Hanbit RealTime 123

하루 만에 배우는 도커

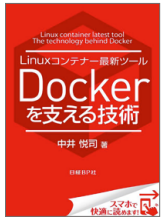
나카이 에츠지 지음
김성재 옮김

한빛미디어 Hanbit Media, Inc.

Hanbit RealTime 123

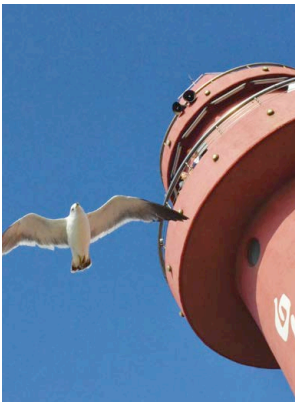
하루 만에 배우는 도커

나카이 에츠지 지음
김성재 옮김



이 도서는
Linuxコンテナー最新ツール
Dockerを支える技術(日経BP社)의
번역서입니다

한빛미디어
Hanbit Media, Inc.



표지 사진 **이성호**

이 책의 표지는 이성호 님이 보내 주신 풍경사진을 담았습니다.
리얼타임은 독자의 시선을 담은 풍경사진을 책 표지로 보여주고자 합니다.

사진 보내기 **ebookwriter@hanbit.co.kr**

하루 만에 배우는 도커

초판발행 2016년 1월 22일

지은이 나카이 에츠지 / **옮긴이** 김성재 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-796-5 15000 / **정가** 7,000원

총괄 전태호 / **책임편집** 김창수 / **기획·편집** 김상민
디자인 표지/내지 여동일, 조판 김경수
마케팅 박상용, 송경석 / **영업** 김형진, 김진불, 조유미

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

저자 소개

지은이_ **나카이 에츠지**

1971년 오사카에서 출생했으며 첫 직업은 학원 강사였다. 외국계 업체에서 Linux/OSS를 중심으로 하는 프로젝트를 이끌면서, 다수의 기술 가이드, 잡지 기사 등을 집필했다. 이후 레드햇에 이직하여 에반젤리스트로서 기업 시스템에서의 Linux/OSS 활용 촉진에 힘쓰고 있다.

역자 소개

옮긴이_ **김성재**

기술 분야 전문 번역가. 관심 분야는 IT 기술과 일본어 교육 콘텐츠 등이다. 최근에는 업무에 필요한 맥 OS와 iOS 애플리케이션의 개발과 리뷰, 환경 구축에 관심이 있다. 번역서로는 『구글 웹로그 분석』, 『엔지니어를 위한 데이터 시각화』, 『하이브리드 앱을 구현하는 기술』, 『28일 동안 배우는 리눅스 서버 관리』, 『R로 배우는 데이터 분석 기본기 데이터 시각화』, 『스프링3 입문』, 『실전 클라우드 VMware View 가상화』, 『빅데이터의 충격』, 『만들면서 배우는 인터프리터』, 『우분투 환경에서 C언어로 배우는 리눅스 프로그래밍』, 『만들면서 배우는 기계 학습』(이상, 한빛미디어) 등이 있다.

차례

chapter 1

리눅스 컨테이너의 기초

Docker는 미국의 Docker사에서 오픈소스 소프트웨어로 개발을 진행하는 '리눅스 컨테이너' 관리 도구입니다. 리눅스 컨테이너에 관해서는 뒤에서 자세히 설명하지만, 특히 Docker는 컨테이너에 할당된 디스크 이미지를 간편하게 관리할 수 있는 점이 특징입니다. 이미 애플리케이션이 설치된 이미지를 내려받아 이용할 수도 있고 자신이 새로운 이미지를 만들어 공개할 수도 있습니다.

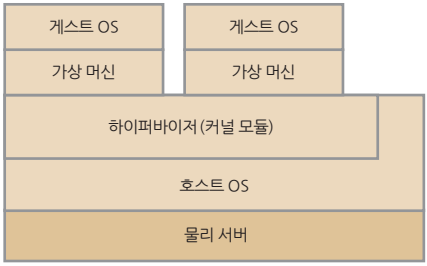
이 책에서는 Docker 사용법과 함께 Docker의 기반이 되는 기술을 설명합니다. Docker에서는 리눅스 컨테이너를 비롯한 다양한 리눅스의 최신 기술이 활용됩니다. 이 책은 Docker를 통해서 리눅스 활용 능력을 한 단계 끌어올리는 걸 목표로 합니다.

