

Docker 로 Node.js 배포하기

22 AUGUST 2016 on node.js, docker, nginx, 도커, 노드, 엔진엑스



얼마전 Dockercon 16 이 성공적으로 막을 내린걸로 알고 있다. 바야흐로 Docker 세상이 도래하고 있다. Docker 는 영어권에서는 다커로 발음하고 있는 것 같으니 다커로 발음하시면서 읽으시면 되겠다. 도커로 하셔도 되지만...

Node.js 와 Single-threaded 모델

본격적으로 Docker 이야기로 넘어가기 전에 Node.js 먼저 언급해야겠다. Node.js 를 묘사하는 대표적인 키워드 3가지 1) single-threaded 2) asynchronous non-block I/O 3) event-driven 를 아마 귀에 딱지 붙을 정도로 많이 들었을텐데 다른 것보다 싱글 스레드에 집중해보자.

~~다~~이는 얘기를 한번 더 하자면 대표적인 웹서버인 Apache 와 같은 경우 전부 한 리퀘스트를 처리하는 동안 다른 리퀘스트가 들어오면 새롭게 스레드가 만들어져서 처리하게 된다. 물론 이 스레드를 무한정 늘릴순 없는것이고, 스레드가 꽉 차면 더이상 처리할 수 없어 클라이언트는 서버에서의 대답을 기다려야 하는 hang 이 발생하게 된다.

node.js 반면 멀티 스레드를 지원하지 않는다. 그럼 드는 생각은 "아니 그럼 접속을 두명이 하면 멈출텐데?" 물론 그렇지 않다. 브라우저와 비슷한 형태로 Event-loop 를 사용해 비동기 방식으로 멈추지 않고 응답을 기다렸다가 리턴 하는 방식으로 동시 접속을 처리하게 된다. 여기서 다루려고 하는 내용은 아니니 자세한 건 아래 Youtube 동영상을 참조하자. ~~반 정도 번역했는데 귀찮아서 요즘 못하고 있...~~

Philip Roberts: What the heck is the event loop anyway? | JSConf EU 2014



왜 굳이 아파치 얘기까지 꺼내와서 이런 이야기를 하나면, node.js 는 싱글 스레드 결국 하나의 cpu core 만 사용한다는 이야기인데, 요즘 시대가 어느 시댄대 64 core 서버가 준비한 가운데 코어를 하나만 쓴다니 무슨 이 시대 착오적인 소리인가 싶다. 물론 실제로 서비스할때는 nginx 를 앞단에 두고 reverse proxy 와 함께 load balancing 을 하는 경우가 대다수이겠지만, 새로운 서버에 배포를 해야 하는 경우가 생기거나 하면 매우 귀찮은게 사실이다.

Docker 와 Virtualization

Docker 는 복잡하게는 Linux Container 어찌구로 들어가지만 쉽게 생각하면 OS 코어와 가까운 **가벼운 가상머신** 정도라고 생각하면 편하다. 이렇게 설명하는 것이 Docker 를 지나치게 단순화 시키는 것이긴 하지만, Docker 의 개념을 이해하기 보다는 사용하면서 느끼는 것에 좀 더 집중하고 싶다. Docker 에 대해서 더 궁금한 사람은 가장 빨리 만나는 Docker 원고가 웹으로 공개되어 있으니 참고하자.

앞서 이야기했던 것처럼 Node.js 어플리케이션은 80 포트로 바인딩해서 사용하기 보다는 리버스 프록싱을 사용하게 된다. 그러다보니 대표적인 비동기 웹서버인 Nginx 가 쉽고 성능이 좋아 궁합이 잘 맞다. 또한 Nginx 는 자체적으로 Load Balancing 을 지원하기 때문에 호스트 하나에 동일한 서비스를 여러개 띄워서 사용할 수도 있다. 다만, 한번 배포시에 여러 서비스를 관리해야 하기 때문에 일반적인 케이스에서는 문제가 될 여지가 별로 없겠지만, 긴급 패치 시, 호스트를 추가해야하는 경우 등에서 여러가지 복잡한 절차를 거쳐야 하는 것이 사실이다.

