# Конспект лекции Web серверы. NGINX.

## Цель и задачи лекции

Цель – Познакомиться с web серверами на примере nginx.

Задачи:

1. Понять что такое web сервер.
2. Иметь представление об архитектуре nginx
3. Разобраться в отличиях nginx от apache
4. Освоить базовые настройки проксирования
5. Освоить настройки prerender
6. Освоить настройки HTTPS
7. Практика

## План занятия

1. Что такое web сервер.
2. Uniform Resource Identifier
3. HyperText Transfer Protocol
4. Описание Nginx
5. Преимущество nginx над apache
6. Структура конфигурационного файла
7. Раздача статического содержимого
8. Конфигурация prerender
9. Конфигурация https
10. Практические задание
11. Ответы на вопросы

# Конфигурирование – параметры

Различные параметры этого файла могут быть изменены при необходимости, но NGINX настолько прост, что все будет работать с настройками по умолчанию. Рассмотрим некоторые важные параметры конфигурационного файла:

* **worker\_processes**: эта настройка описывает число рабочих процессов, которые сервер будет использовать. Поскольку NGINX является однопоточным, это число обычно эквивалентно количеству ядер процессора.
* **worker\_connections**: максимальное число одновременных подключений для каждого worker\_processes, оно сообщает, сколько людей одновременно NGINX сможет обслужить. Чем это число больше, тем больше запросов пользователей сможет обработать веб-сервер.
* **access\_log & error\_log**: это файлы, которые сервер использует для логирования всех ошибок и попыток входа. Эти логи нужно в первую очередь проверять при возникновении проблем и при поиске неисправностей.
* **gzip**: это свойство устанавливает настройки сжатия GZIP для NGINX ответов. Если включить его в сочетании с другими параметрами, производительность ресурса может значительно вырасти. Из дополнительных параметров GZIP следует отметить gzip\_comp\_level, который означает уровень компрессии и бывает от 1 до 10. Обычно, это значение не должно превышать 6 т. к. при большем числе прирост производительности незначительный. gzip\_types – это список типов ответов, к которым будет применяться сжатие.

# Пример конфига

upstream remoteApplicationServer {

server 10.10.10.10;

}

upstream remoteAPIServer {

server 20.20.20.20;

server 20.20.20.21;

server 20.20.20.22;

server 20.20.20.23;

}

server {

listen 80;

server\_name www.customapp.com customapp.com

root /var/www/html;

index index.html

location / {

alias /var/www/html/customapp/;

try\_files $uri $uri/ =404;

}

location /remoteapp {

proxy\_set\_header Host $host:$server\_port;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_pass http://remoteAPIServer/;

}

location /api/v1/ {

proxy\_pass https://remoteAPIServer/api/v1/;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

proxy\_redirect http:// https://;

}

}

## 10 отличий Apache от Nginx

#### 1. Метод обработки соединений с клиентами

Издавна, **Apache** на каждый запрос от клиента создает отдельный процесс (или поток, зависит от выбранного mpm модуля). Выглядит это следующим образом - клиент отправляет запрос, веб-сервер создает отдельный процесс на этот запрос, отвечает клиенту и блокирует процесс до тех пор, пока клиент не закроет соединение. Это легко и просто в реализации, дебаге и мониторниге, но … Как вы могли бы догадаться, если у вас highload проект, то.. дела плохи. Процесс в любой ОС требует памяти и ресурсов, а когда процессов становиться неприлично много, обработка соединений неприлично замедляется, память кончается, CPU растет. Для мелких проектов такая реализация архитектуры обработки соединений не добавит головной боли, но для высоконагруженных проектов придется ставить очень мощное железо или искать альтернативные варианты.

**Nginx** состоит из master-процесса и нескольких дочерних процессов. Мастер процесс обычно один — он создает дочерние процессы (воркеры, загрузчик кеша и кеш менеджер), считывает конфигурацию и открывает порты. Воркеров обычно несколько, разработчики nginx советуют количество воркеров определять равным числу ядер машины. Эти дочерние процессы буду обслуживать все соединения с клиентами в неблокирующей манере. В nginx используется бесконечный цикл, который бежит по всем соединениями и отвечает на запросы клиентов. Когда соединение закрывается, оно удаляется из event loop. Это решение идеально подходит для проектов, которые обслуживающих 10к+ соединений одновременно. При этом, загрузка CPU и использование памяти обычно равномерны, без видимых пиков.