1.1 리눅스 컨테이너의 개념

우선 리눅스 컨테이너의 개요를 설명하겠습니다. Docker는 발전 중인 기술입니다. Docker를 이용할 때는 Docker의 구조를 확실하게 이해한 뒤, 적절한 이용 방법과 이용 상황을 찾을 필요가 있습니다. 그리고 Docker의 구조를 이해하려면 반드시 그 기초가 되는 리눅스 컨테이너를 알아야 합니다.

[그림 1-1]은 리눅스 KVM에 의한 서버 가상화 구조입니다.

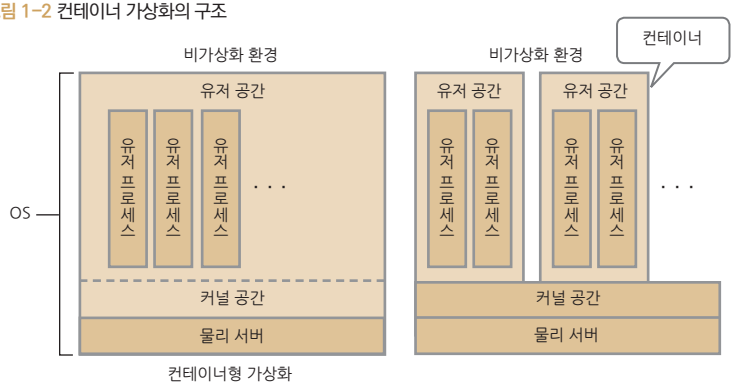
그림 1-1 리눅스 KVM에 의한 서버 가상화 구조



물리 서버에 호스트 OS로 Red Hat Enterprise Linux(RHEL) 등의 리눅스를 설치하면 호스트 OS가 하이퍼바이저 기능을 제공합니다. 하이퍼바이저는 호스트 리눅스 위에 복수의 가상 머신을 만들고, 각 가상 머신에서는 개별 게스트 OS가 동작합니다.

반면에, 리눅스 컨테이너에는 가상 머신이라는 개념이 없습니다. 비가상화 환경의 OS는 물리 하드웨어를 관리하는 커널 부분(커널 공간)과 그 위에서 동작하는 유저 프로세스 군(유저 공간)으로 구성됩니다. 컨테이너형 가상화에서는 리눅스 커널의 기능으로 유저 공간을 여러 개로 나눕니다. 나뉜 각 유저 공간이 '컨테이너'에 해당합니다([그림 1-2]).

그림 1-2 컨테이너 가상화의 구조



그렇다면, 각 컨테이너에선 무엇이 나뉜 것일까요? 구체적으로는 다음과 같습니다.

- 프로세스 테이블
- 파일 시스템
- 네트워크 설정
- CPU, 메모리 할당량

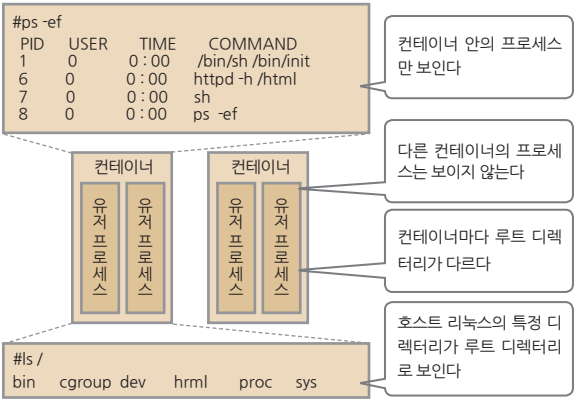
컨테이너별로 이런 리소스들을 개별적으로 할당함으로써, 각 컨테이너의 내부 프로세스는 다른 컨테이너와 격리된 환경에서 실행됩니다. 바꿔 말하면, 컨테이너별로 독립된 환경에서 애플리케이션을 실행할 수 있게 됩니다.

