Практична робота №8

МОНІТОРИНГ ТА АЛЕРТИНГ NGINX З ВИКОРИСТАННЯМ PROMETHEUS, GRAFANA ТА GRAFANA ALERTING

Мета заняття: набути практичних навичок розгортання системи моніторингу вебсервера Nginx з використанням Prometheus та Grafana; навчитися збирати метрики Nginx через nginx-prometheus-exporter, візуалізувати їх на дашбордах Grafana, налаштовувати правила алертингу Grafana Alerting та отримувати сповіщення електронною поштою через Postfix relay із SendGrid.

Завдання на роботу

1. Підготовка середовища.

Перед початком роботи необхідно переконатися, що інфраструктура з практичної роботи №7 (Grafana, Loki, Alloy) функціонує, та додати прокидання портів для нових сервісів — Prometheus та Grafana. Також необхідно встановити Postfix — поштовий сервер, який буде використовуватися як SMTP relay для відправки email-алертів.

a. Перевірити наявність проброшених портів у `Vagrantfile`. Переконатися, що порти 3000 (Grafana) та 9090 (Prometheus) проброшені на хост-машину. Якщо відповідні рядки відсутні — додати їх (рис. 1.a):

```
# Vagrantfile (додати всередині блоку config.vm)

# Grafana — UI моніторингу та дашборди
config.vm.network "forwarded_port", guest: 3000, host: 3000

# Prometheus — збір та зберігання метрик
config.vm.network "forwarded_port", guest: 9090, host: 9090
```

Рис. 1.a – Прокидання портів Grafana та Prometheus у Vagrantfile

Прокидання портів дозволяє звертатися до сервісів, що працюють у віртуальній машині, безпосередньо з браузера хост-машини. Порт 3000

використовується Grafana для веб-інтерфейсу, порт 9090 — Prometheus для веб-інтерфейсу та API запитів до бази метрик.

Якщо рядки були додані або змінені, виконати перезавантаження віртуальної машини для застосування нової конфігурації мережі:

```
vagrant reload
```

b. Підключитися до віртуальної машини та перевірити стан існуючого стеку логування з практичної роботи №7:

```
vagrant ssh
docker compose -f ~/logging-stack/docker-compose.yml ps
```

У виводі команди мають відображатися контейнери Grafana, Loki та Alloy зі статусом "Up". Якщо контейнери зупинені або не відображаються у списку, запустити стек:

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

Дочекатися повного запуску всіх контейнерів (статус має бути "Up" або "healthy"). Перевірити доступність Grafana у браузері хост-машини за адресою http://localhost:3000 — має відобразитися інтерфейс авторизації Grafana.

c. Встановити додаткові пакети, необхідні для відправки електронних листів через SMTP relay:

```
sudo apt update && sudo apt install -y postfix libsasl2-modules
mailutils
```

При встановленні `postfix` інсталятор запропонує обрати тип конфігурації у діалоговому вікні ncurses. Обрати варіант **"Internet Site"** — це означає, що Postfix буде відправляти пошту безпосередньо через мережу (у нашому випадку — через SendGrid relay). Інші варіанти ("Local only", "Satellite system") не підходять для нашого сценарію.

У полі **"System mail name"** ввести власний домен у форматі `username.pp.ua` (домен, який буде зареєстровано у завданні 3). Це значення визначає домен у полі "From:" вихідних листів, а також використовується як значення змінної `$mydomain` у конфігурації Postfix.

Пакет `libsasl2-modules` надає модулі SASL-автентифікації (Simple Authentication and Security Layer), необхідні для підключення Postfix до

SendGrid із використанням API Key. Без цього пакету Postfix не зможе автентифікуватися на SMTP-сервері SendGrid. Пакет `mailutils` надає утиліту `mail`, яка дозволяє відправляти тестові листи з командного рядка для перевірки коректності налаштування relay.

Перевірити застосовану конфігурацію, виконавши команду:

```
dpkg -l | grep -E "postfix|libsasl2-modules|mailutils"
```

Зробити висновки за отриманими результатами.

2. Створення тестового сайту для практичної роботи.

Для моніторингу необхідно мати діючий віртуальний хост Nginx з окремими журналами доступу та помилок. Тестовий сайт дозволить генерувати контрольований HTTP-трафік та спостерігати за зміною метрик у Prometheus та на дашбордах Grafana. Також у конфігурації буде увімкнено модуль stub_status — джерело базових метрик Nginx для prometheus-exporter.

a. Створити кореневу директорію сайту та необхідну структуру каталогів:

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

Замінити `xxx-yyy-zzz` на власні ідентифікатори: група-варіант-ініціали (наприклад, `monitoring-lab-301-05-ivanov`). Ця конвенція іменування дозволяє однозначно ідентифікувати роботу кожного студента.

b. Створити файл `index.html` з базовою розміткою сторінки (рис. 2.b):

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Monitoring Lab — xxx-yyy-zzz</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
    <header>
        <img src="/img/logo.png" alt="Logo" class="logo">
        <h1>Практична робота №8 — Моніторинг Nginx</h1>
    </header>
    <main>
        <p>Студент: xxx-yyy-zzz</p>
        <p>Тема: Prometheus + Grafana + Alerting</p>
    </main>
</body>
</html>
```

Рис. 2.b – Вміст файлу /var/www/monitoring-lab-xxx-yyy-zzz/index.html

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

c. Створити файл стилів `css/style.css` (рис. 2.c):

```
/* /var/www/monitoring-lab-xxx-yyy-zzz/css/style.css */
body {
    font-family: "Segoe UI", Arial, sans-serif;
    max-width: 800px;
    margin: 0 auto;
    padding: 20px;
    background: #f8f9fa;
}
header { text-align: center; margin-bottom: 30px; }
.logo { width: 120px; height: auto; margin-bottom: 10px; }
h1 { color: #2c3e50; }
main {
    background: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,.1);
}
```

Рис. 2.c – Вміст файлу css/style.css

```
sudo vi /var/www/monitoring-lab-xxx-yyy-zzz/css/style.css
```

d. Створити зображення-заглушку для логотипу:

```
sudo convert -size 120x120 xc:#3498db -fill white -gravity center -
pointsize 20 -annotate 0 "LOGO" /var/www/monitoring-lab-xxx-yyy-
zzz/img/logo.png 2>/dev/null || echo "placeholder" | sudo tee
/var/www/monitoring-lab-xxx-yyy-zzz/img/logo.png
```

Якщо утиліта `imagemagick` не встановлена, створити будь-яке PNG-зображення розміром 120×120 пікселів та скопіювати його у каталог `img/`. Зображення використовується як логотип на головній сторінці тестового сайту.

e. Створити конфігурацію віртуального хоста Nginx (рис. 2.e). Формати журналів `extended` та `json_combined` вже визначені у `nginx.conf` з практичної роботи №7:

```
# /etc/nginx/sites-available/monitoring-lab-xxx-yyy-zzz

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

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

    # Окремі журнали для моніторингу
    # extended — розширений формат з часом відповіді
```

```
    access_log /var/log/nginx/monitoring-lab-access.log extended;
    error_log  /var/log/nginx/monitoring-lab-error.log warn;

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

    # ── Сторінка статусу Nginx ──────────────────────
    # Джерело базових метрик для nginx-prometheus-exporter.
    # Модуль stub_status відображає:
    #   Active connections — поточна кількість з'єднань
    #   accepts/handled/requests — лічильники з'єднань
    #   Reading/Writing/Waiting — стани з'єднань
    location /nginx_status {
        stub_status;
        allow 127.0.0.1;     # доступ з localhost
        allow 172.16.0.0/12; # доступ з Docker-мережі
        deny all;            # блокувати решту
    }
}
```

Рис. 2.e – Конфігурація віртуального хоста з окремими журналами та stub_status

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

Директива `stub_status` вмикає внутрішній модуль Nginx `ngx_http_stub_status_module`, який відображає базову статистику сервера у текстовому форматі. Сторінка статусу містить: поточну кількість активних з'єднань (Active connections), загальну кількість прийнятих з'єднань (accepts), кількість оброблених з'єднань (handled), загальну кількість оброблених HTTP-запитів (requests), та розподіл з'єднань за станами (Reading, Writing, Waiting).

Доступ до `/nginx_status` обмежено директивами `allow` та `deny`: дозволено лише з localhost (127.0.0.1) та з Docker-мережі (172.16.0.0/12), щоб exporter, який працює у Docker-контейнері, мав змогу опитувати endpoint. Усі інші IP-адреси отримають помилку 403 Forbidden. Це важливо з точки зору безпеки — сторінка статусу може розкрити інформацію про навантаження сервера.

f. Активувати сайт, додати запис до /etc/hosts та перезавантажити Nginx:

```
sudo ln -sf /etc/nginx/sites-available/monitoring-lab-xxx-yyy-zzz
/etc/nginx/sites-enabled/
echo "127.0.0.1 monitoring-lab-xxx-yyy-zzz.local" | sudo tee -a
/etc/hosts
sudo nginx -t && sudo systemctl reload nginx
```

Команда `nginx -t` перевіряє синтаксис конфігураційних файлів без перезапуску сервера. Оператор `&&` забезпечує виконання `systemctl reload` лише у випадку успішної перевірки. Директива `reload` застосовує нову конфігурацію без переривання обробки поточних з'єднань (graceful reload).

Перевірити застосовану конфігурацію, виконавши команду:

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

Зробити висновки за отриманими результатами.

Перевірити також доступність сторінки статусу Nginx:

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s http://monitoring-lab-xxx-yyy-zzz.local/nginx_status
```

Зробити висновки за отриманими результатами.

3. Реєстрація домену та налаштування DNS для автентифікації відправника.

Для надійної доставки алертів електронною поштою необхідно мати власний домен з коректно налаштованими DNS-записами автентифікації (SPF, DKIM, DMARC). Без автентифікації відправника поштові сервери (Gmail, Outlook, корпоративні сервери) з високою ймовірністю класифікують листи як спам або повністю відхилять їх. У цьому завданні буде зареєстровано безкоштовний домен та налаштовано всі необхідні DNS-записи.

a. **Реєстрація домену username.pp.ua на NIC.UA.**

Відкрити https://nic.ua у браузері хост-машини. Зареєструватися або увійти в особистий кабінет. У рядку пошуку доменів ввести бажане ім'я у форматі username.pp.ua (замінити username на власне ім'я або псевдонім латиницею). Домени у зоні .pp.ua є безкоштовними для фізичних осіб — обрати варіант безкоштовної реєстрації та заповнити необхідні контактні дані.

Після успішної реєстрації домен з'явиться у списку "Мої домени" в особистому кабінеті. Перейти до управління DNS-зонами домену — саме тут будуть додані записи для автентифікації відправника.

Домен необхідний для автентифікації відправника листів. Сучасні поштові системи перевіряють три механізми автентифікації: SPF (Sender Policy Framework) — визначає, які сервери мають право відправляти пошту від імені домену; DKIM (DomainKeys Identified Mail) — цифровий підпис листа, який підтверджує його цілісність та автентичність; DMARC (Domain-based Message Authentication, Reporting and Conformance) — політика обробки листів, що не пройшли перевірку SPF або DKIM.

b. **Реєстрація у SendGrid та отримання API Key.**

Відкрити https://sendgrid.com та натиснути "Sign Up". Безкоштовний план (Free tier) дозволяє відправляти до 100 листів на день без введення даних кредитної картки — цього цілком достатньо для навчальних алертів. Верифікувати обліковий запис через електронну пошту.

Після верифікації перейти до розділу Settings → API Keys → Create API Key. Налаштувати ключ наступним чином: назва — "Postfix Relay"; тип доступу — "Restricted Access"; дозвіл — Mail Send → Full Access. Натиснути "Create & View" та скопіювати API Key (рядок, що починається з SG.).

> **ВАЖЛИВА ПРИМІТКА:** API Key показується лише один раз — зберегти його у надійному місці (наприклад, у менеджері паролів або текстовому файлі з обмеженими правами доступу). Якщо ключ буде втрачено, доведеться створити новий.

SendGrid використовує API Key замість звичайного пароля для SMTP-автентифікації. При підключенні через SMTP-протокол login — завжди літеральний рядок `"apikey"` (не адреса електронної пошти облікового запису SendGrid), а пароль — згенерований API Key. Обмежений

доступ (Restricted Access) з дозволом лише Mail Send мінімізує потенційну шкоду у разі компрометації ключа.

c. **Верифікація домену у SendGrid та отримання DNS-записів.**

Перейти до SendGrid → Settings → Sender Authentication → Authenticate Your Domain → Get Started. На першому кроці обрати DNS host: "Other Host (Not Listed)". На другому кроці ввести домен: username.pp.ua та натиснути Next.

