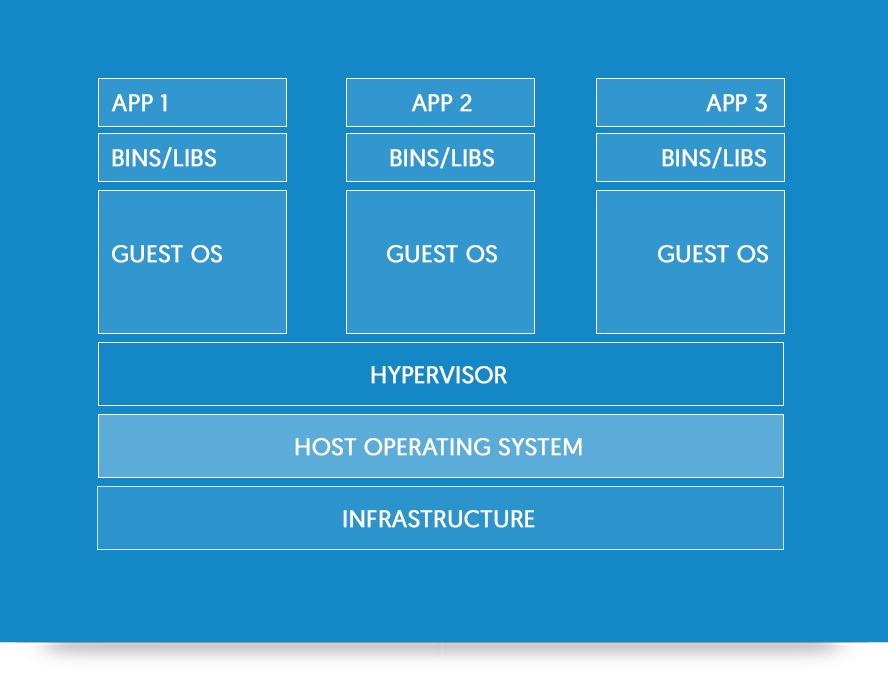
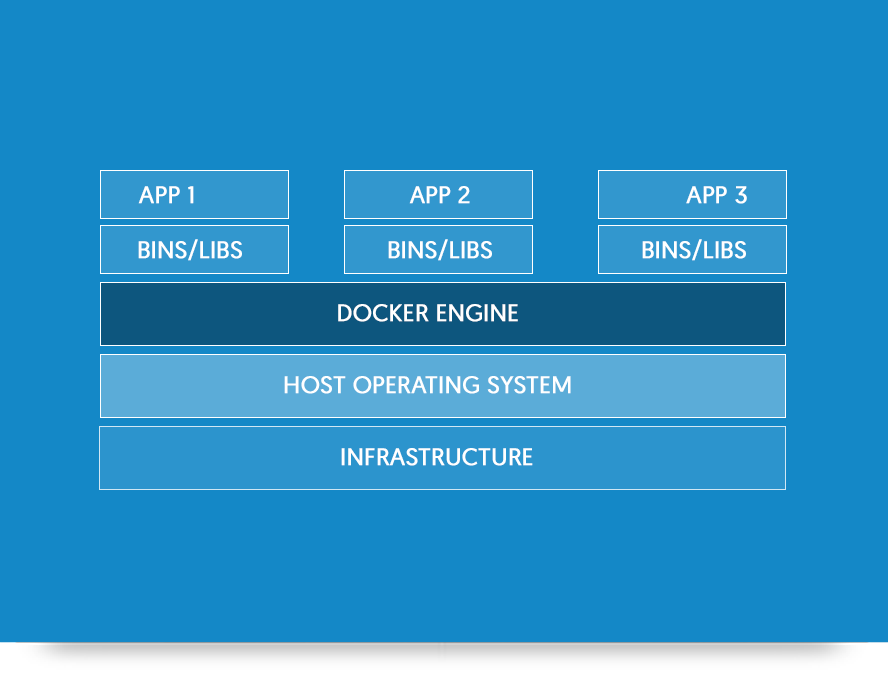
Docker 간보기

