

2.1 도커를 사용한 컨테이너 이미지 생성, 실행, 공유

🕒 생성일	@2021년 3월 12일 오전 12:52
🏷 태그	

1. 도커 설치 및 Hello World 컨테이너 실행

예제: 도커에 Hello World 컨테이너 실행

```
docker run busybox echo "Hello world"
```

저거 치면 알아서 Pull 해주고 실행해줌;; 도커샐기,, 머썬놈

```
jiseonsim@simjiseon-ui-MacBook-Air ~$ docker run busybox echo "Hello world"
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
8b3d7e226fab: Pull complete
Digest: sha256:ce2360d5189a033012fbad1635e037be86f23b65cfd676b436d0931af390a2ac
Status: Downloaded newer image for busybox:latest
Hello world
jiseonsim@simjiseon-ui-MacBook-Air ~$
```

1. 도커 허브 레지스트리에서 busybox:latest를 다운받음
2. 이미지로부터 컨테이너 생성
3. 컨테이너 내부에서 실행
4. echo "Hello world"
5. 이후 프로세스 중단
6. 컨테이너도 중지

다른 이미지 실행하기

(나는 이미 mysql 이미지가 있지만...)

```
docker run mysql:5.7
docker run mysql:latest
```

두 명령어를 치면 각각의 버전 이미지 (2개)가 생성됨

2. 간단한 node.js 애플리케이션 생성하기

app.js

```
const http = require('http');
const os = require('os');

console.log("Kubia server starting...");

var handler = function(request, response) {
  console.log("Received request from " + request.connection.remoteAddress);
  response.writeHead(200);
  response.end("You've hit " + os.hostname() + "\n");
};

var www = http.createServer(handler);
www.listen(8080);
```

- HTTP Request/Response : 앱이 실행 중인 머신의 호스트 이름을 받아옴
- 애플리케이션이 실행 중인 컨테이너 내부의 hostname을 바라봄
 - 다수의 앱 인스턴스를 가동하는 scale-out을 할때 유용하게 사용됨
- 8080 포트로 HTTP 서버 시작 → 요청에 대한 응답값을 console에 로깅

3. 이미지를 위한 Dockerfile 생성

앱을 이미지로 패키징하기 위해 먼저 Dockerfile(도커가 이미지를 생성하기 위해 수행해야 할 지시사항)을 생성한다.



Dockerfile은 app.js 파일과 동일한 디렉터리에 있어야 함

Dockerfile

```
FROM node:7
ADD app.js /app.js
ENTRYPOINT ["node", "app.js"]
```

FROM 으로 시작점으로 사용할 컨테이너 이미지 정의 (이미지 생성의 기반이 되는 기본이미지)

4. 컨테이너 이미지 생성

이미지 빌드

```
docker build -t kubia .
```

1. 위 명령어를 콘솔에 입력
2. 도커 클라이언트가 디렉터리의 콘텐츠(**Dockerfile** , **app.js**)를 도커 데몬(가상머신 내부)에 업로드
3. 도커 데몬
 - 이미지가 아직 로컬에 저장돼 있지 않은 경우 도커가 도커허브에서 **node:7.0** 이미지를 pull 한다
4. 새로운 이미지 (kubia:latest)를 빌드한다

```

jiseon@jiseon-Ubuntu:~$ docker build -t kubia .
[+] Building 21.8s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> transferring dockerfile: 102B
=> [internal] load .dockerignore
=> transferring context: 2B
=> [internal] load metadata for docker.io/library/node:7
=> [internal] load build context
=> transferring context: 348B
=> [1/2] FROM docker.io/library/node:7:sha256:a5c2c6ac8bc3fa372ac831ef60c45a285eba7bce9e9ed66dad3a01e29ab8d
=> resolve docker.io/library/node:7:sha256:a5c2c6ac8bc3fa372ac831ef60c45a285eba7bce9e9ed66dad3a01e29ab8d
=> sha256:a5c2c6ac8bc3fa372ac831ef60c45a285eba7bce9e9ed66dad3a01e29ab8d 2.01kB / 2.01kB
=> sha256:a5c2c6ac8bc3fa372ac831ef60c45a285eba7bce9e9ed66dad3a01e29ab8d 18.75B / 18.75B
=> sha256:2b32b8b8e8c213ad33783500183fc35841ec6478d1c481e613628ad9e00 19.20MB / 19.20MB
=> sha256:d9aed20b88a4a40a396dc03a881c43531097404838e4de68db7ed28f95974aa 7.21kB / 7.21kB
=> sha256:a98b32516edd1566ed6ef069454e0925772e0b0c203192c4602c682e06ba 43.23MB / 43.23MB
=> sha256:3245b5a1c32cbf0ae23d948f094ef7b321e3dc54e13c3f6c7f9951ed8237f03e 131.89MB / 131.89MB
=> sha256:a5a875743392fc2e79375c44e3ef2857775b722bbf27d01f8fe3789144e977fc 4.38kB / 4.38kB
=> sha256:9f09f21641cd120516153447e21bad87648cde5e87bcb6edc112a9fa3 119.15kB / 119.15kB
=> sha256:3746ad268b6c6c0ba3f4322c4736446e25d3517408373ecce8df44e9993 15.40MB / 15.40MB
=> sha256:49c0ed398b49ee8dec473e3277b89a88d79a5eeebb96c0e1a49069630c7bb 900.58kB / 900.58kB
=> extracting sha256:a074af05f5a240cf9459a1c7718628c2ae0b587eb510beef639aca3e566f
=> extracting sha256:2b32b8b8e8c213ad33783500183fc35841ec6478d1c481e613628ad9e00
=> extracting sha256:a98b32516edd1566ed6ef069454e0925772e0b0c203192c4602c682e06ba
=> extracting sha256:3245b5a1c32cbf0ae23d948f094ef7b321e3dc54e13c3f6c7f9951ed8237f03e
=> extracting sha256:a5a875743392fc2e79375c44e3ef2857775b722bbf27d01f8fe3789144e977fc
=> extracting sha256:9f09f21641cd120516153447e21bad87648cde5e87bcb6edc112a9fa3
=> extracting sha256:3746ad268b6c6c0ba3f4322c4736446e25d3517408373ecce8df44e9993
=> extracting sha256:49c0ed398b49ee8dec473e3277b89a88d79a5eeebb96c0e1a49069630c7bb
=> [2/2] ADD app.js /app.js
=> exporting to image
=> exporting layers
=> writing image sha256:908672eb2a10f2ee369f3b32225c9a6a4172a269dc32a29c3e9d3b5b363505
=> naming to docker.io/library/kubia
=> pushing to docker.io/library/kubia
=> push image: docker.io/library/kubia:latest

