Nginx question answer

**1) Explain what is Nginx?**

Nginx is a web server and a reverse proxy server for HTTP, HTTPS, SMTP, POP3 and IMAP protocols.

**2) Mention some special features of Nginx?**

Special features of the Nginx server includes

* Reverse proxy/ L7 Load Balancer
* Embedded Perl interpreter
* On the fly binary upgrade
* Useful for re-writing URLs and awesome PCRE support

**3) Mention what is the difference between Nginx and Apache?**

|  |  |
| --- | --- |
| **Nginx** | **Apache** |
| Nginx is an event based web server | Apache is a process based server |
| All request are handled by a single thread. | Single thread handles a single request. |
| Nginx avoids child processes idea. | Apache is based on child processes |
| Nginx resembles speed | Apache resemble power |
| Nginx is better when it comes to memory consumption and connection | Apache is not up-to the mark when it comes to memory consumption and connection |
| Nginx is better when you want load-balancing | Apache will refuse new connections when traffic reaches the limit of processes |
| For PHP, Nginx might be preferable as it supports PHP internally | Apache support’s PHP, Python, Perl and other languages using plugins. It is useful when application is based on Python or Ruby |
| Nginx do not support O.S like IBMi and OpenVMS. | Apache support much wider range of O.S |
| Nginx comes only with core features | Apache provides lot more functionality than Nginx |
| Nginx performance and scalability do not depend on hardware | Apache is dependent on hardware components like CPU and memory |

**4) Explain how Nginx can handle HTTP requests?**

Nginx uses the reactor pattern.  The main event loop waits for the OS to signal a readiness event- such that the data is accessible to read from a socket, at which instance it is read into the buffer and processed.  A Single thread can serve tens of thousands of simultaneous connections.

**5) In Nginx how you can prevent processing requests with undefined server names?**

A server that just drops the requests can be defined as

Server {

listen                80;

server\_name  “ “ ;

return              444;

}

Here the server name is kept as an empty string which will match request without the “Host” header field, and a special Nginx’s non-standard code 444 is returned that terminates the connection.

**6) What is the advantage of using a “reverse proxy server”?**

The reverse proxy server can hide the presence and characteristics of the origin server. It acts as an intermediate between internet cloud and web server. It is good for security reason especially when you are using web hosting services.

**7) Mention what is the best usage of Nginx server?**

The best usage of Nginx server is to deploy dynamic HTTP content on a network with using SCGI, WSGI application servers, FastCGI handlers for scripts.  It can also serve as a load balancer.

**8) Mention what is the Master and Worker Processes in Nginx Server?**

* **Master processes:** It reads as well as evaluates configuration and maintains worker processes.
* **Worker processes:** It actually does the processing of the requests.

**9) Explain how you can start Nginx through a different port other than 80?**

To start Nginx through a different port, you have to go to /etc/Nginx/sites-enabled/ and if this is the default file, then you have to open file called “default.” Edit the file and put the port you want

Like server { listen 81; }

**10) Explain is it possible to replace Nginx errors like 502 error with 503?**

* 502= Bad gateway
* 503= Server overloaded

Yes, it is possible but you to ensure that **fastcgi\_intercept\_errors**is set to ON, and use the error page directive.

Location / {

fastcgi\_pass 127.0.01:9001;

fastcgi\_intercept\_errors on;

error\_page 502 =503/error\_page.html;

#…

}

**11) In Nginx, explain how you can keep double slashes in URLs?**

To keep double slashes in URLs you have to use **merge\_slashes\_off;**

Syntax: merge\_slashes **[on/off]**

Default: merge\_slashes on

Context: http, server

**12) Explain what is ngx\_http\_upstream\_module is used for?**

The ngx\_http\_upstream\_module is used to define groups of servers that can reference by the fastcgi pass, proxy pass, uwsgi pass, memcached pass and scgi pass directives.

**13) Explain what is C10K problem?**

C10K problem is referred for the network socket unable to handle a large number of client (10,000) at the same time.

**14) Mention what is the use of stub\_status and sub\_filter directives?**

* **Stub\_status directive**: This directive is used to know the current status of Nginx like current active connection, total connection accepted and handled current number of read/write/wait connection
* **Sub\_filter directive:**It is used to search and replace the content in response, and quick fix for stale data

**15) Explain does Nginx support compress the request to the upstream?**

You can compress the request to the upstream by using the Nginx module **gunzip.**The gunzip module is a filter that decompresses responses with “Content Encoding: gzip” for clients or servers that do not support “gzip” encoding method.

