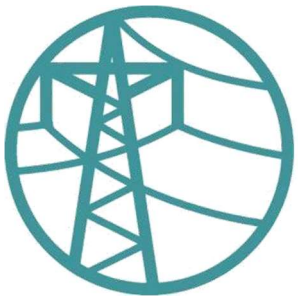


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Krzysztof Kołodziej

System monitorowania infrastruktury przy użyciu Grafany i Prometheusa

Systemy Integracyjne - projekt

Rzeszów, rok 2025

Spis treści

1. Wstęp

W dzisiejszym dynamicznym środowisku IT monitorowanie infrastruktury i aplikacji odgrywa kluczową rolę w zapewnieniu niezawodności, wydajności oraz bezpieczeństwa systemów informatycznych. Istnieje wiele narzędzi dostępnych na rynku, które umożliwiają monitorowanie różnorodnych aspektów infrastruktury – od wydajności serwerów, przez działanie aplikacji, po ruch sieciowy. Najpopularniejsze z nich to: **Nagios**, **Zabbix**, **Prometheus**, **Grafana**, **Elastic Stack (ELK)**, **Datadog**, **New Relic** czy **Splunk**. Każde z tych narzędzi ma swoje unikalne funkcje i zastosowania, dzięki czemu mogą być dostosowane do specyficznych potrzeb organizacji.

Systemy monitorowania pozwalają na śledzenie szerokiego zakresu parametrów, takich jak:

- **Wydajność serwerów**: obciążenie procesora, zużycie pamięci RAM, wykorzystanie przestrzeni dyskowej.
- **Stan aplikacji**: czas odpowiedzi, liczba zapytań, błędy aplikacyjne.
- **Sieć**: przepustowość, opóźnienia, utracone pakiety, dostępność usług sieciowych.
- **Zasoby w chmurze**: wykorzystanie maszyn wirtualnych, kontenery, funkcje serverless.

Jedną z kluczowych cech nowoczesnych narzędzi monitorujących, takich jak **Prometheus** i **Grafana**, są wbudowane mechanizmy alertowania. Pozwalają one na automatyczne wykrywanie anomalii, takich jak wysokie obciążenie procesora, brak dostępności usług czy nietypowy wzrost opóźnień w sieci. Dzięki alertom administratorzy mogą być natychmiast informowani o potencjalnych problemach za pośrednictwem e-maila, komunikatorów (np. Slack) lub SMS-a. Takie podejście umożliwia szybką diagnostykę i reakcję, co minimalizuje ryzyko przestojów oraz zapewnia nieprzerwane działanie systemów.

2. Cel projektu

Celem projektu jest zaprojektowanie i wdrożenie systemu monitorowania infrastruktury IT z wykorzystaniem technologii konteneryzacji Docker oraz narzędzi Prometheus, Grafana i Alertmanager, z dodatkowymi funkcjonalnościami, takimi jak testowanie wydajności przy użyciu K6 oraz monitorowanie serwera webowego za pomocą eksportera metryk. Projekt ma na celu zapewnienie kompleksowego rozwiązania umożliwiającego gromadzenie, wizualizację i analizę danych o stanie systemów, a także szybką diagnostykę i reakcję na potencjalne problemy.

3. Wykorzystanie konteneryzacji

Konteneryzacja, realizowana za pomocą **Dockera**, stanowi fundament projektu, umożliwiając efektywne zarządzanie i uruchamianie wszystkich komponentów systemu monitorowania w sposób wydajny, spójny i niezależny od środowiska. Kluczowe korzyści wynikające z zastosowania konteneryzacji to:

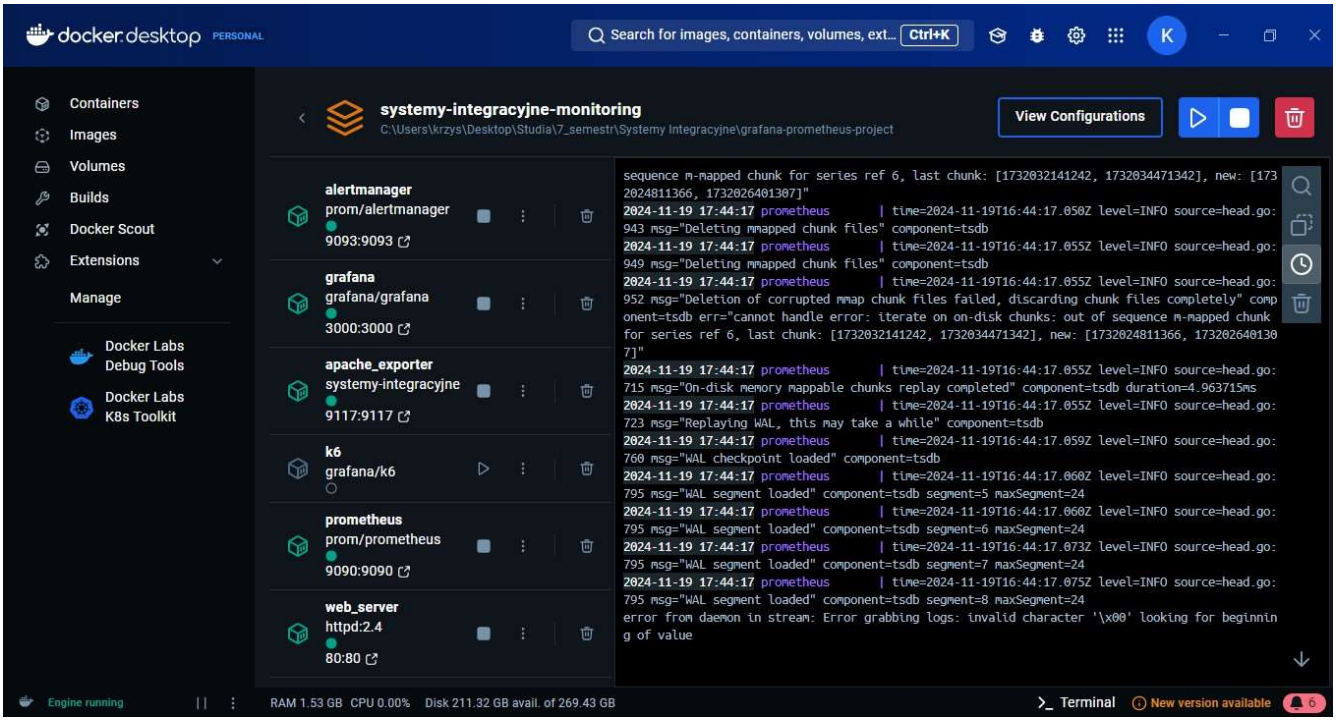
- **Przenośność**: Kontenery mogą być uruchamiane na różnych platformach bez potrzeby modyfikacji konfiguracji, co znacząco ułatwia wdrażanie rozwiązania na różnych etapach (np. w środowiskach deweloperskich, testowych i produkcyjnych).

- **Izolacja**: Każda usługa działa w swoim własnym kontenerze, co eliminuje konflikty między bibliotekami lub zależnościami różnych aplikacji.

- **Reprodukowalność**: Konfiguracja systemu jest zapisana w pliku docker-compose.yml, co pozwala odtworzyć identyczne środowisko w każdym miejscu, gdzie Docker jest dostępny.

- **Łatwe skalowanie**: Kontenery mogą być łatwo skalowane, co umożliwia szybkie dostosowanie systemu do zwiększonego ruchu czy zapotrzebowania na zasoby.

- **Szybkie wdrażanie i aktualizacja**: Zmiany w konfiguracji, obrazie kontenera lub zależnościach mogą być wprowadzane i wdrażane w krótkim czasie.



Rysunek 1 - kontenery utworzone i działające na potrzeby projektu widoczne w aplikacji Docker Desktop

W tym projekcie, Docker Desktop jest używany jako narzędzie do uruchamiania i zarządzania kontenerami, które są niezbędne do monitorowania i testowania aplikacji.

W ramach projektu są utworzone następujące kontenery:

- **web_server** - kontener z serwerem Apache, który będzie odpowiedzialny za obsługę zapytań HTTP. To na tym serwerze będą monitorowane metryki, takie jak liczba zapytań, obciążenie CPU czy statusy połączeń.

- **prometheus** - kontener, który zbiera metryki z różnych źródeł (w tym z serwera Apache przez apache_exporter) i przechowuje je w swojej bazie danych.

- **apache_exporter** - kontener, który zbiera metryki z serwera Apache i udostępnia je Prometheusowi. Monitoruje on różne wskaźniki związane z Apache, takie jak liczba zapytań, połączeń, procesów czy obciążenie CPU, które następnie są wykorzystywane do analiz i generowania alertów.

- **alertmanager** - kontener, który zarządza alertami generowanymi przez Prometheusa. Wysyła powiadomienia o problemach z wydajnością lub dostępnością, takie jak zbyt duże obciążenie czy problemy z serwerem Apache.

