Практична робота №7

ЛОГУВАННЯ ТА ЗБІР ЖУРНАЛІВ NGINX

З ВИКОРИСТАННЯМ GRAFANA ALLOY ТА LOKI

Мета заняття: набути практичних навичок налаштування структурованого логування у Nginx, ротації та аналізу журналів командним рядком, а також розгортання стека збору та централізованої візуалізації логів на основі Grafana Alloy і Loki з доступом до графічного інтерфейсу з хост-машини.

Завдання на роботу

1. Підготовка середовища.

Перед виконанням основних завдань необхідно налаштувати мережу Vagrant-машини для забезпечення доступу з хост-машини, встановити необхідні утиліти та Docker.

a. Відкрити `Vagrantfile` та переконатись у наявності рядка приватної мережі (рис. 1.a):

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/jammy64"

  # Прокидання HTTP та HTTPS портів на хост-машину
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.network "forwarded_port", guest: 443, host: 8443
  config.vm.network "forwarded_port", guest: 3000, host: 3000



  # Приватна мережа для IP-фільтрації
  config.vm.network "private_network", ip: "192.168.56.10"
end
```

Рис. 1.a – Рядок приватної мережі у Vagrantfile

IP-адреса `192.168.56.10` буде доступна з хост-машини — за нею відкриватиметься Grafana у браузері (`http://192.168.56.10:3000`).

Перевірити застосовану конфігурацію, виконавши команду:

```
vagrant reload
```

Зробити висновки за отриманими результатами.

b. Зайти у VM та встановити необхідні утиліти:

```
vagrant ssh
sudo apt update && sudo apt install -y curl jq apache2-utils
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl --version && jq --version && ab -V 2>&1 | head -1
```

Зробити висновки за отриманими результатами.

c. Перевірити або встановити Nginx:

```
sudo apt install -y nginx
sudo systemctl enable nginx && sudo systemctl start nginx
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo systemctl status nginx --no-pager
```

Зробити висновки за отриманими результатами.

d. Встановити Docker та docker compose:

```
sudo apt install -y docker.io docker-compose-v2
sudo usermod -aG docker vagrant && newgrp docker
```

Перевірити застосовану конфігурацію, виконавши команду:

```
docker --version && docker compose version
```

Зробити висновки за отриманими результатами.

2. Створення тестового сайту для практичної роботи.

Кожна практична робота використовує власний тестовий сайт з унікальним доменним ім'ям. Це забезпечує чітку ідентифікацію трафіку у журналах Nginx та усуває неоднозначність при наявності кількох vhost-ів від попередніх практичних. Тестовий сайт містить статичні ресурси (CSS, зображення), наявність яких необхідна для демонстрації умовного логування у завданні 5.

a. Створити структуру директорій та файл `index.html` (рис. 2.a):

```
sudo mkdir -p /var/www/logging-lab-xxx-yyy-zzz/{css,img}
```

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Logging Lab — Практична №7</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
    <h1>Logging Lab</h1>
    <p>Тестовий сайт для практичної роботи №7.</p>
    <img src="/img/logo.png" alt="logo">
</body>
</html>
```

Рис. 2.a – Вміст /var/www/logging-lab-xxx-yyy-zzz/index.html

```
sudo vi /var/www/logging-lab-xxx-yyy-zzz/index.html
```

b. Створити CSS-файл та файл-заглушку зображення (рис. 2.b):

```
body { font-family: Arial, sans-serif; margin: 40px; }
h1   { color: #2c3e50; }
```

Рис. 2.b – Вміст /var/www/logging-lab-xxx-yyy-zzz/css/style.css

```
sudo vi /var/www/logging-lab-xxx-yyy-zzz/css/style.css
sudo dd if=/dev/urandom bs=1k count=2 2>/dev/null | sudo tee
/var/www/logging-lab-xxx-yyy-zzz/img/logo.png > /dev/null
```

Перевірити застосовану конфігурацію, виконавши команду:

```
ls -la /var/www/logging-lab-xxx-yyy-zzz/css/ /var/www/logging-lab-xxx-
yyy-zzz/img/
```

Зробити висновки за отриманими результатами.

