도커 오케스트레이션

Docker Orchestration

애플리케이션 빌드,
테스트,
배포의 통합 관리

슈리크리슈나 홀라 지음
이기곤 옮김

PACKT PUBLISHING
한빛미디어 Hanbit Media, Inc.

도커 오케스트레이션

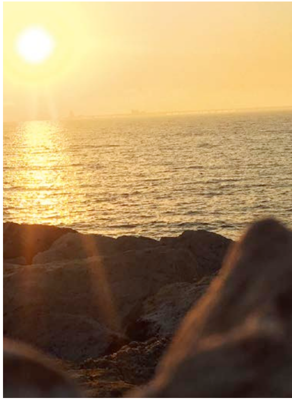
Docker Orchestration

애플리케이션 빌드,
테스트,
배포의 통합 관리

슈리크리슈나 홀라 지음
이기곤 옮김

이 도서는
Orchestrating Docker(PACKT publishing)의
번역서입니다

PACKT PUBLISHING
한빛미디어 Hanbit Media, Inc.



표지 사진 **안상규**

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

사진 보내기 ebookwriter@hanbit.co.kr

도커 오케스트레이션 애플리케이션 빌드, 테스트, 배포의 통합 관리

초판발행 2015년 12월 10일

지은이 슈리크리슈나 홀라 / **옮김이** 이기곤 / **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 978-89-6848-793-4 15000 / **정가** 13,000원

총괄 배용석 / **책임편집** 김창수 / **기획·편집** 정지연 / **교정** 이미연
디자인 표지/내지 여동일, 조판 최송실
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

Published by HANBIT Media, Inc. Printed in Korea
Copyright © Packt Publishing 2015. First published in the English language under the title 'Orchestrating Docker'(9781783984787). This translation is published and sold by permission of Packt Publishing, which owns or controls all rights to publish and sell the same.
이 책의 저작권은 Packt Publishing과 한빛미디어(주)에 있습니다.
저작권법에 의해 보호를 받는 저작물이므로 무단 복제 및 무단 전재를 금합니다.

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

지은이_ **슈리크리슈나 홀라**Shrikrishna Holla

슈리크리슈나 홀라는 인도의 풀스택full-stack(프론트엔드부터 DB, 인프라 구축까지 모든 걸 하는) 개발자입니다. 자전거 타기와 음악 듣기를 좋아하며 가끔은 그림을 그리기도 합니다. 해커톤에서 후드 티를 입고 밤을 새우기 위해 레드불을 마시는 그의 모습을 자주 볼 수 있습니다. 현재는 클라우드 기반의 고객 서비스 플랫폼인 Freshdesk의 제품 개발자로 일하고 있습니다.

저자와 연락하려면 트위터(@srikrishnaholla) 또는 Docker IRC 채널(Freenode의 #docker)에서 shrikrishna 핸들을 이용하면 됩니다.

역자 소개

옮긴이_ **이기곤**

아이렌소프트(AirenSoft)에서 풀스택 개발자로 일하고 있으며, 주로 C/C++ 언어를 사용하며 멀티미디어 분야에 관심이 있습니다. 저서로는 『FFmpeg 라이브러리: 코덱과 영상 변환을 중심으로』(한빛미디어, 2015)가 있습니다.

저자 서문

Docker와 함께 시작하세요. Docker는 애플리케이션 샌드박싱[01] 기술에 혁명을 가져온 리눅스의 컨테이너화 기술입니다. 이 책을 통해 여러분은 개발 환경을 빠르게 구성하고 애플리케이션 배포 환경을 간단하게 만들기 위해 어떻게 Docker를 사용하는지 배울 수 있습니다.

이 책은 가상화된 Docker 컨테이너 안에서 애플리케이션을 실행하는 방법부터 완성된 컨테이너를 어디서든지 실행하는 방법까지 설명합니다. Docker는 여러분의 개발 환경, 사설 서버, 심지어는 가상화된 클라우드 서비스에서도 실행이 가능할 뿐만 아니라 독립적인 PaaS Platform as a Service[02]를 구성하여 애플리케이션을 배포할 수 있으며 이 모든 것은 여러분의 개발 환경에서 쉽게 구현할 수 있습니다.

저는 Docker를 개발한 분들께 감사의 인사를 드리고 싶습니다. 이분들이 없었다면 이 책은 세상의 빛을 보지 못했을 것입니다. 편집자인 파리타 Parita와 라리사 Larissa에게, 긴 여정이었지만 저에게 매주 많은 도움을 주셨습니다. 제게는 항상 가장 어두운 감옥에서 마지막 한 줄기 빛과도 같은 존재였으며 앞으로도 그럴 나의 부모님께, 내가 우울할 때마다 위로하는 말을 해준 여동생에게, 나의 진로를 결정하는 데 도와주신 선생님들께, 나의 모든 걱정을 잊게 해준 친구 모두에게 감사의 마음을 전합니다. 수많은 분께서 제게 격려와 조언 그리고 피드백을 주셨고 그분들의 도움 없이는 이 일을 이루어내지 못했을 것입니다. 마지막으로 저를 믿고 책을 보시는 독자분들께도 감사의 말씀을 드립니다.

01 역자주_격리된 환경 안에서 프로그램을 관리하는 방식

02 역자주_플랫폼 구성에 필요한 요소를 설치할 필요 없이 서비스하는 개념

불과 몇 년 전까지만 해도 서비스를 확장하기 위해서는 많은 노력이 필요했습니다. 새로운 서버를 설치하는 과정부터 애플리케이션을 실행하기 위한 환경까지 일일이 신경 써야 했기 때문입니다.

점점 복잡해지는 서비스와 거대해지는 데이터를 해결하기 위해서 수많은 사람의 연구가 진행되었습니다. 이러한 노력의 하나로 클라우드 서비스가 등장하면서 새로운 서버를 설치하기 위한 과정은 더는 고민거리가 되지 않았습니다. 더불어 많은 기업이 클라우드 환경에서 서비스를 구성하기 위한 인프라를 구축하기 시작했습니다. 이 외에도 Ansible, Puppet, Chef와 같은 자동화 관리 도구가 발전하면서 애플리케이션을 실행하기 위한 환경을 구성하는 일도 점점 간소화되었습니다.

Docker는 이러한 노력의 연장선에 있는 오픈소스 프로젝트로, 격리화된 컨테이너 기술을 통해 애플리케이션을 실행하기 위한 환경을 구성할 필요 없이 일관성 있고 효율적인 배포를 가능하게 합니다. 컨테이너는 완벽하게 격리되어 있어서 개발자는 라이브러리 충돌이나 의존성과 같은 기술적인 문제에 대해 전혀 신경을 쓰지 않아도 되며 인프라 관리자는 단지 컨테이너를 배포하기만 하면 끝입니다. 또한, 컨테이너별로 필요한 자원을 직접 지정할 수 있으므로 효율적인 운영이 가능합니다.