이 포스트에서는 Docker 를 이용하여 Nginx 웹 서버를 통해 여러개의 동일한 Node.js 어플리케이션을 배포하는 방법을 알아보려 한다.

Node.js Application

샘플 어플리케이션을 만들어보자.

```
$ npm init -f
```

npm init 명령어로 프로젝트를 생성하자. 프로젝트와 큰 관계가 없는 설정 부분은 생략한다. **express** 를 이용하여 간단한 API 서버를 만들고, 각 API 서버를 식별하기 위한 **uuid** 를 생성하기 위해 각각 패키지를 설치한다.

```
$ npm i --save express uuid
```

다음과 같이 **index.js** 파일을 작성하자.

index.js

```
// 디펜던시
var express = require('express');
var uuid = require('uuid');

var app = express();
var id = uuid.v4();
var port = 3000;

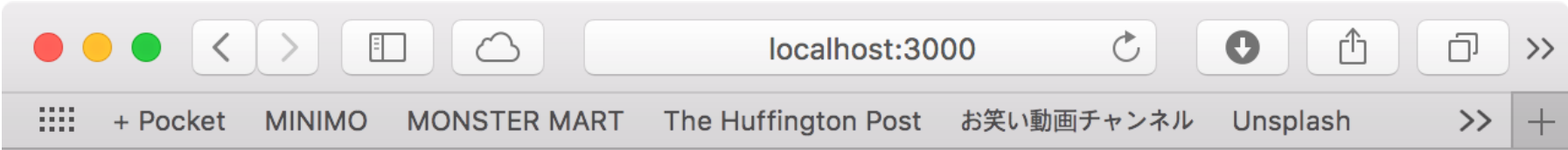
app.get('/', function (req, res) {
  res.send(id)
});

app.listen(port, function () {
  console.log('Example app listening on port: ' + port);
});
```

이제 서버를 실행한다.

```
$ node index.js
```

그리고 브라우저를 통해 `http://localhost:3000` 로 접속하면 다음과 같은 내용이 출력된다.
uuid 를 서버 생성할때 한번만 만들기 때문에 여러번 새로고침을 해도 동일한 내용이 출력된다.



31229714-9165-4a18-be3d-ad9339165391

테스트를 완료하면 앱을 중지하고 다음 단계로 넘어가자.

Docker 로 Node.js 어플리케이션 실행하기

다음은 만들어진 Node.js 어플리케이션을 Docker 를 통해 실행해보자. 설치 방법은 [이곳](#) 을 참고한다. Docker 는 기본적으로 `Dockerfile` 를 레시피로 이미지를 생성한다. Docker 는 기본적으로 [Docker Hub](#) 에 있는 이미지를 베이스 이미지로 해서 만들게 되는데, 공식 [Node.js](#) 이미지가 있으니 그것을 사용하도록 하자.

먼저 `node_modules` 을 직접 복사하는 일이 없도록 `.dockerignore` 파일을 작성한다.

.dockerignore

```
node_modules/
```

Dockerfile

```
FROM node:6
COPY package.json /src/package.json
RUN cd /src; npm install
COPY . /src
EXPOSE 3000
WORKDIR /src

CMD node index.js
```

작성이 끝나면 아래 명령어를 실행시킨다.

```
$ docker build --tag node-nginx:test .
```

Docker 는 위에서부터 차례차례 실행시키며 이미지를 생성한다.

```
$ docker build --tag node-nginx:test .
Sending build context to Docker daemon 7.168 kB
Step 1 : FROM node:6
6: Pulling from library/node
357ea8c3d80b: Already exists
52befadefd24: Already exists
3c0732d5313c: Pull complete
ceb711c7e301: Pull complete
868b1d0e2aad: Pull complete
61d10f626f84: Pull complete
Digest: sha256:12899eea666e85f23e9850bd3c309b1ee28dd0869f554a7a6895fc962d9094a3
Status: Downloaded newer image for node:6
----> 800da22d0e7b
Step 2 : COPY package.json /src/package.json
----> 7f3344975b1e
Removing intermediate container ae1d0482e982
Step 3 : RUN cd /src; npm install --production
----> Running in 222a0585301b

...
```

```
Successfully built 08a5b1c92fcf
```

위와 같은 명령어가 출력되면 제대로 생성되었는지 확인해보자.

