Load Balancing and Optimization

The concept of load balancing has the potential to solve problems pertaining to scalability, availability and performance.

Concept of Load Balancing

The load balancing concept consists of distributing the worload (CPU, HD, etc) across several servers, completely transparent to the visitors.

There are several techniques available for load balancing. The simplet is ?DNS load balancing. To achieve DNS load balancing simply associate multiple IP addresses with a domain. The OS of the visitors will select one of the IP addresses by a round-robin algorithm. This solution cannot always be applied to high traffic websites for many reasons.

Session Affinity

Session affinity is an expression that designates the persistent assgnment of a client to a particular server in a load-balanced infraestrctura. The word session describe a set of requests performed by a client on a server. When a visitor browses a website, the often visit more than one page. So the server conserves the data related to the operations performed during the visit, as session, shopping cart, login credentials, etc.

DNS load balancing does not ensure session affinity. Nginx helps to achieve it.

Upstream Module

The implementation of load balancing in Nginx is particularly clever as it allows you to distributo the load at several levels of the infrastructure. It isn't limited to proxying HTTP requests acrsoss backend servers: also offers to distribute requests across FastCGI backends, or queries to memcached servers.

The first step is to declare this group of servers with the help of the upstream block, which must be placed within http blockl. Within the upstream block, declare one or more servers with the server directive:

```
hettp {
    upstream MyUpstream {
        server 10.0.0.201;
        server 10.0.0.202;
        server 10.0.0.203;
    }
}
```

Now the server group is declared, you can reference it in your virtual host configuration. For example, distribute the incoming HTTP requests across the server group by simply proying them

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

```
        listen 80;
        root /home/example.com/www;
        # proxy all requests to the MyUpstream serve rgroup
        proxy_pass http://MyUpstream;
        ...
    }
```

Request Distribution Mechanism

Weight Flag Problem

- Weight flag: Servidor de arquitectura heterogenea.Balancear la carga entre los servidores

```
upstream MyUpstream {
    server 10.0.0.201 weight=3;
    server 10.0.0.202 weight=2;
    server 10.0.0.203;
}
```

by default server have a weight 1, unless otherwise. So every 6 http requests received, Nginx will systematically distribute 3 requests to the .201, 2 to 202 and 1 to 203.

State of Servers

- fail_timeout=N; # N is the number of seconds before a requests id considered to have failed
- max_fails=N; # once by default
- backup mark: Only use it if another server fails it
- dow nmarks: as permanently unavailable.

Achieve Session Afinity

- directives in the upstream block
- up_hash: instructs Nginx to calculate a hash from the first three bytes of the client's IP address. Keep the client assigned to a particular server based on the hash. As long as the client's ipa ddress remains the asme, Nginx will always forward requets to the same server.

```
upstream {
    server .201;
    server .202;
    ip_hash;
}
```

To deal with dynamic IP addresses, instead of client's IP address, separate the requests based on the criteria of your choice.

- hash $cookie_username; # for example

Nginx as TCP Load Balancer

- Distribute load across any form networked servers (database servers, e-mail servers, web server, etc)

Stream Module

The TCP load balancing works similar to HTTP load balancing. The module is not include in default build.

OFfers a new block called stream which must be placed the the root of the configuration file. (outside of the http block).

```
--with-stream
```

directives

- server: declares a TCP server listening on a particular port and optionally a network interface, with or without SSL.
- upstream: defines a server group in a similar manner, as seen previously.

Example of MySQL Load Balancing

Nginx configured to receive MySQL connections and balance them across two backend servers, as follows:

```
stream {
    upstream MyGroup {
        # IP Address Distribution
        hash $remote_addr;
        server 10.0.0.201 weight=2;
        server 10.0.0.202;
        server 10.0.0.203 backup; # back up only
    }
    server {
        # listen on the default MySQL port
        listen 3306;
        proxy_pass MyGroup;
    }
}
```

Additional documentation nginx.org