

# WK1 DevOps In-Class Lab - Set up Nginx Load Balancer using docker

Reference:

<https://levelup.gitconnected.com/nginx-load-balancing-and-using-with-docker-7e16c49f5d9>

## Prerequisite:

1. Installed Docker
2. Installed k6 <https://k6.io/docs/getting-started/installation>

## Step 1- Clone the repo:

```
ywang@C02TM07WH03Y ~/DevOps  
$ git clone https://github.com/mrceylan/DockerNginxLoadBalancer.git
```

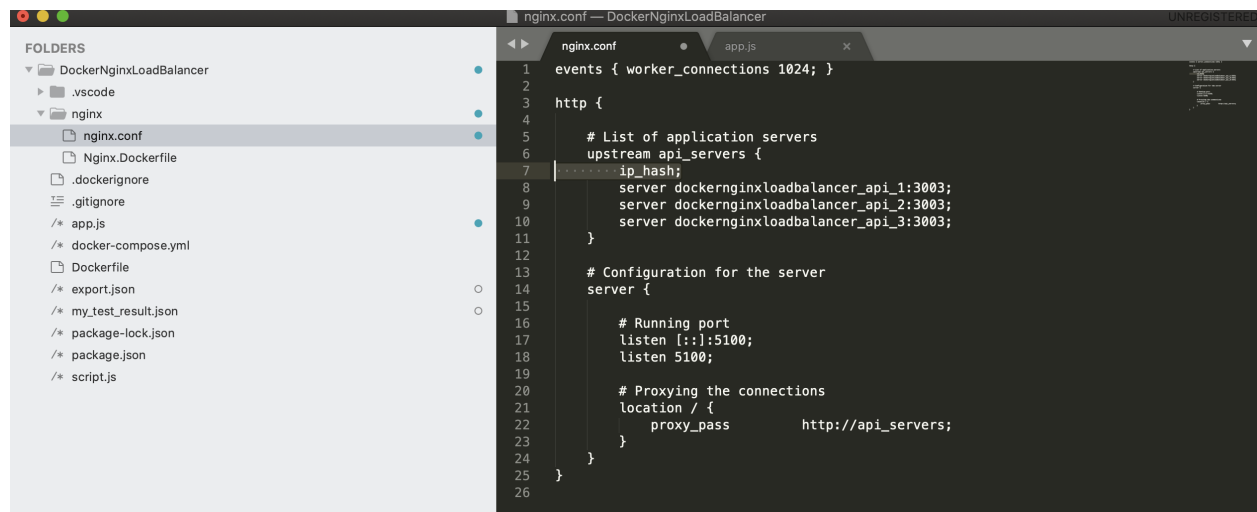
git clone <https://github.com/mrceylan/DockerNginxLoadBalancer.git>

## Step 2 - Get into the folder/directory:

```
ywang@C02TM07WH03Y ~/DevOps  
$ cd DockerNginxLoadBalancer/
```

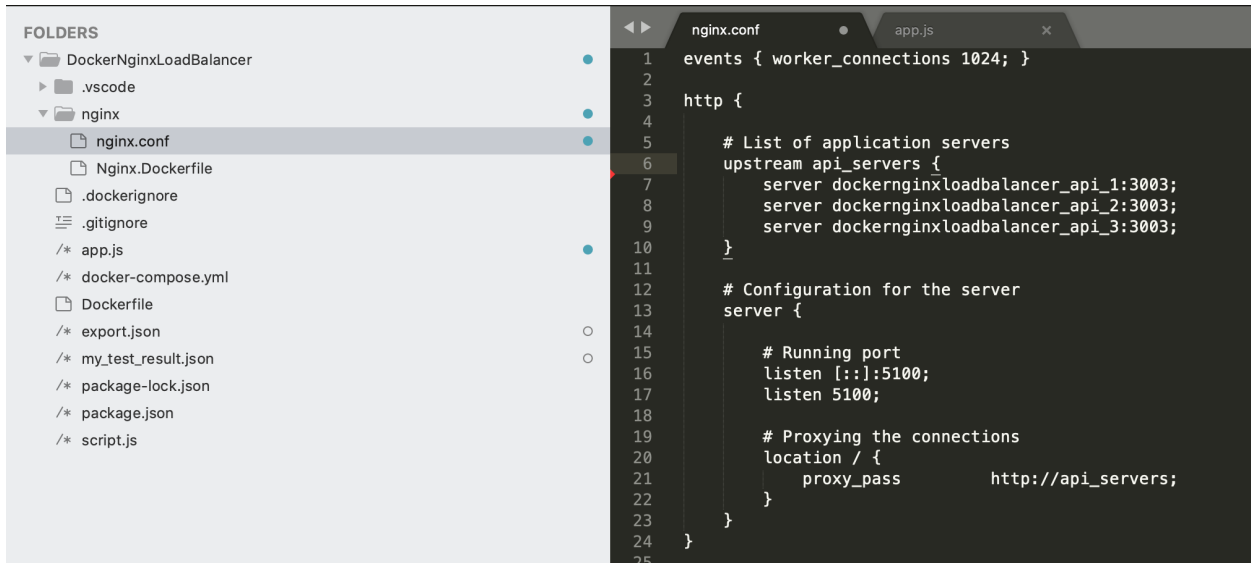
## Step 3 - Remove line 7 ip\_hash from nginx/nginx.conf file