**16) Explain how you can get the current time in Nginx?**

To get the current time in Nginx, you have to use variables from SSI module, $date\_gmt and $date\_local.

* Proxy\_set\_header THE-TIME $date\_gmt;

**17) Explain what is the purpose of –s with Nginx Server?**

To run the executable file of Nginx –s parameter is used.

**18) Explain how to add modules in Nginx Server?**

During the compilation process, Nginx modules must be selected as such run-time selection of modules is not supported by Nginx.

**What are all feature of Nginx Server?**

* Ability to handle more than 10,000 simultaneous connections with a low memory.
* Handling of static files, index files, and auto-indexing
* Reverse proxy with caching
* Load balancing with in-band health checks
* Fault tolerance
* TLS/SSL with SNI and OCSP stapling support, via OpenSSL.
* FastCGI, SCGI, uWSGI support with caching
* FastCGI via PHP (PHP-FPM) support with caching
* Name- and IP address-based virtual servers
* IPv6-compatible
* SPDY protocol support
* WebSockets and HTTP/1.1 Upgrade (101 Switching Protocols)
* FLV and MP4 streaming
* Web page access authentication
* gzip compression and decompression
* URL rewriting
* Custom logging with on-the-fly gzip compression
* Response rate and concurrent requests limiting
* Server Side Includes
* IP address-based Geo-Location
* User tracking
* WebDAV
* XSLT data processing
* Embedded Perl scripting
* TLS/SSL support
* STARTTLS support
* SMTP, POP3, and IMAP proxy
* Authentication using an external HTTP server
* Upgrading executable and configuration without client connections loss and a module-based architecture.

**What is the Master and Worker Processes in Nginx Server?**  
Master process read and evaluate configuration, and maintain worker processes.  
Worker processes do actual processing of requests.

**What is Nginx Server?**

Nginx (pronounced "engine x") is an open source web server and a reverse proxy server for HTTP, SMTP, POP3, and IMAP protocols, with a strong focus on high concurrency, performance and low memory usage.

**What is the Best Usage of Ngins Server?**

Nginx can deploy dynamic HTTP content on a network using FastCGI, SCGI handlers for scripts, WSGI application servers or Phusion Passenger module, and it can serve as a software load balancer.

**What’s the Difference between Apache Web Server and Nginx?**

Nginx uses an asynchronous event-driven approach to handling requests, instead of the Apache HTTP Server model that defaults to a threaded or process-oriented approach. Nginx's event-driven approach can provide more predictable performance under high loads.

**Describe Some Best Features of Nginx?**

Simultaneous Connections with low memory, Auto Indexing, Load Balancing, Reverse Proxy with Caching, Fault Tolerance

**What is the Configuration File for Nginx and where it can be in UNIX like Systems?**

Configuration file is named nginx.conf and placed in the directory;

 /usr/local/nginx/conf, /etc/nginx     or

/usr/local/etc/nginx          depends on distribution.

**What is the Master and Worker Processes in Nginx Server?**

The main purpose of the master process is to read and evaluate configuration, and maintain worker processes. Worker processes do actual processing of requests.

**How to define Worker Processes?**

The number of worker processes is defined in the configuration file and may be fixed for a given configuration or automatically adjusted to the number of available CPU cores

**Where the Process ID does for Nginx Server is written?**

The process ID of the master process is written to the file /usr/local/nginx/logs/nginx.pid

**What are the controls used in Nginx Server?**

There are only few controls that are assosiated with Nginx Server and these are as below;

Nginx -s [stop | quit | reopen | reload]

**How to reload configuration file of Nginx Server?**

You can reload Nginx configuration file by running this command: nginx -s reload

**How to reopening the log files in Nginx Server?**

You can use nginx –s reopen

**What is the purpose of –s with Nginx Server?**

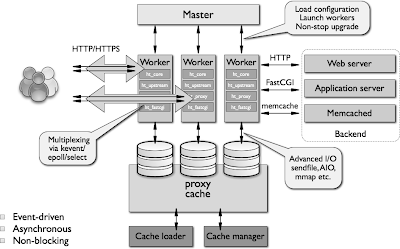
-s parameter is used to run the executable file of nginx.

**How to add Modules in Nginx Server?**

Nginx modules must be selected during compile, run-time selection of modules is not currently supported.

