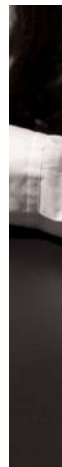
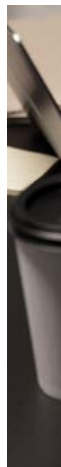


항공권 조회 서비스를 위한 웹 개발 및 서버 구축

Team Daisy 박효주 우슬기 정한솔 한찬희



목차

01

프로젝트 개요

02

인프라 구성

03

웹 개발 및 배포

04

DB 구성

05

서버 모니터링

01 프로젝트 개요

Project Overview



프로젝트 배경

- 코로나 종식을 기원하며 앞으로 여행의 수요가 높아질 것을 예상해 다수의 접속으로 인한 트래픽 증가를 감당할 수 있는 항공권 조회 사이트를 구축

프로젝트 목표

- Private Docker Registry (Harbor) 구축
 - DNS 서버 구축
 - Docker Swarm을 이용한 쉽고 빠른 분산 서버 관리
 - node.js 이용한 웹 개발
 - 도커 스택을 배포하여 유동적인 서비스 스케일 변동 및 업데이트
 - DB Replication으로 데이터 동기화
 - 서버 모니터링 시스템 (Telegraf + Influxdb + Grafana) 구현
-



박효주

- 전체 프로젝트 총괄
- 기본 인프라 구성
- DNS 서버 구축
- 사설 저장소 구축



정한솔

- DB 스키마 설계
- DB Replication 구성



우슬기

- 모니터링 구현
- 알람 설정



한찬희

- 웹 개발 총괄
 - 서비스 이미지화, 배포
 - ppt 관리
-

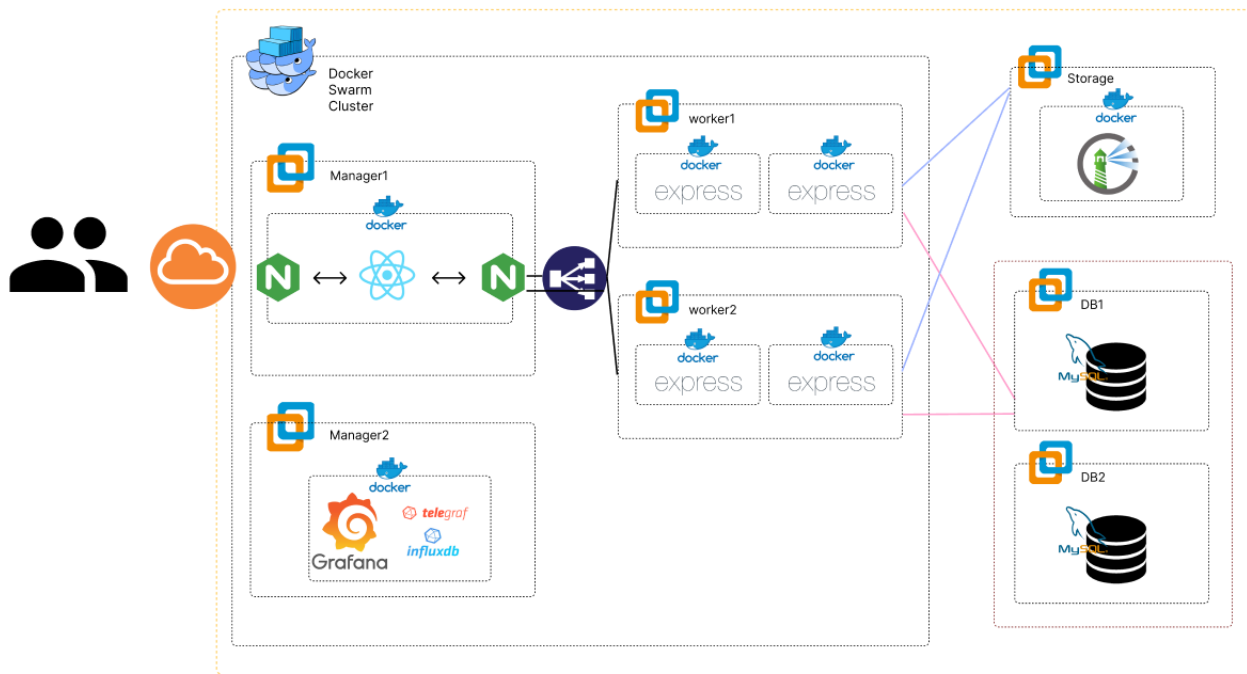
Platform



Software



	Manager1	Manager2	Worker1	Worker2	DB1	DB2	Storage
CPU	4 Cores	4 Cores	2 Cores	2 Cores	2 Cores	2 Cores	2 Cores
Memory	4 GB	4 GB	2 GB	2 GB	2 GB	2 GB	2 GB
HDD	20 GB	20 GB	20 GB	20 GB	20 GB	20 GB	80 GB
NIC - bridge (10.5.104.0/8)	.101	.102	.103	.104	.105	.106	.107



02 인프라 구성

Infrastructure Configuration
By TonyHan(한찬희)





```
[root@mgmtadb ~]# mkdir /root/.ssh
```

```
[root@mgmtadb ~]# chmod 700 /root/.ssh
```

```
[root@mgmtadb ~]# ssh-keygen -q -N ""
```

```
[root@mgmtadb ~]# scp ~/.ssh/id_rsa.pub
```

```
10.5.104.102~104:~/.ssh/authorized_keys
```

설정을 통해 manager1에서 나머지 서버를 원격으로 관리할 수 있게 해주었다.

```
[root@manager1 ~]# ssh root@manager2 ls
anaconda-ks.cfg
swarm-daisy
swarm-tig
```

```
[root@manager1 ~]# ssh root@worker1 ls
anaconda-ks.cfg
dead.letter
grafana-enterprise-8.4.4-1.x86_64.rpm
telegraf-1.13.4-1.x86_64.rpm
test.txt
```

```
[root@manager1 ~]# ssh root@worker2 ls
anaconda-ks.cfg
daisy
grafana-enterprise-8.4.4-1.x86_64.rpm
```



```
[root@mgmtadb .ssh]# vi /etc/ssh/ssh_config
```

mgmtadb에서는 ssh_config파일의 맨 마지막에

StrictHostKeyChecking no

을 넣어 관리서버에서 다른 서버에게 접속할 때 패스워드를 묻지 않도록 했다.

```
[root@manager1 ~]# cat /etc/ssh/ssh_config | grep Strict
# StrictHostKeyChecking ask
    StrictHostKeyChecking no
```



```
[root@manager1 ~]# dockerswarminit--advertise-addr10.5.104.101
```

을 통해 manager1에서 발행한 managertoken과 workertoken으로 스웜 클러스터를 구성했다.

```
[root@manager1 ~]# ssh root@manager2 $(docker swarm join-token manager | grep SWMTKN)
```

```
[root@manager1 ~]# ssh root@worker1 $(docker swarm join-token worker | grep SWMTKN)
```

```
[root@manager1 ~]# ssh root@worker2 $(docker swarm join-token worker | grep SWMTKN)
```

```
[root@manager1 ~]# docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
mlrvqjr4a7if4wi2r3vplz8gj *	manager1	Ready	Active	Leader	20.10.14
de7kxvj6g7tfdzzt9d393q9f9	manager2	Ready	Active	Reachable	20.10.14
mti0f9vclmg0zw0pgrl35zg5c	worker1	Ready	Active		20.10.14
xak3qwc6y1ulqa8fwrogf8nld	worker2	Ready	Active		20.10.14



```
[root@pstorage certs]# ls
10.5.104.107.cert  10.5.104.107.csr  ca.crt  ca.srl
10.5.104.107.crt   10.5.104.107.key  ca.key  v3.ext
```

https 연결을 위해 CA Certificates 및 Server Certificates, 인증키 생성을 했다.

```
[root@pstorage certs]# cp 10.5.104.107.crt /etc/pki/ca-trust/source/anchors/harbor-server.crt
[root@pstorage certs]# cp ca.crt /etc/pki/ca-trust/source/anchors/harbor-ca.crt
[root@pstorage certs]# update-ca-trust
```

Linux에 신뢰할 수 있는 인증서 적용



```
[root@pstorage ~]# docker image push 10.5.104.107/daisy/daisy_httpd:1.0
The push refers to repository [10.5.104.107/daisy/daisy_httpd]
e825bfd70e3d: Layer already exists
59fde81347af: Layer already exists
818410a5e575: Layer already exists
1bf88df2ac46: Layer already exists
608f3a074261: Layer already exists
1.0: digest: sha256:10ed1591781d9fdbaeafaafef77067f12e833c699c84ed4e21706ccbd5229fd0a size: 1365
```

📦 daisy | System Admin

Summary Repositories Members Labels Scanner P2P Preheat Policy Robot Accounts Webhooks Logs C

✕ DELETE

<input type="checkbox"/>	Name	Artifacts	Pulls
<input type="checkbox"/>	daisy/daisy_httpd	1	0

자유롭게 이미지를 push / pull할 수 있다.



manager1에 DNS 서버를 구축하기 위해 bind 패키지를 설치한 후 설정을 진행한다.

etc/sysconfig/network-scripts/ifcfg-ens32 에

DNS2=10.5.104.101

<- 자신의 IP를 입력해 준다.

