

Part 1: Getting started

어서 오십시오! 여러분이 Docker를 배우고 싶어 해서 기쁩니다.

What is a container?

이제 컨테이너를 실행했으므로 컨테이너란 무엇입니까? 간단히 말해서 컨테이너는 호스트 시스템의 다른 모든 프로세스와 격리된 시스템의 샌드박스 프로세스입니다. 이러한 격리는 오랫동안 Linux에 있었던 기능인 kernel namespaces and cgroups을 활용합니다. Docker는 이러한 기능을 접근하기 쉽고 사용하기 쉽게 만들기 위해 노력했습니다.

요약하자면, 컨테이너란:

- 이미지의 실행 가능한 인스턴스입니다. DockerAPI 또는 CLI를 사용하여 컨테이너를 생성, 시작, 중지, 이동 또는 삭제할 수 있습니다.
- 로컬 머신, 가상 머신에서 실행하거나 클라우드에 배포할 수 있습니다.
- 이식 가능(모든 OS에서 실행 가능)
- 컨테이너는 서로 격리되어 있으며 자체 소프트웨어, 바이너리 및 구성을 실행합니다.

What is a container image?

컨테이너를 실행할 때 격리된 파일 시스템을 사용합니다.

이 사용자 지정 파일 시스템은 컨테이너 이미지에서 제공됩니다.

이미지에는 컨테이너의 파일 시스템이 포함되어 있으므로 애플리케이션을 실행하는 데 필요한 모든 것(모든 종속성, 구성, 스크립트, 바이너리 등)이 포함되어야 합니다.

이미지에는 환경 변수, 실행할 기본 명령 및 기타 메타데이터와 같은 컨테이너에 대한 기타 구성도 포함되어 있습니다.

Start the tutorial

명령 프롬프트 또는 bash 창을 열고 다음 명령을 실행합니다.

```
$ docker run -d -p 80:80 docker/getting-started
```

몇 가지 플래그가 사용되고 있음을 알 수 있습니다. 이에 대한 추가 정보는 다음과 같습니다.

- -d 분리(detached) 모드에서 컨테이너 실행 (in the background)
- -p 80:80 호스트의 포트 80을 컨테이너의 포트 80에 매핑합니다.
- docker/getting-started 사용할 이미지

단일 문자 플래그를 결합하여 전체 명령을 단축할 수 있습니다. 예를 들어 위의 명령은 다음과 같이 작성할 수 있습니다.

```
$ docker run -dp 80:80 docker/getting-started
```

CLI commands에 대한 추가 문서는 다음 주제를 참조하십시오.