c. Створити конфігурацію Nginx для тестового сайту (рис. 2.c):

```
server {
    listen 80;
    server_name logging-lab-xxx-yyy-zzz.local;

    root /var/www/logging-lab-xxx-yyy-zzz;
    index index.html;

    # Окремі журнали для цього vhost-а
    access_log /var/log/nginx/logging-lab-access.log;
    error_log  /var/log/nginx/logging-lab-error.log warn;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Рис. 2.c – Конфігурація Nginx для тестового сайту

```
sudo vi /etc/nginx/sites-available/logging-lab-xxx-yyy-zzz
```

d. Активувати сайт та перезавантажити Nginx:

```
sudo ln -s /etc/nginx/sites-available/logging-lab-xxx-yyy-zzz
/etc/nginx/sites-enabled/
sudo nginx -t && sudo systemctl reload nginx
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo nginx -T | grep "logging-lab"
```

e. Додати домен до `/etc/hosts` та перевірити доступність сайту:

```
echo "127.0.0.1  logging-lab-xxx-yyy-zzz.local" | sudo tee -a
/etc/hosts
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s -o /dev/null -w "%{http_code}" http://logging-lab-xxx-yyy-
zzz.local
```

f. Згенерувати початковий тестовий трафік та перевірити журнал:

```
ab -n 20 -c 2 http://logging-lab-xxx-yyy-zzz.local/
curl -s http://logging-lab-xxx-yyy-zzz.local/css/style.css -o
/dev/null
curl -s http://logging-lab-xxx-yyy-zzz.local/img/logo.png -o /dev/null
sudo tail -5 /var/log/nginx/logging-lab-access.log
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo wc -l /var/log/nginx/logging-lab-access.log
```

3. Базове налаштування журналів Nginx.

Nginx веде два типи журналів: журнал доступу (access log) та журнал помилок (error log). Журнал доступу фіксує кожен HTTP-запит, журнал помилок — внутрішні події. Рівень деталізації error_log визначає мінімальну серйозність події для запису: debug, info, notice, warn, error, crit, alert, emerg. Кожен рівень включає всі більш серйозні рівні.

a. Ознайомитись з поточною конфігурацією журналювання у nginx.conf:

```
sudo grep -n "log" /etc/nginx/nginx.conf
```

Директиви `access_log` та `error_log` у секції `http {}` є глобальними налаштуваннями. Директиви у секції `server {}` тестового сайту (завдання 2) перекривають їх для цього vhost-а.

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo cat /etc/nginx/nginx.conf | grep -A1 -B1 "log"
```

Зробити висновки за отриманими результатами.

b. Переглянути структуру рядків у журналі доступу тестового сайту (рис. 3.b):

```
sudo tail -10 /var/log/nginx/logging-lab-access.log
```

```
# Формат combined (за замовчуванням):
# $remote_addr - $remote_user [$time_local] "$request"
# $status $body_bytes_sent "$http_referer" "$http_user_agent"


# Приклад:
127.0.0.1 - - [04/Jan/2025:12:00:01 +0000]
  "GET / HTTP/1.1" 200 347 "-" "ApacheBench/2.3"


127.0.0.1 - - [04/Jan/2025:12:00:02 +0000]
  "GET /css/style.css HTTP/1.1" 200 58
  "http://logging-lab-xxx-yyy-zzz.local/" "curl/7.81.0"
```

Рис. 3.b – Структура рядка формату combined та приклади записів

Поля: IP клієнта, ident (-), користувач (-), дата/час, рядок запиту, код відповіді, розмір тіла, Referer, User-Agent.

c. Згенерувати помилкові запити та переглянути журнал помилок:

```
curl -s http://logging-lab-xxx-yyy-zzz.local/nonexistent -o /dev/null
-w "%{http_code}"
sudo tail -10 /var/log/nginx/logging-lab-error.log
```

Відповідь `404` є штатною HTTP-відповіддю і не потрапляє до error_log. До error_log записуються внутрішні помилки Nginx: збої `fastcgi`, проблеми з upstream, помилки `SSL`, перевищення лімітів тощо. Рівень `warn` налаштований у завданні 2.