하지만 여러분은 자동화라는 단어에 대해 조심스럽게 생각해야 합니다. 비록 Docker가 모든 것을 자동화하는 것은 사실이지만, 제대로 관리되지 않는다면 큰 재앙을 불러올 수 있습니다. 또한, Docker가 상용 환경에서 안정적으로 제 기능을 수행하려면 다양한 안전장치가 필요합니다. 컨테이너를 관리하는 방법, 장애 대응, 취약점 대비와 같은 것들이 이러한 장치에 속합니다.

그럼에도 Docker가 정말 멋진 도구라는 사실은 변하지 않으며 여러분은 이 책을 통해 Docker가 가진 특별한 매력을 느낄 수 있습니다.

이 책에서 다루는 것들

1장 Docker 뜯어보기에서는 여러분의 개발 환경에서 Docker를 실행하는 방법을 설명합니다.

2장 Docker 명령어와 Dockerfile에서는 Docker 프로그램에 익숙해지기 위한 과정을 포함하여 여러분만의 컨테이너를 Dockerfile을 통해 생성할 수 있게 도와드립니다.

3장 Docker 컨테이너 설정에서는 컨테이너가 사용하는 시스템 자원을 세부적으로 관리하기 위한 여러 방법을 설명합니다.

4장 자동화와 보안에서는 컨테이너를 관리하기 위한 다양한 기술들을 다룹니다. 수퍼바이저supervisor를 이용한 여러 서비스의 조직화, 서비스 탐색, Docker의 보안과 관련된 지식을 이 장에서 살펴볼 수 있습니다.

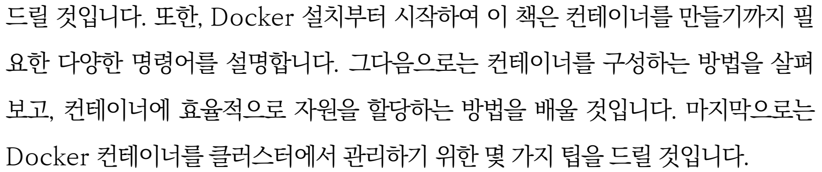
5장 Docker의 친구들에서는 Docker를 둘러싼 세계를 보여드립니다. Docker를 사용하는 오픈소스 프로젝트들을 만나볼 수 있고, 마지막으로는 CoreOS 운영체제를 사용하여 독립적인 PaaS와 클러스터cluster를 구성하는 방법을 설명할 것입니다.

이 책을 위해 필요한 것들

이 책은 리눅스와 Git에 어느 정도 익숙한 독자를 대상으로 합니다. 이러한 기술에 익숙하지 않다면 이 책에서 제공하는 예제를 실행하는 데 어려움을 겪을 수 있습니다. Docker를 설치하기 위해서는 관리자 권한이 필요하며 윈도우와 OS X 사용자는 VirtualBox를 설치해야 합니다.

이 책이 필요한 독자

개발자거나 시스템 관리자 또는 그 사이에 있는 사람이라면 이 책은 여러분에게 Docker를 구성하는 방법과 함께 애플리케이션을 테스트하고 배포하는 방법을 알려



드릴 것입니다. 또한, Docker 설치부터 시작하여 이 책은 컨테이너를 만들기까지 필요한 다양한 명령어를 설명합니다. 그다음으로는 컨테이너를 구성하는 방법을 살펴보고, 컨테이너에 효율적으로 자원을 할당하는 방법을 배울 것입니다. 마지막으로는 Docker 컨테이너를 클러스터에서 관리하기 위한 몇 가지 팁을 드릴 것입니다.

각 장을 순서대로 진행하다 보면 빠르게 Docker를 마스터할 수 있고 애플리케이션을 배포할 때 밤을 새울 일은 없어질 것입니다.

예제 파일

이 책의 예제 코드는 다음 링크에서 다운로드합니다.

- https://www.hanbit.co.kr/exam/2793/

한빛 리얼타임

한빛 리얼타임은 IT 개발자를 위한 전자책입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾기도 쉽지 않습니다. 또한, 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1 eBook First – 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다

500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 좀 더 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한, 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2 무료로 업데이트되는 전자책 전용 서비스입니다

종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3 독자의 편의를 위해 DRM-Free로 제공합니다

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 어려운 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가겠습니다.

지금 보시는 전자책에 소유 권한을 표시한 문구가 없거나 타인의 소유권한을 표시한 문구가 있다면 위법하게 사용하고 있을 가능성이 큽니다. 이 경우 저작권법에 따라 불이익을 받으실 수 있습니다.

다양한 기기에 사용할 수 있습니다. 또한, 한빛미디어 사이트에서 구매하신 후에는 횟수와 관계없이 다운로드할 수 있습니다.

한빛미디어 전자책은 인쇄, 검색, 복사하여 붙이기가 가능합니다.

전자책은 오탈자 교정이나 내용의 수정·보완이 이뤄지면 업데이트 관련 공지를 이메일로 알려 드리며, 구매하신 전자책의 수정본은 무료로 내려받으실 수 있습니다.

이런 특별한 권한은 한빛미디어 사이트에서 구매하신 독자에게만 제공되며, 다른 사람에게 양도나 이전은 허락되지 않습니다.

차례

chapter 1 Docker 뜯어보기 015

1.1 Docker 설치 018
1.1.1 우분투에서의 Docker 설치 018
1.1.2 Docker 업그레이드 021
1.1.3 맥 OS X와 윈도우 021
1.2 오픈스택 025
1.2.1 데브스택을 이용한 Docker 설치 025
1.2.2 Docker 수동 설치 026
1.2.3 노바 설정 027
1.2.4 글랜스 설정 028
1.2.5 Docker와 오픈스택 사이의 워크플로우 028
1.3 인셉션: Docker 안의 Docker 030
1.3.1 의존성 패키지 030
1.3.2 소스로부터 Docker 빌드하기 031
1.4 설치 검증 032
1.5 유용한 팁 034
1.5.1 루트가 아닌 계정에 권한 부여하기 034
1.5.2 방화벽 설정 034
1.6 요약 035

chapter 2 Docker 명령어와 Dockerfile 037

2.1 Docker에서 사용하는 용어 037
2.1.1 Docker 컨테이너 038
2.1.2 Docker 데몬 039
2.1.3 Docker 클라이언트 039
2.1.4 Dockerfile 040
2.1.5 Docker 레지스트리 040