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
node-nginx           test               08a5b1c92fcf       About a
minute ago          654.5 MB
```

만들어진 이미지를 실행한다.

```
$ docker run --name node-nginx-instance -p 3000:3000 node-nginx:test
Example app listening on port: 3000
```

다시 브라우저로 <http://localhost:3000> 으로 접속하면 동일한 결과가 나오는 것을 확인할 수 있다. 물론 uuid 는 변경된다. 터미널을 하나 더 열어 `$ docker ps` 로 확인해보자.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS             PORTS              NAMES
30179995521c       node-nginx:test    "/bin/sh -c 'node ind"   About a minute ago
Up About a minute   0.0.0.0:3000->3000/tcp   node-nginx-instance
```

`$ docker run` 이 열려있는 터미널에서 `ctrl + c` 로 중지하거나 `$ docker stop node-nginx-instance` 로 실행되고 있는 인스턴스를 중지한다.

```
$ docker rm node-nginx-instance
```

이제 여러개의 인스턴스를 동시에 실행해보자. 그전에 위처럼 만들어져있는 instance 를 삭제한다.

```
$ docker run -d --name node-nginx-instance-0 -p 3000:3000 node-nginx:test
$ docker run -d --name node-nginx-instance-1 -p 30001:3000 node-nginx:test
$ docker run -d --name node-nginx-instance-2 -p 3002:3000 node-nginx:test
```

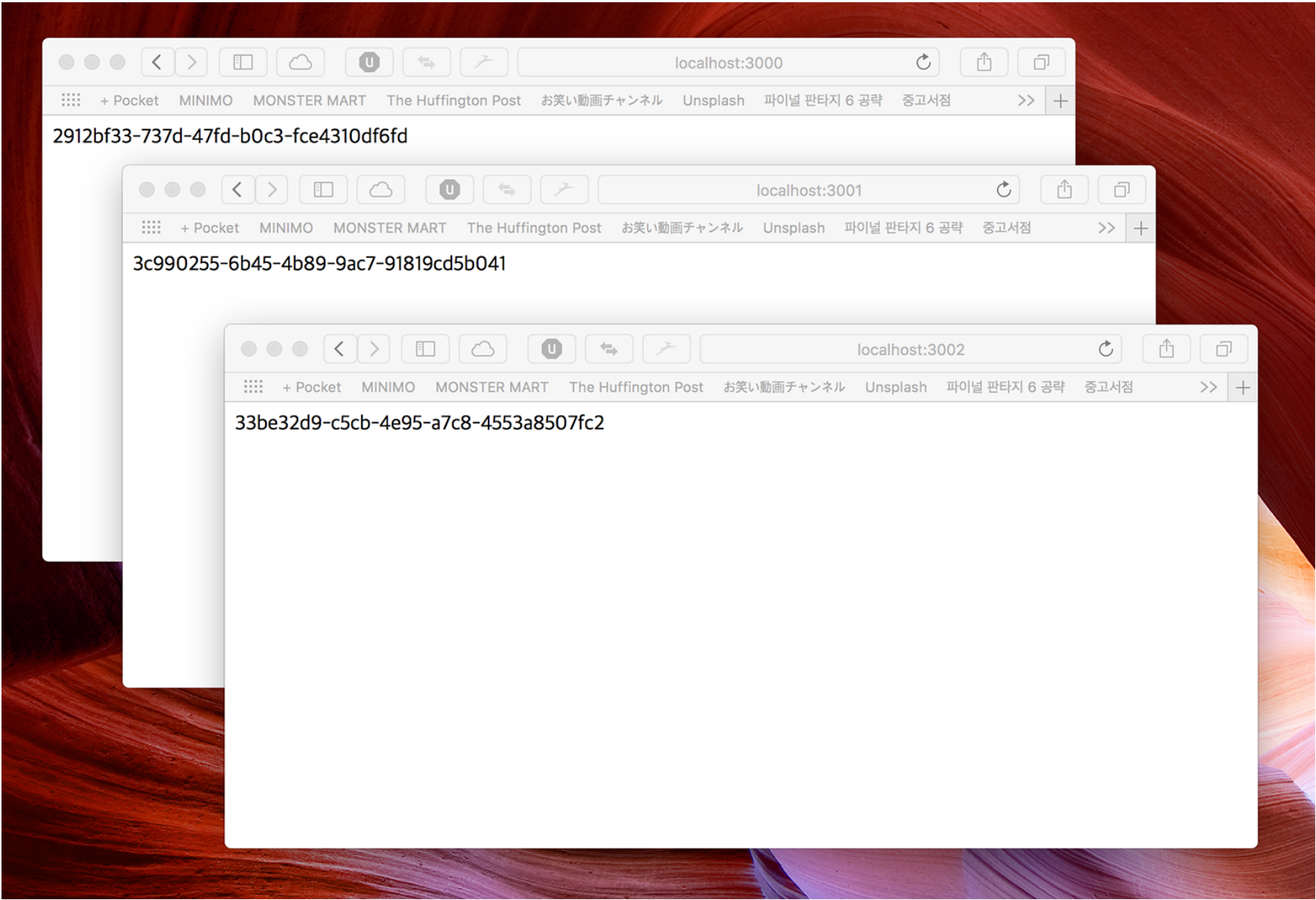
`-d` 옵션을 주어 데몬의 형태로 인스턴스로 만들어주었다.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS             PORTS              NAMES
450a19e72bb4       node-nginx:test    "/bin/sh -c 'node ind"   32 seconds ago
Up 30 seconds      0.0.0.0:3002->3000/tcp   node-nginx-instance-2
```