1.2 컨테이너에 의한 리소스 분할

이야기가 조금 추상적이 되었으므로, 이번에는 구체적으로 리소스 분할 구조를 설명하겠습니다. 또한, 이 뒤에서는 RHEL 6.5 환경에서 실제로 컨테이너를 만들고 확인합니다. 다음 장부터는 Docker를 이용해서 컨테이너를 만들지만, 이 장에선 컨테이너의 기초를 이해하고자 libvirt를 이용한 가상화 방법을 소개합니다. Docker보다 조금 절차가 복잡하고 손이 많이 가지만, 이를 통해 'Docker에선 어떤 점이 편리해졌는지' 본질을 이해할 수 있게 됩니다.

우선, '프로세스 테이블'과 '파일 시스템' 분할을 살펴봅시다([그림 1-3]).

그림 1-3 프로세스 테이블과 파일 시스템 분할



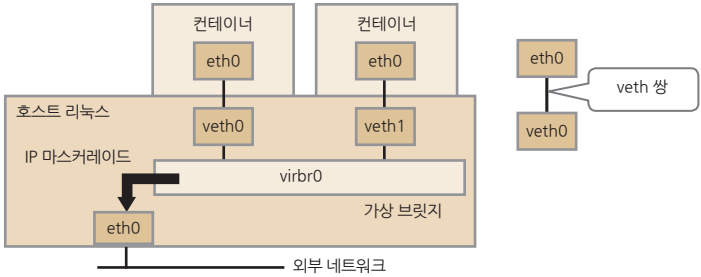
프로세스 테이블은 리눅스 상에서 동작하는 프로세스의 목록을 말합니다. 일반 환경에서 ps 커맨드를 실행하면 서버에서 실행 중인 모든 프로세스 목록이 표시됩니다. 반면에 컨테이너의 내부에서 ps 커맨드를 실행하면 컨테이너 내부에서 실행 중인 프로세스만 표시됩니다. 다른 컨테이너에서 동작하는 프로세스의 존재는 알 수 없습니다.

파일 시스템은 컨테이너 안에서 보이는 디렉터리를 말합니다. 호스트 리눅스 상의 특정 디렉터리를 컨테이너의 루트 디렉터리로 할당합니다. 이전부터 사용된 chroot(체인지 루트)와 같은 메커니즘으로 생각하면 됩니다.

다음 '네트워크 설정'은 조금 재미있는 구조로, 리눅스의 네트워크 이름 공간(netns) 기능을 이용합니다. 네트워크 이외의 이름 공간에 관해서는 이 장 끝에 있는 '1.6 [칼럼]이름 공간의 진화와 컨테이너 구현'을 참조하세요. 네트워크 이름 공간은 한 대의 리눅스 서버에서 복수의 네트워크 설정을 사용하는 기능입니다. 복수의 이름 공간을 준비하면 각 이름 공간에서 NIC(네트워크 인터페이스 카드), IP 주소, 라우팅 테이블 등을 따로 설정할 수 있습니다.

이 뒤에 소개할 libvirt를 이용하면 [그림 1-4]와 같은 네트워크가 구성됩니다.

그림 1-4 리눅스 컨테이너의 네트워크 구성



그림에 있는 veth란 직결된 가상 NIC 쌍을 만드는 기능입니다. 이 페어의 한쪽을 컨테이너 내의 네임 스페이스에 넣고 컨테이너 안에서만 보이게 합니다. 그리고 다른 한쪽을 가상 브릿지에 연결하면 가상 브릿지를 거쳐 컨테이너 간의 통신이 가능해집니다. 또한, 가상 브릿지에 IP 어드레스를 할당함으로써 호스트 리눅스에서 컨테이너에 접속하거나 컨테이너에서 외부 네트워크로 접속할 수도 있습니다. 이 구조는 리눅스 KVM의 서버 가상화 환경에서 가상 네트워크의 구조와 거의 같습니다.

