

We Help You Build Systems

That Make You

A DIGITAL LEADER

[Find out more](#)



ThoughtWorks®

©ThoughtWorks 2020 Commercial in Confidence

Layered JARs

Optimise your Docker images with Layered JARs

By Johanna Lang and Albert Attard

jlang@thoughtworks.com albert.attard@thoughtworks.com

Agenda

Docker

Layers

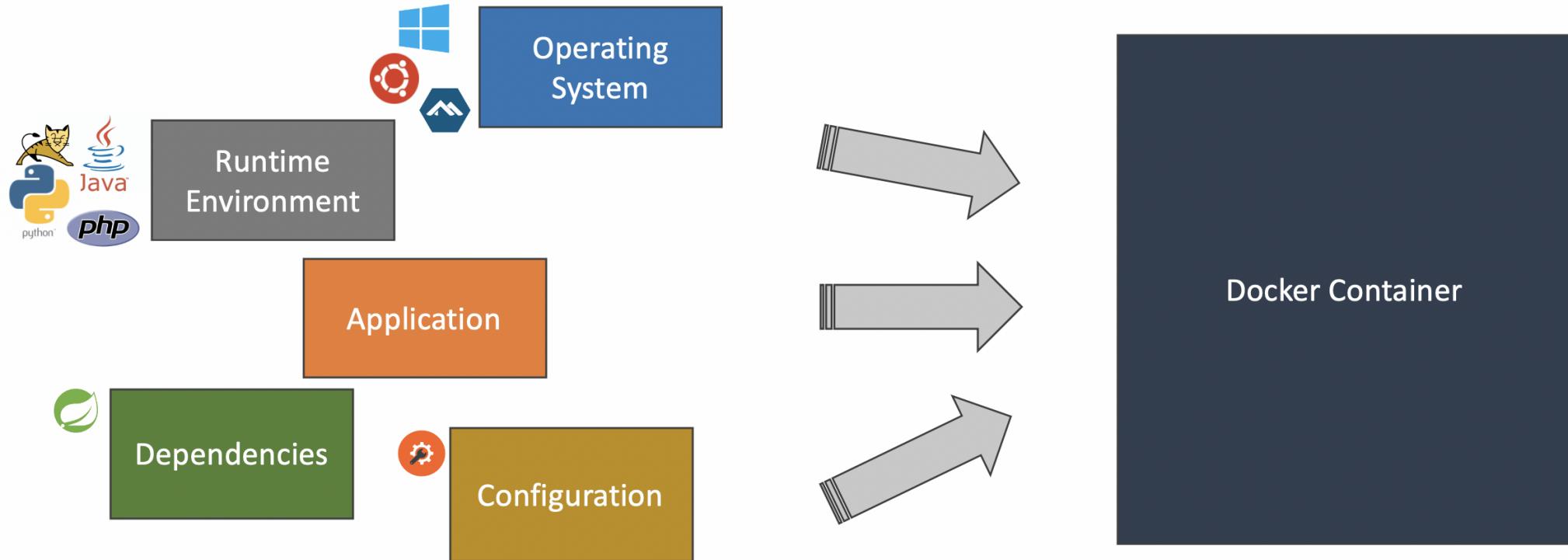
Spring Boot

Beyond Spring Boot

Docker

What is a Docker container?

A standard deployment unit that encapsulates an application and all of its dependencies



How is a Docker container created?

- By running a *Docker image*

```
$ docker run \
--rm \
--name docker-container-demo \
-p 8080:8080 \
spkane/quantum-game:latest
```

► We don't have to worry about any specific runtime environment or any particular dependency version as everything is encapsulated in the container

What is a Docker image?

- A **read-only** filesystem that contains
 - operating system
 - programs needed by the application (e.g. Java Runtime Environment)
 - application executable, its dependencies, and configuration
- **Immutable** (cannot be modified once built)
 - **New image gets created every time a new version of our application is dockerized**

Demo 2

Overview of `dive` (OS tool for exploring a docker image)

How is a Docker image created?

- By building a *Dockerfile* :

```
$ docker build . -t boot-fat-jar:local
```

The above command creates an image and tags it as `boot-fat-jar:local`

- We can run this Docker image, creating a Docker container when doing so, once this is built

```
$ docker run \  
  --rm \  
  --name boot-fat-jar-demo \  
  -p 8080:8080 \  
  boot-fat-jar:local
```

What is a Dockerfile?

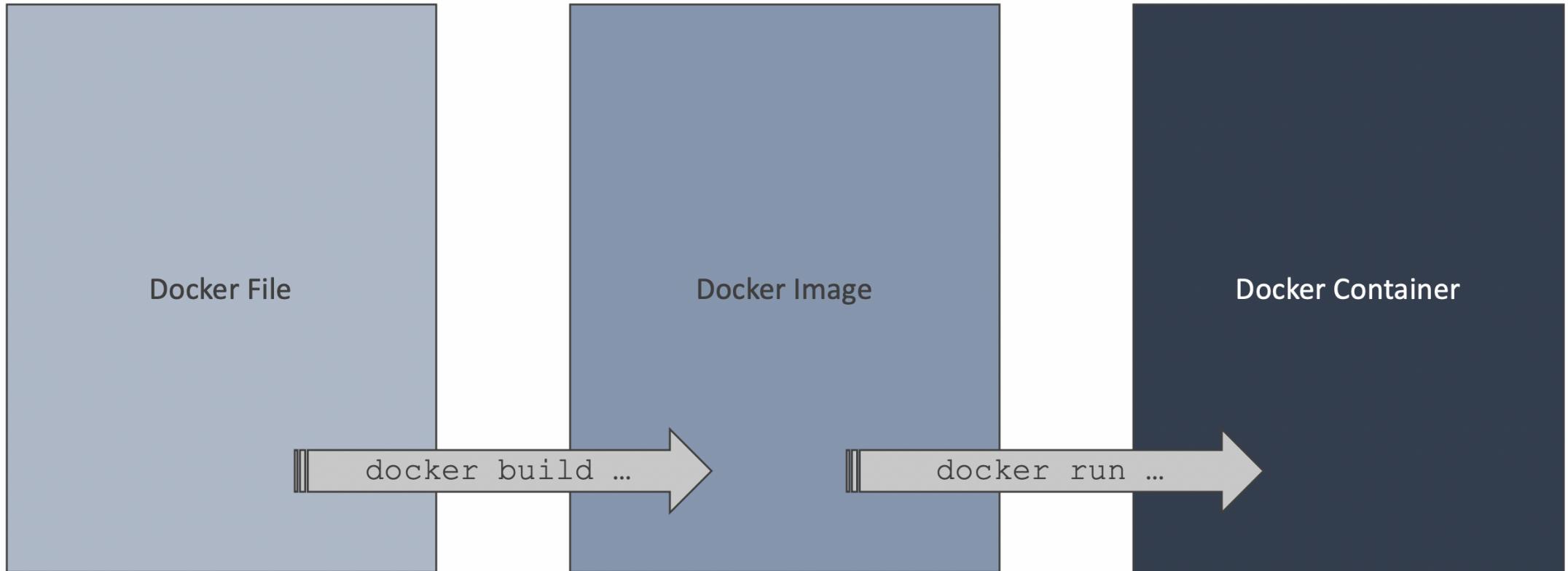
A text file, usually named `Dockerfile`, that contains a set of instructions (used to create the Docker image)

What does a Dockerfile look like?