2.2 Docker 명령어 040

2.2.1 Docker 데몬 실행 041
2.2.2 version 042
2.2.3 info 042
2.2.4 run 043
2.2.5 search 049
2.2.6 pull 050
2.2.7 start 050
2.2.8 stop 051
2.2.9 restart 051
2.2.10 rm 052
2.2.11 ps 052
2.2.12 logs 053
2.2.13 inspect 054
2.2.14 top 056
2.2.15 attach 056
2.2.16 kill 057
2.2.17 cp 057
2.2.18 port 058

2.3 프로젝트 시작하기 059

2.3.1 diff 060
2.3.2 commit 061
2.3.3 images 062
2.3.4 rmi 064
2.3.5 save 065
2.3.6 load 065
2.3.7 export 065
2.3.8 import 066
2.3.9 tag 066
2.3.10 login 067
2.3.11 push 067
2.3.12 history 068

2.3.13 events 068
2.3.14 wait 069
2.3.15 build 069
2.3.16 Docker 데몬으로 업로드하기 070

2.4 Dockerfile 074

2.4.1 FROM 075
2.4.2 MAINTAINER 076
2.4.3 RUN 076
2.4.4 CMD 077
2.4.5 ENTRYPOINT 079
2.4.6 WORKDIR 081
2.4.7 EXPOSE 081
2.4.8 ENV 082
2.4.9 USER 082
2.4.10 VOLUME 082
2.4.11 ADD 083
2.4.12 COPY 084
2.4.13 ONBUILD 084

2.5 Docker의 작업 흐름도 088

2.6 자동화 빌드 구성 089

2.6.1 빌드 트리거 092
2.6.2 웹훅 092

2.7 요약 094

chapter 3 Docker 컨테이너 설정 095

3.1 자원 제한 097

3.1.1 CPU 우선순위 097
3.1.2 메모리 제한 098
3.1.3 디바이스 매퍼를 활용한 디스크 제한 099

3.2 컨테이너 내부 데이터 관리 103
3.2.1 데이터 전용 컨테이너 103
3.2.2 다른 컨테이너에서 볼륨 사용하기 104
3.2.3 활용 사례 – Docker에서 MongoDB 사용하기 104
3.3 스토리지 드라이버 설정 105
3.3.1 디바이스 매퍼를 스토리지 드라이버로 사용 105
3.3.2 Btrfs를 스토리지 드라이버로 사용하기 106
3.4 Docker 네트워크 설정 107
3.4.1 컨테이너와 호스트 간의 포트포워딩 설정 110
3.4.2 사설 아이피 설정 111
3.5 컨테이너 연결 112
3.5.1 동일한 호스트 간 컨테이너 연결 112
3.5.2 앰배서더 컨테이너를 이용한 호스트 간 컨테이너 연결 113
3.6 요약 115

chapter 4 자동화와 보안 117

4.1 Docker 원격 API 118
4.1.1 컨테이너 API 120
4.1.2 이미지 API 122
4.1.3 기타 API 123
4.1.4 run 명령어가 실행되는 과정 125
4.2 실행 중인 컨테이너에 프로세스를 추가하는 방법 126
4.3 서비스 발견 127
4.3.1 link 옵션과 앰배서더 컨테이너를 활용한 컨테이너 연결 128
4.3.2 etcd를 이용한 서비스 발견 130
4.3.3 Docker 오케스트레이션 133
4.3.4 Docker Machine 133
4.3.5 Docker Swarm 134
4.3.6 Docker Compose 135

4.4 보안 137
4.4.1 커널 네임스페이스 139
4.4.2 컨트롤 그룹 140
4.4.3 컨테이너 안의 루트 권한 140
4.4.4 Docker 데몬 공격 142
4.4.5 보안을 위한 최고의 수칙 143
4.5 요약 144

chapter 5 Docker의 친구들 145

5.1 Chef와 Puppet을 활용한 Docker 146
5.1.1 Chef로 활용하는 Docker 147
5.1.2 Puppet으로 활용하는 Docker 148
5.2 apt-cacher 설정 149
5.2.1 apt-cache를 활용한 이미지 빌드 150
5.3 미니 Heroku 설정 151
5.3.1 부트스트래퍼 스크립트를 사용한 Dokku 설치 152
5.3.2 Vagrant를 사용한 Dokku 설치 153
5.3.3 호스트 이름 설정과 공용 키 추가 154
5.3.4 애플리케이션 배포 155
5.4 고가용성 서비스 설정 156
5.4.1 의존성 패키지 설치 158
5.4.2 Vagrantfile 설정 158
5.5 요약 165

chapter 3

Docker 컨테이너 설정

이전 장에서는 Docker에서 사용하는 다양한 명령어들을 살펴보았습니다. 이미지를 가져오거나 컨테이너를 실행한 뒤 새로운 이미지를 생성하여 저장소로 푸시하는 과정들을 배웠고, 자동화를 위한 Dockerfile을 구성하는 방법도 배웠습니다.

이번 장에서는 컨테이너를 다루는 방법에 대해 자세히 살펴보겠습니다. Docker의 컨테이너들은 모두 잘 격리되어 있지만, 격리가 잘 되어 있다고 해서 악의적인 프로세스로부터 시스템 자원을 갉아먹는 행위들을 방지할 수는 없습니다. 예를 들어, 다음 명령어 하나를 소개하겠습니다(절대 이 명령어를 실행해서는 안 됩니다!).

```
$ docker run ubuntu /bin/bash -c ":(){ :|:& };:"
```

포크 폭탄fork bomb이라고도 하는 이 명령어는 실행하는 순간 여러분의 모든 시스템 자원을 갉아먹을 것입니다. 위키피디아Wikipedia에서는 포크 폭탄을 다음과 같이 정의하고 있습니다.

> "포크 폭탄은 끊임없이 자기 자신을 복제하여 모든 시스템의 자원을 고갈하게 해 시스템 장애를 일으키는 서비스 거부 공격denial-of-service attack입니다."

Docker가 상용 소프트웨어의 일부로 사용되기 시작하면서 이와 같이 시스템 자원을 고갈시키는 행위는 매우 치명적이게 될 것입니다. Docker는 이러한 상황을 방지하기 위해 컨테이너가 사용하는 시스템 자원들을 제한하는 기능들을 가지고 있으며 이 기능들이 바로 이번 장에서 살펴볼 내용입니다.

이전 장에서는 Docker를 사용하는 데 필요한 기능만을 간략하게 설명했습니다.

이제 조금 더 깊숙하게 파고들어 가서 무엇이 중요하고 최선인지를 상세하게 설명하겠습니다. 이 외에도 Docker 데몬이 사용하는 스토리지 드라이버도 바꾸어 볼 것입니다.