```
d2f5ec891c75      node-nginx:test      "/bin/sh -c 'node ind"    37 seconds ago
Up 35 seconds      0.0.0.0:3001->3000/tcp  node-nginx-instance-1
a38935fd7d4f      node-nginx:test      "/bin/sh -c 'node ind"    51 seconds ago
Up 50 seconds      0.0.0.0:3000->3000/tcp  node-nginx-instance-0
```

세 인스턴스가 돌아가고 있다. 브라우저로 접속해서 확인해보자.



NGINX 로 Load Balancing

다음과 같이 `nginx/` 폴더 아래에 `nginx.conf` 파일을 작성하자. 단 `upstream` 의 `server` 설정에서 `YOUR_IP_ADDRESS` 를 현재 호스트의 IP 로 지정해준다. <http://ifconfig.co> 와 같은 사이트를 사용하면 편리하다.

nginx/nginx.conf

```
worker_processes 4;

events { worker_connections 1024; }

http {
    upstream node-app {
        least_conn;

        server YOUR_IP_ADDRESS:3000 weight=10 max_fails=3 fail_timeout=30s;
        server YOUR_IP_ADDRESS:3001 weight=10 max_fails=3 fail_timeout=30s;
        server YOUR_IP_ADDRESS:3002 weight=10 max_fails=3 fail_timeout=30s;
    }

    server {
```

```
listen 80;

location / {
    proxy_pass http://node-app;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
}
}
```

Dockerfile 도 만들어준다. 역시 Docker hub 의 공식 이미지를 사용하였다.

nginx/Dockerfile

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
```

실행해보자.

```
$ cd nginx
$ docker build --tag node-nginx-lb:test .
$ docker run -d --name node-nginx-instance-lb -p 4000:80 node-nginx-lb:test
```