```

이미지 레이어의 이해

도커 이미지 : 여러개의 레이어로 구성

- 서로 다른 이미지가 여러개 레이어를 공유
- **node:7** 이미지가 다른 이미지에서 쓰이다라도 기본 이미지를 구성하는 모든 레이어는 단 한번만 저장됨
- 이미지를 가져올때도 각 레이어를 개별적으로 다운로드 (저장되지 않은 레이어만 다운로드)

이미지 빌드시..

1. 기본 이미지의 모든 레이어를 가져온 다음 도커는 그 위에 새로운 레이어를 생성하고 app.js 파일을 그 위에 추가
2. 이미지가 실행할때 수행돼야 할 명령을 지정하는 또 하나의 레이어를 추가
 - 이 마지막 레이어는 **kubia:latest** 라고 태그 지정

로컬에 저장된 이미지 리스트 조회

```
docker images
```

```
jiseonsim@simjiseon-ui-MacBook-Air ~ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
busybox	latest	a9d583973f65	43 hours ago	1.23MB
mysql	5.7	d54bd1054823	12 days ago	449MB
mysql	latest	8457e9155715	12 days ago	546MB

5. 컨테이너 이미지 실행

kubia 이미지에서 kubia-container라는 이름의 새 컨테이너 실행

```
docker run --name kubia-container -p 8080:8080 -d kubia
```

- 컨테이너는 콘솔에서 분리돼(`-d kubia`) 백그라운드에서 실행됨을 의미
- 로컬 머신의 8080 포트 ↔ 컨테이너 내부의 8080 포트와 매핑됨 (`-p 8080:8080`)
 - <http://localhost:8080> 으로 애플리케이션에 접근 가능

```
jiseonsim@simjiseon-ui-MacBook-Air ~/Desktop/git/KubeStudy-practice docker run --name kubia-container -p 8080:8080 -d kubia
21724f4003af39459c5473326e099d2b6bf2061ad2e1418954b22bed8ae48944
jiseonsim@simjiseon-ui-MacBook-Air ~/Desktop/git/KubeStudy-practice
```

애플리케이션 접근하기

```
curl localhost:8080
```

```
jiseonsim@simjiseon-ui-MacBook-Air ~/Desktop/git/KubeStudy-practice curl localhost:8080
You've hit 21724f4003af
```

이 16진수는 도커 컨테이너의 ID (≠ 호스트 머신의 호스트 이름)

실행 중인 모든 컨테이너 조회하기

```
docker ps
```

```
jiseonsim@simjiseon-ui-MacBook-Air ~/Desktop/git/KubeStudy-practice$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
21724f4003af	kubia	"node app.js"	4 minutes ago	Up 4 minutes	0.0.0.0:8080->8080/tcp	kubia-container
2db819050552	mysql:5.7	"docker-entrypoint.s..."	3 days ago	Up 4 hours	0.0.0.0:3306->3306/tcp, 33060/tcp	spring-batch-monitoring

