**Контрольні запитання**

1. **Що таке NGINX? Перелічіть області застосування NGINX.**

**NGINX [engine x]** - це HTTP-сервер і зворотній проксі-сервер, поштовий проксі-сервер, а також TCP / UDP проксі-сервер загального призначення, спочатку написаний Ігорем Сисоєвим. Уже тривалий час він обслуговує сервери багатьох високонавантажених сайтів.

Практичне застосування NGINX:

* При наявності великої кількості статичного контенту або файлів для завантаження, можна налаштувати на окремому порту або IP, щоб здійснювати роздачу. При великій кількості запитів рекомендується ставити окремий сервер і підключати до нього Nginx.
* Балансування навантаження
* Стрімінг
* API gateway

1. **Поясніть процес конфігурації NGINX.**

NGINX використовує контексти конфігурації, які описані в різних файлах конфігурації. Базовий файл nginx.conf. Розташування цього файлу буде залежати від того, як ви встановили програмне забезпечення на свій комп'ютер. Для багатьох дистрибутивів файл буде знаходитися в /etc/nginx/nginx.conf. Якщо він там не існує, він також може знаходитися в /usr/local/nginx/conf/nginx.conf або /usr/local/etc/nginx/nginx.conf.

**Основний контекст**

Перше, що ви повинні помітити при перегляді основного файлу конфігурації, це те, що він виглядає організованим у вигляді дерева, визначеної наборами дужок (які виглядають як + {+ і +} +). Мовою Nginx області, які визначають ці дужки, називаються «контекстами», тому що вони містять деталі конфігурації, які розділені відповідно до галузі їх інтересів. По суті, ці підрозділи забезпечують організаційну структуру разом з деякою умовною логікою, щоб вирішити, чи застосовувати конфігурації всередині.

Оскільки контексти можуть бути нашаровані один на одного, Nginx забезпечує рівень успадкування директив. Як правило, якщо директива дійсна в декількох вкладених областях, оголошення в ширшому контексті буде передано будь-кому дочірньому контексту як значення за замовчуванням. Дочірні контексти можуть перевизначати ці значення за бажанням. Варто відзначити, що перевизначення будь-яких директив типу масиву replace попереднє значення, а не додається до нього.

Перша група контекстів, які ми обговоримо, — це основні контексти, які Nginx використовує для створення ієрархічного дерева і поділу проблем дискретних блоків конфігурації. Це контексти, які складають основну структуру конфігурації Nginx.

# The main context is here, outside any other contexts

. . .{ . . .}

Кажуть, що будь-яка директива, яка існує повністю поза цих блоків, мешкає в «головному» контексті. Пам'ятайте, що якщо ваша конфігурація Nginx налаштована за модульним принципом, деякі файли будуть містити інструкції, які, мабуть, існують поза контекстом, укладеного у дужки, але які будуть включені в такий контекст, коли конфігурація зшита разом.

**Контекст подій**

Контекст «подій» міститься в «основному» контексті. Він використовується для установки глобальних параметрів, які впливають на те, як Nginx обробляє з'єднання на загальному рівні. У конфігурації Nginx може бути тільки один контекст подій.

# main context

events {

# events context

. . .}

**HTTP - контекст**

Під час налаштування Nginx в якості веб-сервера або зворотного проксі-сервера контекст http буде містити більшу частину конфігурації. Цей контекст буде містити всі директиви та інші контексти, необхідні для визначення того, як програма буде обробляти з'єднання HTTP або HTTPS.

Контекст http є родинним контекстом подій, тому вони повинні бути перераховані поруч, а не вкладеними. Вони обидва діти основного контексту:

# main context

events {

# events context

. . .}

http {

# http context . . .}

**Контекст сервера**

Контекст «сервера» оголошується в контексті «http». Це наш перший приклад вкладених контекстів. Це також перший контекст, який допускає кілька оголошень.

