

# Building container images

## Dockerfiles

일반적으로 container image로 만들 때 활용하는 `Dockerfile` 을 정의해 사용하는 방법

Spring project의 결과물을 container image로 만들 때, 아주 간단히 `Dockerfile` 을 작성하면 다음과 같이 작성할 수 있습니다.

- Gradle인 경우, `./gradlew bootJar` 을 통해 쉽게 jar로 패키징할 수 있습니다.
- 패키징 이후, `docker build` 명령어를 통해 도커 이미지를 만들 수 있습니다.

```
FROM openjdk:11-jre-slim
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```


위 Dockerfile 대로 동작하면 이상은 없겠지만, 프로젝트가 커질 수록. 즉, 의존성 및 코드의 양이 증가되어 jar 파일이 무거워질수록 Container image를 만드는 시간이 오래걸리게 됩니다.

Docker는 layer를 활용하여 기존 layer가 local에 있으면 재사용하는 식(캐쉬)으로 진행되어 효율화를 이룬 아키텍처를 가지고 있 는데, 위 Dockerfile처럼 구성하면 Java의 모든 구조가 jar 파일로 되는 것이기 때문에 layer를 재사용하기 어렵습니다. 다시 말해서, 코드를 한 줄이라도 수정하게 되면 캐쉬 해놓은 layer를 사용할 수 없기 때문에 다시 만드는 작업이 필요하게 됩니다.

이러한 문제를 해결하기 위해서 layering feature를 사용합니다. 쉽게 말하면 하나로 이루어졌던 구조를 4개의 Layer로 나눔으로써 변경이 적은 부분은 캐싱된 것을 사용하게 만들고, 변경이 잦은 부분만을 새로 구축해 효율적으로 container image를 만들 수 있는 방법입니다.

4개의 Layer는 아래의 표의 4가지로 구성되어져 있습니다.

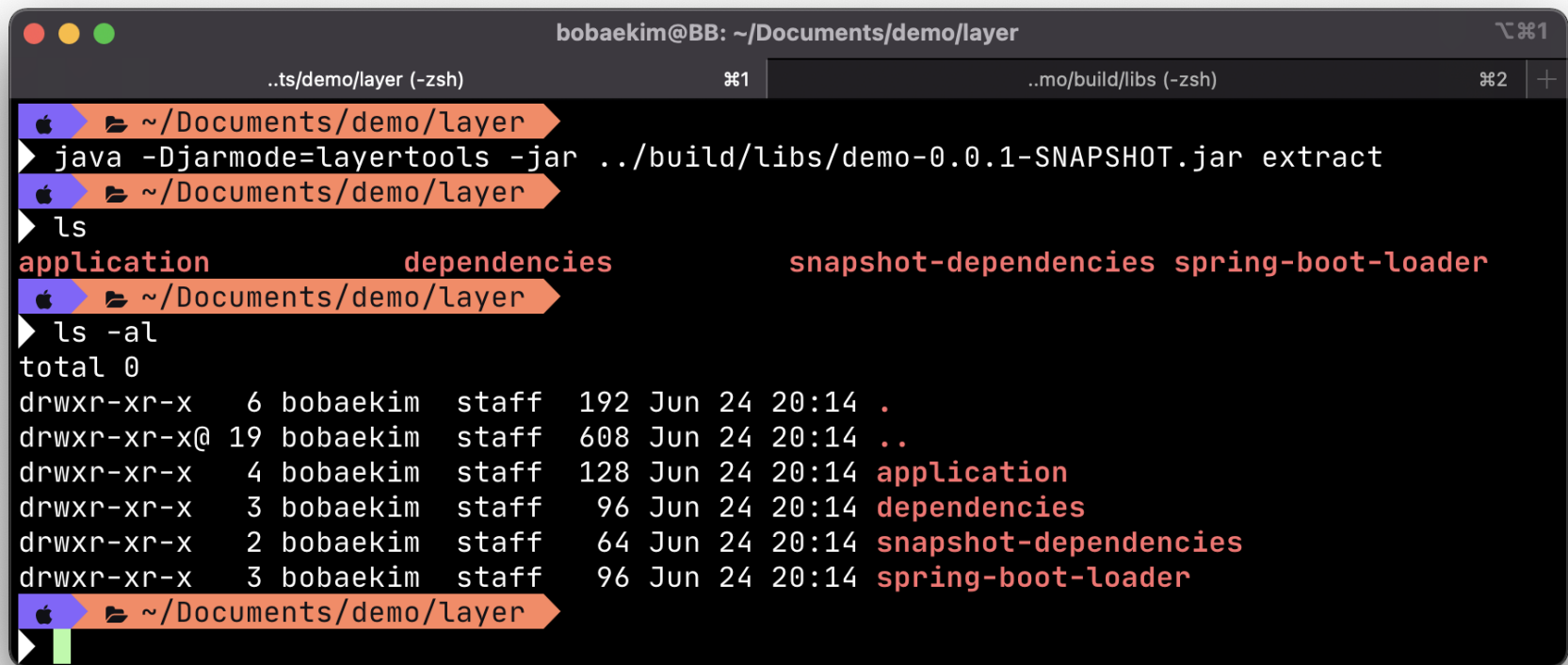
### Layer

<u>Aa</u> layer name (directory)	 description
<u>application</u>	application classes and resources
<u>snapshot-dependencies</u>	any dependency whose version contains SNAPSHOT
<u>spring-boot-loader</u>	the jar loader classes
<u>dependencies</u>	any dependency whose version does not contain SNAPSHOT

- dependencies가 제일 변경이 적고, application이 제일 변경이 잦은 부분이라고 보시면 됩니다.

그럼, 이제 jar 파일을 위의 4가지 Layer로 분리해보도록 하겠습니다. 아래의 명령어를 통해 jar 파일을 4개의 layer로 분리할 수 있습니다.