SendGrid згенерує три CNAME-записи, які необхідно додати до DNS-зони домену на NIC.UA (рис. 3.c):

```
# DNS-записи, які згенерує SendGrid (приклад значень):

# 1. CNAME для трекінгу відправки (click/open tracking)
em1234.username.pp.ua  CNAME  u1234567.wl001.sendgrid.net

# 2. DKIM-підпис — ключ 1 (основний)
s1._domainkey.username.pp.ua  CNAME
s1.domainkey.u1234567.wl001.sendgrid.net

# 3. DKIM-підпис — ключ 2 (резервний, для ротації)
s2._domainkey.username.pp.ua  CNAME
s2.domainkey.u1234567.wl001.sendgrid.net
```

Рис. 3.c – Приклад DNS-записів для автентифікації домену у SendGrid

SendGrid використовує CNAME-записи замість класичних TXT-записів для DKIM. Перевага такого підходу: SendGrid може автоматично ротувати криптографічні ключі підпису без участі власника домену — при зміні ключа оновлюється лише цільовий запис на серверах SendGrid, а CNAME-делегування залишається незмінним. SPF також налаштовується автоматично через CNAME-делегування — SendGrid додає IP-адреси своїх серверів до SPF-запису.

d. **Додавання DNS-записів у панелі NIC.UA.**

Перейти до управління DNS-зоною username.pp.ua на NIC.UA. Додати три CNAME-записи відповідно до значень, отриманих від SendGrid у підзавданні (c). Конкретні значення будуть індивідуальними для

кожного студента — скопіювати їх зі сторінки SendGrid. Також додати запис DMARC (рис. 3.d):

```
# DMARC-запис для DNS-зони домену
# Тип: TXT
# Ім'я: _dmarc
# Значення:
v=DMARC1; p=none; rua=mailto:username@student.ztu.edu.ua

# Призначення полів:
#   v=DMARC1        — версія протоколу DMARC
#   p=none          — політика: лише збір звітів
#                     без блокування (безпечно для
#                     початку, не впливає на доставку)
#   rua=mailto:...  — адреса для отримання агрегованих
#                     XML-звітів про результати перевірки
#                     SPF/DKIM (1 раз на добу)
```

Рис. 3.d – DMARC-запис для DNS-зони домену

DNS-записи поширюються протягом 5–30 хвилин залежно від TTL (Time To Live) зони. Перевірити поширення записів за допомогою утиліти dig:

```
dig CNAME s1._domainkey.username.pp.ua
dig TXT _dmarc.username.pp.ua
```

CNAME для DKIM делегує підпис листів до SendGrid — при відправці листа SendGrid додає цифровий підпис DKIM від імені `username.pp.ua`. Поштовий сервер отримувача перевіряє підпис, запитуючи публічний ключ через DNS-запит до `s1._domainkey.username.pp.ua`. Якщо CNAME коректно налаштований, DNS повертає ключ з серверів SendGrid, і підпис верифікується.

DMARC визначає політику обробки листів, що не пройшли перевірку SPF або DKIM. Значення `p=none` означає лише збір статистики без блокування — поштові сервери будуть доставляти листи навіть при помилках автентифікації, але надсилатимуть звіти на вказану email-адресу. Це безпечний варіант для початкового розгортання. У виробничому середовищі рекомендується поступово переходити до `p=quarantine` (карантин) та `p=reject` (відхилення).

e. **Верифікація домену у SendGrid.**

Повернутися до SendGrid → Sender Authentication. Натиснути "Verify" поруч з username.pp.ua. SendGrid виконає DNS-запити для перевірки наявності CNAME-записів. Дочекатися статусу "Verified" для всіх трьох записів. Якщо верифікація не проходить — зачекати 15–30 хвилин для поширення DNS та повторити.

Після верифікації домену додати Single Sender: перейти до Settings → Sender Authentication → Verify a Single Address. Ввести адресу alerts@username.pp.ua як адресу відправника алертів. SendGrid надішле лист для підтвердження — перейти за посиланням у листі.

Перевірити застосовану конфігурацію, виконавши команду:

```
dig CNAME s1._domainkey.username.pp.ua +short
```

Зробити висновки за отриманими результатами.

Також перевірити другий DKIM-ключ та DMARC-запис:

```
dig CNAME s2._domainkey.username.pp.ua +short
dig TXT _dmarc.username.pp.ua +short
```

4. Налаштування Postfix як SMTP relay через SendGrid.

Postfix — це Mail Transfer Agent (MTA), який виступатиме як локальний SMTP-сервер. У нашій архітектурі Postfix виконує роль relay — приймає листи від локальних програм (зокрема Grafana) та перенаправляє їх через SendGrid до кінцевого отримувача. Така архітектура дозволяє централізувати відправку пошти, забезпечити автентифікацію через DKIM/SPF та не розголошувати SendGrid API Key окремим сервісам.

a. Налаштувати основний конфігураційний файл Postfix. Відкрити `/etc/postfix/main.cf` та додати або змінити параметри (рис. 4.a):

```
# /etc/postfix/main.cf — параметри relay через SendGrid

# ── Основні параметри ──────────────────────────────────────
# myhostname — FQDN поштового сервера
myhostname = username.pp.ua

# mydomain — домен, від імені якого відправляється пошта
mydomain = username.pp.ua

# myorigin — домен, який підставляється у поле From:
```

```
# якщо адреса не містить домену
myorigin = $mydomain

# ── Relay через SendGrid ──────────────────────────────────
# relayhost — всю пошту перенаправляти через цей сервер
# Квадратні дужки вимикають MX-lookup — Postfix звертається
# напряму до smtp.sendgrid.net на порту 587 (submission)
relayhost = [smtp.sendgrid.net]:587

# ── SASL автентифікація ───────────────────────────────────
# Увімкнути автентифікацію клієнта при з'єднанні з relay
smtp_sasl_auth_enable = yes

# Файл із парами login:password для кожного relay-хоста
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd

# Заборонити анонімну автентифікацію — вимагати логін
smtp_sasl_security_options = noanonymous

# ── TLS шифрування ────────────────────────────────────────
# Вимагати шифрування з'єднання (SendGrid не приймає
# нешифрований трафік на порту 587)
smtp_tls_security_level = encrypt
smtp_tls_CAfile = /etc/ssl/certs/ca-certificates.crt
smtp_use_tls = yes
```

Рис. 4.a – Параметри relay у /etc/postfix/main.cf

`sudo vi /etc/postfix/main.cf`

Параметр `relayhost` у квадратних дужках `[smtp.sendgrid.net]:587` вказує Postfix не виконувати MX-запит (Mail Exchanger lookup) для визначення поштового сервера домену отримувача, а звертатися напряму до вказаного хоста. Без квадратних дужок Postfix спочатку запитує MX-записи домену `sendgrid.net`, що може призвести до підключення до іншого сервера. Порт 587 (submission) є стандартним для відправки пошти з автентифікацією згідно з RFC 6409.