'CPU, 메모리 할당'은 리눅스의 'Control Groups(cgroups)' 기능을 이용합니다. cgroups로 컨테이너 내의 프로세스에 대한 CPU 할당 우선순위, 최대 메모리 사용량 등을 설정합니다.

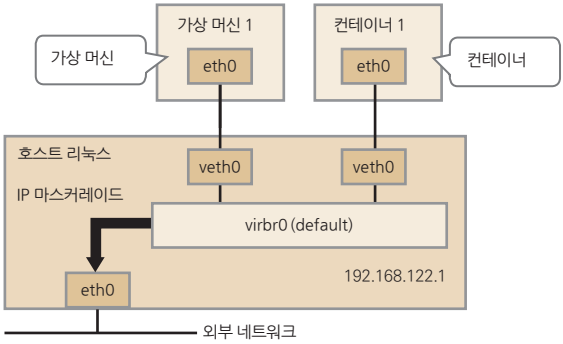
1.3 libvirt로 컨테이너를 체험한다

그럼, 실제로 컨테이너를 만들어 '컨테이너의 내부'를 경험해 봅시다. 여기서는 RHEL 6.5에 표준으로 포함된 가상화 관리 도구 libvirt를 이용하여 컨테이너를 만듭니다. libvirt는 리눅스 KVM의 가상 머신을 만들고 관리하는 도구이지만, 사실은 컨테이너를 만들고 관리하는 기능도 갖추고 있습니다.[1]

리눅스 KVM의 호스트 리눅스로 RHEL 6.5를 도입하면 KVM 가상 머신과 나란히 컨테이너를 만들고 관리할 수 있게 됩니다([그림 1-5]).

1 RHEL 6에서 컨테이너 이용은 Red Hat사의 정식 지원 대상은 아닙니다. 자기 책임으로 이용해 주세요.

그림 1-5 libvirt로 가상 머신과 컨테이너를 동시에 관리한다



RHEL6.5 설치 순서는 레드햇 사이트를 참고로 해주세요.

다음으로 컨테이너에 보여줄 디렉터리를 준비합니다. 여기서는 'busybox'를 이용한 간이 웹 서버를 컨테이너에서 실행합니다. 필요한 파일을 디렉터리 '/lxcguest01' 아래에 준비하고, 컨테이너의 루트 디렉터리로 할당합니다. 처음으로 실행 파일을 저장할 디렉터리 '/lxcguest01/bin'과 HTML 파일을 저장할 디렉터리 '/lxcguest01/html'을 준비합니다.

```
# mkdir -p /lxcguest01/bin
# mkdir -p /lxcguest01/html
```

이어서 디렉터리 '/lxcguest01/bin' 안에 busybox 커맨드 파일을 복사한 다음, 각종 커맨드의 심볼릭 링크를 만듭니다.

```
# cd /lxcguest01/bin
# cp /sbin/busybox ./
# for i in echo grep rm \
ifconfig kill ps route \
test which cat false head \
ls pwd sh true date find \
httpd ip ping sleep wget; \
```

```
do ln -s busybox $i; done
```

※ \는 커맨드 도중에서 행을 바꿀 때 입력하는 기호

이것은 busybox의 독특한 이용법입니다. busybox는 하나의 커맨드 파일로 다양한 커맨드의 기능을 모아서 제공하는 툴입니다. 예를 들어 ls라는 이름의 심볼릭 링크에서 busybox를 실행하면 ls 커맨드로 기능합니다. 앞 코드에서는 httpd라는 이름의 심볼릭 링크도 준비되어 있어 이 심볼릭 링크로 실행하면 busybox는 HTTP 데몬으로서 동작합니다.

이어서 컨테이너를 시작했을 때, 최초로 실행할 스크립트를 '/lxcguest01/bin/init.sh'에 작성합니다.