Following is a typical *Dockerfile* that hosts a Java 8 application:

```
FROM adoptopenjdk:8u262-b10-jre-hotspot
WORKDIR /opt/app
COPY ./build/libs/*.jar application.jar
ENTRYPOINT ["java", "-jar", "application.jar"]
```

Lifecycle



Layers

What are layers?

- Remember our *Dockerfile*:

```
FROM adoptopenjdk:8u262-b10-jre-hotspot
WORKDIR /opt/app
COPY ./build/libs/*.jar application.jar
ENTRYPOINT ["java", "-jar", "application.jar"]
```

- The above Dockerfile has four layers:

- FROM ...
- WORKDIR ...
- COPY ...
- ENTRYPOINT ...

Intermediate Images

 Docker Layers

Demo

Build docker image and analyse layers with *dive*

- Build a docker image
- Discuss layers and see docker takes advantage of caching
- Analyse the docker image, using `dive`

JAR file

.jar (= java archive) is a package file format

FatJAR

- A very common way to package a JVM based application is a **FatJAR**
- A FatJAR contains
 - The application
 - All its dependencies
- it is standalone and can be executed like this:

```
$ java -jar application.jar
```

FatJAR in a Docker image

- The FatJAR is copied from our laptop into the Docker image using the `COPY` instruction:

```
COPY ./build/libs/*.jar application.jar
```

- This creates a new layer every time a new FatJAR file is created, e.g. every time the code changes
- Creating many large layers may consume large amounts of disk space

Size of FatJAR

[● Layers]			[Current Layer Contents]				
Cmp	Image ID	Size	Command	Permission	UID:GID	Size	Filetree
	sha256:b187ff70b2e47a4cf3	63 MB	#(nop) ADD file:1e8d02626176dc8141df	drwxr-xr-x	0:0	16 MB	└ opt
	sha256:5930c9e5703fbc1f5	988 kB	[-z "\$(apt-get indextargets)"]	drwxr-xr-x	0:0	16 MB	└ app
	sha256:c64c52ea2c16427957	745 B	set -xe && echo '#!/bin/sh' > /usr	-rw-r--r--	0:0	16 MB	└ application.jar
	sha256:ddc500d8499442f954	7 B	mkdir -p /run/systemd && echo 'docke				
	sha256:80b956beb7fc101928	34 MB	apt-get update && apt-get instal				
	sha256:840c5fa83b6d4aece6	108 MB	set -eux; ARCH="\$(dpkg --print-a				
	sha256:8254d5025e79d1004c	0 B	#(nop) WORKDIR /opt/app				
	sha256:a3c254cc73434bb267	16 MB	FROM sha256:a3c254cc73434bb267				

[Layer Details]

Digest: sha256:a3c254cc73434bb267828ded3b4f207903d08ad53320f81b5ab5dbe0e307
6bab

Command:

```
#(nop) COPY file:8c6c6a784fe809fcec718e69098fab4b9354ea1ba603d2b00b1eeb05d9  
cf9d5d in application.jar
```

[Image Details]

Total Image size: 222 MB
Potential wasted space: 2.7 MB
Image efficiency score: 99 %

Our FatJAR creates a layer of 16MB

Space requirements

- Consider a team working 5 days a week and committing code 20 times per day
- Each commit is followed by a push, which triggers an automated build pipeline, which builds the application and **creates a new docker image**