Загальний формат для контексту сервера може виглядати приблизно так. Пам'ятайте, що вони знаходяться в контексті http:

# main context

http {

# http context

server { # first server context

}server {

# second server context }}

Причиною дозволу кількох оголошень контексту сервера є те, що кожен екземпляр визначає певний віртуальний сервер для обробки клієнтських запитів. Ви можете мати стільки серверних блоків, скільки вам потрібно, кожен з яких може обробляти певну підмножину з'єднань.

Через можливість і ймовірність використання декількох серверних блоків цей тип контексту також є першим, що Nginx повинен використовувати алгоритм вибору для прийняття рішень. Кожен клієнтський запит буде оброблятися відповідно з конфігурацією, визначеною в контексті одного сервера, тому Nginx повинен вирішити, який контекст сервера є найбільш гідною кандидатурою, ґрунтуючись на деталях запиту. Директиви, які вирішують, чи буде використовуватися блок сервера для відповіді на запит:

* Listen: комбінація IP-адреси / порту, на яку призначений цей блок сервера. Якщо клієнт робить запит, відповідний цим значенням, цей блок потенційно буде обраний для обробки з'єднання.
* Server\_name: ця директива є іншим компонентом, використовуваним для вибору блоку сервера для обробки. За наявності кількох серверних блоків з директивами listen однаковою специфічності, які можуть обробити запит, Nginx проаналізує заголовок «Host» запиту і порівняє його з цією директивою.

**Контекст місцезнаходження**

Наступний контекст, з яким ви будете мати справу регулярно, це контекст розташування. Контексти розташування мають багато реляційних якостей з контекстами сервера. Наприклад, можуть бути визначені кілька контекстів місця розташування, кожне місце розташування використовується для обробки певного типу клієнтського запиту, і кожне місце розташування вибирається за допомогою зіставлення визначення місця розташування з клієнтським запитом за допомогою алгоритму вибору.

У той час як директиви, що визначають, чи слід вибирати блок сервера, визначені на сервері context, компонент, який визначає здатність розташування обробляти запит, знаходиться в розташуванні definition (рядок, яка відкриває блок розташування).

Блоки розташування живуть в контексті сервера і, на відміну від блоків сервера, можуть бути вкладені одна в одну. Це може бути корисно для створення більш загального контексту розташування, щоб зловити певну підмножину трафіку, і потім подальшої обробки його на основі більш певних критеріїв з додатковими контекстами усередині:

# main context

server {

# server context

location {

# first location context } location {

# second location context

location {

# first nested location

} location {

# second nested location

} }}

1. **Що таке балансування навантаження? Перелічіть і поясніть методи балансування.**

У термінології комп'ютерних мереж балансування навантаження або вирівнювання навантаження (англ. Load balancing) - метод розподілу завдань між декількома мережевими пристроями (наприклад, серверами) з метою оптимізації використання ресурсів, скорочення часу обслуговування запитів, горизонтального масштабування кластера (динамічне додавання / видалення пристроїв), а також забезпечення відмовостійкості (резервування).

* **Round Robin**

Перший з більш організованих методів балансування навантаження, круговий робот дуже схожий на однойменний стиль ігрового турніру. Кожному серверу в серверному пулі призначається місце в загальному порядку використання, і кожного разу, коли надходить новий трафік, він переходить на наступний сервер у списку.

Round Robin гарантує, що кожен сервер може адресувати вхідний трафік. Проблеми виникають, однак, коли враховується довжина або обробка попиту на з'єднання. Коли довгі з'єднання або з'єднання, що протікають через них значного трафіку, починають складатись на сервері, деякі сервери можуть закінчувати набагато більший трафік, ніж інші, попри те, що серверам надано рівне з'єднання.

* **Weighted Round Robin**

Модифікація методу Round Robin, яка також бере до уваги ваги сервера.

* **Least connections, weighted least connections**

У подібному до хешування класу джерела IP метод найменшого підключення фокусує свої зусилля щодо збалансування навантаження на розподіл трафіку на сервери, які в даний час мають найменше активне з'єднання. Ідея полягає в тому, що будь-який один сервер у пулі серверів ніколи не повинен закінчуватись значно більшою кількістю активних з'єднань, ніж будь-який інший.