**Define some of Nginx Architerture?**

Here is brief nginx's architecture in diagram.



**what are the configuration files available for Nginx ?**

       1.      Default configuration directory: /etc/nginx/  
2.      Default SSL and vhost config directory: /etc/nginx/conf.d/  
3.      Default log file directory: /var/log/nginx/  
4.      Default document root directory: /usr/share/nginx/html  
5.      Default configuration file: /etc/nginx/nginx.conf  
6.      Default server access log file: /var/log/nginx/access.log  
7.      Default server access log file: /var/log/nginx/error.lo

**Sticky Session Load Balancer with Nginx**

Load balancer distributes incoming requests across multiple computing application servers. It aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource.

**Nginx supports below load balancing mechanisms**:

1. **Round-Robin** — Requests to the application servers are distributed in a round-robin fashion.  
   2. **Least-Connected** — Next request is assigned to the server with the least number of active connections.  
   3. **Ip-Hash** — A hash-function is used to determine what server should be selected for the next request (based on the client’s IP address).

Start any number of tomcat instances which are a part of a cluster, for me it’s 3 running on a localhost on port 8080, 8081, 8082 and define upstream block in /usr/local/nginx/conf/nginx.conf file, as follows:

upstream cluster-tomcat {

server 127.0.0.1:8080;

server 127.0.0.1:8081;

server 127.0.0.1:8082;

}

Above all requests are proxied to the server group cluster-tomcat in a round-robin fashion and Nginx applies HTTP load balancing to distribute the requests.

Step 4: Replace location block under server with below:

location / {

proxy\_pass http://cluster-tomcat;

}

Step 5: Enable Sticky Session adding ip\_hash directive to the server (upstream) group configuration, as follows:

upstream cluster-tomcat {

ip\_hash;

server 127.0.0.1:8080;

server 127.0.0.1:8081;

server 127.0.0.1:8082;

}

After making changes, complete http block should look like:

http {

include mime.types;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

upstream cluster-tomcat {

ip\_hash;

server 127.0.0.1:8080;

server 127.0.0.1:8081;

server 127.0.0.1:8082;

}

server {

listen 80;

server\_name localhost;

location / {

proxy\_pass http://cluster-tomcat;

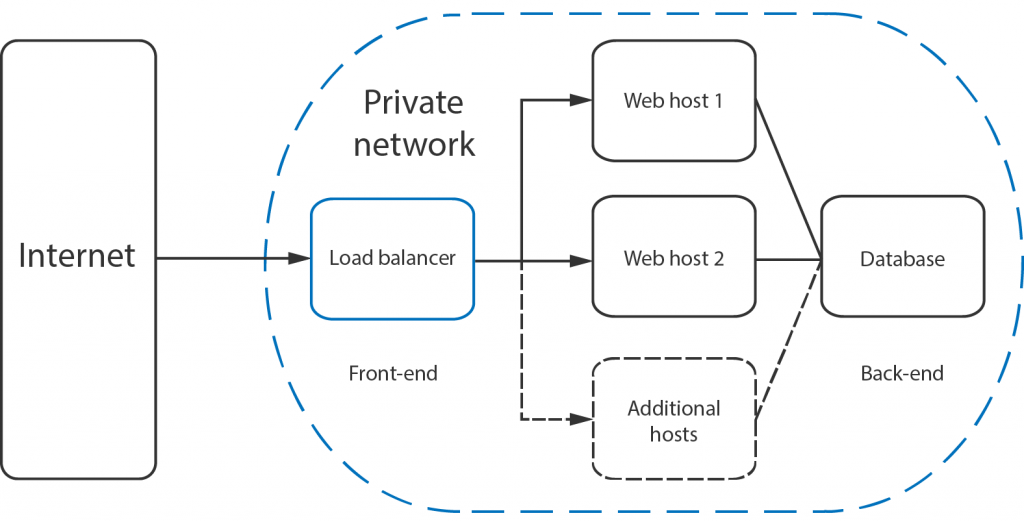
}

}

}

# How to Configure Load Balancing with Nginx

Load balancing is an excellent way to scale out your application and increase it’s performance and redundancy. Nginx, which is a popular web server software, can be configured as a simple yet powerful load balancer to improve your servers resource availability and efficiency. In a load balancing configuration nginx acts as single entrance point to a distributed web application working on multiple separate servers.