이제 <http://localhost:4000> 로 접속하면 똑같은 화면을 볼 수 있다. 단 이 상태에서 새로고침을 하면 세 인스턴스를 번갈아 가며 접속하는 것을 확인할 수 있다.

```
$ docker ps
CONTAINER ID      IMAGE               COMMAND             CREATED
STATUS           PORTS              NAMES
68588d475364     node-nginx-lb:test  "nginx -g 'daemon off'" 3 minutes ago
Up 3 minutes     443/tcp, 0.0.0.0:4000->80/tcp  node-nginx-instance-lb
450a19e72bb4     08a5b1c92fcf       "/bin/sh -c 'node ind" 18 minutes ago
Up 18 minutes    0.0.0.0:3002->3000/tcp          node-nginx-instance-2
d2f5ec891c75     08a5b1c92fcf       "/bin/sh -c 'node ind" 18 minutes ago
Up 18 minutes    0.0.0.0:3001->3000/tcp          node-nginx-instance-1
a38935fd7d4f     08a5b1c92fcf       "/bin/sh -c 'node ind" 18 minutes ago
Up 18 minutes    0.0.0.0:3000->3000/tcp          node-nginx-instance-0
```

docker-compose up

뭔가 이상하다. 로컬에서 로드밸런싱을 하고 있음에도 Docker instance 는 각각의 localhost 를 가지고 있기 때문에 localhost 에 바인딩 하는 방식으로는 nginx 를 사용할 수 없다. 보통 192.168 로 시작하는 내부 IP 로 접근이 가능하긴 하지만 이 역시 바뀔때마다 새로 설정해줘야 하는 번거로움이 있다.

docker-compose 는 docker 를 실행할때마다 입력해줘야 하는 설정들 `-d -p -name` 을 파일로 관리할 수 있게 하는 해준다. `docker-compose` 를 설치하고 좀 더 편리하게 서버를 실행해보자.

우선 현재 실행되고 있는 모든 컨테이너를 삭제한다.

```
$ docker rm -f $(docker ps -a -q)
```

docker-compose 는 `Python` 으로 작성되어 있기 때문에 설정파일은 `.yml` 형식을 따른다.

docker-compose.yml

```
version: '2'

services:
  nginx:
    container_name: node-nginx-lb
    build: ./nginx
    links:
      - app-1:app-1
      - app-2:app-2
      - app-3:app-3
    ports:
      - 3000:80
    depends_on:
      - app-1
      - app-2
      - app-3

  app-1:
    container_name: node-nginx-1
    image: node-nginx:test
    ports:
      - 3000

  app-2:
    container_name: node-nginx-2
    image: node-nginx:test
    ports:
      - 3000

  app-3:
    container_name: node-nginx-3
    image: node-nginx:test
    ports:
      - 3000
```


그리고 `nginx/nginx.conf` 의 IP 를 다음과 같이 수정해준다.

nginx/nginx.conf

```
...

upstream node-app {
    least_conn;
    server app-1:3000 weight=10 max_fails=3 fail_timeout=30s;
    server app-2:3000 weight=10 max_fails=3 fail_timeout=30s;
    server app-3:3000 weight=10 max_fails=3 fail_timeout=30s;
}

...
```

이제 다시 원래 `node` 프로젝트 디렉토리로 돌아와서 `docker-compose` 명령어를 실행하자.

```
$ cd ..
$ docker-compose up
Recreating node-nginx-3
Recreating node-nginx-1
Recreating node-nginx-2
Recreating node-nginx-lb
Attaching to node-nginx-1, node-nginx-2, node-nginx-3, node-nginx-lb
node-nginx-1 | Example app listening on port: 3000
node-nginx-2 | Example app listening on port: 3000
node-nginx-3 | Example app listening on port: 3000
node-nginx-lb | 172.19.0.1 - - [22/Aug/2016:16:07:32 +0000] "GET / HTTP/1.1" 200 36
node-nginx-lb | "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12) AppleWebKit/602.1.50 (KHTML,
node-nginx-lb | like Gecko) Version/10.0 Safari/602.1.50"
node-nginx-lb | 172.19.0.1 - - [22/Aug/2016:16:07:32 +0000] "GET / HTTP/1.1" 200 36
node-nginx-lb | "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12) AppleWebKit/602.1.50 (KHTML,
node-nginx-lb | like Gecko) Version/10.0 Safari/602.1.50"
node-nginx-lb | 172.19.0.1 - - [22/Aug/2016:16:07:33 +0000] "GET / HTTP/1.1" 200 36
node-nginx-lb | "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12) AppleWebKit/602.1.50 (KHTML,
node-nginx-lb | like Gecko) Version/10.0 Safari/602.1.50"
```

