Corso di Sistemi Distribuiti e Cloud Computing

Corso di Laurea Magistrale in Ingegneria Informatica A.A. 2023/24

DIMES - Università degli Studi della Calabria

NGINX AS REVERSE PROXY

A SHORT GUIDE FOR DEPLOYING WEB APP

Ing. Loris Belcastro

Nginx

- **NGINX** is a high performance web server developed to facilitate the increasing needs of the modern web. It focuses on high performance, high concurrency, and low resource usage.
- Although it's mostly known as a web server, NGINX at its core is a reverse proxy server.
- NGINX is not the only web server on the market, though. One of its biggest competitors is **Apache HTTP Server** (httpd), first released back on 1995.
- Apache HTTP Server is more flexible, but server admins often prefer NGINX for two main reasons:
  - It can handle a higher number of concurrent requests.
  - It has faster static content delivery with low resource usage.

Nginx

- NGINX is widely used for serving web content. It is known for its high performance, stability, and efficiency, making it a popular choice for various web applications and websites.
- Some key features of NGINX are:
  - **Web Server:** It can serve static content (such as HTML, CSS and images) directly to clients, making it efficient for delivering web pages.
  - **Reverse Proxy:** It can act a reverse proxy by forwarding client's requests to other servers. It is a commonly used practice for security and scalability.
  - **Load Balancer:** It can route incoming traffic to different server instances ensuring optimal resource utilization.
  - **SSL/TLS Termination:** It can handle SSL/TLS termination, encrypting and decrypting data between clients and servers.
  - **Performance Optimization:** It is designed to be lightweight and efficient, capable of handling a large number of concurrent connections with low resource usage.

Nginx – Configuration files

- NGINX consists of modules which are controlled by **directives** specified in the configuration file.
- A **simple directive** consists of the name and parameters separated by spaces and ends with a semicolon (;).
- A **block directive** has the same structure as a simple directive, but instead of the semicolon, it ends with a set of additional instructions surrounded by braces ({ and }).
- If a block directive can have other directives inside braces, it is called a context (examples: *events*, *http*, *server*, and *location*).
- Directives placed in the configuration file outside of any contexts are considered to be in the **main context**.
  - The *events* and *http* directives reside in the *main* context
  - *server* in *http*, and *location* in server.
- The rest of a line after the # sign is considered a comment.

Nginx – Configuration files

- The **root** directive specifies the root directory for requests. When a client sends a request to the Nginx server, the server will look for files to serve within the directory specified by the "root" directive.

- For example, the code below tells Nginx that when a request comes in for "example.com:80", it should look for files to serve in the "/var/www/html" directory on the server.
- If someone requests "http://example.com/index.html", Nginx will look for the file "/var/www/html/index.html" and serve it if it exists. If the file doesn't exist, Nginx will return a 404 error.

```
server {
    listen 80;
    server_name example.com;
    root /var/www/html;
}
```

Nginx – Configuration files

- The **location** directive is used to define how Nginx should handle different URIs or URL patterns. It allows you to specify configuration directives that apply only to certain URIs or URL patterns. This is particularly useful for tasks like proxying requests to different backend servers, serving static files, or handling dynamic content like PHP scripts.
- In this example:
  - The first "location" block handles requests to the root URI ("/").
  - The second "location" block handles requests to URIs starting with "/images/".
  - The third "location" block uses a regular expression (denoted by the tilde "~") to match URIs ending with ".php".

```
server {
    listen 80;
    server_name example.com;

    location / {
        # Configuration directives for handling requests to the root URI
    }

    location /images/ {
        # Configuration directives for handling requests to URIs starting with /images/
    }

    Location ~ \.php$ {
        # Configuration directives for handling requests to URIs ending with .php
    }
}
```