- **grafana** - kontener, który służy do wizualizacji zebranych metryk z Prometheusa. Za jego pomocą użytkownicy mogą monitorować wydajność serwera Apache w czasie rzeczywistym na różnych dashboardach.

- **k6** - kontener używany do przeprowadzania testów wydajnościowych. Po jego uruchomieniu, k6 generuje obciążenie na serwerze webowym, symulując zapytania HTTP w dużej ilości, aby sprawdzić jak serwer Apache radzi sobie z dużym ruchem. Kontener działa podczas trwania testu i automatycznie zatrzymuje się, gdy test wydajnościowy zostanie zakończony.

Wszystkie kontenery współpracują ze sobą, zapewniając kompleksowe środowisko do monitorowania, testowania wydajności i zarządzania alertami w projekcie.

4. Rola docker-compose w projekcie

Docker-compose to narzędzie służące do zarządzania wieloma kontenerami w sposób zautomatyzowany i przejrzysty. W projekcie pełni kluczową rolę, pozwalając na:

- **Centralne zarządzanie konfiguracją**

Wszystkie usługi są zdefiniowane w jednym pliku (docker-compose.yml), co umożliwia łatwe zarządzanie ich konfiguracją, portami, zależnościami i wolumenami.

- **Definiowanie zależności między usługami**

Dzięki dyrektywie depends_on usługi takie jak Grafana i Alertmanager uruchamiają się dopiero po Prometheusie, zapewniając prawidłową kolejność startu systemu.

- **Mapowanie danych i konfiguracji**

Wolumeny umożliwiają przechowywanie i łatwe zarządzanie plikami konfiguracyjnymi (np. prometheus.yml, dashboardy Grafany) oraz ich automatyczne ładowanie do odpowiednich kontenerów.

- **Tworzenie wspólnej sieci**

Docker-compose automatycznie konfiguruje sieć, umożliwiając komunikację między kontenerami za pomocą nazw usług, co upraszcza integrację systemu.

- **Automatyzacja wdrożenia**

Uruchomienie całego systemu sprowadza się do jednego polecenia: **docker-compose up -d**, wszystkie kontenery są uruchamiane automatycznie z wcześniej zdefiniowaną konfiguracją.

- **Elastyczność i skalowalność**

Docker-compose umożliwia łatwe skalowanie usług, np. dodanie nowych instancji serwera Apache: **docker-compose up –scale web_server=3**

Zastosowanie docker-compose w tym projekcie usprawnia zarządzanie wieloma usługami, zapewniając ich prawidłowe działanie i komunikację. Dzięki temu środowisko monitorujące jest łatwe do wdrożenia, skalowania i dalszego rozwoju.

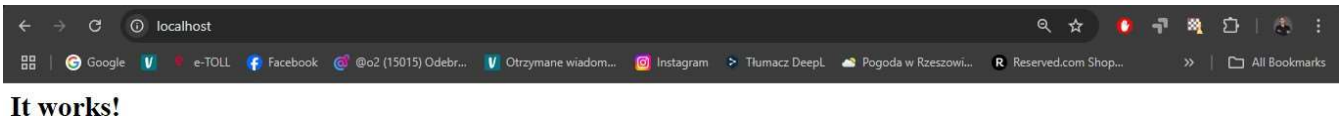
5. Implementacja prostego serwera webowego

W projekcie monitorowany jest prosty serwer webowy Apache, który pełni rolę serwera HTTP, odpowiedzialnego za obsługę zapytań i dostarczanie zawartości stron internetowych do użytkowników. Jego głównym zadaniem jest zapewnienie stabilnej i szybkiej obsługi żądań HTTP, co stanowi fundament wielu aplikacji webowych. Serwer ten będzie również zbierał dane o swoim stanie i wydajności, które następnie będą monitorowane przez Prometheusa oraz apache_exporter, co pozwoli na śledzenie kondycji serwera oraz identyfikowanie ewentualnych problemów.

Listing 1 - implementacja kontenera serwera webowego

```
web_server:
  container_name: web_server
  image: httpd:2.4
  ports:
    - '80:80'
  restart: always
```

Kontener **web_server** jest oparty na oficjalnym obrazie **httpd:2.4**, który jest popularnym obrazem Dockerowym dla serwera Apache w wersji 2.4. Kontener działa na porcie 80, co oznacza, że serwer webowy będzie komunikować się z użytkownikami za pomocą standardowego portu HTTP. Kontener został skonfigurowany do automatycznego restartu (**restart: always**), co zapewnia jego ponowne uruchomienie w przypadku awarii.



Rysunek 2 - działanie serwera webowego

Po uruchomieniu kontenera i wejściu na adres http://localhost:80 jest prezentowana domyślna strona serwera Apache. Jest to standardowa strona, która informuje, że serwer działa poprawnie i jest gotowy do obsługi zapytań. W późniejszym etapie, strona ta może zostać zastąpiona przez własną zawartość.

6. Implementacja Grafany

W projekcie Grafana jest wykorzystywana jako główne narzędzie do wizualizacji metryk zbieranych przez Prometheus. Dzięki zintegrowanym dashboardom użytkownicy mogą monitorować stan infrastruktury oraz analizować kluczowe wskaźniki wydajności w czasie rzeczywistym.

Listing 2 - konfiguracja kontenera grafany

```
name: systemy-integracyjne-monitoring
services:
  grafana:
    container_name: grafana
    image: grafana/grafana
    ports:
      - '3000:3000'
    depends_on:
      - prometheus
    volumes:
      - ./grafana/provisioning/datasources:/etc/grafana/provisioning/datasources
      - ./grafana/provisioning/dashboards:/etc/grafana/provisioning/dashboards
      - ./grafana/dashboards:/var/lib/grafana/dashboards
      - ./grafana/grafana.ini:/etc/grafana/grafana.ini
    restart: always
    environment:
      - GRAFANA_ADMIN_USER=${GRAFANA_ADMIN_USER}
      - GRAFANA_ADMIN_PASSWORD=${GRAFANA_ADMIN_PASSWORD}
```

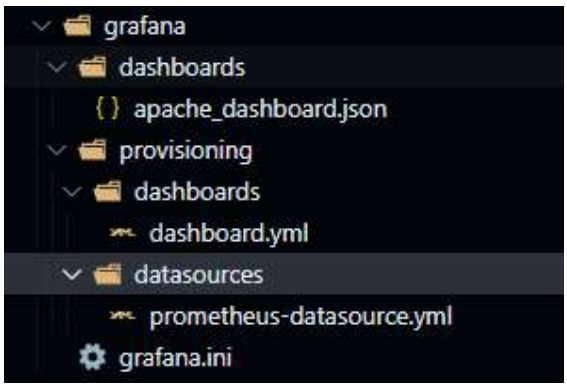
Do uruchomienia Grafany wykorzystano oficjalny obraz Dockera o nazwie grafana/grafana. Kontener jest dostępny na porcie 3000, który został zmapowany na ten sam port na hoście, co umożliwia dostęp do interfejsu webowego za pomocą przeglądarki. Grafana została skonfigurowana jako usługa zależna od Prometheusa, dzięki czemu narzędzie uruchamia się dopiero po zapewnieniu dostępności źródła danych.

Grafana została skonfigurowana z użyciem zmiennych środowiskowych, które definiują dane logowania administratora, takie jak nazwa użytkownika (GRAFANA_ADMIN_USER) i hasło (GRAFANA_ADMIN_PASSWORD). To podejście umożliwia łatwe zarządzanie danymi uwierzytelniającymi, przy zachowaniu bezpieczeństwa.

Aby zapewnić niezawodność działania, usługa Grafany ma ustawiony automatyczny restart

(restart: always), co sprawia, że kontener zostanie ponownie uruchomiony w przypadku ewentualnych awarii. Taka konfiguracja gwarantuje, że Grafana będzie dostępna w sposób ciągły i odporna na chwilowe problemy systemowe.

W celu elastycznego zarządzania danymi i konfiguracją zastosowano wolumeny. Pliki konfiguracyjne, takie jak **datasources** (definicja źródeł danych), **dashboards** (predefiniowane dashboardy) oraz **grafana.ini** (globalne ustawienia Grafany), zostały załadowane z katalogów lokalnych na hosta. Taki sposób konfiguracji ułatwia dostosowanie systemu do wymagań projektu, jednocześnie zapewniając trwałość danych między restartami kontenera.



Rysunek 3 – Pliki przekazywane do kontenera grafany przez wolumen

Struktura folderów zawiera kilka ważnych plików i katalogów. W katalogu **grafana/dashboards/** znajduje się plik **apache_dashboard.json**, który jest używany do ładowania predefiniowanego dashboardu w Grafanie.

Folder **grafana/provisioning/** zawiera dwa podkatalogi: **datasources** i **dashboards**. W katalogu datasources znajduje się plik **prometheus-datasource.yml**, który służy do konfiguracji źródła danych Prometheus w Grafanie. Ten plik umożliwia Grafanie prawidłowe pobieranie danych z Prometheusa. Z kolei katalog dashboards zawiera plik dashboard.yml, który pozwala na automatyczne załadowanie konfiguracji dashboardów w Grafanie.