```
options {  
    listen-on port 53 { any; };  
    listen-on-v6 port 53 { none; };  
    directory      "/var/named";  
    dump-file      "/var/named/data/cache_dump.db";  
    statistics-file "/var/named/data/named_stats.txt";  
    memstatistics-file "/var/named/data/named_mem_stats.txt";  
    recursing-file  "/var/named/data/named.recursing";  
    secroots-file   "/var/named/data/named.secroots";  
    allow-query     { any; };
```

/etc/named.conf 파일에서

DNS 서비스를 제공 받을 IP를 : any로 변경

listen-on-v6 : IPv6는 사용하지 않기 때문에 none으로 변경

allow-query : 쿼리를 주고 받을 수 있는 IP -> any로 변경



/etc/named.rfc1912.zones

```
zone "triply.co.kr" IN {  
    type master;  
    file "triply.zone";  
    allow-update { none; };  
};  
  
zone "104.5.10.in-addr.arpa" IN {  
    type master;  
    file "triply.rev";  
    allow-update { none; };  
};
```



ForwardZone 파일 설정

/var/named/triply.zone

```
$TTL 600
@      IN SOA  triply.co.kr.  root.triply.co.kr. (
                                0      ; serial
                                10     ; refresh
                                1H     ; retry
                                1W     ; expire
                                3H )   ; minimum

      IN NS   triply.co.kr.
      IN A    10.5.104.101
www   IN A    10.5.104.101
```

NS: Name Server를 지정

A: 좌측에 작성한 도메인의 IP 지정

PTR: 역방향 DNS에 사용되는 레코드로 IP주소로 도메인을 찾는 데 사용

ReverseZone 파일 설정

/var/named/triply.rev

설정파일 검증 확인

```
$TTL 600
@      IN SOA  triply.co.kr.  root.triply.co.kr. (
                                0      ; serial
                                10     ; refresh
                                1H     ; retry
                                1W     ; expire
                                3H )   ; minimum

      IN NS   triply.co.kr.
101   IN PTR  triply.co.kr.
101   IN PTR  www.triply.co.kr.]
```

```
[root@manager1 ~]# named-checkconf /etc/named.conf
[root@manager1 ~]# named-checkzone triply.com /var/named/triply.zone
zone triply.com/IN: loaded serial 0
OK
```



```
[root@manager1 ~]# nslookup www.triply.co.kr
Server:      10.5.104.101
Address:     10.5.104.101#53
```