```
$ java -Djarmode=layertools -jar my-app.jar extract
```



```
bobaekim@BB: ~/Documents/demo/layer
..ts/demo/layer (-zsh)
> java -Djarmode=layertools -jar ../build/libs/demo-0.0.1-SNAPSHOT.jar extract
> ls
application      dependencies      snapshot-dependencies  spring-boot-loader
> ls -al
total 0
drwxr-xr-x  6 bobaekim  staff  192 Jun 24 20:14 .
drwxr-xr-x@ 19 bobaekim  staff  608 Jun 24 20:14 ..
drwxr-xr-x  4 bobaekim  staff  128 Jun 24 20:14 application
drwxr-xr-x  3 bobaekim  staff   96 Jun 24 20:14 dependencies
drwxr-xr-x  2 bobaekim  staff   64 Jun 24 20:14 snapshot-dependencies
drwxr-xr-x  3 bobaekim  staff   96 Jun 24 20:14 spring-boot-loader
```

이제 이 분할한 layer를 Dockerfile에서 사용한다면, 다음과 같이 작성할 수 있습니다.

```
FROM adoptopenjdk:11-jre-hotspot as builder
WORKDIR application
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} application.jar
RUN java -Djarmode=layertools -jar application.jar extract

FROM adoptopenjdk:11-jre-hotspot
WORKDIR application
COPY --from=builder application/dependencies/ ./
COPY --from=builder application/spring-boot-loader/ ./
COPY --from=builder application/snapshot-dependencies/ ./
COPY --from=builder application/application/ ./
ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]
```

- `docker build` 명령어를 통해 Optimized container image를 생성해낼 수 있습니다.

## Cloud Native Buildpacks

위 방식처럼 Dockerfile을 직접 만들어 Container image를 생성할 수 있지만 최근에는 Maven, Gradle에서 바로 Container image를 생성할 수 있도록 방법을 제공하고 있습니다. 즉, Dockerfile을 정의할 필요 없이 어디서든 실행할 수 있도록 Container image를 만들 수 있도록 지원하고 있는 것입니다. 그게 바로 Buildpack 입니다.

Cloud Native Buildpacks은 어플리케이션 코드를 어느 클라우드에서나 돌아갈 수 있도록 이미지로 전환을 해주는 기술입니다.

Cloud Native Buildpacks transform your application source code into images that can run on any cloud.

위에서 직접 Dockerfile로 생성하는 것처럼 효율적으로 Container image를 생성해줍니다.

Buildpack은 CNCF 프로젝트 중 하나로 표준의 기술의 하나로 인정받고 있습니다.

## Incubating

[View on the CNCF landscape »](#)


Continuous Integration & Delivery



Buildpacks.io

Application Definition & Image Build



cloudevents

Streaming & Messaging



CNI

Cloud Native Network



Service Proxy



cortex

Monitoring



cri-o

Container Runtime



Dragonfly

Container Registry



- 관심 있다면 자세한 내용은 [공식 Docs](#)를 참조 부탁드립니다.

## Maven

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>build-image</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

- `mvn spring-boot:build-image`

## Gradle

- `./gradlew bootBuildImage`