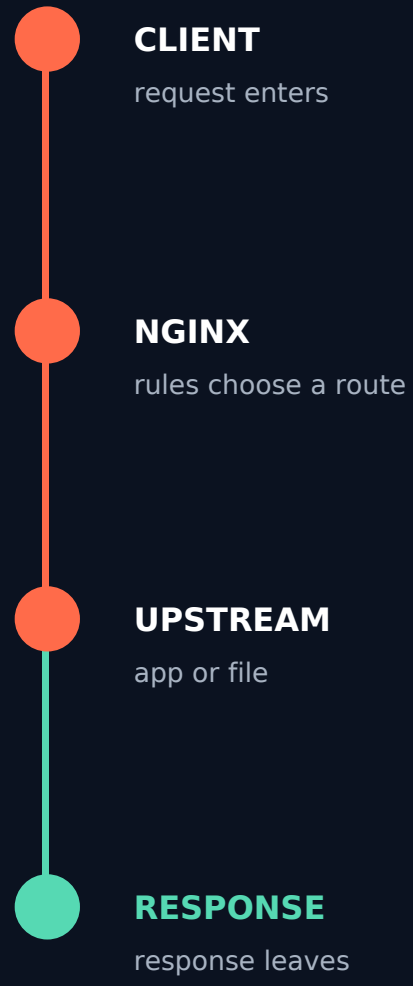
NGINX

Как запрос проходит конфиг: где ломается защита

Маршрут → правило → тест

Практический разбор: маршрут, границы доверия, проверки



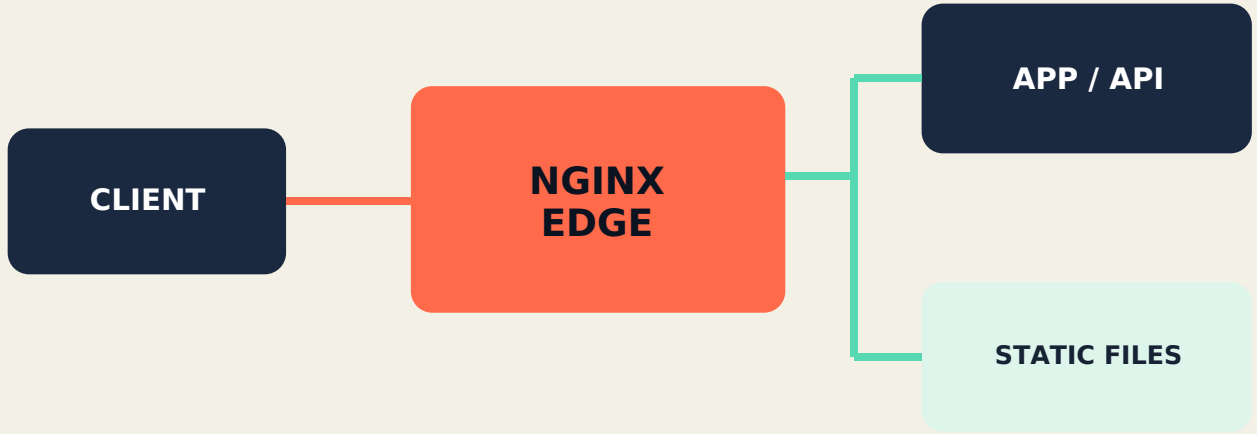
Один компонент на краю. Несколько ролей.

NGINX принимает HTTP-трафик и решает, куда вести запрос.

WEB-SERVER
отдаёт статику

REVERSE PROXY
передаёт запрос приложению

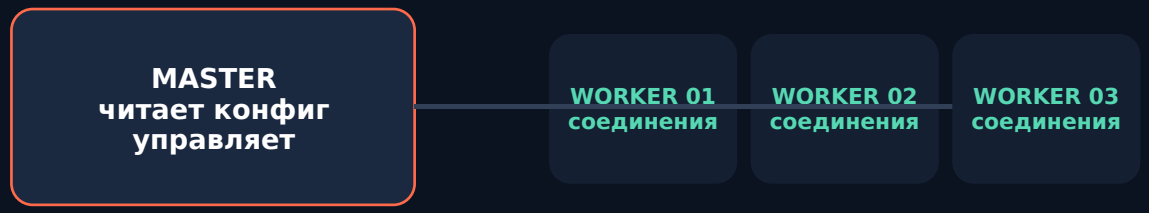
LOAD BALANCER
распределяет запросы по upstream



NGINX не заменяет приложение: бизнес-право пользователя обычно проверяется в приложении.