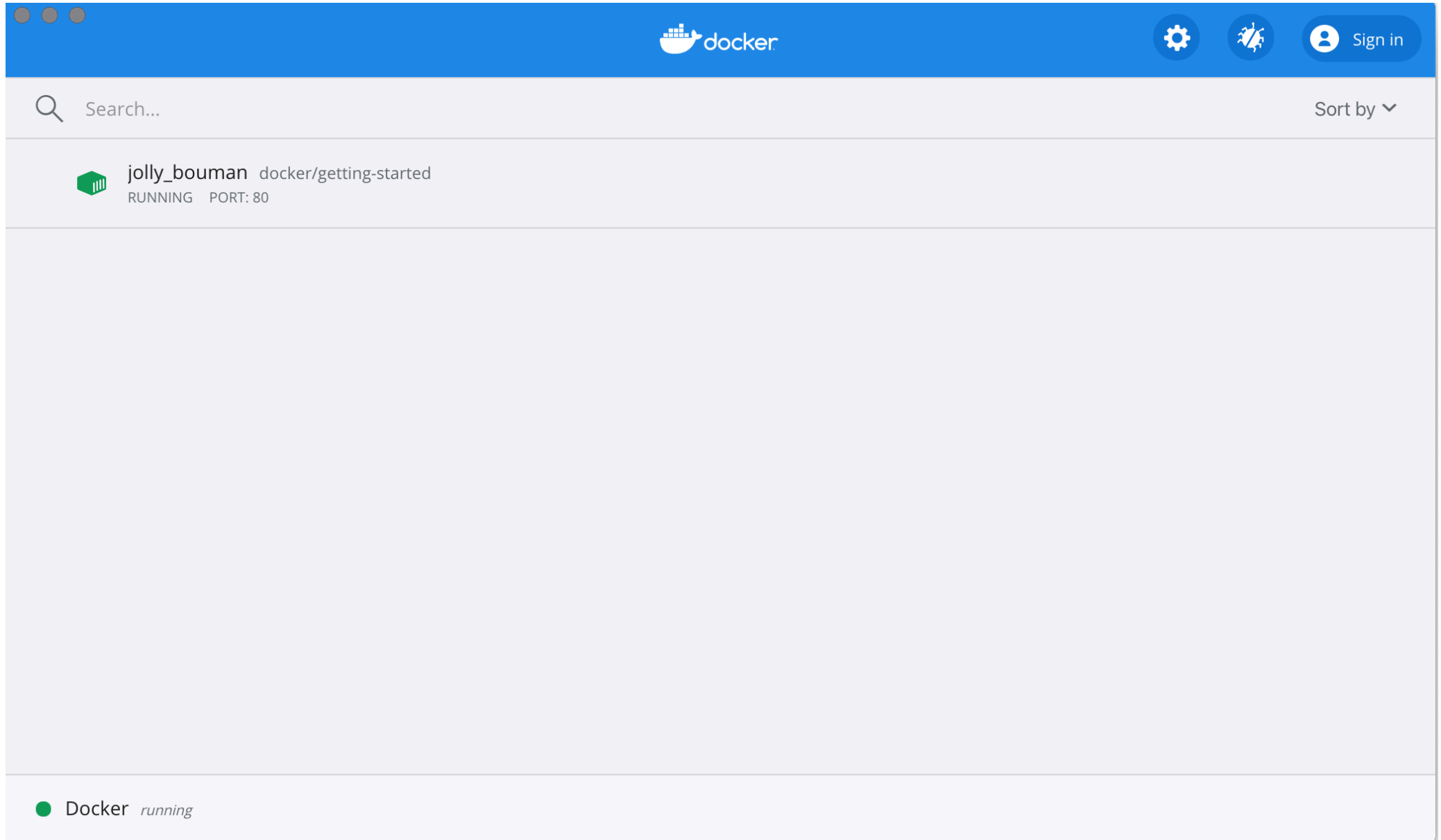
- docker version
- docker run
- docker image
- docker container

The Docker Dashboard

- kitematic is Docker dashboard for Linux
(preconditions : Manage Docker as a non-root user)

너무 진행하기 전에 머신에서 실행 중인 컨테이너를 빠르게 볼 수 있는 Docker Dashboard를 강조하고 싶습니다. Docker Dashboard는 Mac and Windows에서 사용할 수 있습니다. 컨테이너 로그에 빠르게 액세스할 수 있고, 컨테이너 내부에 셸을 가져올 수 있으며, 컨테이너 관리(중지, 제거 등)를 쉽게 할 수 있습니다.

대시보드에 액세스하려면 Docker Desktop 설명서의 지침을 따르십시오. 지금 대시보드를 열면 이 튜토리얼이 실행되는 것을 볼 수 있습니다! 컨테이너 이름(아래 jolly_bouman)은 임의로 생성된 이름입니다. 따라서 다른 이름을 가질 가능성이 큼니다.



Part 2: Sample application

컨테이너 이미지를 구축하는 것에 대한 매우 기본적인 사항을 배우고 이를 위해 Dockerfile을 생성하고 이미지를 빌드한 후 컨테이너를 시작하고 앱을 실행시켜 봅니다.