Параметр `smtp_tls_security_level = encrypt` вимагає обов'язкового шифрування TLS при з'єднанні з relay-хостом. SendGrid не приймає нешифровані з'єднання на порту 587. Файл `ca-certificates.crt` містить кореневі сертифікати CA для верифікації TLS-сертифіката SendGrid.

b. Створити файл автентифікації SASL (рис. 4.b). Файл містить облікові дані для підключення Postfix до SendGrid через SMTP-протокол:

```
# /etc/postfix/sasl_passwd
# Формат: [relay_host]:port  login:password
[smtp.sendgrid.net]:587  apikey:SG.xxxxxxxxxxxxxxxxx

# УВАГА: замінити SG.xxxxxxxxxxxxxxxxx на реальний
# API Key, отриманий у SendGrid (завдання 3.b)

# Рядок "apikey" є літеральним — це вимога SendGrid
# для SMTP SASL автентифікації. Не замінювати на email.
```

Рис. 4.b – Файл автентифікації /etc/postfix/sasl_passwd

```
sudo vi /etc/postfix/sasl_passwd
```

Скомпілювати текстовий файл у бінарний формат, захистити від читання та перезапустити Postfix:

```
sudo postmap /etc/postfix/sasl_passwd
sudo chmod 600 /etc/postfix/sasl_passwd /etc/postfix/sasl_passwd.db
sudo systemctl restart postfix
```

Команда `postmap` компілює текстовий файл у формат Berkeley DB (файл з розширенням `.db`) — бінарний формат для швидкого ключ-значення пошуку. Postfix при з'єднанні з relay-хостом шукає відповідний запис у `.db`-файлі та використовує знайдені login:password для SASL-автентифікації. Права доступу `600` (читання та запис лише для root) захищають API Key від читання іншими користувачами системи.

c. Відправити тестовий лист та перевірити заголовки автентифікації:

```
# Відправка тестового листа через ланцюг:
# mail → Postfix → SendGrid SMTP → email отримувача
echo "Test alert from Postfix via SendGrid" | \
  mail -s "Test Grafana Alert" \
  -a "From: alerts@username.pp.ua" \
  username@student.ztu.edu.ua
```

Рис. 4.c – Команда відправки тестового листа через Postfix relay

Перевірити, що лист дійшов та не потрапив у папку "Спам". Відкрити лист → натиснути "Показати оригінал" (Show original) у поштовому клієнті (Gmail: три крапки → "Show original"; Outlook: File → Properties). Переконатися у наявності заголовків автентифікації (рис. 4.c2):

```
# Очікувані заголовки у тестовому листі:

# Результати автентифікації — всі три перевірки пройдені
Authentication-Results: dkim=pass spf=pass dmarc=pass

# DKIM-підпис, доданий SendGrid від імені домену
DKIM-Signature: v=1; a=rsa-sha256; d=username.pp.ua; ...

# Заголовок SendGrid — підтверджує проходження через relay
X-SG-EID: ... (унікальний ідентифікатор SendGrid)
```

Рис. 4.c2 – Очікувані заголовки автентифікації у тестовому листі

Наявність `dkim=pass` підтверджує, що CNAME-записи для DKIM коректно делегують підпис до SendGrid. Наявність `spf=pass` підтверджує, що SendGrid авторизований відправляти пошту від імені домену. Наявність `dmarc=pass` підтверджує, що обидві перевірки (SPF та DKIM) пройшли, і лист відповідає політиці DMARC.

Якщо `dkim=fail` — перевірити поширення CNAME-записів командою `dig CNAME s1._domainkey.username.pp.ua`. Якщо записи відсутні — зачекати 15–30 хвилин та повторити перевірку. Якщо `spf=fail` — перевірити, що CNAME для трекінгу (em1234) налаштований коректно.

d. Перевірити активність відправки у панелі SendGrid:

Відкрити https://app.sendgrid.com → Activity. Знайти тестовий лист у списку подій — переглянути статус (Delivered означає успішну доставку) та деталі: адреса отримувача, час відправки, результат доставки, IP-адреса відправника.

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo tail -20 /var/log/mail.log | grep -i "status=sent"
```

Зробити висновки за отриманими результатами.

5. Розгортання Prometheus та nginx-prometheus-exporter.

Prometheus — відкрита система моніторингу з вбудованою часовою базою даних (time-series database), розроблена компанією SoundCloud та підтримувана Cloud Native Computing Foundation (CNCF). Prometheus

використовує pull-модель збору метрик: сервер періодично опитує (scrape) цільові HTTP-endpoints, які повертають метрики у стандартизованому текстовому форматі. `nginx-prometheus-exporter` — офіційний exporter від F5/NGINX Inc., який перетворює дані зі сторінки `stub_status` Nginx у формат метрик Prometheus.

a. Створити конфігураційний файл Prometheus (рис. 5.a). Розмістити його у директорії `~/logging-stack/` поруч з існуючим `docker-compose.yml`:

```
# ~/logging-stack/prometheus.yml
# Конфігурація збору метрик Prometheus

global:
  # Інтервал опитування targets (за замовч. 1m)
  scrape_interval: 15s

  # Інтервал обчислення recording/alerting rules
  evaluation_interval: 15s

  # Таймаут одного scrape-запиту
  scrape_timeout: 10s

scrape_configs:
  # ── Метрики самого Prometheus (self-monitoring) ─────────
  - job_name: "prometheus"
    static_configs:
      - targets: ["localhost:9090"]
        labels:
          instance: "prometheus-server"

  # ── Метрики Nginx через exporter ────────────────────────
  - job_name: "nginx"
    static_configs:
      - targets: ["nginx-exporter:9113"]
        labels:
          instance: "nginx-main"
    # Додаткова мітка для фільтрації у Grafana
    metric_relabel_configs:
      - target_label: environment
        replacement: "lab"
```

Рис. 5.a – Конфігурація Prometheus — ~/logging-stack/prometheus.yml

```
vi ~/logging-stack/prometheus.yml
```

Параметр `scrape_interval: 15s` визначає частоту опитування кожного target. Кожні 15 секунд Prometheus надсилає HTTP GET-запит до endpoint `/metrics` кожного target і зберігає отримані метрики з часовою міткою у своїй TSDB (Time Series Database). Для навчального середовища 15 секунд є оптимальним — це забезпечує достатню

деталізацію графіків. У виробничому середовищі типове значення складає 15–60 секунд залежно від кількості target та обсягу метрик.

Секція `scrape_configs` містить список цілей (targets) для збору метрик. Кожен `job_name` групує метрики від одного типу сервісів. Target `nginx-exporter:9113` вказує на контейнер exporter по імені Docker-сервісу — ім'я `nginx-exporter` відповідає назві сервісу у docker-compose.yml. Порт 9113 є стандартним для nginx-prometheus-exporter.

b. Розширити існуючий `~/logging-stack/docker-compose.yml`, додавши сервіси Prometheus та nginx-prometheus-exporter до секції `services` (рис. 5.b):

```
# ── ~/logging-stack/docker-compose.yml ──────────────────
# Додати до секції services: (після існуючих сервісів)

# ── Prometheus — збір та зберігання метрик ──────────────
prometheus:
  image: prom/prometheus:v2.53.0
  container_name: prometheus
  restart: unless-stopped
  ports:
    - "9090:9090"        # веб-інтерфейс та API
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml:ro
    - prometheus_data:/prometheus  # постійне сховище
  command:
    - "--config.file=/etc/prometheus/prometheus.yml"
    - "--storage.tsdb.path=/prometheus"
    - "--storage.tsdb.retention.time=7d"
    - "--web.enable-lifecycle"
  networks:
    - logging