Dodatkowo, plik grafana.ini w katalogu grafana to główny plik konfiguracyjny, w którym znajdują się globalne ustawienia, takie jak preferencje użytkowników, dostępność dashboardów czy opcje dotyczące bezpieczeństwa.

Listing 3 - konfiguracja źródła danych Prometheusa (prometheus-datasource.yml)

```yaml
apiVersion: 1

datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
```

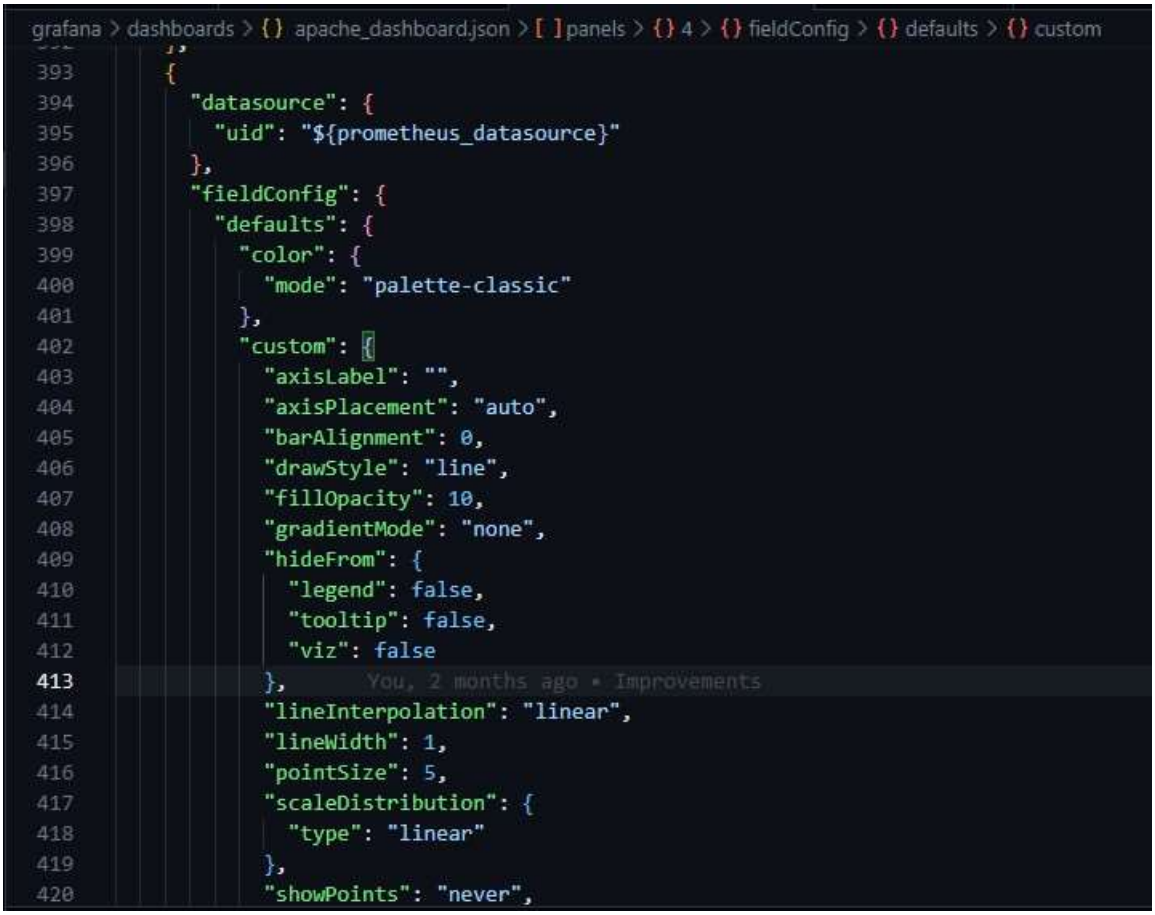
```
    isDefault: true
    editable: true
```

W przedstawionej konfiguracji znajduje się definicja źródła danych dla Grafany, które wskazuje na Prometheusa jako źródło metryk. Określa ona wersję API jako "1", a sekcja "datasources" definiuje, jakie źródła danych Grafana będzie wykorzystywać do pobierania metryk. Źródło danych o nazwie "Prometheus" jest typu "prometheus", co oznacza, że Grafana będzie komunikować się z Prometheusem w celu pobrania danych. Dostęp do Prometheusa jest realizowany poprzez serwer Grafany za pomocą opcji "proxy". Adres URL, pod którym Prometheus jest dostępny, to "http://prometheus:9090", co wskazuje na komunikację z kontenerem o nazwie "prometheus" na porcie 9090. Źródło danych "Prometheus" zostało oznaczone jako domyślne ("isDefault: true"), co oznacza, że będzie używane do pobierania metryk na wszystkich dashboardach w Grafanie. Dodatkowo, opcja "editable: true" pozwala na edytowanie tego źródła danych w interfejsie użytkownika Grafany.

Listing 4 – plik konfiguracji dashboardów (dashboard.yml)

```
apiVersion: 1

providers:
  - name: 'default'
    orgId: 1
    folder: ''
    type: file
    disableDeletion: false
    updateIntervalSeconds: 10
    options:
      path: /var/lib/grafana/dashboards
```

W przedstawionej konfiguracji znajduje się definicja dostawcy dashboardów dla Grafany. Sekcja "apiVersion: 1" określa wersję API, a sekcja "providers" definiuje sposób, w jaki Grafana będzie ładować i zarządzać dashboardami. Dostawca o nazwie "default" jest przypisany do organizacji o identyfikatorze "1". Wartość "folder" jest pusta, co oznacza, że dashboardy nie są przypisane do żadnego folderu. Typ dostawcy to "file", co oznacza, że Grafana będzie pobierać dashboardy z plików przechowywanych na dysku. Opcja "disableDeletion: false" umożliwia usuwanie dashboardów, a "updateIntervalSeconds: 10" ustawia czas, co jaki Grafana będzie sprawdzać i aktualizować dostępne dashboardy. Opcja "path" określa ścieżkę do katalogu, w którym znajdują się pliki dashboardów, w tym przypadku "/var/lib/grafana/dashboards".



Rysunek 4 – predefiniowany dashboard Apache HTTP server (apache_dashboard.json)

Plik **apache_dashboard.json** to plik w formacie JSON, który zawiera zdefiniowane wykresy, panele i konfigurację dashboardu dla Grafany, służący do wizualizacji metryk związanych z serwerem Apache. Zawiera on wszystkie elementy potrzebne do załadowania gotowego dashboardu do interfejsu Grafany.

Listing 5 - plik globalnej konfiguracji Grafany (grafana.ini)

```
[paths]
data = /var/lib/grafana/data
logs = /var/log/grafana
plugins = /var/lib/grafana/plugins
provisioning = /etc/grafana/provisioning

[server]
http_port = 3000

[security]
admin_user = ${GRAFANA_ADMIN_USER}
admin_password = ${GRAFANA_ADMIN_PASSWORD}
```

Przedstawiony fragment konfiguracji dotyczy ustawień kontenera Grafany. W sekcji [paths] zdefiniowane są ścieżki, w których Grafana przechowuje swoje dane. data wskazuje katalog, w którym przechowywane są dane aplikacji, takie jak bazy danych i metryki, logs to miejsce, gdzie zapisane są logi