물론 네트워크에 대한 내용도 설명합니다. 실행 중인 컨테이너를 살펴보면, Docker가 임의로 서브넷subnet을 지정하고 IP 주소를 할당한다는 것을 눈치챌 수 있습니다(기본으로는 172.17.42.0/16을 사용합니다). 이번 장에서는 이러한 설정들을 여러분만의 네트워크 환경으로 변경해봅니다.

다양한 상황에서 컨테이너끼리 통신이 필요할 때가 있습니다(한쪽에서는 애플리케이션이 실행 중이고, 다른 쪽은 DB를 위한 컨테이너가 있을 때와 같은 상황을 예로 들 수 있습니다). 이미지를 생성할 때 IP 주소를 설정할 수 없는 데다가 이를 미리 설정하는 것도 현실적인 방법은 아닙니다. 이에 대해 이번 장에서는 같은 호스트 또는 다른 호스트에서의 컨테이너 간 통신을 해결할 방법을 설명합니다.

앞의 내용을 조금 더 간단하게 정리하면 이 장에서는 다음과 같은 내용을 설명합니다.

- **컨테이너 자원 관리**
  CPU
  메모리
  디스크
- **컨테이너 내부 데이터 관리**
- **스토리지 드라이버 설정**
- **네트워크 설정**
  포트 포워딩
  사설 아이피 설정
- **컨테이너 연결**
  동일한 호스트에서의 컨테이너 연결
  다른 호스트 간 컨테이너 연결

3.1 자원 제한

샌드박싱을 지원하는 소프트웨어에서 할당 가능한 자원을 제한할 수 있게 만드는 것은 매우 중요합니다. Docker는 컨테이너가 실행될 때 사용 가능한 CPU나 메모리 같은 자원을 제한하는 기능을 제공합니다.

3.1.1 CPU 우선순위

컨테이너가 사용할 수 있는 CPU의 우선순위는 run 명령어에서 -c 플래그를 통해 지정할 수 있습니다.

```
$ docker run -c 10 -it ubuntu /bin/bash
```

인자로 받은 숫자(10)는 상대적인 우선순위를 뜻합니다. 기본으로 모든 컨테이너는 동일한 우선순위를 제공받지만, 앞의 예제와 같이 임의의 우선순위를 지정할 수도 있습니다.

실행되는 컨테이너의 CPU 우선순위는 호스트의 터미널에서 다음 명령어를 실행하여 확인할 수 있습니다(윈도우나 OS X 사용자는 SSH로 리눅스 호스트에 접근한 후 사용하면 됩니다).

```
$ cat /sys/fs/cgroup/cpu/docker/<컨테이너 ID>/cpu.shares
```

그럼 이미 실행 중인 컨테이너의 CPU 우선순위를 제어하는 것도 가능할까요? 물론 가능합니다. 앞에서 언급한 우선순위 경로에 원하는 값을 지정하면 됩니다.

```
$ echo 우선순위 값 > /sys/fs/cgroup/cpu/docker/<컨테이너 ID>/cpu.shares
```

NOTE

앞에서 언급한 경로가 존재하지 않을 때는 다음과 같이 명령어를 실행하여 cgroup이 마운트된 위치를 확인할 수 있습니다.

```
cat /proc/mounts | grep -w cpu
```

이 방법은 공식적으로 지원하는 방법이 아니지만, 추후 Docker가 실행 중인 컨테이너의 CPU 우선순위를 변경하는 기능을 제공할 가능성은 있습니다. 이와 관련된 내용은 https://groups.google.com/forum/#!topic/docker-user/-pP8-KgJJGg에서 확인할 수 있습니다.

3.1.2 메모리 제한

사용할 수 있는 최대 메모리 양을 제한하는 방법은 CPU 우선순위 변경과 마찬가지로 run 명령어를 수행할 때 -m 플래그를 통해 지정할 수 있습니다.

```
$ docker run -m <값><단위>
```

플래그에 지정할 수 있는 메모리 단위에는 바이트, 킬로바이트, 메가바이트와 기가바이트가 있으며 각각 b, k, m, g로 표기합니다. 메모리 단위는 다음과 같이 사용할 수 있습니다. 이 명령어는 실행되는 컨테이너가 메모리를 최대 1GB까지만 사용할 수 있게 제한합니다.

```
$ docker run -m 1024m -dit ubuntu /bin/bash
```

CPU 우선순위와 마찬가지로 최대로 사용 가능한 메모리는 다음 명령어로 확인할 수 있습니다.

```
$ cat /sys/fs/cgroup/memory/docker/<컨테이너 ID>/memory.limit_in_bytes
18446744073709551615
```

파일 이름에서 알 수 있듯이 이 명령어는 사용 가능한 메모리의 양을 바이트 단위로 출력하는데, 출력된 메모리 양은 1.8×1010 기가바이트로 제한이 없다고 봐도 무방합니다.

그렇다면 컨테이너가 실행 중일 때도 메모리 양을 제한하는 것이 가능할까요? CPU 우선순위와 마찬가지로 cgroup 파일을 수정하는 것으로 실행 중인 컨테이너가 사용할 수 있는 최대 메모리 양을 제한할 수 있습니다.

```
$ echo 바이트 단위 메모리 양 > /sys/fs/cgroup/memory/docker/<컨테이너 ID>/memory.
limit_in_bytes
```

NOTE

앞에서 언급한 경로가 존재하지 않을 때는 다음 명령어를 실행하여 cgroup이 마운트된 위치를 확인할 수 있습니다.

```
$ grep -w cgroup /proc/mounts | grep -w memory
```

마찬가지로 이 방법 또한 공식적으로 지원하지는 않지만, 추후 Docker가 실행 중인 컨테이너의 메모리 제한을 변경하는 기능을 지원할 가능성은 있습니다. 이와 관련된 자세한 내용은 https://groups.google.com/forum/#!topic/docker-user/-pP8-KgJJGg에서 확인할 수 있습니다.

3.1.3 디바이스 매퍼를 활용한 디스크 제한

컨테이너가 사용할 수 있는 최대 디스크 크기를 직접 제한하는 방법이 아직까지 없기 때문에 디스크 사용량을 제한하는 방법은 어렵습니다. 기본으로 사용하는 스토리지 드라이버인 AUFS도 일반적으로는 디스크를 제한하는 방법을 제공하지 않습니다(AUFS는 블록 디바이스를 제공하지 않기 때문입니다. 자세한 정보는 http://aufs.sourceforge.net/aufs.html에서 확인할 수 있습니다).

이 책을 쓰는 시점에서 Docker 사용자들은 컨테이너별로 디스크를 제한하는 방법으로 디바이스 매퍼devicemapper 드라이버를 사용합니다. 물론 현재도 공식적으로 디스크 사용량을 제한하는 방법을 개발 중이고, 언젠가는 기능이 추가될 것입니다.

