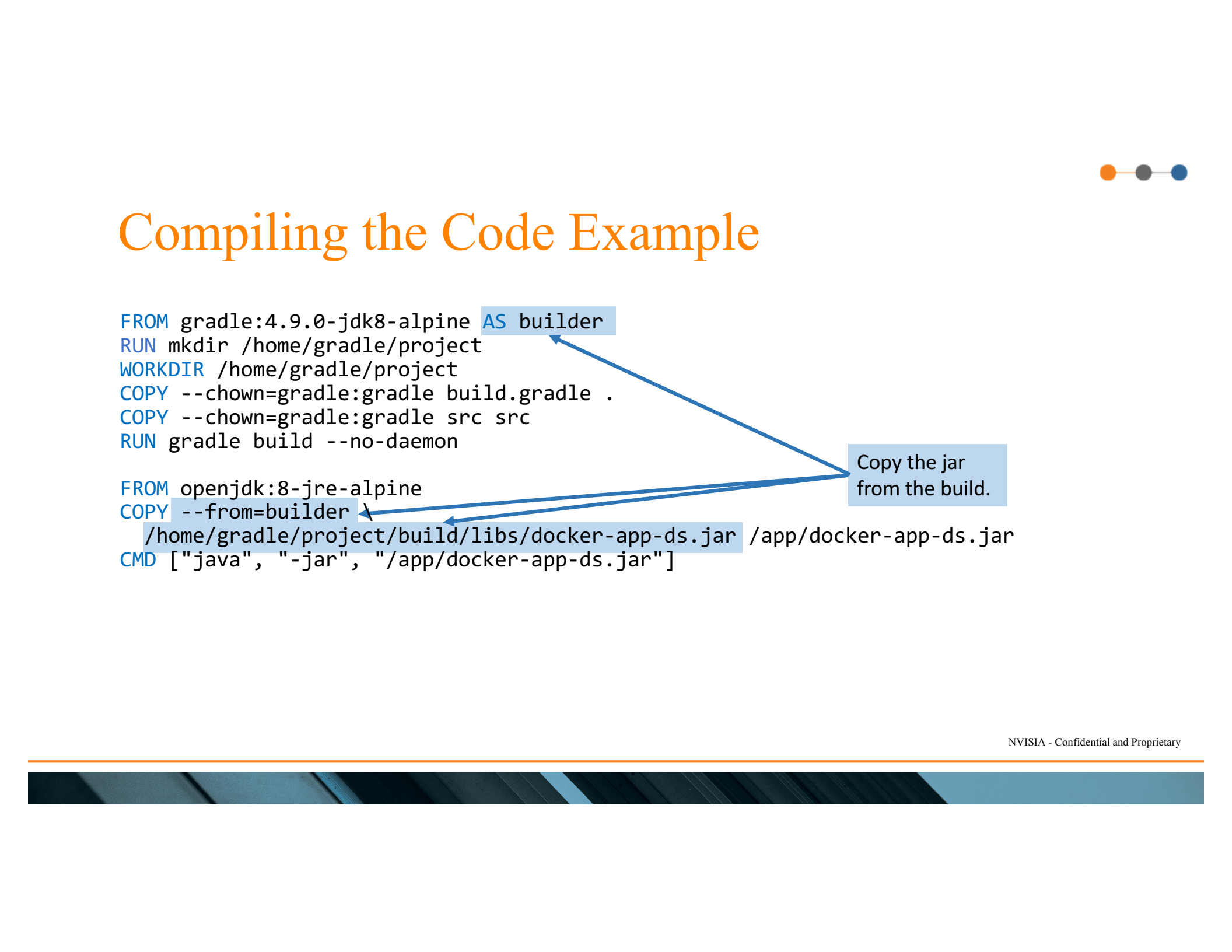
Compiling the Code Example

```
FROM gradle:4.9.0-jdk8-alpine
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon

FROM openjdk:8-jre-alpine
COPY build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

This jar no longer exists.

Compiling the Code Example

