

Прокси и сжатие в nginx

Таймауты, gzip и brotli, WebSocket за обратным прокси

gegemonia.com

Стандартная история: включают `gzip on`, радуются, потом кто-то замечает, что JSON-ответы API приходят несжатыми. Причина почти всегда в одном: `gzip_types` по умолчанию содержит только `text/html`.

Что настроить в первую очередь

```
gzip on;
gzip_vary on;
gzip_comp_level 5;
gzip_min_length 512;
gzip_proxied any;
gzip_types
    text/css
    text/plain
    text/xml
    application/json
    application/javascript
    application/xml
    image/svg+xml;
```

`gzip_comp_level 5` вместо дефолтной единицы — это разница между 12% и 45% экономии для JSON при почти нулевой разнице в процессорном времени. Девятка даёт ещё пару процентов и втрое больше CPU; на маленьком сервере это плохая сделка.

`gzip_vary on` обязателен, если перед `nginx` стоит кэширующий прокси: без него кэш отдаст сжатый ответ клиенту, который его не понимает.

Brotli

Если `nginx` собран с `ngx_brotli`, картина такая: `brotli` выигрывает у `gzip` на тексте примерно 15–20%, но статику надо сжимать **заранее**, а не на лету.

```
brotli on;
brotli_comp_level 6;
brotli_static on;
brotli_types text/css application/javascript application/json image/svg+xml;
```

`brotli_static on` заставляет `nginx` искать готовый `.br` рядом с файлом. Это единственный вариант, где `brotli` не стоит ничего.

О чём забывают

Не сжимать уже сжатое. JPEG, PNG, WOFF2, MP4 — они уже упакованы; попытка их `gzip`-нуть тратит процессор и иногда увеличивает размер.

Не сжимать ответы, которые отдаёт приложение потоком. Стриминговые события (SSE, длинные выгрузки) при сжатии буферизуются, и «поток» превращается в «приходит в конце». Лечится `gzip off`; для конкретного `location`.

Проверять надо не на глаз. Одна команда отвечает на все вопросы:

```
curl -sI -H 'Accept-Encoding: gzip, br' https://example.com/api/items \
| grep -iE 'content-encoding|content-length|vary'
```

Если `content-encoding` пуст, а `content-length` равен несжатому размеру — сжатие не

включилось, и весь ваш конфиг выше работает вхолостую.

И самое неочевидное: экономия от сжатия заметна на медленном канале и почти не заметна на быстром. Прежде чем оптимизировать, стоит измерить, что вообще является узким местом.

Симптом: чат работает, потом через минуту-две сообщения перестают доходить, а клиент об этом не знает. Перезагрузка страницы помогает. В логах — тишина.

Это почти всегда не WebSocket, а обратный прокси, который закрывает «простаивающее» соединение.

<figure> </figure>

Минимальный правильный блок

```
location /ws {
    proxy_pass http://127.0.0.1:8080;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;

    proxy_read_timeout 3600s;
    proxy_send_timeout 3600s;
    proxy_buffering off;
}
```

Четыре вещи здесь важны все:

- `proxy_http_version 1.1` — без него Upgrade не пройдёт, и вы получите бесконечный 101-цикл или тихий отказ.
- `proxy_set_header Upgrade/Connection` — собственно переключение протокола.
- `proxy_read_timeout` — дефолт 60 секунд. Если сервер не отправляет ничего минуту, nginx прибьёт соединение. Для чата это ровно та минута, через которую «всё сломалось».
- `proxy_buffering off` — иначе nginx копит ответы сервера и выдаёт их пачками.

Почему keeralive не спасает

Клиенты и серверы часто шлют ping каждые 30 секунд именно чтобы обойти таймаут. Это работает, но создаёт другую проблему: если ping не проходит, вы узнаете об этом только по отсутствию pong — то есть должны сами считать время. В приложении обязана быть проверка: если последний pong старше двух интервалов, соединение пересоздаётся.

Как проверить, что дело в прокси

```
timeout 120 curl -sN -o /dev/null -w '%{http_code} %{time_total}\n' \
-H 'Connection: Upgrade' -H 'Upgrade: websocket' \
-H 'Sec-WebSocket-Version: 13' -H 'Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==' \
http://127.0.0.1:8080/ws
```

Если `curl` висит ровно 60 секунд и выходит — таймаут прокси. Если рвётся сразу — дело в бэкенде.

Ещё один слой, о котором забывают: балансировщики и облачные LB поверх `nginx` держат свои таймауты, и самый маленький из них и определяет реальное время жизни соединения. Ставить `3600s` в `nginx` бессмысленно, если перед ним стоит LB с лимитом 30 секунд.