NOTE

디바이스 매퍼 드라이버는 볼륨 관리를 위한 리눅스 커널 프레임워크로, 블록 디바이스를 높은 수준에서 가상화된 블록 디바이스로 매핑하는 기능을 합니다.

디바이스 매퍼 드라이버는 두 가지 블록 디바이스를 이용하여 하나의 씬 풀thin pool을 생성합니다. 두 가지 블록 디바이스는 데이터와 메타 데이터를 위해 사용되며 기본으로는 마운트할 때 스파스 파일sparse file들을 루프백loopback 디바이스로 생성됩니다.

> NOTE
>
> 스파스 파일은 대부분 빈 공간으로 이루어진 하나의 파일입니다. 따라서 100GB가 할당된 스파스 파일이 있더라도 실제로는 파일의 시작과 끝 부분에만 일부 데이터가 있고, 나머지는 모두 비어 있습니다(사용되지 않은 공간은 실제로 할당되지 않습니다). 그런데도 겉으로는 100GB 모두를 사용하고 있는 것처럼 보입니다. 스파스 파일을 읽을 때 파일 시스템은 공간을 마치 0으로 초기화된 블록을 읽는 것과 같이 변환합니다. 또한, 스파스 파일은 메타 데이터를 통해 파일이 쓰인 부분과 그렇지 않은 부분을 추적할 수 있습니다.
>
> 유닉스와 같은 운영체제에서 루프백 디바이스란 하나의 파일을 블록 디바이스로 만든 가상 디바이스를 뜻합니다.

씬 풀이라 불리는 이유는 실제로 데이터를 쓰기 전까지는 공간을 할당하지 않고 표시만 해두기 때문입니다. 각각의 컨테이너는 정해진 크기의 기본 씬 디바이스를 할당받고, 컨테이너는 이 크기를 넘어서는 데이터를 쓸 수 없습니다.

기본으로 사용할 수 있는 씬 풀의 크기는 100GB입니다. 하지만 씬 풀을 구성하는 루프백 디바이스로 스파스 파일이 사용되므로 실제로 100GB가 할당되지는 않습니다.

각각의 컨테이너와 이미지를 위해 할당되는 디바이스 크기는 10GB입니다. 물론 앞의 이유와 마찬가지로 실제로 10GB가 할당되지는 않습니다. 하지만 물리적으로 사용되는 블록 크기가 점점 커져 디바이스 블록에 할당된 크기를 넘어서게 되면, 디바이스 블록은 더 많은 공간이 필요하다고 인식하고 크기를 점점 늘릴 것입니다.

그렇다면 이 값들을 어떻게 변경할 수 있을까요? Docker 데몬을 실행할 때 `--storage-opts` 플래그를 통해 스토리지 드라이버 옵션을 변경할 수 있습니다.

NOTE

이번 절에서 설명하는 명령어들을 실행하기 전에 모든 이미지를 save와 stop 명령어로 백업하기 바랍니다. 또한, Docker가 이미지 파일을 저장하는 공간인 /var/lib/docker를 완전하게 비워두어 변경된 스토리지 드라이버로 인해 일어날 수 있는 충돌을 방지할 수 있습니다.

디바이스 매퍼는 다음과 같이 다양한 설정을 제공합니다.

- **dm.basesize** 이 옵션은 컨테이너와 이미지가 기본으로 사용할 디바이스 크기를 지정하는데, 이 크기는 10GB로 지정됩니다. 앞에서 설명했듯이 이 크기는 물리적으로 데이터를 쓰기 전까지는 할당되지 않으며 최대 크기를 채우기 전까지 사용할 수 있습니다. 사용법은 다음과 같습니다.

```
$ docker -d -s devicemapper --storage-opt dm.basesize=50G
```

- **dm.loopdatasize** 이 옵션은 최대로 사용할 수 있는 씬 풀(디스크 크기)을 지정합니다. dm.basesize 옵션과 마찬가지로 실제로 데이터를 쓰기 전까지는 할당되지 않으며 최대 크기를 채우기 전까지 사용할 수 있습니다. 사용법은 다음과 같습니다.

```
$ docker -d -s devicemapper --storage-opt dm.loopdatasize=1024G
```

- **dm.loopmetadatasize** 앞에서 언급했듯이 씬 풀은 두 가지의 블록 디바이스로 구성됩니다. 하나는 데이터를 위한 블록이고, 다른 하나는 메타 데이터를 위한 블록으로 사용합니다. 이 옵션은 메타 데이터를 위한 블록 크기를 지정할 때 사용합니다. 기본으로는 2GB를 할당하며 다른 옵션과 마찬가지로 스파스 파일을 사용하므로 실제로는 사용한 만큼만 할당됩니다. 메타 데이터를 위한 블록 크기는 대부분 전체 풀 크기의 1% 정도를 사용하는 편이 좋습니다. 사용법은 다음과 같습니다.

```
$ docker -d -s devicemapper --storage-opt dm.loopmetadatasize=10G
```

- **dm.fs** 이 옵션은 기본 디바이스가 사용할 파일 시스템 타입을 지정합니다. ext4와 xfs 파일 시스템을 지원하며, 기본으로는 ext4를 사용합니다. 사용법은 다음과 같습니다.

```
$ docker -d -s devicemapper --storage-opt dm.fs=xfs
```

- **dm.datadev** 이 옵션은 루프백 디바이스 대신 씬 풀에서 사용할 블록 디바이스를 지정합니다. 이 옵션을 사용할 경우에는 루프백 디바이스를 완벽하게 제외하기 위해 메타 데이터를 위한 블록 디바이스까지 같이 지정하는 것이 좋습니다. 사용법은 다음과 같습니다.

```
$ docker -d -s devicemapper --storage-opt dm.datadev=/dev/sdb1
--storage-opt dm.metadatadev=/dev/sdc1
```

이곳에 소개되지 않은 디바이스 매퍼 옵션들과 이들이 어떻게 작동하는지에 대한 설명은 https://github.com/docker/docker/blob/master/daemon/graphdriver/devmapper/README.md에서 살펴볼 수 있습니다. 이 외에 볼만한 문서로는 Docker 컨트리뷰터Contributor중 한 명인 제롬 페타조니Jerome Petazzoni가 작성한 '디바이스 매퍼를 이용해 Docker 컨테이너 크기를 조절하는 방법[01]'이 있습니다.

> NOTE
>
> 스토리지 드라이버를 변경하면 변경하기 전에 생성된 이미지와 컨테이너는 더 이상 보이지 않게 됩니다.