목차﻿

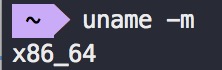
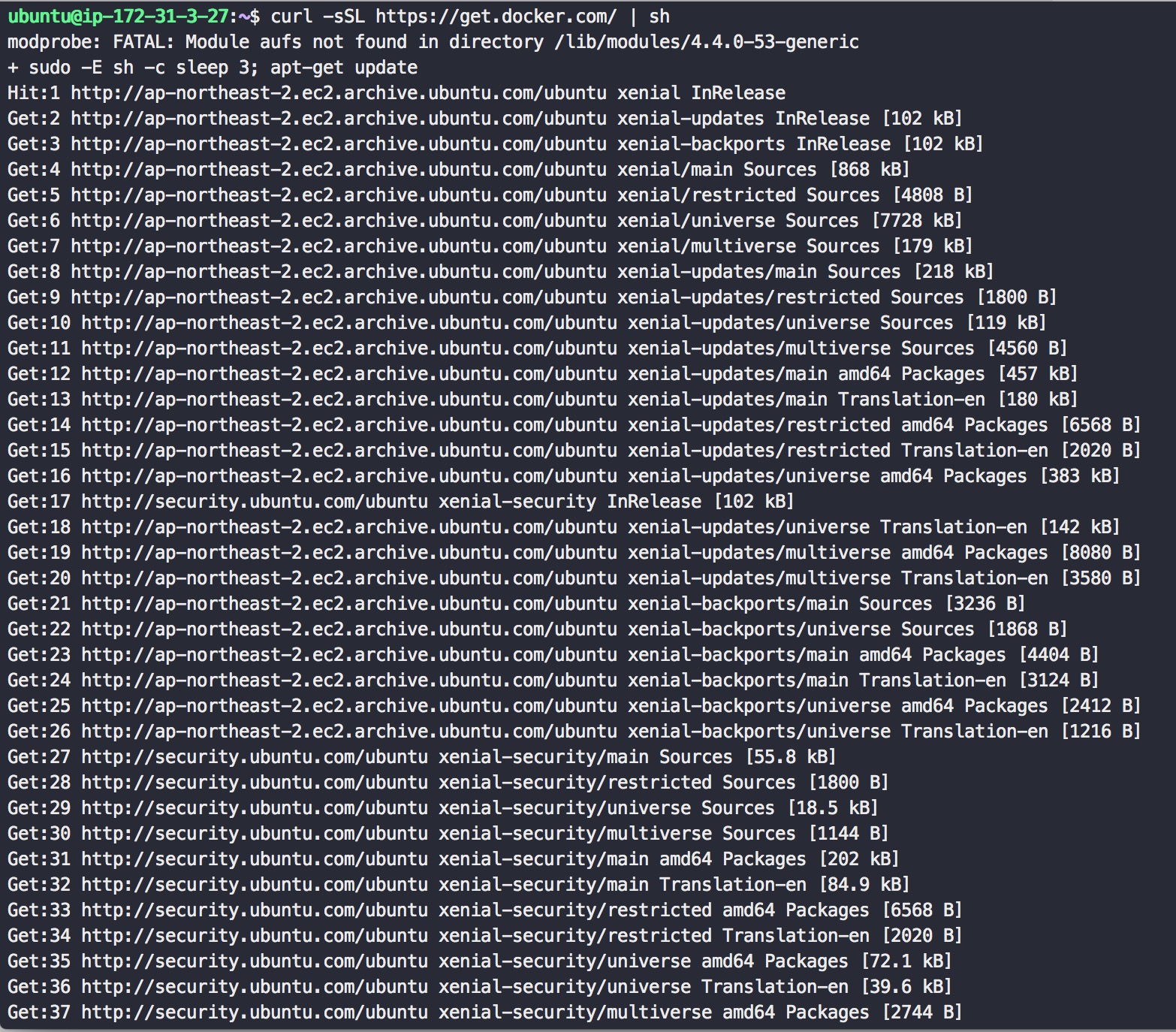
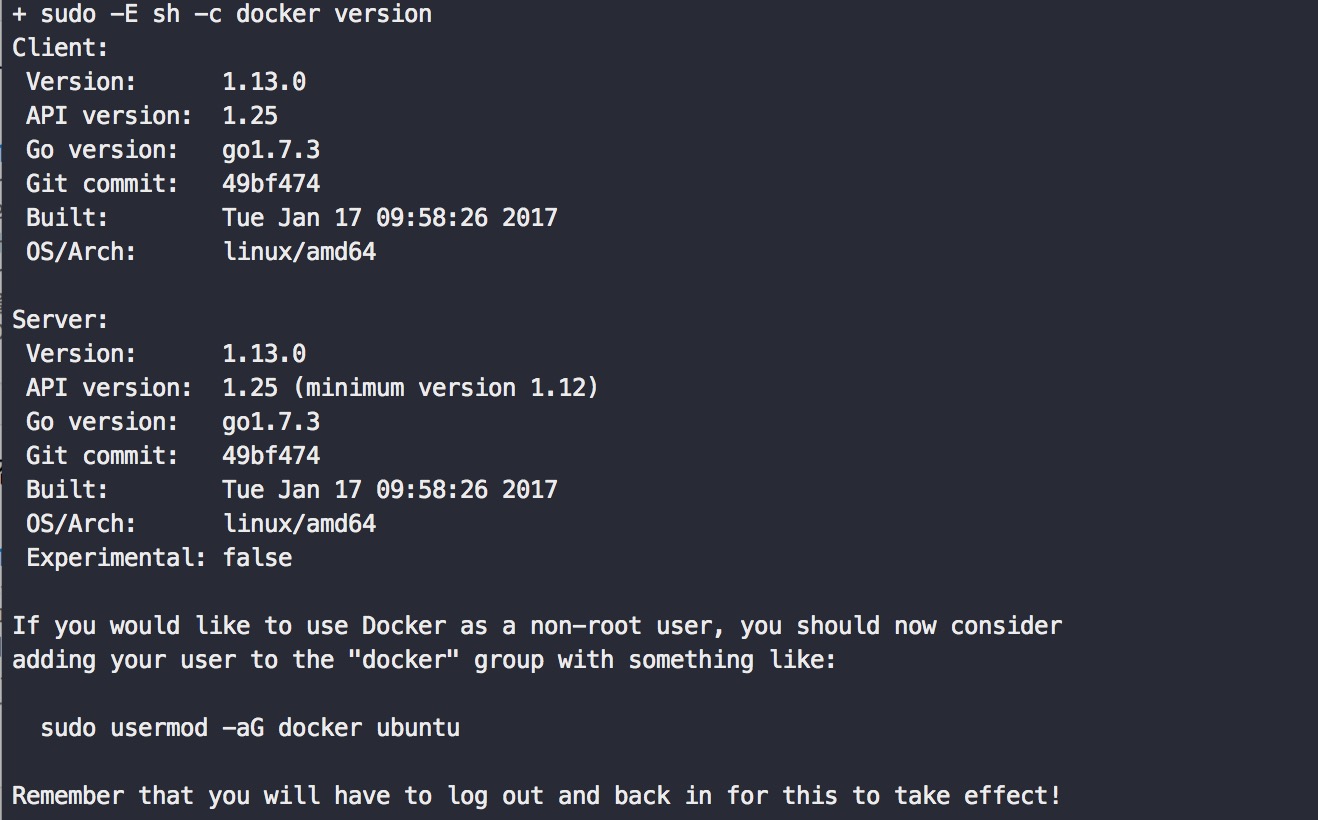
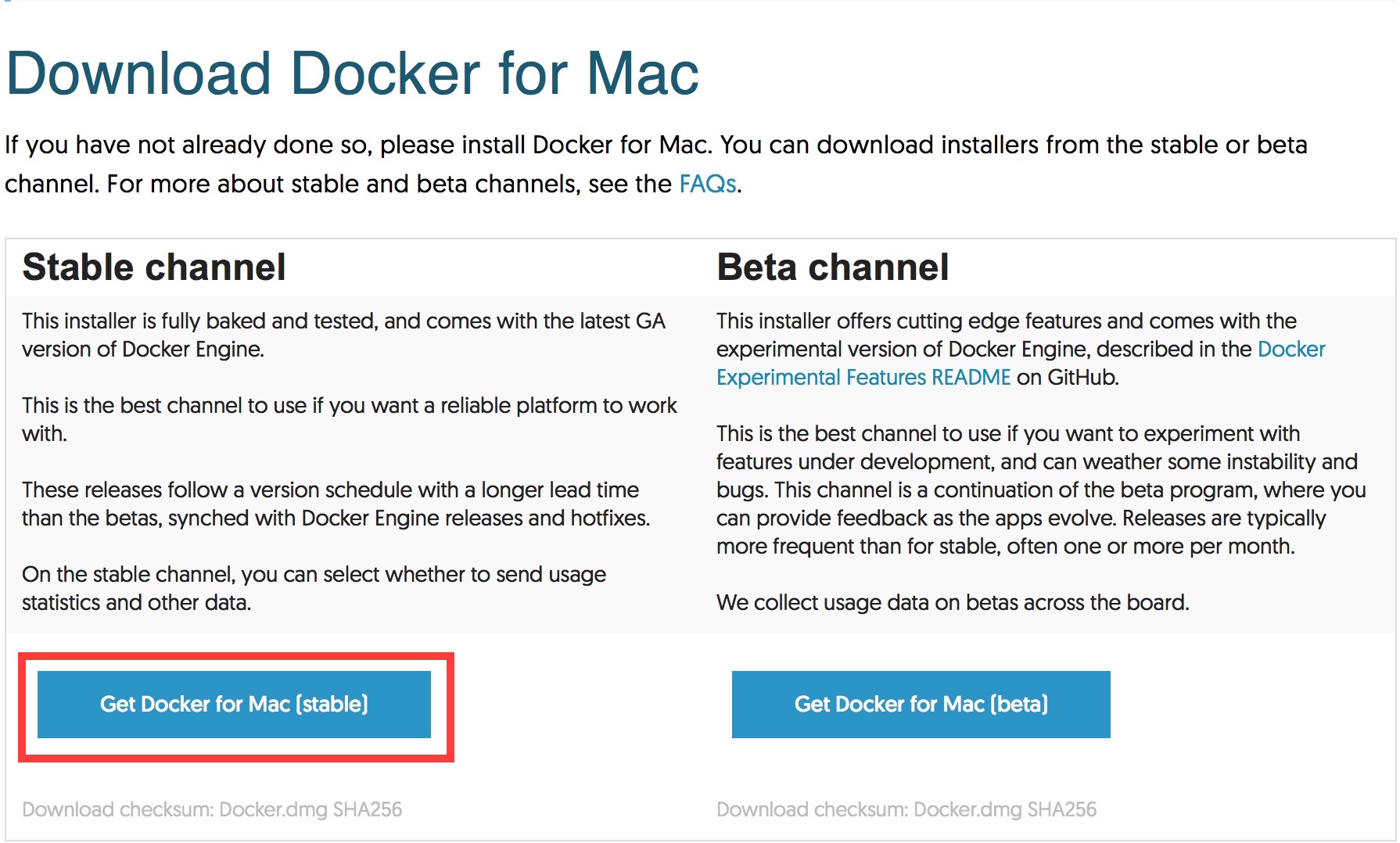
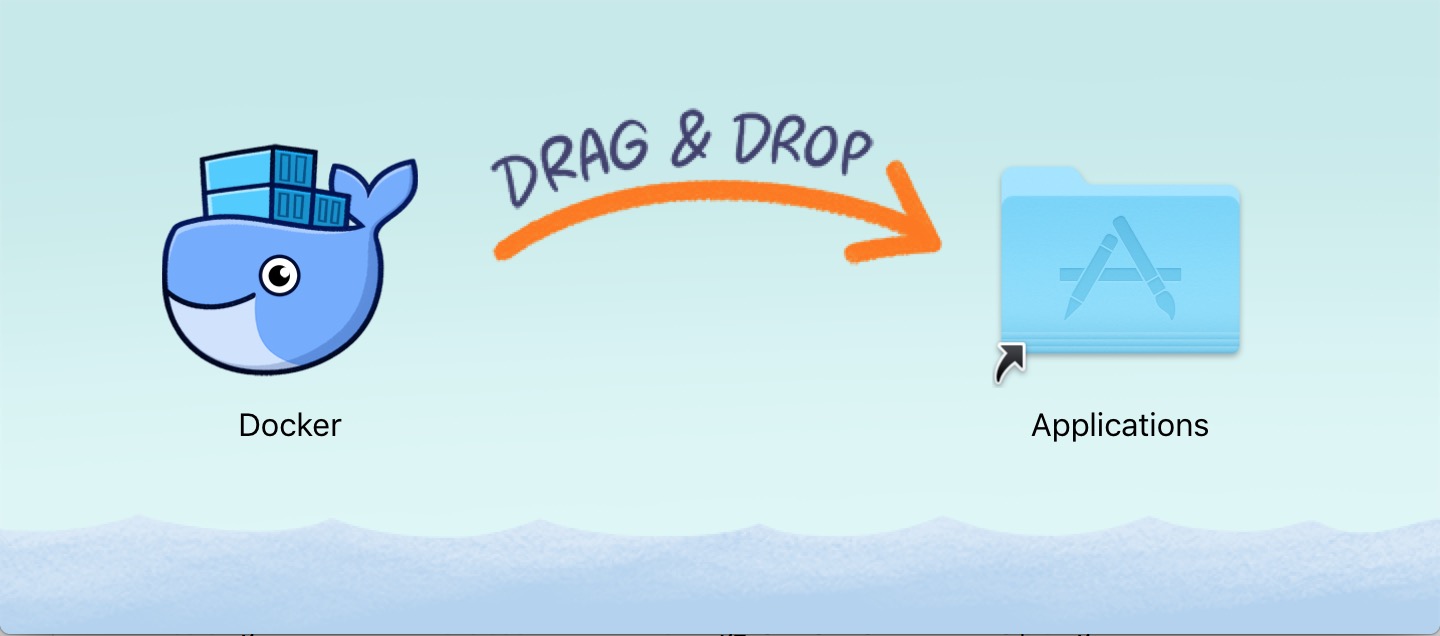
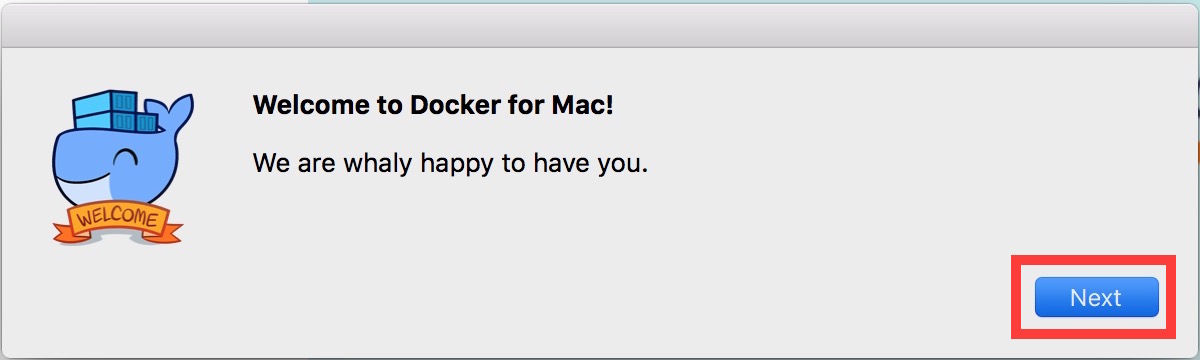
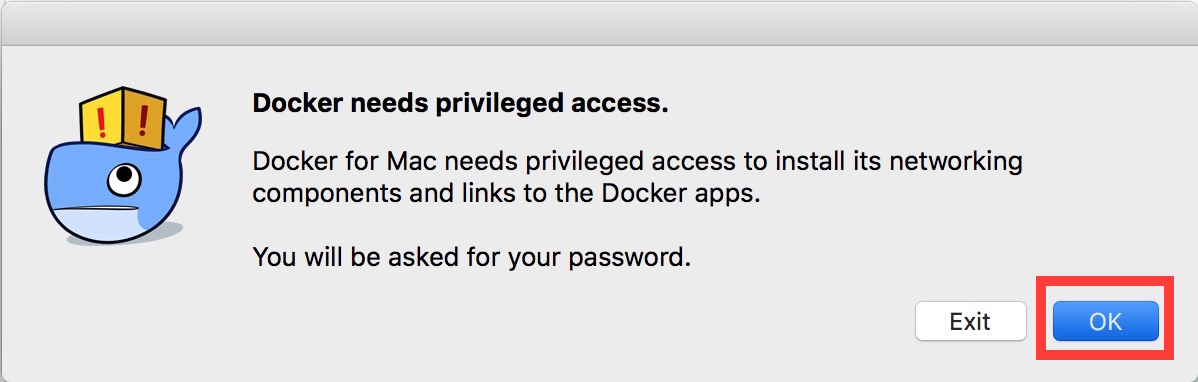
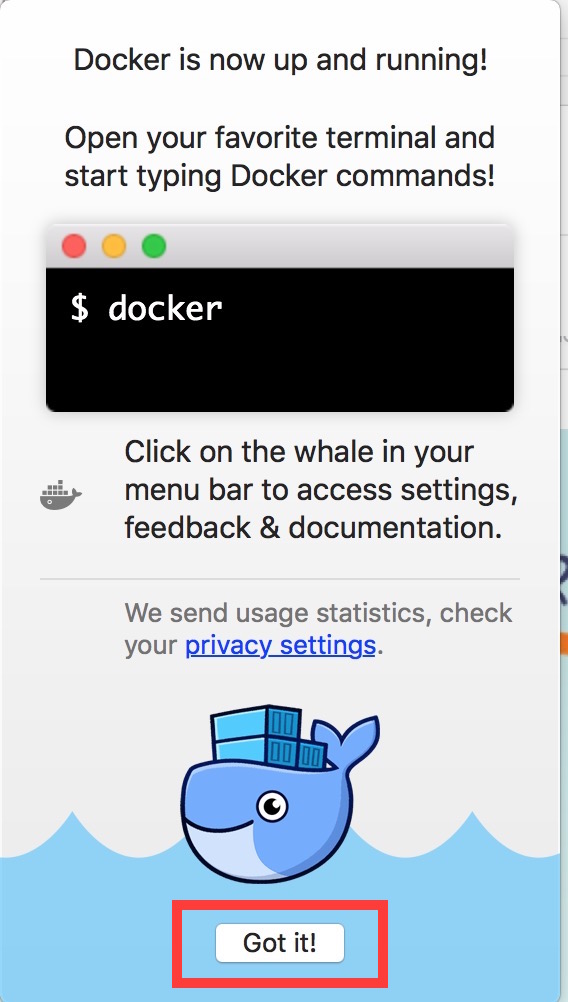
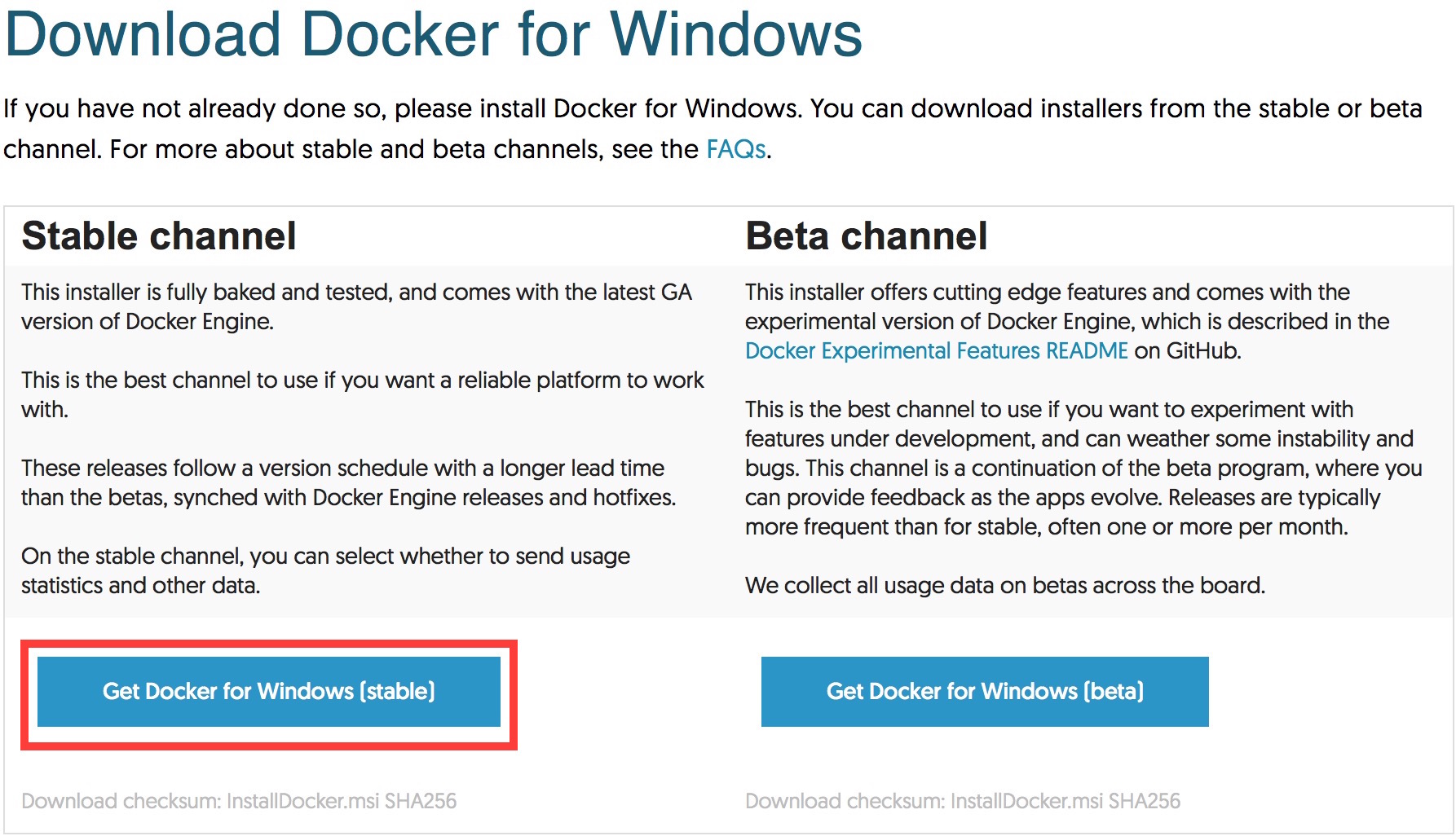
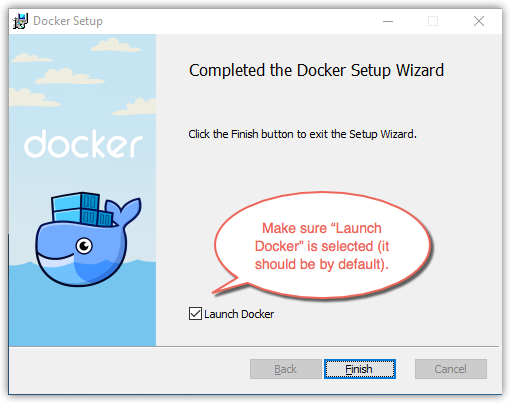
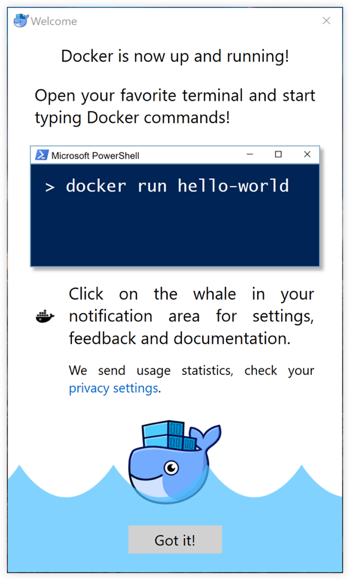
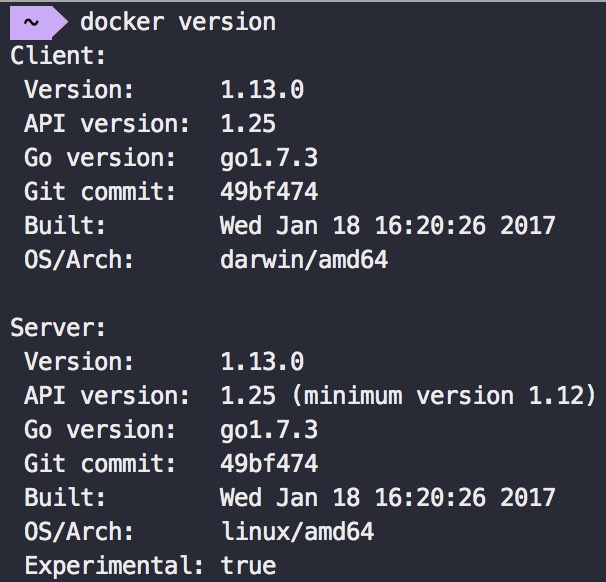
* 컨테이너란 무엇이며 왜 컨테이너인가
* Docker 설치하기
* Docker 실행하기
* Docker의 기초
* Docker 개발환경 세팅 삽질해보기

컨테이너란 무엇이며 왜 컨테이너인가

* 컨테이너
  + 응용프로그램의 종속성과 함께 응용프로그램 자체를 캡슐화한 것
  + 일반적인 VM과 같이 운영체제(OS)의 격리된 인스턴스 형태
  + VM에서는 어렵거나 불가능한 것들을 가능하게 제공
    - 호스트 OS 자원 공유하여 자원을 좀 더 효율적으로 사용
    - 빠른 실행 및 중지
    - 컨테이너에서 실행되는 응용프로그램은 호스트OS에서 네이티브로 실행되는 응용프로그램에 비해 발생시키는 오버헤드는 거의 없음
    - 운영 환경의 변경으로 인해 발생하는 모든 버그를 없앨 수 있음(이식성 = portability)
      * “내 컴퓨터에서는 잘 동작하는데!” 따위 이야기는 할 필요가 없다.
    - 가벼워서 많은 수의 컨테이너를 동시에 실행 가능
      * 운영 엔지니어는 VM 자체를 운영하는 것이 아니라 하나의 호스트 머신에서 여러 개의 컨테이너를 운영
    - 컨테이너는 최종 사용자와 개발자가 클라우드 배포 가능
* VM VS 컨테이너
  + VM = 외부 환경을 완전하게 에뮬레이트하는 목적
    - 
    - 출처 : <https://www.docker.com/what-docker>
    - Hypervisor(Virtualbox, VMWare 등)는 VM들을 생성하고 실행 및 기본 OS와 하드웨어 접근 제어
    - 각 VM들은 OS전체, 실행할 응용프로그램과 모든 라이브러리들을 가지고 있어야함
  + 컨테이너 = 응용프로그램의 이식성과 독립성에 목적
    - 
    - 출처 : <https://www.docker.com/what-docker>
    - VM과 달리 호스트의 커널은 실행되는 컨테이너들 사이 공유 = 컨테이너들은 항상 호스트와 같은 커널에서 실행
  + VM과 컨테이너는 응용프로그램을 같은 호스트에서 실행되는 다른 응용프로그램으로 부터 격리
* Docker와 컨테이너
  + Docker 플랫폼은 두 개의 독립된 컴포넌트로 구성
    - Docker Engine
      * 컨테이너 생성, 실행
      * 빠르고 칸편한 인터페이스 제공
      * Docker 이전 리눅스 컨테이너(LXC) 같은 기술이 있었지만 전문적인 지식 및 작업이 어려워 불편
    - Docker Hub
      * 컨테이너 배포를 위한 클라우드 서비스 제공
      * 다운로드할 수 있는 막대한 양의 컨테이너 이미지 제공
      * 다른 사람들이 이미 작업해 둔 애용 그대로 이용 가능
    - Docker가 개발한 도구 제공
      * 클러스터링 관리자 - Swarm
      * GUI를 이용해 컨테이너 작업 - 카이트매틱
      * Docker Host를 생성하기 위한 명령 유틸리티 - Machine
* Docker의 역사
  + 2008년 솔로몬 하익스(Solomon Hykes)가 언어 중립적인 PaaS를 제공하는 dotCloud 설립(현재는 Docker의 CTO)
  + 
  + 출처 : <https://www.docker.com/company/management>
  + 당시 타사 PaaS는 특정 언어 종속 - Heroku : Ruby, Google App Engine : Java, Python
  + dotCloud 자사 핵심 기술 Docker를 커뮤니티 주도 프로젝트로 전환
  + 6개월만에 github star 6700개, 직원 외 contributor 175명
  + dotClout => Docker 회사명 변경
  + Docker 0.1 버전 => 1.0 버전 15개월(2014년 6월 발표)
  + 이미 당시 Spotify, Baidu 운영계 사용
  + Docker Hub(컨테이너들의 공용 저장소) 출시하며 플랫폼으로 진화 노력
  + 2014년 12월 Docker Swarm 발표
    - Docker와 Docker Machine을 위한 클러스터 관리자
    - Docker host를 프로비저닝(사용자의 요구에 맞게 시스템 자원을 할당, 배치, 배포해 두었다가 필요 시 시스템을 즉시 사용할 수 있는 상태로 미리 준비)하는 CLI도구 제공
  + 2015년 6월 FreeBSD 프로젝트 Docker 지원 발표
  + 2015년 8월 MS 윈도우 서버용 Docker Engine tech preview 버전 공개
* 64-bit linux
  + 현재 기준 안정적이면서 상용화 가능한 Docker Platform은 linux 64bit
  + 윈도우, os x 사용자는 VM 안에서 Docker 실행가능
  + BSD, solaris, 윈도우 서버 등과 같은 다른 플랫폼의 네이티브 컨테이너에 대한 지원은 각기 다른 단계에서 개발 진행
  + Docker는 기본적으로 가상화 관련 작업 하지 않음 그래서 container는 호스트 커널과 항상 일치해야 함