```
FROM gradle:4.9.0-jdk8-alpine AS builder
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon

FROM openjdk:8-jre-alpine
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```



Compiling the Code Example

```
FROM gradle:4.9.0-jdk8-alpine AS builder
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon

FROM openjdk:8-jre-alpine
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

NO CODE CHANGE…
NO REBUILD!

Build Stats	
Build Time (Initial)	~ 100 Seconds
Build Time (Code change)	~ 75 Seconds
Image Size	220 MB

Security

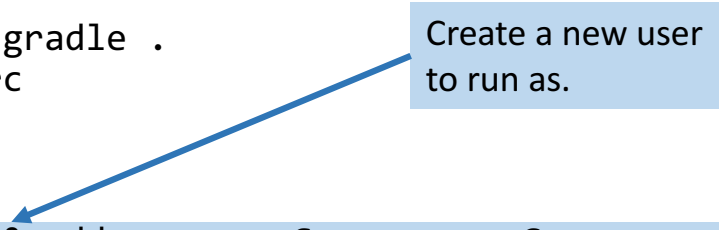
Security Example

```
FROM gradle:4.9.0-jdk8-alpine AS builder
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon

FROM openjdk:8-jre-alpine
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

OpenJDK runs as "root" by default.

Security Example

```
FROM gradle:4.9.0-jdk8-alpine AS builder
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon

FROM openjdk:8-jre-alpine
RUN addgroup -g 1001 -S appuser && adduser -u -S appuser -G appuser
USER appuser
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```



Security Considerations

- Run as the least privileged user
- Use trusted, minimal images as a base
- Patch your images for security fixes
  - Containers don’t get patched, images do
- Don’t put secure information in Dockerfiles and/or ENV variables
- Add additional security checks to your pipeline:
  - Image Scans
  - Image Signing

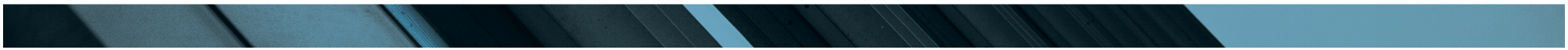
Multi-Stage Builds

Multi-Stage Builds

- We are already using a multi stage build
  - Most common use of multi-stage build
- You can have more than two stages
- Some examples:
  - Pulling artifacts from different images
  - Building different versions of your image
    - Different base image (different flavors of linux: alpine, centos, ubuntu)
    - Different images for different uses (production vs dev testing)

Multi-Stage Builds

```
FROM node:8.11.3-alpine AS fe-build
RUN apk add --no-cache --virtual .gyp python make g++
RUN npm install –g @angular/cli && npm install –g tslint && \
    npm install –g concurrently && npm install
WORKDIR /app
COPY docker-app-ui/ .
RUN npm run build:prod

FROM microsoft/dotnet:2.1-sdk-alpine AS be-build
WORKDIR /app
COPY docker-app-api/ .
RUN dotnet restore && publish –c Release –o out

FROM microsoft/dotnet:2.1-aspnetcore-runtime
WORKDIR /app
COPY --from=be-build /app/out ./
COPY --from=fe-build /app/dist wwwroot
ENTRYPOINT ["dotnet", "docker-app-api.dll"]
```

Multi-Stage Builds

```
FROM gradle:4.9.0-jdk8-alpine AS builder
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon

FROM ubuntu AS dev
RUN apt-get update && \
    apt-get -y install openjdk-8-jdk ssh vim
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]

FROM openjdk:8-jre-alpine AS prod
RUN addgroup -g 1001 -S appuser && adduser -u -S appuser -G appuser
USER appuser
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

```
docker build \
  --target prod \
  -t docker-app-api:1.0
  .
```

Determines where to stop

BuildKit

BuildKit

Docker Build enhancements for 18.09 release introduces a much-needed overhaul of the build architecture. By integrating BuildKit, users should see an improvement on performance, storage management, feature functionality, and security.

