

Docker java udemy course

1장 : Docker 소개

▼ 도커 명령어들

Manage images

`docker image pull`

(Chapter 4): download an image from DockerHub

`docker image ls` Chapter 5

(Chapter 5): list all local images

`docker image build -t`

.

(Chapter 6): build an image with a tag (note the dot!)

`docker image push`

(Chapter 9): publish an image to dockerhub

`docker image tag`

(Chapter 9): tag an image - either alias an existing image or apply a :tag to one

Manage Containers

`docker container run -p :`

(Chapter 4): run a container from an image, publishing the specified ports

`docker container ls -a`

(Chapter 4): list all containers, even the stopped ones

`docker container stop`

(Chapter 4) stop a running container

`docker container start`

(Chapter 4) restart a stopped container

`docker container rm`

(Chapter 4) remove a stopped container

Docker for Java Developers - Quick Command Reference 2

Manage Containers (ctd)

`docker container prune`

(Chapter 4) remove all stopped containers

`docker container run -it`

(Chapter 5): run a container with interactive terminal

`docker container run -d`

(Chapter 5): run a container detached (or in a daemon like way)

`docker container exec -it`

(Chapter 5): run a command in a container

`docker container exec -it bash`

(Chapter 5): special form of the above, runs a bash shell, connected to your local terminal (your distro needs to have bash, alpine will require /bin/sh)

`docker container logs -f`

(Chapter 5) Follow the log (STDIN/System.out) of the container

`docker container commit -a "author"`

(Chapter 6) Take a snapshot image of a container

Manage your (local) Virtual Machine

`docker-machine ip`

(Chapter 4): Find the IP address of your VirtualMachine, required for Docker

Toolbox users only

Docker for Java Developers - Quick Command Reference 3

Manage Networks

`docker network ls`

(Chapter 10): list all networks

`docker network create`

(Chapter 10): create a network using the bridge driver

Manage Volumes

`docker volume ls`

(Chapter 11): list all volumes

`docker volume prune`

(Chapter 11): delete all volumes that are not currently mounted to a container

`docker volume inspect`

(Chapter 11): inspect a volume (can find out the mount point, the location of the volume on the host system)

`docker volume rm`

(Chapter 11): remove a volume

Docker Compose

`docker-compose up`

(Chapter 13): process the default `docker-compose.yml` file, starting any containers as required. If containers are already running they are ignored, meaning this command also serves as a "redploy".

`docker-compose up -d`

(Chapter 13): run containers in the detached state. Note the order of the command line arguments!

`docker-compose logs -f`

(Chapter 13): follow the log for the specified service. Omit the `-f` to tail the log.

`docker-compose down`

(Chapter 13): stop all the containers (services) listed in the default compose file.

Docker for Java Developers - Quick Command Reference 4

Manage a Swarm

`docker swarm init (--advertise-addr)`

(Chapter 14): Switch the machine into Swarm mode. We didn't cover how to stop swarm mode: `docker swarm leave --force`

`docker service create`

(Chapter 14): Start a service in the swarm. The args are largely the same as those you will have used in docker container run.

`docker network create --driver overlay`

(Chapter 14): Create a network suitable for using in a swarm.

`docker service ls`

(Chapter 14): List all services

`docker node ls`

(Chapter 14): List all nodes in the swarm

`docker service logs -f`

(Chapter 14): Follow the log for the service. This feature is a new feature in Docker and may not be available on your version (especially if using Linux Repository Packages).

`docker service ps`

(Chapter 15): List full details of the service - in particular the node on which it is running and any previous failed containers from the service.

`docker swarm join-token`

(Chapter 16): Get a join token to enable a new node to connect to the swarm, either as a worker or manager.

Manage Stacks

`docker stack ls`

(Chapter 15): list all stacks on this swarm.

`docker stack deploy -c`

(Chapter 15): deploy (or re-deploy) a stack based on a standard compose file.

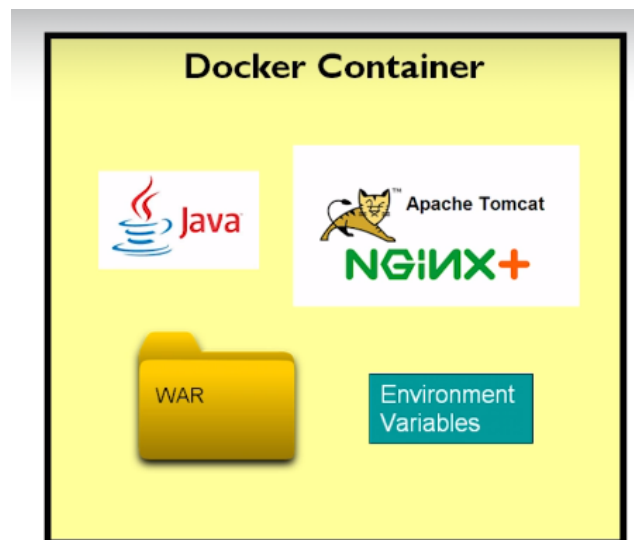
`docker stack rm`

(Chapter 15): delete a stack and its corresponding services/networks/etc.

jar 파일 , war 파일이 최종 결과물로 나온다.

격리되어 있는 앱으로 실행되는 jar 파일을 배포하지 않고 컨테이너를 실행하는 것.