(윈도우 서버 container는 윈도우 서버 호스트에서만 실행, 64bit linux container는 64bit linux 호스트에서만 실행 가능)

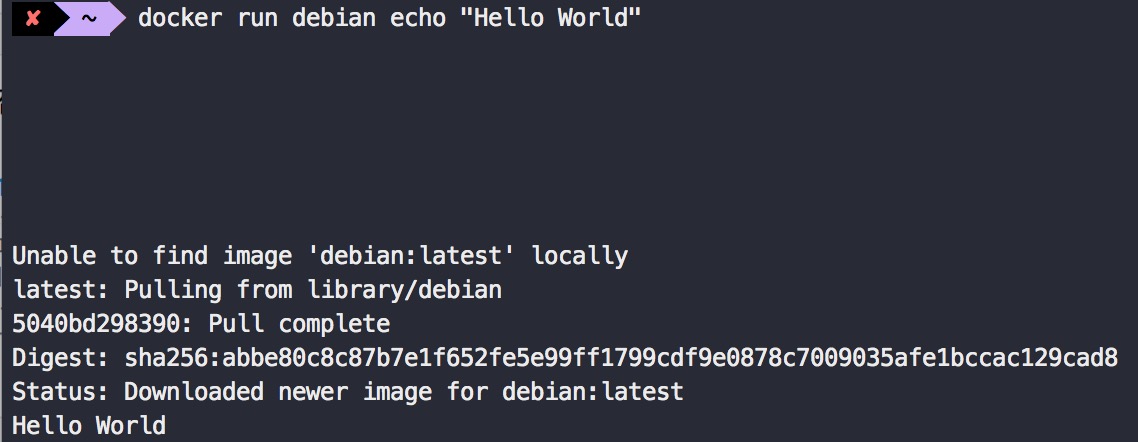
Docker 설치하기

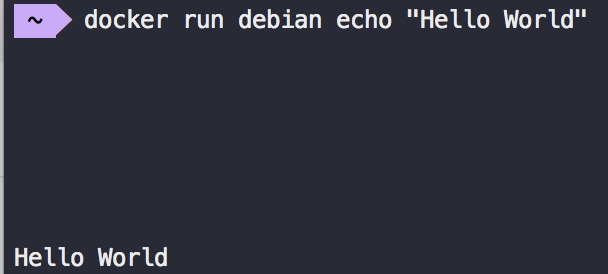
* Linux
  1. 최선의 방법은 Docker에서 제공하는 설치 스크립트 이용
  2. 유명 Linux 배포판들은 자체적으로 Docker Package들을 포함하고 있긴 하지만 Docker update 보다 느림
  3. Docker 실행위해 64bit 필요
     1. uname -m
     2. 
  4. [https://get.docker.com](https://get.docker.com/) 에 자동 설치 스크립트 제공
* curl -sSL <https://get.docker.com/> | sh or
* wget -qO- <https://get.docker.com/> | ti
  1. 
  2. 
  3. RHEL, CentOS, Fedora 포함한 RedHat 기반의 배포판은 SELinux를 허용 모드(permissive mode)로 실행 권장
  4. 강제 모드(enforcing mode)로 실행시 “사용 권한이 없습니다(Permission Denied)”라는 애매한 오류 발생
     1. SELinux을 허용 모드로 변경하려면 sudo setenforce 0 실행
  5. sudo 없이 실행하기
     1. Docker는 권한이 있는 바이러니이기 때문에 실행을 하려면 명령 앞에 sudo를 명시 해야함
     2. 사용자를 Docker Group에 추가하면 sudo 생략 가능
        1. Ubuntu
           1. sudo usermod -a -G docker ubuntu
           2. 상위 명령어 실행하면 docker group 생성하고 현재 사용자를 그룹에 추가
           3. 로그아웃 이후 로그인
           4. sudo service docker restart
* OS X 또는 윈도우
  1. 완전한 형태의 VM 솔루션
  2. Docker Toolbox
     1. boot2docker vm, 텀포즈 스웜 등 포함
  3. OS X의 Homebrew 이용한다면 boot2docker용 brew crcipe 이용하기를 권장
  4. Docker for Mac
     1. <https://docs.docker.com/docker-for-mac/>
     2. Mac 용 Docker는 MMU 가상화를위한 Intel의 하드웨어 지원을 통해 2010 년 또는 최신 Mac에서 실행되는 OS X El Capitan 10.11 이상을 필요로합니다. 앱은 10.10.3 Yosemite에서 실행되지만 제한적으로 지원된다.
     3. 실제 네이티브로 설치 된것 같지만 내부적으로는 [HyperKit](https://github.com/docker/HyperKit/) 가상머신에 설치됨. (Docker for Mac는 VirtualBox가 아니라 MacOS 10.10 이상의 요세미티에서 Hypervisor.framework 위에 구축 된 경량 MacOS 가상화 솔루션인 [HyperKit](https://github.com/docker/HyperKit/) 사용)
     4. 
     5. 다운로드 받은 Docker.dmg 파일을 더블 클릭하여 설치 프로그램을 실행하고 Docker app을 Applications 폴더로 옮긴다.
     6. 
     7. 
     8. 
     9. 
  5. Docker for Windows
     1. <https://docs.docker.com/docker-for-windows/>
     2. Docker for windows는 64 비트 Windows 10 Pro, Enterprise 및 Education (11 월 15 일 업데이트, 빌드 10586 이상) 및 Microsoft [Hyper-V](https://en.wikipedia.org/wiki/Hyper-V)가 필요하다.
     3. 
     4. 
     5. 출처 : <https://docs.docker.com/docker-for-windows/>
     6. 
     7. 출처 : <https://docs.docker.com/docker-for-windows/>
  6. 실행해보기
     1. 터미널을 실행해서 Docker가 정상적으로 실행 가능한지 확인
  7. $docker*—version*
  8. Docker version 1.13.0-rc3, build 4d92237
  9. $docker-compose*—version*
  10. docker-compose version 1.9.0, build 2585387
  11. $docker-machine*—version*
  12. docker-machine version 0.9.0-rc2, build 7b19591
      1. 
  13. docker version
      1. 
      2. 위와 같이 client, server 정보가 보이지 않는 경우는 Docker demon이 실행 중이지 않거나 client에 접근할 수 없는 상황
      3. 문제 해결은 **sudo docker daemon** 으로 Docker Demon을 수동으로 시작

상위 명령어로 실행하면 문제 확인 가능한 정보들을 확인 가능

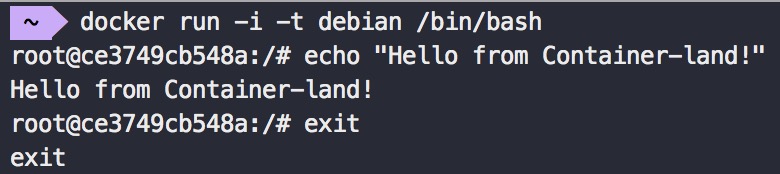
Docker 실행하기

* 이미지 실행하기
  1. Docker가 잘 설치되었는지를 테스트하기 위해서 다음 명령을 실행
* docker run debian echo "Hello World"



* + - 명령어 프로세스
      * docker run 명령은 컨테이너를 시작하는 역할 수행
      * 인수로 사용된 debian은 사용하고자 하는 이미지의 이름(debian은 리눅스 배포판의 기본적인 이름)
      * Unable to find image ‘debian:latest’ locally는 로컬에 Debian 이미지가 없음을 나타내고 Docker는 온라인으로 Docker hub를 확인하고 최신 버전의  Debian 이미지를 다운
      * 다운로드 완료 이후 Docker는 이미지를 실행 상태의 컨테이너로 전환
      * echo “Hello World” 명령어 실행
    - 같은 명령을 다시 실행시 이미지를 다운로드 하지 않고 바로 컨테이너를 실행
    - 
  + docker run -i -t debian /bin/bash

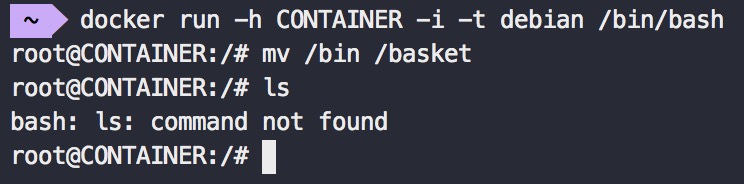
상위 명령어를 이용하면 container 내부에서 쉘(shell)을 수행하도록 Docker 사용



* 기본 명령어
  1. Container에 새로운 호스트 이름 부여
  2. docker run -h CONTAINER -i -t debian /bin/bash

강제로 container를 고장 내보자.

mv /bin /basket



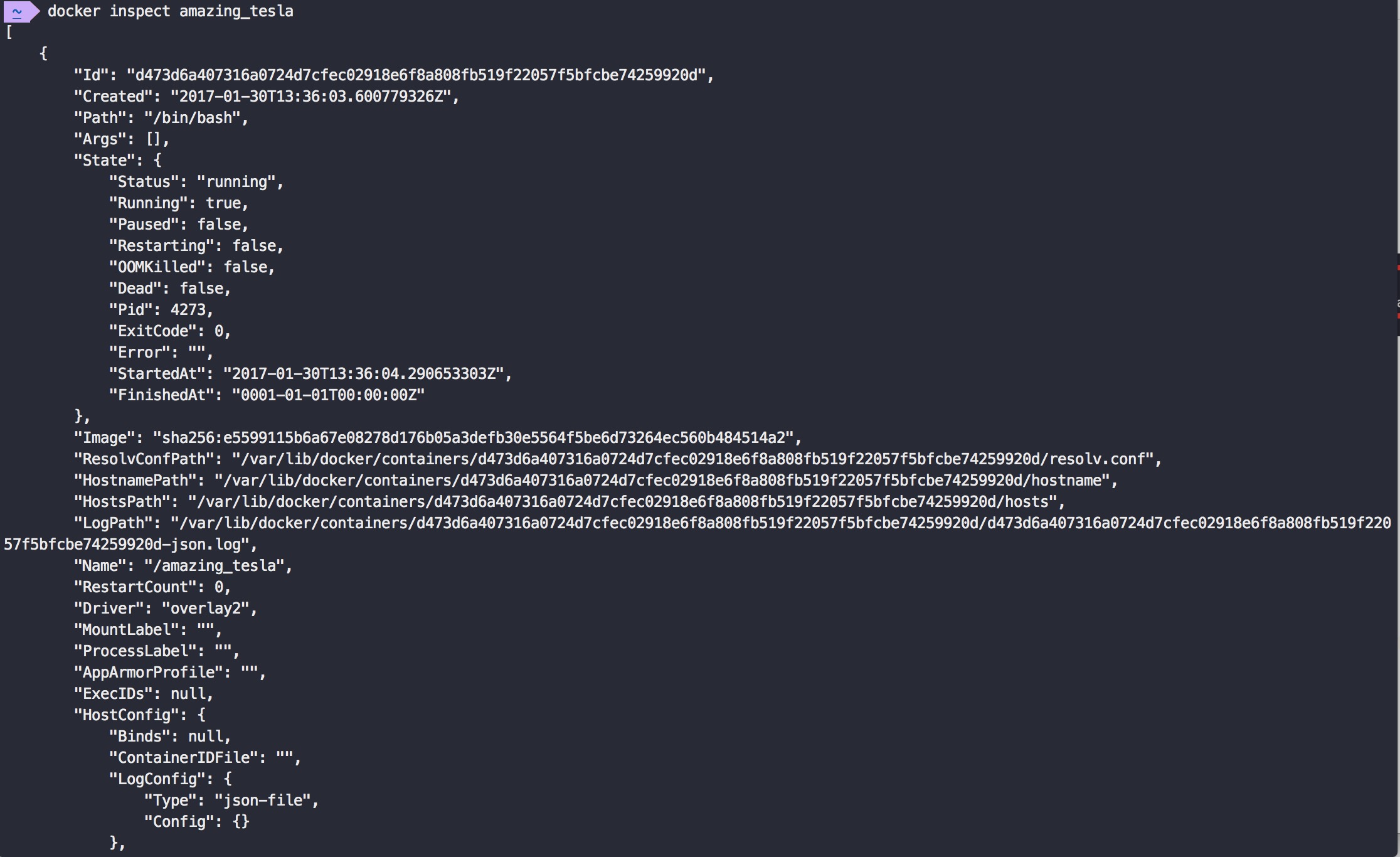
* + 새로운 터미널 세션에서 docker process 를 확인
* docker ps



docker는 host를 구분할 수 있도록 container에 이름을 부여 상위는 "amazing\_tesla” 라는 이름으로 주어졌음

(docker는 임의의 형용사와 유명한 과학자, 엔지니어, 해커 등의 이름을 붙여 호스트 이름 생성 또는 —name 인수로 직접 할당 가능)

