Load balancing

What is load balancing?

Modern high traffic websites must serve hundreds of thousands of concurrent requests from users or clients and return the correct text, images, or application data, all in a fast and reliable manner. To cost effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers.

Load balancing is a technique used in computing and networking to distribute incoming traffic or workload across multiple resources, such as servers, CPUs, or network links. The main goal of load balancing is to optimize resource utilization, maximize throughput, minimize response time, and avoid overload on any single resource.

In the context of web servers or application servers, load balancing involves distributing incoming client requests across a cluster of servers. This ensures that no single server becomes overwhelmed with requests while others remain idle, thus improving the overall performance, reliability, and availability of the system.

A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

Some examples of algorithms for selecting resources while load balancing:

- Round Robin: Requests are distributed evenly among the available servers in a circular order.
- Least Connections: New requests are forwarded to the server with the fewest active connections.
- Weighted Round Robin: Servers are assigned weights, and requests are distributed according to these weights.
- Least Response Time: Requests are forwarded to the server with the fastest response time.
- IP Hash: The client's IP address is used to determine which server receives the request, ensuring that requests from the same client are always directed to the same server.

How does a load balancer work?

While configuring a load balancer you must define an IP, or a DNS server for the load balancer. Next step is adding a load balancing pool, meaning you add the list of actual servers that the load balancer will serve. This list of addresses must only be accessible internally via the load balancer to ensure a proper workflow (no external requests are served by the server that can overwhelm it). Next step is deploying the load balancer, either as a proxy between the app servers and the clients, or as a gateway which assigns a client to

a server once and then leaves the user to interact directly with the server. Final step is the redirection of requests, which takes place according to the administrator's chosen algorithm.

Load balancing with NGINX

To set up a simple NGINX load balancer we will first have to create a file named ***nginx.conf***. This will act as the configuration file for our load balancer:

```
upstream backend {
    server service1:3000;
    server service2:3001;
    server service3:3002;
}

server {
    listen 80;
    location / {
        proxy_pass http://backend;
    }
}
```

For the purposes of our tutorial, this will be the nginx.conf file. The upstream section defines the servers that the load balancer will act as a proxy for. In our case service1, service2 and service3 (docker hosts) are the hosts of the services that the balancer will act as a proxy for. The server section defines the port the load balancer will listen for and the location where the load balancing will take place. The proxy_pass section defines the reverse proxy for our setup.

To deploy the load balancer as a docker container, a *Dockerfile* must be created

```
FROM nginx:stable-alpine

COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

This file acts as a script telling docker how to set up our load balancer, setting up the config in the container, exposing the port 80 and running the load balancer.

For our services, we will setup some simple Express NodeJS services. This is our index.js setup for service1

```
const express = require('express'),
    app = express();

app.use(express.urlencoded({ extended: true }))
app.use(express.json())