https://docs.docker.com/develop/develop-images/build_enhancements/

Enabling BuildKit

- Enable on a per build basis
  - Use an environment variable (DOCKER_BUILDKIT)

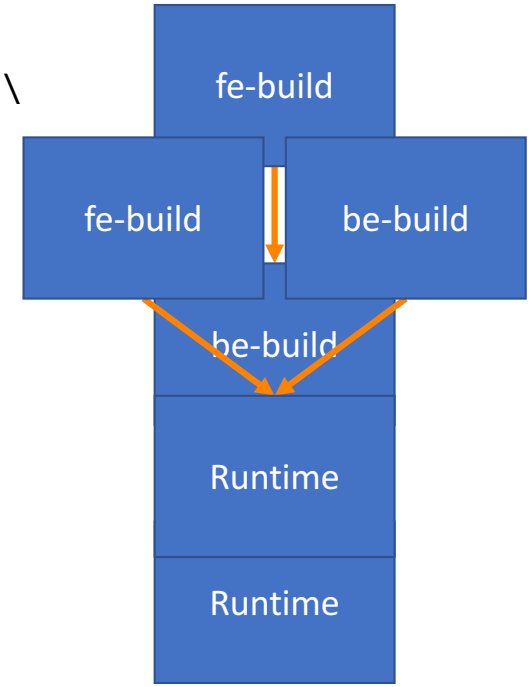
```
DOCKER_BUILDKIT=1 docker build .
```

- Enable for all builds
  - Update the daemon.json or Docker Desktop Settings

```
{ "features": { "buildkit": true } }
```

Build Concurrency

```
FROM node:8.11.3-alpine AS fe-build
RUN apk add --no-cache --virtual .gyp python make g++
RUN npm install -g @angular/cli && npm install -g tslint && \
    npm install -g concurrently && npm install
WORKDIR /app
COPY docker-app-ui/ .
RUN npm run build:prod

FROM microsoft/dotnet:2.1-sdk-alpine AS be-build
WORKDIR /app
COPY docker-app-api/ .
RUN dotnet restore && publish -c Release -o out

FROM microsoft/dotnet:2.1-aspnetcore-runtime
WORKDIR /app
COPY --from=be-build /app/out ./
COPY --from=fe-build /app/dist wwwroot
ENTRYPOINT ["dotnet", "docker-app-api.dll"]
```



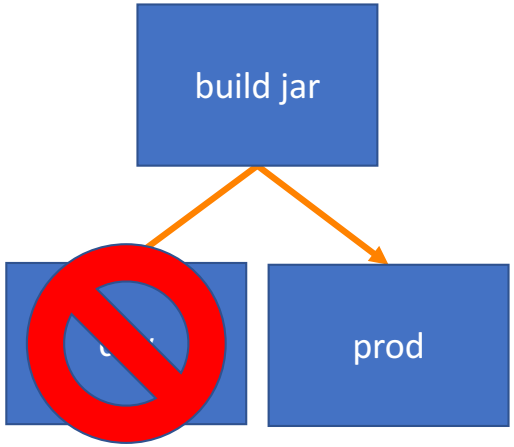
Build Concurrency

```
FROM node:8.11.3-alpine AS fe-build
RUN apk add --no-cache --virtual .gyp python make g++
RUN npm install -g @angular/cli && npm install -g tslint && \
    npm install -g concurrently && npm install
WORKDIR /app
COPY docker-app-ui/ .
RUN npm run build:prod

FROM microsoft/dotnet:2.1-sdk-alpine AS be-build
WORKDIR /app
COPY docker-app-api/ .
RUN dotnet restore && publish -c Release -o out

FROM microsoft/dotnet:2.1-aspnetcore-runtime
WORKDIR /app
COPY --from=be-build /app/out ./
COPY --from=fe-build /app/dist wwwroot
ENTRYPOINT ["dotnet", "docker-app-api.dll"]
```