# ── nginx-prometheus-exporter — збір метрик Nginx ──────
nginx-exporter:
  image: nginx/nginx-prometheus-exporter:1.3.0
  container_name: nginx-exporter
  restart: unless-stopped
  command:
    - "--nginx.scrape-uri=http://host.docker.internal:80/nginx_status"
  ports:
    - "9113:9113"        # endpoint метрик
  extra_hosts:
    - "host.docker.internal:host-gateway"
  networks:
    - logging
```

Рис. 5.b – Сервіси Prometheus та nginx-exporter у docker-compose.yml

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

Також додати том `prometheus_data` до секції `volumes` у кінці файлу (рис. 5.b2):

```
# Додати до секції volumes: (якщо вже є — додати рядок)
volumes:
  prometheus_data:      # постійне сховище метрик Prometheus
  # ... інші існуючі томи з ПР №7 ...
```

Рис. 5.b2 – Визначення тому для даних Prometheus

Параметр `--storage.tsdb.retention.time=7d` обмежує зберігання метрик 7 днями — це достатньо для навчального середовища та запобігає надмірному використанню диску. Параметр `--web.enable-lifecycle` дозволяє перезавантажувати конфігурацію Prometheus через HTTP POST-запит до `/-/reload` без перезапуску контейнера — корисно при додаванні нових targets.

Exporter підключається до Nginx через `host.docker.internal` — спеціальне DNS-ім'я, що резолвиться у IP-адресу хоста Docker (у даному випадку — VM, де працює Nginx). Директива `extra_hosts` з `host-gateway` додає запис у `/etc/hosts` контейнера для резолвінгу цього імені. Це необхідно, оскільки Nginx працює не в Docker-контейнері, а безпосередньо на хост-машині.

c. Запустити нові сервіси та перевірити їх стан:

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

Дочекатися, поки всі контейнери отримають статус "Up". Перевірити, що exporter успішно підключився до Nginx та повертає метрики:

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s http://localhost:9113/metrics | head -20
```

Зробити висновки за отриманими результатами.

У виводі мають бути присутні рядки з метриками Nginx: `nginx_connections_active`, `nginx_connections_accepted`,

`nginx_http_requests_total` тощо. Кожна метрика має формат `metric_name{labels} value` — стандартний exposition format Prometheus.

d. Перевірити роботу Prometheus через веб-інтерфейс:

Відкрити http://localhost:9090 у браузері хост-машини. Перейти до Status → Targets — обидві цілі (prometheus та nginx) мають мати статус "UP" із зеленим індикатором. Статус "DOWN" означає, що Prometheus не може підключитися до target — перевірити мережеву доступність та конфігурацію.

Виконати тестовий запит у Prometheus. Перейти до вкладки Graph → у поле "Expression" ввести запит:

```
nginx_http_requests_total
```

Натиснути "Execute" — у таблиці має відобразитися поточне значення лічильника HTTP-запитів Nginx. Згенерувати декілька запитів для створення навантаження та перевірки зміни лічильника:

```
for i in $(seq 1 50); do curl -s -o /dev/null http://monitoring-lab-xxx-yyy-zzz.local; done
```

Зачекати 15 секунд (один цикл scrape) та повторити запит — значення лічильника має збільшитися на ~50.

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s "http://localhost:9090/api/v1/query?query=nginx_http_requests_total" | jq .data.result[0].value[1]
```

Зробити висновки за отриманими результатами.

6. Підключення Prometheus до Grafana та створення дашборду моніторингу.

Grafana надає зручний інтерфейс для візуалізації метрик, що зберігаються у Prometheus. У цьому завданні необхідно підключити Prometheus як джерело даних (datasource) та створити дашборд з ключовими метриками Nginx: швидкість запитів, кількість активних з'єднань, розподіл станів з'єднань та порівняння прийнятих і оброблених з'єднань.

a. **Додавання Prometheus як datasource у Grafana.**

Відкрити Grafana за адресою http://localhost:3000 (логін/пароль за замовчуванням: admin/admin, якщо не змінювався у ПР №7). Перейти до Connections → Data sources → Add data source. У списку типів обрати "Prometheus".

Налаштувати параметри підключення (рис. 6.a):

```
# Налаштування Prometheus datasource у Grafana:

# Name: Prometheus (за замовчуванням)

# Prometheus server URL:
# Контейнер Grafana звертається до Prometheus
# по імені Docker-сервісу через внутрішню мережу
http://prometheus:9090

# Scrape interval: 15s
# (має відповідати scrape_interval у prometheus.yml)

# HTTP Method: GET (за замовчуванням)
# Timeout: 60s
```

Рис. 6.a – Параметри підключення Prometheus до Grafana

URL `http://prometheus:9090` використовує ім'я Docker-сервісу замість IP-адреси. Контейнери Grafana та Prometheus знаходяться в одній Docker-мережі `logging`, тому DNS-резолвер Docker автоматично перетворює ім'я сервісу у IP-адресу відповідного контейнера. Це надійніше за використання IP-адрес, які можуть змінюватися при перестворенні контейнерів.

Натиснути "Save & test" — має з'явитися зелене повідомлення "Successfully queried the Prometheus API." Якщо з'являється помилка — перевірити, що контейнер Prometheus працює та знаходиться у мережі logging.

b. **Створення дашборду моніторингу Nginx.**

Перейти до Dashboards → New → New Dashboard → Add visualization. Обрати datasource "Prometheus". Створити чотири панелі для комплексного моніторингу Nginx.

Панель 1: Active Connections (Stat). Створити панель типу "Stat" для відображення поточної кількості активних з'єднань (рис. 6.b1):

```
# PromQL запит — поточна кількість активних з'єднань Nginx
# Це gauge-метрика: відображає миттєве значення
nginx_connections_active

# Налаштування панелі (Panel options):
#   Title: Active Connections
#   Description: Поточна кількість активних з'єднань Nginx

# Visualization: Stat
#   Stat styles → Color mode: Background Gradient
#   Thresholds:
#     Base: green (0)
#     50: yellow (попередження)
#     100: red (критично)
```

Рис. 6.b1 – Запит PromQL для панелі Active Connections

Панель 2: HTTP Requests per Second (Time series). Додати нову панель типу "Time series" для відображення динаміки HTTP-запитів (рис. 6.b2):

```
# PromQL запит — швидкість HTTP-запитів (запитів/сек)
#
# rate() — функція PromQL, яка обчислює середню швидкість
# зміни лічильника (counter) за вказаний інтервал.
# nginx_http_requests_total — це counter, який монотонно
# зростає. rate() перетворює його у швидкість (req/s).
rate(nginx_http_requests_total[1m])