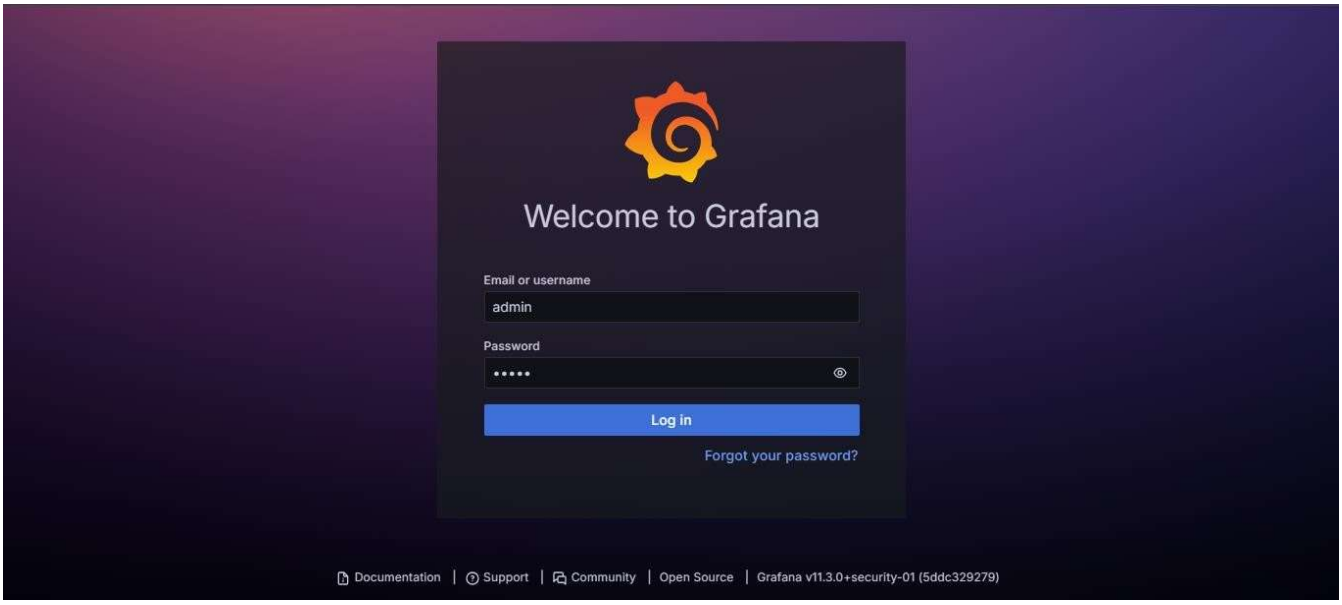
Grafany, plugins wskazuje folder na wtyczki, a provisioning określa lokalizację plików konfiguracyjnych służących do automatycznego provisioningu, takich jak źródła danych i dashboardy.

W sekcji [server] zdefiniowana jest port, na którym nasłuchuje serwer Grafany. Ustawienie http_port = 3000 oznacza, że interfejs webowy Grafany będzie dostępny pod tym portem.

W sekcji [security] zdefiniowane są zmienne środowiskowe do konfiguracji konta administratora. admin_user i admin_password odpowiadają za dane logowania do Grafany, gdzie wartości są pobierane z zmiennych środowiskowych GRAFANA_ADMIN_USER i GRAFANA_ADMIN_PASSWORD, co pozwala na bezpieczne przechowywanie haseł poza samą konfiguracją.

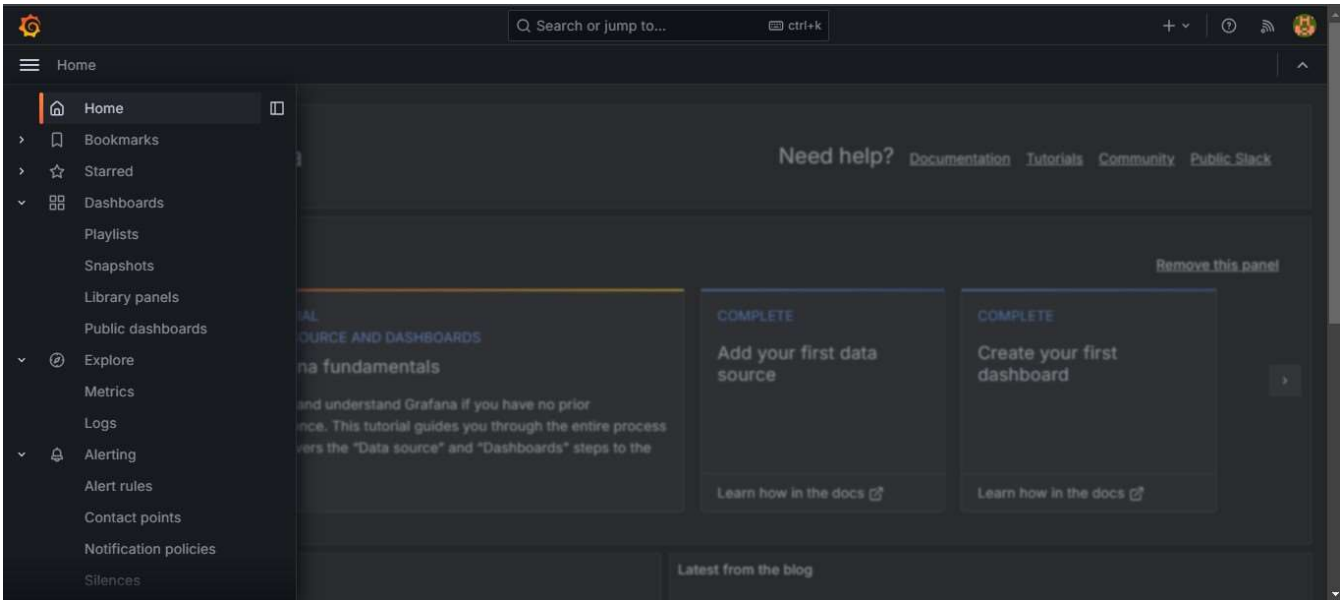
7. Prezentacja działania Grafany

Wchodząc na stronę **localhost:3000**, wita nas ekran logowania. To miejsce, w którym użytkownicy mogą wprowadzić swoje dane logowania, aby uzyskać dostęp do interfejsu Grafany.



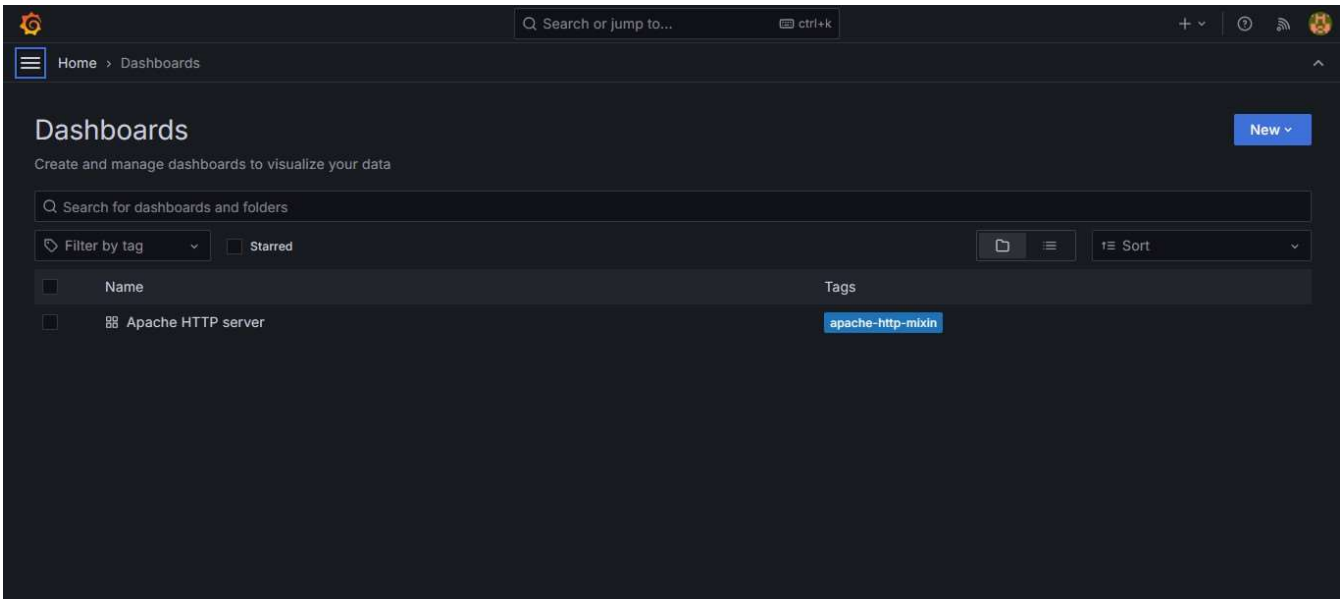
Rysunek 5 - ekran logowania do Grafany

Po poprawnym zalogowaniu, użytkownik zostaje przekierowany do strony głównej Grafany. Na tej stronie, rozwijając pasek po lewej stronie, mamy dostęp do różnych sekcji, takich jak dashboardy, źródła danych, alerty czy ustawienia.

