ch06 스프링부트 컨테이너화

6.0 주요내용

- 도커에서 컨테이너 이미지로 작업
- 스프링부터 애플리케이션을 컨테이너 이미지로 패키징
- 도커 컴포즈를 사용한 스프링부트 컨테이너 관리
- 깃허브 액션을 사용한 이미지 빌드 및 푸시 자동화

6.1 도커에서 컨테이너 이미지로 작업하기

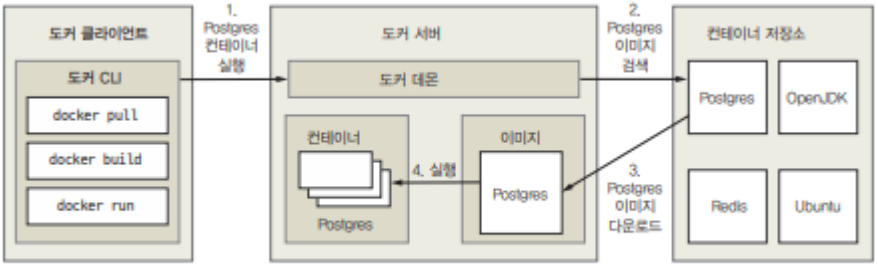


그림 6.1 도커 엔진은 클라이언트/서버 아키텍처를 가지고 있고 컨테이너 저장소와 상호작용한다.

도커서버는 컨테이너 저장소와 연결해 이미지를 업로드하고 다운로드.

컨테이너 이미지는 애플리케이션을 실행하는 데 필요한 모든 것을 그 안에 가지고 있는 경량 실행 파일 패키지다.

6.1.1 컨테이너 이미지 이해

컨테이너 이미지는 여러개의 명령을 순서대로 실행한 결과물로,

각 명령의 실행한 결과로 만들어진 레이어로 이루어진다.

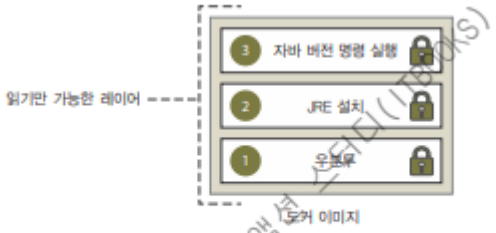


그림 6.2 컨테이너 이미지는 수정이 불가능한 읽기 전용 레이어를 순서대로 쌓아 올려 구성된다. 첫 번째 레이어는 베이스 이미지를 나타내며, 나머지는 그 위에 적용된 수정을 나타낸다.

컨테이너 이미지의 모든 레이어는 읽기 전용이다.

무언가를 변경해야 한다면, 기존의 레이어 위에 새로운 레이어를 적용해야한다(카피 온 라이트)

이미지가 컨테이너로 실행되면, 컨테이너 레이어라고 부르는 마지막 레이어가

기존의 레이어 위에 자동으로 적용된다.

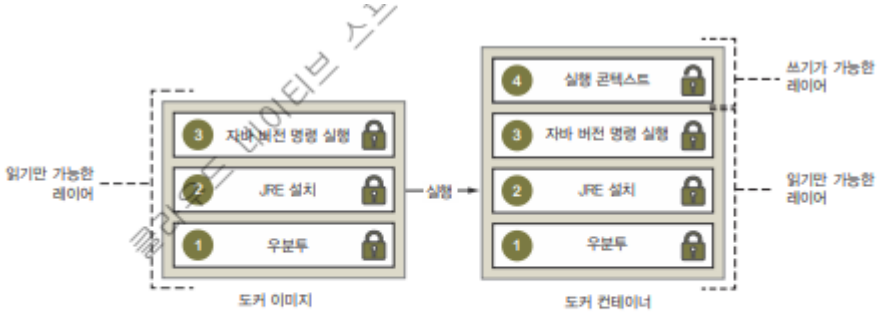


그림 6.3 실행 중에 있는 컨테이너에는 이미지 레이어 위에 여분의 레이어가 있다. 유일하게 쓸 수 있는 레이어지만 휘발성이 있다는 것을 기억해야 한다.

해당 레이어는 유일하게 쓰기가 가능한 레이어로, 컨테이너 실행 중 생성되는 휘발성 데이터를 저장하기 위해 사용된다.

6.1.2 도커파일을 통한 이미지 생성

OCI 형식에 따라 도커파일이라고 부르는 특정 파일에 명어를 순서대로 나열해 컨테이너 이미지를 정의한다.