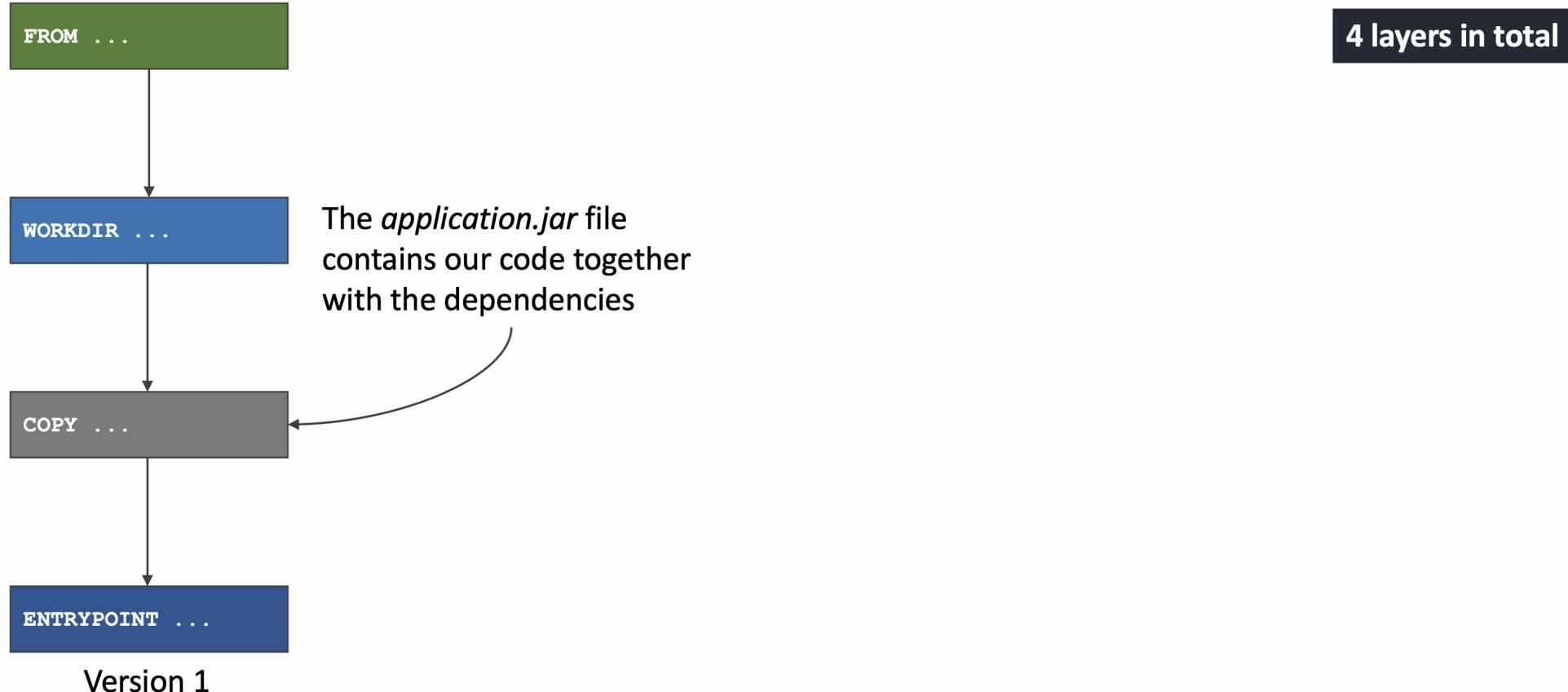


The challenge

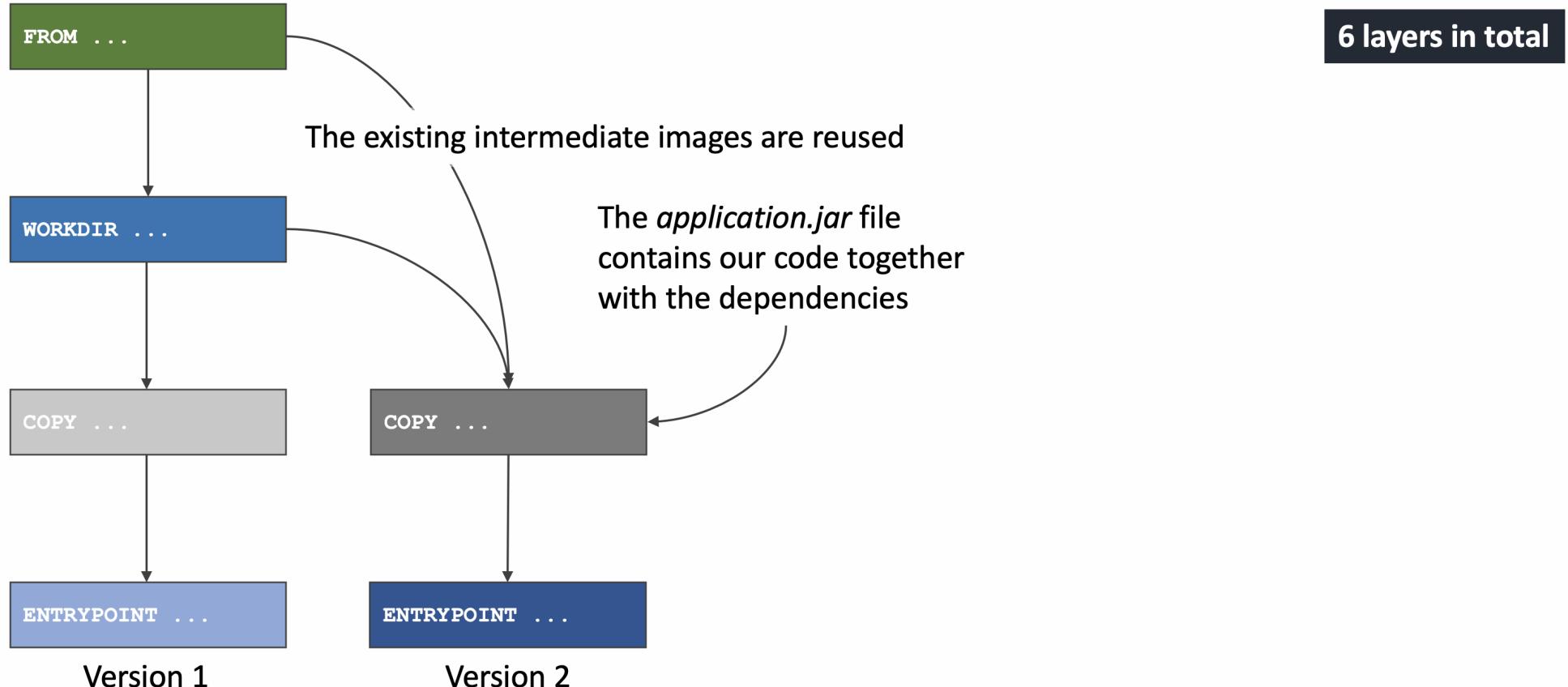
- Our application (FatJAR) contains our code **and** its dependencies
- When new features are added, the dependencies are not necessarily updated

However: each small change in the code, creates a new docker layer of about 16MB in size