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo wc -l /var/log/nginx/logging-lab-error.log
/var/log/nginx/logging-lab-access.log
```

Зробити висновки за отриманими результатами.

4. Налаштування кастомного та JSON-формату журналів.

Стандартний формат combined не містить метрик продуктивності — часу обробки запиту та часу відповіді upstream-сервера. Директива log_format дозволяє визначити власний формат із будь-якими змінними Nginx. JSON-формат з параметром escape=json спрощує подальшу автоматизовану обробку агентами збору журналів — зокрема Grafana Alloy.

a. Визначити кастомний формат `extended` у секції `http {}` файлу `/etc/nginx/nginx.conf` (рис. 4.a):

```
http {
    # Розширений формат з метриками продуктивності
    log_format extended
        '$remote_addr - $remote_user [$time_local] '
        '"$request" $status $body_bytes_sent '
        '"$http_referer" "$http_user_agent" '
        'rt=$request_time '
        'urt="$upstream_response_time" '
        'ucs="$upstream_cache_status"';
    ...
}
```

Рис. 4.a – Формат extended у секції http {}

Змінна `$request_time` — час від отримання першого байта запиту до відправки останнього байта відповіді (секунди). `$upstream_response_time` — час відповіді бекенду (актуально для proxy/fastcgi). `$upstream_cache_status` — результат перевірки кешу.

```
sudo nginx -t && sudo systemctl reload nginx
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo nginx -T | grep -A8 "log_format extended"
```

Зробити висновки за отриманими результатами.

b. Визначити JSON-формат `json_combined` відразу після `extended` (рис. 4.b):

```
    # JSON-формат для парсингу агентом Grafana Alloy
    log_format json_combined escape=json
        '{'
        '"time":"$time_iso8601",'
        '"remote_addr":"$remote_addr",'
        '"request_method":"$request_method",'
        '"request_uri":"$request_uri",'
        '"server_protocol":"$server_protocol",'
        '"host":"$host",'
        '"status":$status,'
        '"body_bytes_sent":$body_bytes_sent,'
        '"http_referer":"$http_referer",'
        '"http_user_agent":"$http_user_agent",'
        '"request_time":$request_time,'
        '"upstream_response_time":"$upstream_response_time",'
        '"upstream_cache_status":"$upstream_cache_status"'
        '}';
```

Рис. 4.b – JSON-формат журналу json_combined

Параметр `escape=json` екранує спеціальні символи у значеннях змінних — лапки, зворотні слеші, керуючі символи. Без нього User-Agent або

URI з лапками зламають структуру JSON. Поле `"host"` фіксує ім'я vhost-а і використовуватиметься як мітка у Loki.

```
sudo nginx -t && sudo systemctl reload nginx
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo nginx -T | grep -A15 "log_format json_combined"
```

Зробити висновки за отриманими результатами.

c. Підключити обидва формати у конфігурацію тестового сайту. Оновити `/etc/nginx/sites-available/logging-lab-xxx-yyy-zzz` (рис. 4.c):

```
server {
    listen 80;
    server_name logging-lab-xxx-yyy-zzz.local;

    root /var/www/logging-lab-xxx-yyy-zzz;
    index index.html;

    # Журнал у розширеному форматі (для читання людиною)
    access_log /var/log/nginx/logging-lab-access.log extended;

    # Журнал у JSON-форматі (для Grafana Alloy)
    access_log /var/log/nginx/logging-lab-access-json.log json_combined;

    error_log /var/log/nginx/logging-lab-error.log warn;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Рис. 4.c – Оновлений server {} з двома директивами access_log

Nginx підтримує кілька директив `access_log` в одному блоці — кожен запит записується в обидва файли незалежно. Буферизацію буде додано у завданні 6.

```
sudo nginx -t && sudo systemctl reload nginx
```

d. Згенерувати трафік та перевірити обидва формати:

```
ab -n 30 -c 3 http://logging-lab-xxx-yyy-zzz.local/
sudo tail -3 /var/log/nginx/logging-lab-access.log
sudo tail -3 /var/log/nginx/logging-lab-access-json.log | jq .
```

Рядки у `logging-lab-access.log` повинні містити поля `rt=` та `urt=`. Рядки у `logging-lab-access-json.log` повинні парситись `jq` без помилок і містити поле `"host":"logging-lab-xxx-yyy-zzz.local"`.