아래 표는 도커파일에서 정의할 수 있는 일반적인 명령어다

표 6.1 컨테이너 이미지를 만들기 위해 도커파일에서 사용하는 가장 일반적인 명령어

명령	설명	예
FROM	후속 명령어의 대상이 될 베이스 이미지를 정의한다. 도커파일에서 가장 처음에 와야 하는 명령이다.	FROM ubuntu:22.04
LABEL	키-값 형식에 따라 이미지에 메타데이터를 추가한다. LABEL 명령은 여러 번 정의할 수 있다.	LABEL version="1.2.1"
ARG	사용자가 빌드 시에 전달할 수 있는 변수를 정의한다. ARG 명령은 여러 번 정의할 수 있다.	ARG JAR_FILE
RUN	기존 레이어 위에서 인자로 전달된 명령어를 실행해 새로운 레이어를 생성한다. RUN 명령은 여러 번 정의할 수 있다.	RUN apt-get update && apt-get install -y default-jre
COPY	호스트 파일 시스템의 파일 또는 디렉터리를 컨테이너 내부의 파일 또는 디렉터리로 복사한다.	COPY app-0.0.1-SNAPSHOT.jar app.jar
USER	모든 후속 명령어 및 컨테이너 실행을 위한 사용자를 정의한다.	USER sheldon
ENTRYPOINT	이미지가 컨테이너로 실행될 때 실행할 프로그램을 정의한다. 여러 개의 ENTRYPOINT 명령이 있으면 마지막 명령만 고려된다.	ENTRYPOINT ["/bin/bash"]
CMD	실행 중인 컨테이너에 대한 기본 설정을 지정한다. ENTRYPOINT 명령이 정의되어 있으면 그 명령의 인수로 전달되고 그렇지 않으면 실행 파일도 가능하다. 도커파일에 여러 개의 CMD 명령이 있으면 마지막 명령만 고려된다.	CMD ["sleep", "10"]

도커파일에 컨테이너 이미지를 만들기 위한 명세를 다 마치면,

docker build 명령을 실행해 도커파일 내의 모든 명령을 하나씩 수행하면서,

각 명령에 대한 새로운 레이어가 생성된다.

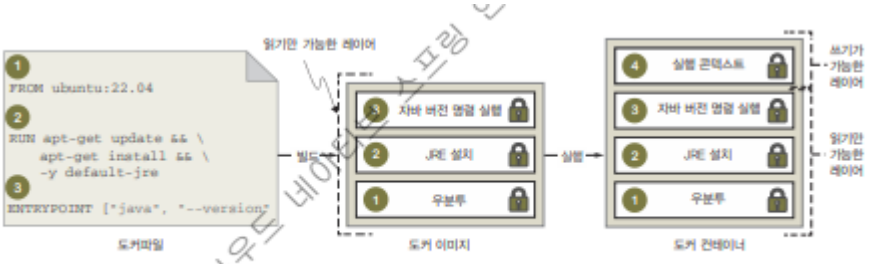


그림 6.4 이미지 생성은 도커파일부터 시작된다. 도커파일의 각 명령은 이미지의 레이어를 순서대로 만든다.

docker build 명령어 실행 예시는 아래 그림과 같다.

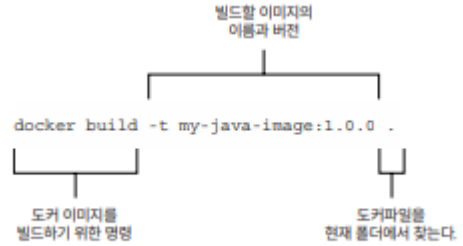


그림 6.5 주어진 이름과 버전으로 새 이미지를 빌드하는 도커 CLI 명령

빌드가 완료된 이미지는 docker image 명령어를 통해 상세 정보를 확인할 수 있다.

해당 이미지는 docker run 명령으로 실행이 가능하다.

6.1.3 깃허브 컨테이너 저장소로 이미지 저장

컨테이너 저장소에 이미지를 업로드 하는 방법을 살펴보자

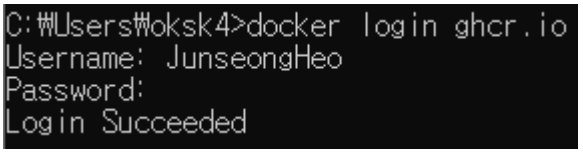
깃허브 컨테이너 저장소를 사용해보자.