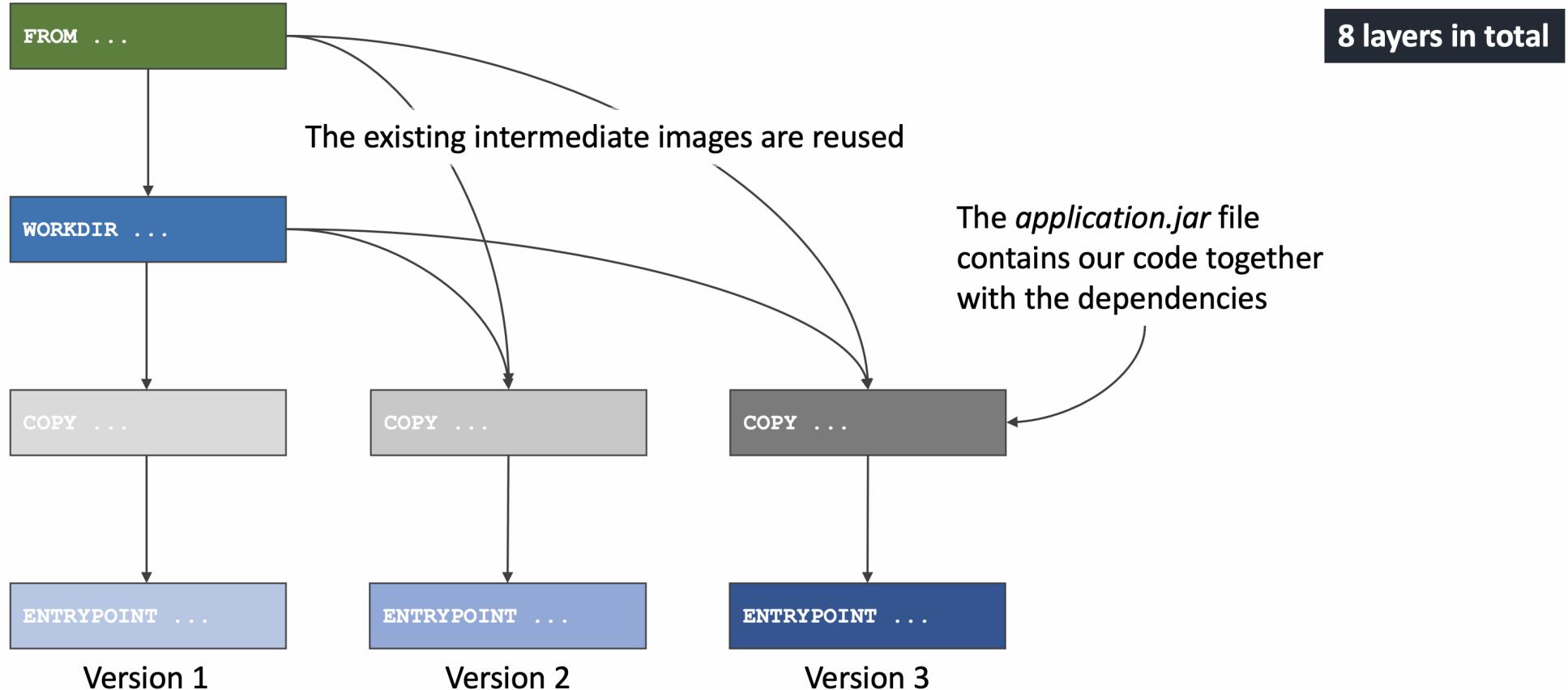
The challenge - Version 1



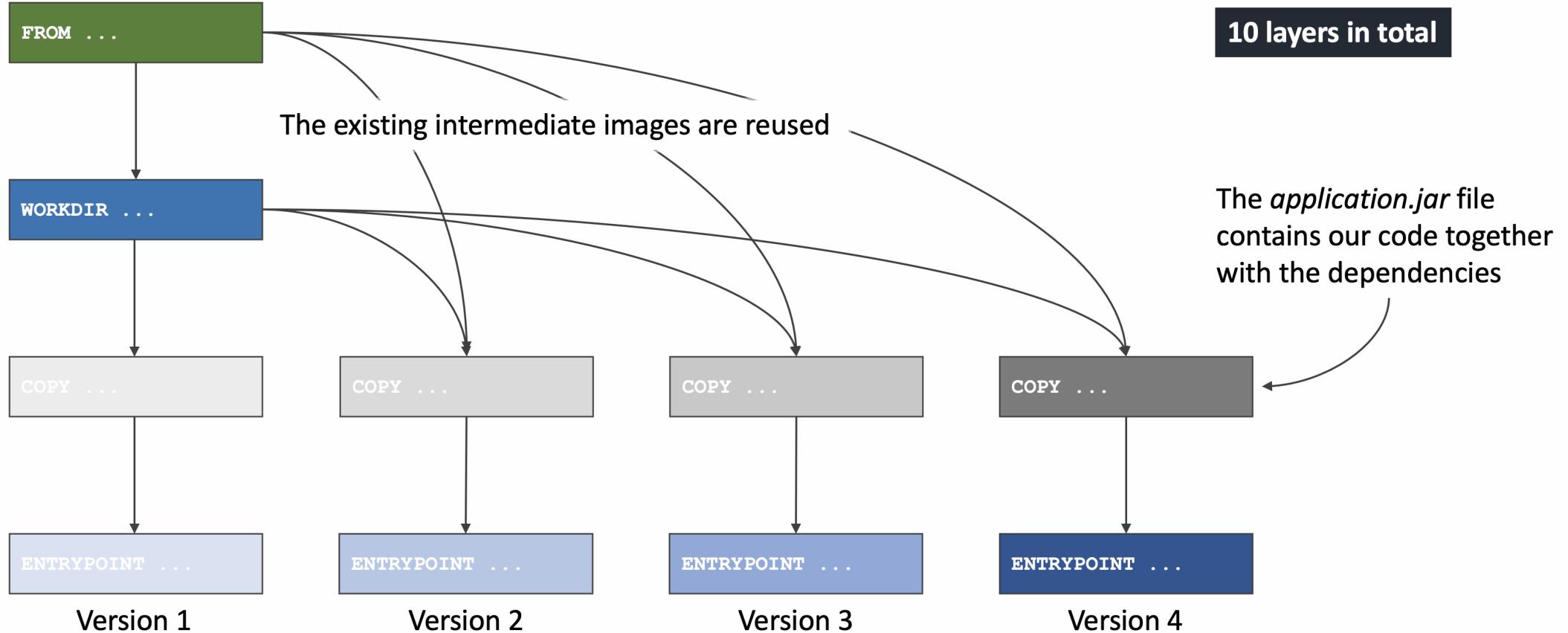
The challenge - Version 2



The challenge - Version 3



The challenge - Version 4

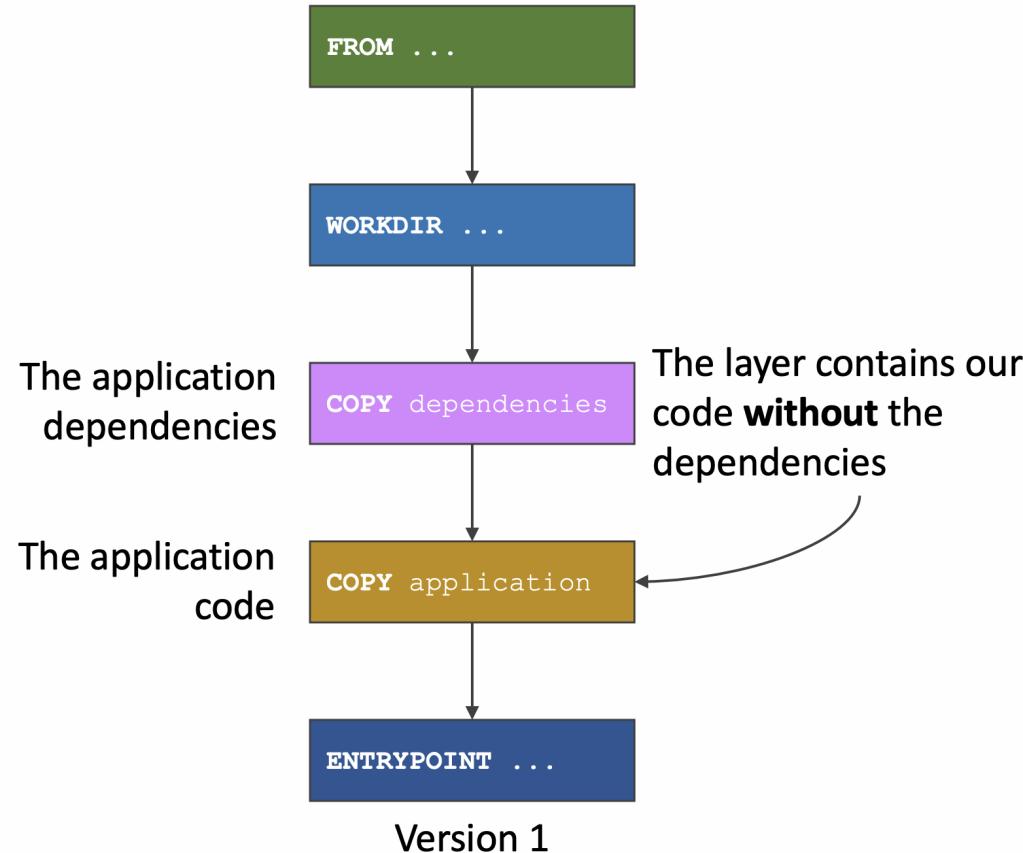


Solution: Splitting the FatJAR