```
Name:  www.triply.co.kr
Address: 10.5.104.101
```

```
[root@manager1 ~]# nslookup 10.5.104.101
101.104.5.10.in-addr.arpa    name = triply.co.kr.
101.104.5.10.in-addr.arpa    name = www.triply.co.kr.
```

이제 도메인 이름을 통해 IP 주소를 찾아내거나
IP 주소를 통해 도메인 이름을 찾아낼 수 있다.

인터넷 프로토콜 버전 4(TCP/IPV4) 속성

일반

네트워크가 IP 자동 설정 기능을 지원하면 IP 설정이 자동으로 할당되도록 할 수 있습니다. 지원하지 않으면, 네트워크 관리자에게 적절한 IP 설정값을 문의해야 합니다.

☐ 자동으로 IP 주소 받기(O)

☒ 다음 IP 주소 사용(S):

IP 주소(I): 10 . 5 . 1 . 121

서브넷 마스크(U): 255 . 0 . 0 . 0

기본 게이트웨이(D): 10 . 0 . 0 . 1

☐ 자동으로 DNS 서버 주소 받기(B)

☒ 다음 DNS 서버 주소 사용(E):

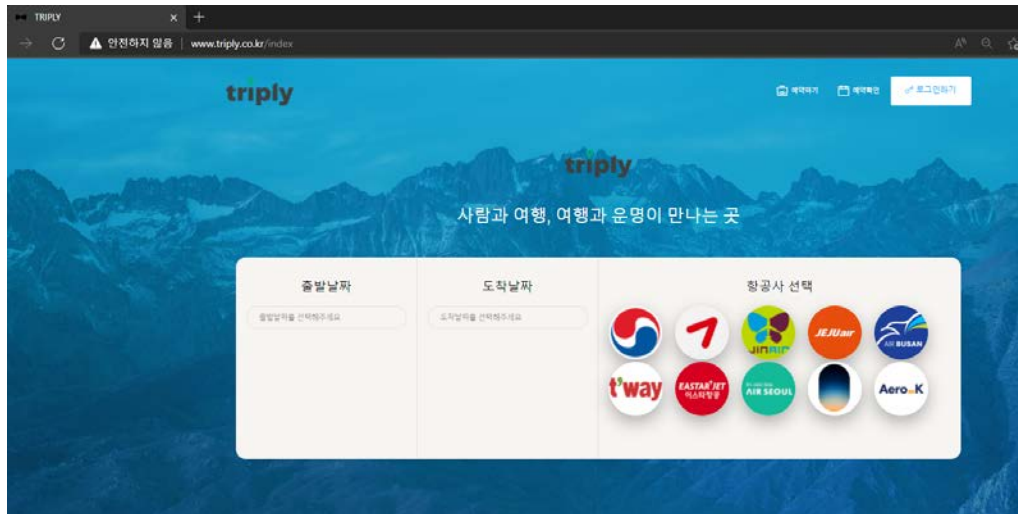
기본 설정 DNS 서버(P): 10 . 5 . 104 . 101

보조 DNS 서버(A): 8 . 8 . 8 . 8

☐ 끝낼 때 설정 유효성 검사(L) 고급(V)...

확인 취소

사용 중인 WIN10의 설정에 구축한 DNS 서버 주소를 넣는다.



실제 PC의 웹 브라우저에서 도메인 이름으로 접속할 수 있게 되었다.

03 웹 개발, 배포

Web Development and Deployment





웹개발을 어떻게 하면 모두가 이해할 수 있게 할 수 있을까?

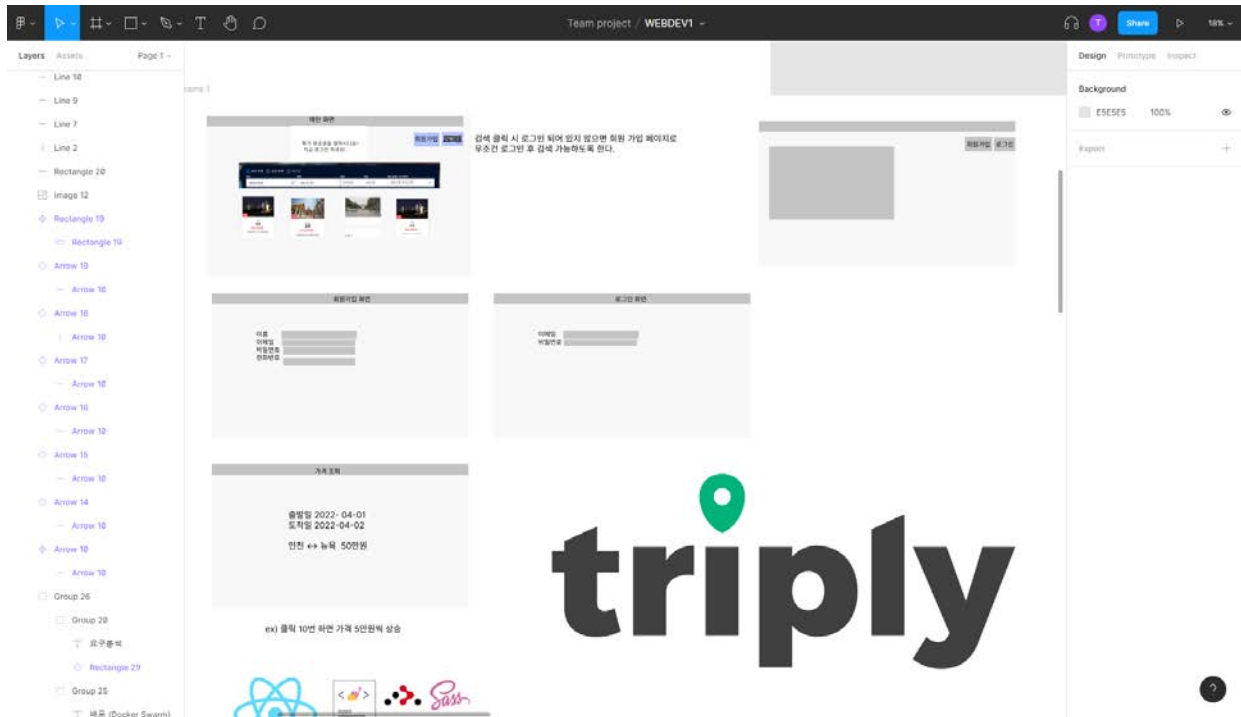


Daisy
DDD(Domain Driven Design)

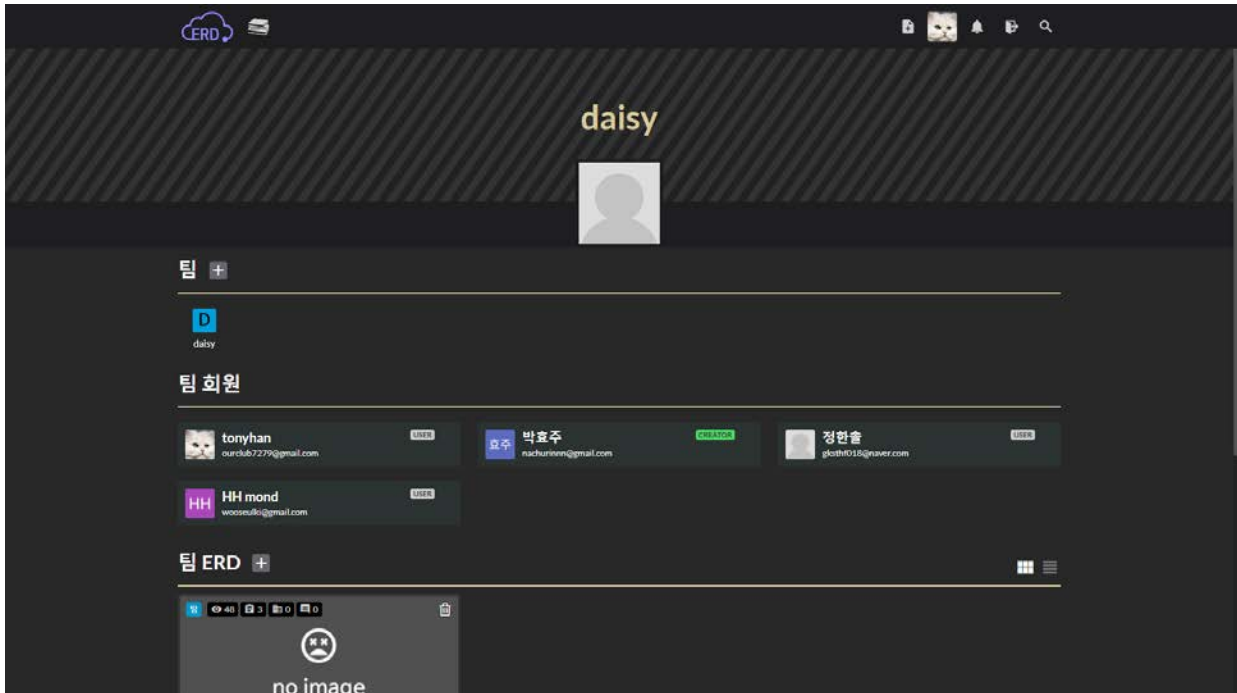


DDD 프로세스

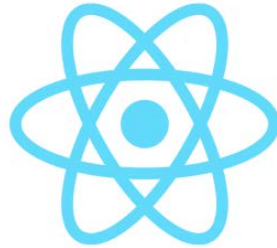
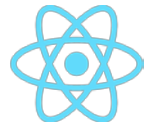
1. 필요한 기능 정리
2. 필요한 DB 정의
3. 필요한 시스템 정의
4. 연관기능 묶기



각자 생각하는 웹페이지를 디자인하고 합의점을 구하기



DDD에서 도출된 DB를 DBCLOUD에서 직접 ERD 만들기
자세한건 DB 파트에서

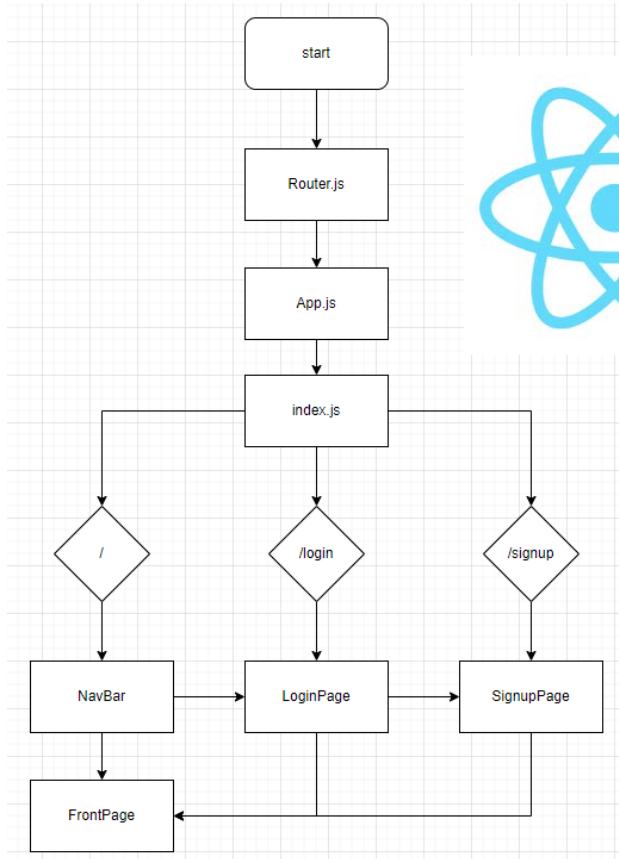


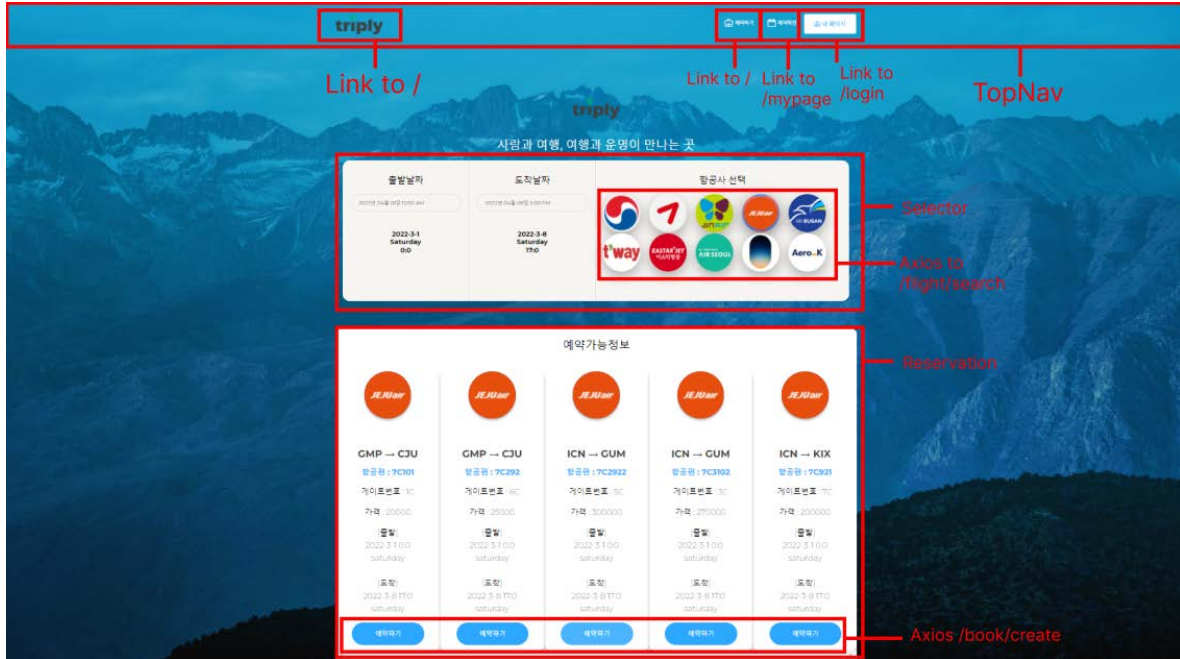
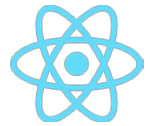
⇒ axios



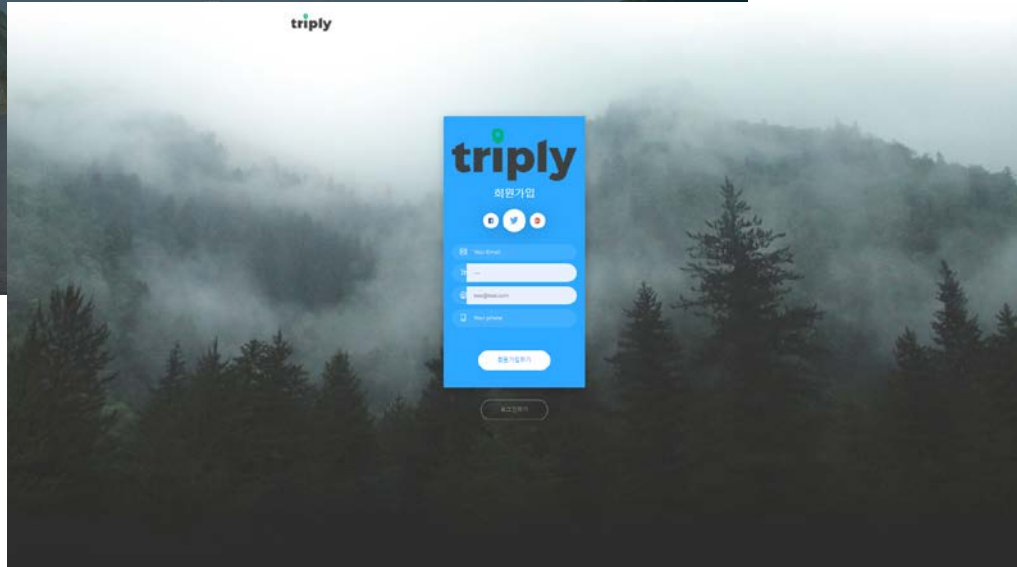
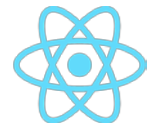
DDD를 보고

1. 필요한 페이지 계획
2. 컴포넌트 설계
3. 필요한 패키지 체크





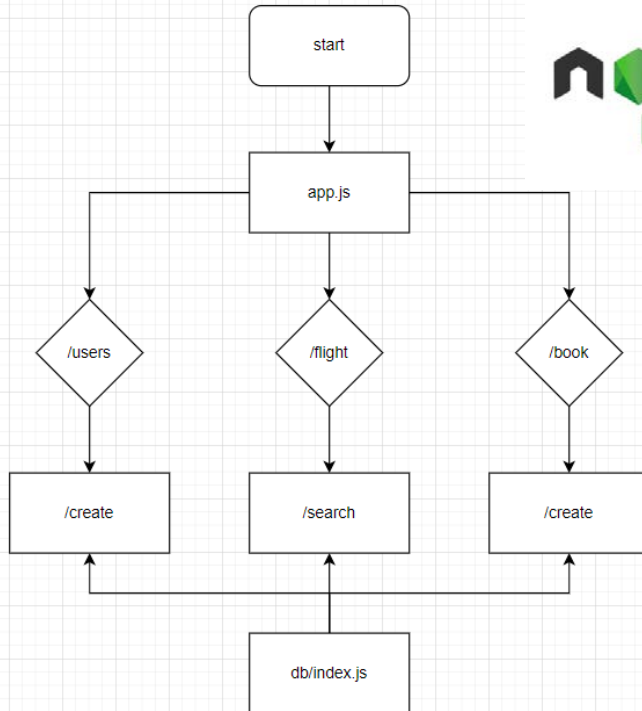
설계도를 보고 컴포넌트와 Axios 통신 위치 결정



Login, Signup 페이지 개발

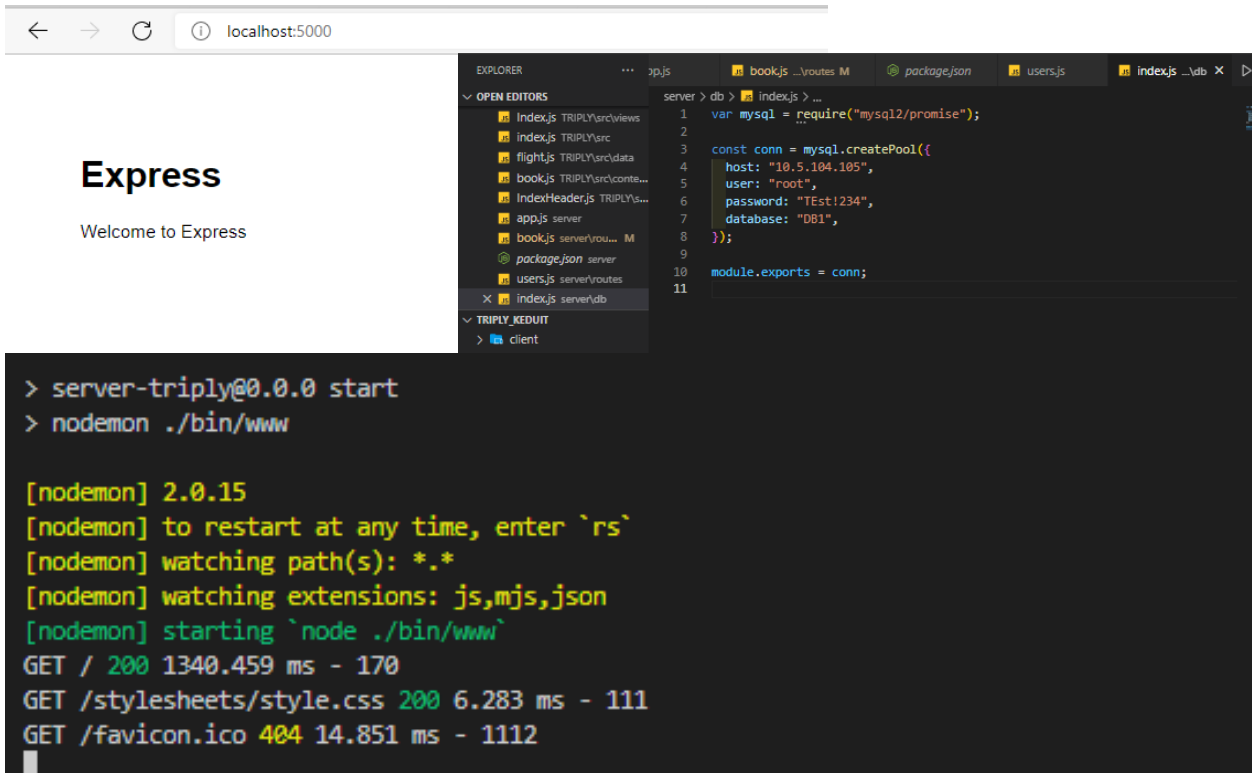


express

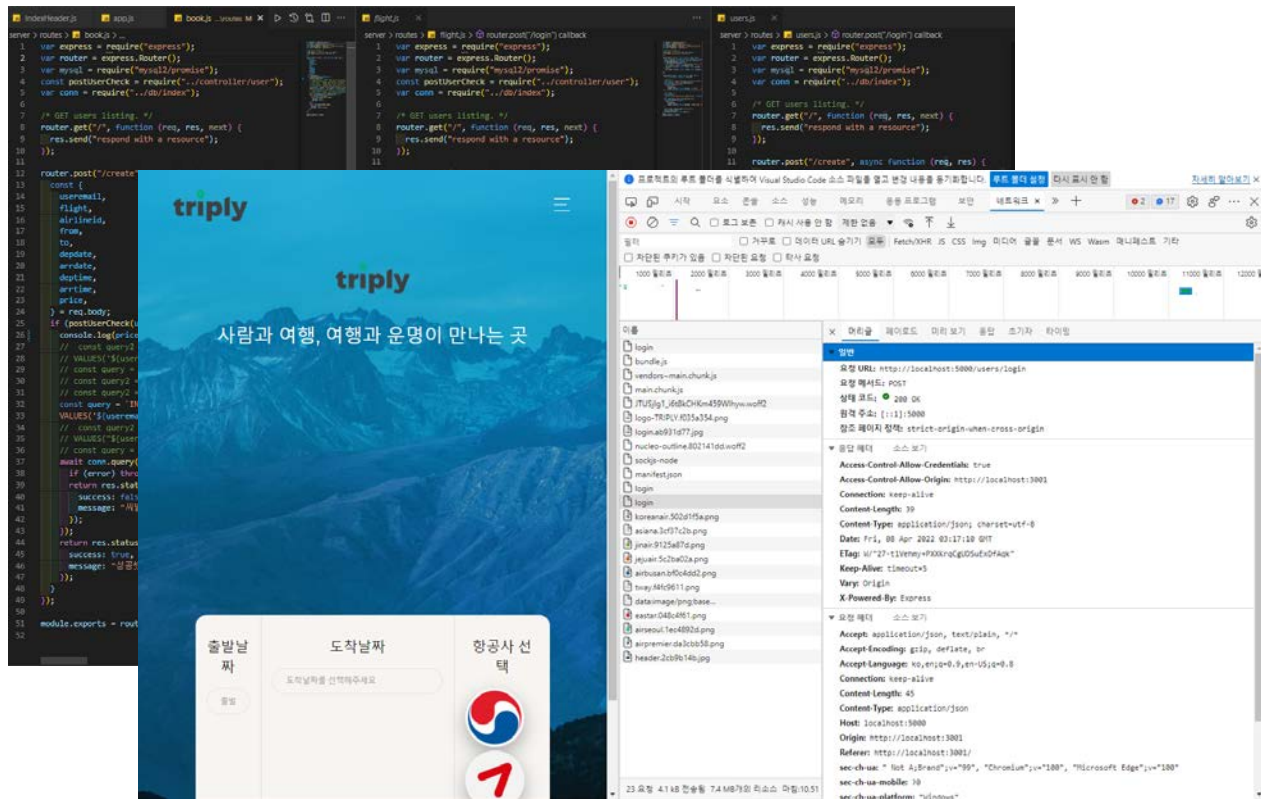


DDD를 보고

1. 필요한 도메인 고르기
2. 필요한 패키지 체크



Express-React 연결 확인 + DB 연결확인



The image is a composite of three parts illustrating the development and deployment of an Express.js application.

Top Part: Code Snippets