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo cat /var/log/nginx/logging-lab-access-json.log | jq
'{status,request_uri,request_time,host}'
```

Зробити висновки за отриманими результатами.

5. Умовне логування — виключення статичних ресурсів.

У виробничих середовищах до 70–80% запитів можуть бути до статичних файлів (CSS, зображення, шрифти, favicon) — вони не несуть аналітичної цінності, але займають місце у журналах та ускладнюють пошук. Умовне логування через директиву map дозволяє вибірково відключати запис для конкретних URI. Цей підхід є безпечним антипатерном — на відміну від використання if у location {}, блок map обчислюється один раз на запит у фазі обробки змінних.

a. Додати блок `map` у секцію `http {}` файлу `/etc/nginx/nginx.conf` перед першим `server {}` (рис. 5.a):

```
http {
    ...  # log_format extended та json_combined вже тут

    # Умовне логування: 0 = не записувати, 1 = записувати
    map $request_uri $loggable {
        default                                 1;
        ~*\.(css|js|ico|gif|png|jpg|jpeg|svg)$   0;
        ~*\.(woff2|woff|ttf|eot|map|webp)$       0;
        /favicon.ico                              0;
        /robots.txt                               0;
        /healthz                                  0;
        /ping                                     0;
    }
    ...
}
```

Рис. 5.a – Блок map для умовного логування у секції http {}

Директива `map` зіставляє `$request_uri` з шаблонами зверху вниз. Перший збіг визначає значення `$loggable`. Префікс `~*` означає регулярний вираз без урахування регістру. Рядок `default 1` гарантує, що всі URI, що не відповідають жодному шаблону, будуть записуватись.

```
sudo nginx -t && sudo systemctl reload nginx
```

b. Застосувати умову `if=$loggable` у директивах `access_log` тестового сайту (рис. 5.b):

```
server {
    ...
    # Параметр if= відключає запис, якщо $loggable = 0
    access_log /var/log/nginx/logging-lab-access.log
               extended      if=$loggable;

    access_log /var/log/nginx/logging-lab-access-json.log
               json_combined if=$loggable;

    error_log /var/log/nginx/logging-lab-error.log warn;
    ...
}
```

Рис. 5.b – Застосування if=$loggable у директивах access_log

Параметр `if=` приймає змінну: якщо її значення `0` або порожній рядок — запит не записується до журналу. Це єдиний офіційно рекомендований спосіб умовного логування без використання `if {}` у location.

```
sudo nginx -t && sudo systemctl reload nginx
```

c. Перевірити умовне логування на практиці:

```
sudo truncate -s 0 /var/log/nginx/logging-lab-access.log
```

Команда `truncate -s 0` очищає файл без видалення — Nginx продовжує писати у той самий дескриптор.

```
curl -s http://logging-lab-xxx-yyy-zzz.local/ -o /dev/null
curl -s http://logging-lab-xxx-yyy-zzz.local/css/style.css -o
/dev/null
curl -s http://logging-lab-xxx-yyy-zzz.local/img/logo.png -o /dev/null
curl -s http://logging-lab-xxx-yyy-zzz.local/favicon.ico -o /dev/null
sudo cat /var/log/nginx/logging-lab-access.log
```

У журналі повинен бути лише один рядок — запит до `/`. Запити до `/css/style.css`, `/img/logo.png` та `/favicon.ico` не записуються.

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo wc -l /var/log/nginx/logging-lab-access.log
```

Зробити висновки за отриманими результатами.

d. Перевірити фільтрацію 5xx-помилок у журналі — налаштувати тимчасове повернення 500 для тестового URI:

Додати тимчасовий блок до конфігурації сайту (рис. 5.d):

```
location /test-error {
    # Тимчасово повертаємо 500 для тестування журналювання помилок
    return 500 "Internal Server Error Test";
}
```

Рис. 5.d – Тимчасовий location для генерації помилок 500

```
sudo nginx -t && sudo systemctl reload nginx
curl -s http://logging-lab-xxx-yyy-zzz.local/test-error -o /dev/null -
w "%{http_code}"
sudo grep " 5[0-9][0-9] " /var/log/nginx/logging-lab-access.log
```

Команда `grep ' 5[0-9][0-9] '` фільтрує рядки, де дев'яте поле (код відповіді) містить число від 500 до 599. URI `/test-error` не є статичним ресурсом, тому `$loggable = 1` і запит потрапляє до журналу.