이 장의 첫 부분에서 AUFS는 일반적으로 디스크 크기를 제한할 수 없다고 했습니다. 물론 디스크 크기를 제한할 수 있는 특별한 방법이 있습니다. 이 방법은 ext4 파일 시스템에 기반한 루프백 파일 시스템을 생성하여 볼륨을 마운트하는 과정을 통해 컨테이너가 제한된 크기의 디스크만을 사용하도록 합니다. 명령어는 다음과 같습니다.

```
$ DIR=$(mktemp -d)
$ DB_DIR=$(mktemp -d)
$ dd if=/dev/zero of=$DIR/data count=102400
$ yes | mkfs -t ext4 $DIR/data
$ mkdir $DB_DIR/db
$ sudo mount -o loop=/dev/loop0 $DIR/data $DB_DIR
```

이제 다음과 같이 run 명령어로 컨테이너를 실행할 때 -v를 이용해 $DB_DIR로 생성된 디렉터리를 컨테이너에서 사용할 볼륨으로 지정할 수 있습니다.

```
$ docker run -v $DB_DIR:/var/lib/mysql mysql mysqld_safe
```

01 http://jpetazzo.github.io/2014/01/29/docker-device-mapper-resize/

3.2 컨테이너 내부 데이터 관리

Docker에서 사용할 수 있는 볼륨은 다음과 같은 특징이 있습니다.

- 볼륨은 컨테이너의 루트 파일 시스템으로부터 분리된 하나의 디렉터리입니다.
- 볼륨은 Docker 데몬에 의해 직접 관리되며 컨테이너끼리 공유가 가능합니다.
- 볼륨은 호스트에 존재하는 디렉토리를 컨테이너 내부에 마운트할 때 사용할 수 있습니다.
- 실행 중인 컨테이너가 이미지로 업데이트될 때 볼륨은 업데이트에 포함되지 않습니다.
- 볼륨은 컨테이너와는 별도로 이루어진 파일 시스템이기 때문에 이미지를 구성하는 레이어에 포함되지 않습니다. 따라서 볼륨을 읽거나 쓰는 행위는 레이어를 통하지 않고 직접 이루어집니다.
- 여러 컨테이너가 하나의 볼륨을 사용하고 있다면 해당 볼륨은 모든 컨테이너가 종료되기 전까지 유지됩니다.

볼륨을 생성하는 방법은 매우 간단합니다. -v 옵션을 주면 볼륨을 생성할 수 있습니다.

```
$ docker run -d -p 80:80 —name apache-1 -v /var/www apache
```

볼륨은 컨테이너 또는 이미지처럼 식별이 가능한 ID가 없습니다. 또한, 컨테이너가 종료될 때 더는 참조되지 않는 볼륨은 소멸하게 됩니다. 이번 절에서는 이러한 상황을 방지하기 위해 데이터 전용 컨테이너를 생성하는 방법을 설명합니다.

> NOTE
>
> Docker 1.1 버전부터는 -v 옵션을 통해 호스트의 파일 시스템 전체를 컨테이너로 연결할 수 있게 되었습니다. 사용법은 다음과 같습니다.
>
> ```
> $ docker run -v /:/<마운트 경로> ubuntu:ro ls /<마운트 경로>
> ```
>
> 하지만 보안상의 이유로 컨테이너의 루트(/)경로로 마운트하는 것은 허용되지 않습니다.

3.2.1 데이터 전용 컨테이너

데이터 전용 컨테이너는 아무런 기능 없이 다른 컨테이너가 사용할 수 있는 볼륨

을 제공하는 컨테이너로, 볼륨이 더는 사용되지 않을 때 소멸하는 것을 방지하기 위해 사용합니다.

3.2.2 다른 컨테이너에서 볼륨 사용하기

컨테이너를 실행할 때 -v 옵션을 통해 볼륨을 생성할 수 있지만, --volumes-from 옵션을 통해 이미 생성된 볼륨을 사용할 수도 있습니다. 이 옵션을 사용할 만한 상황으로는 데이터베이스 백업, 로그 기록, 유저 데이터가 필요한 경우 등이 있습니다.

3.2.3 활용 사례 – Docker에서 MongoDB 사용하기

볼륨을 다른 컨테이너에서 사용해볼 수 있는 사례로는 MongoDB가 있습니다. 상용 단계에서 MongoDB를 사용하려면 MongoDB 뿐만 아니라 cron을 이용한 자동화, 주기적인 데이터 백업과 같은 부가적인 기능들이 필요합니다.

> NOTE
>
> MongoDB는 고성능과 고가용성, 편리한 확장을 제공하는 문서 기반[Document-Oriented] 데이터베이스입니다. 자세한 정보는 http://www.mongodb.org에서 살펴볼 수 있습니다.

다음은 Docker 볼륨으로 MongoDB를 설정하는 방법입니다.

1. 가장 먼저 데이터 전용 컨테이너가 필요합니다. 이 컨테이너는 오직 MongoDB가 데이터를 저장하기 위한 볼륨을 유지하는 기능만 수행할 뿐입니다.

```
$ docker run -v /data/db --name data-only mongo echo "MongoDB stores all its data in /data/db"
```

2. 다음으로 생성된 볼륨을 사용할 MongoDB 서버를 실행합니다.

```
$ docker run -d --volumes-from data-only -p 27017:27017 --name mongodb-server mongo mongod
```

NOTE

mongod 명령어는 MongoDB 서버를 실행하며 일반적으로는 데몬/서비스 형태로 실행됩니다. MongoDB는 27017 포트로 접근할 수 있습니다.

3. 마지막으로 생성할 컨테이너는 MongoDB가 실행된 서버로 연결하여 데이터를 백업한 뒤 호스트에서 명령어를 실행한 위치로 저장하는 기능을 합니다.

```
$ docker run --rm --name mongo-backup --link mongodb-server:server -v
$(pwd):/backup mongo bash -c 'mkdir -p /backup && cd /backup && mongodump
--host=server'
```

NOTE

앞에서 살펴본 MongoDB의 활용 사례는 상용 서버에 적용하기에 완벽하지 않습니다. 여러분은 아마 MongoDB의 서버 상태를 주기적으로 살펴볼 수 있는 프로세스가 필요하거나 마지막 예제와 같이 다른 컨테이너에서 MongoDB 컨테이너에 접근할 수 있게 만들어주어야 합니다(이 부분은 3.5 컨테이너 연결에서 다시 설명하겠습니다).

3.3 스토리지 드라이버 설정

Docker에서 사용할 스토리지 드라이버를 변경하기 전에 여러분이 가지고 있는 모든 이미지를 save와 stop 명령어로 백업해두기 바랍니다. 모든 이미지를 백업한 후 /var/lib/docker 디렉터리 안에 있는 모든 파일을 삭제하면 스토리지 드라이버를 변경할 모든 준비가 끝납니다. 백업해둔 이미지들은 스토리지 드라이버가 변경된 이후 다시 복원할 수 있습니다.