Part 3: Update the application

앱을 수정하고 실행 중인 애플리케이션을 새 이미지로 업데이트하는 방법을 알아보겠습니다. 이 과정에서 몇 가지 다른 유용한 명령을 배우게 됩니다.

```
docker rm 명령에 "force" 플래그를 추가하여 단일 명령으로 컨테이너를 중지하고 제거할 수 있습니다.  
For example: docker rm -f <the-container-id>
```

Part 4: Share the application

Docker hub에 로그인하고 이미지를 레지스트리에 푸시하여 공유하여 완전히 새로운 인스턴스에서 Docker hub의 레포지토리에 서 이미지를 가져오고 앱을 실행 할 수 있습니다.

Part 5: Persist the DB

volume 을 이용하여 컨테이너의 특정 파일 시스템 경로를 호스트 시스템에 다시 연결합니다. 컨테이너의 파일 시스템에 있는 SQLite 데이터베이스와 컨테이너를 실행하며 -v 플래그에 대해 알아보고 직접 named volume 에 액세스하는 docker volume inspect 에 대해 알아봅니다.

Part 6: Use bind mounts

Part 5에서 알아본 명명된 볼륨은 데이터가 저장 되는 위치 에 대해 걱정할 필요가 없기 때문에 단순히 데이터를 저장하려는 경우에 좋습니다. bind mounts 를 사용 하면 호스트의 정확한 마운트 지점을 제어합니다. 이것을 사용하여 데이터를 유지할 수 있지만 컨테이너에 추가 데이터를 제공하는 데 자주 사용됩니다. 애플리케이션에서 작업할 때 바인드 마운트를 사용하여 소스 코드를 컨테이너에 마운트하여 코드 변경 사항을 보고 응답하고 변경 사항을 즉시 볼 수 있도록 할 수 있습니다. 바인드 마운트를 사용하는 것은 로컬 개발 설정에서 매우 일반적입니다. 장점은 개발 머신에 모든 빌드 도구와 환경을 설치할 필요가 없다는 것입니다. 단일 docker run 명령으로 개발 환경을 가져와 사용할 수 있습니다. Docker Compose는 명령을 단순화하는 데 도움이 되기 때문에 향후 단계에서 Docker Compose에 대해 이야기할 것입니다.

Part 7: Multi container apps

지금까지 우리는 단일 컨테이너 앱으로 작업했습니다. 컨테이너는 기본적으로 격리되어 실행되며 동일한 시스템의 다른 프로세스나 컨테이너에 대해 알지 못합니다. 그렇다면 한 컨테이너가 다른 컨테이너와 통신하도록 하려면 어떻게 해야 할까요? 대답은 네트워킹 입니다. 이제 네트워크 엔지니어가 될 필요가 없습니다(만세!). 이 규칙만 기억하세요...

두 개의 컨테이너가 동일한 네트워크에 있으면 서로 통신할 수 있습니다. 그렇지 않으면 할 수 없습니다.

Part 7에서는 네트워크를 만들고, 컨테이너를 시작하고, 모든 환경 변수를 지정하고, 포트를 노출하는 등의 작업을 해야 합니다! 그것은 기억해야 할 것이 많고 확실히 다른 사람에게 전달하기 어렵게 만듭니다.

다음 섹션에서는 Docker Compose에 대해 설명합니다. Docker Compose를 사용하면 훨씬 쉽게 애플리케이션 스택을 공유할 수 있고 다른 사람들이 단일(간단한) 명령으로 스택을 가동할 수 있습니다!

Part 8: Use Docker Compose

Docker Compose 는 다중 컨테이너 애플리케이션을 정의하고 공유하는 데 도움이 되도록 개발된 도구입니다. 서비스를 정의하는 YAML 파일을 생성할 수 있으며 단일 명령으로 모든 것을 회전시키거나 분해할 수 있습니다.