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo grep " 5[0-9][0-9] " /var/log/nginx/logging-lab-access.log | wc -
l
```

Зробити висновки за отриманими результатами.

e. Видалити тимчасовий location /test-error після перевірки та перезавантажити Nginx:

```
sudo vi /etc/nginx/sites-available/logging-lab-xxx-yyy-zzz
sudo nginx -t && sudo systemctl reload nginx
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s http://logging-lab-xxx-yyy-zzz.local/test-error -o /dev/null -
w "%{http_code}"
```

Зробити висновки за отриманими результатами.

6. Ротація та буферизація журналів.

Без ротації журнали Nginx необмежено зростають і можуть заповнити файлову систему. Утиліта logrotate автоматично архівує та видаляє старі файли. Особливість Nginx: він тримає файловий дескриптор журналу відкритим — просте перейменування файлу не переключить запис. Саме тому у postrotate надсилається сигнал USR1: він повідомляє Nginx закрити старі та відкрити нові файли журналів без перезапуску. Буферизація запису зменшує кількість I/O-операцій при великому трафіку.

a. Переглянути стандартну конфігурацію ротації, що встановлюється разом з пакетом Nginx:

```
cat /etc/logrotate.d/nginx
```

Зверніть увагу на наявний блок `postrotate` — він надсилає сигнал `USR1` процесу Nginx для повторного відкриття файлів. Директива `sharedscripts` гарантує, що postrotate виконується лише один раз для всіх файлів журналів, а не окремо для кожного.

b. Замінити вміст `/etc/logrotate.d/nginx` розширеною конфігурацією з підтримкою журналів тестового сайту (рис. 6.b):

```
/var/log/nginx/*.log {
    daily                       # ротація щоденно
    missingok                   # не помилятися, якщо файл відсутній
    rotate 14                   # зберігати 14 архівів (2 тижні)
    compress                    # стискати архіви gzip
    delaycompress               # стискати попередній архів (не поточний)
    notifempty                  # не ротувати порожній файл
    create 0640 www-data adm    # права та власник нового файлу
    sharedscripts
    postrotate
        # Сигнал USR1: Nginx закриває старі та відкриває нові файли
        if [ -f /run/nginx.pid ]; then
            kill -USR1 $(cat /run/nginx.pid)
        fi
    endscript
}
```

Рис. 6.b – Розширена конфігурація logrotate для Nginx

Директива `delaycompress` залишає щойно ротований файл нестиснутим протягом однієї ротації — це дає можливість іншим утилітам (наприклад, `tail -f`) продовжувати читати журнал. Наступного циклу файл буде стиснуто.

```
sudo vi /etc/logrotate.d/nginx
```

c. Перевірити конфігурацію logrotate у тестовому режимі (без реального виконання):

```
sudo logrotate -d /etc/logrotate.d/nginx 2>&1 | grep -E
"(rotating|would|error)"
```

Опція `-d` (debug) виводить план дій без реального виконання. Рядки `rotating log ...` підтверджують, що logrotate знайшов файли журналів. Рядки зі словом `error` вказують на проблеми в конфігурації.

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo logrotate -d /etc/logrotate.d/nginx 2>&1 | grep "rotating"
```

Зробити висновки за отриманими результатами.

d. Налаштувати буферизацію запису у директивах `access_log` тестового сайту (рис. 6.d):

```
server {
    ...
    # buffer=32k: накопичувати до 32 КБ у пам'яті перед записом на диск
    # flush=5s: записувати не рідше ніж раз на 5 секунд
    access_log /var/log/nginx/logging-lab-access.log
               extended if=$loggable buffer=32k flush=5s;

    access_log /var/log/nginx/logging-lab-access-json.log
               json_combined if=$loggable buffer=32k flush=5s;

    error_log /var/log/nginx/logging-lab-error.log warn;
    ...
}
```

Рис. 6.d – Буферизація access_log з параметрами buffer та flush

При `buffer=32k` Nginx накопичує дані у пам'яті та виконує одну операцію запису замість окремого запису для кожного запиту. `flush=5s` гарантує, що дані не затримаються у буфері більше 5 секунд, навіть якщо трафік відсутній.

```
sudo nginx -t && sudo systemctl reload nginx
ab -n 100 -c 10 http://logging-lab-xxx-yyy-zzz.local/
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo tail -5 /var/log/nginx/logging-lab-access-json.log
```

Зробити висновки за отриманими результатами.

7. Аналіз журналів командним рядком.