컨테이너 = 완전하고 독립적인 환경

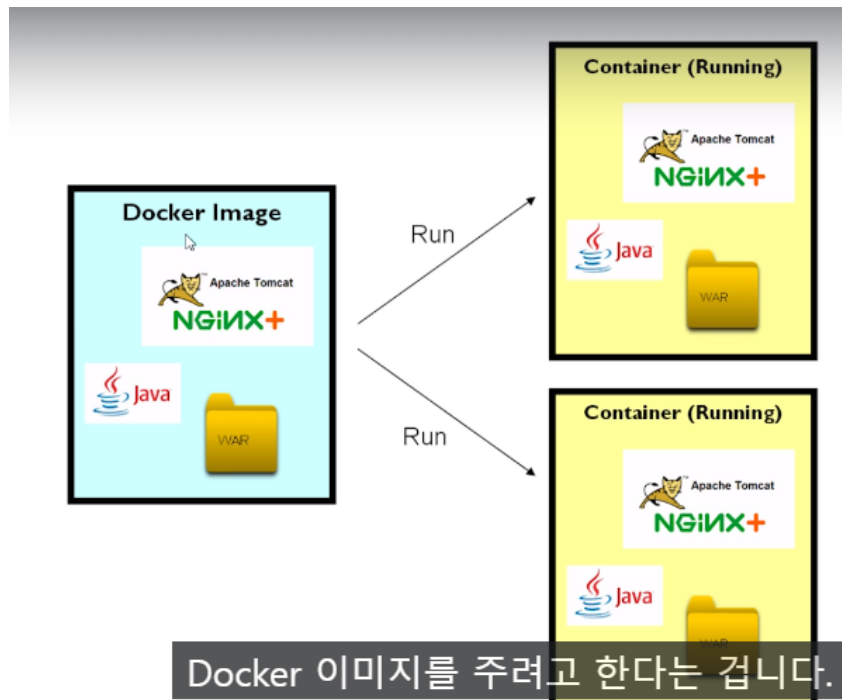


Docker 컨테이너를 정의하려면 먼저 이미지를 구축해야한다.

이미지는 도커에서 컨테이너의 정의이다. 이미지에는 전체 환경의 정의가 포함된다. 환경 변수, war 파일, jar 파일, Java JDK, Tomcat 등.

실제 이미지 한개가 배포 단위가 된다.

핵심: 컨테이너는 이미지의 인스턴스이며, 간단히 실행하여 이미지 중 하나를 인스턴스화 하는것 , war 파일을 주는게 아니라 Docker 이미지를 주는것!

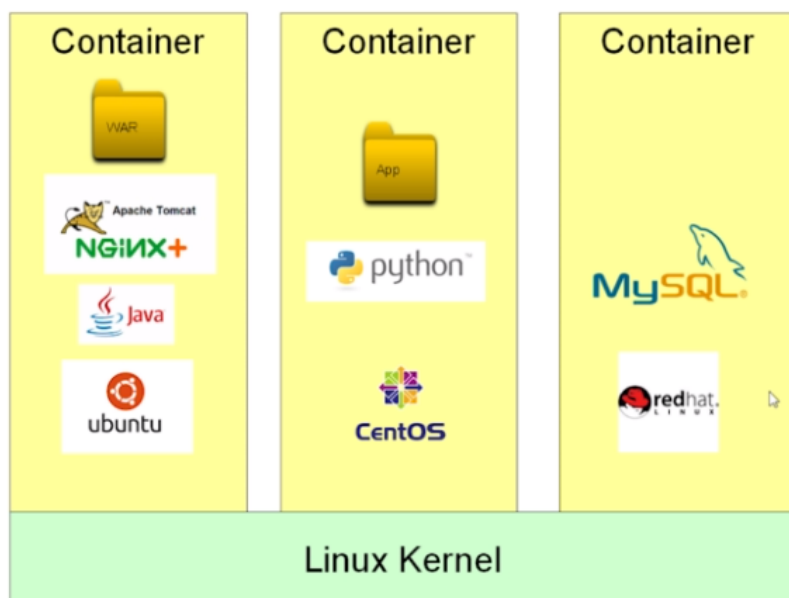


도커 이미지에는 추가 구성이 필요하지 않은 완전한 환경이 포함된다.

컨테이너와 가상머신의 차이?

컨테이너는 전체 운영 시스템을 포함하지 않으므로 훨씬 가볍고 효율적

리눅스가 효율적. (컨테이너를 실행하면, 호스트 운영 체제 커널에서 컨테이너가 실행된다.)



우분투 센토스 레드햇 리눅스는 운영체제가 아니라 애플리케이션 및 툴의 집합이다. 컨테이너들은 같은 리눅스 커널을 공유한다.

컨테이너 기술은 2008년 linux container로 등장, 도커는 현재 다른 스키마와 전략을 채택함. 윈도우에서도 Docker를 사용할 수 있다.

▼ BIOS 에서 가상화 지원 실행.

BIOS에서 가상화 지원 실행

Mac 또는 Windows를 사용하는 경우, Linux 가상 머신을 백그라운드에서 실행하기 위해 Virtualization에 의존하는 Docker 설치를 사용하게 됩니다.

이 기능을 사용하려면 컴퓨터에서 가상화에 대한 하드웨어 지원이 있어야 하며 이 기능을 사용하도록 설정해야 합니다. 내가 아는 한, 대부분의 최신 컴퓨터에는 이 지원 기능이 내장되어 있지만 제조업체에서 기본적으로 이 기능을 사용하지 않도록 설정하는 것이 일반적입니다.

따라서 시스템 BIOS에서 가상화를 수동으로 켜야 할 가능성이 높습니다.

가상화를 사용하도록 설정하는 방법

주로 BIOS마다 다르기 때문에 이 프로세스를 비디오로 만들 수 없습니다. 따라서 시스템 제조업체의 핸드북을 사용하여 BIOS에 액세스하는 방법과 옵션을 찾을 수 있는 위치를 확인해야 합니다.

BIOS에 들어가면 다음과 같은 기능을 찾을 수 있습니다:

꼭 이렇게 해야 할까요?

가상화가 이미 활성화되어 있을 수 있으므로 다음 비디오에서처럼 Docker를 설치하고 실행하는 것이 좋습니다. 그런 다음 오류가 표시되면:

"이 컴퓨터에는 VT-X/AMD-v가 설정되어 있지 않습니다. BIOS에서 활성화하는 것은 필수입니다

그러면 여기로 돌아와서 BIOS에 들어가야 합니다. Docker에는 이 옵션을 활성화했더라도 오류가 발생할 수 있는 버그가 있습니다. 이 경우 "Docker Toolbox 설치"의 비디오를 확인하십시오. 이 버그는 AMD 버그이므로 쉽게 해결할 수 있습니다.



3장 4장 도커 설치와 배포 시나리오

도커 이미지는 훨씬 쉽게 배포할 수 있는 장점이 있다.

도커 이미지를 구축하고 컨테이너를 실행하는 의미는 다른 기기에 쉽게 배포할 수 있어야 한다.