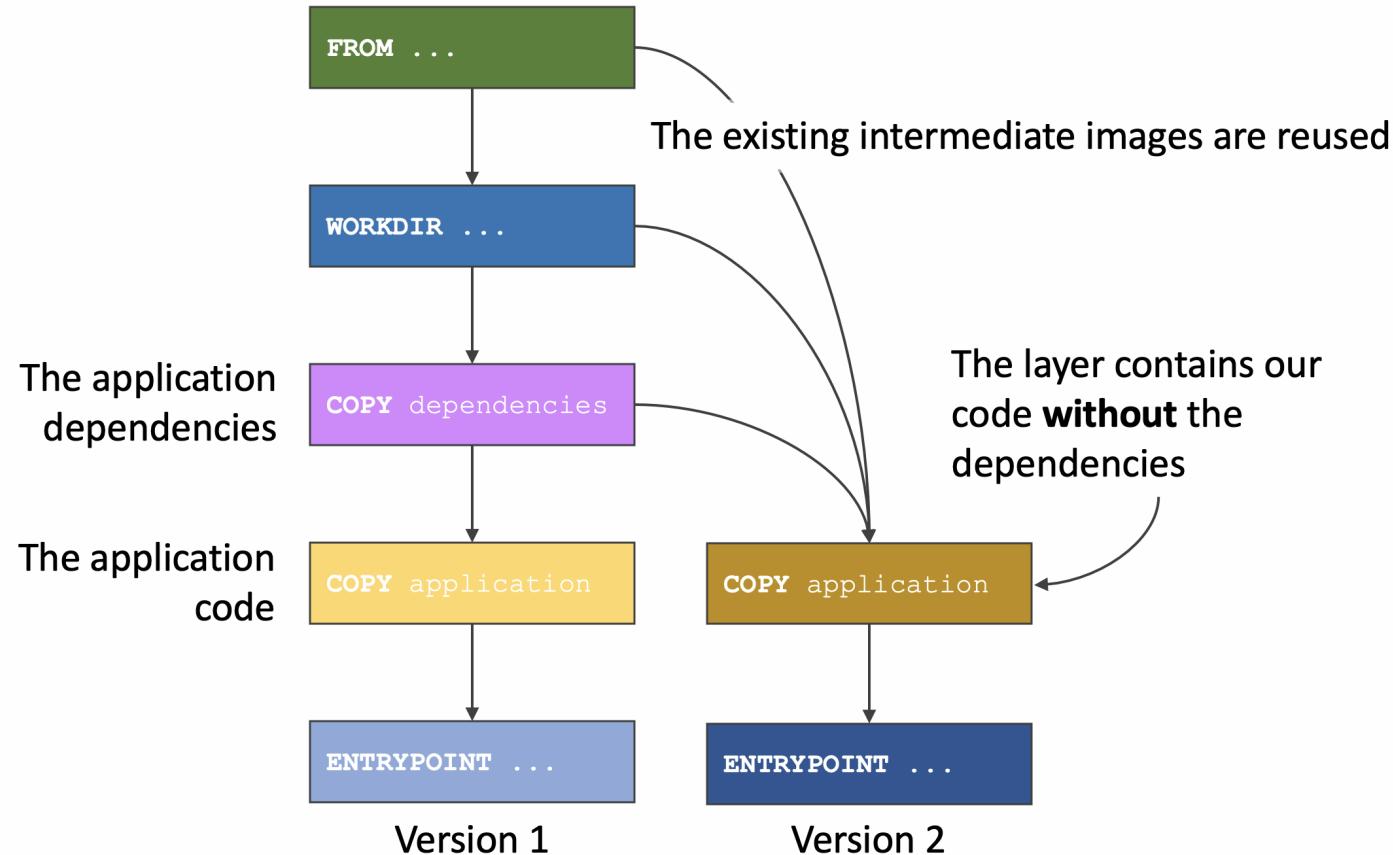
- Given that some parts of the FatJAR, the dependencies, change less frequently but take up a lot of space

➤ **Solution: Splitting the dependencies from the code by creating a new layer and taking advantage of caching**