#### 2. Отдаваемый контент

**Apache** — может генерировать как статический контент, так и динамический. С этим никаких проблем нет. Прекрасно подойдет тем, кто не хочет заморачиваться с проксированием и настройкой дополнительного инструмента для генерации динамики, ведь Apache — это готовое работающее решение.

**Nginx** — отдает только статику и из коробки генерировать динамический контент не умеет. Если вы используете nginx и хотите генерировать динамический контент на своем сайте, то вам придется проксировать запросы тому, кто это делать умеет (apache, php-fpm и др.). Поэтому, разработчикам придется настраивать дополнительную связку, которая усложняет архитектуру, например nginx+apache (кстати в этой связке, Apache называют бекенд сервером, а Nginx — фронтендом), nginx + phpfpm, nginx + python и др.

#### 3. Конфигурирование

**Apache** полюбился разработчикам и сисадминам не в последнюю очередь из-за возможности конфигурировать обработку соединений на уровне директорий. Делается это с помощью скрытого файла .htaccess, позволяющего настраивать права доступа, авторизацию, аутентификацию, политику кеширования и др правила. Это довольно-таки удобное решение для пользователей, потому что позволяет менять конфигурацию на лету, без перезагрузки сервера и без наличия доступа к основному конфигу сервера. Но также имеется маленький минус — Apache каждый раз при обработке соединений ищет файл .htacces и считывает с него информацию, что естественно замедляет выдачу ответа клиенту (кстати, поддержку настройки конфигурирования на уровне директорий можно и отключить).

**Nginx** не поддерживает конфигурирование на уровне каталогов. Существует один конфигурационный файл на весь проект, который обрабатывает master. Если вы хотите обновить конфигурацию, то необходимо отправить сигнал SIGHUP мастеру, который в свою очередь перезагружает конфигурацию и плавно завершает работу воркеров.

#### 4. Работа с модулями

**Apache** за долгое время существования обзавелся около 60 официальными модулями, и еще большим числом неофициальных. Модули динамически подключаются, не требуют сборки и перезагрузки веб-сервера.

**Nginx** имеет около 130 официальных [модулей](https://www.nginx.com/resources/wiki/modules/). В отличие от Apache, модули Nginx не могут быть динамически загружены на лету и требуют сборки. Это гораздо сложнее, но считается безопаснее.

#### 5. Интерпретация запросов

**Apache** имеет возможность интерпретировать запрос как физический ресурс в файловой системе или как URI, который требует дополнительной обработки.

**Nginx** создан, чтобы работать и в качестве веб-сервера, и в качестве прокси-сервера. По этой причине он работает в первую очередь с URI, транслируя их при необходимости в запросы к файловой системе.

#### 6. Работа со скриптовыми языками

В **Apache** есть один модуль mod\_php и все хосты вынуждены работать с одной и той же версией php и одним конфигурационным файлом.

В случае с **nginx**, каждый виртуалхост будет выполняться в отдельном процессе и, соответственно, может использовать разные версии php (python/ruby/perl и др.). Каждый процесс может иметь свою собственную независимую конфигурацию.

Вообще, в высоконагруженных проектах удобнее держать раздельно nginx и php. По отдельности их проще мониторить, ловить баги или узкие места. «Все-в-одном» Apache+mod\_php в этом плане менее удобен.

От основных архитектурных и функциональных отличий переходим к показательным отличиям, отличиям инфрастктуры и всего того, что также является не менее важным при выборе веб-сервера.