즉, 최종 목표는 로컬에서 개발하고 테스트한 것과 동일한 이미지를 배포할 수 있어야 한다. Docker로 작업하는 데는 두 가지 측면이 있다. 각자의 개발 환경인 로컬 도커와 함께, 대상 환경이 있다. 클라우드, 공급자, 서버 등이다. 대상 환경에서도 Docker를 설치하고 실행해야 한다. 이부분은 강의 뒷부분.

실제로 컨테이너는 리눅스에서 실행되는 프로세스일 뿐이다, 윈도우 커널이 아님. 개발할때 리눅스 외부에서 도커를 실행할 수 있어야 한다.

컴퓨터에 가상머신을 갖추는 것이 윈도우에서 사용할 수 있는 해결 방법이다.

가상 머신안에 Docker가 설치되어 있어야 하고, 설정을 거쳐서 사용한다.

한편, 이전에 Windows 10 Home에 필요했던 Docker Toolbox는 더 이상 사용할 수 없습니다. 그러나 이제 Windows 10 Home 사용자는 Docker Desktop을 설치할 수 있습니다.

따라서 Windows 10 Home을 사용하는 경우 Docker Desktop에 대한 강의를 따르고 Docker Toolbox를 무시하십시오!

Windows 10 Pro or Enterprise → Docker Desktop을 사용할 수 있음.

Docker Desktop 은 사실상 가상화된 Linux 머신이다.

Docker desktop 설치 후 powershell 입력.

```
docker container run hello-world
```

개발자와 운영자가 있다. Image → virtualpairprogrammers/fleetman-webapp 같은 이미지

dockerfile → 일종의 파일링크 두가지를 개발자가 제공.

Tomcat app , db가 필요, war파일이 필요.

```
docker image pull virtualpairprogrammers/fleetman-webapp
```

```
- docker container run -p 8080:8080 virtualpairprogrammers/fleetman-webapp
-p (publishing) , 앞 8080 외부 노출 포트 , : 뒤 8080 컨테이너 내부 포트
브라우저에서 Ip주소:8080을 입력하면, 트래픽이 컨테이너 내부의 톰캣포트로 연결된다.

- docker container ls
실행중인 컨테이너 목록 확인 가능. ctrl+c를 해도 컨테이너는 종료되지 않음

- docker container stop (container ID)
* container ID는 전부 입력하지 않고 부분까지만 입력해도 된다.
```

-p를 입력하지 않으면 8080 포트가 외부에 드러나지 않는다.

-p 80:8080 하면 포트 숫자가 전혀 필요하지 않고 로컬호스트만 입력해서 접근 가능.

ec2에서 실행해도 과정은 똑같음, 이미지를 미리 받아오지 않아도 로컬에서 확인 후 없으면 자동으로 도커허브에서 받아온다.

5장: 컨테이너 관리

이미지의 또 다른 용도는 이미지를 받아서 확장할 수 있음. 실제로는 수정할 수는 없음.

자신의 새로운 이미지를 만드는 방법으로만 확장

베이스 이미지 찾기

virtualpairprogrammers/fleetman-webapp 에서 슬래시 앞은 소유자, 뒤는 이름.

docker에서 직접 만든 공식 이미지, 안전하고, 바이러스 없음. official 이고 소유자가 없음.

dockerhub에서 필요한 베이스 이미지를 검색으로 찾기 ex) ubuntu.

기존과 현재의 Docker 커맨드

```
docker pull ubuntu
docker image pull ubuntu 같은 효과

docker ps : 실행중인 컨테이너 보기 (이전 커맨드)
docker ps -a : 중지된 것까지 포함해 모든 컨테이너 보기

docker container ls --> 현재 커맨드
docker container ls -a

docker image ls
docker --> 모든 커맨드 목록 보기.
docker container --help
docker image --help
```

기존의 커맨드 : Docker + pull 등의 명령어 + 필요한 매개변수

베이스 컨테이너 실행

ubuntu는 서버가 아니기 때문에 포트를 열 필요가 없음.

```
docker container run ubuntu
docker container ls (아무것도 보이지 않음)
docker container ls -a (bash) 스크립트가 없다면 즉시 종료된다.
docker container run -it ubuntu (== -i -t , -i는 대화형, t는 컨테이너를 터미널에 연결함 의미)
컨테이너를 대화형으로 실행한다는 뜻

linux bash가 실행된다. ls 등의 리눅스 명령어 사용 가능.
apt-get update && apt-get install git 처럼 컨테이너 내부에 것 설치도 가능.
exit 명령어로 명령줄 종료.
```

이미지를 가져오는 또다른 방법은 위처럼 내부에서 bash 셸을 실행하는 방법이다.

컨테이너 생명주기

예제 컨테이너가 길게 살아있는 이유 ? catalina.sh 커맨드이기 때문!

```
- docker container start a1 (ID) --> 백그라운드에서 실행(콘솔에 로깅 나타나지 않음)
- docker container rm a7 --> 컨테이너 제거 가능
- docker container prune --> 중단된 컨테이너 모두 제거!
- docker container run -d -p 8080:8080 virtualpairprogrammers/fleetman-webapp
  -d는 분리된(detached)의미, daemon처럼 사용가능( 백그라운드 실행 의미)

- docker container logs 20 --> 백그라운드에서 실행한 컨테이너 로그 확인 가능
- docker container logs -f 20 --> 추가 로깅이 있다면 대화형으로 확인 가능
```

디버깅 수행 또는 컨테이너에 특정 패키지 설치 확인하는 목적의 SSH는 필요 없음.
- docker container exec EE(ID) bash(command) --> 컨테이너 내부에서 커맨드 실행 가능.
- docker container exec -it ee bash --> 대화형으로 컨테이너 내부의 bash 셸에 접근!!
디버깅을 해야할 때 유용함.

```
docker container run -d -p 80:8080 Tomcat --> 현재 최신 latest 버전 지정 이미지 다운로드
--> 예러 가능성 때문에 권장하지 않음.
docker container run -d -p 80:8080 Tomcat:8.5.16-jre8-domcat
--> 이처럼 이미지 버전 명시하여 정확히 동일한 결과 얻는 것을 권장.
```

실전: Tomcat 컨테이너 실행

Tag --> 이미지의 또다른 버전

Alpine --> 아주작은 단위의 Linux 배포판 tomcat은 이제 alpine 사용 x

```
docker container run -d -p 80:8080 Tomcat:8.5.16-jre8-domcat
```



tomcat

DOCKER OFFICIAL IMAGE · 500M+ · 3.5K

Apache Tomcat is an open source implementation of the Java Servlet and JavaServer Pages technologies

docker pull tomcat

Overview

Tags

Sort by

Newest

Filter Tags

TAG

[8.5.85-jre8-temurin-jammy](#) ✓

Last pushed 3 days ago by [doijanky](#)

docker pull tomcat:8.5.85-jr...