3.3.1 디바이스 매퍼를 스토리지 드라이버로 사용

스토리지 드라이버를 디바이스 매퍼로 변경하는 방법은 매우 간단합니다. Docker 데몬을 시작할 때 -s 옵션으로 디바이스 매퍼를 지정하기만 하면 됩니다.

추가로 --storage-opts에서 사용하는 플래그를 통해 다양한 디바이스 매퍼 옵션을 지정할 수 있습니다. 이와 관련된 설명은 3.1.3 디바이스 매퍼를 활용한 디스크 제한을 참고하기 바랍니다.

> NOTE
>
> AUFS를 기본으로 지원하지 않는 레드햇/페도라 계열의 리눅스를 사용 중이라면 Docker는 자동으로 디바이스 매퍼를 기본 스토리지 드라이버로 사용합니다.

스토리지 드라이버를 변경한 후 다음 명령어로 제대로 설정되었는지 확인할 수 있습니다.

```
docker info
```

3.3.2 Btrfs를 스토리지 드라이버로 사용하기

Btrfs^B-tree file system^를 스토리지 드라이버로 사용하려면 먼저 설치부터 시작해야 합니다. 이번에 소개할 예제는 우분투 14.04 운영체제를 사용하고 있다고 가정하고 진행합니다. 하지만 다른 배포 버전의 리눅스에서 사용하는 명령어와 비슷하므로 큰 차이는 없습니다.

다음은 블록 디바이스를 Btrfs 파일 시스템으로 변경하기 위한 단계입니다.

1. 가장 먼저 Btrfs를 사용하기 위해 필요한 패키지들을 설치합니다.

   ```
   # apt-get install -y btrfs-tools
   ```

2. 그다음은 Btrfs 파일 시스템 타입의 블록 디바이스를 생성할 차례입니다. 이 책에서는 Btrfs 파일 시스템을 사용하기 위한 예제로 1GB의 루프백 디바이스를 사용합니다.

   ```
   # dd if=/dev/zero of=/tmp/btrfs-vol0.img bs=1G count=1
   # losetup /dev/loop0 /tmp/btrfs-vol0.img // 사용이 끝나면 umount /dev/loop0
   를 통해 장치를 제거합니다.
   # mkfs.btrfs /dev/loop0
   ```

3. 만약 /var/lib/docker를 삭제했다면 디렉터리를 생성합니다(이미지를 백업하지 않은 상태라면 백업을 먼저 한 후 삭제해야 합니다).

```
# mkdir /var/lib/docker
```

4. 이제 Btrfs 블록 디바이스를 /var/lib/docker로 마운트합니다.

```
# mount /dev/loop0 var/lib/docker
```

5. 마운트가 성공했는지 확인합니다.

```
$ mount ¦ grep btrfs
/dev/loop0 on /var/lib/docker type btrfs (rw)
```

이제 -s 옵션과 함께 Docker를 시작하면 됩니다.

```
$ docker -d -s btrfs
```

스토리지 드라이버를 변경한 후 제대로 설정되었는지 다음 명령어로 확인할 수 있습니다.

```
docker info
```

3.4 Docker 네트워크 설정

Docker는 컨테이너별로 독립된 네트워크 환경과 'docker0'라는 가상 인터페이스를 생성하여 컨테이너 간 네트워크 통신과 호스트와 컨테이너 간 네트워크 통신을 관리합니다.

Docker는 run 명령어로 컨테이너를 실행할 때 설정할 수 있는 몇 가지 네트워크 설정이 있습니다.

- **--dns** http://www.docker.io와 같은 URL을 IP 주소로 식별할 수 있는 DNS 서버를 선택합니다.
- **--dns-search** DNS 검색 서버를 선택합니다.

NOTE

http://superuser.com/a/184366에 있는 내용을 참고하면 DNS 검색 서버를 사용할 때는 도메인의 모든 주소를 입력하지 않아도 됩니다. 예를 들어, example.com이 DNS 검색 도메인으로 지정되면 브라우저에 abc를 입력하는 것만으로 abc.example.com에 해당하는 IP 주소를 가져올 수 있습니다. DNS 검색 서버는 많은 서브 도메인이 있는 웹 사이트에 자주 접속할 때 유용하게 사용됩니다. 매번 모든 URL을 입력하는 것은 고통스럽기까지 하므로 검색 도메인으로 판단하기 힘든 URL(xyz.ab.com과 같은)을 입력할 경우에는 DNS 검색 서버가 이 URL을 판단하여 검색 도메인으로 추가합니다.

- **-h 또는 --hostname** 호스트 이름을 설정합니다. 이곳에 지정된 이름은 컨테이너의 /etc/hosts 경로에 컨테이너의 IP와 함께 등록됩니다.
- **--link** 컨테이너를 시작할 때 줄 수 있는 또 다른 옵션 중 하나입니다. 이 옵션은 다른 컨테이너의 IP 주소를 몰라도 통신할 수 있게 연결해 주는 기능을 합니다.
- **--net** 컨테이너에 네트워크 모드를 설정합니다. 이 옵션은 다음 4가지 값을 가질 수 있습니다.

  bridge docker : 브릿지 위에서 새로운 네트워크를 구성합니다.
  none : 네트워크를 구성하지 않습니다. 따라서 완벽하게 격리된 상태가 됩니다.
  container:〈이름¦ID〉 : 다른 컨테이너의 네트워크 구성을 사용합니다.
  host : 호스트의 네트워크를 사용합니다.

NOTE

호스트의 네트워크를 사용한다는 것은 곧 컨테이너에서 호스트의 모든 서비스에 접근할 수 있다는 뜻이므로 이 옵션을 사용하면 보안이 취약해집니다.

- **--expose** 호스트에서 컨테이너로 접근할 수 있는 포트를 지정합니다.
- **--publish-all** 호스트에서 컨테이너로 접근할 수 있는 모든 포트를 외부로 노출합니다.
- **--publish** 다음 형식으로 컨테이너로 접근할 수 있는 포트를 외부로 노출합니다.

```
IP:호스트_포트:컨테이너_포트 ¦ IP::컨테이너_포트 ¦ 호스트_포트:컨테이너_포트 ¦ 컨테이너_포트
```

NOTE

--dns 또는 --dns-search가 옵션으로 주어지지 않을 경우 컨테이너의 /etc/resolv.conf 파일은 Docker 데몬이 동작하는 호스트의 dns 설정과 동일하게 변경됩니다.