* + container의 자세한 정보 확인
* docker inspect amazing\_tesla

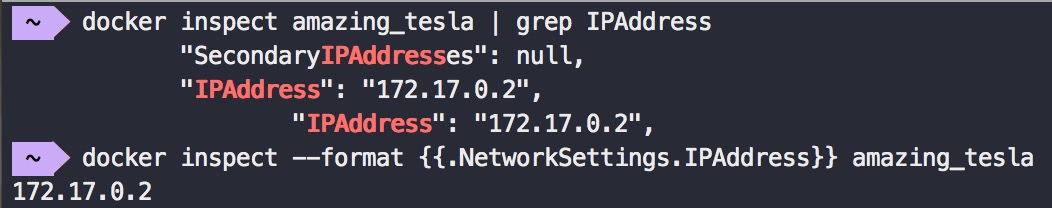


상위 정보에 대한 결과를 grep이나 —format 인수를 이용하여 필터링 가능

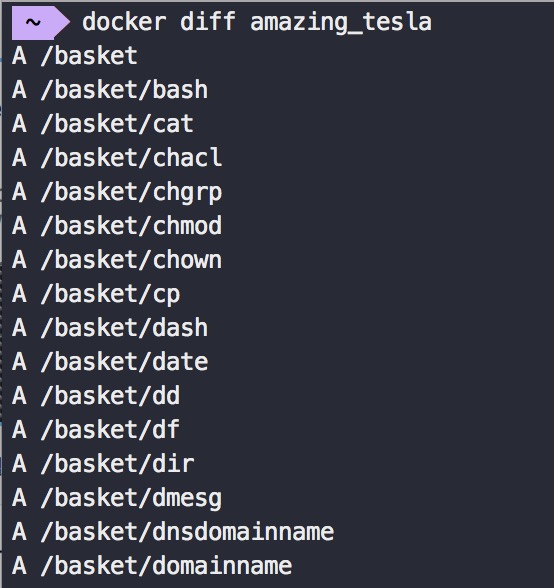
docker inspect amazing\_tesla | grep IPAddress

docker inspect --format {{.NetworkSettings.IPAddress}} amazing\_tesla

상위 두 명령어는 실행 중인 container의 IP 주소를 반환



* + container가 시작된 이후에 변경된 파일의 목록 조회
* docker diff amazing\_tesla



여기에서는 /bin 에서 삭제된 목록과 /basket에 추가된 모든 파일과 저장소 드라이버와 관련되어 생성된 파일들 조회

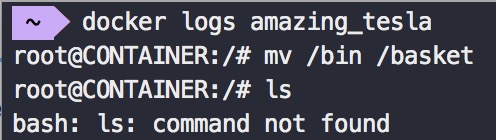
docker는 container에 union file system(UFS)를 사용

UFS는 여러 개의 파일 시스템이 계층 구조로 마운트되어 하나의 파일 시스템 처럼 보이도록 해줌

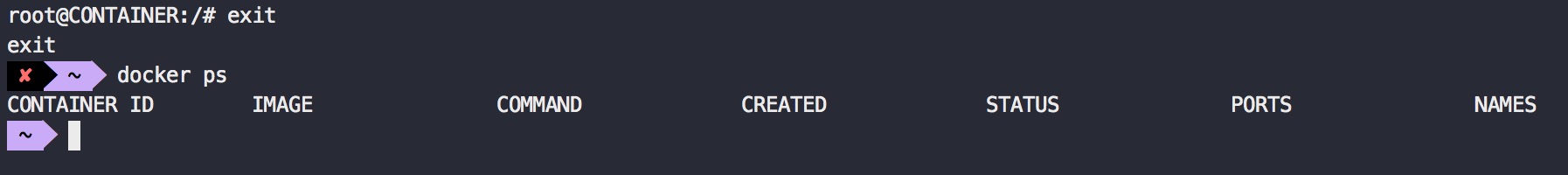
이미지의 파일 시스템은 읽기 전용 계층으로 마운트, 실행 중인 container에서 변경된 모든 내용들은 container에 마운트된 읽기-쓰기 계층에 쓰여짐

이러한 방식 때문에 실행 중인 시스템에서 변경된 내용을 찾으려면 docker는 최상위 읽기-쓰기 계층만 살펴보면 된다.

* + container 작업 로그 확인
* docker logs amazing\_tesla

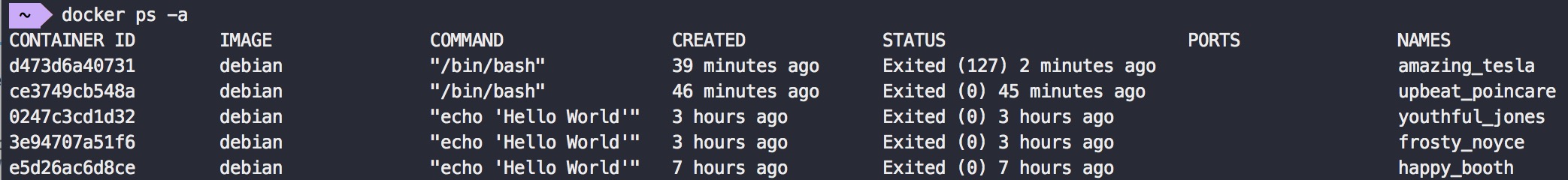


* 1. container 빠져나오기
* exit



container에서 실행되는 유일한 프로세스가 쉘이기 때문에 쉘에서 빠져나오게 되면 container는 중지된다.