```

server > routes > book.js > router.post('/login') callback
1 var express = require('express');
2 var router = express.Router();
3 var mysql = require('mysql2/promise');
4 const postUserCheck = require('../controller/user');
5 var conn = require('../db/index');
6
7 /* GET users listing. */
8 router.get('/', function (req, res, next) {
9   res.send('respond with a resource');
10 });
11
12 router.post('/create')
13 const {
14   username,
15   flight,
16   airline,
17   from,
18   to,
19   departure,
20   arrival,
21   departure,
22   arrival,
23   price,
24 } = req.body;
25 if (postUserCheck) {
26   console.log('create user');
27   // const query2 =
28   // VALUES('user',
29   // const query =
30   // VALUES('user',
31   const query = '
32   VALUES(' + user +
33   VALUES(' + user +
34   // const query2 =
35   // VALUES('user',
36   // const query =
37   multi conn.query(
38   if (error) then
39     return res.status(
40     success: false,
41     message: '실패'
42   );
43   });
44   return res.status(
45   success: true,
46   message: '성공'
47   );
48   });
49   });
50 }
51 module.exports = router;

```

Middle Part: Web Application Interface

The screenshot shows the 'triply' web application interface. The header features the 'triply' logo and a navigation menu. The main content area displays the text '사람과 여행, 여행과 운명이 만나는 곳' (Where people and travel, travel and destiny meet). Below this, there are three buttons: '출발날짜' (Departure Date), '도착날짜' (Arrival Date), and '항공사 선택' (Select Airline). The '도착날짜' button is highlighted with a tooltip that says '도착날짜를 선택하십시오' (Please select the arrival date).

Bottom Part: Visual Studio Code Interface

The screenshot shows the Visual Studio Code interface with the Express.js application code. The 'Output' panel is open, displaying the response of the application. The response is a JSON object containing various headers and body information.