깃허브 Settings > Developer Settings > Personal access Tokens로 이동하여 Generate New Token

이름 입력, repo, write:package 선택

생성한 토큰값 복사

터미널에서 깃허브저장소 로그인.



docker tag명령어로 컨테이너 저장소에 저장하기 위한 이름 지정

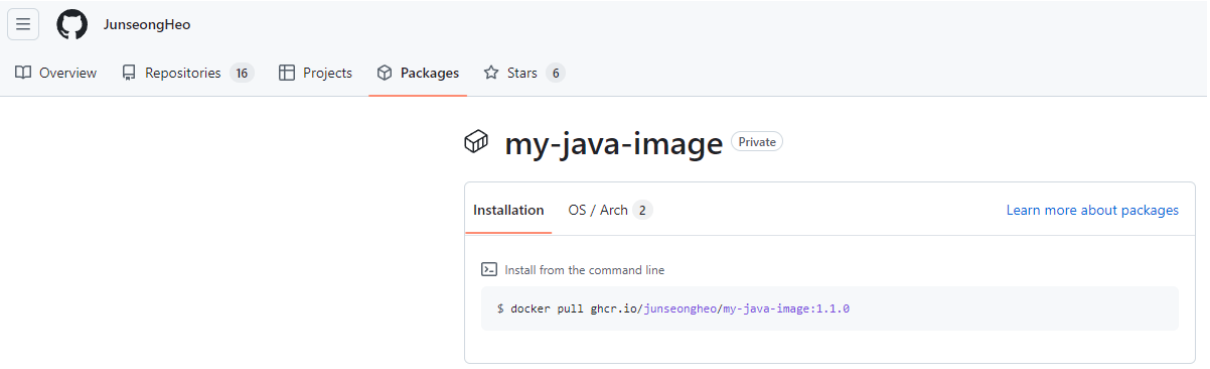
```
docker tag my-java-image:1.1.0 ghcr.io/junseongheo/my-java-im
```

지정된 이름은 docker images에서 확인 가능함.

깃허브 컨테이너 저장소에 이미지 업로드

```
docker push ghcr.io/junseongheo/my-java-image:1.1.0
```

업로드 된 이미지는 깃허브 > your profile > packeages에서 확인



6.2 스프링부트 애플리케이션 컨테이너 이미지로 패키지

6.2.1 스프링부트 컨테이너 화를 위한 준비

- 포트 전달을 사용한 애플리케이션 서비스 노출

  포트포워딩으로 외부에서 컨테이너화된 애플리케이션 액세스 설정

  ```
  docker run 실행컨테이너 -p 8080:8080
  ```

- 서비스 발견을 위한 도커 내장 DNS 서버의 사용

  도커는 동일한 네트워크 컨테이너가 호스트 이름이나 IP 주소가 아닌 컨테이너 이름을 사용해 서로를 찾을 수 있는 DNS 서버를 내장하고 있음.

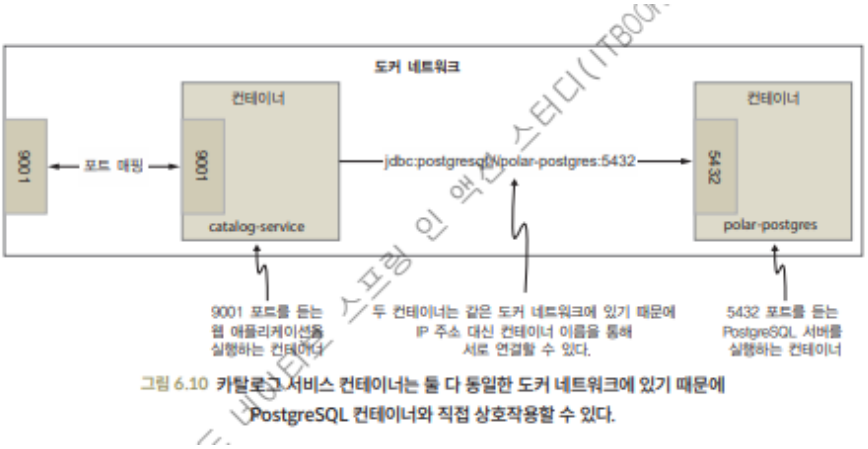


그림 6.10 카탈로그 서비스 컨테이너는 둘 다 동일한 도커 네트워크에 있기 때문에 PostgreSQL 컨테이너와 직접 상호작용할 수 있다.

해당 DNS서버를 활용하여 도커 네트워크를 만들어보자.

```
// 네트워크 생성
docker network create catalog-network

// 네트워크 확인
docker network ls
```

그런 다음 —net 인수를 활용하여 PostgreSQL 컨테이너를 시작하면서 카탈로그 네트워크에 연결하도록 지정할 수 있다.

```
docker run -d --name polar-postgres --net catalog-network -e
```

6.2.2 도커파일로 스프링부트 컨테이너화

먼저 베이스 이미지를 정하고, 도커 파일을 작성한다.