[리스트 1-1] /lxcguest01/bin/init.sh

```
#!/bin/sh
ifconfig eth0 192.168.122.190
route add default gw 192.168.122.1 eth0
httpd -h /html
while [[ true ]]; do
sh
done
```

또한, 다음 커맨드로 실행 권한을 설정합니다.

```
# chmod u+x /lxcguest01/bin/init.sh
```

이 스크립트는 컨테이너에 할당된 가상 NIC eth0에 IP 어드레스 '192.168.122.190'을 설정하여 HTTP 데몬을 시작한 다음, 셸 'sh'를 실행하고 있습니다. 여기서는 셸을 종료해도 다시 같은 셸을 시작하게 되어 있습니다.

마지막으로 웹 서버에서 공개하는 콘텐츠로 '/lxcguest01/html/index.html'

을 준비합니다.

[리스트 1-2] /lxcguest01/html/index.html

```
<html>
<head>
<title>Linux Container</title>
</head>
<body>
<h1>Welcome to Linux Container!</h1>
</body>
</html>
```

1.4 컨테이너 내부와 호스트 리눅스의 관계

그럼 실제로 컨테이너를 만들어 보겠습니다. libvirt로 컨테이너를 만들 때 사전에 XML 정의 파일을 준비합니다.

[리스트 1-3] lxcguest01.xml

```
<domain type='lxc'>
<name>lxcguest01</name>
<memory>200000</memory>
<os>
<type>exe</type>
<init>/bin/init.sh</init>
</os>
<devices>
<interface type='network'>
<source network='default'/>
</interface>
<console type='pty'/>
<filesystem type='mount'>
<source dir='/lxcguest01'/>
<target dir='/'/>
</filesystem>
</devices>
```

```
</domain>
```

이 XML 파일을 현재 디렉터리에 'lxcguest01.xml'로 저장한 후, 다음 단계에서 컨테이너를 정의하고 실행합니다.

커맨드 1-1 컨테이너를 정의하고 시작하는 모습

```
# vi init.sh
# chmod u+x /lxcguest01/bin/init.sh
# vi /lxcguest01/html/index.html
# vi lxcguest01.xml
# virsh -c lxc:/// define lxcguest01.xml
Domain lxcguest01 defined from lxcguest01.xml

]# virsh -c lxc:/// list --all
 Id    Name                           State
----------------------------------------------------
 -     lxcguest01                     shut off

# virsh -c lxc:/// start lxcguest01
Domain lxcguest01 started
```

첫 번째 커맨드로 XML 파일로 컨테이너를 정의하고 두 번째 커맨드로 결과를 확인합니다. 마지막 커맨드로 실제로 컨테이너를 시작합니다. 이때 호스트 리눅스 상에서 ps -efww 커맨드를 실행하면, 출력의 끝에 [커맨드 1-2]처럼 내용을 확인할 수 있습니다.

커맨드 1-2 호스트 리눅스에서 본 컨테이너 내의 프로세스

```
root      9646     1  0 01:14 ?        00:00:00 /usr/libexec/libvirt_lxc
--name lxcguest01 --console 22 --security=none --handshake 25 --background
--veth veth1
root      9647  9646  0 01:14 ?        00:00:00 /bin/sh /bin/init.sh
root      9664  9647  0 01:14 ?        00:00:00 httpd -h /html
root      9665  9647  0 01:14 ?        00:00:00 sh
```

출력된 결과를 보면 컨테이너 관리 툴 'libvirt_lxc'가 컨테이너 컨테이너를 만들고, 그 자식 프로세스로서 먼저 준비한 init.sh를 실행하고 있습니다. 그리고 init.sh에서 HTTP 데몬인 httpds와 sh를 실행하고 있습니다.

호스트 리눅스의 데스크톱에서 firefox를 실행하여 'http://192.168.122.190'에 액세스하면 [리스트 1-2]의 콘텐츠가 표시됩니다([그림 1-6]).

그림 1-6 컨테이너 내의 웹 서버에 접속

단, 이것은 컨테이너 밖에서 본 모습입니다. 컨테이너 내부에서는 같은 컨테이너 안의 프로세스만 보일 것입니다. 아래는 컨테이너 내부에서 실행되는 셸에 접속하여 컨테이너 내부 환경을 확인한 모습입니다.