Before:



```
1 events { worker_connections 1024; }  
2  
3 http {  
4  
5     # List of application servers  
6     upstream api_servers {  
7         ip_hash;  
8         server dockernginxloadbalancer_api_1:3003;  
9         server dockernginxloadbalancer_api_2:3003;  
10        server dockernginxloadbalancer_api_3:3003;  
11    }  
12  
13    # Configuration for the server  
14    server {  
15  
16        # Running port  
17        listen [::]:5100;  
18        listen 5100;  
19  
20        # Proxying the connections  
21        location / {  
22            proxy_pass      http://api_servers;  
23        }  
24    }  
25 }  
26
```

After:

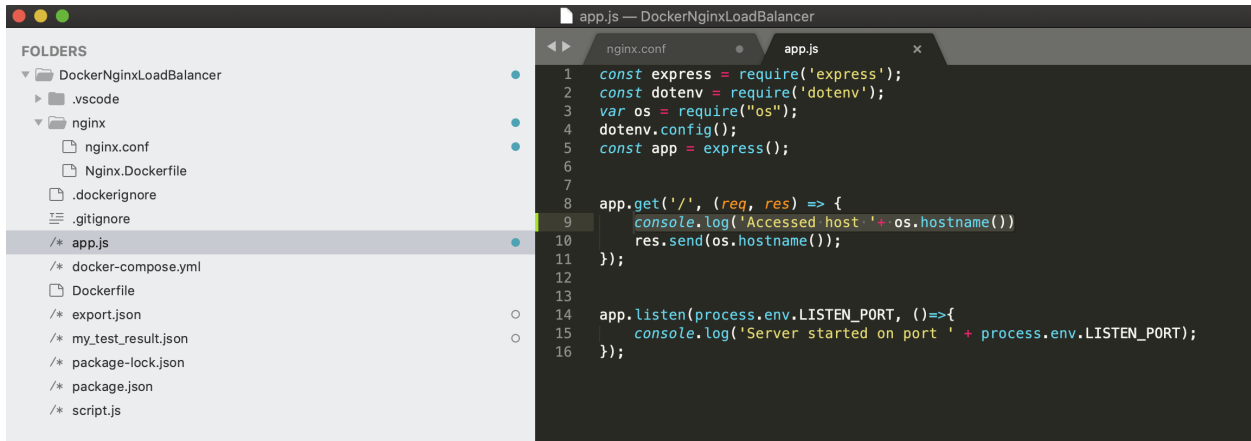


The screenshot shows the VS Code interface. On the left, the 'FOLDERS' sidebar displays the project structure: DockerNginxLoadBalancer (containing .vscode and nginx) and nginx (containing nginx.conf, Nginx.Dockerfile, .dockerignore, .gitignore, app.js, docker-compose.yml, Dockerfile, export.json, my\_test\_result.json, package-lock.json, package.json, and script.js). The main editor shows the nginx.conf file with the following content:

```
1 events { worker_connections 1024; }
2
3 http {
4
5     # List of application servers
6     upstream api_servers {
7         server dockernginxloadbalancer_api_1:3003;
8         server dockernginxloadbalancer_api_2:3003;
9         server dockernginxloadbalancer_api_3:3003;
10    }
11
12    # Configuration for the server
13    server {
14
15        # Running port
16        listen [::]:5100;
17        listen 5100;
18
19        # Proxying the connections
20        location / {
21            proxy_pass http://api_servers;
22        }
23    }
24
25 }
```

Step 4 - Add the following line into app.js to print out logs:

`console.log('Accessed host '+ os.hostname())`

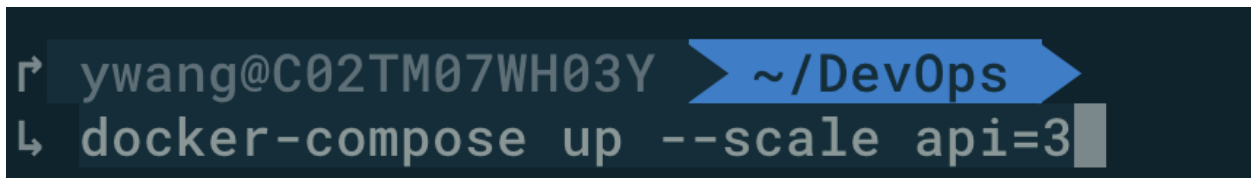


The screenshot shows the VS Code interface. On the left, the 'FOLDERS' sidebar displays the project structure: DockerNginxLoadBalancer (containing .vscode and nginx) and nginx (containing nginx.conf, Nginx.Dockerfile, .dockerignore, .gitignore, app.js, docker-compose.yml, Dockerfile, export.json, my\_test\_result.json, package-lock.json, package.json, and script.js). The main editor shows the app.js file with the following content:

```
1 const express = require('express');
2 const dotenv = require('dotenv');
3 var os = require("os");
4 dotenv.config();
5 const app = express();
6
7
8 app.get('/', (req, res) => {
9     console.log('Accessed host ' + os.hostname());
10    res.send(os.hostname());
11 });
12
13
14 app.listen(process.env.LISTEN_PORT, ()=>{
15     console.log('Server started on port ' + process.env.LISTEN_PORT);
16 });
```