```

{
  "Access-Control-Allow-Credentials": true,
  "Access-Control-Allow-Origin": "http://localhost:3001",
  "Connection": "keep-alive",
  "Content-Length": 39,
  "Content-Type": "application/json; charset=utf-8",
  "Date": "Fri, 08 Apr 2022 03:17:10 GMT",
  "ETag": "W/\"27-11vemy9X00xRqG05uex0FAq\"",
  "Keep-Alive": "timeout=4",
  "Vary": "Origin",
  "X-Powered-By": "Express",
  "body": {
    "username": "user",
    "password": "1234",
    "email": "user@example.com",
    "phone": "010-1234-5678",
    "address": "1234567890",
    "city": "Seoul",
    "country": "Korea",
    "created_at": "2022-04-08T03:17:10.000Z",
    "updated_at": "2022-04-08T03:17:10.000Z"
  }
}

```

도메인별 개발, DB와의 연동확인

tonyhan18 / TRIPLY_keduit

main

File	Last commit	Time ago
TRIPLY	last dance	2 days ago
server	last dance	2 days ago
(22장)Daisy_pdf.pdf	last dance	15 hours ago
(22장)Daisy_시연...	last dance	15 hours ago
.DS_Store	0401	12 days ago
.gitignore	update dir	16 days ago
README.md	Update README.md	15 hours ago
docker-compose.y...	last dance	2 days ago
final.gif	last dance	2 days ago
img-1.png	last dance	2 days ago

TRIPLY_KEDUIT

[개발회고] Docker Swarm(Docker Stack) + React + Express(Node Server) + Nginx + Reverse Proxy (tistory.com)

위의 링크로 가면 만든 것을 확인할 수 있다.

[이전내용](#)

사실상 위 프로젝트에서 client와 server만 현재 폴더로 바꾸어준 것이라고 보면 된다.

<https://tonyhan18.tistory.com/329>

About

No description, website, or topics provided.

Releases

No releases published
[Create a new release](#)

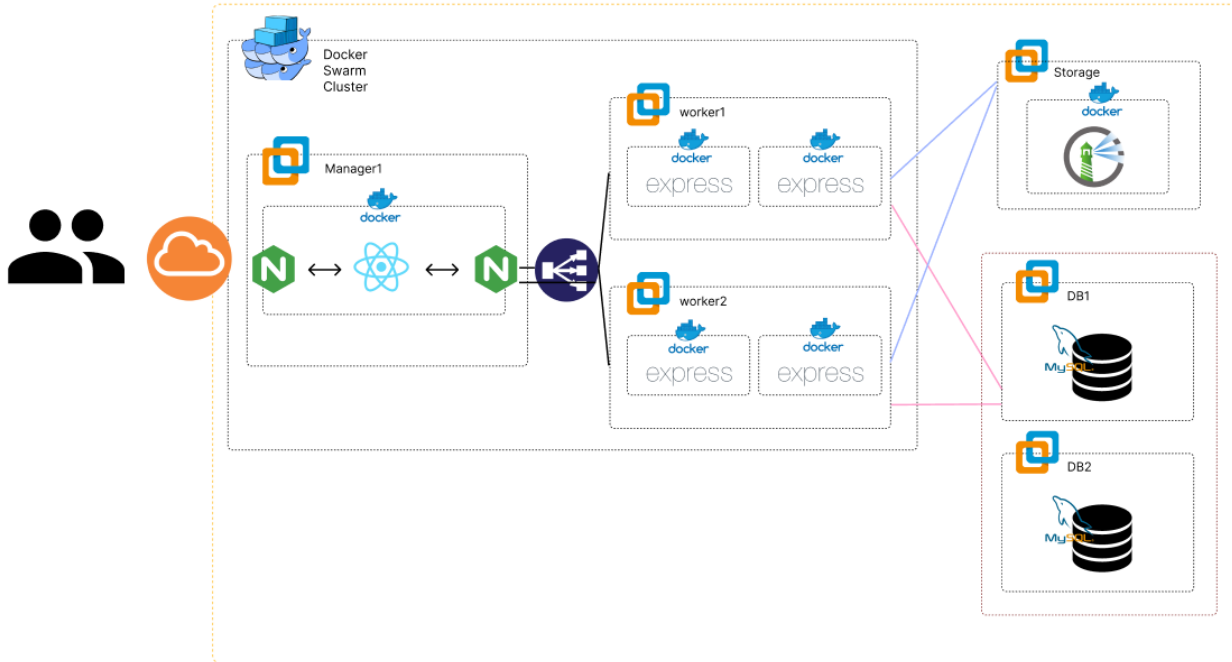
Packages

No packages published
[Publish your first package](#)

Languages

Language	Percentage
CSS	35.9%
JavaScript	34.9%
SCSS	28.7%
Other	0.5%

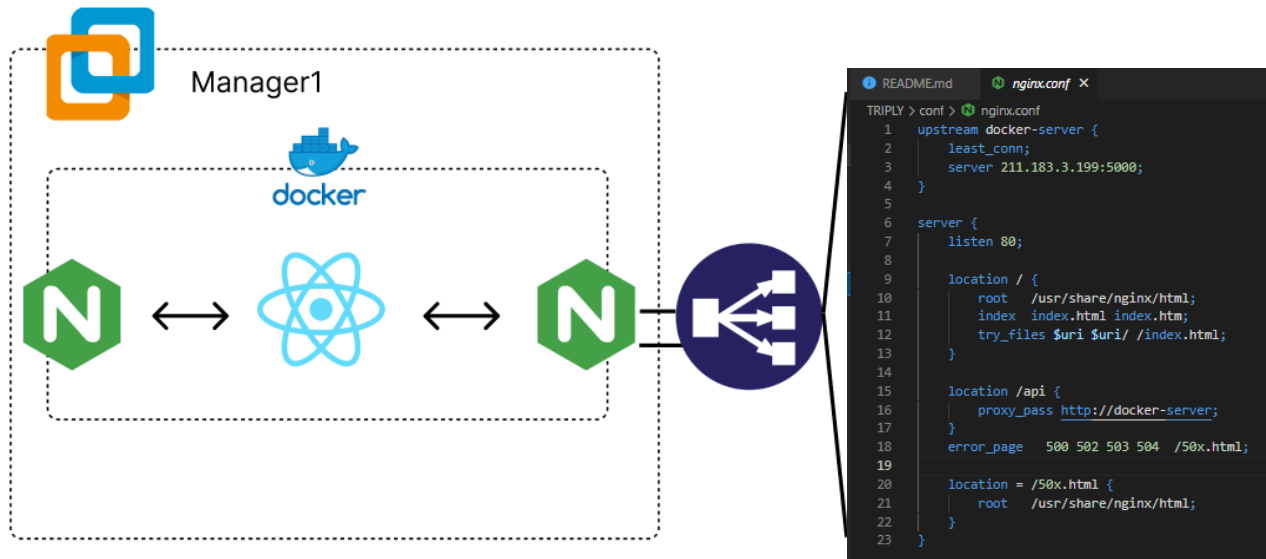
https://github.com/tonyhan18/TRIPLY_keduit



배포를 위한 전체 구조 설계



컨테이너화하면 둘 간의 통신이 안된다. 그리고 보안상 서버가 노출되는 취약점이 생긴다



Forward Proxy로 80번 포트 접근을 허용해 편의성을 늘리고
Reverse Proxy를 두어서 통신과 보안 두마리 토끼 잡기

컨테이너 환경에서 React와 Express 통신을 위한 Reverse Proxy 설계



23 lines (18 sloC) | 505 Bytes

```

1 ##### REACT to BUILD FILE
2 FROM node:16-alpine as builder
3
4 # # make environment
5 RUN mkdir -p /usr/src/app
6 WORKDIR /usr/src/app
7 COPY package.json .
8 RUN npm install --silent
9 RUN npm install react-scripts@5.0.0 -g --silent
10
11 # # COPY files
12 COPY . ./
13 RUN npm run build
14
15 ##### nginx CMD install
16 FROM nginx:latest
17 RUN rm -rf /etc/nginx/conf.d/default.d
18 COPY conf/nginx.conf /etc/nginx/conf.d/default.conf
19
20 COPY --from=builder /usr/src/app/build /usr/share/nginx/html
21
22 EXPOSE 80
23 CMD ["nginx", "-g", "daemon off;"]

```

Frontend

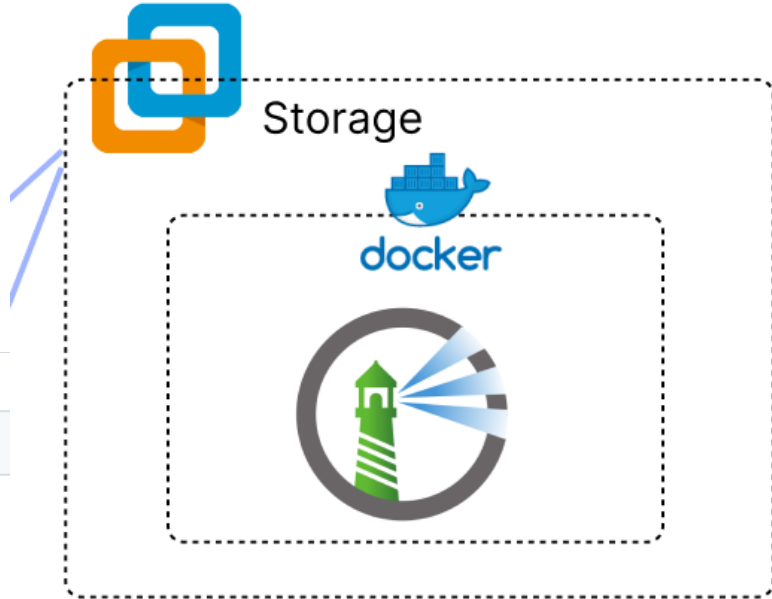
7 lines (7 sloC) | 93 Bytes