이제 <http://localhost:3000> 으로 접속 하면 각 컨테이너에 번갈아 접속하면서 로그를 출력하는 것을 확인할 수 있다. 최종적인 `docker` 컨테이너 상태는 다음과 같다.

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
8b0fe632e458	dockernodenginx_nginx	"nginx -g 'daemon off'"	1 seconds ago

Up Less than a second	443/tcp, 0.0.0.0:3000->80/tcp	node-nginx-lb
6dafacd1e4d2	node-nginx:test	"/bin/sh -c 'node ind" 2 seconds ago
Up 1 seconds	0.0.0.0:32776->3000/tcp	node-nginx-2
cf6bb53c5397	node-nginx:test	"/bin/sh -c 'node ind" 2 seconds ago
Up 1 seconds	0.0.0.0:32775->3000/tcp	node-nginx-3
618b74662d16	node-nginx:test	"/bin/sh -c 'node ind" 2 seconds ago
Up 1 seconds	0.0.0.0:32774->3000/tcp	node-nginx-1

현재 로컬에 `node-nginx:test` 이미지가 존재하기 때문에 별도의 추가 다운로드 없이 `docker-compose` 는 3개의 Node.js 어플리케이션 container 와 1개의 nginx 로드 밸런서를 생성한 후 실행한다. 만약 로컬에 이미지가 존재하지 않을 경우 docker hub 에서 다운로드를 시도한다.

정리

Docker 는 서버를 관리하고 배포를 구성할때 매우 편리한 도구이다. 최근에 출시된 `docker swarm` 이나 `Kuberanates` 와 같은 서비스를 활용하면 여러대의 가상서버에 서비스를 자유자재로 배포하기 매우 편리하다. 현재 쏘카에서는 ZEROCAR API 서버를 위와 유사한 형태로 구성하여 ~~이보다 훨씬 복잡하지만~~ 사용하고 있다.

배포의 편리함은 물론이거니와 이제 윈도우 및 맥용 `docker` 베타 버전 도 출시된 만큼, 클라이언트 개발자에게 디펜던시 걱정없이 개발 서버를 설정할 수 있게 하는 것도 가능하다. 또한 스테이징 서버를 거의 완전하게 동일한 형태로 운영하거나, 서버를 추가할 때 배포 자동화를 매우 편리하다는 점 등 최근 `DevOps` 기술 스택에서 가장 각광받고 있다고 해도 과언이 아니다. 특히 Node.js 의 경우 싱글 스레드 컴퓨팅의 한계를 아주 편리하게 극복할 수 있다 보니 적극적으로 사용하면 좋을 것이다.

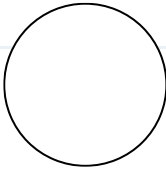
이 포스트에서 사용한 코드는 <https://github.com/colusoo1/docker-nginx-node> 에서 확인할 수 있다.

참고자료




- [Github Repo](#)
- [A sample Docker workflow with Nginx, Node.js and Redis](#)

Seokjun Kim

Read [more posts](#) by this author.



Share this post



15 Comments

Seokjun's Blog

Login

Recommend 2

Share

Sort by Best



Join the discussion...



jinsang • 2 months ago

우선 글 감사합니다.

궁금한게요 위에 node-nginx와 -lb를 각각 docker hub에 올린 뒤
다른 컴퓨터에서 pull 땡겼을 때 yml 파일이 없이 image 파일만 받는데요 이때는 docker-compose up
을 어떻게 하나요?