다음으로, catalog-service 애플리케이션을 jar 아티팩트로 빌드한다.

```
gradlew.bat clean bootJar
```

그 다음으로 컨테이너 이미지를 생성한다.

```
docker build -t catalog-service .
```

생성된 컨테이너 이미지를 실행시킨다

```
docker run -d \
  --name catalog-service \
  --net catalog-network \
  -e 9001:9001 \
  -e SPRING_DATASOURCE_URL=jdbc:postgresql://polar-postgres:5
  -e SPRING_PROFILES_ACTIVE=testdata \
  catalog-service
```

6.2.3 프로덕션을 위한 컨테이너 이미지 빌드

앞에서 만든 이미지를 개선하는 방법.

스프링부트가 제공하는 계층화된 JAR 기능을 사용해 보다 효율적인 이미지 만들기

- 성능

  컨테이너 이미지를 만들 때, 빌드할 때와 런타임 성능을 고려해야 함.

  변경할때마다 전체 레이어를 다시 작성하는 것은 비효율적.
  계층화된 JAR 모드를 사용한 패키징은 컨테이너 이미지와 유사하게 여러 레이어로 만들어진다.
  자주 변경되는 클래스를 분리할 수 있다.
  기본적으로 스프링 부트 애플리케이션은 다음과 같은 계층으로 이루어진다.

  - 의존성 계층 : 프로젝트에 추가된 모든 주요 의존성
  - 스프링부트 로더 계층 : 스프링부트 로더 컴포넌트가 사용하는 클래스
  - 스냅숏 의존성 계층 : 모든 스냅숏 의존성
  - 애플리케이션 계층 : 애플리케이션 클래스 및 리소스

  기존 애플리케이션에 새로운 REST 앤드포인트를 추가하는 시나리오라면 애플리케이션 계층만 작성하면 된다.

원래의 JAR 파일은 이미지에 있지 않도록 도커가 제공하는 다단계 빌드를 활용한다.
첫 번째 단계는 JAR파일에서 계층을 추출하고,
두 번째 단계는 각 JAR계층을 별도의 이미지 레이어로 배치한다.
최종적으로 첫 번째 단계의 결과는 폐기되고 두 번째 단계에서 이미지가 생성된다.

```
FROM eclipse-temurin:17 AS builder
WORKDIR workspace
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} catalog-service.jar
RUN java -Djarmode=layertools -jar catalog-service.jar extract

FROM eclipse-temurin:17
WORKDIR workspace
COPY --from=builder workspace/dependencies/ ./
COPY --from=builder workspace/spring-boot-loader/ ./
COPY --from=builder workspace/snapshot-dependencies/ ./
COPY --from=builder workspace/application/ ./
ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]
```

- 보안

  컨테이너는 기본적으로 루트 사용자로 실행되므로, 잠재적으로 도커 호스트에 루트 액세스가 허용된다.

  최소 권한의 원칙에 따라 필요 이상의 권한을 갖지 않는 사용자를 만들어
  해당 사용자가 도커파일에 정의된 진입점 프로세스를 실행하게 되면 위험을 완화할 수 있다.

  도커파일에서 최신 베이스 이미지와 라이브러리를 사용하는 것도 중요하다.

  grype 명령어를 통해 이미지에 취약성이 있는지 검사할 수 있다.

  ```
  grype catalog-service
  ```

- 도커파일 v.s. 빌드팩

  도커파일은 결과를 세밀하게 제어할 수 있으나, 유지보수 에 추가적인 노력이 필요하고 가치흐름에 몇가지 문제가 발생할 수 있다.

  개발자가 성능 및 보안 문제를 모두 다루고 싶지 않을 수도있으며,
  애플리케이션 코드에 집중하기를 원할 수 있다.

  클라우드 네이티브 빌드팩은 일관성, 보안, 성능 및 거버넌스에 중점을 둔 다른 접근법을 제공한다.

  개발자가 도커파일을 작성하지 않고도 적절한 이미지를 자동으로 생성할 수 있다.

6.2.4 클라우드 네이티브 빌드팩을 이용한 스프링부트 컨테이너 화

빌드팩이 제공하는 기능은 다음과 같다.

- 애플리케이션 유형을 자동으로 감지해 도커파일 없이 패키지 생성
- 다양한 언어와 플랫폼을 지원
- 캐시와 레이어를 통해 높은 성능을 갖는다
- 재현 가능한 빌드를 보장한다.
- 보안 측면에서 모범 사례를 사용한다.