Стандартні утиліти Linux — awk, sort, uniq, grep — дозволяють швидко аналізувати журнали навіть за відсутності графічного інтерфейсу. Утиліта awk обробляє текст рядок за рядком та звертається до полів через позиційні змінні ($1, $2, ...). Утиліта jq є потужним процесором JSON: підтримує фільтрацію, трансформацію та статистичну обробку. Вміння аналізувати журнали вручну є базовою навичкою адміністратора вебсерверів.

a. Згенерувати різноманітний тестовий трафік для аналізу:

```
ab -n 200 -c 10 http://logging-lab-xxx-yyy-zzz.local/
ab -n 50 -c 5 http://logging-lab-xxx-yyy-zzz.local/nonexistent
curl -s http://logging-lab-xxx-yyy-zzz.local/css/style.css -o
/dev/null
curl -s http://logging-lab-xxx-yyy-zzz.local/img/logo.png -o /dev/null
```

```
sudo tail -3 /var/log/nginx/logging-lab-access.log
```

Три серії запитів: 200 успішних до `/` (код 200), 50 до неіснуючої сторінки (404), запити до статичних ресурсів (не записуватимуться через `if=$loggable`).

b. Визначити топ IP-адрес за кількістю запитів:

```
sudo awk '{print $1}' /var/log/nginx/logging-lab-access.log | sort |
uniq -c | sort -rn | head -10
```

Поле `$1` у форматі `extended` — IP-адреса клієнта. `sort | uniq -c` підраховує унікальні значення. `sort -rn` сортує числово у спадному порядку.

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo awk '{print $1}' /var/log/nginx/logging-lab-access.log | sort |
uniq -c | sort -rn | head -5
```

Зробити висновки за отриманими результатами.

c. Визначити топ URI та розподіл HTTP-кодів відповідей:

```
sudo awk '{print $7}' /var/log/nginx/logging-lab-access.log | sort |
uniq -c | sort -rn | head -10
sudo awk '{print $9}' /var/log/nginx/logging-lab-access.log | sort |
uniq -c | sort -rn
```

Поле `$7` у форматі `extended` — URI запиту. Поле `$9` — HTTP-код відповіді. Розподіл кодів дозволяє виявити масові 4xx (помилки клієнта) або 5xx (помилки сервера).

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo awk '{print $9}' /var/log/nginx/logging-lab-access.log | sort |
uniq -c
```

Зробити висновки за отриманими результатами.

d. Відфільтрувати лише 5xx-помилки та визначити проблемні URI:

```
sudo grep " 5[0-9][0-9] " /var/log/nginx/logging-lab-access.log | awk
'{print $9, $7}' | sort | uniq -c | sort -rn
```

Шаблон `' 5[0-9][0-9] '` (з пробілами) запобігає хибним збігам у полях User-Agent або URI, що можуть містити три цифри. Якщо 5xx відсутні — результат порожній.

```
sudo grep " 5[0-9][0-9] " /var/log/nginx/logging-lab-access.log | wc -
l
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo grep " 4[0-9][0-9] " /var/log/nginx/logging-lab-access.log | awk
'{print $9, $7}' | sort | uniq -c
```

Зробити висновки за отриманими результатами.

e. Аналіз JSON-журналу через jq — статистика часу обробки запитів:

```
sudo cat /var/log/nginx/logging-lab-access-json.log | jq -r
'select(.request_time > 0.005) | [.remote_addr, .request_uri,
.request_time, .status] | @tsv'
```

Вираз `select(.request_time > 0.005)` фільтрує запити, що виконувались довше 5 мс. `@tsv` форматує масив як рядок з табуляцією.

```
sudo cat /var/log/nginx/logging-lab-access-json.log | jq -s
'[.[].request_time] | {count:length, min:min, max:max,
avg:(add/length)}'
```

Прапор `-s` (slurp) зчитує всі рядки JSON у масив. Функція `add/length` обчислює середнє значення `request_time` по всіх запитах.