const port = 3000
app.get('/',
    (req, res) => res.send('Response sent from Service 1 running on port ' + port))
app.listen(port,
    () => console.log(`⚡ [bootup]: Server is running at port: `+ port));
```

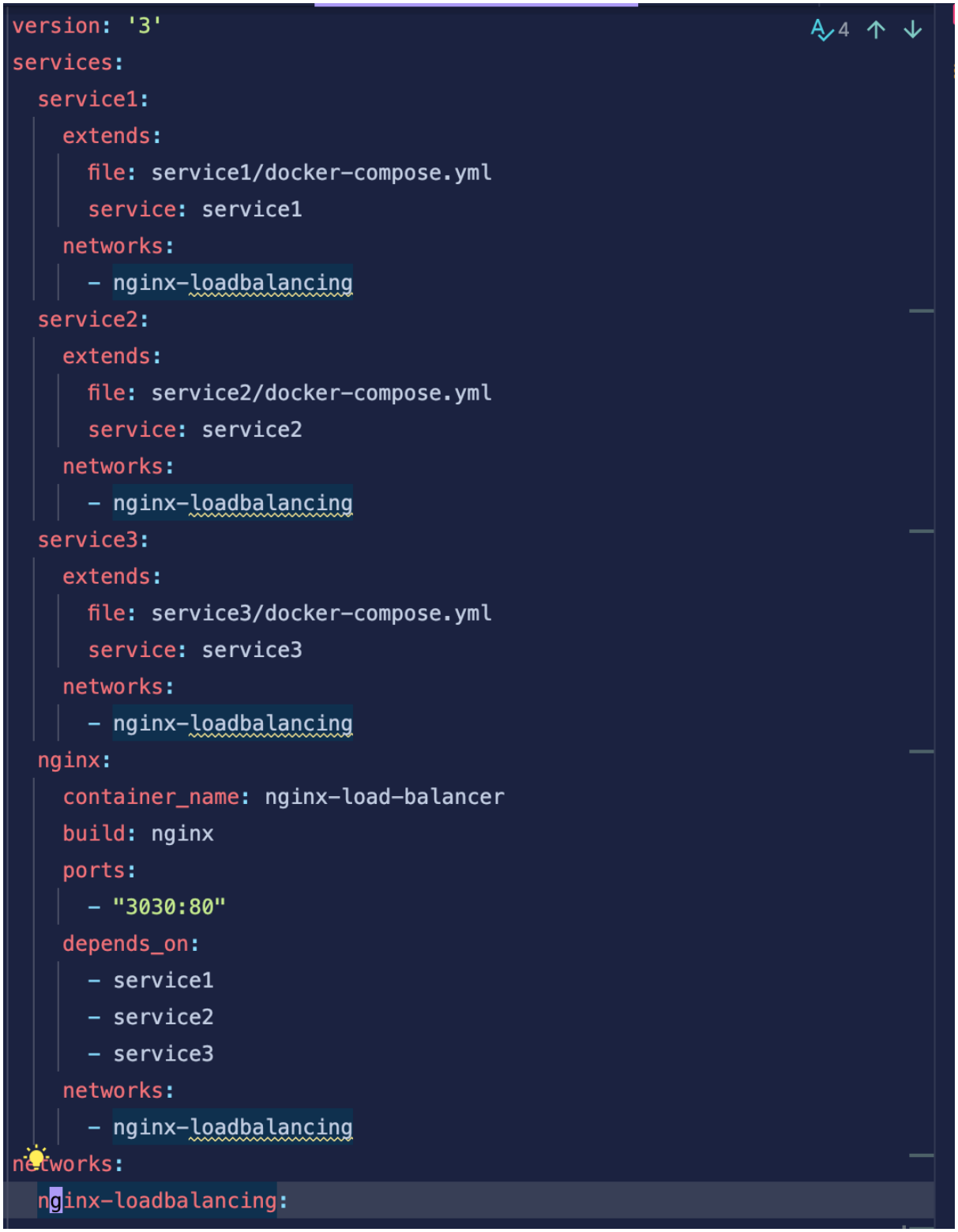
The setup is the same for the remaining 2 services, just choose another port and change the returned message if desired.

Each service has defined a Dockerfile and a docker-compose.yml as such

```
FROM node:slim

ENV NODE_ENV development

WORKDIR /express-docker

COPY . .

RUN npm install

CMD [ "node", "index.js" ]

EXPOSE 3001
```

```
version: '3'
services:
  service1:
    image: service1
    build: .
    ports:
      - "3000:3000"
```

These files will deploy our services as docker containers.
To deploy all our services and the load balancer, we must create a general docker-compose.yml file in the root folder of the project.



```yaml
version: '3'
services:
  service1:
    extends:
      file: service1/docker-compose.yml
      service: service1
    networks:
      - nginx-loadbalancing
  service2:
    extends:
      file: service2/docker-compose.yml
      service: service2
    networks:
      - nginx-loadbalancing
  service3:
    extends:
      file: service3/docker-compose.yml
      service: service3
    networks:
      - nginx-loadbalancing
  nginx:
    container_name: nginx-load-balancer
    build: nginx
    ports:
      - "3030:80"
    depends_on:
      - service1
      - service2
      - service3
    networks:
      - nginx-loadbalancing
networks:
  nginx-loadbalancing:
```

This file deploys all our necessary resources.

To test our basic setup, we must execute docker-compose up -d in the terminal, which will create and deploy to docker our services and load balancer then execute calls to http://localhost:3030/ which is the address of the load balancer. For each response the payload will be different, either from S1, S2, S3 in a sequential order as NGINX uses a Round robin strategy for balancing by default.