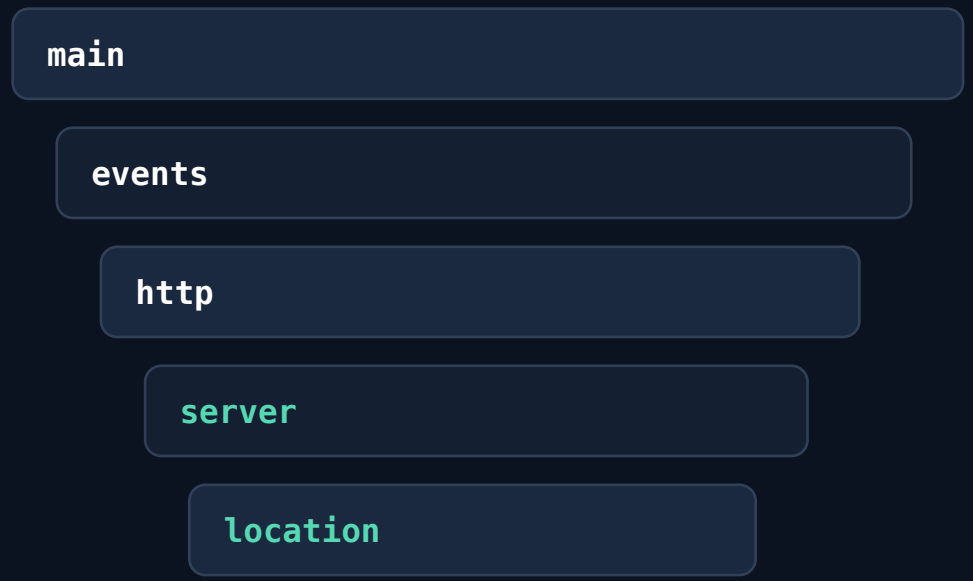
Master управляет. Workers обслуживают.

Событийная модель обслуживает множество соединений рабочими процессами.



Workers обрабатывают соединения; master управляет процессами.

КОНТЕКСТ КОНФИГУРАЦИИ



Директива должна стоять в допустимом контексте.

Пять ворот до ответа

Ошибку ищи по трассе, а не по одному подозрительному блоку.



Следующий слой: проверка прав в приложении.

location: prefix может проиграть regex

Exact (=) → самый длинный prefix → первый regex; ^~ останавливает проверку regex.

```
location /private/ { deny all; }
location ~ \.php$ { proxy_pass http://app:9000; }
```

/private/export.php может попасть в regex и уйти в app.

```
location = /private { return 403; }
location ^~ /private/ { return 403; }
```

ТЕСТ-ЗАПРОСЫ

- `/private`
- `/private/`
- `/private/export.php`
- `соседний легитимный путь`

Если доступ зависит от роли — проверяй в приложении.

root и alias задают разные пути

Сопоставь URI с файловым путём, затем проверь, что каталог вообще можно публиковать.

URI → ПУТЬ НА ДИСКЕ

root /srv/site; — `/assets/a.css → /srv/site/assets/a.css`

alias /srv/assets/; — `/assets/a.css → /srv/assets/a.css`

ЧТО МОЖЕТ УТЕЧЬ

- autoindex on → листинг каталогов
- широкий web-root → dotfiles / backups
- секреты держать вне public-каталога

Отдельный public-root · листинг выключен · dotfiles и backups закрыты точечными правилами

Неизвестный Host не должен выбирать случайный сайт

server_name _; не является wildcard. Fallback задаёт default_server для listen-адреса и порта.

HOST → ВЫБОР VIRTUAL SERVER

- known.example → свой server
- api.example → свой server
- unknown.example → default_server

```
server {
listen 80 default_server;
server_name _;
return 404;
}
```

Проверь и Host, и поведение TLS/SNI на установленной версии.

Один / меняет путь приложения

proxy_pass с URI заменяет совпавший prefix; без URI сохраняет исходный путь.

С URI В PROXY_PASS

```
location /api/ {
proxy_pass http://api:8080/;
}
```

/api/users → /users

БЕЗ URI В PROXY_PASS

```
location /api/ {
proxy_pass http://api:8080;
}
```

/api/users → /api/users

Сверь путь, который реально ждёт backend. Не угадывай по слэшу.

Forwarded-заголовок не удостоверяет личность

Клиент может прислать X-Forwarded-For сам. Доверяй только известной proxy-цепочке.




```
# NGINX direct edge
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $remote_addr;
```

Если перед NGINX есть LB, задай точные trusted CIDR. Права на объект проверяет приложение.

TLS защищает транспорт. Заголовки задают отдельную политику.

Проверь сертификат и ключ, установленную TLS-политику и ответы не только с кодом 200.

TLS EDGE

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_certificate /etc/nginx/tls/fullchain.pem;
ssl_certificate_key /etc/nginx/tls/private.key;
```

HTTP RESPONSE

```
add_header X-Content-Type-Options nosniff always;
```

- Ключ доступен только нужному процессу
- Правило add_header может не унаследоваться ожидаемым образом
- HSTS includeSubDomains включай, только если все поддомены готовы к HTTPS

Лимиты: часть контракта сервиса

Ограничение не должно спасать от DoS ценой поломки нормального upload или входа.