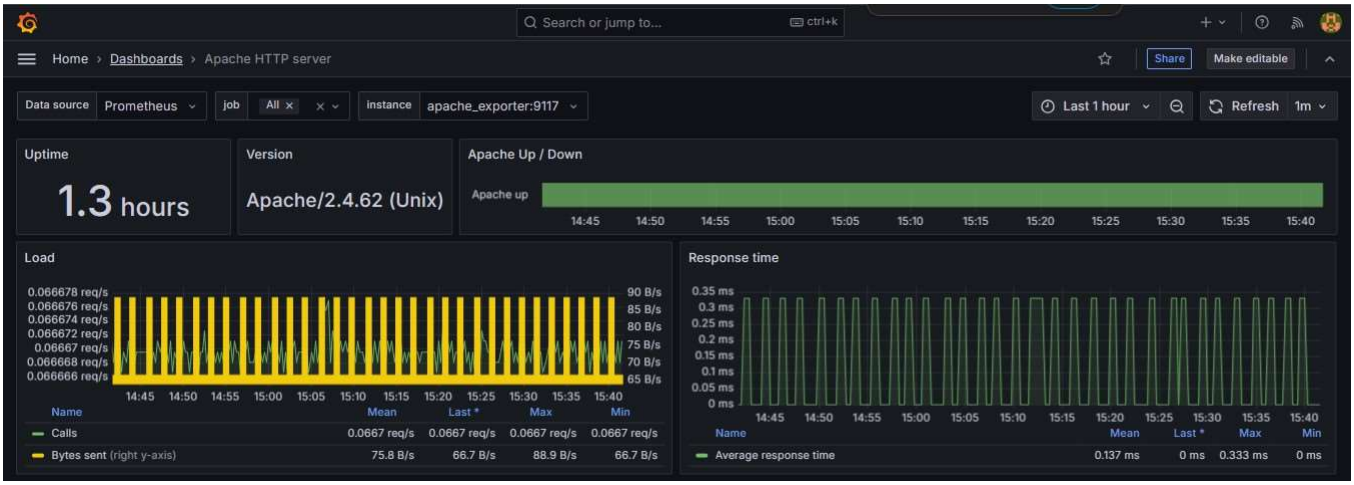


Rysunek 6 - strona głowna Grafany

Po przejściu do sekcji **Dashboards**, zobaczymy nasz provisionowany **Apache Dashboard**. To gotowy panel, który został wcześniej skonfigurowany i podany do kontenera przez wolumen.



Rysunek 7 - lista dashboardów



Rysunek 8 - Apache HTTP server dashboard - część 1

Rysunek 9 - Apache HTTP server dashboard - część 2

Po przejściu do sekcji **Dashboards**, na naszym provisionowanym **Apache http server dashboard** widzimy szereg kluczowych metryk monitorujących wydajność serwera Apache. Dashboard zawiera różnorodne wykresy i informacje, które pomagają w zarządzaniu serwerem i jego optymalizacji.

- **Uptime** – Wskazuje, jak długo serwer Apache działa bez przerwy. Jest to ważny wskaźnik, który pozwala na ocenę stabilności i dostępności serwera.

- **Version** – Pokazuje wersję zainstalowanego oprogramowania Apache. Dzięki tej informacji możemy szybko zweryfikować, która wersja Apache jest uruchomiona na serwerze, co jest istotne dla zgodności z różnymi funkcjami i modułami.

- **Apache Up/Down** – Informacja o stanie serwera Apache, wskazująca, czy serwer jest włączony (up) czy wyłączony (down). Jest to podstawowa metryka dostępności, która pozwala na szybkie wykrycie problemów z działaniem usługi.

- **Load** – Pokazuje obciążenie serwera Apache w danym momencie. Wskaźnik ten mierzy, jak intensywnie wykorzystywany jest serwer, co pomaga w ocenie wydajności i w identyfikacji potencjalnych przeciążeń systemu.

- **Response Time** – Mierzy czas odpowiedzi serwera na zapytania. Jest to istotny wskaźnik wydajności, który pozwala ocenić, jak szybko serwer reaguje na żądania użytkowników.

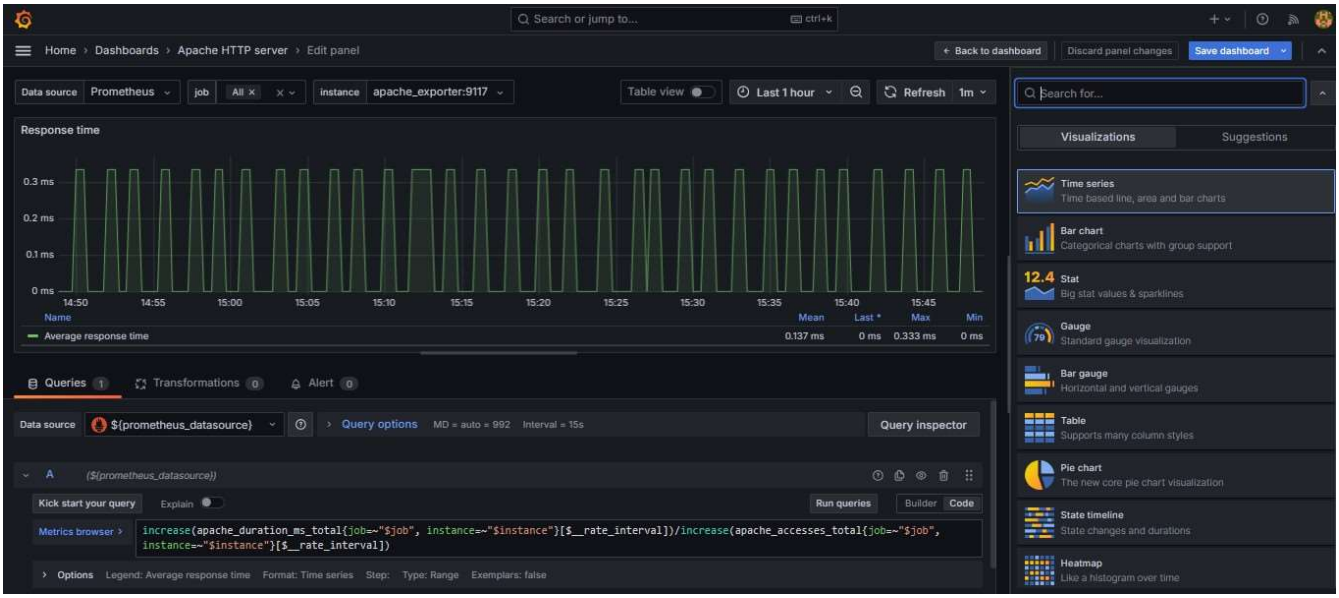
- **Apache Scoreboard Statuses** – Pokazuje różne statusy w ramach tzw. "scoreboard" Apache, które przedstawiają stan poszczególnych procesów w serwerze, takie jak aktywność w obsłudze zapytań, oczekiwanie na dane itp. Jest to szczególnie pomocne w diagnozowaniu problemów z wydajnością serwera.

- **Apache Worker Statuses** – Prezentuje statusy poszczególnych workerów Apache, które odpowiadają za obsługę zapytań. Dzięki temu możemy zobaczyć, jak efektywnie serwer rozdziela zadania między różne procesy.

- **Apache CPU Load** – Mierzy obciążenie procesora przez serwer Apache. Jest to kluczowy wskaźnik w ocenie, czy serwer nie jest przeciążony i czy potrzebne są optymalizacje pod kątem zasobów CPU.

Dzięki tym wszystkim metrykom możemy uzyskać pełny obraz stanu naszego serwera Apache w czasie rzeczywistym.

Co ważne, Grafana pozwala na dużą elastyczność w dostosowywaniu tych informacji. Możemy tworzyć i edytować własne wykresy, które będą odpowiadały naszym potrzebom monitoringu. Grafana oferuje również stronę z gotowymi dashboardami https://grafana.com/grafana/dashboards/, które zostały stworzone przez innych użytkowników i firmy, co umożliwia szybkie wdrożenie najlepszych praktyk i rozwiązań stosowanych w branży. Możemy korzystać z tych gotowych dashboardów lub na ich podstawie tworzyć własne panele, dostosowane do specyficznych wymagań.



Rysunek 10 - widok edycji wykresu

W Grafanie, w widoku edycji wykresu, użytkownicy mają pełną swobodę w dostosowywaniu wykresów do swoich potrzeb. Można łatwo zmieniać różne parametry wykresu, aby uzyskać najbardziej odpowiednią wizualizację danych. W sekcji edycji możemy modyfikować zapytanie (query), które służy do pobierania danych z wybranego źródła (np. bazy danych, Prometheusa czy InfluxDB). W zależności od potrzeb, możemy zmieniać zapytania, dodawać nowe metryki lub dostosowywać filtry, aby uzyskać dokładnie te dane, które chcemy monitorować.

Grafana umożliwia także zmianę typu wykresu na różne formy wizualizacji, takie jak wykresy liniowe, słupkowe, obszarowe, wykresy radarowe, tabele, wykresy z mapami ciepła i wiele innych. Dzięki temu użytkownik może dostosować prezentację danych do typu analizowanej metryki, co sprawia, że wykresy są bardziej czytelne i łatwiejsze do interpretacji. W edytorze można również dostosować szczegóły wykresu, takie jak oś Y, jednostki, skala logarytmiczna, kolory, legendy, tytuły, etykiety osi i inne elementy wizualne. Dodatkowo, można ustawić progi, które będą wskazywać na istotne wartości, np. alarmy, lub dostosować zakresy danych, aby lepiej pasowały do kontekstu analizy.

Edycja wykresów w Grafanie pozwala również na dodanie interakcji z wykresami, na przykład możliwość najechania kursorem na wykres, co pokaże szczegółowe dane dla danego punktu na wykresie. Dzięki temu użytkownicy mogą dokładnie przeanalizować dane, nie tylko patrząc na ogólną wizualizację, ale także zgłębiając szczegóły. Edytowanie wykresów w Grafanie jest więc bardzo elastyczne, co pozwala na pełne dostosowanie paneli do specyficznych potrzeb użytkownika i tworzenie intuicyjnych oraz funkcjonalnych widoków monitoringu.