#### Installing nginx

The first thing to do is to set up a new host that will serve as your load balancer. Go ahead and deploy a new instance at your UpCloud Control Panel if you haven’t already. Currently nginx packages are available on the latest versions of CentOS, Debian and Ubuntu, so pick which ever of these you prefer.

After setting up the server the way you like, adding users, running updates, and so on, install the latest stable nginx using one of the following methods.

# Debian

sudo nano /etc/apt/sources.list

# Add the following to the end of the list

deb http://nginx.org/packages/debian/ jessie nginx

deb-src http://nginx.org/packages/debian/ jessie nginx

# Save the file, exit the editor and then run the following commands

sudo aptitude update

sudo aptitude install nginx

# Ubuntu

nginx=stable

sudo add-apt-repository ppa:nginx/$nginx

sudo apt-get update

sudo apt-get install nginx

# CentOS

sudo vi /etc/yum.repos.d/nginx.repo

# Enter the following to the file

[nginx]

name=nginx repo

baseurl=http://nginx.org/packages/centos/$releasever/$basearch/

gpgcheck=0

enabled=1

#Save and exit, then use the command below

sudo yum update

sudo yum install nginx

Once installed change directory into the nginx main configuration folder.

cd /etc/nginx/

Now depending on your OS the web server configuration files will be in one of two places.

Ubuntu and Debian follow a rule for storing virtual host files in /etc/nginx/sites-available/, which are enabled through symbolic links to /etc/nginx/sites-enabled/. You can use the command below to enable any new virtual host files.

sudo ln -s /etc/nginx/sites-available/<vhost> /etc/nginx/sites-enabled/<vhost>

CentOS users can find their host configuration files under /etc/nginx/conf.d/ in which any .conf -type virtual host file gets loaded.

Check that you can find at least the default configuration and then restart nginx.

sudo service nginx restart

Test that the server replies to HTTP requests by opening the load balancer server’s public IP address in your web browser. When you see the default welcoming page for nginx the installation was successful.

Nginx default welcome page.

If you are having trouble loading the page, check that a firewall is not blocking your connection. For example on CentOS 7 the default firewall rules do not allow HTTP traffic, enable it with the commands below.

sudo firewall-cmd --add-service=http --permanent

sudo firewall-cmd --reload

Then try reloading your browser.

Configuring nginx as a load balancer

With nginx installed and tested you can start configuring it for load balancing. In essence all you need to do is setup nginx with instructions for which type of connections to listen to and where to redirect them. To accomplish this, create a new configuration file using which ever text editor you prefer, for example with nano:

sudo nano /etc/nginx/conf.d/load-balancer.conf

In the load-balancer.conf you’ll need to define the following two segments, upstream and server, see the examples below.

# Define which servers to include in the load balancing scheme.

# It's best to use the servers' private IPs for better performance and security.

# You can find the private IPs at your UpCloud Control Panel Network section.

upstream backend {

server 10.1.0.101;

server 10.1.0.102;

server 10.1.0.103;

}

# This server accepts all traffic to port 80 and passes it to the upstream.

# Notice that the upstream name and the proxy\_pass need to match.

server {

listen 80;

location / {

proxy\_pass http://backend;

}

}

Then save the file and exit the editor.

Next you’ll need to disable the default server configuration you earlier tested was working after the installation. Again depending on your OS this part differs slightly.

On Debian and Ubuntu systems you’ll need to remove the default symbolic link from the sites-enabled folder.

sudo rm /etc/nginx/sites-enabled/default

CentOS hosts don’t use the same linking, instead simply rename the default.conf in the conf.d/ directory to something that doesn’t end with .conf, for example:

sudo mv /etc/nginx/conf.d/default.conf /etc/nginx/conf.d/default.conf.disabled

Then use the following to restart nginx.

sudo service nginx restart

Check that nginx starts successfully. If the restart fails, take a look at the /etc/nginx/conf.d/load-balancer.conf you just created to make sure there are no mistypes or missing semicolons.

You should now be passed to one of your back-end servers when entering the load balancer’s public IP address in your web browser.

**Load balancing methods**

Load balancing with nginx uses **round-robin algorithm by default**, if no other method is defined, like in the first example above. With round-robin scheme each server is selected in turns according to the order you set them in the load-balancer.conf -file. This balances the number of requests equally for short operations.

**Least connections based load balancing is an other straight forward method**. As the name suggests, this method directs **the requests to the server with the least active connections at that time**. It works more fairly than round-robin would with applications where requests might sometimes take longer to complete.