```
http {
limit_req_zone $binary_remote_addr zone=login:10m rate=5r/s;
server {
location /login/ {
client_max_body_size 2m;
limit_req zone=login burst=10 nodelay;
limit_req_status 429;
}
}
}
```

ОЖИДАЕМЫЙ РЕЗУЛЬТАТ

валидный запрос проходит

пороговый размер проходит

слишком большой → ожидаемый отказ

бурст → настроенный rate-limit статус

5 r/s и 2 MB указаны только для примера.

Опубликованный backend позволяет обойти NGINX

Конфиг на edge не защищает прямой публичный порт приложения.

НЕПРАВИЛЬНО



ПРАВИЛЬНО



Публикуй только задуманный edge-порт. Backend оставляй в нужной Compose-сети.

DOCS [12] [13] [14]

Сначала собери фактический маршрут

Мини-процедура перед правкой незнакомого nginx.conf.

01 `nginx -t`

синтаксис и загрузка конфигурации

02 `nginx -T`

итоговая конфигурация + includes

03 `listen → server → location`

какой маршрут выигрывает?

04 `root / alias / proxy_pass`

куда реально уходит URI?

05 `негативный + позитивный curl`

доступ закрыт? функция жива?

06 `повторить тесты после фикса`

security и regression вместе

nginx -T может показать чувствительные значения, не публикуй вывод целиком.

Найди 4 ошибки

Контракт: backend ждёт путь /api/...; /private/ должен быть закрыт всем.

```
server {
listen 80;
server_name _;
root /srv/app;
autoindex on;

location /private/ { deny all; }
location ~ \.php$ { proxy_pass http://app:9000; }
location /api/ { proxy_pass http://api:8080/; }
}
```

ЗАДАЧА

Найди четыре проблемы. Для каждой назови тест, который подтвердит эффект.

После каждого фикса: негативный и позитивный тест.

Пауза → следующий слайд

Фикс: правило и доказательство

Каждому изменению сопоставь ожидаемый тест.

HOST	listen 80 default_server; + явный отказ неизвестному Host
FILES	root /srv/app/public; · autoindex off; · секреты вне public
PRIVATE	`location = /private + location ^~ /private/;` права пользователя проверяет приложение
API	убери завершающий `/`, если backend должен получить `/api/...`

Нужный отказ работает · легитимный путь работает · backend недоступен напрямую · nginx -t чистый

Пять вопросов, которые ловят большую часть ошибок

Сначала докажи маршрут. Потом меняй правило.



01 Какой listen принимает соединение?

02 Какой server выбран для этого Host/SNI?

03 Какой location реально победил?

04 Какой файл или URI получил upstream?

05 Кто проверяет право пользователя: edge или приложение?

`PORT → SERVER → LOCATION → FILE / UPSTREAM → APP AUTH`

Источники · NGINX

Кликабельные первичные документы; актуальные значения сверяй с установленной версией.

[1] **NGINX Beginner’s Guide**

nginx.org/en/docs/beginners_guide.html

[2] **How nginx processes a request**

nginx.org/en/docs/http/request_processing.html

[3] **NGINX core module**

nginx.org/en/docs/http/ngx_http_core_module.html

[4] **NGINX autoindex module**

nginx.org/en/docs/http/ngx_http_autoindex_module.html

[5] **NGINX proxy module**

nginx.org/en/docs/http/ngx_http_proxy_module.html

[6] **NGINX realip module**

nginx.org/en/docs/http/ngx_http_realip_module.html

[7] **NGINX headers module**

nginx.org/en/docs/http/ngx_http_headers_module.html

[8] **NGINX SSL module**

nginx.org/en/docs/http/ngx_http_ssl_module.html

Проверено 3 ноября 2026 · документация NGINX, Docker и OWASP

Источники · защита и deployment

Кликабельные первичные документы; актуальные значения сверяй с установленной версией.

[9] **NGINX limit_req module**

nginx.org/en/docs/http/ngx_http_limit_req_module.html

[10] **NGINX command-line parameters**

nginx.org/en/docs/switches.html

[11] **Using nginx as HTTP load balancer**

nginx.org/en/docs/http/load_balancing.html

[12] **Docker Compose services**

docs.docker.com/reference/compose-file/services

[13] **Docker Compose networks**

docs.docker.com/reference/compose-file/networks

[14] **OWASP Authorization Cheat Sheet**

cheatsheetseries.owasp.org/…/orization_Cheat_Sheet.html

[15] **OWASP HTTP Headers Cheat Sheet**

cheatsheetseries.owasp.org/…/P_Headers_Cheat_Sheet.html

[16] **NGINX server names**

nginx.org/en/docs/http/server_names.html

Проверено 3 ноября 2026 · документация NGINX, Docker и OWASP