Попри те, що у цього методу є проблеми із більшим часом з’єднання трафіку, що зберігаються на одному сервері, він також, за задумом, вирішує цю проблему краще, ніж інші методи. Навіть при триваліших або більш складних сесіях, розміщених на сервері, сервер ніколи не потрапить на значно більшу кількість користувачів, ніж будь-який інший сервер, що допомагає стримувати питання попиту.

* **Source IP hash**

IP-хешування працює для розподілу навантаження на основі вхідної IP-адреси серверного запиту, що робить його набагато складнішим, ніж раніше згадані методи. Вхідному навантаженню алгоритмічно присвоюється хеш-ключ на основі його вихідної IP-адреси та призначення, який потім використовується для призначення сервера для обробки вхідного навантаження.

IP-хешування може бути надзвичайно ефективним способом обробки вхідного трафіку, але є улов: Що робити, якщо з однієї IP-адреси надходить тонна трафіку? Це може призвести до перевантаження на одному сервері. Подолання цієї проблеми передбачає встановлення правил місткості або для кількості підключень на одному сервері з одного джерела, або для кількості підключень з одного джерела.

* **Generic Hash**

В цьому методі ми можемо назначити власну hash - функцію для балансування навантаження.

* **Random**

На сьогодні найменш організований з усіх методів балансування навантаження, випадкове присвоєння виконує саме те, що говорить: Він випадковим чином присвоює кожне робоче навантаження серверу в групі серверів (пул серверів).

Теорія, що стоїть за випадковим призначенням, звучить складніше, ніж є. У теорії ймовірностей Закон великих чисел говорить про те, що зі збільшенням кількості вибірки середній (середній) результат у наборі вибірки з часом буде відповідати середньому (середньому) результату. Застосовуваний тут, це означає, що чим більше випадковим чином навантаження робочим навантаженням присвоюється серверу в пулі, врешті-решт кожен сервер у пулі буде обробляти приблизно однакові робочі навантаження, хоча завантаження можуть спочатку бути неоднаковими.

1. **Поясніть конфігурацію NGINX для балансування навантаження.**

Найпростіша конфігурація для балансування навантаження з nginx може виглядати наступним чином:

http {

upstream myapp1 {

server srv1.example.com;

server srv2.example.com;

server srv3.example.com;

} server {

listen 80;

location / {

proxy\_pass http://myapp1; } }}

У наведеному вище прикладі є 3 екземпляри однієї програми, що працює на srv1-srv3. Коли метод балансування навантаження не вказаний, він за замовчуванням є Round Robin. Усі запити передаються в проксі до групи серверів myapp1, і nginx застосовує балансування навантаження HTTP для розподілу запитів.

Інший метод балансування навантаження є least-connected. Метод least-connected в nginx активується, коли директива least-connected використовується як частина конфігурації групи серверів:

upstream myapp1 {

least\_conn;

server srv1.example.com;

server srv2.example.com;

server srv3.example.com;

}

Зауважте, що при балансуванні навантаження Round Robin або Least Connected кожен наступний запит клієнта може бути потенційно розподілений на інший сервер. Немає гарантії того, що той самий клієнт завжди буде спрямований на один і той же сервер.

Якщо є необхідність прив’язати клієнта до конкретного сервера додатків — іншими словами, зробіть сеанс клієнта "липким" або "стійким" з точки зору того, щоб завжди намагатися вибрати конкретний сервер — метод ip\_hash може бути вибраний.

upstream myapp1 {

ip\_hash;

server srv1.example.com;

server srv2.example.com;

server srv3.example.com;}

Також можливо ще більше впливати на алгоритми балансування навантаження nginx, використовуючи ваги сервера.

upstream myapp1 {

server srv1.example.com weight=3;

server srv2.example.com;

server srv3.example.com;}

upstream myapp1 {

least\_conn;

server srv1.example.com weight=3;

server srv2.example.com;

server srv3.example.com;}