# Налаштування панелі:
#   Title: HTTP Requests per Second
#   Visualization: Time series
#   Standard options → Unit: reqps (requests/sec)
#   Graph styles → Fill opacity: 20
#   Graph styles → Line width: 2
#   Graph styles → Gradient mode: Scheme
```

Рис. 6.b2 – Запит PromQL для панелі HTTP Requests per Second

Функція `rate()` є ключовою для роботи з counter-метриками у PromQL. Counter — це лічильник, який лише збільшується (наприклад, загальна кількість HTTP-запитів). Безпосередньо значення counter не є інформативним — потрібна саме швидкість його зміни. `rate(metric[1m])` обчислює середнє значення за останню хвилину, згладжуючи короткочасні сплески.

Панель 3: Connection States (Time series). Додати панель для відображення розподілу з'єднань за станами (рис. 6.b3):

```
# PromQL запити — стани з'єднань Nginx
# Додати три окремі запити (Query A, B, C):

# Query A — з'єднання у стані Reading
# (Nginx зчитує заголовок запиту від клієнта)
nginx_connections_reading

# Query B — з'єднання у стані Writing
# (Nginx відправляє відповідь клієнту)
nginx_connections_writing

# Query C — з'єднання у стані Waiting
# (keep-alive з'єднання, очікує новий запит)
nginx_connections_waiting

# Налаштування панелі:
#   Title: Connection States
#   Visualization: Time series
#   Legend: Table, показати назви метрик
#   Graph styles → Stack series: Normal
#   Graph styles → Fill opacity: 80
```

Рис. 6.b3 – Запити PromQL для панелі Connection States

Панель 4: Accepted vs Handled Connections (Stat). Додати порівняльну панель для виявлення відхилених з'єднань (рис. 6.b4):

```
# PromQL запити для порівняння:

# Query A — загальна кількість прийнятих з'єднань
nginx_connections_accepted

# Query B — загальна кількість оброблених з'єднань
nginx_connections_handled

# Налаштування панелі:
#   Title: Accepted vs Handled Connections
#   Visualization: Stat
#   Показати обидва значення поруч
#
# Інтерпретація:
#   Accepted = Handled — норма (всі з'єднання оброблені)
#   Accepted > Handled — Nginx відхиляє з'єднання
#   (перевантаження або вичерпання worker_connections)
```

Рис. 6.b4 – Запити PromQL для панелі Accepted vs Handled Connections

c. Зберегти дашборд та згенерувати тестове навантаження для перевірки візуалізації:

Натиснути іконку збереження (дискета) у верхньому правому куті. Назвати дашборд "Nginx Monitoring — xxx-yyy-zzz". Обрати папку "General". Натиснути "Save".

Згенерувати навантаження для заповнення графіків реальними даними (рис. 6.c):

```
# Генерація тестового навантаження (200 запитів з паузою)
for i in $(seq 1 200); do
    curl -s -o /dev/null http://monitoring-lab-xxx-yyy-zzz.local
    sleep 0.1
done


# Генерація помилок 404 для різноманітності метрик
for i in $(seq 1 20); do
    curl -s -o /dev/null http://monitoring-lab-xxx-yyy-
zzz.local/nonexistent-page
done
```

Рис. 6.c – Скрипт генерації тестового навантаження

Після виконання скрипту зачекати 30–60 секунд (два-три цикли scrape Prometheus) та оновити дашборд у Grafana (натиснути іконку оновлення або клавішу "r"). Панелі мають відобразити зміни метрик: сплеск на графіку "HTTP Requests per Second", зміну розподілу станів з'єднань, оновлення лічильників.

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s
"http://localhost:9090/api/v1/query?query=rate(nginx_http_requests_tot
al[1m])" | jq .data.result[0].value[1]
```

Зробити висновки за отриманими результатами.

7. Налаштування Grafana Alerting для сповіщень електронною поштою.

Grafana Alerting — вбудована система алертингу, що дозволяє визначати правила на основі запитів до джерел даних та надсилати сповіщення через різні канали (email, Slack, Telegram, PagerDuty тощо). Система алертингу складається з трьох ключових компонентів: Alert Rules (правила, що визначають умови спрацювання), Contact Points (канали відправки сповіщень), та Notification Policies (правила маршрутизації алертів до contact points).

а. **Налаштування SMTP у Grafana.** Для відправки email-сповіщень необхідно налаштувати параметри SMTP-з'єднання. Оскільки Grafana працює у Docker-контейнері, налаштування передаються через змінні оточення. Додати змінні до сервісу `grafana` у `docker-compose.yml` (рис. 7.а):

```
# ~/logging-stack/docker-compose.yml
# Додати/доповнити змінні оточення сервісу grafana:
grafana:
  # ... існуючі параметри залишаються без змін ...
  environment:
    # ... існуючі змінні (якщо є) ...

    # ── SMTP для алертів через Postfix relay ──────────
    # Увімкнути відправку email
    - GF_SMTP_ENABLED=true

    # Підключення до Postfix на хост-машині (порт 25)
    - GF_SMTP_HOST=host.docker.internal:25

    # Адреса відправника алертних листів
    - GF_SMTP_FROM_ADDRESS=alerts@username.pp.ua
    - GF_SMTP_FROM_NAME=Grafana Alerts

    # Вимкнути перевірку TLS для локального з'єднання
    - GF_SMTP_SKIP_VERIFY=true
  extra_hosts:
    - "host.docker.internal:host-gateway"
```

Рис. 7.а – Змінні оточення SMTP для сервісу Grafana

Grafana підключається до Postfix на хост-машині через `host.docker.internal:25` — стандартний порт SMTP для локальної доставки. Параметр `GF_SMTP_SKIP_VERIFY=true` вимикає перевірку TLS-сертифікату для з'єднання з Postfix — це безпечно, оскільки з'єднання відбувається всередині VM між контейнером та хостом, а шифрування забезпечується на наступній ділянці: Postfix → SendGrid (порт 587 з TLS).

Важливо: Grafana не автентифікується на Postfix — вона просто передає лист як локальний клієнт. Postfix приймає лист без автентифікації від довірених мереж (параметр `mynetworks`). Далі Postfix додає SASL-автентифікацію та передає лист на SendGrid.