Nginx - Redirects

- To install Nginx on your system, you typically need to follow these general steps. I'll provide instructions for Ubuntu and CentOS, two common Linux distributions:


```
sudo apt update && \\
sudo apt install nginx && \\
sudo systemctl start nginx && \\
sudo systemctl enable nginx && \\
sudo systemctl status nginx
```

```
sudo yum update && \\
sudo yum install nginx && \\
sudo systemctl start nginx && \\
sudo systemctl enable nginx && \\
sudo systemctl status nginx
```

Nginx – Directory Structure

- Nginx keeps it's configuration in the expected /etc/nginx directory. This directory is broken up as follows:

Path	Purpose
./conf.d/*.conf	Extra configuration files.
./fastcgi.conf	Commonly configured directives (nginx packaging team)
./fastcgi_params	Commonly configured directives (upstream version)
./koi-utf	Nginx Character Set
./koi-win	Nginx Character Set
./mime.types	Maps file name extensions to MIME types of responses
./nginx.conf	The primary configuration file.
./proxy_params	Commonly configured directives
./scgi_params	Commonly configured directives
./sites-available/*	Extra virtual host configuration files
./sites-enabled/*	Symlink to sites-available/<file> to enable vhost
./snippets/*.conf	Configuration snippets that can be included in configs
./apps.d/*.conf	Files included by /etc/nginx/sites-available/default
./uwsgi_params	Commonly configured directives
./win-utf	Nginx Character Set

Nginx - Redirects

- It is quite easy to set up redirections using NGINX. Let's update the configuration to redirect all incoming traffic from /birds to /animals.
  - HTTP 307 => Temporary Redirect
  - HTTP 301 => Moved (Permanently)

```
events {}
http {
    include /opt/homebrew/etc/nginx/mime.types;

    server {
        listen  8080;
        root    /var/www/tutorials/nginx-test/html;
        location /animals {
            root /var/www/tutorials/nginx-test/html;
        }
        location /birds {
            return 307 /animals;
        }
    }
}
```

Nginx – Reverse Proxy

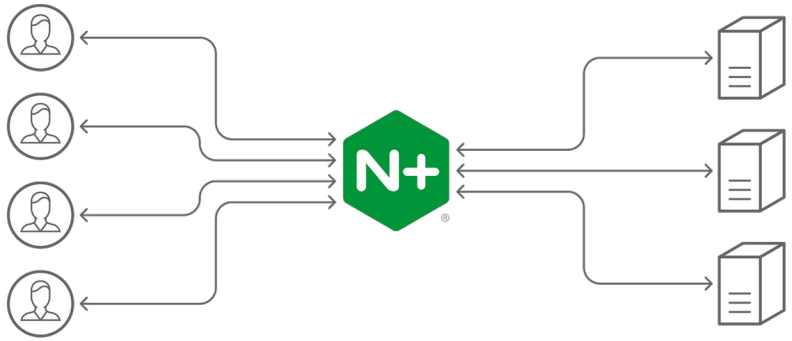
- To forward requests to another server we can use the **proxy_pass** directive.
- As shown in the example below, all the requests to localhost:8080 will be forwarded to http://192.168.10.5:8080.
- This is helpful when you don't want to reveal the actual server or want to relay requests to different servers based on different paths.

```
events {}

http {
    include /opt/homebrew/etc/nginx/mime.types;

    server {
        listen 8080;

        location / {
            proxy_pass http://192.168.10.5:8080;
        }
    }
}
```