1 ^ | v • Reply • Share ›



Seokjun Kim Mod → jinsang • 2 months ago

yml 파일은 직접 만드시면 됩니다. docker-compose 자체는 python 으로 스크립트 러닝하는
어플리케이션이에요. 배포 자동화를 위해서면 scp 같은걸로 올리시는게 좋습니다.

^ | v • Reply • Share ›



jinsang → Seokjun Kim • 2 months ago

빠른 답변 감사합니다.

node-nginx는 1개의 node js 컨테이너를 위한 docker image를 허브로 배포하고

다른 컴퓨터에서 위 이미지를 docker hub로부터 pull 받아 docker-compose.yml,
nginx의 Dockerfile, nginx.conf를 작성해서 docker-compose up으로 하니까 되네요

1 ^ | v • Reply • Share ›



Seokjun Kim Mod → jinsang • 2 months ago

넵! 그렇게 하시면 됩니다~

^ | v • Reply • Share ›



Yunseop Kim • 2 months ago

안녕하세요. Seokjun님, 포스팅 감사합니다.

포스팅 해주신대로 진행하다가 도중에 막히는 부분이 있어 질문드립니다.

nginx 로드밸런식 부분에서부터 막히는데요,

```
docker run -d --name node-nginx-instance-lb -p 4000:80 node-nginx-lb:test
```

위 명령 실행시에 없는 이미지라고 뜹니다. 뒤에 '-lb' 를 제거하고 하니까 예제에서 설명하신바와 같
이 실행이 되더군요.

docker-compose.yml 같은 경우에는

docker-compose 를 apt-get install로 인스톨 후에 실행했는데...

ERROR: In file './docker-compose.yml' service 'version' doesn't have any configuration options. All
top level keys in your docker-compose.yml must map to a dictionary of configuration options.

위와 같은 메시지가 뜹니다.

혹시 무엇이 문제인지 아신다면... 답변좀 부탁드립니다!!

1 ^ | v • Reply • Share ›



Seokjun Kim Mod → Yunseop Kim • 2 months ago

우선

1) build 할때 `docker run -d --name node-nginx-instance-lb -p 4000:80 node-nginx-lb:test` 윗
부분 코드에 오타가 있었네요. 포스트는 수정했습니다. `node-nginx-lb:test` 요게 이미지 네
임이라서 build 시 tag 뒤의 node-nginx-lb:test 로 맞춰주시면 됩니다. 빌드 부분을 아래 처럼
해주세요.

```
$ docker build --tag node-nginx-lb:test .
```

2) docker-compose 는 apt-get 으로 설치하는 건 아니고 pip install 로 설치하는 python 모듈
입니다. 우선 이걸 확인해주시면 될 것 같고. `docker-compose.yml` 의 경우는 오타가 있으
신건 아닌지 <https://github.com/colus001/do...> 랑 비교해보시면 좋을 것 같습니다. 이래도 안
되시면 댓글 다시 달아주세요!

^ | v • Reply • Share ›



Yunseop Kim → Seokjun Kim • 2 months ago

석준님, 답변 감사합니다. 덕분에 해결이 됐습니다~!

docker-compose 같은 경우에는 <https://docs.docker.com/compos...> 에도 친절히 설명이 되어있었는데 이놈의 귀차니즘 때문에 apt-get install 해서 엉뚱한걸 깔아버린것 같네요^^;;

친절한 설명 감사드리고, 좋은 하루 되시길 바랍니다~!!

1 ^ | v • Reply • Share ›



Seokjun Kim Mod ➔ Yunseop Kim • 2 months ago

본의아니게 QA 해주셔서 감사합니다! :D

^ | v • Reply • Share ›



Francis Kim • 2 months ago

글 감사합니다! pm2도 비슷한 기능이 있는데 써보지는 않았습니다.

1 ^ | v • Reply • Share ›



Seokjun Kim Mod ➔ Francis Kim • 4 days ago

pm2 에서 클러스터링을 지원하긴 하는데요. docker 에서는 pm2 나 forever 같은 프로세스 관리 툴을 지양해서 (docker 자체의 restart 룰과 충돌) 저는 안쓰고 있습니다. 물론 docker 에서는 dockerfile 을 이용한 서버 환경 자동화 같은 다른 장점도 있구요.