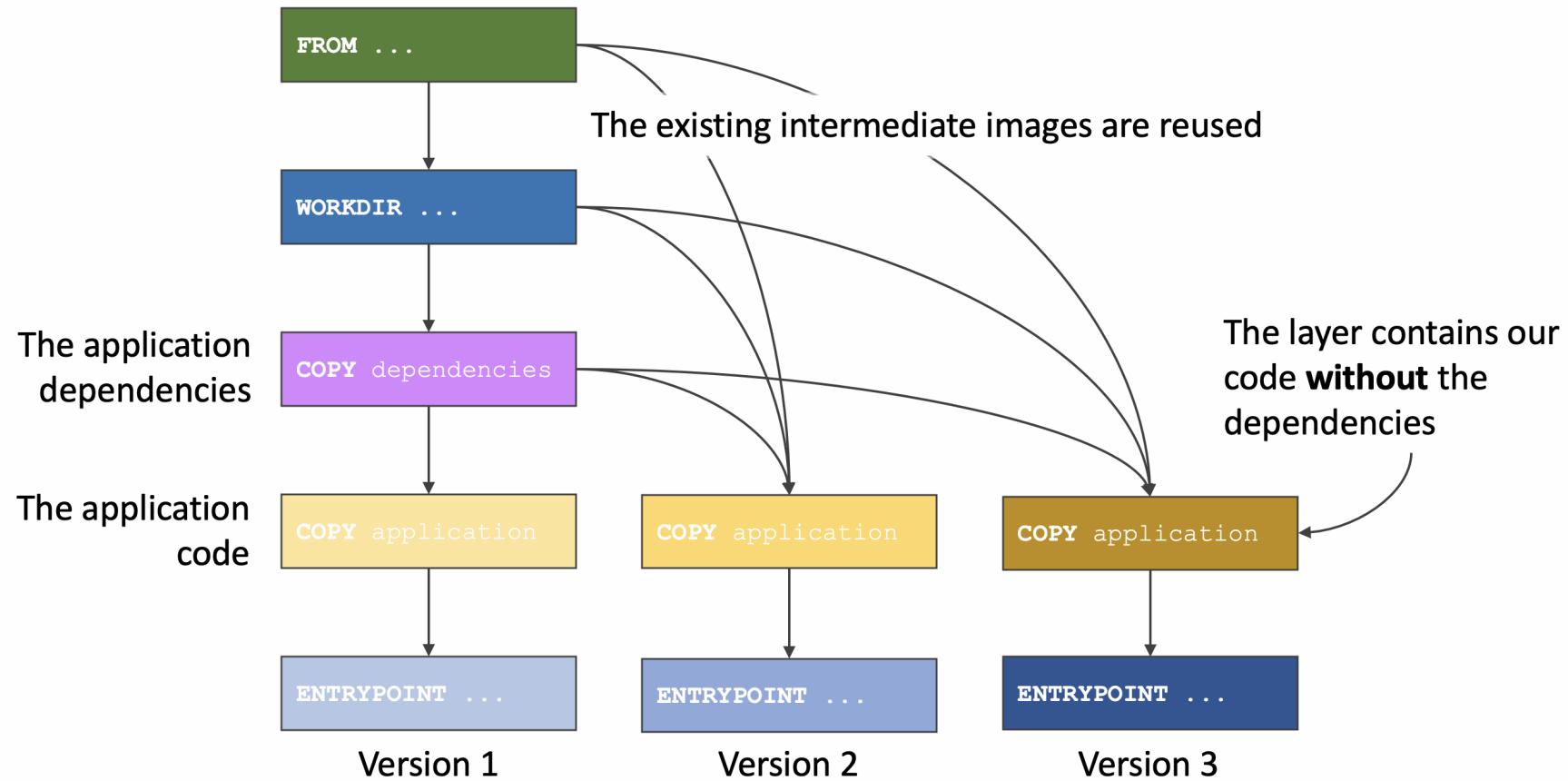
Splitting the FatJAR - Version 1



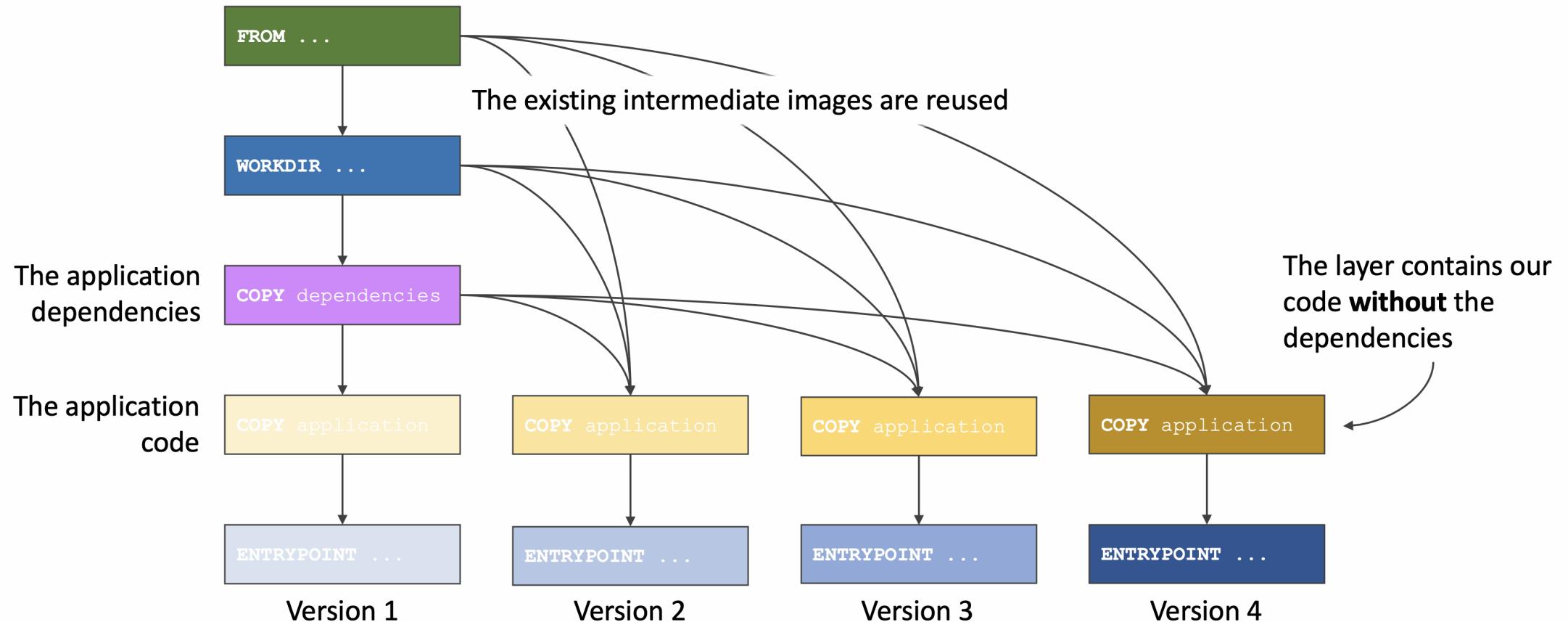
Splitting the FatJAR - Version 2



Splitting the FatJAR - Version 3



Splitting the FatJAR - Version 4



Spring Boot

Spring Boot

Spring Boot is a very popular framework that promotes productivity



<https://spring.io/projects/spring-boot>

Layered JAR

- Spring Boot 2.3 comes with a new feature, *Layered JAR*
- This can be enabled by simply adding a configuration to the Gradle `build.gradle` file

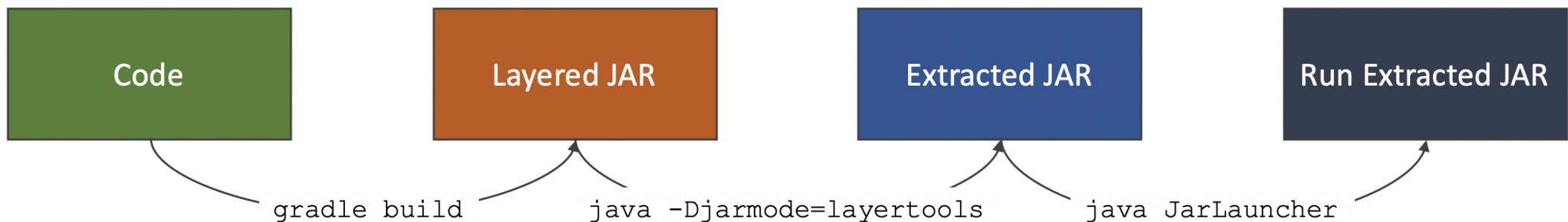
```
bootJar {  
    layered()  
}
```

An equivalent plugin is available for Maven too

- Spring Boot 2.4 will have *Layered JAR* enabled by default

How does this work?