```

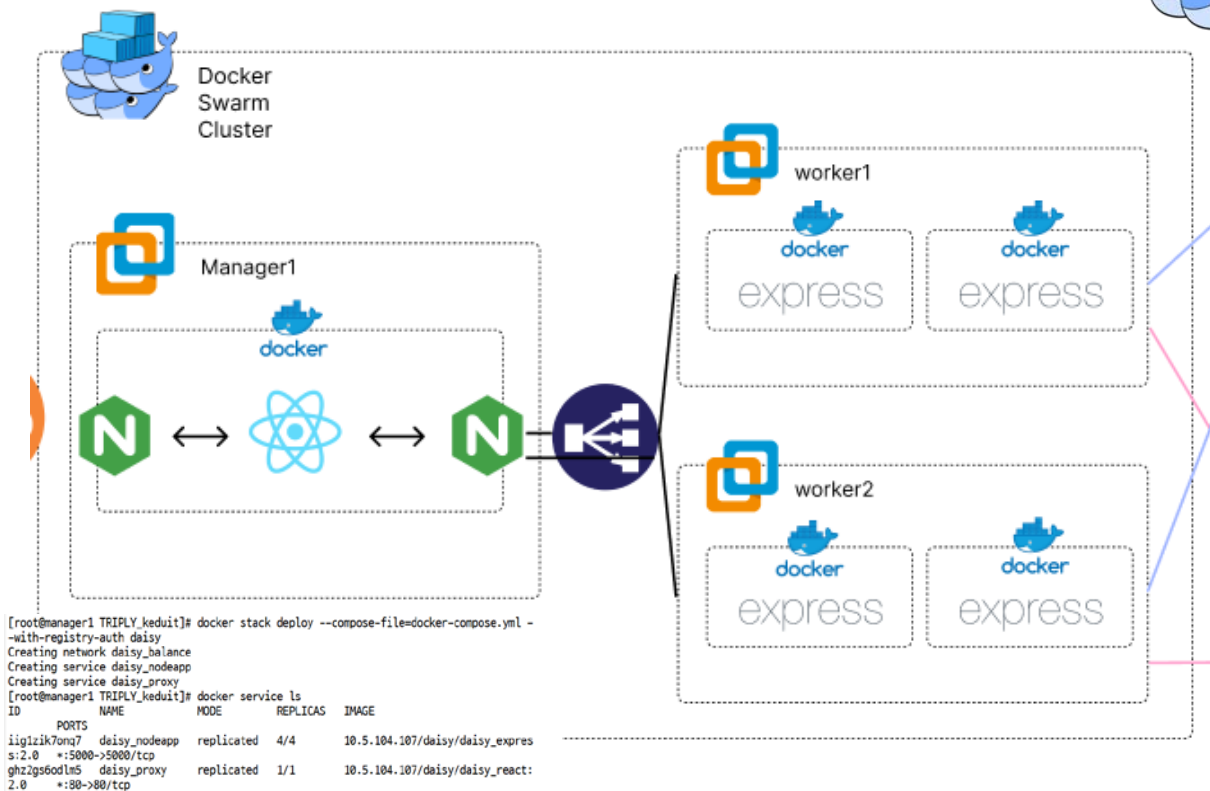
1 FROM node
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 COPY . .
6 CMD npm start
7 EXPOSE 5000

```

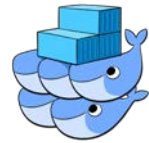
Backend



Docker Stack 배포를 위한 프론트와 백 이미지화
+ 사설 저장소에 업로드



배포 아키텍처 설계



35 lines (32 sloc) | 599 Bytes

```

1  version: "3"
2
3  services:
4    nodeapp:
5      image: tonyhan18/express:1.0
6      ports:
7        - 5000:5000
8      deploy:
9        replicas: 6
10       restart_policy:
11         max_attempts: 3
12         condition: on-failure
13       update_config:
14         parallelism: 3
15       delay: 10s
16       placement:
17         constraints: [node.role == worker]
18     networks:
19       - balance
20
21   proxy:
22     image: tonyhan18/react:1.0
23     ports:
24       - 80:80
25     depends_on:
26       - nodeapp
27     deploy:
28       placement:
29         constraints: [node.role == manager]
30     networks:
31       - balance
32
33   networks:
34     balance:
35       driver: overlay

```



Edit src/App.js and save to reload.

[Learn React](#)
 Data From
 15d41d1de76c
 Get List
 test1/테스트1
 test2/테스트2
 test3/테스트3
 test4/테스트4
 Get Item
 test1/테스트1

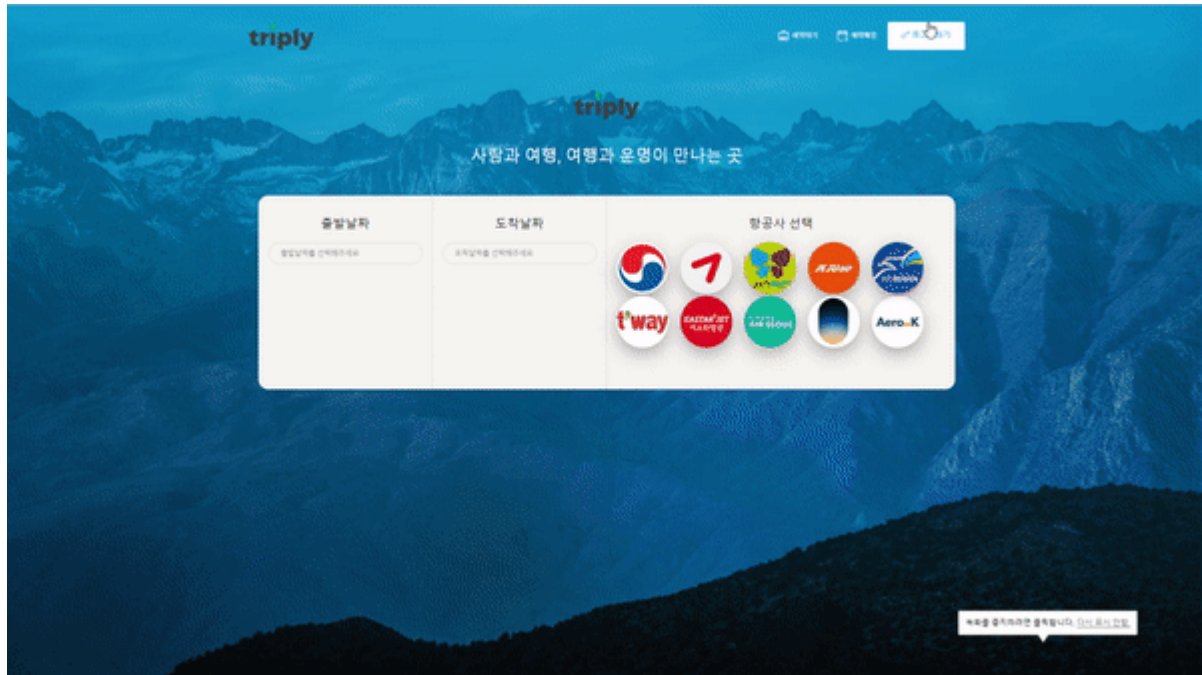
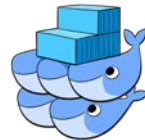


Edit src/App.js and save to reload.

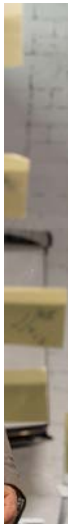
[Learn React](#)
 Data From
 c4c9f66adabd
 Get List
 test1/테스트1
 test2/테스트2
 test3/테스트3
 test4/테스트4
 Get Item
 test1/테스트1

Docker stack deploy -c docker-compose.yml swarm

Load Balancing 체크
 컴퓨터 OS의 정보가 뜨는 것을 확인할 수 있다.






정상 배포 확인



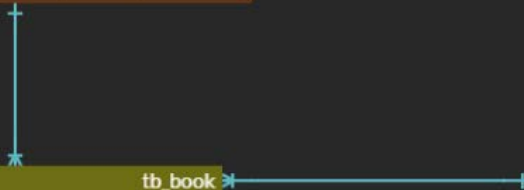
04 DB 구성

Database Configuration

사용자		tb_user		
	회원아이디	useremail	Domain	char(30)
	회원비밀번호	userpw	Domain	char(100)
	회원이름	username	Domain	char(10)
	회원전화번호	usertel	Domain	char(20)
	salt	salt	Domain	char(100)

예약 테이블		tb_book		
	예약번호	booknum	Domain	int
	회원아이디	useremail	Domain	char(30)
	항공편이름	flight	Domain	char(10)
	항공사	airlineid	Domain	char(3)
	탑승객명	username	Domain	char(10)
	출발지	start	Domain	char(5)
	도착지	end	Domain	char(5)
	출발날짜	deptime	Domain	date
	도착날짜	arrdate	Domain	date
	출발시간	deptime	Domain	time
	도착시간	arrtime	Domain	time
	가격	price	Domain	int

항공편		tb_flight		
	항공편이름	flight	Domain	char(10)
	여객기	airline	Domain	char(3)
	출발지	from	Domain	char(3)
	도착지	to	Domain	char(3)
	날짜	date	Domain	date
	출발시간	deptime	Domain	time
	도착시간	arrtime	Domain	time
	탑승게이트	gatenum	Domain	char(2)
	가격	price	Domain	int




```
mysql> desc tb_flight;
```

Field	Type	Null	Key	Default	Extra
flight	char(10)	NO	PRI	NULL	
airline	char(3)	YES		NULL	
from	char(3)	NO		NULL	
to	char(3)	NO		NULL	
deptime	time	YES		NULL	
arrtime	time	YES		NULL	
gatenum	char(2)	NO		NULL	
price	int	YES		NULL	

8 rows in set (0.01 sec)

사용자의 항공편 정보를 저장하는 테이블



```
mysql> desc tb_user;
```

Field	Type	Null	Key	Default	Extra
useremail	char(30)	NO	PRI	NULL	
userpw	char(100)	NO		NULL	
username	char(10)	NO		NULL	
usertel	char(20)	YES		NULL	
salt	char(100)	YES		0	

5 rows in set (0.00 sec)

사용자의 회원가입 정보를 저장하는 테이블



```
mysql> desc tb_book;
```

Field	Type	Null	Key	Default	Extra
booknum	int	NO	PRI	NULL	auto_increment
useremail	char(30)	NO	PRI	NULL	
flight	char(10)	NO	PRI	NULL	
airlineid	char(3)	YES		NULL	
username	char(10)	YES		NULL	
start	char(5)	NO		NULL	
end	char(5)	NO		NULL	
depdate	date	YES		NULL	
arrdate	date	YES		NULL	
deptime	time	YES		NULL	
arrtime	time	YES		NULL	
price	int	YES		NULL	

```
12 rows in set (0.01 sec)
```

tb_user의 useremail과
tb_flight의 flight를
tb_book의 FK로 지정

사용자의 예약 정보를 저장하는 테이블



```
[root@DB1 ~]# mysqldump -u root -p -B DB1 > dump.sql
```

Enter password: 데이터베이스 DB1을 dump.sql 이라는 이름의 파일로 백업 진행