Для того, щоб Postfix приймав з'єднання з Docker-мережі, необхідно налаштувати параметри мережі у `/etc/postfix/main.cf` (рис. 7.a2):

```
# /etc/postfix/main.cf — додати для прийому з Docker-мережі

# Дозволити з'єднання без автентифікації з довірених мереж:
# 127.0.0.0/8     — localhost (локальні програми)
# 172.16.0.0/12   — Docker bridge-мережі
mynetworks = 127.0.0.0/8 172.16.0.0/12

# Прослуховувати на всіх мережевих інтерфейсах
# (за замовч. Postfix слухає лише на loopback 127.0.0.1)
inet_interfaces = all
```

Рис. 7.a2 – Параметри Postfix для прийому з Docker-мережі

```
sudo vi /etc/postfix/main.cf
sudo systemctl restart postfix
```

Перезапустити стек Docker для застосування нових змінних оточення Grafana:

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

b. **Створення Contact Point для email-сповіщень.**

У Grafana перейти до Alerting → Contact points → Add contact point. Налаштувати параметри каналу сповіщень (рис. 7.b):

```
# Параметри Contact Point:

# Name: Email Alerts
# Integration: Email

# Addresses (адреса отримувача алертів):
username@student.ztu.edu.ua

# Subject (шаблон теми листа — залишити за замовч.):
{{ template "default.title" . }}

# Optional settings:
#   Single email: увімкнути
#     (один лист для всіх алертів замість окремих)
#   Message: залишити за замовчуванням (HTML-шаблон)
```

Рис. 7.b – Параметри Contact Point для email-сповіщень

Натиснути "Test" для відправки тестового сповіщення. Перевірити, що тестовий лист надійшов на пошту студента (тема буде "[FIRING:1] (TestAlert Grafana)"). Натиснути "Save contact point".

c. **Налаштування Notification Policy.**

Перейти до Alerting → Notification policies. Змінити Default policy — натиснути "..." → Edit. Обрати "Email Alerts" як Default contact point. У полі "Group wait" залишити 30s, "Group interval" — 5m, "Repeat interval" — 4h. Зберегти зміни.

Notification policy визначає маршрутизацію алертів до contact points. Default policy застосовується до всіх алертів, які не потрапляють під жодне специфічне правило маршрутизації. Параметр `Group wait` визначає час очікування перед відправкою першого сповіщення нової групи. `Group interval` — мінімальний інтервал між повторними сповіщеннями однієї групи. `Repeat interval` — інтервал повторної відправки для алертів, що залишаються у стані Firing.

d. **Створення Alert Rule — High Request Rate.**

Перейти до Alerting → Alert rules → New alert rule. Налаштувати правило спрацювання при високій швидкості HTTP-запитів (рис. 7.d):

```
# ── Alert Rule: High Request Rate ──────────────────────

# Rule name: High Nginx Request Rate
# Folder: Nginx Alerts (створити нову папку)
# Evaluation group: Request Monitoring (створити)

# ── Section 1: Define query and alert condition ─────────
# Data source: Prometheus

# Query A — швидкість запитів за останню хвилину:
rate(nginx_http_requests_total[1m])

# Expression B — Reduce:
#   Function: Last   (останнє значення ряду)
#   Input: A
#   Mode: Strict     (NaN = алерт не спрацьовує)

# Expression C — Threshold:
#   Input: B
#   IS ABOVE: 10     (поріг: 10 запитів/сек)
#   Set as alert condition: увімкнути

# ── Section 2: Set evaluation behavior ──────────────────
# Evaluation group: Request Monitoring
# Evaluation interval: 1m  (перевіряти щохвилини)
# Pending period: 2m
#   (алерт спрацює лише після 2 хвилин
```

```
#    безперервного перевищення порогу)

# ── Section 3: Configure no data and error handling ─────
# Alert state if no data: OK
# Alert state if execution error: Alerting

# ── Section 4: Add annotations ──────────────────────────
# Summary: Nginx HTTP request rate exceeds 10 req/s
# Description: Current rate: {{ $values.B }} req/s
#              Server: monitoring-lab-xxx-yyy-zzz

# ── Section 5: Add labels ───────────────────────────────
# severity: warning
# service: nginx
```

Рис. 7.d – Конфігурація Alert Rule — High Request Rate

Pending period (2m) — критично важливий параметр, який запобігає хибним спрацюванням (false positives) при короткочасних сплесках навантаження. Алерт проходить через три стани: Normal → Pending (умова виконана, але час очікування не вичерпано) → Firing (умова виконана безперервно протягом pending period). Лише при переході у стан Firing відправляється email-сповіщення.

e. **Створення Alert Rule — Low Active Connections (для тестування).**

Створити додаткове правило, яке спрацьовує при низькій кількості активних з'єднань. Це дозволить негайно перевірити роботу всього ланцюга алертингу без необхідності генерувати інтенсивне навантаження (рис. 7.e):

```
# ── Alert Rule: Low Active Connections ──────────────────

# Rule name: Nginx Low Active Connections
# Folder: Nginx Alerts
# Evaluation group: Connection Monitoring (створити)

# Query A:
nginx_connections_active

# Expression B — Reduce:
#   Function: Last
#   Input: A
#   Mode: Strict

# Expression C — Threshold:
#   Input: B
#   IS BELOW: 5
```

```
#   Set as alert condition: увімкнути

# Evaluation interval: 1m
# Pending period: 1m

# Annotations:
#   Summary: Low number of active Nginx connections
#   Description: Active connections: {{ $values.B }}

# Labels:
#   severity: info
#   service: nginx
```

Рис. 7.e – Конфігурація Alert Rule — Low Active Connections

Це правило спрацює при менше ніж 5 активних з'єднаннях — типова ситуація для навчального сервера без навантаження. Алерт дозволить перевірити повний ланцюг сповіщень: Grafana Alerting → Postfix (порт 25) → SendGrid SMTP (порт 587 з TLS) → поштова скринька студента.

Зберегти правило та зачекати 2–3 хвилини (1 хвилина evaluation interval + 1 хвилина pending period + запас). Перевірити стан алертів: Alerting → Alert rules — правило "Nginx Low Active Connections" має перейти у стан "Firing" (червоний індикатор).

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s http://localhost:3000/api/alertmanager/grafana/api/v2/alerts | jq .[].labels.alertname
```

Зробити висновки за отриманими результатами.

8. Тестування повного ланцюга моніторингу та алертингу.

У цьому завданні необхідно переконатися, що весь ланцюг моніторингу працює коректно: від збору метрик nginx-prometheus-exporter → зберігання у Prometheus → візуалізація у Grafana → спрацювання Alert Rule → відправка через Postfix/SendGrid → отримання email. Також необхідно перевірити поведінку алертів при зміні стану (Firing → Resolved).

a. **Перевірка отримання email-алерту.**

Перевірити поштову скриньку `username@student.ztu.edu.ua` — має надійти лист від `alerts@username.pp.ua` із заголовком, що містить "[FIRING:1]" та назву алерту. Відкрити лист та перевірити його вміст:

має містити назву алерту, поточне значення метрики, анотації (Summary та Description) та посилання на Grafana.

Якщо лист не надійшов протягом 5 хвилин — перевірити журнали на кожному етапі ланцюга:

```
# 1. Журнали Grafana — чи намагалась відправити email
docker logs grafana 2>&1 | grep -i "email\|smtp\|alert" | tail -20