Build Stats	
Build Time (Initial)	~ 400 Seconds
Build Time (Code change)	Varies
Image Size	263 MB

Build Concurrency

```
FROM gradle:4.9.0-jdk8-alpine AS builder
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon

FROM ubuntu AS dev
RUN apt-get update && \
    apt-get -y install openjdk-8-jdk ssh vim
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]

FROM openjdk:8-jre-alpine AS prod
RUN addgroup -g 1001 -S appuser && adduser -u -S appuser -G appuser
USER appuser
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```



Build Concurrency

```
FROM gradle:4.9.0-jdk8-alpine AS builder
RUN mkdir /home/gradle/project
WORKDIR /home/gradle/project
COPY --chown=gradle:gradle build.gradle .
COPY --chown=gradle:gradle src src
RUN gradle build --no-daemon

FROM ubuntu AS dev
RUN apt-get update && \
    apt-get -y install openjdk-8-jdk ssh vim
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]

FROM openjdk:8-jre-alpine AS prod
RUN addgroup -g 1001 -S appuser && adduser -u -S appuser -G appuser
USER appuser
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Build Stats	
Build Time Prod (Initial)	~ 320 Seconds
Build Time (Code change)	~ 75 Seconds
Image Size Prod	116 MB

Experimental Dockerfile Options

- To enable, start the file with this:

```
# syntax=docker/dockerfile:experimental
```

- Enables:
  - --mount syntax
  - --security syntax
  - --network syntax
- All executed via RUN commands

Security Syntax

- Allows commands to run in insecure mode
- Equivalent to `docker run --privileged`
- Options are insecure & sandbox
- Requires the security.insecure entitlement
- Executed Per `RUN` command

```
# syntax = docker/dockerfile:experimental
FROM ubuntu
RUN --security=insecure cat /proc/self/status | grep CapEff
```

Network Syntax

- Allows commands to run with different network options
- Options are none, host & default
- Requires the network.host entitlement
- Executed Per RUN command

```
# syntax = docker/dockerfile:experimental
FROM python:3.6
ADD mypackage.tgz wheels/
RUN --network=none pip install \
  --find-links wheels mypackage
```

Mount Syntaxes

Numerous mount options are available:

- **RUN `--mount=type=bind`**
  - Let's you bind directories instead of copying (still needs to be in the context)
- **RUN `--mount=type=cache`**
  - Cache directories for compilers and packages
- **RUN `--mount=type=tmpfs`**
  - Mount a temporary file system
- **RUN `--mount=type=secret`**
  - Mount a secret that can be used, but not stored in the image.
- **RUN `--mount=type=ssh`**
  - Access files via SSH (for instance, log into Git using your existing SSH key)