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo cat /var/log/nginx/logging-lab-access-json.log | jq '.status' |
sort | uniq -c
```

Зробити висновки за отриманими результатами.

8. Розгортання стека Grafana Alloy + Loki + Grafana для централізованого збору та візуалізації журналів.

Grafana Alloy — уніфікований агент збору телеметрії від Grafana Labs, наступник Promtail та Grafana Agent (обидва deprecated). Alloy є сумісним з OpenTelemetry Protocol (OTLP) та нативним протоколом Loki. Loki — система агрегації журналів, що зберігає текст стиснутим у блоках та індексує лише мітки (labels) — це суттєво знижує витрати на зберігання порівняно з Elasticsearch. Grafana — графічний інтерфейс для запитів і візуалізації. Усі три компоненти розгортаються у Docker-контейнерах і доступні з хост-машини.

a. Створити робочу директорію для стека:

```
mkdir -p ~/logging-stack/alloy && cd ~/logging-stack
```

Усі конфігураційні файли стека зберігатимуться у цій директорії.

b. Переконатись, що Alloy матиме доступ до журналів Nginx. Перевірити права на директорію:

```
ls -la /var/log/nginx/
```

Файли журналів Nginx належать групі `adm`. Контейнер Alloy монтує директорію у режимі read-only — права на читання мають бути відкриті для всіх (`o+r`). Якщо права закриті — додати vagrant до групи adm:

```
sudo usermod -aG adm vagrant && newgrp adm
```

Перевірити застосовану конфігурацію, виконавши команду:

```
ls -la /var/log/nginx/logging-lab-access-json.log
```

Зробити висновки за отриманими результатами.

c. Створити конфігурацію Grafana Alloy. Зберегти у файл `~/logging-stack/alloy/config.alloy` (рис. 8.c):

```
// Alloy конфігурація для збору журналів Nginx та відправки до Loki
// 1. Вибір файлів для читання
local.file_match "nginx_json_logs" {
  path_targets = [{
    "__path__" = "/var/log/nginx/logging-lab-access-json.log",
    "job"      = "nginx",
    "vhost"    = "logging-lab",
  }]
}

// 2. Читання файлу (tail)
loki.source.file "nginx_access" {
  targets    = local.file_match.nginx_json_logs.targets
  forward_to = [loki.process.parse_nginx.receiver]
}
// 3. Парсинг JSON та витяг міток
loki.process "parse_nginx" {
  stage.json {
    expressions = {
      status         = "status",
      request_method = "request_method",
      host           = "host",
      request_time   = "request_time",
    }
  }
  stage.labels {
    values = {
      status         = "status",
      request_method = "request_method",
      host           = "host",
    }
  }
  forward_to = [loki.write.local.receiver]
}

// 4. Відправка до Loki
loki.write "local" {
  endpoint {
    url = "http://loki:3100/loki/api/v1/push"
  }
}
```

Рис. 8.c – Конфігурація Grafana Alloy — файл config.alloy

Компонент `loki.process` витягує значення JSON-полів через `stage.json`, а `stage.labels` перетворює їх на мітки Loki. Мітки `status`, `request_method` та `host` дозволяють фільтрувати журнали у Grafana без повнотекстового пошуку.

`vi ~/logging-stack/alloy/config.alloy`

d. Створити `~/logging-stack/docker-compose.yml` для запуску всіх компонентів стека (рис. 8.d):

```
services:

  loki:
    image: grafana/loki:3.0.0
    container_name: loki
    ports:
      - "3100:3100"
    command: -config.file=/etc/loki/local-config.yaml
    networks: [logging]

  alloy:
    image: grafana/alloy:v1.8.0
    container_name: alloy
    volumes:
      - ./alloy/config.alloy:/etc/alloy/config.alloy:ro
      - /var/log/nginx:/var/log/nginx:ro
    command: run /etc/alloy/config.alloy
    depends_on: [loki]
    networks: [logging]

  grafana:
    image: grafana/grafana:11.0.0
    container_name: grafana
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin123
      - GF_USERS_ALLOW_SIGN_UP=false
    depends_on: [loki]
    networks: [logging]

networks:
  logging:
    driver: bridge
```

Рис. 8.d – docker-compose.yml для стека Loki + Alloy + Grafana

Монтування `/var/log/nginx:/var/log/nginx:ro` надає Alloy доступ до директорії журналів Nginx у режимі read-only. Порт `3000` пробрасується на хост-машину — Grafana буде доступна за адресою `http://192.168.56.10:3000`.

`vi ~/logging-stack/docker-compose.yml`

e. Запустити стек та перевірити стан контейнерів:

```
cd ~/logging-stack && docker compose up -d
docker compose ps
```

Усі три контейнери повинні мати статус `running`. При першому запуску Docker завантажує образи з Docker Hub — це займає 2–5 хвилин. Якщо контейнер не запускається — переглянути журнали:

```
docker compose logs alloy --tail=30
```

Перевірити застосовану конфігурацію, виконавши команду:

```
docker compose ps --format "table {{.Name}}\t{{.Status}}"
```

Зробити висновки за отриманими результатами.

f. Перевірити готовність Loki та появу міток після генерації трафіку:

```
curl -s http://localhost:3100/ready
ab -n 50 -c 5 http://logging-lab-xxx-yyy-zzz.local/ && sleep 15
curl -s http://localhost:3100/loki/api/v1/labels | jq .data
```

Відповідь `ready` підтверджує ініціалізацію Loki. Після генерації трафіку та затримки 15 секунд (цикл flush Alloy) у списку міток мають з'явитись `"status"`, `"request_method"`, `"host"`.

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s http://localhost:3100/loki/api/v1/labels | jq .data
```

Зробити висновки за отриманими результатами.

g. Відкрити Grafana у браузері на хост-машині та підключити Loki як джерело даних:

У браузері відкрити: `http://192.168.56.10:3000`. Логін: `admin`, пароль: `admin123`.

Перейти до: **Connections → Data sources → Add data source → Loki**.

У полі **"URL"** ввести `http://loki:3100` (ім'я сервісу у Docker-мережі, не localhost). Натиснути **"Save & test"** — відповідь "Data source connected and labels found" підтверджує успішне підключення.

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s http://192.168.56.10:3000/api/health | jq .database
```

Зробити висновки за отриманими результатами.

h. Виконати LogQL-запити у розділі Grafana Explore:

Перейти до: **Explore** → вибрати джерело `"Loki"`. Виконати наступні запити послідовно:

```
# 1. Усі журнали тестового сайту
{job="nginx"}

# 2. Лише помилки 4xx та 5xx
{job="nginx"} | json | status >= 400

# 3. Лише POST-запити
{job="nginx", request_method="POST"}

# 4. Частота запитів за хвилину (метрика)
rate({job="nginx"}[1m])
```

Рис. 8.h – LogQL-запити для аналізу журналів Nginx у Grafana Explore

Оператор `| json` активує парсинг рядка журналу як JSON та робить всі поля доступними для фільтрації. Функція `rate()` будує графік частоти нових рядків журналу за часовий інтервал.

Перевірити застосовану конфігурацію, виконавши команду:

```
docker compose logs alloy --tail=10
```

Зробити висновки за отриманими результатами.

i. Зупинити стек після завершення роботи:

```
cd ~/logging-stack && docker compose down
```

Команда `docker compose down` зупиняє та видаляє контейнери, зберігаючи конфігураційні файли. Для видалення разом з накопиченими даними Loki додати прапор `-v`: `docker compose down -v`.

Перевірити застосовану конфігурацію, виконавши команду:

```
docker compose ps
```

Зробити висновки за отриманими результатами.

Контрольні запитання

1. Яку роль виконують директиви access_log та error_log у Nginx? У яких секціях конфігурації вони можуть бути визначені та як відбувається їх успадкування?
2. У чому полягає різниця між рівнями деталізації error_log: warn, error та debug? Коли доцільно використовувати кожен рівень?
3. Як відбувається визначення кастомного формату журналу через директиву log_format? Які змінні Nginx є найбільш інформативними для аналізу продуктивності вебсервера?
4. Яким чином параметр escape=json у директиві log_format впливає на вміст журналу та чому він є необхідним для коректного JSON-виводу?
5. Як реалізується умовне логування через директиву map та параметр if= у Nginx? Чому цей підхід є безпечнішим за використання if {} у location {}?
6. Яку роль виконує сигнал USR1 при ротації журналів Nginx? Що відбудеться, якщо перейменувати файл журналу без надсилання цього сигналу?
7. Які переваги надають параметри buffer= та flush= директиви access_log? Як вони впливають на надійність збереження даних при аварійному завершенні процесу?
8. Як відбувається збір журналів агентом Grafana Alloy? Яка роль компонентів loki.source.file, loki.process та loki.write у конфігурації?
9. У чому полягає відмінність підходу Loki до зберігання журналів від систем повнотекстового пошуку (Elasticsearch)? Які переваги та обмеження має індексування лише міток?

10. Яким чином мова запитів LogQL дозволяє фільтрувати журнали за мітками та вмістом рядків і будувати метрики? Навести приклад запиту для відображення лише помилок 5xx та приклад функції rate() для підрахунку частоти запитів.