Tutorial: Setting Up Nginx as a Load Balancer with Docker Compose

Introduction

Load balancing is a crucial aspect of modern web applications, distributing incoming traffic across multiple servers to ensure high availability, scalability, and reliability. In this tutorial, we'll explore how to set up Nginx as a load balancer in the context of our system.

We'll enhance our load balancing setup by integrating it into a Docker Compose configuration. Docker Compose simplifies the deployment and management of multi-container applications.

Objective

The goal of this tutorial is to configure Nginx to distribute incoming requests among multiple instances of our web server and microservices, achieving a scalable and fault-tolerant architecture.

Benefits

- High Performance: Nginx is designed to handle a large number of concurrent connections efficiently. Its asynchronous, event-driven architecture allows it to scale and handle high traffic loads with low resource consumption.
- Load Balancing: Nginx acts as a reliable load balancer, distributing incoming network traffic across multiple servers. This helps optimize resource utilization and ensures high availability and fault tolerance.
- Reverse Proxy: Nginx serves as a reverse proxy, sitting between clients and web servers. It can handle tasks such as SSL termination, caching, and compressing, offloading these responsibilities from the application servers and enhancing overall performance.
- SSL/TLS Termination: Nginx can terminate SSL/TLS connections, managing encryption and decryption processes. This offloads the cryptographic workload from application servers, improving overall system efficiency.
- Caching: Nginx includes built-in caching capabilities, reducing the load on backend servers by serving static content directly from memory. This leads to faster response times and improved website performance.

- Security: Nginx provides various security features, including access control, rate limiting, and the ability to mitigate common web vulnerabilities. Its modular architecture allows for easy integration with security modules and third-party tools.

Configure Nginx as a Load Balancer

Add a Nginx configuration file to your project, that can be named application.conf containing the following lines:

```
server{
    listen <port>;

    location / {
    proxy_pass http://<server>;
    }

}
```

- Replace the <port> with the port of your server. (For SpringBoot apps it is the server.port present in the application.properties)

```
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto=update
server.port=80
```

- Replace <server> with the server service label present in the docker-compose.yml file:

```
spring-back-end
  build:
    context: ./backend
    dockerfile: Dockerfile
  environment:
    VIRTUAL_PORT: 80
    VIRTUAL_HOST: springbackend.local
  depends_on:
    - mysql-container
    - rabbitmq-container
```

- The location / block inside application.conf specifies that any request received by Nginx should be forwarded to the server service. This ensures that the load balancer correctly routes traffic to the server services.

Docker Compose YML configuration for Nginx

As a configuration for the Nginx service in the docker-compose.yml file you can use the following:

```yaml
loadbalancer:
  image: nginx:latest
  container_name: 'loadbalancer'
  volumes:
    - "./confd:/etc/nginx/conf.d"
  ports:
    - 80:80
  environment:
    DEFAULT_HOST: springbackend.local
  depends_on:
    - spring-back-end
```

image: nginx:latest:

- Specifies the Docker image to be used for the loadbalancer service(nginx).
- Specifies the version of the Nginx image available on Docker Hub (latest).

container_name: 'loadbalancer':

- Assigns a custom name to the Docker container for easy identification.

volumes:

- Defines volumes to be mounted in the container. Volumes are used for persistent data or configuration.
    - "./confd:/etc/nginx/conf.d" mounts the local confd directory to the /etc/nginx/conf.d directory inside the container.

ports:

- Specifies the port mappings between the host and the container.
    - 80:80 maps port 80 on the host to port 80 on the container.

environment:

- Sets environment variables for the container.

- DEFAULT_HOST: springbackend.local is setting the DEFAULT_HOST environment variable to 'springbackend.local.'

depends_on:

- It ensures that other services listed are started before this service.

Docker command deploy server

Using the --scale option in Docker Compose to replicate a service, is a common practice for achieving load balancing and increased service availability. It's a straightforward way to create multiple instances of the same service.

```
docker compose -f docker-compose.yml up -d --scale spring-back-end=2
```

Once you have successfully deployed multiple instances of the server, you will observe that successive HTTP requests alternate between servers, with the first request being handled by one server, the second by another, and so forth.