```
[root@DB1 ~]# ll
```

```
total 2452
```

```
-rw-----, 1 root root    1466 Mar 29 22:49 anaconda-ks.cfg
```

```
drwxr-xr-x  2 root root         6 Mar 30 00:08 daisy
```

```
-rw-r--r--  1 root root   5052 Mar 31 19:26 dump.sql DB1 서버에 생성된 데이터베이스 백업 파일
```

```
-rw-r--r--  1 root root 1245701 Mar 31 02:24 repluser@10.5.104.106
```

```
-rw-r--r--  1 root root 1245701 Mar 31 02:37 root@10.5.104.106
```

```
[root@DB1 ~]# scp dump.sql root@10.5.104.106:/root/ 데이터베이스 백업파일을 slave서버로 전송
```

```
dump.sql                                100% 5052    3.4MB/s   00:00
```

```
mysql> create user 'repluser'@'%' identified by 'TEst!234';
```

```
Query OK, 0 rows affected (0.00 sec) Replication 계정 생성 및 권한 부여
```

```
mysql> grant replication slave on *.* to 'repluser'@'%;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> flush privileges ;
```

```
Query OK, 0 rows affected (0.00 sec)
```

Master 서버(DB1) 설정



```
[root@DB2 ~]# ll
```

```
total 1224
```

```
-rw-----. 1 root root 1466 Mar 30 11:13 anaconda-ks.cfg
```

```
-rw-r--r-- 1 root root 1245701 Mar 31 10:19 dump.sql
```

Slave 서버에 생성된 데이터베이스 DB1의 백업파일

```
[root@DB2 ~]# mysql -u root -p < dump.sql
```

```
Enter password:
```

DB1 복원

```
mysql> stop slave;
```

```
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

```
mysql> reset slave;
```

```
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

```
mysql> change master to
```

```
-> master_host='10.5.104.105',
```

```
-> master_user='repluser',
```

```
-> master_password='TEst!234',
```

```
-> master_log_file='mysql-bin.000001',
```

```
-> master_log_pos=462;
```

Master 서버 주소

생성한 replication 유저

패스워드

Master 서버 파일

Master 서버 포지션

```
Query OK, 0 rows affected, 8 warnings (0.01 sec)
```

```
mysql> start slave;
```

```
Query OK, 0 rows affected, 1 warning (0.02 sec)
```

Slave 서버(DB2) 설정



10.5.104.105

```

Connection id: 20
Current database: DB1

+-----+
| Tables_in_DB1 |
+-----+
| deletetbl      |
| reservation    |
| tb_book        |
| tb_flight      |
| tb_user        |
| user           |
+-----+
6 rows in set (3.48 sec)

```

10.5.104.106

```

Connection id: 14
Current database: DB1

+-----+
| Tables_in_DB1 |
+-----+
| deletetbl      |
| reservation    |
| tb_book        |
| tb_flight      |
| tb_user        |
| user           |
+-----+
6 rows in set (0.00 sec)

```

10.5.104.105

```

mysql> select * from tb_user;
+-----+-----+-----+-----+-----+
| useremail      | userpw      | username | usertel      | salt |
+-----+-----+-----+-----+-----+
| chulsoo@gmail.com | chulsoo123 | 김철수   | 01012341234 | 0     |
| ourclub7279@gmail.com | h470200qq | TonyHan  | 010-4258-9667 | 0     |
+-----+-----+-----+-----+-----+

```

10.5.104.106

```

mysql> select * from tb_user;
+-----+-----+-----+-----+-----+
| useremail      | userpw      | username | usertel      | salt |
+-----+-----+-----+-----+-----+
| chulsoo@gmail.com | chulsoo123 | 김철수   | 01012341234 | 0     |
| ourclub7279@gmail.com | h470200qq | TonyHan  | 010-4258-9667 | 0     |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

Master 서버와 Slave 서버간의 데이터 동기화



www.triply.co.kr의 메시지
가입성공!

확인

triply

회원가입

Facebook Twitter Google Plus

✉ chulsoo@gmail.com

T

@ 김철수

☎ 01012341234

회원가입하기

MySQL Workbench

daisydb x

File Edit View Query Database Server Tools Scripting Help

Navigator: tb_user x tb_flight tb_book

SCHEMAS

Filter on

DB1

1 • SELECT * FROM DB1.tb_user;

Result Grid

useremail	userpw	username	usertel	salt
chulsoo@gmail.com	chulsoo...	김철수	01012341234	0
ourclub7279@gmail...	h4702...	TonyHan	010-4258-9667	0

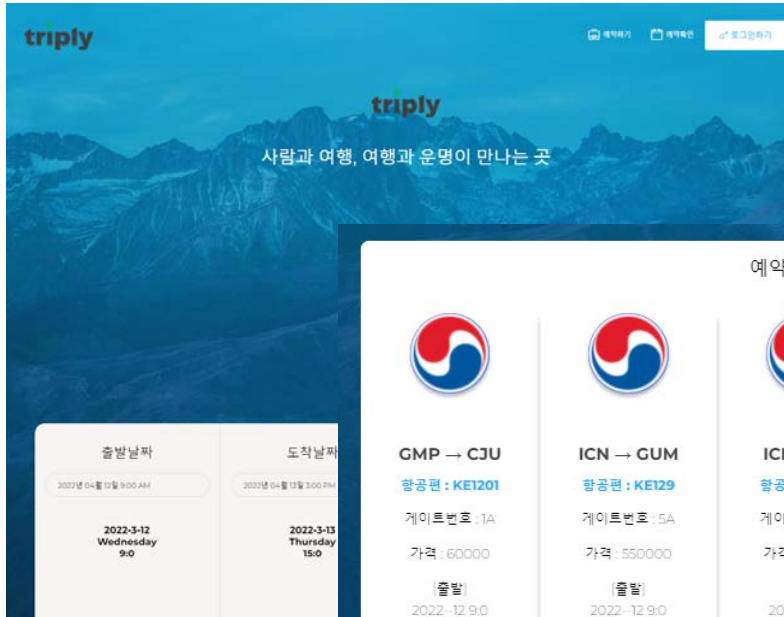
tb_user 1 x






Output

Action Output

사용자의 회원가입 정보인 이메일, 비밀번호, 이름, 전화번호가
유저 테이블인 tb_user에 저장되며
웹페이지에는 가입성공을 출력한다

사용자의 정보 DB 저장



예약가능정보				
				
GMP → CJU	ICN → GUM	ICN → KIX	GMP → CJU	ICN → KIX
항공편 : KE1201	항공편 : KE129	항공편 : KE184	항공편 : KE192	항공편 : KE723
게이트번호 : 1A	게이트번호 : 5A	게이트번호 : 6A	게이트번호 : 8A	게이트번호 : 4A
가격 : 60000	가격 : 550000	가격 : 360000	가격 : 50000	가격 : 340000
[출발] 2022-12 9:0 wednesday	[출발] 2022-12 9:0 wednesday	[출발] 2022-12 9:0 wednesday	[출발] 2022-12 9:0 wednesday	[출발] 2022-12 9:0 wednesday
[도착] 2022-13 15:0 thursday	[도착] 2022-13 15:0 thursday	[도착] 2022-13 15:0 thursday	[도착] 2022-13 15:0 thursday	[도착] 2022-13 15:0 thursday
예약하기	예약하기	예약하기	예약하기	예약하기

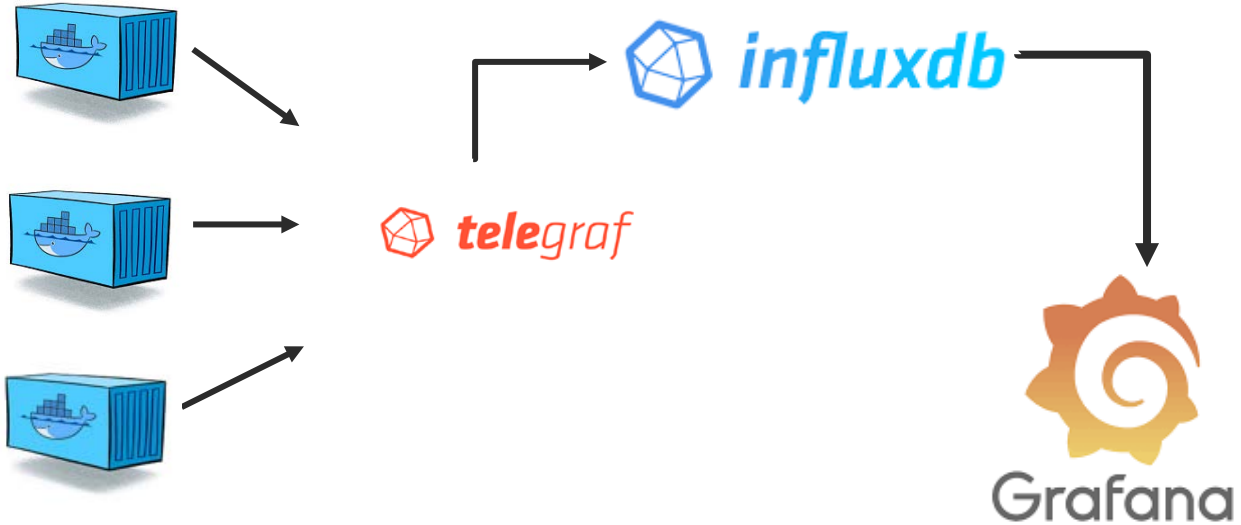
DB 정보 불러오기



05 서버 모니터링

Monitoring





- Telegraf : 시스템에서 원하는 데이터를 측정하고, 이를 데이터베이스(InfluxDB)로 보내주는 역할을 수행한다.
- InfluxDB : 시계열 데이터 특화 데이터베이스이다. 종류(온도-실수, 팬속도-정수 등)를 시간에 따라 저장한다.
- Grafana : 시계열 매트릭 데이터를 시각화 하는데 가장 최적화된 대시보드를 제공하는 오픈소스 툴킷