물론 이 외에도 Docker 데몬을 시작할 때 설정할 수 있는 네트워크 옵션들이 있으며 다음과 같은 형식을 가지고 있습니다.

> NOTE
>
> 이 옵션들은 -d를 사용하여 Docker 데몬을 시작할 때만 설정할 수 있으며 한 번 실행한 후로는 변경할 수 없습니다.

- **--ip** 호스트가 사용하는 가상 인터페이스인 docker0의 IP를 변경합니다. 따라서 외부로 노출할 컨테이너 포트들은 이 옵션으로 변경된 IP를 통해 접근할 수 있습니다. 이 옵션은 다음과 같이 사용할 수 있습니다.

```
$ docker -d --ip 172.16.42.1
```

- **--ip-forward** 이 옵션은 참 또는 거짓만을 지정할 수 있습니다. 거짓으로 지정된 경우 Docker 데몬은 컨테이너 간의 패킷 또는 외부로부터 오는 모든 패킷을 받지 않습니다. 따라서 사설 네트워크 망과 같은 격리된 네트워크가 생성됩니다.

> NOTE
>
> 이 옵션은 sysctl 명령어로 확인할 수 있습니다.
>
> ```
> $ sysctl net.ipv4.ip_forward
> net.ipv4.ip_forward = 1
> ```

- **--icc** 이 옵션도 참 또는 거짓으로 지정하며 내부 컨테이너와의 통신을 허용할지 결정합니다. 거짓으로 설정하면 각각의 컨테이너는 서로 연결될 수 없으며 독립적인 환경이 구성됩니다. 하지만 HTTP 요청과 같은 표준 프로토콜을 통해 연결은 가능합니다.

> NOTE
>
> 컨테이너 간 통신이 필요할 때만 활성화하는 방법은 3.5 컨테이너 연결을 참고하기 바랍니다.

- **-b 또는 --bridge** Docker가 사용하는 인터페이스(docker0) 대신 지정된 브릿지 인터페이스를 사용하게 합니다. 브릿지 생성은 이 책에서 설명하진 않지만 관심이 있다면 http://docs.docker.com/articles/networking/#building-your-own-bridge 에서 자세한 정보를 얻을 수 있습니다.

- **-H 또는 --host** Docker는 기본적으로 RESTful API를 사용하고 데몬은 API 서버로 실행됩니다. 따라서 클라이언트에서 ps나 run 같은 명령어를 실행하면 GET, POST 형태의 요청이 서버로 전달되어 처리됩니다. -H 플래그는 Docker 데몬이 어떤 클라이언트로부터 요청을 받을 것인지 지정할 때 사용됩니다. 이 옵션은 동시에 여러 인자를 받을 수 있으며 다음과 같은 형식을 따릅니다.

  TCP 소켓 형식 : tcp://<호스트>:<포트>
  유닉스 소켓 형식 : unix://<소켓 경로>

3.4.1 컨테이너와 호스트 간의 포트포워딩 설정

컨테이너는 특별한 설정 없이 호스트 외부로 통신할 수 있습니다. 하지만 반대로 호스트 밖에서 컨테이너와 통신을 하려면 특별한 설정이 필요합니다. 보안 관점에서 보면 왜 그런지 충분히 이해할 수 있습니다. 모든 컨테이너는 가상 인터페이스를 통해 호스트와 연결되어 있어서 기본적으로 외부와는 격리된 상태이기 때문입니다. 하지만 외부와 내부 컨테이너가 반드시 통신해야만 할 때는 어떻게 해야 할까요?

포트 포워딩은 컨테이너의 포트를 외부로 노출하는 가장 쉬운 방법입니다. 따라서 Dockerfile을 생성할 때에도 노출할 포트를 미리 지정하는 것이 바람직한 선택입니다. 초창기 버전의 Docker는 Dockerfile 안에서 연결할 호스트의 포트를 직접 지정하는 것이 가능했습니다. 하지만 이 방법은 이미 실행 중인 컨테이너를 간섭하는 일이 잦기 때문에 더는 사용할 수 없게 되었습니다. 물론 지금도 Dockerfile 안에서 EXPOSE 항목을 이용해 노출할 포트를 지정할 수는 있습니다. 하지만 원하는 포트를 호스트의 포트와 연결하는 것은 컨테이너를 실행할 때만 가능합니다.

컨테이너를 시작할 때 호스트로 연결할 포트를 지정하는 방법은 다음 두 가지가 있습니다.

- **-P 또는 --publish-all** -P 옵션을 지정하여 컨테이너를 실행하면 이미지가 생성될 때

Dockerfile 안에 지정된 EXPOSE 항목에 있는 모든 포트가 개방됩니다. Docker는 개방된 포트들을 49000~49900 사이의 임의의 호스트 포트로 연결해 외부로 노출합니다.

- **-p 또는 --publish** 이 옵션은 Docker에 어떤 포트를 외부로 노출해야 할지를 다음과 같은 형식으로 직접 지정할 수 있습니다(당연한 이야기겠지만 이 형식에서 지정된 호스트의 IP는 호스트의 인터페이스와 연결된 IP 중 하나여야 합니다).

1. docker run -p IP:호스트_포트:컨테이너_포트
2. docker run -p IP::컨테이너_포트
3. docker run -p 호스트_포트:컨테이너_포트

3.4.2 사설 아이피 설정

앞에서 컨테이너의 포트를 호스트 포트로 연결하는 방법과 컨테이너의 DNS를 설정하는 방법, 심지어 가상 인터페이스의 IP를 바꾸는 방법까지 배웠습니다. 하지만 호스트와 컨테이너 간의 네트워크 서브넷Subnet을 구성해야 할 때는 어떻게 해야 할까요? Docker는 RFC 1918을 따르는 사설 주소 공간 내에서의 가상 IP와 서브넷을 제공합니다.

서브넷을 구성하는 것은 놀라울 만큼 간단합니다. Docker 데몬을 실행할 때 --bip 옵션을 지정하여 생성될 컨테이너가 사용할 가상 인터페이스의 아이피 주소와 함께 서브넷을 지정할 수 있습니다.

다음 명령어는 Docker 데몬을 실행하면서 가상 인터페이스의 IP주소를 192.168.0.1로 지정합니다. 또한, IP 주소와 함께 서브넷 범위(192.168.0.0/24)도 같이 지정합니다. 따라서 가상 인터페이스가 사용할 수 있는 IP의 범위는 192.168.0.2부터 192.168.0.254까지 총 252개가 됩니다.

```
$ docker -d —bip 192.168.0.1/24
```

이 외에도 https://docs.docker.com/articles/networking/에서 더욱 다양한 네트워크 설정들을 볼 수 있으니 한번쯤 둘러보는 것을 추천합니다.