- 현재 스키마 버전 및 호환성 매트릭스에 대한 Compose 파일 참조 를 볼 수 있습니다.
- Port, Volumes 지정에 Long syntax를 사용할 수도 있습니다.
- Database 환경설정 레퍼런스를 참조하여 volumes를 정의해야 합니다.

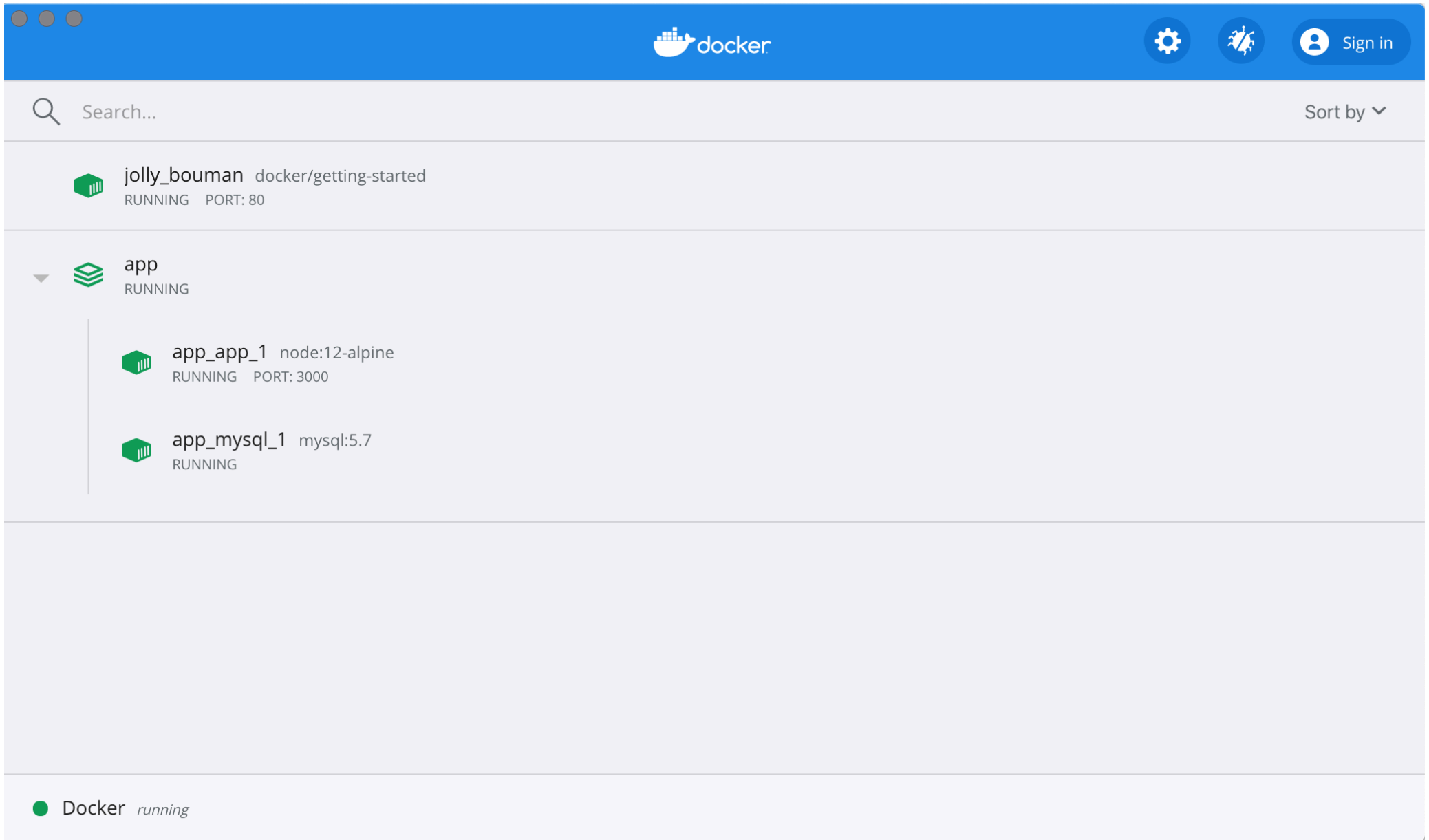
```
version: "3.7" ## 스키마 버전을 정의합니다.

services: ## 애플리케이션의 서비스 목록을 정의합니다.
  app:
    image: node:12-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 3000:3000 ## -p 3000:3000
    working_dir: /app ## 작업 디렉토리를 (-w /app) 마이그레이션 합니다.
    volumes:
      - ./:/app ## 볼륨 매핑을 (-v "$(pwd):/app") 마이그레이션 합니다.
    environment: ## 환경 변수 정의를 마이그레이션 합니다.
      MYSQL_HOST: mysql ## -e MYSQL_HOST=mysql
      MYSQL_USER: root ## -e MYSQL_USER=root
      MYSQL_PASSWORD: secret ## -e MYSQL_PASSWORD=secret
      MYSQL_DB: todos ## -e MYSQL_DB=todos
```

```
mysql: ## MySQL 서비스를 정의합니다. 여기서는 네트워크 별칭을 자동으로 가져옵니다.
image: mysql:5.7 ## 사용할 이미지를 지정합니다.
volumes:
  - todo-mysql-data:/var/lib/mysql ## 최상위 섹션에서 볼륨을 정의한 다음 다음 서비스 구성에서 마운트 지점을 지정해야 합니다. 단순히 볼륨
environment:
  MYSQL_ROOT_PASSWORD: secret
  MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

docker-compose.yml을 통하여 컨테이너를 함께 그룹화 해보았습니다.



이 섹션에서는 Docker Compose와 다중 서비스 애플리케이션의 정의 및 공유를 극적으로 단순화하는 데 Docker Compose에 대해 배웠습니다. 사용하고 있던 명령을 적절한 작성 형식으로 변환하여 작성 파일을 만들었습니다.

이 시점에서 우리는 튜토리얼을 마무리하기 시작합니다. 그러나 우리가 사용하고 있는 Dockerfile에 큰 문제가 있기 때문에 다루고자 하는 이미지 빌드에 대한 몇 가지 모범 사례가 있습니다. 자, 살펴보겠습니다!