- Build the Layered JAR (using *Gradle*)
- Extract the Layered JAR (using *layertool*)
- Run the Extracted JAR (using *JarLauncher*)



Demo 4

Build layered JAR, extract it and run extracted JAR

- Build layered JAR
- Extract Layered JAR
- Run Extracted JAR

How does this work with docker?

- We can take advantage of multistage docker builds

```
FROM adoptopenjdk:8u262-b10-jre-hotspot as builder
WORKDIR /opt/app
COPY ./build/libs/*.jar application.jar
RUN java -Djarmode=layer-tools -jar application.jar extract

FROM adoptopenjdk:8u262-b10-jre-hotspot
WORKDIR /opt/app
COPY --from=builder /opt/app/dependencies ./
COPY --from=builder /opt/app/spring-boot-loader ./
COPY --from=builder /opt/app/snapshot-dependencies ./
COPY --from=builder /opt/app/application ./
ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]
```

Builder stage

- Copy the layered JAR created by Gradle

```
FROM adoptopenjdk:8u262-b10-jre-hotspot as builder
WORKDIR /opt/app
COPY ./build/libs/*.jar application.jar
```

- Extract the layered JAR

```
RUN java -Djarmode=layer-tools -jar application.jar extract
```

This command will create four folders in the builder stage, which we will copy in the final stage

Final stage

- Starts with a Java image

```
FROM adoptopenjdk:8u262-b10-jre-hotspot
WORKDIR /opt/app
```

- Copy the extracted folders from the builder stage

```
COPY --from=builder /opt/app/dependencies ./
COPY --from=builder /opt/app/spring-boot-loader ./
COPY --from=builder /opt/app/snapshot-dependencies ./
COPY --from=builder /opt/app/application ./
```

- Run the application using `JarLauncher`

```
ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]
```

Demo 5

Create docker image using layered JAR and analyse it with dive

- Create docker image (using multistage and layered JAR)
- Inspect with Dive

Size of Layered JAR

[● Layers]			[Current Layer Contents]									
Cmp	Image ID	Size	Command	Permission	UID:GID	Size	Filetree					
	sha256:b187ff70b2e47a4cf3	63 MB	#(nop) ADD file:1e8d02626176dc8141df	drwxr-xr-x	0:0	5.4 kB	└ opt					
	sha256:5930c9e5703fbc1f5	988 kB	[-z "\$(apt-get indextargets)"]	drwxr-xr-x	0:0	5.4 kB	└ app					
	sha256:c64c52ea2c16427957	745 B	set -xe && echo '#!/bin/sh' > /usr	drwxr-xr-x	0:0	5.1 kB	└ BOOT-INF					
	sha256:ddc500d8499442f954	7 B	mkdir -p /run/systemd && echo 'docke	-----	0:0	3.8 kB	└ classes					
	sha256:80b956beb7fc101928	34 MB	apt-get update && apt-get instal	-rw-r--r--	0:0	0 B	└ application.yaml					
	sha256:840c5fa83b6d4aece6	108 MB	set -eux; ARCH="\$(dpkg --print-a	-rw-r--r--	0:0	841 B	└ banner.txt					
	sha256:8254d5025e79d1004c	0 B	#(nop) WORKDIR /opt/app	-----	0:0	2.9 kB	└ demo					
	sha256:692506c763db9b5d6e	16 MB	#(nop) COPY dir:27eb428921bb39f342ab	-----	0:0	2.9 kB	└ boot					
	sha256:b6862a0645df6fc4ff	235 kB	#(nop) COPY dir:474092a2fe876588a3be	-rw-r--r--	0:0	716 B	└ LayeredJarDe					
	sha256:f698f3e6e37098fdcf	0 B	#(nop) COPY dir:21af5f98e4096969a58f	-rw-r--r--	0:0	1.5 kB	└ Message.clas					
	sha256:a6a16933a06959a667	5.4 kB	FROM sha256:a6a16933a06959a667	-rw-r--r--	0:0	669 B	└ MessageContr					
				-rw-r--r--	0:0	1.1 kB	└ classpath.idx					
				-rw-r--r--	0:0	212 B	└ layers.idx					
				-----	0:0	329 B	└ META-INF					
				-rw-r--r--	0:0	329 B	└ MANIFEST.MF					
[Layer Details]			Our Layered JAR creates a layer of 5.4KB									
Digest: sha256:a6a16933a06959a6677df48a2800a5aa5f2b4ba29cf525206f0b2a21c278 4db2												
Command: #(nop) COPY dir:ded5188ae0fcd248095114af7db0165e908616f58b6d4679e93d762945a 2d3bd in ./												
[Image Details]												

Comparison

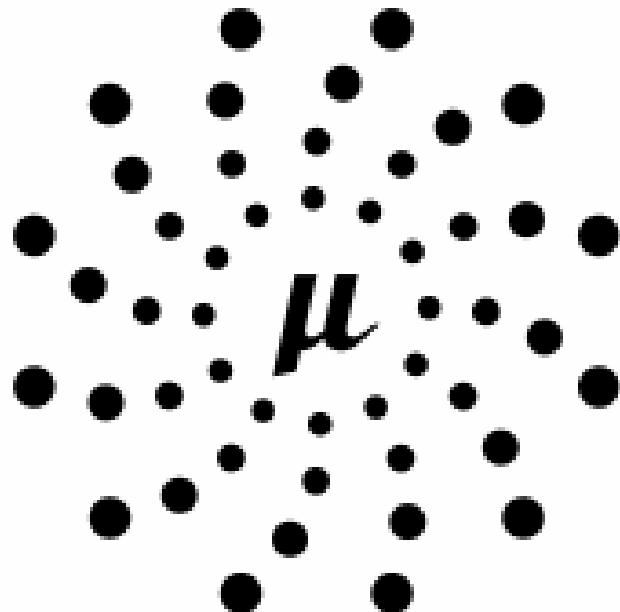
- Comparing these two approaches we will find that a layered JAR is far more efficient than a FatJAR in terms of disk space