# 2. Журнали Postfix — чи прийняв лист від Grafana
sudo tail -30 /var/log/mail.log

# 3. Перевірити, чи Postfix слухає на всіх інтерфейсах
sudo ss -tlnp | grep ":25"

# 4. Перевірити стан алертів у Grafana API
curl -s http://localhost:3000/api/alertmanager/grafana/api/v2/alerts | jq
.
```

Рис. 8.a – Команди діагностики ланцюга алертингу

Типові причини відсутності листів: Postfix не слухає на інтерфейсі Docker (перевірити `inet_interfaces = all` у `main.cf`); Docker-мережа не включена до `mynetworks` (перевірити наявність `172.16.0.0/12`); невірний SMTP-порт у Grafana (має бути 25 для Postfix, не 587); SendGrid API Key невалідний або прострочений; contact point не призначений у notification policy.

b. **Генерація навантаження для тригерування алерту High Request Rate.**

Згенерувати інтенсивне навантаження, щоб перевищити поріг 10 req/s та спровокувати спрацювання алерту "High Nginx Request Rate" (рис. 8.b):

```
# Інтенсивне навантаження: ~20 req/s протягом 3 хвилин
# (перевищує поріг 10 req/s з запасом для впевненого
# спрацювання після pending period)
for i in $(seq 1 3600); do
    curl -s -o /dev/null http://monitoring-lab-xxx-yyy-zzz.local
    sleep 0.05   # ~20 запитів на секунду
done &

# Зберегти PID фонового процесу для зупинення
LOAD_PID=$!
echo "Load generator started, PID: $LOAD_PID"
```

Рис. 8.b – Скрипт генерації інтенсивного навантаження

Зачекати 3–4 хвилини (1 хвилина evaluation interval + 2 хвилини pending period + запас на scrape та обчислення). Перевірити стан алерту у Grafana: Alerting → Alert rules — правило "High Nginx Request Rate" має послідовно перейти зі стану Normal → Pending → Firing.

Перевірити отримання email-сповіщення про спрацювання алерту. Тема листа має містити "[FIRING:1] High Nginx Request Rate".

c. **Перевірка переходу алерту у стан Resolved.**

Зупинити генератор навантаження:

```
kill $LOAD_PID 2>/dev/null || kill %1 2>/dev/null; echo "Load generator stopped"
```

Після зупинення навантаження функція `rate(nginx_http_requests_total[1m])` поступово зменшить значення до нуля протягом хвилини (інтервал [1m] у запиті). Зачекати 3–4 хвилини — алерт "High Nginx Request Rate" має перейти зі стану Firing у стан Normal. Grafana автоматично відправить email-сповіщення про перехід у стан Resolved.

Перевірити отримання resolved-сповіщення на пошті — тема листа має містити "[RESOLVED]". Цей механізм важливий у виробничому середовищі — адміністратор бачить не лише проблеми, а й їх автоматичне вирішення.

d. **Огляд дашборду під час та після навантаження.**

Відкрити дашборд "Nginx Monitoring — xxx-yyy-zzz" у Grafana. Встановити часовий діапазон "Last 30 minutes" (іконка годинника у верхньому правому куті). Проаналізувати візуалізації:

— панель "HTTP Requests per Second" має показувати чіткий сплеск (~20 req/s) під час навантаження та різке падіння до ~0 після зупинення генератора;

— панель "Active Connections" має відображати збільшення кількості з'єднань під час навантаження — типово 2–5 для 20 req/s;

— панель "Connection States" має демонструвати зміну розподілу: збільшення Writing (Nginx відповідає клієнтам) та зменшення Waiting (менше idle-з'єднань) під час навантаження;

— панель "Accepted vs Handled" — значення мають бути рівними (Nginx не відхиляє з'єднання при такому навантаженні).

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s
"http://localhost:9090/api/v1/query?query=nginx_connections_active" |
jq .data.result[0].value[1]
```

Зробити висновки за отриманими результатами.

е. **Фінальна перевірка стану всіх компонентів системи моніторингу.**

Виконати комплексну перевірку працездатності всіх компонентів ланцюга моніторингу та алертингу (рис. 8.е):

```
# ── Перевірка стану Docker-контейнерів ──────────────────
echo "=== Docker Containers ==="
docker compose -f ~/logging-stack/docker-compose.yml ps

# ── Перевірка targets Prometheus ────────────────────────
echo "\n=== Prometheus Targets ==="
curl -s http://localhost:9090/api/v1/targets | \
  jq '.data.activeTargets[] | {job: .labels.job, health: .health}'

# ── Перевірка метрик exporter ───────────────────────────
echo "\n=== Nginx Metrics Count ==="
curl -s http://localhost:9113/metrics | grep "^nginx_" | wc -l

# ── Перевірка Postfix ───────────────────────────────────
echo "\n=== Postfix Status ==="
sudo systemctl status postfix --no-pager | head -5

# ── Перевірка алертів Grafana ───────────────────────────
echo "\n=== Grafana Alert Rules ==="
curl -s http://localhost:3000/api/alertmanager/grafana/api/v2/alerts | \
  jq '.[].labels.alertname'
```

Рис. 8.е – Скрипт комплексної перевірки всіх компонентів

Перевірити застосовану конфігурацію, виконавши команду:

```
docker compose -f ~/logging-stack/docker-compose.yml ps --format
"table {{.Name}}\t{{.Status}}"
```

Зробити висновки за отриманими результатами.

Контрольні запитання

1. Яку роль виконує nginx-prometheus-exporter у системі моніторингу та які саме метрики Nginx він збирає зі сторінки stub_status?
2. У чому полягає різниця між метриками-лічильниками (counter) та метриками-вимірниками (gauge) у Prometheus? Навести приклади кожного типу серед метрик Nginx.
3. Як функція rate() у PromQL обчислює швидкість зміни лічильника та чому для неї необхідно вказувати часовий інтервал (наприклад, [1m])?
4. Яким чином Grafana Alerting визначає момент переходу алерту зі стану Pending у стан Firing? Яку роль відіграє параметр Pending period?
5. Які переваги використання Postfix як проміжного SMTP relay замість прямого підключення Grafana до зовнішнього SMTP-сервера SendGrid?
6. У чому полягає призначення DNS-записів DKIM та DMARC у зоні домену відправника? Як вони впливають на доставку алертних листів?
7. Як відбувається взаємодія між Docker-контейнерами та хост-машиною через механізм host.docker.internal? Чому nginx-exporter та Grafana потребують цього механізму?
8. Які параметри конфігурації Prometheus визначають частоту збору метрик та тривалість їх зберігання? Як обрати оптимальні значення для виробничого середовища?
9. Яким чином параметр relayhost у конфігурації Postfix впливає на маршрутизацію вихідної пошти? Чому адреса хоста вказується у квадратних дужках?
10. Які кроки діагностики необхідно виконати, коли алерт у Grafana знаходиться у стані Firing, але email-сповіщення не надходить на пошту?