Nginx – Load Balancing

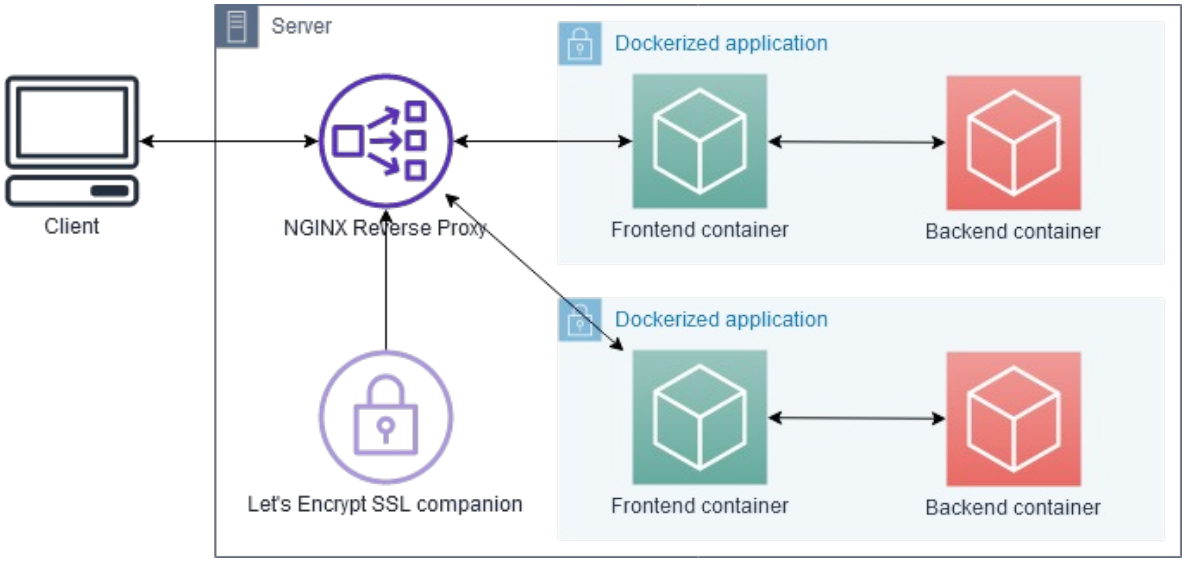
- Sometimes, in high traffic situations, we might spin up multiple instances to handle the load. NGINX can act as a load balancer and evenly distribute requests among all the instances.

```
events {}

http {
    include /opt/homebrew/etc/nginx/mime.types;

    upstream instances {
        server 192.168.10.2:8080;
        server 192.168.10.3:8080;
        server 192.168.10.4:8080;
    }

    server {
        listen 8080;

        location / {
            proxy_pass http://instances;
        }
    }
}
```



Nginx – Reverse Proxy and SSL

- The NGINX reverse proxy can monitor external ports **80** and **443** and route requests to the appropriate **Docker containers**, shielding their internal operations and ports from direct exposure to the outside world.
- Moreover, when paired with the SSL companion container, the proxy seamlessly redirects all HTTP requests to HTTPS and manages SSL encryption for all traffic, including certificate management.
- NGINX can be also dockerized, leveraging Docker DNS service to reach other containers using their hostnames.



Let’s Encrypt + Certbot

- **Let’s Encrypt** is a free, automated, and open certificate authority (CA), run for the public’s benefit. It gives people the digital certificates they need in order to enable HTTPS (SSL/TLS) for websites, for free.
  - Each free certificate has a duration of 90 days, but can be renewed (manually or automatically)
- **Certbot** is a free, open source software tool for automatically using Let’s Encrypt certificates on manually-administrated websites to enable HTTPS.
  - It can be installed on different OS. As an example, Certbot and it’s Nginx plugin can be installed on Ubuntu with apt:

```
sudo apt install certbot python3-certbot-nginx
```

Let’s Encrypt + Certbot + Nginx

- Certbot provides a variety of ways to obtain SSL certificates through plugins. The Nginx plugin will take care of reconfiguring Nginx and reloading the config whenever necessary.
- To use this plugin, type the following command to add a certificate to domains example.com/www.example.com, managed by the local instance of Nginx:

```
sudo certbot --nginx -d example.com -d www.example.com
```

Let’s Encrypt + Certbot + Nginx: A demo

```
https://blog.tericcabrel.com/deploy-spring-boot-
application-docker-nginx-reverse-proxy/
```