DIGEST

[d8cc07b7dc63](#)

[2ad9ef93a709](#)

[8d84cb15ad72](#)

[2437f31df4e8](#)

OS/ARCH

linux/amd64

linux/arm/v7

linux/arm64/v8

linux/ppc64le

COMPRESSED SIZE

91.9 MB

85.97 MB

88.95 MB

97.19 MB

```
docker image pull tomcat
docker container run -d -p tomcat -- (이미 80 포트가 사용중이라면 에러 발생)

docker container ls -a
docker container stop ee
docker container run -d -p tomcat
docker container logs -f c0
```

6장 : Dockerfile

“commit” 으로 이미지 구축

이미지를 만드는 두가지 방법:

- 실행중인 컨테이너의 스냅샷으로 만들기

```
docker container run -it ubuntu --> 컨테이너 내부에서 직접 작업.
(컨테이너 중지하고 제거하면 사라지지만 스냅샷 생성이 가능함.)
apt-get update
apt-cache search jdk
apt-get install -y openjdk-8-jdk
clear
javac

docker container commit -a "kks@gmail.com" 5f(ID) myjdkimage
docker image ls
docker container run -it myjdkimage
```


Dockerfiles

```
FROM ubuntu:latest

MAINTAINER Richard Chesterwood "contact@virtualpairprogrammers.com"

RUN apt-get update && apt-get install -y openjdk-8-jdk

CMD ["/bin/bash"]
```

거의 대부분 Dockerfile을 대신 사용해 이미지를 생성한다.

Dockerfile 은 이미지 구축 방법을 설명하는 텍스트 파일이다.

maintainer → 작성자

Cmd → 컨테이너를 실행할 때 기본적으로 수행할 커맨드

```
docker image build -t jdk-image-from-dockerfile .
(-t로 이미지에 태그 지정 가능)
마지막에 .으로 마무리하는 걸 잊으면 안된다.
(.은 Dockerfile의 빌드 컨텍스트를 의미함)
docker는 중간 레이어를 여러개로 두고 실행되어
캐시로 필요한 이미지를 가져올 수 있어서 편하고 효율적이다.

docker run container run -it jdk-image-from-dockerfile
```

이미지에 파일 복사

```
java -jar test-program.jar

두번째 매개변수는 컨테이너 내부의 파일시스템에서 저장하려는 폴더를 참조

docker run container run -it jdk-image-from-dockerfile
pwd
ls
opt 폴더-> 선택적 소프트웨어용
사용자폴더,usr 읽기전용 폴더 등 다양한 곳에 jar 파일 놓을 수 있음.
java 프로그램을 영구적으로 컨테이너에 설치하는것이므로 usr폴더가 적당함.
```

```
-----
FROM ubuntu:latest

MAINTAINER Richard Chesterwood "contact@virtualpairprogrammers.com"

RUN apt-get update && apt-get install -y openjdk-8-jdk

[COPY test-program.jar /usr/local/bin/]
또는
[WORKDIR /usr/local/bin/ (저장 폴더 지정)

ADD test-program.jar . ]
-----
```

수정 후 다시

```
docker image build -t jdk-image-from-dockerfile .
docker run container run -it jdk-image-from-dockerfile
```

* 이 폴더와 모든 하위 폴더에 있는 파일이 Dockerfile에 표시된다. /로 운영체제의 다른 폴더를 참조해도 소용 없음.

이미지 커맨드(CMD)

컨테이너가 시작될 때 프로그램이 자동 실행되게 하려고 한다. →CMD 커맨드 사용
컨테이너에서 단일서비스를 구현하는 건 아주 일반적인 모범사례임.

```
CMD["java" , "-jar" , "test-program.jar"] 추가 --> 바로 실행된다.  
java 실행, 명령행 인자 -jar , test-program.jar 인자. 커맨드를 지정하는 기본형식  
  
docker container start ae  
docker container logs -f ae  
  
log4j를 쓰지 않아도 System.out도 로그로 표시해준다.  
Log4j를 쓸때도 표준출력을 출력하도록 구성해야 한다.  
  
이 때 수동으로 꺼야 container가 종료된다.  
  
ENTRYPOINT ["java", "-jar", "test-program.jar"]
```

COPY와 Add 비교

ADD 커맨드는 url로 작동 가능 , 아카이브의 압축을 풀수도 있음.
docker 는 COPY를 추천함.

ENTRYPOINT 는 CMD와 유사함.

ENTRYPOINT는 항상 실행되지만 CMD는 기본값일 뿐이다.

CMD["java" , "-jar" , "test-program.jar"]
docker container run -it jdk-image-from-dockerfile /bin/bash
(CMD로 하는 경우) --> 사용자가 원하는 cmd 명령어를 재정의 할 수 있음.

ENTRYPOINT ["java", "-jar", "test-program.jar"]
반면에 ENTRYPOINT는 하드 코딩된 커맨드이다.

Label과 MAINTAINER 비교

MAINTAINER 필드는 공식적으로 사용하지 말 것 권장

Dockerfile LABEL --> 키 값 쌍 사용 (key value)
LABEL maintainer="Richard Chesterwood"
LABEL creationdate="19 November 2019"

빌드 다시하면, LABEL부터 중간레이어가 실행되어서 오래 걸림 (자바 다시 깔림)

라벨로 메타데이터를 제공하면, 사람들의 이해에 도움이 된다.