Part 9: Image-building best practices

이미지를 구축했으면 `docker scan` 명령을 사용하여 보안 취약점을 스캔하는 것이 좋습니다.

이미지를 스캔하려면 Docker hub에 로그인해야 합니다. `docker scan --login` 명령을 실행 한 다음 `docker scan <image-name>` 을 사용하여 이미지를 스캔합니다.


취약점 유형, 자세히 알아볼 수 있는 URL, 그리고 중요하게는 취약점을 수정하는 관련 라이브러리 버전이 나열됩니다. `docker scan` 문서에서 읽을 수 있는 몇 가지 다른 옵션이 있습니다.

명령줄에서 새로 빌드된 이미지를 스캔할 뿐만 아니라 새로 푸시된 모든 이미지를 자동으로 스캔 하도록 Docker Hub 를 구성할 수 있으며, 그런 다음 Docker Hub와 Docker Desktop 모두에서 결과를 볼 수 있습니다.

Image Layers

Vulnerabilities148











SCANNING BY



Use Snyk to fix these vulnerabilities

- Base image upgrade guidance
- Fix advice for both the container and the code
- Continue monitoring in Kubernetes

See the detailed analysis at [Snyk.io](#)

Severity & Vulnerability		Package	Version	Fixed in
 9.8 Out-of-bounds Write	CVE-2019-12900	bzip2/libbz2	1.0.6-r6	1.0.6-r7
▶  9.8 Out-of-bounds Write	CVE-2019-12900	bzip2/libbz2	1.0.6-r6	1.0.6-r7
 9.8 Out-of-Bounds	CVE-2019-3822	curl/curl	7.61.0-r0	7.61.1-r2
 9.8 Out-of-Bounds	CVE-2018-16839	curl/curl	7.61.0-r0	7.61.1-r1
 9.8 Integer Overflow or Wraparound	CVE-2018-14618	curl/curl	7.61.0-r0	7.61.1-r0
 9.8 Use After Free	CVE-2018-16840	curl/curl	7.61.0-r0	7.61.1-r1
 9.8 Double Free	CVE-2019-5481	curl/curl	7.61.0-r0	7.61.1-r3
 9.8 Buffer Overflow	CVE-2019-5482	curl/curl	7.61.0-r0	7.61.1-r3