8. Implementacja Prometheusa

W projekcie Prometheus pełni rolę głównego narzędzia do zbierania i przechowywania metryk, które są następnie wizualizowane w Grafanie. **Metryka** to jednostka danych, która reprezentuje zmienną, monitorowaną właściwość systemu lub aplikacji, związaną z jego wydajnością, stanem lub zachowaniem. Metryki mogą obejmować różne wskaźniki, takie jak obciążenie procesora, ilość zapytań HTTP, czas odpowiedzi serwera, dostępność usług, wykorzystanie pamięci czy liczba błędów.

Listing 6 - konfiguracja kontenera Prometheusa

```
prometheus:
    container_name: prometheus
    image: prom/prometheus
    ports:
      - '9090:9090'
    volumes:
      - ./prometheus:/etc/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--web.enable-lifecycle'
    restart: always
```

Do uruchomienia Prometheusa wykorzystano oficjalny obraz Dockera o nazwie **prom/prometheus**. Kontener jest dostępny na porcie 9090, który został zmapowany na ten sam port na hoście, co umożliwia dostęp do interfejsu webowego Prometheusa za pomocą przeglądarki. Prometheus został skonfigurowany tak, aby korzystać z pliku konfiguracyjnego **prometheus.yml**, który definiuje źródła danych i zasady zbierania metryk.

Kontener Prometheusa został wyposażony w odpowiednią komendę uruchomieniową, która wskazuje na lokalizację pliku konfiguracyjnego i umożliwia uruchomienie funkcji związanych z zarządzaniem cyklem życia usługi, takich jak uruchamianie i zatrzymywanie Prometheusa.

Aby zapewnić elastyczność w zarządzaniu konfiguracją i danymi, zastosowano wolumeny, które umożliwiają montowanie lokalnych plików konfiguracyjnych na hosta do kontenera. W tym przypadku, katalog lokalny **./prometheus** zawiera pliki konfiguracyjne, które są używane przez Prometheusa, takie jak **prometheus.yml** oraz inne ustawienia, pozwalające na dostosowanie zbierania i przechowywania metryk.

Podobnie jak w przypadku Grafany, Prometheus ma ustawiony automatyczny restart (restart: always), co zapewnia jego ciągłą dostępność. W przypadku awarii kontenera, Prometheus zostanie automatycznie ponownie uruchomiony, gwarantując, że zbieranie metryk nie zostanie przerwane i cały system monitoringu pozostanie w pełni funkcjonalny.

Listing 7 - plik konfiguracyjny prometheus.yml

```yaml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'apache_exporter'
    static_configs:
      - targets: ['apache_exporter:9117']

rule_files:
  - "/etc/prometheus/rules.yml"

alerting:
  alertmanagers:
  - api_version: v2
    enable_http2: True
    follow_redirects: True
    scheme: http
    timeout: 10s
    static_configs:
      - targets: ['alertmanager:9093']
```

Konfiguracja Prometheusa zawiera kilka kluczowych sekcji, które umożliwiają zbieranie metryk, definiowanie zasad alarmowania i integrację z systemem alertów.

W sekcji **global** ustawiono domyślny interwał zbierania danych (scrape_interval) na 15 sekund, co oznacza, że Prometheus będzie co 15 sekund zbierał metryki z podłączonych źródeł.

W sekcji **scrape_configs** zdefiniowano jedno źródło danych, którym jest **apache_exporter**. Jest to narzędzie służące do zbierania metryk związanych z serwerem Apache. Prometheus zbiera metryki z tego eksportera, który działa na porcie 9117. W tym przypadku adres apache_exporter:9117 wskazuje na miejsce, z którego Prometheus będzie pobierał dane o stanie i wydajności serwera Apache.

Sekcja **rule_files** wskazuje na plik z zasadami alarmów, który znajduje się w lokalizacji /etc/prometheus/rules.yml. Dzięki temu Prometheus jest w stanie reagować na zmiany w metrykach i generować powiadomienia, jeśli określone progi zostaną przekroczone.

W sekcji **alerting** zdefiniowano konfigurację dla systemu powiadomień, który korzysta z **Alertmanagera**. Alertmanager jest odpowiedzialny za zarządzanie i wysyłanie powiadomień o alarmach. Zawiera on informacje o wersji API, włączeniu HTTP/2, oraz ustawieniach połączenia z Alertmanagerem, w tym adres docelowy alertmanager:9093, który wskazuje na usługę Alertmanagera działającą na porcie 9093. Dzięki tym ustawieniom Prometheus jest w stanie powiadamiać administratorów o przekroczeniu ważnych progów metryk, umożliwiając szybkie reagowanie na potencjalne problemy w infrastrukturze.

Ta konfiguracja umożliwia zbieranie metryk z serwera Apache, monitorowanie ich w czasie rzeczywistym i automatyczne generowanie alarmów, jeśli coś nie działa zgodnie z oczekiwaniami.

Listing 8 - definicja zasad alertów rules.yml

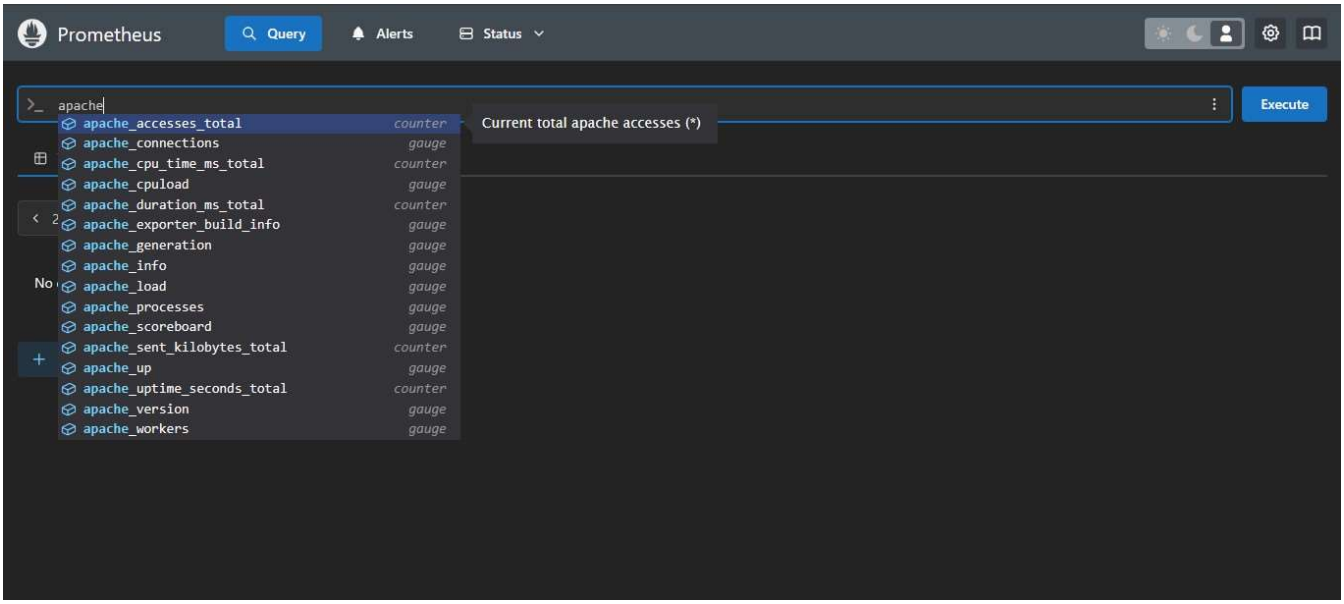
```
groups:
- name: alert_rules
  rules:
  - alert: HighRequestRate
    expr: rate(http_requests_total{job="web_server"}[5m]) > 100
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: High request rate on web server
      description: '{{ $labels.instance }} has high request rate (>100 req/min) for
the last 5 minutes.'

  - alert: HighCPULoad
    expr: node_load1 > 0.8
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: High CPU load detected
      description: '{{ $labels.instance }} has CPU load >0.8 for the last 5 minutes.'

  - alert: InstanceDown
    expr: apache_up == 0
    for: 30s
    labels:
      severity: critical
    annotations:
      summary: Instance down
      description: '{{ $labels.instance }} is down for more than 30 seconds.'
```

Konfiguracja zawiera trzy reguły alarmowe dla Prometheusa. Pierwsza, **HighRequestRate**, uruchamia alarm, gdy średnia liczba zapytań HTTP na minutę przekroczy 100 w ciągu ostatnich 5 minut. Druga reguła, **HighCPULoad**, generuje ostrzeżenie, gdy obciążenie CPU (mierzone jako node_load1) przekroczy wartość 0.8 przez co najmniej 5 minut. Trzecia reguła, **InstanceDown**, włącza alarm krytyczny, gdy serwer Apache jest niedostępny przez ponad 30 sekund (metryka apache_up równa się 0). Każdy z alarmów zawiera etykiety i opisy, które pomagają zidentyfikować przyczynę problemu.