표준 label schema
키 값쌍에 표준을 제공.
created, authors, url , documentation,source,version등의 필드 명시.
실제로 라벨 사용을 별로 하진 않음.

```
LABEL \n  org.opencontainers.image.authors="Tobias Maier <tobias.maier@baucloud.com>" \n  org.opencontainers.image.description="This docker image installs docker-compose on top of the\n  org.opencontainers.image.licenses="MIT" \n  org.opencontainers.image.source="https://github.com/tmaier/docker-compose" \n  org.opencontainers.image.title="Docker Compose on docker base image" \n  org.opencontainers.image.vendor="BauCloud GmbH" \n  org.opencontainers.image.version="${DOCKER_VERSION} with docker-compose ${COMPOSE_VERSION}"
```

아직 정확히 사용해야하는 규칙이 정해지지 않았음. (maintainer, label 등) 선택사항임.

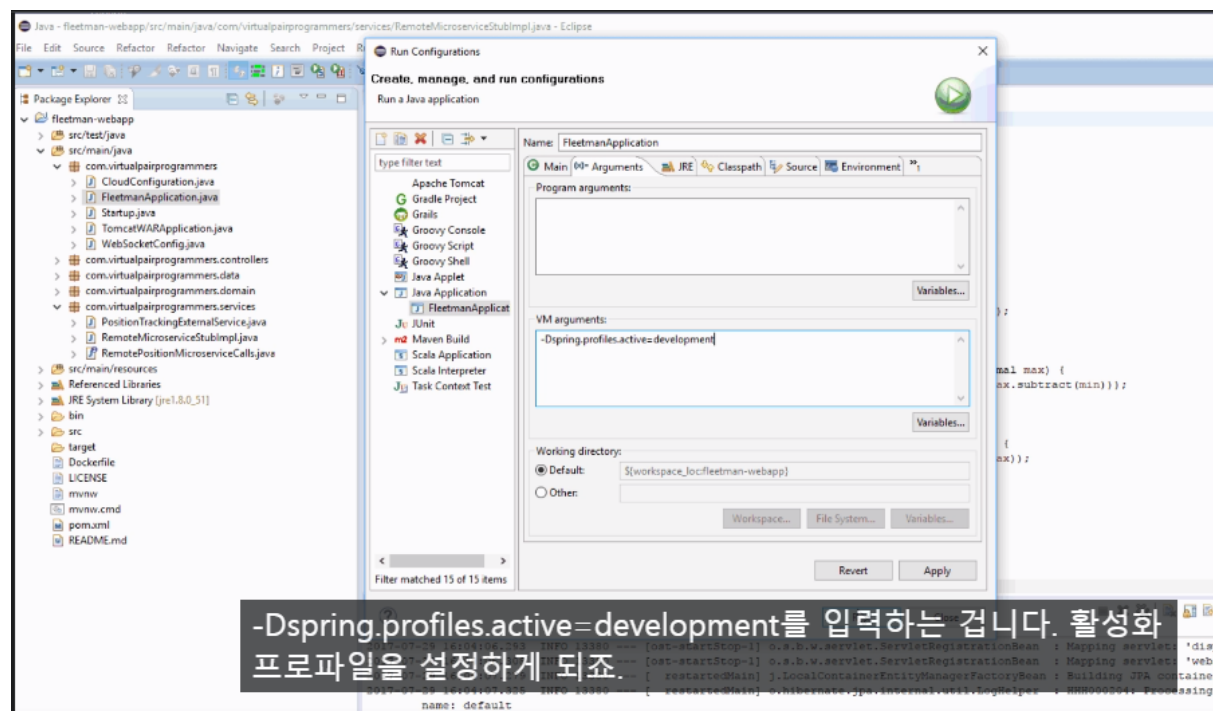
7장 Tomcat 애플리케이션

IDE 설정

7장 예제파일이 들어있는 경로에서 이클립스 실행, —> new java project—>이름 똑같이 만들기
pom.xml 우클릭 —> run as —> maven build —> goal eclipse:eclipse —> 빌드 후 새로고침.

src/main/java /services StubImpl —> @Profile({"development","docker-demo"})

중요한 것은 Docker에서 모든 마이크로서비스를 통합함.



run as —> run configuration...—> java application —> arguments에 입력하고 실행.

기본적으로 부트 프로젝트는 war파일 필요x, tomcat도 내장임.

따라서 SpringBootServletInitializer를 확장한 앱 클래스 설정.

```

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <!-- For standalone JAR file <configuration><executable>false</executable></configuration> -->
      <configuration>
        <mainClass>com.virtualpairprogrammers.TomcatWARApplication</mainClass>
      </configuration>
    </plugin>
  </plugins>
</build>

```

pom.xml에 추가함.

war 파일 생성법.

pom.xml 우클릭 maven build →goals에 package →Run

C:\SSAFY\workspace\docker\Chapter+7+v2\Chapter 7\fleetman-webapp\target 에 war 파일 생성

Docker 베이스 이미지 찾는 방법

도커파일 구성

1. 베이스 이미지 결정
2. maintainer or label 구성
3. 컨테이너 기본 커맨드

Tomcat은 세계적으로 Java 애플리케이션에 사용되고 있다.

이미지는 삭제되지 않아서 도커허브에서 점점 쌓이게 된다.

슬림버전은 가볍고 실행만 하기 좋고 일반버전은 리눅스 배포판을 완전히 가지고 있다.

별칭이 존재하기 때문에 가장 구체적으로 명시된 버전을 사용하는게 좋은 엔지니어링이다.

태그를 지정안하면 latest가 붙은 버전을 받는다.

```

1 FROM tomcat:8.5.47-jdk8-openjdk
2
3 MAINTAINER "Richard Chesterwood"
4
5 # Transfer our WAR
6
7 CMD ["catalina.sh", "run"]

```

Docker에서 RUN 커맨드 설정 방법

CMD를 사용하지 않으면 default값을 사용하는데 tomcat은 catalin.sh , run이다. 하지만 CMD를 명시적으로 사용하는 것이 사용자에게 더 좋을 것이다.

Dockerfile이 있는 곳에서 powershell , docker image build -t fleetman-webapp .

sending build context to Docker daemon → 전체 폴더가 압축되어 Docker daemon으로 전송.

Docker daemon 이란 백그라운드에서 실행중인 Docker 엔진, 대부분 가상머신에서 실행

docker container run -p 8080:8080 -it fleetman-webapp

Docker EXPOSE 커맨드의 역할

톰캣이 아닌 다른 서버나 프레임워크 포트번호를 알아내는 방법?