* + container 상태별 조회
* docker ps -a

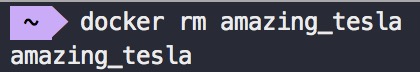


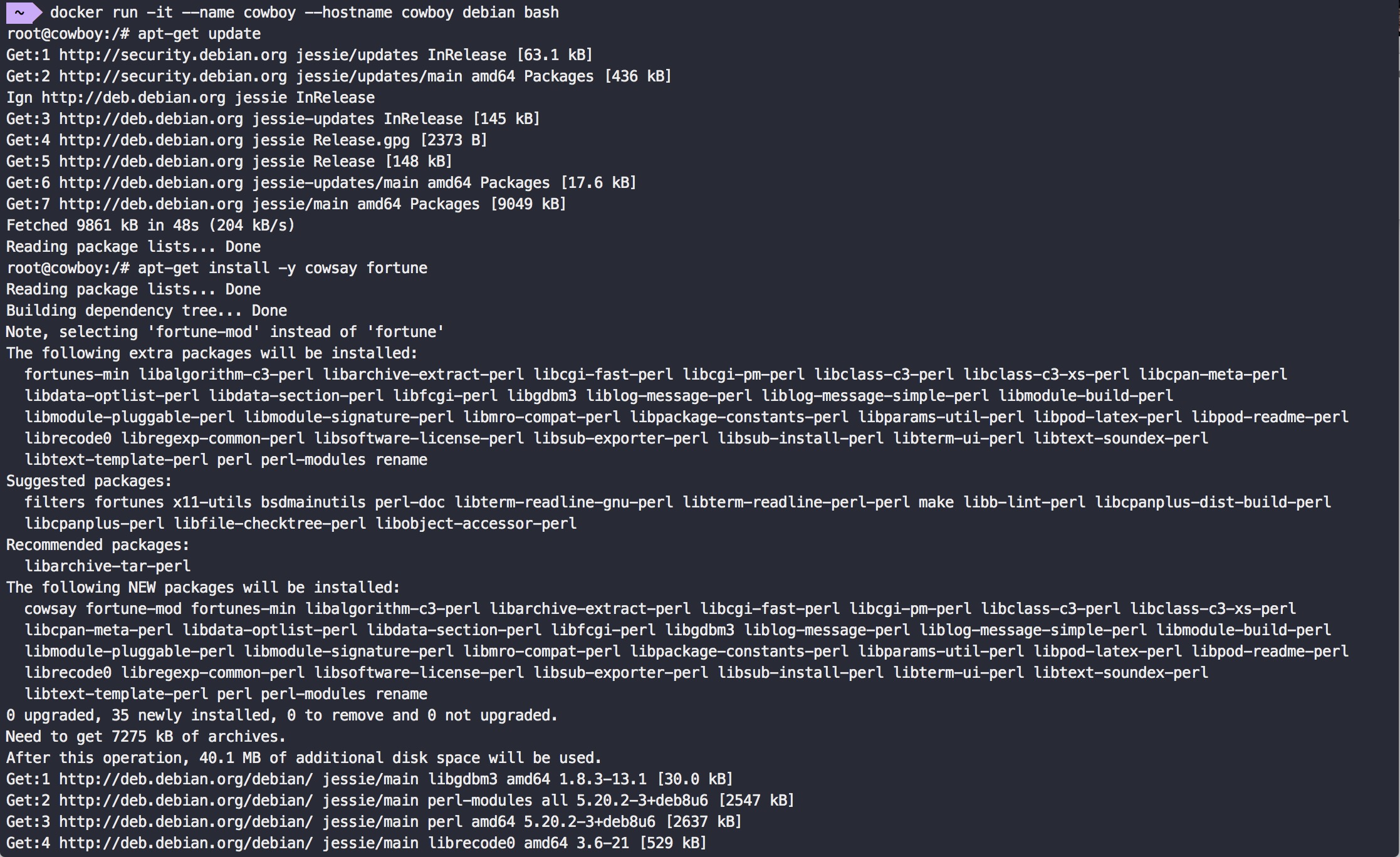
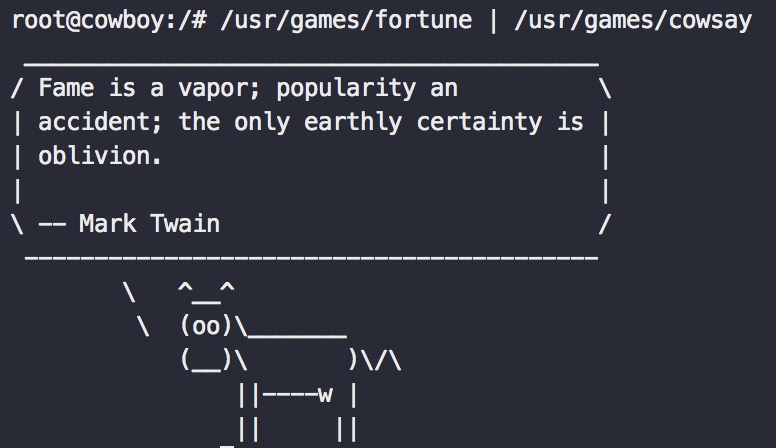
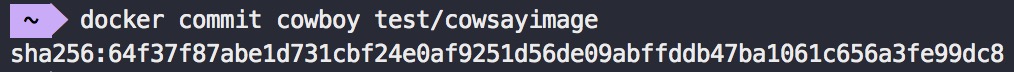
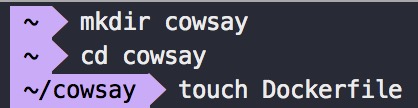
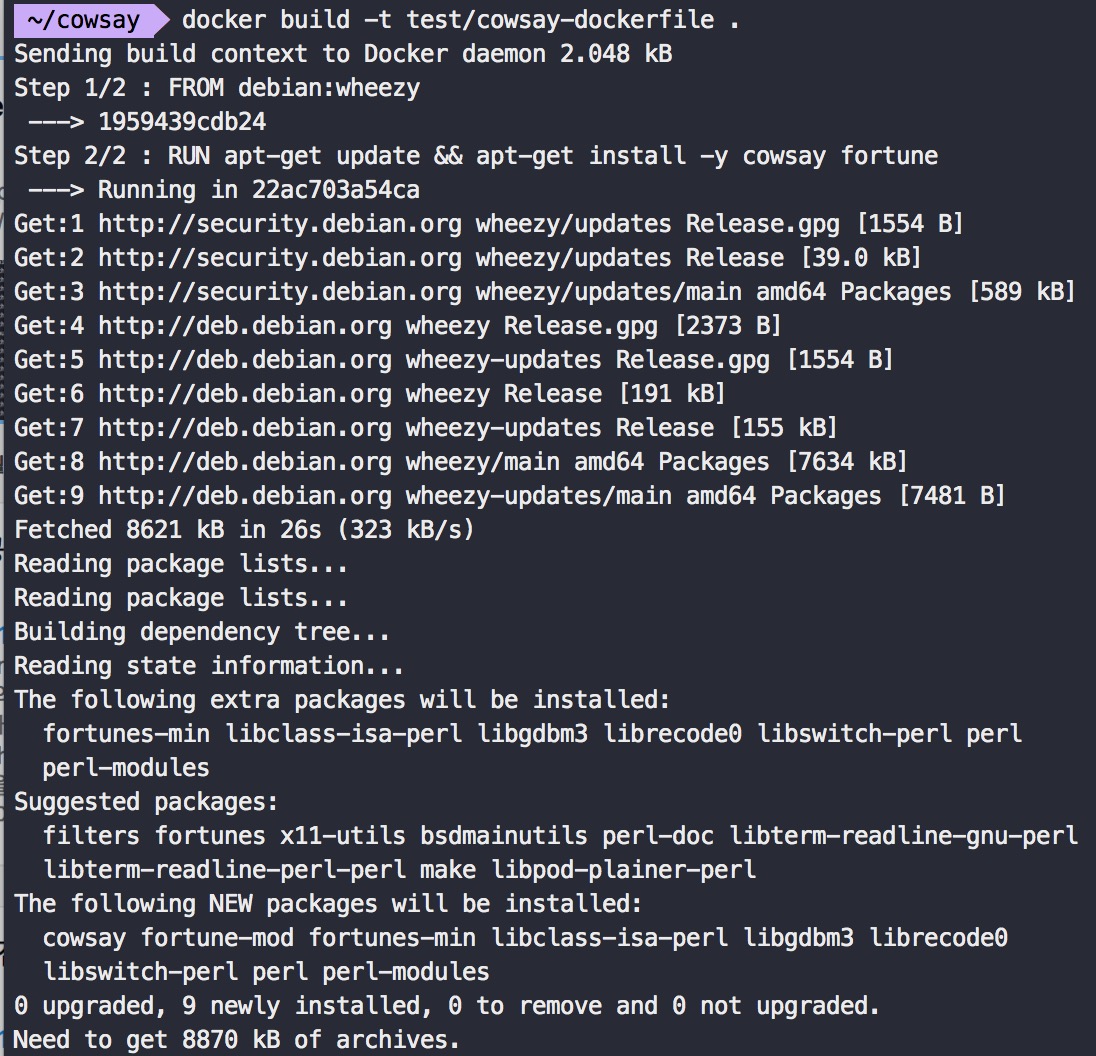
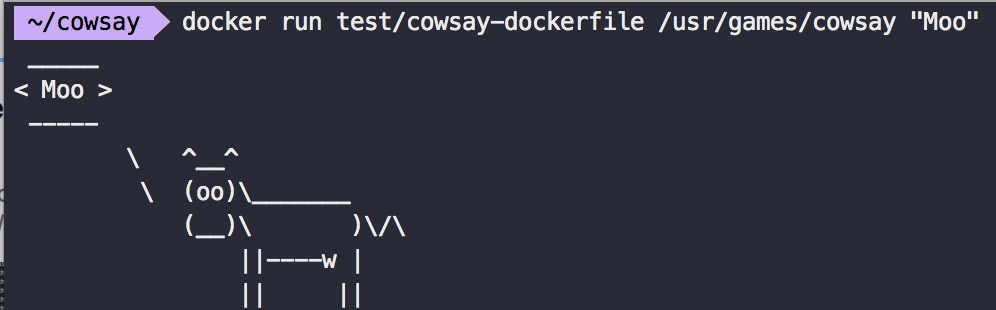
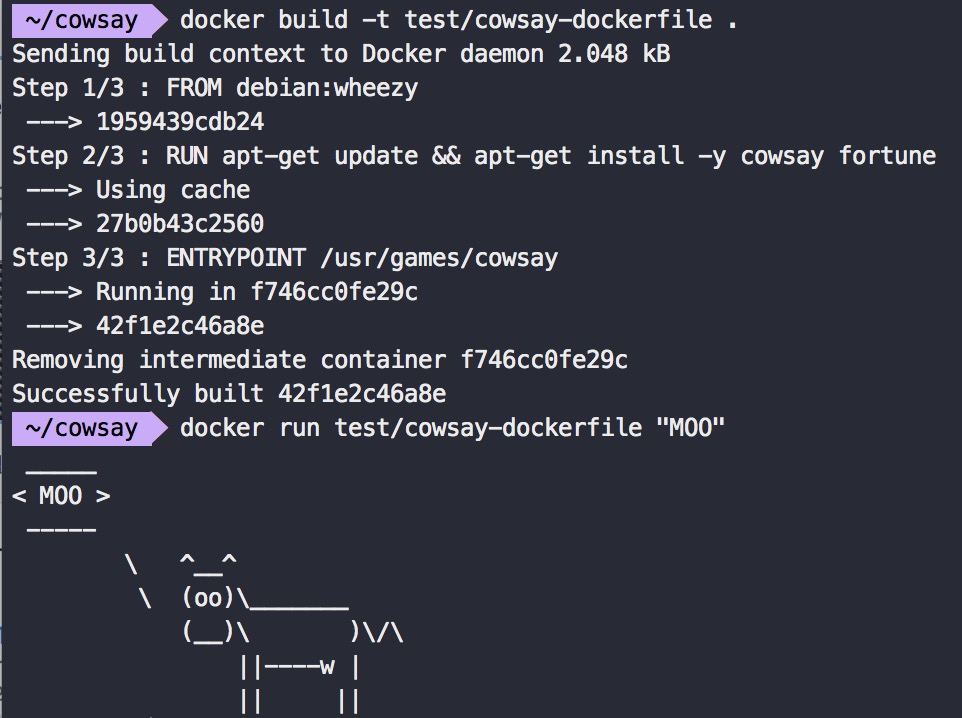
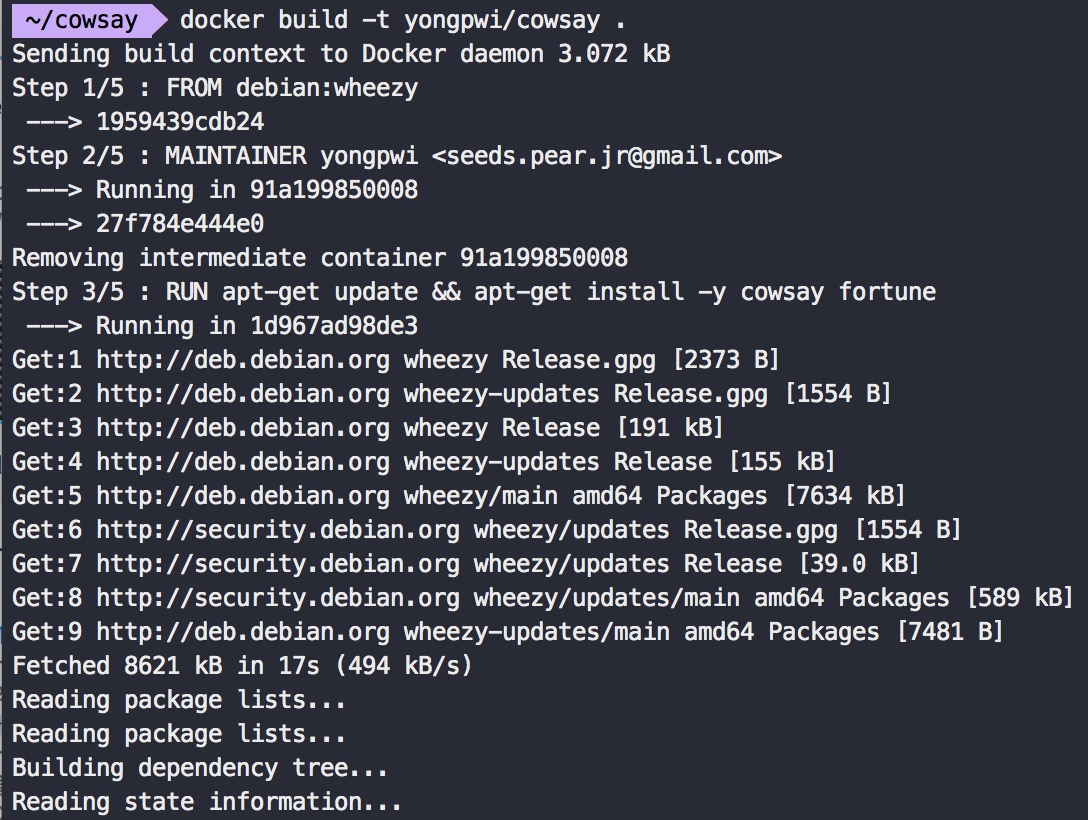
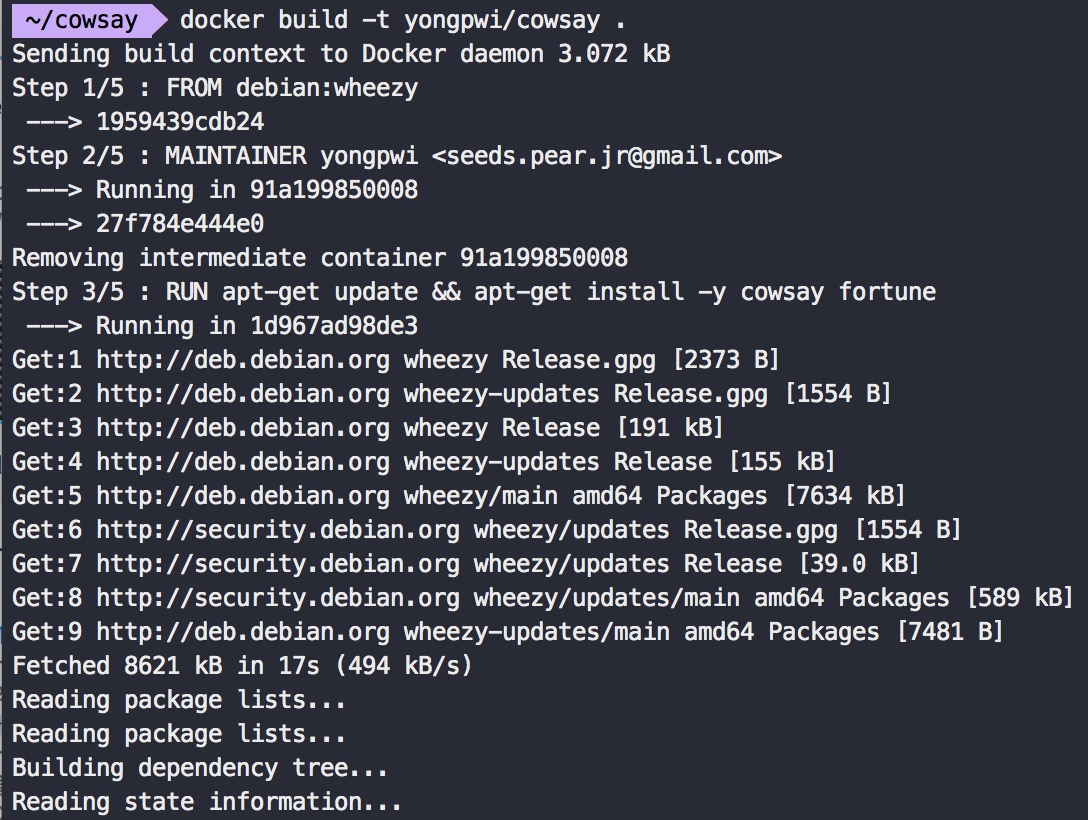
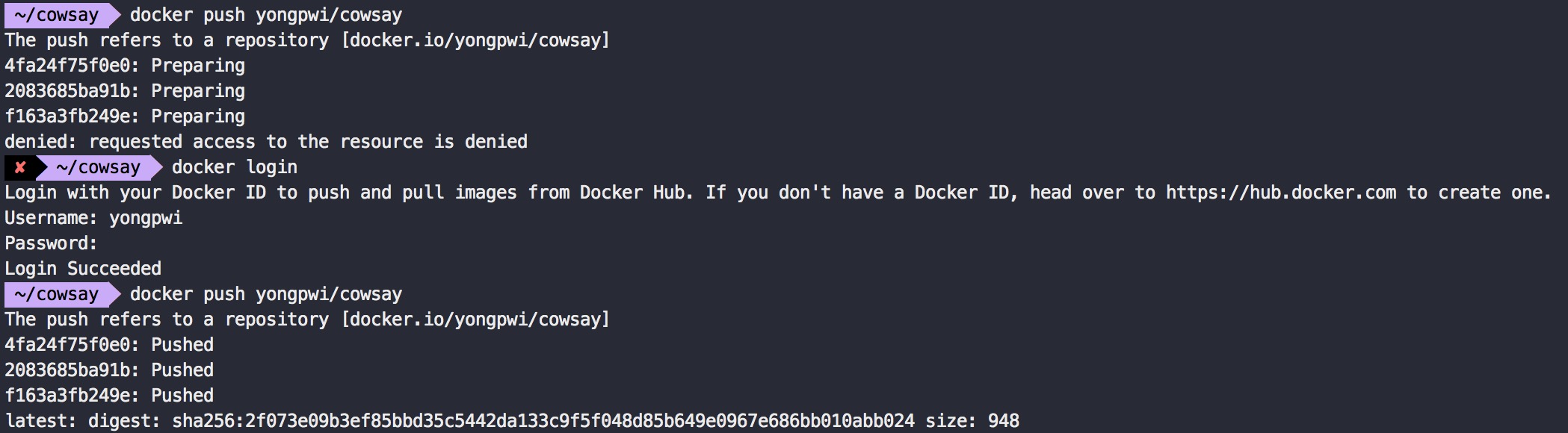
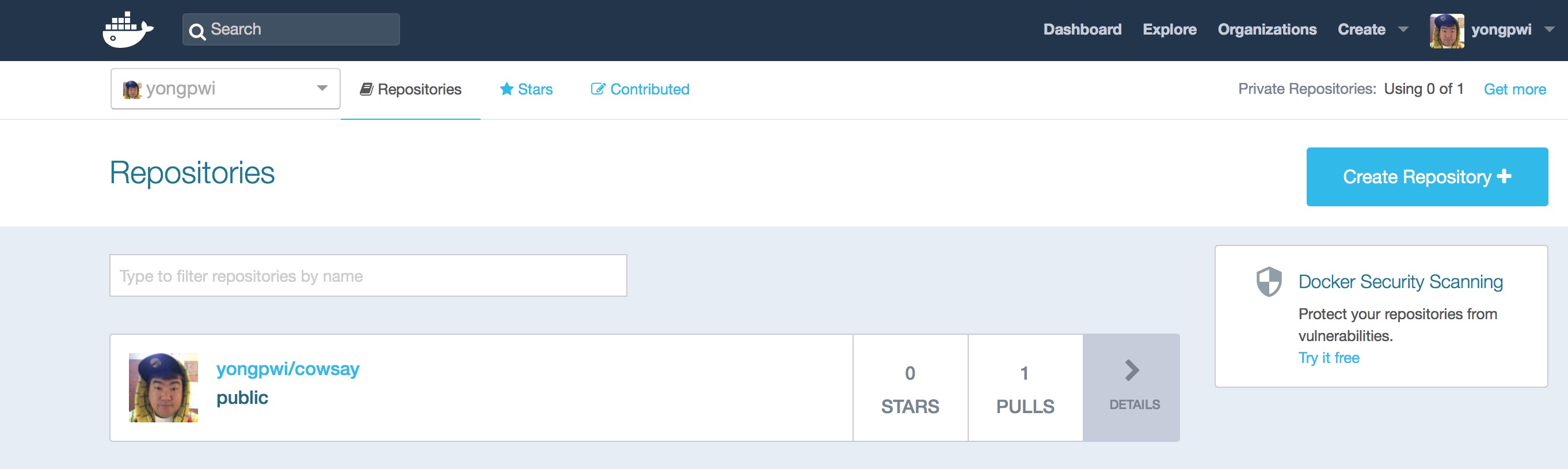
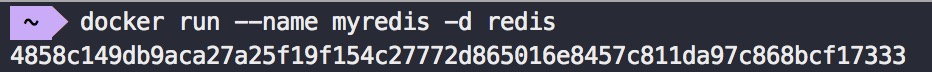
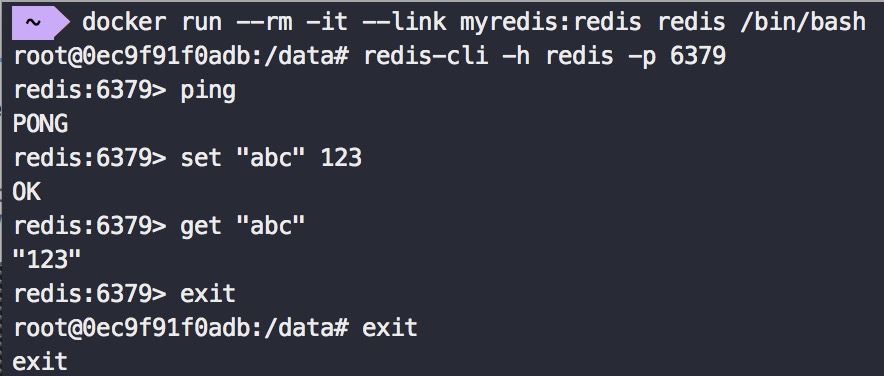
중지된(stopped) 컨테이너 = 공식적으로는 종료된(exited) 라고 표현

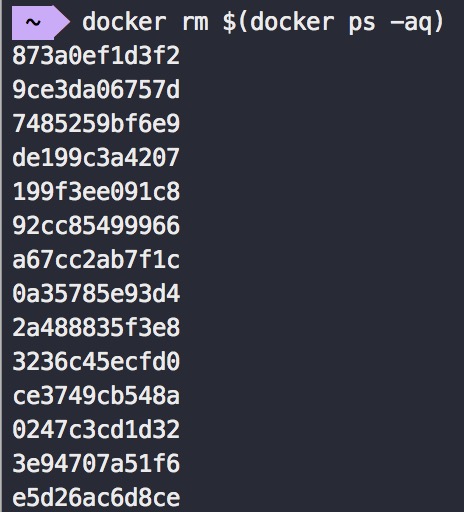
종료된 container는 docker start 명령을 실행하면 재시작 가능