컨테이너에 관한 추가 정보 얻기

`docker ps` 보다 자세한 정보를 출력해준다

```
docker inspect kubia-container
```

```
jiseonsim@simjiseon-ui-MacBook-Air ~/Desktop/git/KubeStudy-practice$ docker inspect kubia-container
[
  {
    "Id": "21724f4003af39459c5473326e099d2b6bf2061ad2e1418954b22bed8ae48944",
    "Created": "2021-03-11T17:00:06.9553869Z",
    "Path": "node",
    "Args": [
      "app.js"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 6821,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2021-03-11T17:00:08.1026893Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "Image": "sha256:920b72eb2a10f2ee369f3b32225c59a6a41f2a2b9dc32a29c3e9d3bc5b3635b5",
    "ResolvConfPath": "/var/lib/docker/containers/21724f4003af39459c5473326e099d2b6bf2061ad2e1418954b22bed8ae48944/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/21724f4003af39459c5473326e099d2b6bf2061ad2e1418954b22bed8ae48944/hostname",
    "HostsPath": "/var/lib/docker/containers/21724f4003af39459c5473326e099d2b6bf2061ad2e1418954b22bed8ae48944/hosts",
    "LogPath": "/var/lib/docker/containers/21724f4003af39459c5473326e099d2b6bf2061ad2e1418954b22bed8ae48944/21724f4003af39459c5473326e099d2b6bf2061ad2e1418954b22bed8ae48944-json.log",
    "Name": "/kubia-container",
    "RestartCount": 0,
  }
]
```

6. 실행중인 컨테이너 내부 탐색하기

추가 프로세스를 실행해서 컨테이너 내부를 살펴보자

실행중인 컨테이너 내부에서 셸 실행해서 컨테이너 내부의 프로세스 조회

`node.js` 는 bash shell을 포함하고 있으므로 컨테이너 내부에서 셸 실행 가능

```
docker exec -it kubia-container bash
ps aux
ps aux | grep app.js
ls -al
```

```
jiseonsim@simjiseon-ui-MacBook-Air ~/Desktop/git/KubeStudy-practice$ docker exec -it kubia-container bash
root@21724f4003af:/# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.1  1.3 614436 26512 ?        Ssl   17:00   0:00 node app.js
root      11   1.1  0.1  20248  3196 pts/0    Ss   17:08   0:00 bash
root      17   0.0  0.1  17504  2064 pts/0    R+   17:09   0:00 ps aux
root@21724f4003af:/#
```

7. 컨테이너 중지과 삭제

```
docker stop kubia-container // kubia-container 중지
docker rm kubia-container // 컨테이너 삭제
```

8. 이미지 레지스트리에 이미지 푸시

원격에서도 로컬에 저장된 이미지를 사용할 수 있게끔 외부 이미지 저장소에 이미지를 푸시하자

dockerhub.com 에 superjisonic으로 계정 생성

추가 태그로 태그 이미지 지정

컨테이너 이미지는 여러개의 태그를 가질 수 있다

```
docker tag kubia superjisonic/kubia
```

```
jiseonsim@simjiseon-ui-MacBook-Air ~/Desktop/git/KubeStudy-practice$ docker tag kubia superjisonic/kubia
jiseonsim@simjiseon-ui-MacBook-Air ~/Desktop/git/KubeStudy-practice$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
superjisonic/kubia	latest	920b72eb2a10	19 minutes ago	660MB
kubia	latest	920b72eb2a10	19 minutes ago	660MB
busybox	latest	a9d583973f65	44 hours ago	1.23MB
mysql	5.7	d54bd1054823	12 days ago	449MB
mysql	latest	8457e9155715	12 days ago	546MB
docker/getting-started	latest	3c156928aee	10 months ago	24.8MB

잘 추가됐다..

도커 허브에 이미지 푸시

도커 데스크탑에서 도커허브 아이디로 로그인한 뒤

```
docker push superjisonic/kubia
```

```
jiseonsim@simjiseon-ui-MacBook-Air ~/Desktop/git/KubeStudy-practice docker push superjisonic/kubia
Using default tag: latest
The push refers to repository [docker.io/superjisonic/kubia]
bc374d40b03f: Pushed
ab90d83fa34a: Mounted from library/node
8ee318e54723: Mounted from library/node
e6695624484e: Mounted from library/node
da59b99bbd3b: Mounted from library/node
5616a6292c16: Mounted from library/node
f3ed6cb59ab0: Mounted from library/node
654f45ecb7e3: Mounted from library/node
2c40c66f7667: Mounted from library/node
latest: digest: sha256:a00c4d3c0cd58cffb0e3bf7ba22fcf7e6ea5b9420c6d8373d4c9e031314e07ae size: 2213
```

다른 머신에서 이미지 실행하기

(다른 머신에서 이렇게 명령하면된다)

```
docker run -p 8080:8080 -d superjisonic/kubia
```

호스트 머신에 `node.js` 가 설치되어 있어도 이미지 내부에 설치된 것을 사용한다.