도커파일 따라가서 , EXPOSE 8080을 볼 수 있다. EXPOSE 커맨드는 설명을 위한 역할이다.

Tomcat 기본 애플리케이션 제거

기본으로 제공되는 hello world같은 war파일을 제거해야 함.

```
docker container run -p 80:8080 -d fleetman-webapp
-it 대신 -d로 분리모드로 실행. -->백그라운드 실행

docker container logs 48(id)

docker container exec -it 48(id) bash

docker container exec -it 48 sh (좀더 기능이 적은 명령줄)

WORKDIR 커맨드에서 기본 파일 경로 설정하기 때문에 기본 디렉토리 확인 가능.

cd webapps cd ROOT index.jsp 등의 파일 확인 가능.

docker container stop 48

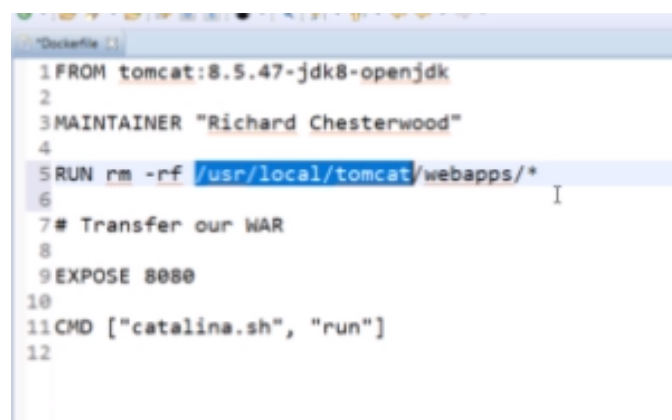
RUN 커맨드로 LINUX 커맨드 사용 가능.
RUN rm -rf /usr/local/tomcat/webapps/* webapps안의 폴더 전부 삭제

docker image build -t fleetman-webapp .

docker container run -d -p 80:8080 fleetman-webapp

docker container exec -it 18 bash
--> webapps 폴더가 깨끗해진것 확인 가능.
```

```
ENV CATALINA_HOME /usr/local/tomcat
ENV PATH $CATALINA_HOME/bin:$PATH
RUN mkdir -p "$CATALINA_HOME"
WORKDIR $CATALINA_HOME
```



```
1 FROM tomcat:8.5.47-jdk8-openjdk
2
3 MAINTAINER "Richard Chesterwood"
4
5 RUN rm -rf /usr/local/tomcat/webapps/*
6
7 # Transfer our WAR
8
9 EXPOSE 8080
10
11 CMD ["catalina.sh", "run"]
12
```

WAR 파일 설치

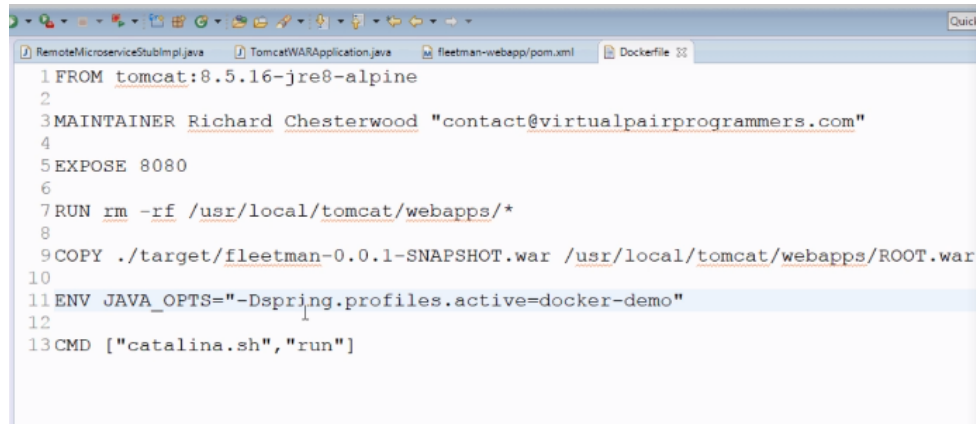
```
COPY ./target/fleetman-0.0.1-SNAPSHOT.war /usr/local/tomcat/webapps/ROOT.war 추가
```

```
docker image build -t fleetman-webapp .
```

```
docker container run -p 80:8080 -it fleetman-webapp
```

docker-demo에서 실행 가능 하므로 환경변수를 java에 전달해야함.

```
ENV JAVA_OPTS = "-Dspring.profiles.active=docker-demo"
```



```
1 FROM tomcat:8.5.16-jre8-alpine
2
3 MAINTAINER Richard Chesterwood "contact@virtualpairprogrammers.com"
4
5 EXPOSE 8080
6
7 RUN rm -rf /usr/local/tomcat/webapps/*
8
9 COPY ./target/fleetman-0.0.1-SNAPSHOT.war /usr/local/tomcat/webapps/ROOT.war
10
11 ENV JAVA_OPTS="-Dspring.profiles.active=docker-demo"
12
13 CMD ["catalina.sh", "run"]
```

8장 스프링부트 어플리케이션

war 파일은 jsp를 사용할때 씬. 보통은 jar 파일.

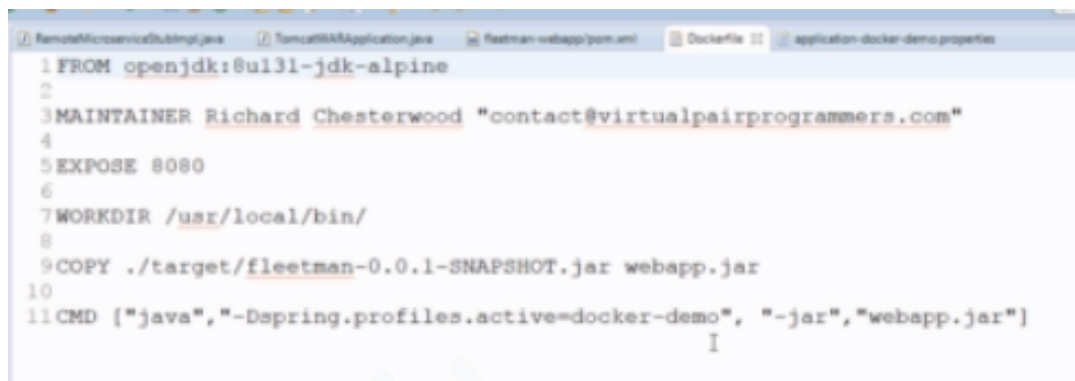
pom.xml packaging에서 war를jar로 변경.