앞서 종료된 container(amazing\_tesla)는 경로를 깨뜨려(mv /bin /basket) 버렸기 때문에 실행 불가

* 1. container 삭제
* docker rm amazing\_tesla



* + 중지된 container 정리(삭제)
  + docker rm -v $(docker ps -aq -f status-exited)
  + -v 인자는 docker가 관리하는 모든 볼륨들을 삭제(다른 container에 의해서 참조되지 않는 상태에 있는 볼륨만 삭제)
  + cowsay 응용프로그램 생성
  + docker run -it --name cowboy --hostname cowboy debian bash
  + apt-get update
  + apt-get install -y cowsay fortune
  + 
  + 응용프로그램 시작
  + /usr/games/fortune | /usr/games/cowsay
  + 
  + container를 이미지로 바꾸기
  + docker commit cowboy test/cowsayimage (상위에서 docker name이 cowboy로 오타가 가서,,,)
  + container가 실행 상태이건 중지 상태이건 상관없이 동작
  + 명령을 실핼할 때 container의 이름(cowboy). 이미지 이름(cowsayimage). 이미지 저장소(test) 명시 필요
  + 생성된 이미지를 구분 가능한 ID 반환
  + 이제 cowsay가 설치되어서 실행할 수 있는 이미지가 만들어졌다.
  + 
  + 만든 이미지 실행
  + docker run test/cowsayimage /usr/games/cowsay "Moo”
  + 
    - 현 이미지의 문제점
      1. 변경 필요시 필요한 부분까지 앞서 실행했던 절차들을 수동으로 다시 반복 필요
      2. ex : 다른 기본 이미지 사용하려면 처음부터 모든 작업 다시 시작
      3. 반복적으로 사용할 수 있는 쉬운 방법이 없음
* Dockerfile로 이미지 만들기
  + Dockerfile은 docker image를 생성하기 위해서 사용될 수 있는 일련의 절차들을 담고 있는 텍스트 파일
  + mkdir cowsay
  + cd cowsay
  + touch Dockerfile
  + 
  + 다음 내용을 dockerfile에 추가
  + FROM debian:wheezy
  + RUN apt-get update && apt-get install -y cowsay fortune
  + FROM 설정에는 사용할 기본 이미지 - 앞에서는 debian을 사용, 이번에는 wheezy라는 태그가 표기된 버전 사용
  + 모든 docker file에는 첫번째 설정으로 FROM 명령이 포함되어야 함
  + RUN 설정은 이미지 내부에서 수행될 쉘 명령을 지정할 때 사용
  + 같은 디렉토리에서 docker build 명령을 실행하면 이미지가 만들짐
  + docker build -t test/cowsay-dockerfile .
  + 
  + 생성한 이미지 실행
  + docker run test/cowsay-dockerfile /usr/games/cowsay "Moo"
  + 
  + container의 상태는 생성(created), 재시작(restarting), 실행 중(running), 일시 중지(paused), 종료(exited) 등으로 구분
    - container가 생성된 상태라는 의미는 docker create 명령으로 시작되었지만, 아직 완전하게 시작되지 않은 상태
    - 종료 상태는 일반적으로 중지된(stopped) 상태
    - container 내부에서 실행되는 프로세스가 없음
    - container는 메인 프로세스가 종료되는 시점에 종료
    - 종료된 container는 docker start 명령으로 다시 시작가능
    - 중지된(stopped) container는 이미지와는 다름
    - 중지된 container는 이미지에 저장되지 않는 IP 주소등과 같은 런타임 구성 정보와 함께 설정, 메타데이터, 파일시스템에 대한 변경된 내용을 그대로 유지
    - 재시작 상태는 실제로 거의 보기 어려움, docker engine이 실패한 container를 재시작하는 경우 발생
  + ENTRYPOINT Dockerfile 설정
    - ENTRYPOINT 설정은 docker run 명령으로 전달되는 모든 인자를 처리하기 위해서 사용되는 실행 파일을 명시
    - Dockerfile에 하위 구문 추가
    - ENTRYPOINT ["/usr/gmaes/cowsay"]
    - docker build -t test/cowsay-dockerfile .
    - docker run test/cowsay-dockerfile "MOO"
    - 
    - 기존과 달리 cowsay 명령 없이 이미지 실행 가능
    - 그러나 container 내부의 fortune(랜던 메세지 프로그램) 명령을 cowsay 응용프로그램의 입력 값으로 사용할 수 있는 기능이 사라짐
    - ENTRYPOINT에 스크립트를 추가하여 수정
    - entrypoint.sh 파일 생성하고 하단 내용 추가
    - #!/bin/bash
    - if [ $# -eq 0 ]; then
    - /usr/games/fortune | /usr/games/cowsay
    - else
    - /usr/games/cowsay "$@"
    - fi
    - chmod +x entrypoint.sh 실행하여 파일 실행 가능하도록 설정
    - 인자 없이 호출되는 경우, 스크립트는 입력 값을 fortune에서 cowsay로 전달하게 됨
    - 인자가 전달되는 경우, 주어진 인자를 이용하여 cowsay 응용프로그램 호출
    - docker file 수정
    - FROM debian:wheezy
    - RUN apt-get update && apt-get install -y cowsay fortune
    - COPY entrypoint.sh /
    - ENTRYPOINT ["/entrypoint.sh"]
    - COPY 설정은 호스트에 있는 파일을 이미지의 파일 시스템으로 복사하는 작업을 수행
    - cp와 유사하게 첫 번째 인자는 호스트에 있는 파일을, 두 번째 인자는 복사될 대상 경로 의미
    - 새로운 이미지를 빌드하고 container를 실행
    - docker build -t test/cowsay-dockerfile .
    - docker run test/cowsay-dockerfile
    - docker run test/cowsay-dockerfile Hello LibQa
    - 
* Registry 를 이용한 작업
  + 이전까지 사용한 debian 이미지는 공식 docker registry인 docker hub에서 이미지를 다운로드 하여 사용
  + 우리가 작성한 이미지를 docker hub에 업로드하고 다른 사람들이 다운로드하여 사용 가능
  + docker hub는 명령 중과 웹 사이트를 통해 접근 가능
  + [http://registry.hub.docker.com](http://registry.hub.docker.com/)
  + 이미지는 계층적인 구조로 저장
    - 레지스트리(Registry)
    - 이미지를 운영하고 배포하는 역할을 담당하는 서비스, 기본 registry는 dock hub임
    - 저장소(Repository)
    - 관련된 이미지들(일반적으로 같은 응용프로그램 또는 서비스의 각기 다른 버전)의 집합
    - 태그(Tag)
    - 저장소에 있는 이미지에 붙여진 알파벳과 숫자로 된 구분자
  + docker pull amouat/revealjs:latest 명령은 docker hub registry의 amouat/revealjs 저장소에 있는 latest 태그가 붙어 있는 이미지를 다운
  + docker file에 소유자 정보를 입력
  + FROM debian:wheezy
  + MAINTAINER yongpwi <seeds.pear.jr@gmail.com>
  + RUN apt-get update && apt-get install -y cowsay fortune
  + COPY entrypoint.sh /
  + ENTRYPOINT ["/entrypoint.sh"]
  + 이미지를 다시 빌드하고 빌드한 이미지를 docker hub에 업로드
  + docker build -t yongpwi/cowsay .
  + docker push yongpwi/cowsay
  + 
  + 업로드시 denied: requested access to the resource is denied 에러가 발생할 경우
  + 생성한 docker hub 계정으로 로그인을 하지 않아서 발생
  + docker login 입력 이후 username, password 입력 이후 실행하면 업로드 가능
  + 
  + 업로드시 태그를 생략하면 latest라는 태그가 자동 할당된다.
  + docker build -t yongpwi/cowsay:stable 과 같이 태그를 할당 가능
  + 업로드가 완료되면 다른 사람들이 docker pull 명령(docker pull yongpwi/cowsay)을 이용하여 이미지 다운 가능하다.
  + 
  + 이미지 네임스페이스
    - yongpwi/cowsay 같이 문자로 된 접두어와 /로 이름이 구성되면, user 네임스페이스에 소속된다.
    - 접두어나 /가 없는 debian, ubuntu 와 같은 이름은 root 네임스페이스에 속하며 해당 이미지들은 docker 에서 관리되며 docker hub에서 사용할 수 있는 일반적인 소프트웨어와 배포판의 공식 이미지들을 위해 예약되어 있음
    - 호스트 이름이나 IP가 접두어로 사용된 이름은 docker hub가 아닌 서드 파티 레지스트리에서 운영되는 이미지
    - 사용자는 네임스페이스로 이미지들의 위치를 정확하게 구분하여 사용가능
* Redis 공식 이미지 이용하기
  + 이미지 설치
  + docker pull redis
  + 
  + redis container 시작
  + docker run --name myredis -d redis
  + 
  + -d 인자는 container를 백그라운드로 실행하라는 의미
  + docker는 다른 방법과 동일하게 container를 시작하지만, -d 명령어가 있는 경우 container에서 출력하는 내용을 보여주지 않고 container의 ID만 출력
  + container는 여전히 백그라운드에서 실행되고 있기 때문에 docker logs 명령어를 실행하면 container에서 출력한 모든 내용 확인 가능
  + redis-cli를 실행하기 위한 container 시작
  + docker run --rm -it --link myredis:redis redis /bin/bash
  + redis-cli -h redis -p 6379
  + ping
  + set "abc" 123
  + get "abc"
  + exit
  + 
  + docker run 명령에 --link myredis:redis 인자를 같이 사용하면 링크가 마법?을 일으킨다
  + 이 명령어는 docker로 하여금 새로운 container와 기존의 myredis container를 연결하고 새로운 container 안에서 myredis container를 redis라는 이름으로 참조하는 작업을 수행함
  + 이런 작업을 수행하려면 docker는 container /etc/hosts 에서 redis 를 위한 진입점을 생성하고 myredis container의 IP 주소를 가리키도록 한다. 이렇게 하면 redis container의 IP 주소를 전달하거나 찾을 필요 없이 redis-cli에서 redis 라는 호스트 이름을 사용 가능하다
  + redis ping 명령을 실행하여 redis 서버로 연결 되었는지 확인하고 set과 put을 이용하여 데이터 추가 및 반환한다.
  + 문제
  + 기능적으로 잘 동작하지만 데이터 유지와 백업에 대한 문제가 있음
  + 이러한 문제 때문에 일반적인 container 파일시트템을 사용해서는 안됨
  + 대신 container와 호스트 또는 다른 container 간에 공유가 용이한 것이 필요
  + docker는 이와 같은 기능을 볼륨(volumes)이라는 개념을 통해 제공
  + 볼륨(volumes)은 다른 container와 공유될 수 잇으며, 모든 변경 사항들은 호스트 파일 시스템에 직접 쓰인다.
  + 디렉터리를 볼륨으로 생성방법
    - docker file 안에 VOLUME 설정(VOLUME /data)
    - -v 플래그 명시(docker run -v /data /test/webserver)
    - 이동성과 보안상의 이유로 docker file 내부에 호스트 디렉터리를 명시할 수 없다
  + 해결
  + docker run --rm -it --link myredis:redis redis /bin/bash
  + redis-cli -h redis -p 6379
  + set "presistence" "test"
  + save
  + exit
  + docker run --rm --volumes-from myredis -v $(pwd)/backup:/backup \debian cp /data/dump.rdb /backup/
  + 
  + -v 인자에는 호스트에 마운트하여 공유하게 되는 디렉터리
  + --volumes-from 인자에는 Redis 데이터베이스 폴더를 공유할 새로운 container 경로
  + container 종료 및 삭제
  + docker stop myredis
  + docker rm -v myredis
  + 
  + 나머지 container 삭제도 가능
  + docker rm $(docker ps -aq)

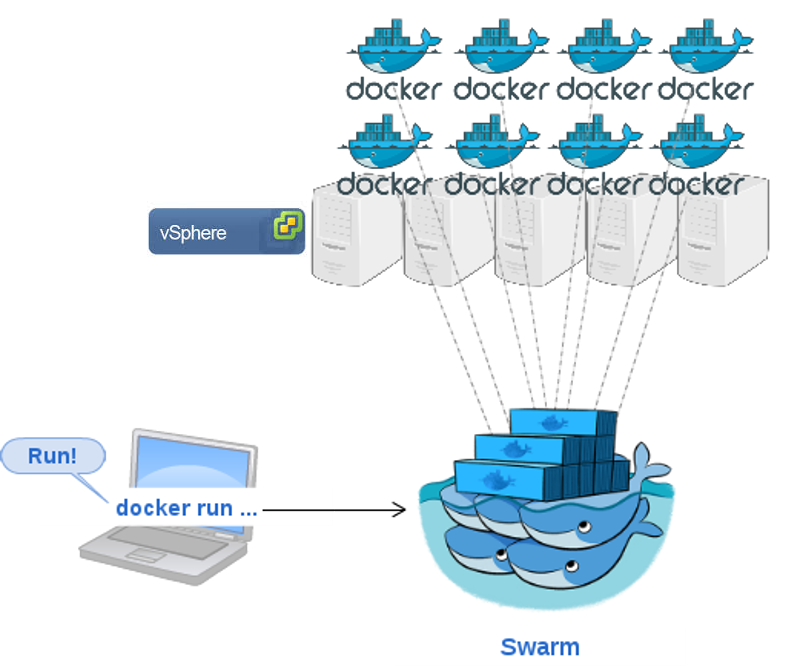
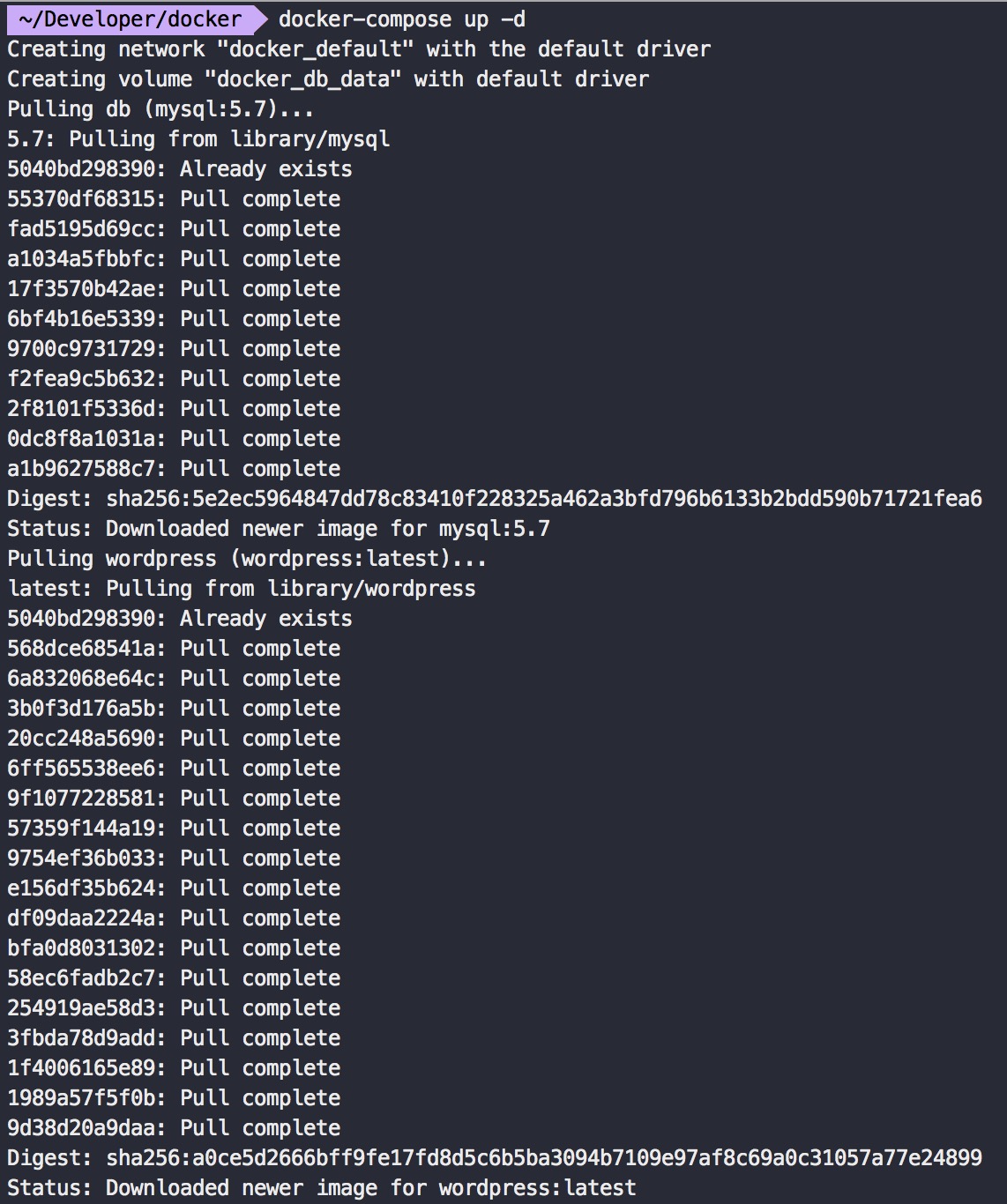
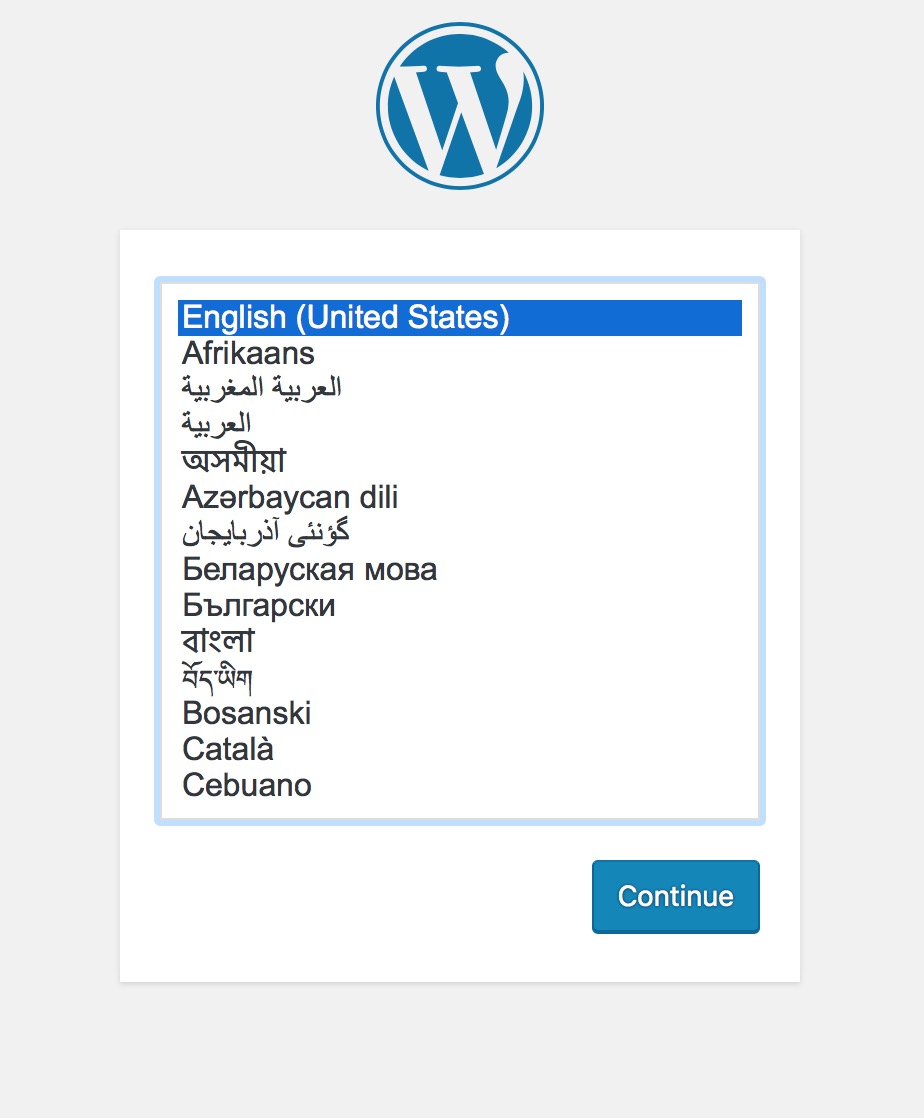