9. Prezentacja działania Prometheusa

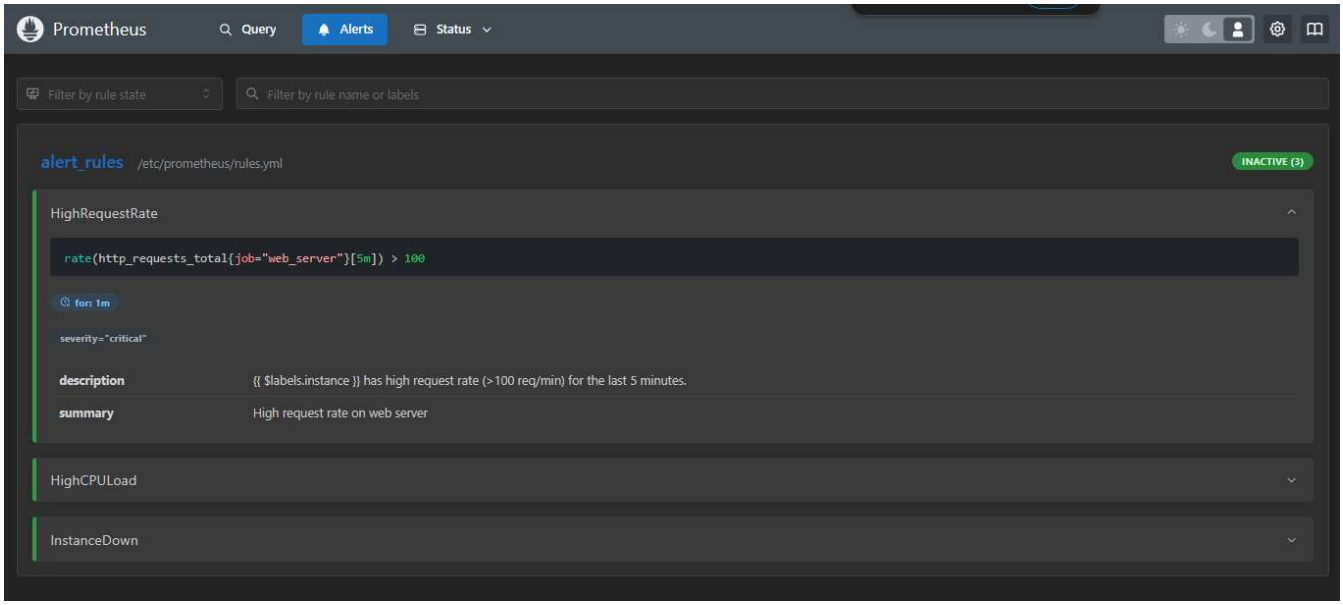
Na adresie http://localhost:9090/ uruchamia się Prometheus, który udostępnia interfejs webowy do monitorowania zebranych metryk. Na stronie głównej Prometheusa widzimy interfejs użytkownika,

który umożliwia wykonywanie zapytań za pomocą języka PromQL oraz przeglądanie zebranych danych w postaci serii czasowych. W przypadku monitorowania serwera Apache, dzięki wykorzystanie apache_exportera, o którym będzie późniejszy rozdział, Prometheus zbiera metryki takie jak apache_up, która informuje o dostępności serwera, oraz inne metryki związane z jego wydajnością, np. liczba zapytań, czas odpowiedzi czy obciążenie serwera. Wszystkie te metryki są dostępne do analizy i dalszego wykorzystania w Grafanie.



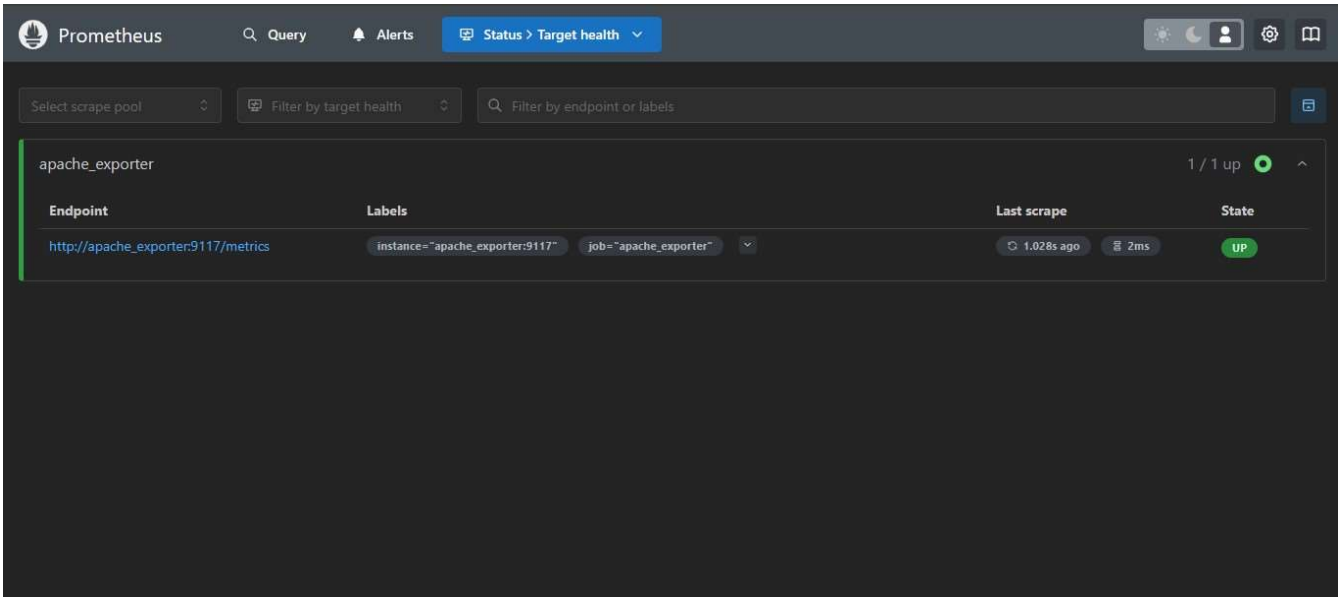
Rysunek 11 - strona główna Prometheusa wraz z widocznymi metrykami serwera Apache

W sekcji alertów Prometheusa możemy zobaczyć wcześniej skonfigurowane alerty, które zostały zdefiniowane w pliku konfiguracyjnym. Na stronie alertów widzimy, że alerty znajdują się w stanie **inactive**, co oznacza, że obecnie nie zostały spełnione warunki do ich aktywacji. Inne możliwe stany alertów to **pending**, co oznacza, że warunki alarmowe zostały spełnione, ale alert nie został jeszcze aktywowany, oraz **firing**, kiedy alert jest aktywny i został wyzwolony, ponieważ spełnione zostały warunki (np. przekroczenie określonego progu metryki). Alerty mogą zmienić swój stan na **firing** wtedy, gdy warunki opisane w regule, takie jak wysoka liczba zapytań czy obciążenie CPU, będą utrzymywały się przez określony czas. Gdy warunki wrócą do normy, stan alertu powróci do **inactive** lub **pending**, w zależności od tego, czy system wymaga ponownego sprawdzenia.