▶  9.8 Out-of-Bounds	CVE-2019-3822	curl/curl	7.61.0-r0	7.61.1-r2
▶  9.8 Out-of-Bounds	CVE-2018-16839	curl/curl	7.61.0-r0	7.61.1-r1

1

2

3

4

5

6

7

8

9

10

>

>>

`docker image history` 이미지를 구성하는 요소를 볼 수 있으며 `--no-trunc` 플래그를 추가하면 전체 출력을 얻을 수 있습니다.

레이어 캐싱을 통하여 Dockerfile을 재구성하여 빌드 속도를 높일 수 있습니다.

다단계 빌드는 여러 단계를 사용하여 이미지를 만드는 데 도움이 되는 매우 강력한 도구입니다. 런타임 종속성에서 빌드 시간 종속성 분리, 앱을 실행하는 데 필요한 것만 제공하여 전체 이미지 크기 줄이기를 할 수 있습니다. 예로 들면 Java 기반 응용 프로그램을 빌드할 때 소스 코드를 컴파일하려면 JDK가 필요하지만 프로덕션에서 필요하지 않기 때문에 최종 이미지에 포함되지 않도록 다단계 빌드를 할 수 있습니다.

Part 10: What next?

워크샵은 끝났지만 컨테이너에 대해 더 배워야 할 것이 많습니다! 여기에서 자세히 다루지는 않겠지만 다음에 살펴볼 몇 가지 다른 영역이 있습니다!

컨테이너 오케스트레이션

프로덕션 환경에서 컨테이너를 실행하는 것은 어렵습니다. 시스템에 로그인하고 단순히 `docker run` 또는 `docker-compose up` 을 실행하고 싶지 않습니다. 왜 안돼? 컨테이너가 죽으면 어떻게 될까요? 여러 시스템에서 어떻게 확장합니까? 컨테이너 오케스트레이션은 이 문제를 해결합니다. Kubernetes, Swarm, Nomad 및 ECS와 같은 도구는 모두 약간 다른 방식으로 이 문제를 해결하는 데 도움이 됩니다.

일반적인 생각으로 변경 사항을 프로덕션 환경에 적용해주는 "관리자"가 있다는 것입니다. 여기서 변경 사항은 "내 웹 앱의 두 인스턴스를 실행하고 포트 80을 노출하고 싶습니다."고 말하면 관리자는 클러스터의 모든 시스템을 살펴보고 "작업자"에게 일부 환경에서 작업을 위임합니다. 관리자는 변경 사항(예: 컨테이너 종료)을 확인한 다음 모든 프로덕션 환경에 변경 사항을 반영하도록 작업합니다.

Cloud Native Computing Foundation 프로젝트

CNCF는 Kubernetes, Prometheus, Envoy, Linkerd, NATS 등을 포함한 다양한 오픈 소스 프로젝트를 위한 벤더 중립적인 홈입니다! 여기에서 졸업 및 배양된 프로젝트 와 전체 CNCF 풍경 을 볼 수 있습니다 . 모니터링, 로깅, 보안, 이미지 레지스트리, 메시징

등과 관련된 문제를 해결하는 데 도움이 되는 많은 프로젝트가 있습니다!