Docker의 기초

Docker 아키텍처



출처 : <https://docs.docker.com/engine/understanding-docker>

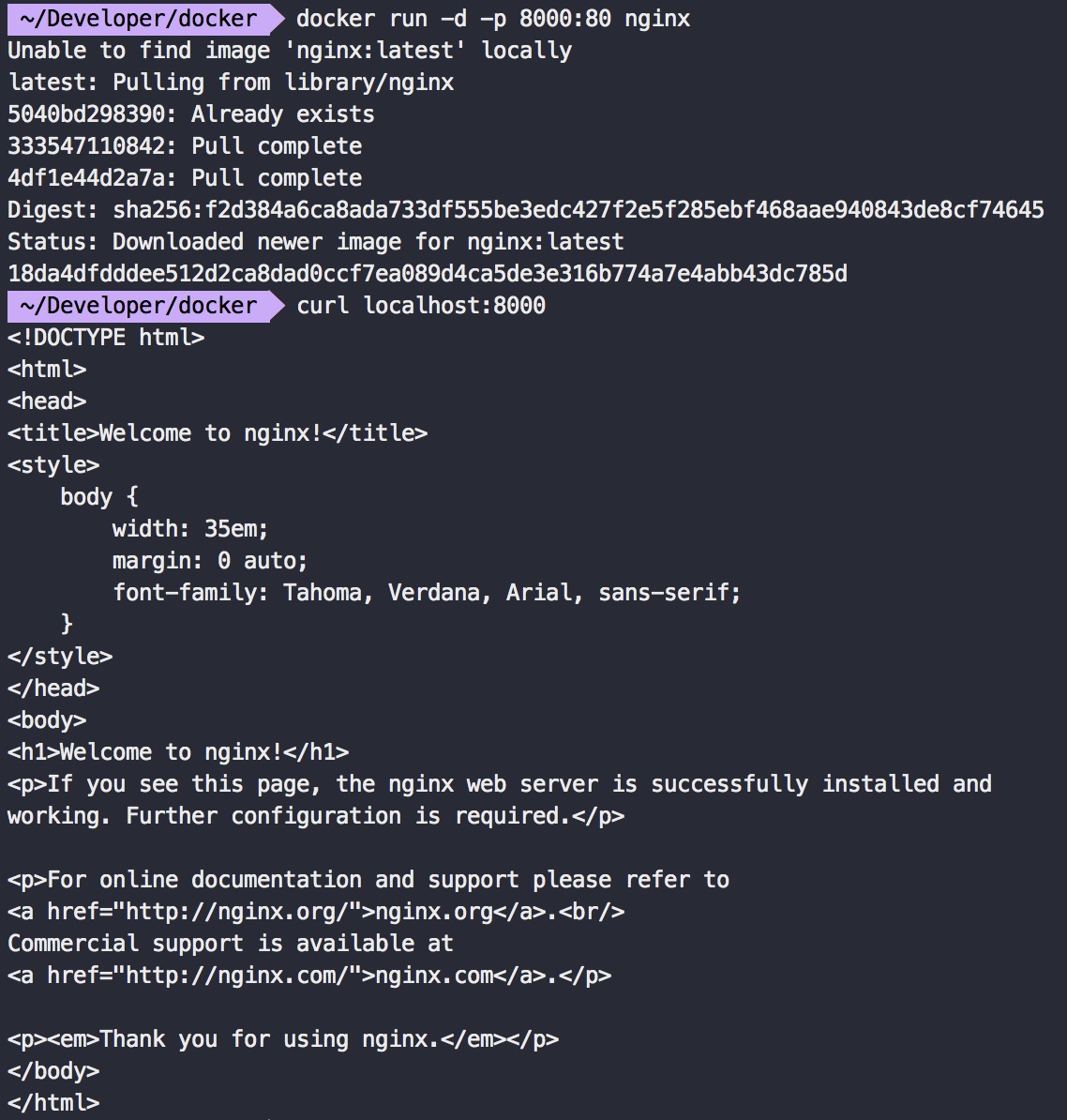
* Docker daemon
  1. 중앙에 있는 Docker daemon은 container를 생성, 실행, 모니터링뿐 아니라 이미지를 생성, 저장 역할까지 담당
  2. Docker daemon은 docker daemon 명령을 실행하여 시작되며 해당 명령은 일반적으로 host OS 의해 관리
* Docker client
  + http를 통하여 docker daemon과 통신
  + 기본적으로 통신은 [Unix domain socket](https://en.wikipedia.org/wiki/Unix_domain_socket) 통해 이루어지지만 원격 클라이언트 또는 systemd가 관리하는 소켓의 파일 기술자(file descriptor)를 활성화하여 TCP 소켓을 이용하는 방법도 있음
  + docker client, docker daemon은 하나의 바이너리로 배포
* Docker Registry
  1. docker registry는 이미지들을 저장하고 배포
  2. 기본 registry는 docker hub
  3. 많은 기업들은 독자적 registry 운영
     1. 상용 이미지, 기업에 중요한 이미지 저장
     2. docker hub에서 다운로드하는 오버헤드 줄이기 위함
* 기반기술
  + docker daemon
    1. 실행 드라이버(execution driver)를 이용하여 container 생성
  + 유니온 파일 시스템(Union File System)
    1. container의 계층들을 저장하는데 사용
    2. AUFS, 디바이스 맵퍼, BTRFS 또는 오버레이와 같은 다양한 저장소 드라이버의 하나로 제공
* 관련기술
* docker engine과 docker hub만으로는 container를 운영하기 위한 완벽한 솔루션 구성이 불가
* 현재 대부분의 시스템은 클러스터 관리, 서비스 검색 도구, 고급 네트워킹 기능과 같은 지원 서비스와 소프트웨어를 필요
* docker는 모든 기능들을 내장하고 있지만 기본 구성 요소들을 서드 파티 제품으로 쉽게 교체 가능한 솔루션을 만들 계획을 가지고 있음
* 현재 docker 지원하는 기술
  + Swarm(스웜)
    - docker의 clustering 솔루션
    - 여러개의 docker host를 하나의 그룹으로 묶을 수 있게 해준다.
    - 사용자 입장에서는 하나의 리소스처럼 사용 가능
    - 
    - 출처 : <http://www.vmtocloud.com/how-to-run-a-docker-swarm-cluster-on-vsphere-with-photon/>
  + Compose
    - 여러 개의 docker container로 구성된 응용 프로그램을 작성하고 실행하기 위한 도구
    - 주로 운영보다는 개발 및 테스트 용도로 사용
    - YAML 파일을 사용하여 다중 container 앱의 원하는 상태를 선언적으로 정의
    - mac, windows용 docker 기본적으로 포함
    - linux 설치 필요
  + curl -L "https://github.com/docker/compose/releases/download/1.10.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose **chmod** +**x** /usr/**local**/bin/docker-compose docker-compose *--version*
    1. compose를 활용한 wordpress 설치
    2. 빈 디렉토리 생성
    3. yml 설정 파일 생성
  + mkdir my\_wordpress cd my\_wordpress touch docker-compose.yml
    - yml 파일 작성
    - version: '2'
    - services:
    - db:
    - image: mysql:5.7
    - volumes:
    - - db\_data:/var/lib/mysql
    - restart: always
    - environment:
    - MYSQL\_ROOT\_PASSWORD: wordpress
    - MYSQL\_DATABASE: wordpress
    - MYSQL\_USER: wordpress
    - MYSQL\_PASSWORD: wordpress
    - wordpress:
    - depends\_on:
    - - db
    - image: wordpress:latest
    - ports:
    - - "8000:80"
    - restart: always
    - environment:
    - WORDPRESS\_DB\_HOST: db:3306
    - WORDPRESS\_DB\_PASSWORD: wordpress
    - volumes:
    - db\_data:
    1. 빌드 프로젝트
  1. docker-compose up -d
  2. + 
     1. 

[http://localhost:8000](http://localhost:8000/)

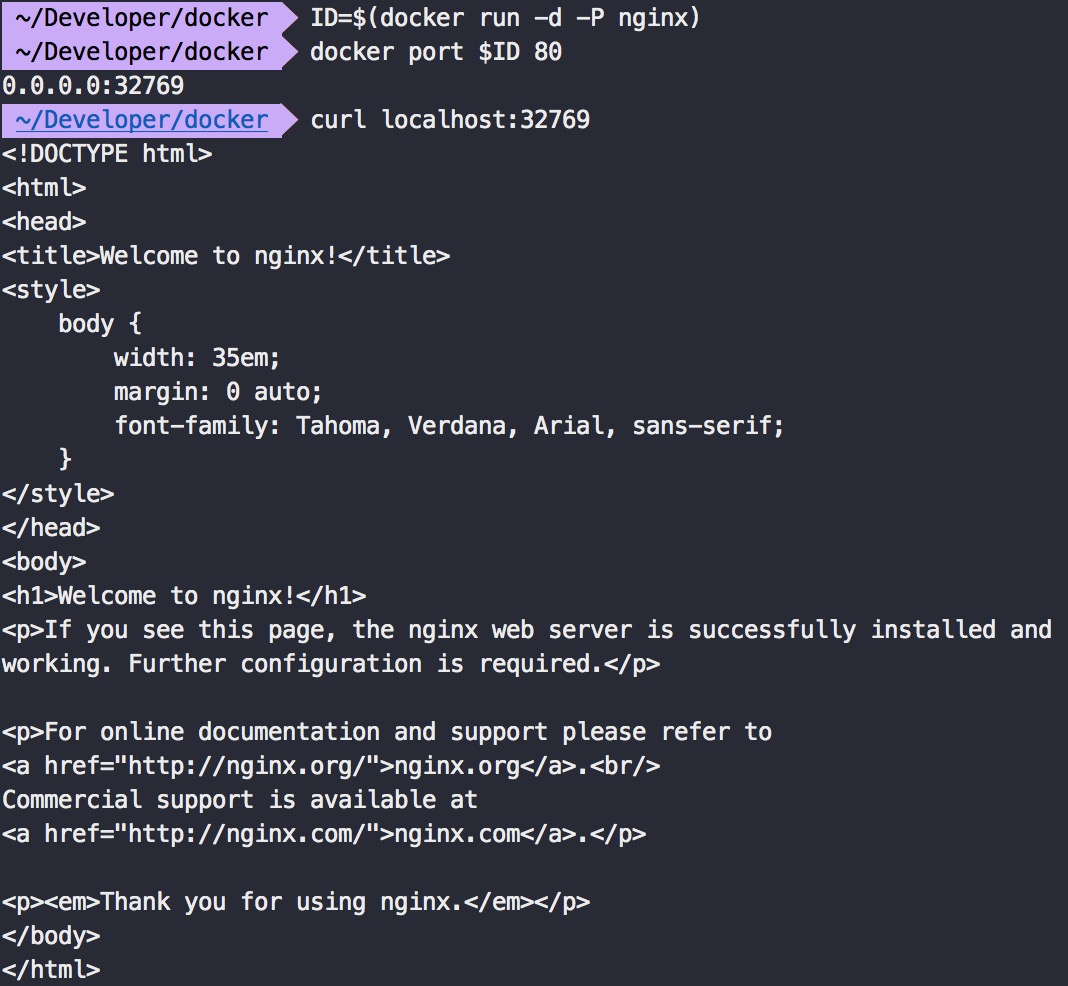
* + Machine
    - 로컬 또는 원격 리소스에서 운영되는 docker를 설치하고 구성하는 기능 제공
    - docker client 구성
    - 간단하게 구성환경간에 전환이 가능
  + Kitematic(카이트매틱)
    - docker container를 실행하고 관리할 수 있는 GUI
    - Windows, OS X용 제공
  + Docker Trusted Registry
    - Docker 이미지들을 저장하고 관리하기 위한 docker의 온프레미스 솔루션
    - docker hub의 로컬버전
    - 기존 보안 인프라와 통합, 기업 데이터 저장소 및 보안과 관련된 법규를 지킬 수 있도록 해준다
    - 현재 docker사의 제품 중에 유일하게 오픈소스화 되지 않음
  + Docker OS
    - docker는 대부분의 리눅스 배포판에서 잘 실행
    - docker만을 위한 크기가 작으면서 유지가 용이한 배포판 프로젝트 진행중, container 운영에 초점을 둔 OS( [Project Atomic](http://www.projectatomic.io/), [코어 OS](https://coreos.com/), [RancherOS](http://rancher.com/rancher-os) )
  1. image cache
     1. docker는 이미지 생성 속도를 높이기 위해서 모든 계층을 cache에 저장
     2. cache 저장 경우
        1. cache에 이전 설정이 존재
        2. cache에 있는 계층이 정확하게 같은 설정과 상위 계층을 가지고 있는 경우
        3. COPY, ADD 설정의 경우 체크섬 또는 메타데이터가 변경된 파일이 있으면 cache 무효화
        4. RUN 설정이 여러 번 호출되더라도 같은 결과를 반환한다는 보장이 없어도 cache에는 저장됨
           1. 파일들을 다운로드, apt-get update 실행 또는 소스 저장소 복제 등의 작업하는 경우 고려 필요
        5. cache 무효화
           1. --no-cache 인자를 이용하여 docker build 명령 실행
           2. cache 무효화하려는 지점 바로 전에 설정을 추가하거나 변경(비추천 : 나중에 이미지를 사용하는 사용자는 혼란스러울 수 있음)

Container를 외부와 연결하기

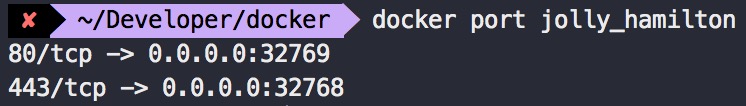
* container 안에 web server를 운영하는 경우 외부에서 해당 web server 접근 가능하도록 설정
  + -p 또는 -P 명령으로 port 게시
* docker run -d -p 8000:80 nginx
* curl localhost:8000



* + docker가 자동으로 port 선택
* ID=$(docker run -d -P nginx) docker port $ID 80 // docker 연결 포트 출력(ex : 0.0.0.0:32771) curl localhost:32771