To enable least connections balancing method add the parameter least\_conn to your upstream -section as shown in the example below.

upstream backend {

least\_conn;

server 10.1.0.101;

server 10.1.0.102;

server 10.1.0.103;

}

While round-robin and least connections balancing schemes are fair and have their uses, they however cannot provide session persistence. If your web application requires that the users are subsequently directed to the same back-end server as during their previous connection, you should use IP hashing method instead. IP hashing uses the visitors IP address as a key to determine which host should be selected to server the request. This allows the visitors to be each time directed to the same server, granted that the server is available and the visitor’s IP address hasn’t changed.

To use this method, add the ip\_hash -parameter to your upstream -segment like in the example underneath.

upstream backend {

ip\_hash;

server 10.1.0.101;

server 10.1.0.102;

server 10.1.0.103;

}

In a server setup where the available resources between different hosts are not equal it might be desirable to favour some servers over others. Defining server weights allows you to further fine tune load balancing with nginx. The server with the highest weight in the load balancer is selected the most often.

upstream backend {

server 10.1.0.101 weight=4;

server 10.1.0.102 weight=2;

server 10.1.0.103;

}

For example in the configuration shown above the first server is selected twice as often as the second, which again gets twice the requests compared to the third.

Load balancing with HTTPS enabled

Enabling HTTPS for your site is a great way to protect your visitors and their data. If you haven’t yet implemented encryption on your web hosts, we highly recommend taking a look at our guide for How to Install Let’s Encrypt on Nginx.

Using encryption with a load balancer is easier than you might think. All you need to do is add an other server section to your load balancer configuration file which listens to HTTPS traffic at port 443 with SSL and set up a proxy\_pass to your upstream segment like with the HTTP in the previous example above.

Open your configuration file again for edit.

sudo nano /etc/nginx/conf.d/load-balancer.conf

Then add the following server segment to the end of the file.

server {

listen 443 ssl;

server\_name <domain name>;

ssl\_certificate /etc/letsencrypt/live/<domain name>/cert.pem;

ssl\_certificate\_key /etc/letsencrypt/live/<domain name>/privkey.pem;

location / {

proxy\_pass http://backend;

}

}

Then save the file, exit the editor and restart nginx again with

sudo service nginx restart

Setting up encryption at your load balancer while using the private network connections to your back-end has some great advantages.

As only your UpCloud servers have access to your private network, it allows you to terminate the SSL at the load balancer and thus only passing forward HTTP connections.

It also greatly simplifies your certificate management as you can obtain and renew the certificates from a single host.

With the HTTPS enabled you also have the option to enforce encryption to all connections to your load balancer. Simply update your server segment listening to port 80 with a server name and a redirection to your HTTPS port, then remove or comment out the location portion as it’s no longer needed. See the example below.

server {

listen 80;

server\_name <domain name>;

return 301 https://$server\_name$request\_uri;

#location / {

# proxy\_pass http://backend;

#}

}

Save the file again after making the changes and then restart nginx.

sudo service nginx restart

Now all connections to your load balancer will be served over encrypted HTTPS connection and requests to the unencrypted HTTP will be redirected to use HTTPS as well. This provides a seamless transition into encryption with nothing required from your visitors.

Health checks

In order to know which servers are available nginx’s implementations of reverse proxy includes passive server health checks. If a server fails to respond to a request or replies with an error, nginx will note the server as failed and will try to avoid directing connections to that server for a time.

The number of consecutive unsuccessful connection attempts within a certain time period can be defined in the load balancer configuration file by setting a parameter max\_fails to the server lines. By default, when no max\_fails is specified, this value is set to 1. Optionally setting the max\_fails to 0 will disable health checks to that server.

If max\_fails is set to a value greater than 1 the subsequent fails must happen within a specific time frame for the fails to count. This time frame is specified by a parameter fail\_timeout, which also defines how long the server should be considered failed. By default the fail\_timeout is set to 10 seconds.

After a server is marked failed and the time set by fail\_timeout has passed, nginx will begin to gracefully probe the server with client requests. If the probes return successful, the server is again marked live and included in the load balancing as normal.

upstream backend {

server 10.1.0.101 weight=5;

server 10.1.0.102 max\_fails=3 fail\_timeout=30s;

server 10.1.0.103;

}

Using the health checks allows you to adapt your server back-end to the current demand by powering up or down hosts as required. Starting up additional servers during high traffic can easily increase your application performance when new resources become automatically available to your load balancer.