Telegraf + InfluxDB + Grafana 를 이용한 모니터링



```
[global_tags]
environment="swarm"

# Read metrics about CPU usage
[[inputs.cpu]]
  percpu = false
  totalcpu = true
  fieldpass = [ "usage*" ]
  name_suffix = "_vm"

# Read metrics about disk usage
[[inputs.disk]]
  fielddrop = [ "inodes*" ]
  mount_points=["/"]
  name_suffix = "_vm"

# Read metrics about network usage
[[inputs.net]]
  interfaces = [ "eth0" ]
  fielddrop = [ "icmp*", "ip*", "tcp*", "udp*" ]
  name_suffix = "_vm"

# Read metrics about memory usage
[[inputs.mem]]
  name_suffix = "_vm"

# Read metrics about swap memory usage
[[inputs.swap]]
  name_suffix = "_vm"

# Read metrics about system load & uptime
[[inputs.system]]
  name_suffix = "_vm"

# Read metrics from docker socket api
[[inputs.docker]]
  endpoint = "unix:///var/run/docker.sock"
  container_names = []
  timeout = "5s"
  perdevice = true
  total = false
  docker_label_include = []
  docker_label_exclude = []
  name_suffix = "_docker"

[[outputs.influxdb]]
  database = "vm_metrics"
  urls = ["http://influxdb:8086"]
  namepass = ["*_vm"]

[[outputs.influxdb]]
  database = "docker_metrics"
  urls = ["http://influxdb:8086"]
  namepass = ["*_docker"]
```

```
version: "3.3"

services:
  telegraf:
    image: telegraf:1.3
    networks:
      - tig-net
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    configs:
      - source: telegraf-config
        target: /etc/telegraf/telegraf.conf
    deploy:
      restart_policy:
        condition: on-failure
      mode: global

  influxdb:
    image: influxdb:1.2
    networks:
      - tig-net
    deploy:
      restart_policy:
        condition: on-failure
      placement:
        constraints:
          - node.role == worker

  grafana:
    container_name: grafana
    image: grafana/grafana:4.3.2
    ports:
      - "3000:3000"
    networks:
      - tig-net
    deploy:
      restart_policy:
        condition: on-failure
      placement:
        constraints:
          - node.role == manager

  configs:
    telegraf-config:
      file: $PWD/conf/telegraf/telegraf.conf

  networks:
    tig-net:
      driver: overlay
```



Telegraf conf 파일의 내용

Yaml 파일 내용

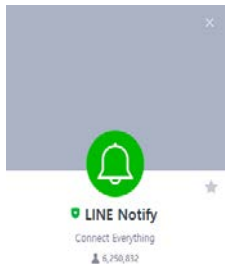
Telegraf + InfluxDB + Grafana 를 이용한 모니터링



Grafana 를 이용한 모니터링

LINE Notify 를 통해 Token 발행

입력 후 알림 범위 설정



Edit Channel

Name	Daisy
Type	LINE
Send on all alerts	<input checked="" type="checkbox"/>
Include image	<input checked="" type="checkbox"/>

LINE notify settings

Token: d1kTrI.FayASG87psk3zhvOQH05KekCEOMb

Save **Send Test**

Alerting

Alert Config

Name: worker2 alert Evaluate every: 60s

Conditions

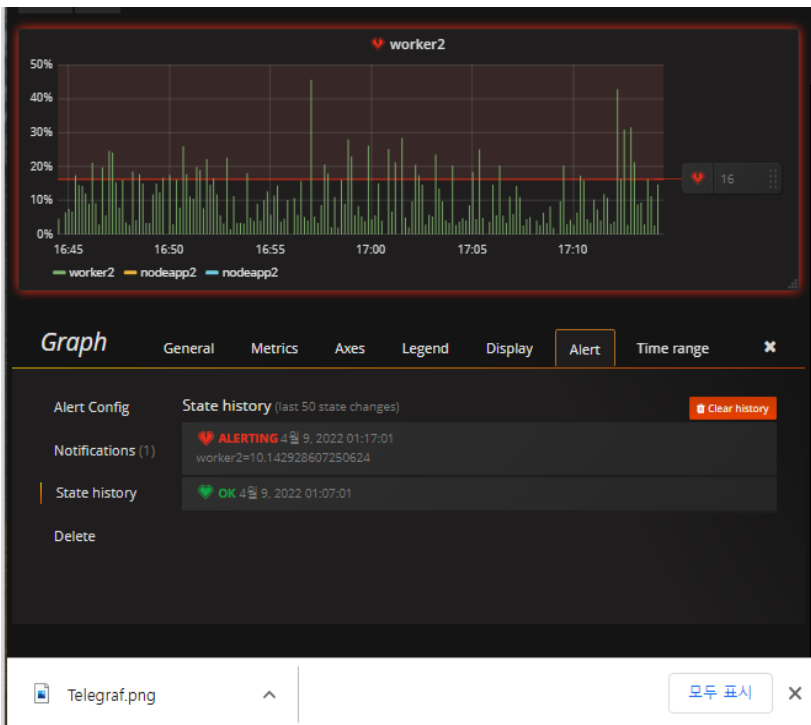
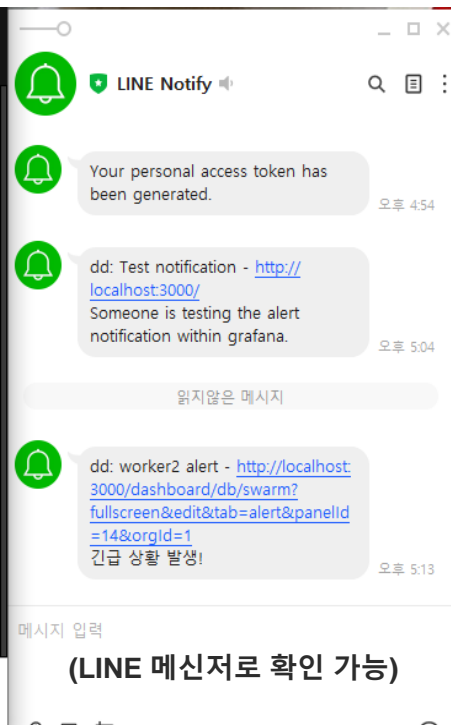
When: query (A, 5m, now) IS ABOVE 40

Text Rule

If no data or all values are null: SET STATUS TO No Data

If execution error or timeout: SET STATUS TO Alerting

Grafana 를 이용한 알림

The figure shows a chat interface for LINE Notify. The messages are as follows:

- dd: Test notification - <http://localhost:3000/> Someone is testing the alert notification within grafana. (오후 5:04)
- dd: worker2 alert - <http://localhost:3000/dashboard/db/swarm?fullscreen&edit&tab=alert&panelId=14&orgId=1> 긴급 상황 발생! (오후 5:13)

At the bottom, there is a text input field labeled '메시지 입력' and a button labeled '(LINE 메신저로 확인 가능)'.

Grafana 를 이용한 알림



filter containers

manager1
manager
3.810G RAM
x86_64/linux

manager2
manager
3.810G RAM
x86_64/linux

paradise_telegraf

image: telegraf:1.3@sha256:3fedf43e36
tag: 1.3@sha256:3fedf43e36e486a245
updated: 8/4 14:59
9b2f800f00599c3c7844650e83f635a0b8
state: running

visualizer_visual

image: visualizer:latest@sha256:530b863
tag: latest@sha256:530b86372e783007f
updated: 8/4 14:14
ba7802ae9879a0ad74087b673ba464a89
state: running

daisy_proxy

image: daisy:react.2.0@sha256:782190a
tag: 2.0@sha256:782190a00d93bf7733b
updated: 8/4 14:35
f5d34f8e662507394e150cc98239060097
state: running

paradise_telegraf

image: telegraf:1.3@sha256:3fedf43e36
tag: 1.3@sha256:3fedf43e36e486a245
updated: 8/4 14:59
c0b10e557d5446684ad7e3c858a9e444f
state: running

paradise_grafana

image: grafana:4.3.2@sha256:3cde55564
tag: 4.3.2@sha256:3cde5556a20f19e4d1f
updated: 8/4 14:59
2ad43636374c853c25eebf39ec04beebcdaf
state: running

worker1
worker
1.907G RAM
x86_64/linux

worker2
worker
1.907G RAM
x86_64/linux

paradise_telegraf

image: telegraf:1.3@sha256:3fedf43e36
tag: 1.3@sha256:3fedf43e36e486a245
updated: 8/4 14:59
1899f3a89e5daf192e02de562a6de6e007
state: running

paradise_influxdb

image: influxdb:1.2@sha256:20459db782
tag: 1.2@sha256:20459db7820dbcc435d
updated: 8/4 14:59
39bb04bf1f02ac7032a07cac0a9f52efbd
state: running

daisy_nodeapp

image: daisy:express.2.0@sha256:5487a
tag: 2.0@sha256:5487ac610546fa4ebcd
updated: 8/4 14:35
99a82e0e9ef91dd938c155ad33b635ec15
state: running

daisy_nodeapp

image: daisy:express.2.0@sha256:5487a
tag: 2.0@sha256:5487ac610546fa4ebcd
updated: 8/4 14:35
5a74f0d907b076b312324e5f03f3d911c
state: running

daisy_nodeapp

image: daisy:express.2.0@sha256:5487a
tag: 2.0@sha256:5487ac610546fa4ebcd
updated: 8/4 14:35
305f278aab84e66ecc0f0f0153cda2e764c
state: running

paradise_telegraf

image: telegraf:1.3@sha256:3fedf43e36
tag: 1.3@sha256:3fedf43e36e486a245
updated: 8/4 14:59
7ecd9c7f982eead8675b7525be49b06c29
state: running

daisy_nodeapp

image: daisy:express.2.0@sha256:5487a
tag: 2.0@sha256:5487ac610546fa4ebcd
updated: 8/4 14:35
9f98893dfb61a38566c807938c3707f9ba
state: running

triply