Cache Mount

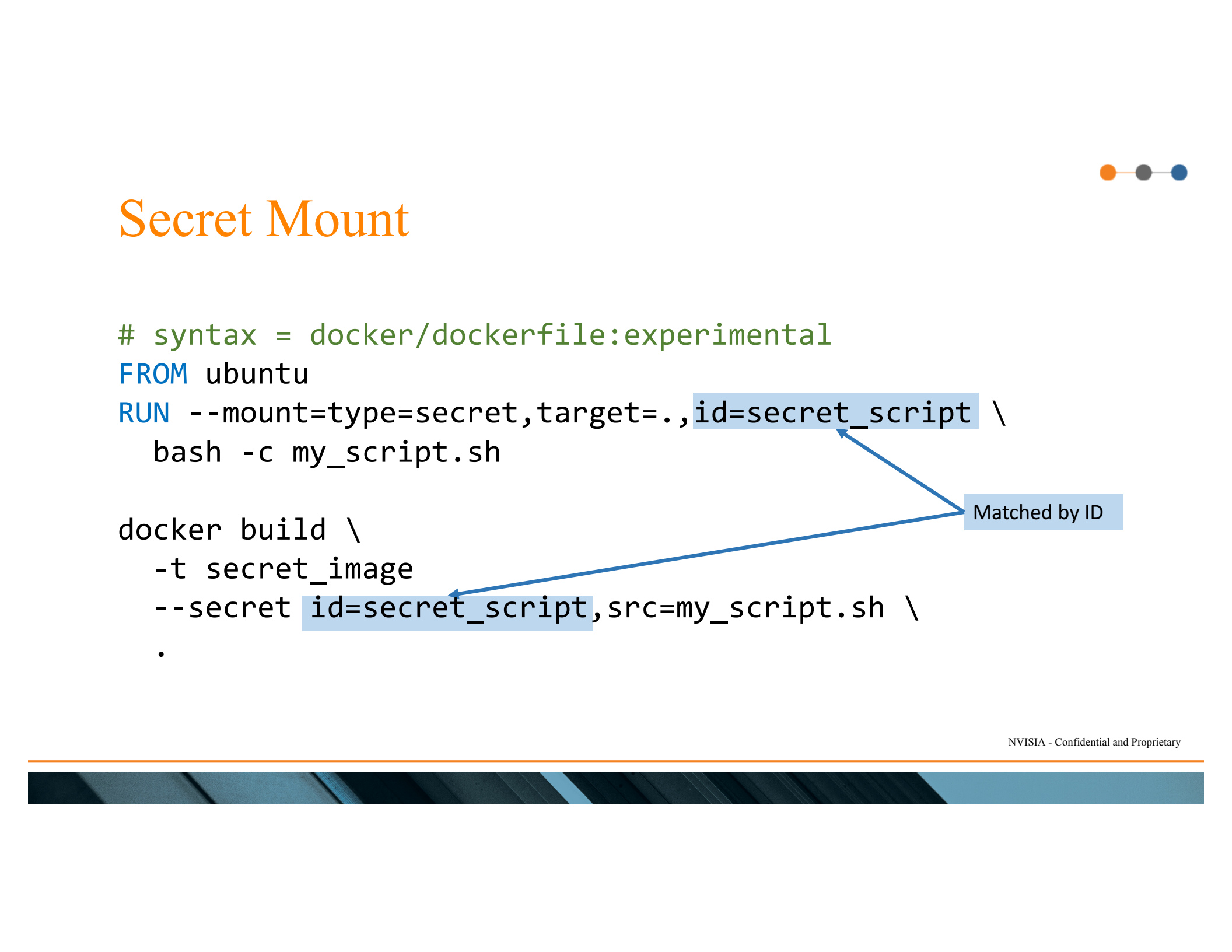
```
# syntax = docker/dockerfile:experimental
FROM gradle:4.9.0-jdk8-alpine AS builder
USER root
WORKDIR /home/gradle/project
COPY build.gradle .
COPY src src
RUN --mount=type=cache,target=/home/gradle/.gradle gradle build --no-daemon

FROM openjdk:8-jre-alpine AS prod
RUN addgroup -g 1001 -S appuser && adduser -u -S appuser -G appuser
USER appuser
COPY --from=builder \
  /home/gradle/project/build/libs/docker-app-ds.jar /app/docker-app-ds.jar
CMD ["java", "-jar", "/app/docker-app-ds.jar"]
```