Beyond Spring Boot

Micronaut

Micronaut is reflection free alternative framework to Spring Boot



M I C R O N A U TTM

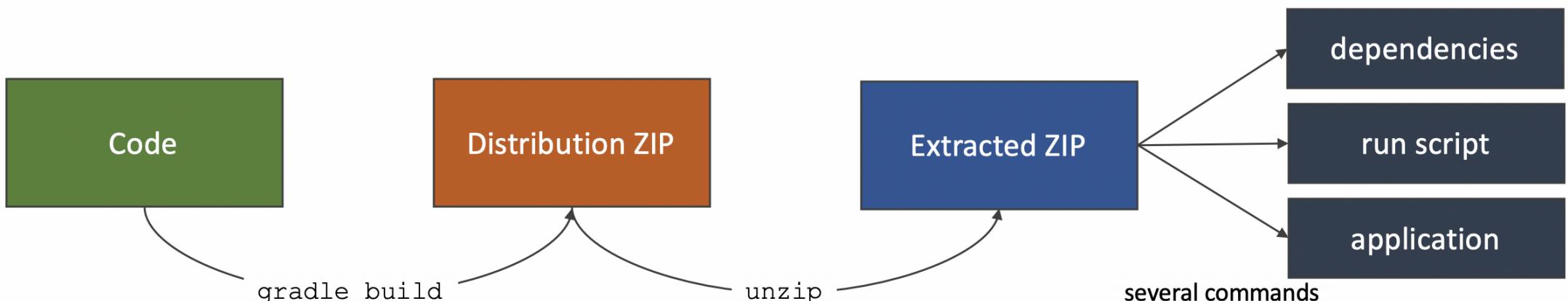
<https://micronaut.io/>

Lack of tooling

- Spring Boot provides the layered JAR functionality as a Gradle task
- This is not available for all other frameworks
- We can still take advantage of the docker multistage to split our dependencies from the application manually

How will this work?

- Extract the distribution ZIP file
generated by the `distribution` Gradle plugin which is applied by the `application` Gradle plugin
- Move our application thin JAR to its own directory
- Update the run script



Multistage to the rescue

```
FROM alpine:3.12.0 as builder
WORKDIR /opt/app
COPY ./build/distributions/*.zip application.zip
RUN unzip application.zip && rm application.zip \
    && mv * dist && rm dist/bin/*.bat && mv dist/bin/* dist/bin/run.original \
    && sed 's|${APP_HOME}/lib/application.jar|${APP_HOME}/app/application.jar|g' dist \
    && chmod +x dist/bin/run \
    && rm dist/bin/run.original \
    && mkdir dist/app \
    && mv dist/lib/application.jar dist/app/application.jar

FROM adoptopenjdk:8u262-b10-jre-hotspot
ENV APP_HOME /opt/app
WORKDIR ${APP_HOME}
COPY --from=builder /opt/app/dist/lib lib/
COPY --from=builder /opt/app/dist/bin bin/
COPY --from=builder /opt/app/dist/app app/
ENTRYPOINT ["./bin/run"]
```

Demo 6

Create docker image using distribution ZIP and analyse it with dive

- Go through the `micronaut-layered-jar-1.0.zip` file
- Go through the multistage docker file
- Create docker image (using multistage)
- Inspect with Dive

Comparison

[Current Layer Contents]					
	Size	Command	Permission	UID:GID	
87ff70b2e47a4cf3	63 MB	#(nop) ADD file:1e8d02626176dc8141df	drwxr-xr-x	0:0	
30c9e5703fbc1f5	988 kB	[-z "\$(apt-get indextargets)"]	drwxr-xr-x	0:0	
4c52ea2c16427957	745 B	set -xe && echo '#!/bin/sh' > /usr	-rw-r--r--	0:0	
c500d8499442f954	7 B	mkdir -p /run/systemd && echo 'docke			
b956beb7fc101928	34 MB	apt-get update && apt-get instal			
0c5fa83b6d4aece6	108 MB	set -eux; ARCH="\$(dpkg --print-a			
54d5025e79d1004c	0 B	#(nop) WORKDIR /opt/app			
6c2f015303133c75	14 MB	FROM sha256:526c2f015303133c75			
s]					
6:526c2f015303133c756c05fd7bbbb11ae4db2d0cb4d87316bca3bda8d2fc					
ile:facc456b0f59c11ede957d55b2c36beec967cbf89255b78b7547206621					
lication.jar					
s]					
ize: 219 MB					
ted space: 2.7 MB					
ency score: 99 %					
Space Path					

[Current Layer Contents]					
	Size	Command	Permission	UID:GID	
187ff70b2e47a4cf3	63 MB	#(nop) ADD file:1e8d02626176dc8141df	drwxr-xr-x	0:0	
930c9e5703fbc1f5	988 kB	[-z "\$(apt-get indextargets)"]	drwxr-xr-x	0:0	
64c52ea2c16427957	745 B	set -xe && echo '#!/bin/sh' > /usr	-----	0:0	
dc500d8499442f954	7 B	mkdir -p /run/systemd && echo 'docke	-rw-r--r--	0:0	
0b956beb7fc101928	34 MB	apt-get update && apt-get instal			
40c5fa83b6d4aece6	108 MB	set -eux; ARCH="\$(dpkg --print-a			
575556da0f805539f	0 B	#(nop) WORKDIR /opt/app			
8650535851277c651	14 MB	\$(nop) COPY dir:800b8003edbc03ed93f0			
d292524ed985dceae	7.5 kB	\$(nop) COPY dir:a1c46104db044df76573			
10c51517e01b71470	10 kB	FROM sha256:710c51517e01b71470			
ls]					
56:710c51517e01b71470a14b35c46d93519757f9522e7cf99d3377b45a15					
dir:05787488b2a186e6ea931543f9c37239b510729f522bde09e38f1ac2879					
/					
ls]					
size: 219 MB					
ted space: 2.7 MB					
ency score: 99 %					
Space Path					

We went down from 14MB to 10KB

Thank You

Feedback makes us better

Please send any feedback to: albert.attard@thoughtworks.com or
jlang@thoughtworks.com

WE ARE HIRING

26 YEARS EXPERIENCE

43 OFFICES IN 14 COUNTRIES

THOUGHT LEADER IN AGILE SOFTWARE

DEVELOPMENT AND CONTINUOUS

DELIVERY

+7.000 THOUGHTWORKERS WORLDWIDE

+300 THOUGHTWORKERS IN GERMANY

de-recruiting@thoughtworks.com