Step 5 - Start our program with 3 API servers:

`docker-compose up --scale api=3`



The screenshot shows a terminal window with the following content:

```
rwang@C02TM07WH03Y ~/Dev0ps
└─ docker-compose up --scale api=3
```

You should see the following once it is built

```
Starting dockernginxloadbalancer_api_1 ... done
Starting dockernginxloadbalancer_api_2 ... done
Starting dockernginxloadbalancer_api_3 ... done
Starting dockernginxloadbalancer_nginx_1 ... done
Attaching to dockernginxloadbalancer_api_1, dockernginxloadbalancer_api_2, dockernginxloadbalancer_api_3, dockernginxloadbalancer_nginx_1
api_1 |
api_1 | > DockerNginxLoadBalancer@1.0.0 start /usr/src/app
api_1 | > node app.js
api_1 |
api_2 |
api_2 | > DockerNginxLoadBalancer@1.0.0 start /usr/src/app
api_2 | > node app.js
api_2 |
api_3 |
api_3 | > DockerNginxLoadBalancer@1.0.0 start /usr/src/app
api_3 | > node app.js
api_3 |
nginx_1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
nginx_1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx_1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
nginx_1 | 10-listen-on-ipv6-by-default.sh: info: IPv6 listen already enabled
nginx_1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
nginx_1 | /docker-entrypoint.sh: Configuration complete; ready for start up
api_2 | Server started on port 3003
api_3 | Server started on port 3003
api_3 | Server started on port 3003
```

## Step 6 - hit localhost:5100 in your browser and see what happens

## Step 7 - run automated k6 tests:

```
k6 run -u 200 -d 30s --summary-export=export.json --out json=my_test_result.json script.js
```

```

      /\   /\   /\   /\   /\
     /\  /\  /\  /\  /\  /\
    /\ /\ /\ /\ /\ /\ /\ /\
   /\ /\ /\ /\ /\ /\ /\ /\
  /\ /\ /\ /\ /\ /\ /\ /\
 /_/\_/\_/\_/\_/\_/\_/\_/.io

execution: local
script: script.js
output: json=my_test_result.json

scenarios: (100.00%) 1 scenario, 200 max VUs, 1m0s max duration (incl. graceful stop):
 * default: 200 looping VUs for 30s (gracefulStop: 30s)


running (0m30.5s), 000/200 VUs, 10539 complete and 0 interrupted iterations
default ✓ [=====] 200 VUs  30s

✓ is status 200

checks.....: 100.00% ✓ 10539 x 0
data_received.....: 2.5 MB 82 kB/s
data_sent.....: 843 kB 28 kB/s
http_req_blocked.....: avg=286.9µs min=1µs med=5µs max=166.39ms p(90)=9µs p(95)=13µs
http_req_connecting.....: avg=248.9µs min=0s med=0s max=166.28ms p(90)=0s p(95)=0s
http_req_duration.....: avg=570.86ms min=29.45ms med=536.13ms max=1.25s p(90)=944.54ms p(95)=1.01s
http_req_receiving.....: avg=91.11µs min=20µs med=80µs max=11.83ms p(90)=121µs p(95)=142µs
http_req_sending.....: avg=77.24µs min=8µs med=24µs max=169.67ms p(90)=39µs p(95)=51µs
http_req_tls_handshaking...: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
http_req_waiting.....: avg=570.69ms min=29.35ms med=536.02ms max=1.24s p(90)=944.4ms p(95)=1.01s
http_reqs.....: 10539 345.497711/s
iteration_duration.....: avg=571.35ms min=29.69ms med=536.3ms max=1.36s p(90)=944.87ms p(95)=1.01s
iterations.....: 10539 345.497711/s
vus.....: 200 min=200 max=200
vus_max.....: 200 min=200 max=200

```

You should see something like this. What is the total number of requests? And how many requests did we make per second?

You just successfully triggered the load balancer round-robin algorithm.

How many requests would you expect to see on each API host?

Could you try to use the least\_conn algorithm and tell us the difference?

What is the use-case of different algorithms in Load balancer?

## Tips:

once you modified any file under a service, you may want to rebuild that service so that it picks up the code that you modified:

```
ywang@C02TM07WH03Y ~/DevOps/DockerNginxLoadBalancer master ● ?  
└─ docker-compose build api
```

```
ywang@C02TM07WH03Y ~/DevOps/DockerNginxLoadBalancer master ● ?  
└─ docker-compose build nginx
```

docker-compose build nginx

docker-compose build api