Mounts as root

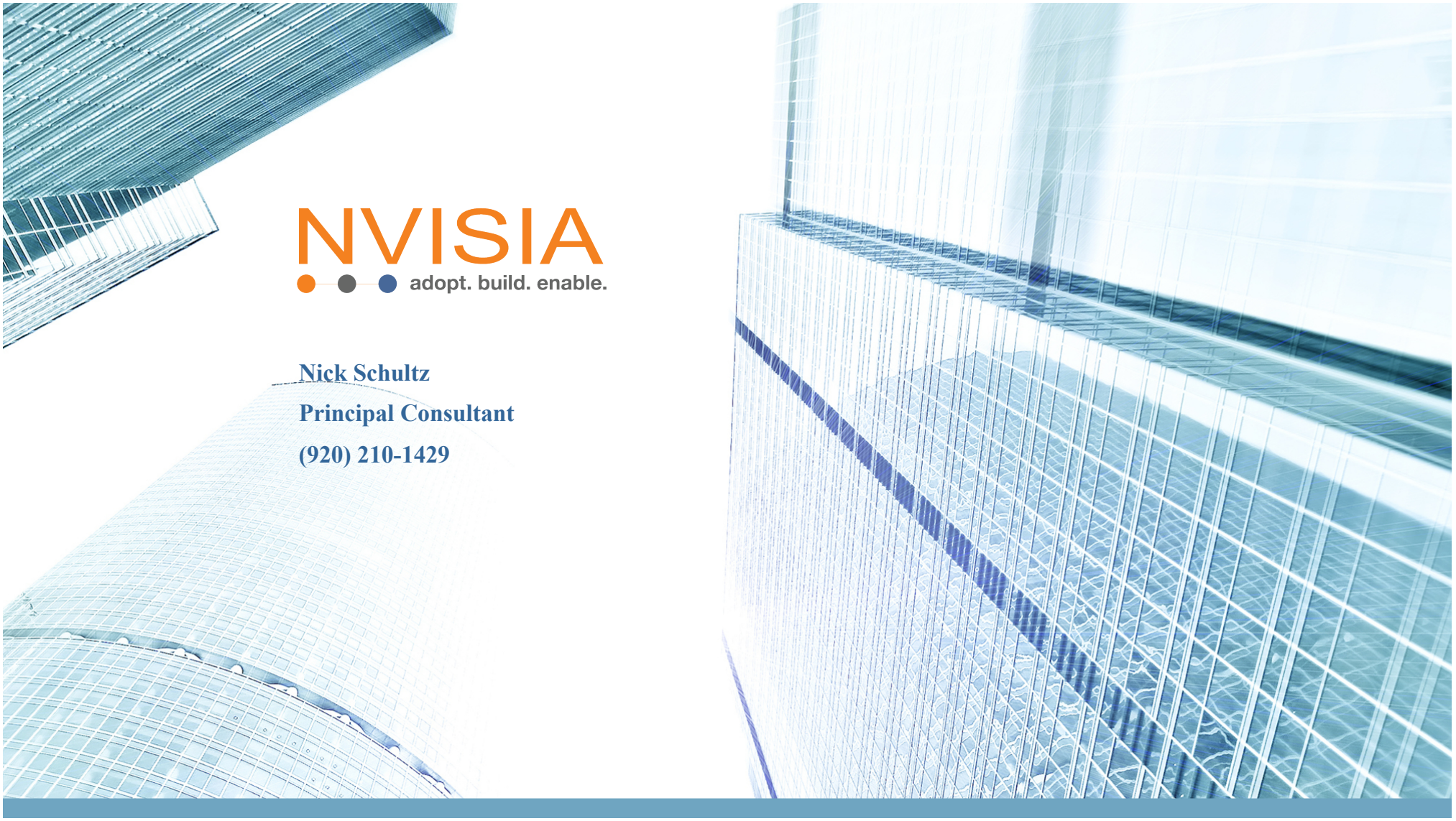
Build Stats	
Build Time Prod (Initial)	~ 120 Seconds
Build Time (Code change)	~ 30 Seconds
Image Size Prod	116 MB

Secret Mount

```
# syntax = docker/dockerfile:experimental
FROM ubuntu
RUN --mount=type=secret,target=.,id=secret_script \
  bash -c my_script.sh

docker build \
  -t secret_image
  --secret id=secret_script,src=my_script.sh \
  .
```



Questions?



NVISIA

adopt. build. enable.

Nick Schultz

Principal Consultant

(920) 210-1429