* + -P 명령의 가장 큰 장점은 할당된 port들을 계속하여 파악하고 있어야 할 필요가 없음
    - 여러 개의 container들이 port를 게시 하고 있다면 꼭 필요
    - docker port 명령 이용하여 docker에 할당한 port 확인



Container 연결하기

* docker link
  + 같은 호스트에 있는 container끼리 통신할 수 있는 가장 간단한 방법
  + docker 기본 네트워킹 모델을 이용하는 경우 container 간의 통신은 내부 docker 네트워크를 통해 이루어짐
    - 통신은 호스트 네트워로 노출 되지 않는다는 의미
  + docker의 향후 버전에서 container와 통신하기 위해서 container를 link하는 것보다는 서비스를 게시 하는 방법 사용 예상
  + 당분간은 link 지원
  + 사용방법
    - docker run 명령을 실행하면서 --link [CONTAINER:ALIAS](container:ALIAS)
      * CONTAINER = link container(연결되는 container)
      * AliAS = master container(link container를 시작하는 책임을 지고 있기 때문에 시작되는 container 라고 한다.)
  + 내부동작
    - 별칭(alias)과 link container의 ID를 master container의 /etc/hosts에도 추가하여 master container에서 이름으로 link container를 찾을수 있음
    - docker는 master container 안에 상당히 많은 환경 변수들을 설정하여 link container와 통신을 용이하게 지원

docker run -d --name myredis redis

docker run --link myredis:redis debian env

이미지 들어가야함

container로 접속하기 위한 정보인 REDIS\_PORT라는 접두어가 붙은 환경 변수들을 docker가 설정

docker는 link container로 부터 환경 변수를 가져오게 되는데, 해당 환경 변수는 REDIS\_ENV로 시작

기본적으로 link의 구성 여부와 상관 없이 container들은 서로 통신이 가능

link되지 않는 container는 통신할 수 없도록 하려면 docker daemon을 시작할 때 --icc=false와 --iptables 인자 사용

* + 단점
    - 정적
      * link는 container가 재시작되더라도 그대로 유지되어야 하지만 link container가 대체되면 갱신되지 않음
      * master container가 시작되기 전에 link container가 시작되어야 함
        + 양방향 link는 불가능

일반적인 Docker 명령어

* run
  + container 라이프사이클, 기본 운영 모드 제어
    - 새로운 container를 시작할 때 사용
    - 가장 일반적인 명령어
    - -a, --attach
      * 주어진 스트림([STDOUT](https://en.wikipedia.org/wiki/Standard_streams#Standard_output_.28stdout.29) 등)을 터미널에 연결
      * -d와 같이 사용 불가
    - -d, --detach
      * container를 분리(detached) 모드(daemon 모드라고도 함)로 실행
      * container를 백그라운드로 실행시키고 container ID 반환
    - -i, --interactive
      * 연결되지 않은 상태라도 [Stdin](https://en.wikipedia.org/wiki/Standard_streams#Standard_input_.28stdin.29)을 열어 둔 상태로 유지
      * 일반적으로 대화형 container session을 시작하기 위해 -t와 같이 사용

docker run -it debian /bin/bash

* + - --restart
      * docker가 종료된 container가 어떤 경우에 재시작할지 구성
      * no
        + container 재시작하지 않음
      * always
        + 종료상태에 상관없이 항상 재시작 시도
      * on-failure
        + container가 0이 아닌 상태로 종료된(exit with a nonzero status) 경우에만 container 재시작 시도
    - -rm
      * container가 종료되면 자동으로 container 삭제
      * -d와 같이 사용 불가
    - -t, --tty
      * 가상 -TTY(pseudo-TTY)를 할당
      * 일반적으로 대화형 container를 시작하기 위해 -i와 같이 사용
  + container 이름, 변수 설정
    - -e, --env
      * container 내부의 환경 변수 설정

docker run -e var1=var -e var2="var 2" debian env

* + - * --env-file
        + 파일로 환경 변수 전달 가능
    - -h, --hostname
      * container 유닉스 호스트 이름 NAME으로 설정

docker run -h "myhost" debian hostname //myhost를 호스트 이름으로 설정

* + - --name NAME
      * NAME이라는 이름을 container에 할당
      * 할당된 이름은 다른 docker 명령어에서 해당 container를 찾을 때 사용
  + volume 구성 옵션
    - -v, --volume
      * container 안의 디렉터리만 명시, 디렉터리는 docke가 정한 host 디렉터리로 바인드
      * 바인드할 호스트 디렉터리를 지정
    - --volumes-form
      * 지정된 container의 volume을 마운트
      * 주로 data container와 같이 사용
    - --expose
      * docker file의 EXPOSE 설정과 동일
      * container에서 사용되는 port 범위를 지정하지만, port를 열지는 않음
      * 실제로는 container를 연결하면서 -P와 함께 사용되는 경우가 일반적
    - --link
      * 지정된 container에 사설 네트워크 인터페이스 구성
    - -p, --publish
      * 호스트에서 container로 접근할 수 있도록 container에 port를 게시
      * 호스트 port가 정의되지 않으면 port는 무작위로 설정
      * port 번호는 docker port 명령으로 확인 가능
      * 호스트 인터페이스의 포트도 지정 가능
    - -P, --publish-all
      * container에 노출된 모든 port를 호스트로 노출
      * port는 무작위로 선택
      * docker port 명령을 이용하면 호스트와 container 간의 port가 어떻게 연결되었는지 확인 가능
  + docker file 설정 무시
    - --entrypoint
      * docker file의 ENTRY POINT 설정 모두 무시
      * 주어진 인자 이용해 container entrypoint 설정
    - -u, --user
      * 명령을 실행할 사용자 이름 또는 UID 설정
      * docker file USER 설정 무시하고 재정의
    - -w, --wordir
      * 지정된 경로를 container 작업 디렉터리로 설정
      * docker file의 모든 값 재정의
* container 관리하기
  + docker create
    - 이미지를 이용하여 container 생성하지만 시작하지는 않음
    - docker run에서 사용되는 대부분의 인자를 그대로 사용 가능
    - 생성된 container를 시작하려면 docker start
  + docker cp
    - container와 호스트 간에 파일과 디렉터리 복사
  + docker exec
    - container sㅐ부에 있는 명령 실행
    - 유지 관리 작업 수행이나 container로 로그인하기 위한 ssh를 대체하는 용도로 사용

ID=$(docker run -d debian sh -c "while true; do sleep1; done;")

docker exec $ID echo "Hello"

//Hello

docker exec -it $ID /bin/bash

* + docker kill
    - container 메인 프로세스(PID 1)로 신호를 보냄
    - 기본적으로 SIGKILL라는 신호를 보내는데 이는 container를 즉시 종료
    - -s 사용시 container ID 반환
  + docker pause
    - 지정된 container 내부 모든 프로세스 일시 중지
    - 일시 중지된 프로세스들은 어떤 신호도 받을수 없음
      * 해당 container 종료, 정리 불가
    - docker unpause 실행시 프로세스 다시 시작
  + docker restart
    - 하나 또는 여러 개의 container 다시 시작
    - container에 대해 docker stop, docker start 명령을 차례로 실행하는 것과 동일
    - -t 인자 사용시 [SIGTERM](https://en.wikipedia.org/wiki/Unix_signal#SIGTERM)으로 강제 종료 전에 container 종료되기까지 기다리는 시간 지정 가능
  + docker rm
    - 하나 또는 여러 개의 cotainer 삭제
    - 정상적으로 삭제된 container name 또는 ID 반환
    - 기본적으로 docker rm은 어떠한 volumeeh 삭제하지 않음
    - -f
      * 실행중인 container 삭제
    - -v
      * container가 생성한 volume(바인드 마운트되어 있지 않거나 다른 container가 사용하지 않는 경우) 삭제
    - 중지된 모든 container 삭제
      * docker rm $(docker ps -aq)
  + docker start
    - 중지된 container 시작
    - 상태가 exit이거나 생성 이우 한 번도 실행되지 않은 container 시작
  + docker stop
    - 하나 또는 여러 개의 container 중지(삭제하지 않음)
    - docker stop 명령이 호출되면 container는 종료(exited) 상태
    - -t 인자 사용시 [SIGTERM](https://en.wikipedia.org/wiki/Unix_signal#SIGTERM)으로 강제 종료 전에 container 종료되기까지 기다리는 시간 지정 가능
  + docker unpause
    - docker pause로 중지시킨 container 다시 시작
* docker 정보
  + docker info
    - docker 시스템과 호스트 정보 출력
  + docker help
    - 명령어 사용 방법 및 도움말 정보 출력
  + docker version
    - 컴파일에 사용된 Go 버전과 함께 client와 server 버전 정보 출력
* container 정보
  + docker diff
    - 이미지로부터 container가 만들어진 이후 container 파일시스템에 적용된 변경 사항 출력

ID=$(docker run -d debian touch /NEW\_FILE)

docker diff $ID

* + docker events
    - daemon 실시간 이벤츠 출력
    - CTRL-C 누르면 종료
  + docker inspect
    - container나 이미지에 대한 상세 정보 제공
  + docker logs
    - container 로그 출력
  + docker port
    - container의 노출된 port 매핑 목록 반환
  + docker ps
    - 이름, ID, 상태 등 현재 container들의 개괄적인 정보 제공
    - -a
      * 모든 container 정보 반환
    - -q
      * container ID만 반환
      * docker rm 같은 다른 명령어의 입력 값으로 사용하기에 유용
  + docker top
    - container에서 실행 중인 프로세스들에 대한 정보 반환
* 이미지 관리하기
  + docker build
    - docker file 이용하여 이미지 생성
  + docker commit
    - container로부터 이미지 생성
    - docker build가 일반적으로 사용
    - commit 전에 일시 중지
    - --pause=false 사용시 일시 중지 안됨
    - -a, -m 인자 사용하여 메타데이터 설정 가능

ID=$(docker run -d redis touch /new-file)

docker commit -a "Joe Bloggs" -m "Comment" $ID commit:test

docker images commit

docker run commit:test ls /new-file

* + docker export
    - container 파일 시스템 내용을 STDOUT에 tar gㅕㅇ식으로 저장
    - 저장된 결과는 docker import를 이용, 로드하여 이미지 생성 가능
    - 파일 시스템만 저장됨(노출된 port, CMD, ENTRYPOINT 설정과 같은 메타데이터는 저장 안됨)
    - 모든 volume 저장되지 않음
    - docker save와 대조
  + docker history
    - 이미지의 각 계층에 대한 정보 출력
  + docker images
    - 로컬 이미지들의 목록을 저장소 이름, 태그 이름, 크기 등과 같은 정보과 함께 반환
    - VIRTUAL SIZE는 이미지 전체 크기, 모든 기본 계층들을 포함한 크기
  + docker import
    - docker export로 생성된 이미지처럼 파일시스템을 포함하고 있는 아카이브 파일로부터 이미지 생성
    - 이미지가 생성되면 해당 이미지 ID 반환
    - 이미지에는 저장소와 태그 이름 등을 태그로 기록 가능
    - import 사용하여 생성된 이미지는 단일 계층, 노출된 port와 CMD 값들과 같이 docker 구성 설정은 잃어 버림
    - docker load와 비교
  + docker load
    - STDIN을 통해 전달된 tar 아카이브로부터 저장소 로드
    - 저장소는 여러 개의 이미지와 태그 포함
    - docker import와 달리 이미지는 히스토리와 메타데이터 정보 포함
    - docker save로 만들어진 아카이브 파일들이 가장 적합한 파일 형태
    - 이미지 배포와 백업 생성을 위한 registry 대안 고려 가능
  + docker rmi
    - 지정된 이미지 삭제
  + docker  save
    - 이미지 또는 저장소를 STDOUT으로 스트림하여 tar 아카이브로 저장
    - 저장소 이름만 주어지면 저장소에 있는 모든 이미지들이 아카이브로 저장(모든 버전의 이미지가 저장)
    - docker load와 같이 사용하면 이미지를 배포 또는 백업 가능

docker save -o /tmp/redis.tar redid:latest

docker rmi redis:latest

//

docker load -i /tmp/redis.tar

docker images redis

* + docker tag
    - 이미지에 저장소와 태그 이름을 지정
    - 이미지는 ID 또는 저장소와 태크(태그가 주어지지 않으면 latest 태그가 사용된 것으로 간주) 지정 가능
* registry 이용하기
* docker는 자격 증명을 홈 디렉터리에 .dockercfg 확장자를 가진 파일로 저장
  + docker login
    - 지정된 registry 등록, 로그인
    - 서버 지정되지 않으면 docker hub 사용으로 간주
  + docker logout
    - docker registry로부터 logout
    - 서버 지정되지 않으면 docker hub 사용으로 간주
  + docker pull
    - 지정된 이미지를 registry에서 다운로드
    - registry는 이미지 이름으로 결정, 기본값은 docker hub
    - tag 이름이 주어지지 않으면 lastest 태그 이미지 자동 다운로드
    - -a 인자 사용시 저장소에 있는 모든 이미지 다운
  + docker push
    - 이미지나 저장소를 registry에 올림
    - tag 이름이 주어지지 않으면 lastest 태그 이미지 자동 설정 및 저장소에 있는 모든 이미지들을 레지스트리로 올림
  + docker search
    - docker hub에서 검색하려는 단어와 일치하는 공용 저장소 목록을 출력
    - 최대 25개 저장소 결과 출력
    - Starts와 automated builds로도 결과 필터링 가능

해당 명령어 사용보단 웹사이트 사용을 권장

# Docker 개발환경 세팅 삽질해보기

### **Nginx**

* dockerfile

FROM nginx:1.11.1

MAINTAINER yongpwi <seeds.pear.jr@gmail.com>

RUN rm -rf /etc/nginx/conf.d/default.conf

COPY libqa.conf /etc/nginx/conf.d/libqa.conf

COPY nginx.conf /etc/nginx/nginx.conf

CMD ["nginx"]

EXPOSE 80

EXPOSE 443

* libqa.conf

server {

listen 80;

server\_name "";

access\_log off;

location / {

proxy\_pass http://libqawar:8080;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto http;

proxy\_max\_temp\_file\_size 0;

proxy\_connect\_timeout 150;

proxy\_send\_timeout 100;

proxy\_read\_timeout 100;

proxy\_buffer\_size 8k;

proxy\_buffers 4 32k;

proxy\_busy\_buffers\_size 64k;

proxy\_temp\_file\_write\_size 64k;

}

}

* nginx.conf

http {

include /etc/nginx/mime.types;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

default\_type application/octet-stream;

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /var/log/nginx/access.log main;

sendfile on;

#tcp\_nopush on;

keepalive\_timeout 65;

client\_max\_body\_size 300m;

client\_body\_buffer\_size 128k;

gzip on;

gzip\_http\_version 1.0;

gzip\_comp\_level 6;

gzip\_min\_length 0;

gzip\_buffers 16 8k;

gzip\_proxied any;

gzip\_types text/plain text/css text/xml text/javascript application/xml application/xml+rss application/javascript application/json;

gzip\_disable "MSIE [1-6]\.";

gzip\_vary on;

include /etc/nginx/conf.d/\*.conf;

}

### **Tomcat**

* dockerfile

# Pull base image

From tomcat:8-jre8

# Maintainer

MAINTAINER "yongpwi <seeds.pear.jr@gmail.com">

ADD application-develop.properties /usr/local/tomcat/

ENV JAVA\_OPTS="-Dspring.config.location=/usr/local/tomcat/application.properties"

# Copy to images tomcat path

ADD libqa.war /usr/local/tomcat/webapps/

EXPOSE 8080

* libqa.war

        프로젝트 파일로 <https://github.com/howlingproject/libqa> 파일 war 배포본

* application-develop.properties

        DB 관련 설정 파일인데 docker run 시 환경설정 파라미터 변수 전달 방법 찾는중

### **정리**

* docker가 tomcat을 먼저 8080 port로 시작
* nginx는 80 port로 오면 docker의 name인 libqawar:8080 으로 포워딩