Nginx Interview Questions and Answers

This document contains a list of potential interview questions and answers related to Nginx. It covers a range of topics from basic concepts to advanced configurations and troubleshooting.

# Basic Questions

Q: What is Nginx, and how does it work?

A: Nginx is a high-performance HTTP server and reverse proxy. It is designed for handling many concurrent connections and can serve as a load balancer and HTTP cache. Nginx uses an event-driven, asynchronous model, making it scalable and efficient for handling high traffic.

Q: How is Nginx different from Apache?

A: Nginx uses an event-driven architecture, while Apache uses a process/thread-based model. This allows Nginx to handle a higher number of concurrent connections with less resource usage. Nginx is also better at serving static files and acting as a reverse proxy, whereas Apache is more feature-rich and suitable for dynamic content.

Q: Explain the architecture of Nginx.

A: Nginx follows an event-driven architecture with an asynchronous, non-blocking model. It uses worker processes to handle client requests and relies on event loops to handle multiple connections simultaneously, which makes it efficient in terms of resource usage and concurrency.

Q: What are the common use cases for Nginx?

A: Common use cases for Nginx include acting as a reverse proxy, load balancing, serving static content, acting as a web server, handling SSL termination, and acting as an HTTP cache.

Q: How do you configure Nginx to serve static files?

A: To serve static files, you can use the 'location' block in Nginx's configuration file, specifying the directory where the static files are located. For example:   
  
`location /static/ {   
 root /var/www/html;   
}`

Q: What is a reverse proxy, and how is it implemented in Nginx?

A: A reverse proxy is a server that forwards requests to one or more backend servers. Nginx can be configured as a reverse proxy by using the `proxy\_pass` directive. Example:   
  
`location / {   
 proxy\_pass http://backend-server;   
}`

Q: What is a load balancer in Nginx, and how does it work?

A: A load balancer in Nginx distributes incoming traffic to multiple backend servers to balance the load. It can be implemented by using the `upstream` directive in Nginx configuration, where a list of backend servers is defined. Example:   
  
`upstream backend {   
 server backend1;   
 server backend2;   
}`

Q: What is the purpose of the worker\_processes directive in Nginx?

A: The `worker\_processes` directive defines the number of worker processes that Nginx will spawn. Each worker can handle multiple connections concurrently, so the number of workers should be set based on the system’s available CPU resources.

Q: What is the difference between HTTP and HTTPS in Nginx?

A: HTTP is a non-secure protocol, while HTTPS uses SSL/TLS to encrypt the communication between the client and server. Nginx can be configured to serve both HTTP and HTTPS, with HTTPS requiring SSL certificates.

Q: How do you enable SSL in Nginx?

A: To enable SSL, you need to configure the `ssl\_certificate` and `ssl\_certificate\_key` directives with the paths to your SSL certificate and private key. Example:   
  
`server {   
 listen 443 ssl;   
 ssl\_certificate /path/to/certificate.pem;   
 ssl\_certificate\_key /path/to/private\_key.pem;   
}`

# Configuration Questions

Q: How would you configure Nginx to forward traffic to multiple backend servers (load balancing)?

A: You can define multiple backend servers using the `upstream` directive and use `proxy\_pass` to forward traffic. Example:  
  
`upstream backend {   
 server backend1;   
 server backend2;   
}`  
`location / {   
 proxy\_pass http://backend;   
}`

Q: What is a location block in Nginx, and how do you use it?

A: A location block in Nginx defines how requests to a particular URI or path should be handled. It is used for routing and applying configurations for specific paths. Example:   
  
`location /images/ {   
 root /var/www/images;   
}`

Q: Explain the use of server\_name directive in Nginx configuration.

A: The `server\_name` directive defines the domain names or IP addresses that the server block will respond to. It helps Nginx match incoming requests to the correct virtual host. Example:   
  
`server\_name example.com;`

Q: How do you configure Nginx for caching?

A: You can configure caching in Nginx by setting the `proxy\_cache\_path` and using `proxy\_cache` in location blocks to store cache data. Example:   
  
`proxy\_cache\_path /tmp/cache levels=1:2 keys\_zone=my\_cache:10m;   
location / {   
 proxy\_cache my\_cache;   
}`

Q: How can you enable gzip compression in Nginx?

A: To enable gzip compression in Nginx, you can use the `gzip` directive in the configuration file. Example:   
  
`gzip on;   
 gzip\_types text/plain text/css application/javascript;`

Q: What is a server block in Nginx, and how do you use it for virtual hosts?

A: A server block is a section in the Nginx configuration file that defines settings for handling requests for a specific domain or IP address. Example:   
  
`server {   
 listen 80;   
 server\_name example.com;   
 root /var/www/html;   
}`

Q: How do you configure Nginx to redirect HTTP traffic to HTTPS?

A: You can configure a 301 redirect in the server block for HTTP to forward traffic to HTTPS. Example:   
  
`server {   
 listen 80;   
 server\_name example.com;   
 return 301 https://$server\_name$request\_uri;   
}`

Q: What is the try\_files directive in Nginx, and when would you use it?

A: The `try\_files` directive checks if a file exists and serves it, or otherwise routes the request to a different URI. It is often used for clean URL structures or fallback handling. Example:   
  
`location / {   
 try\_files $uri $uri/ =404;   
}`

Q: How do you configure Nginx to handle custom error pages?

A: You can define custom error pages using the `error\_page` directive. Example:   
  
`error\_page 404 /404.html;   
location = /404.html {   
 root /usr/share/nginx/html;   
}`

Q: How would you configure Nginx to limit the number of requests per second (rate limiting)?

A: You can use the `limit\_req\_zone` and `limit\_req` directives to limit the rate of requests. Example:   
  
`limit\_req\_zone $binary\_remote\_addr zone=mylimit:10m rate=1r/s;   
location / {   
 limit\_req zone=mylimit;   
}`