- 프로덕션 환경에 적합한 이미지를 만든다.
- 그랄VM을 사용해 네이티브 이미지의 빌드를 지원한다.

스프링부트 플러그인에서 제공하는 빌드팩 통합은 카탈로그 서비스 프로젝트의 build.gradle 파일에서 설정할 수 있다.

설정 후 아래의 명령어를 실행하여 이미지 생성이 가능하다.

```
gradlew.bat bootBuildImage
```

스프링부트 2.4 이후로는 build.gradle의 스프링부트 플러그인 설정으로 이미지를 컨테이너 저장소로 직접 저장할 수도 있다.

```
docker {
    publishRegistry {
        username = project.findProperty("registryUsername")
        password = project.findProperty("registryToken")
        url = project.findProperty("registryUrl")
    }
}
```

크리덴셜 황금률을 위해 해당 설정값들은 실행 명령어에서 인수로 넘겨준다.

6.3 도커 컴포즈를 통한 스프링부트 컨테이너의 관리

빌드팩을 사용하면 도커파일을 직접 작성하지 않고도 스프링부트 애플리케이션을 빠르고 효율적으로 컨테이너화 할 수 있다.

그러나 도커 CLI는 번거롭고, 오타발생의 위험과 버전관리의 어려움이 있다.

도커 컴포즈는 도커 CLI보다 더 나은 사용자경험을 제공한다.

6.3.1 도커 컴포즈를 통한 컨테이너 라이프사이클 관리

도커 컴포즈 구문은 매우 직관적이고 자명하며 도커 CLI 인수에 일대일로 매핑될 때가 많다.

모든 배포 관련 스크립트를 별도의 코드베이스에, 가능하다면 별도의 저장소를 통해 모아놓는 것이 좋다.

도커 컴포즈는 두 개의 컨테이너를 기본 설정상 같은 네트워크에 연결하기 때문에 이전처럼 명시적으로 네트워크를 지정할 필요가 없다.

6.4 배포 파이프라인 : 패키지 및 등록

6.4.1 커밋 단계에서 릴리스 후보 빌드

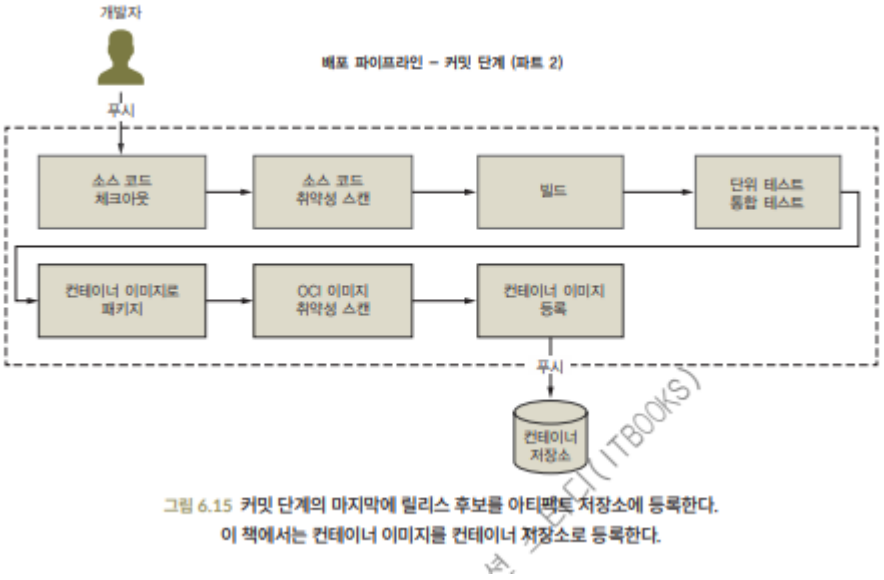


그림 6.15 커밋 단계의 마지막에 릴리스 후보를 아티팩트 저장소에 등록한다. 이 책에서는 컨테이너 이미지를 컨테이너 저장소로 등록한다.

6.4.2 깃허브 액션을 통한 컨테이너 이미지 등록

깃허브 액션은 깃허브 저장소에서 직접 소프웨어 워크플로를 자동화하는데 사용할 수 있는 엔진이다.

카탈로그 서비스 프로젝트에서 커밋 단계에 대한 워크플로를 정의한 YAML파일에 환경 변수를 정의한다.

컨테이너 저장소로 인증하고 릴리스 후보 이미지를 푸쉬한다.