maven build →goal ⇒ package

```
java -D"spring.profiles.active"=development -jar ./fleetman-0.0.1-SNAPSHOT.jar
```

```
docker image build -t fleetman-webapp .
```

```
docker container run -p 80:8080 -it fleetman-webapp
```



```
1 FROM openjdk:8u131-jdk-alpine
2
3 MAINTAINER Richard Chesterwood "contact@virtualpairprogrammers.com"
4
5 EXPOSE 8080
6
7 WORKDIR /usr/local/bin/
8
9 COPY ./target/fleetman-0.0.1-SNAPSHOT.jar webapp.jar
10
11 CMD ["java", "-Dspring.profiles.active=docker-demo", "-jar", "webapp.jar"]
```

톰캣을 설치하지 않고 네이티브 Spring Boot 애플리케이션을 실행

9장 Docker Hub 에 push 하기

```
docker image ls
docker login 으로 도커허브에 로그인 가능.

docker image push fleetman-webapp --> error 소유자/이미지 이름이 필요함.

docker image tag 05c(id) virtualpairprogrammers/test

docker image ls --> 별칭 alias처럼 가능.

docker image push virtualpairprmgammers/test
```

10장 Docker Networking

컨테이너에 추가 서비스를 설치하는 방법이 뭘까?

Docker의 주요 개념중 하나 컨테이너가 하나의 서비스만을 공개하도록 되어 있는 것.

예제의 경우 Springboot 어플리케이션이고, 서비스는 실행.

슈퍼바이징 서비스가 여러개의 서비스를 쓸 수 있도록 도와줌. 하지만 추천하지 않음.

컨테이너에서 여러 서비스를 쓰면 컨테이너의 상태 관리가 힘들어짐.

각 컨테이너를 독립적인 서비스로 생각하는게 훨씬 낫다.

마이크로 서비스가 한가지만 수행하는 것처럼 컨테이너는 하나의 서비스만 클라이언트에 노출해야 한다.

기술적으로는 서비스 여러개를 한 컨테이너에서 쓸 수 있지만 꼭 필요한지 고민해야함.

두 컨테이너를 네트워크로 연결하는 것은 아주 쉽다. 이것을 할줄 알아야 한다.

컨테이너는 자체적인 사설 네트워크 안에 있다.

컨테이너 이름 및 DNS

```
docker container run -e MYSQL_ROOT_PASSWORD=password -d mysql:5
```

```
docker container exec -it 4f bash
```

```
mysql -uroot -ppassword
```

show databases;

다시 보면 스키마가 자동 생성 됨.

docker에서 컨테이너는 이름을 가져와서 해당 컨테이너 이름을 각 컨테이너가 사용하고 있는 DNS 서비스에 추가함.

다른 컨테이너 이름을 도메인 이름에 추가하여 쉽게 연결 가능.

```
docker container run -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=fleetman -d mysql:5

docker container run -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=fleetman -d mysql:5

docker container run -d -p 80:8080 -it fleetman-webapp

docker container exec -it dbe sh (slime, alpine은 bash 없음)

ping google.com

ping (name) ?? --> bad address 오류 ! 발생
```

네트워크 관리

```
docker network ls

default - bridge 기본 네트워크, 모든 컨테이너 연결, 트래픽 바깥 ,
아웃바운드 트래픽 허용하는 NAT처럼 작동

docker network create my-network

docker network ls

docker container run --network my-network --name database -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=fleetman -d mysql:5

docker container run -d -p 80:8080 --network my-network --name fleetman-webapp fleetman-webapp

docker container exec -it fleetman-webapp sh

ping google.com

ping database

spring.datasource.driver-class-name=com.mysql.jdbc.Driver 추가해야함 !!
```

정정: 추가 구성이 필요한 영상

최근에 샘플 프로젝트를 업그레이드하여 Spring Boot 2를 사용하여 버전 9 이후의 최신 JDK에서 코드를 컴파일할 수 있습니다.

한 가지 사소한 문제는 이제 스프링 부트 2가 기본적으로 다른 연결 풀 구현을 사용한다는 것입니다. 과정에서 사용 중인 MySQL5와 이전 버전과의 호환성을 유지하려면 application-docker-demo.properties 파일에 속성을 추가해야 합니다:

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

사용자 이름, 비밀번호 등 뒤에 이것을 추가하세요. 연결 문제가 발생하면 이 줄이 올바른지 다시 확인하십시오!

데이터베이스 컨테이너에 연결

```
spring.datasource.url=jdbc:mysql://database:3306/fleetman
spring.datasource.username=root
```



```

spring.datasource.password=password

-----

docker container stop fleetman-webapp
docker image build -t fleetman-webapp .

docker container rm fleetman-webapp
docker container run -d -p 80:8080 --network my-network --name fleetman-webapp fleetman-webapp

docker container run -d -p 80:8080 --network my-network --name fleetman-webapp --rm fleetman-webapp
뒤에 rm붙여주면 컨테이너 중지시 이름 지워줘서 편함.

docker container logs -f fleetman-webapp

docker container run -it --network my-network alpine (세번째 컨테이너 설치)

ping database
ping fleetman-webapp

alpine에는 ping이 있고, 우분투에는 없음.

모든 배포판에는 적합한 패키지 관리 시스템이 있다. 다양한 패키지 관리시스템을 파악하는게 좋다.

Alpine은 apk이다.

apk add --no-cache mysql-client

mysql -uroot -ppassword -hdatabase

show databases;

use fleetman;

show tables;

select * from vehicle;

```

네트워크 토폴로지

비공개 네트워크를 쉽게 만들 수 있으며, 네트워크에 있는 컨테이너끼리 이름으로 쉽게 확인 가능.

시스템에 여러 네트워크가 있어도 괜찮다.

시작할때 컨테이너를 연결할 네트워크를 정할 수 있다.

11장 볼륨

볼륨 소개

지금까지 컨테이너 안에 있는 모든 데이터는 사라지는 것으로 가정했다.

