Investigate Nginx 413 upload errors across proxy and application limits

An upload that works for a small file but returns HTTP 413 for a larger one is usually hitting a request-size limit. Your website may have several limits: an external proxy, Nginx, PHP or another application runtime, and the application itself. Raising one limit does not guarantee the others agree.

1. Make a small, harmless comparison

Record the failing upload size, URL, UTC time and exact response. Try an ordinary small test file through the same form, using an account and destination you control. Do not repeatedly upload huge archives to a live service. A login timeout, unsupported file type or application validation error should be investigated separately.

2. Find which layer rejected the body

Read the Nginx access and error logs for that request. A message describing a client sending too large a body points toward Nginx's limit. If there is no matching origin request, inspect any upstream proxy's upload policy and logs. A branded error page is helpful evidence but can be customised, so do not identify the source from its colour alone.

```
sudo nginx -t
sudo journalctl -u nginx --since "15 minutes ago" --no-pager
```

The configuration test checks syntax, not whether an upload will succeed. Site-specific error logs may be configured outside the service journal. If you use `nginx -T` to inspect included configuration, keep its full output private: it can include sensitive settings.

3. Locate the effective Nginx limit

The `client_max_body_size` directive can be set in the HTTP, server or location context. Inspect the location that handles the failing URL, including nested includes. Editing a different virtual host or an unused configuration file will not affect the request. The limit is for the request body, so a multipart upload contains some overhead beyond the file's own size.

Choose a bounded value that fits the actual feature. Setting the directive to `0` disables its size check; it is not a useful default fix for a public upload endpoint. Preserve authentication, rate limits and file validation.

4. Check the runtime and application

For PHP uploads, compare `upload_max_filesize` with `post_max_size`; the latter must accommodate the whole POST body. Confirm the configuration used by the web-facing PHP-FPM pool. The command-line PHP configuration can be different. Do not expose a public `phpinfo()` page to inspect it.

Other frameworks have their own request parsers and per-file limits. Check the application's official documentation and available temporary storage. A request that clears Nginx can still fail later for space, permissions or application policy.

Verify a controlled correction

Back up the specific configuration before an administrator changes it, test syntax, and use the service's normal reload procedure. Retest a permitted file just below the chosen limit and confirm an oversized file remains rejected. Check the resulting file through the application, not only the HTTP status. Send support the request time, file size and redacted matching logs if ownership of a proxy layer is unclear.

Related: Nginx 502 and 504 diagnostics and check available VPS storage.

Official references

- Nginx client_max_body_size directive
- PHP POST and upload configuration
- Nginx command-line configuration tests

Revision #1
Created 2026-11-10 10:51:41 UTC by CloudExa Hosting Admin
Updated 2026-11-10 10:51:41 UTC by CloudExa Hosting Admin