커맨드 1-3 컨테이너 내부 환경 확인

```
# virsh -c lxc:/// console lxcguest01
Connected to domain lxcguest01
Escape character is ^]
# ps -ef
PID   USER     TIME   COMMAND
    1 0          0:00 /bin/sh /bin/init.sh
    6 0          0:00 httpd -h /html
```

```
    7 0          0:00 sh
   11 0          0:00 ps -ef
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:00:89:BD:21
          inet addr:192.168.122.190  Bcast:192.168.122.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe89:bd21/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:313 errors:0 dropped:0 overruns:0 frame:0
          TX packets:22 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17531 (17.1 KiB)  TX bytes:2306 (2.2 KiB)

# ls /
bin             dev             lxcguest01.xml  sys
cgroup          html            proc
# Ctrl + ] 키로 컨테이너에서 빠져 나온다
# virsh -c lxc:/// destroy lxcguest01
Domain lxcguest01 destroyed
```

ps 커맨드를 실행하면 컨테이너 내부에서 실행한 프로세스만 표시되고, 가상 NIC 'eth0'에 IP 주소가 192.168.122.190으로 설정된 것도 알 수 있습니다. 또한, 호스트 리눅스의 디렉터리 /lxcguest01에 준비한 내용이 컨테이너 안의 루트 파일 시스템으로 되어 있습니다. 앞의 마지막 커맨드는 컨테이너를 정지합니다.

1.5 정리

Docker의 기초가 되는 '리눅스 컨테이너'를 설명했습니다. libvirt를 이용해 실제로 컨테이너를 만들었지만 몇 가지 번거로운 점도 있었습니다. 특히 본격적인 애플리케이션을 실행할 경우 컨테이너에 할당할 디렉터리 준비가 큰일입니다. 이번 예로 말하자면 lxcguest01 디렉터리 아래에 애플리케이션 실행에 필요한 파

일을 모두 준비해야만 했습니다. 또한, 컨테이너를 정의하는 XML 파일 작성도 번거롭습니다.

그런데 Docker를 이용하면 이런 문제를 깔끔하게 해결할 수 있습니다. Docker에서는 애플리케이션을 도입한 디스크 이미지를 자동으로 만들어 컨테이너에 할당할 수 있고 컨테이너도 커맨드 하나로 생성할 수 있습니다. 다음 장부터는 실제로 Docker를 설치해서 Docker의 편리함을 체험해 봅시다.

1.6 [칼럼]이름 공간의 진화와 컨테이너의 구현

이름 공간이라는 말이 나왔지만 리눅스 커널에는 이 밖에도 다양한 이름 공간의 기능이 있습니다. 주요 이름 공간을 정리했습니다([표 1-1]).

표 1-1 리눅스 커널의 주요 이름 공간

이름 공간	기능	구현 커널 버전
Mount namespace	파일 시스템 분리	2.4.19
UTS namespace	호스트 네임 분리	2.6.19
IPC namespace	프로세스간 통신 분리	2.6.19
User namespace	유저(UID/GID) 분리	3.8
PID namespace	프로세스 테이블 분리	2.6.24
Network namespace	네트워크 설정 분리	2.6.24

이름 공간은 모두 리눅스 상에서 동작하는 프로세스에 할당하는 리소스를 분리하는 기능을 합니다. 리눅스 컨테이너는 이름 공간의 조합으로 구현되며 컨테이너라는 기술이 단독으로 있는 것은 아닙니다. 리눅스에서 컨테이너를 이용하는 도구에는 Docker 외에도 오래전부터 이용되는 'lxctools', 혹은 이번에 사용할 libvirt 등이 있습니다. 이들은 모두 그 이면에서는 이름 공간을 이용하여 컨테이너를 만듭니다.

각각 이름 공간은 커널의 버전 업에 따라서 단계적으로 도입되었습니다. 이런 이름 공간의 진화로 비로소 현재의 컨테이너가 구현된 것입니다. 'User namespace'는 현재의 Docker에서는 사용되지 않으며 이후 버전에서 지원을 검토하고 있습니다. 컨테이너는 앞으로도 계속 진화하는 기술이라고 할 수 있습니다.