#### 7. Скорость работы

Скорость работы веб-сервера обычно измеряют для 2-х случаев отдачи контента: для статики и динамики. На основе тестов производительности, **Nginx** примерно в 2.5 раза быстрее отдает статику, чем **Apache**. Это довольно-таки большое превосходство. Если вам необходимо обслуживать большое количество статического контента, Nginx — лучший выбор. Во время тестирования отдачи динамического контента, Apache и Nginx показывают примерно одинаковые результаты. С точки зрения памяти, оба сервера используют один и тот же объем ресурсов. (Подробнее о тестах скорости отдачи контента можно почитать [здесь](http://www.speedemy.com/apache-vs-nginx-2015/))

#### 8. Поддержка ОС

**Apache** прекрасно работает на Unix-подобных операционных системах, также разработчики этого веб-сервера полностью поддерживают линейку Microsoft Windows, включая последние версии этой ОС.

**Nginx** также поддерживает работу на множестве Unix-подобных ОС и имеет некоторую поддержку Windows, которая не является полной. Но разве кто-то в наше время размещает веб-сервер на Windows?

#### 9. Сообщество и поддержка

**Apache** на рынке с 1995 года, что очень немалый срок, обеспечивший инструменту огромное сообщество и поддержку с его стороны. Практически на все вопросы на Stack Overflow уже есть исчерпывающие ответы. Коммерческой поддержки нет.

**Nginx** веб-сервер более молодой, на рынке он с 2004 года, что также не помешало большому сообществу сформироваться и поддерживать друг друга. Nginx, в отличие от Apache, имеет коммерческую версию Nginx Plus, которая дополнена инструментами балансировки нагрузки, мониторинга, потоковой передачи медиа и др.

#### 10. Документация и обучение

И у **Apache** и у **Nginx** присутствует доступная официальная документация.

Nginx предлагает платное обучение, включающее в себя онлайн курсы, практические занятия и экзамен. По окончании курса все участники получают сертификаты. Например, сдать экзамен по основам nginx и получить официальный сертификат обойдется в 49$ (подробнее [здесь](https://university.nginx.com/)).

## Настройка prerender

location @prerender {

proxy\_set\_header X-Prerender-Token SqQuTRI4qJQwxBJ6bYNT;

set $prerender 0;

if ($http\_user\_agent ~\* "googlebot|bingbot|yandex|yandexbot|YandexBot|baiduspider|twitterbot|facebookexternalhit|rogerbot|linkedinbot|embedly|quora link preview|showyoubot|outbrain|pinterest|slackbot|vkShare|W3C\_Validator") {

set $prerender 1;

}

if ($args ~ "escaped\_fragment") {

set $prerender 1;

}

if ($http\_user\_agent ~ "Prerender") {

set $prerender 0;

}

if ($uri ~\* "\.(js|css|xml|less|png|jpg|jpeg|gif|pdf|doc|txt|ico|rss|zip|mp3|rar|exe|wmv|doc|avi|ppt|mpg|mpeg|tif|wav|mov|psd|ai|xls|mp4|m4a|swf|dat|dmg|iso|flv|m4v|torrent|ttf|woff|svg|eot)") {

set $prerender 0;

}

#resolve using Google's DNS server to force DNS resolution and prevent caching of IPs

resolver 8.8.8.8;

if ($prerender = 1) {

# setting prerender as a variable forces DNS resolution since nginx caches IPs and doesnt play well with load balancing

set $prerender "service.prerender.io";

rewrite .\* /$scheme://$host$request\_uri? break;

proxy\_pass http://$prerender;

}

if ($prerender = 0) {

rewrite .\* /index.html break;

}

}

location / {

try\_files $uri @prerender;

}

## Настройка HTTPS

server {

listen 443 ssl default\_server;

server\_name uds-moscow.ru;

root /var/www/html/some\_site;

ssl on;

ssl\_stapling on;

ssl\_stapling\_verify on;

# наши сертификаты

ssl\_certificate /etc/ssl/bundle.crt;

ssl\_certificate\_key /etc/ssl/privatekey.key;

# настраиваем сессию

ssl\_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;

# проксирование данных

location /api/ {

proxy\_pass http://some\_site/api/;

}

## Литература и ссылки

официальная документация <https://nginx.org/ru/docs/>

## Вопросы для самоконтроля

1. Что такое web сервер?
2. Назовите 3 преимущества nginx над apache.
3. Для чего нужен prerender?
4. Без чего невозможна настройка https?
5. Зачем нужен reverse-proxy.