어떠한 데이터를 유지해야 하는 경우도 있다. 어떤 데이터는 컨테이너보다 오래 남아있기를 바란다.

```

docker container run -d -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=fleetman mysql:5

docker container exec -it 07 bash

mysql -ppassword

```

```
show databaes;

use fleetman;

create table temp (temp varchar(255));

exit

docker container ls docker container stop id

mysql dockerfile을 보면 VOLUME /var/lib/mysql 이 있다.

* 컨테이너를 죽여도 데이터는 호스트 컴퓨터에 남는다.
폴더의 컨텐츠는 사실상 데이터베이스다.

docker container ls

docker container inspect id

제이슨 형태의 문서가 많다. Mounts 안에 name을 기억해두기.

호스트 파일 시스템에 마운트 되어 있다.

볼륨이 컨테이너 외부에 안전하게 저장된다는게 중요하다.

docker volume ls
```

볼륨 이름

볼륨에 이름을 지정하여 제어할 수 있는 방법을 쓰자.

```
백지에서 시작.
docker container run -v mydata:/var/lib/mysql -d -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=fleetman mysql:5

docker volume inspect mydata

docker container exec -it id bash

mysql -ppassword

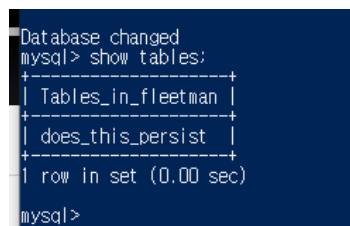
use fleetman;

create table does_this_persist(dummy varchar(255));
exit exit

docker container stop id
docker container rm id

docker container volume ls --> mydata 남아 있음.

docker container run -v mydata:/var/lib/mysql -d -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=fleetman mysql:5
다시 하면.
data가 남아있는 것을 확인 할 수 있다.
```



```
Database changed
mysql> show tables;
+-----+
| Tables_in_fleetman |
+-----+
| does_this_persist |
+-----+
1 row in set (0.00 sec)

mysql>
```

마운트 포인트

이 볼륨 폴더를 관리해야 할 수도 있다. /가 있어서 volume 이름이 아니라는 것을 알 수있다.

```
docker container run -v //c/users/work/mydatabase:/var/lib/mysql -d -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=fleetman mysql:5
```

12장 Maven

Fabric8 Docker Maven 플러그인 (DMP)

2017년 Maven 및 Docker 용으로 가장 인기 있는 플러그인은 fabric8프로젝트이다.

RedtHat과 관계 있으며, 클라우드 기반 시스템이다.

필요한 것은 전체 플러그인이 아니라 소규모의 docker-maven-plugin

`fabric8io/docker-maven-plugin`

Global configuration parameters specify overall behavior like the connection to the Docker host. The corresponding system properties which can be used to set it from the outside are given in parentheses. The docker-maven-plugin uses the Docker remote API so the URL of your Docker Daemon must somehow be specified.

<https://dmp.fabric8.io/>

pom.xml 아래 build 부분에 plugin 내용 추가하기.

Maven Plugin의 빌드 문제는 ?

▼ 글보기

Maven Plugin의 빌드 문제는?

Problems running the Maven Build?

If you're running Docker Toolbox (Mac or Windows before 10 Pro), then you may have problems either connecting to the Daemon, or you may get obscure errors saying:

"unable to find valid certification path to requested target"

The solution is to run on the command line:

`docker-machine env`

This will give an output like:

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="XXX"
export DOCKER_MACHINE_NAME="default"
export COMPOSE_CONVERT_WINDOWS_PATHS="true"
```

Now add the following two tags to the <configuration> tag of the Docker Maven plugin configuration, in the pom.xml file:

```
<dockerHost>tcp://192.168.99.100:2376</dockerHost>
<certPath>XXX</certPath>
```

Make sure the values in the tags match the output you received in the previous step. Full details at StackOverflow here: <https://stackoverflow.com/questions/46598198>

DMP구성

Docker는 네이티브 머신이나 가상머신에서 데몬 프로세스로 실행되고 있다.

window에서 하면 백그라운드에서 실행중인 가상머신이 있다.

Docker host를 구성하지 않아도 된다.

```

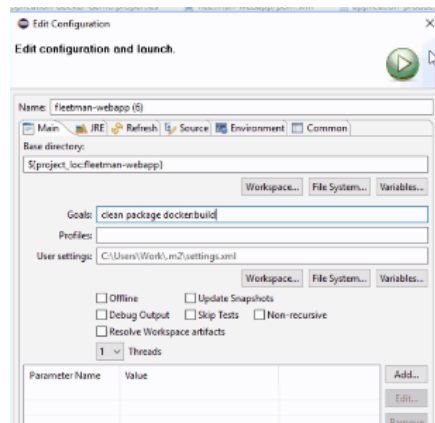
<!-- Needed if pushing to DockerHub: preferred to store these in local environment (see the course) -->
<authConfig>
  <username>YOUR-USERNAME</username>
  <password>YOUR-PASSWORD</password>
</authConfig>

```

docker image build -t fleetman-webapp .

→ .은 Dockerfile을 이 폴더에서 찾도록 Maven 에게 알리며, 이 폴더와 모든 하위 디렉토리에는 Docker에서 이미지를 빌드하는데 필요한 모든 데이터가 들어간다.

root폴더에 넣는건 파일 전체를 압축하기 때문에 비효율적. 이미지 빌드에 필요한건 Dockerfile과 jar 파일 뿐이다.



<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/98f17aa3-b6e4-4d41-bfeb-3258f073aed2/pom.zip>

Maven 생명 주기와 통합

푸시 및 배포 통합

Jenkins

13장 Docker 컴포즈

Docker 컴포즈란?

컴포즈 파일

시작순서

docker-compose 실행

배포 변경

14장 Swarm

오케스트레이션 시스템

Swarm 소개

Swarm 서비스

Play with Docker

서비스 로그 모니터링

15장 스택

Manager와 worker 비교

스택 구축

서비스 생명 주기

복제 서비스

라우팅 메시

Visualizer

롤링 업데이트

16장 EC2에서의 마이크로 서비스

Fleetman 마이크로 서비스 소개

EC2클러스터 시작

클러스터 구성

스택 배포

시스템 복구력

서비스 디스커버리