^ | v • Reply • Share ›



Francis Kim ➔ Seokjun Kim • 4 days ago

AWS의 ELB를 사용해서 autoscaling을 하는게 차라리 나을거같다는 생각이 문득 드는군요.

^ | v • Reply • Share ›



Seokjun Kim Mod ➔ Francis Kim • 3 days ago

물론 로드밸런싱만 보면 그렇습니다. 하지만 배포 환경 설정이나 자동화를 위해서 Docker 를 사용하게 되는게 맞고, 외려 이런 케이스에는 ELB 에 ECS 를 쓰면 더욱 더 좋겠조.

^ | v • Reply • Share ›



Francis Kim ➔ Seokjun Kim • 3 days ago

Docker도 사용중입니다만, ELB + ECS는 아직 못해봤군요. 댓글 감사드립니다!

^ | v • Reply • Share ›



jinsang • a month ago

궁금증이 있습니다.

만약 node-nginx에서 오류가 생겼을 경우

docker logs --follow를 통해 실시간으로 볼수도 있지만 지나친것은 볼수가 없지 않습니까?

node-nginx-lb의 error.log를 봐야 할텐데

어떻게 볼 수 있나요?

어떻게 하면 node-nginx-lb 컨테이너 내의 error.log, access.log를 볼 수 있나요?

^ | v • Reply • Share ›



Seokjun Kim Mod ➔ jinsang • a month ago

그건 볼륨으로 연결해서 호스트 OS 에 error.log 와 access.log 를 쌓으면 됩니다. 이 프로젝트의 목적과는 부합하지 않아서, 언급하지는 않았는데요. 저도 실제 프로젝트에서 사용할때에는 컨테이너 상의 nginx 로그는 따로 기록하지 않아도 (다른 방식으로 기록), node 로그는 호스트 OS 의 /var/log/ 에 저장하고 있습니다. 마찬가지로 nginx 로그도 필요하면 볼륨을 연결하는 것으로 기록 가능합니다. 자세한 내용은 <https://docs.docker.com/engine...> 를 참고하세요.

^ | v • Reply • Share ›

ALSO ON SEOKJUN'S BLOG

나는 어떻게 개발자가 되었나?

5 comments • 2 years ago•

Seung — 잘 읽었습니다. 중간중간 맞춤법 오류가 조금 거슬렸지만 (ex. 안되 -> 안돼, 썼지만 -> 썼지만 등) 그래도 재미있게 읽었네요.

김석준 개발자 출사표

5 comments • a year ago•

hyaline — 인상깊네요^_^ 편하실 때 식사대접을 한 번 하고 싶은데 괜찮으실까요:D? 참고로 저는 게임을 만들고 있고, 회사에서 프라모델 ...

React Native in Production

5 comments • 3 months ago•

손명우 — 글 잘읽었습니다^^혹시 react-native로 개발 하시면서 안드로이드 화면전환은 어떤식으로 구현하셨나요?자체 지원하는 navigator를 ...

1억짜리 창업지원사업을 포기하며

44 comments • 2 years ago•

Guest — 공감이 되는내용도 있고 다른 시각으로 보면 저도 지원을 받고 2-3년정도 되었어요 10년정도 창업하고 직장생활하고 했다가 지원을 ...

READ THIS NEXT

HAProxy 와 Nginx 의 로드밸런싱

NGINX Nginx 는 대표적인 웹서버인 Apache 의 문제점을 해결하면서 만들어진 웹서버로 비동기 방식으로 개발되어 가볍고 빠른 것으로 유명한 오픈소스...

Make It Yourself © 2016

YOU MIGHT ENJOY

React Native in Production

최근 쏘카는 새로운 서비스 제로카를 발표하였고, 100명 모집에 10,000명이 몰리는 등 관심과 인기를 한몸에 받고 있다. 관련 블로그...

Proudly published with **Ghost**