Rysunek 12 - sekcja alertów

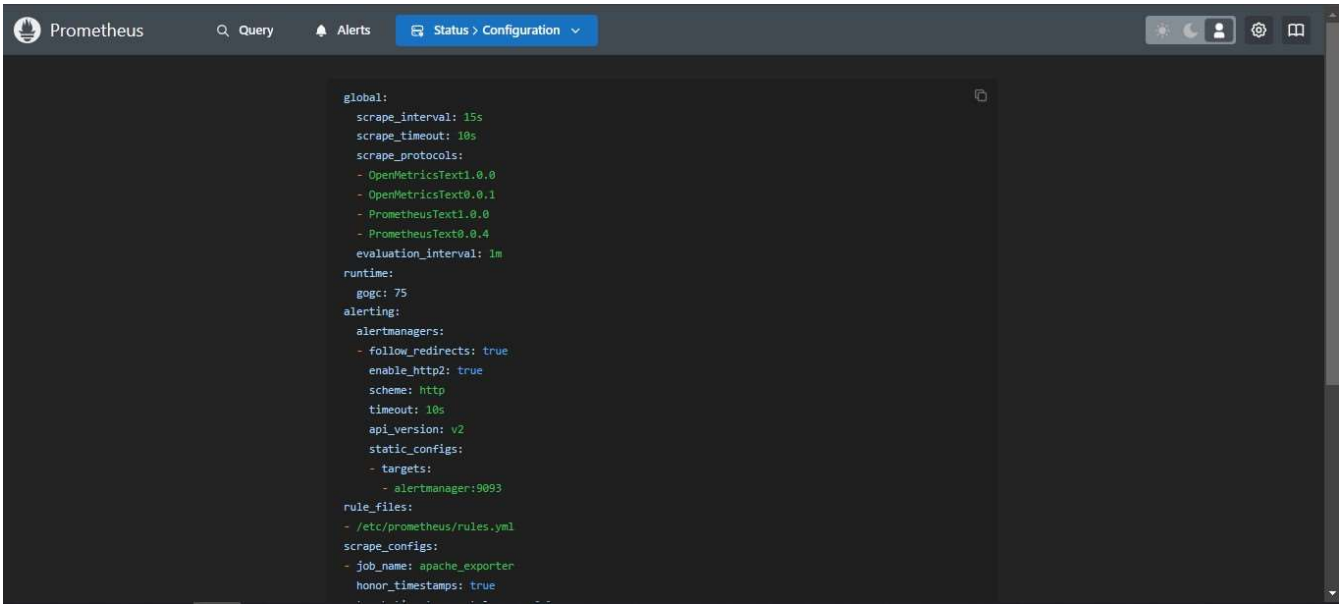
W sekcji **Target Health** Prometheusa możemy zobaczyć stan źródeł danych, w tym **apache_exporter** działający na porcie 9117, który zbiera metryki związane z działaniem serwera Apache, takie jak liczba zapytań czy dostępność serwera – jego implementacja zostanie omówiona później. Oprócz **apache_exporter**, mogą występować inne targety, takie jak eksportery do monitorowania systemu (np. **node_exporter** dla metryk systemowych), baz danych, aplikacji webowych czy usług kontenerowych, które Prometheus zbiera w celu monitorowania wydajności infrastruktury.



Rysunek 13 - sekcja Target Health

W sekcji **Configuration** Prometheusa możemy zobaczyć aktualną konfigurację systemu, w tym szczegóły dotyczące źródeł danych (scrape_configs), reguł alarmów (rule_files) oraz innych ustawień, takich jak interwały zbierania metryk czy konfiguracja alertów. Dzięki tej sekcji użytkownicy mogą

zweryfikować i dostosować konfigurację Prometheusa, aby zapewnić optymalne zbieranie danych i monitorowanie systemu zgodnie z wymaganiami projektu.



Rysunek 14 - widok konfiguracji Prometheusa

10. Implementacja apache_exportera

Apache_exporter to narzędzie zaprojektowane do zbierania metryk dotyczących działania serwera Apache, które są następnie przekazywane do Prometheusa w celu monitorowania i analizy. Prometheus domyślnie nie posiada wbudowanych metryk do zbierania informacji o serwerze webowym, takim jak **web_server**, ponieważ nie jest to jego główna funkcjonalność. Aby Prometheus mógł zbierać dane z tego typu źródła, musimy stworzyć odpowiednie metryki sami lub skorzystać z gotowych rozwiązań, jak np. **exportery**. Exportery to narzędzia, które działają jako pośrednik, zbierając metryki z różnych usług i aplikacji, a następnie udostępniają je Prometheusowi. Więcej informacji na temat exporterów oraz ich implementacji można znaleźć na oficjalnej stronie Prometheusa: https://prometheus.io/docs/instrumenting/exporters/

Listing 9 - implementacja kontenera apache_exportera

```
apache_exporter:
  container_name: apache_exporter
  ports:
    - "9117:9117"
  build:
    context: .
    dockerfile: ./apache_exporter/Dockerfile
  depends_on:
    - web_server
  restart: always
```

W projekcie **apache_exporter** jest uruchamiany w kontenerze Docker. Kontener został nazwany apache_exporter, co pozwala na łatwą identyfikację. Został przypisany port 9117, który jest mapowany na ten sam port na hoście, co umożliwia dostęp do metryk Apache udostępnianych przez exporter. Plik Dockerfile, który znajduje się w katalogu ./apache_exporter, jest używany do zbudowania obrazu kontenera, definiując środowisko i zależności potrzebne do uruchomienia eksportera. Kontener **apache_exporter** ma ustawioną zależność od kontenera **web_server**, co oznacza, że uruchomi się on dopiero wtedy, gdy serwer Apache będzie dostępny. Ponadto, kontener ma również włączony automatyczny restart.

Listing 10 - implementacja obrazu kontenera apache_exportera

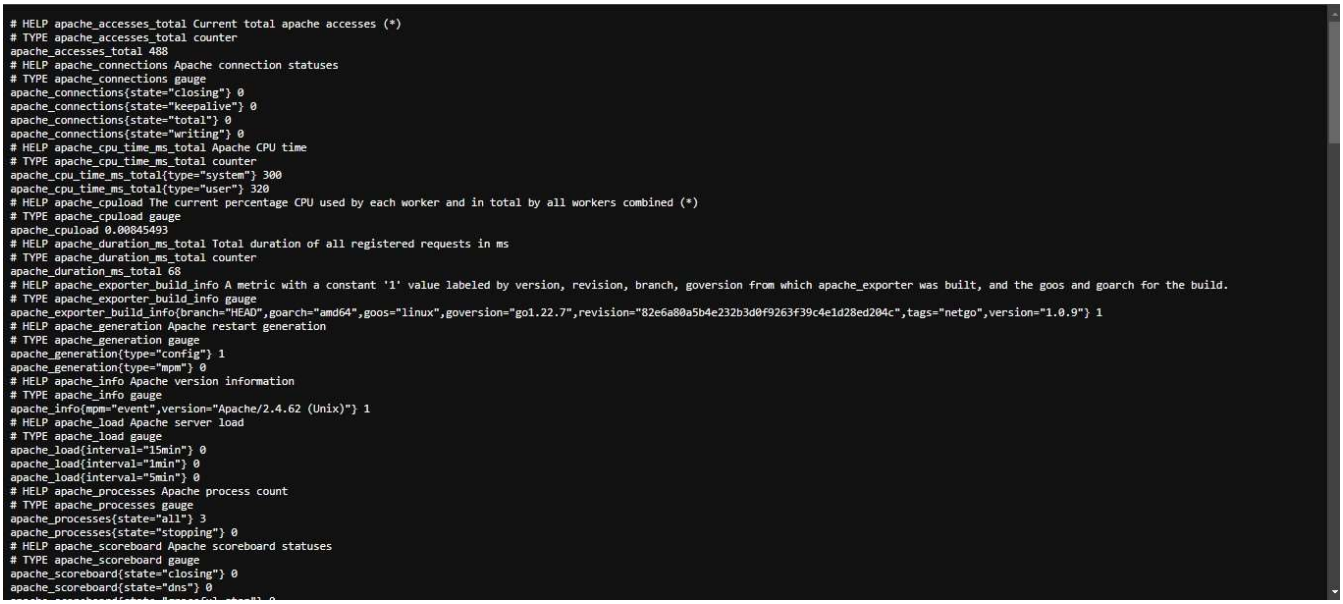
```
FROM ubuntu:22.04

RUN apt-get update && apt-get install -y wget tar curl

RUN wget
https://github.com/Lusitaniae/apache_exporter/releases/download/v1.0.9/apache_exporter
-1.0.9.linux-amd64.tar.gz && \
        tar xvzf apache_exporter-1.0.9.linux-amd64.tar.gz

EXPOSE 9117

CMD sh -c 'apache_exporter-1.0.9.linux-amd64/apache_exporter --
scrape_uri=http://web_server/server-status?auto'
```

Kontener ten został napisany, ponieważ w dostępnych zasobach internetowych nie udało się znaleźć gotowego obrazu, który w pełni odpowiadałby wymaganiom projektu. Istniejące obrazy często nie były wystarczająco dopasowane do specyficznych potrzeb, takich jak integracja z konkretną konfiguracją serwera Apache czy monitorowanie za pomocą **apache_exporter**. Z tego powodu podjęto decyzję o stworzeniu własnego kontenera, który dokładnie spełnia wymagania projektu i umożliwia zbieranie metryk z serwera Apache w sposób elastyczny, umożliwiając dalsze dostosowanie do zmieniających się potrzeb.

Ważną częścią tego kontenera jest odpowiednie ustawienie parametru **scrape_uri**, który wskazuje na adres, z którego **apache_exporter** zbiera metryki z serwera Apache. Parametr ten jest kluczowy, ponieważ pozwala na precyzyjne wskazanie lokalizacji, z której exporter pobiera dane w formacie kompatybilnym z Prometheusem, umożliwiając monitorowanie wydajności serwera w czasie rzeczywistym. W tym przypadku jest to adres y, który udostępnia metryki Apache w dopasowanym formacie.

Po wystartowaniu kontenera, na adresie http://localhost:9117/metrics przedstawione są różne informacje związane z działaniem serwera Apache. Przykładowe metryki, które możemy tam zobaczyć,

to apache_accesses_total, który pokazuje łączną liczbę dostępów do serwera, apache_connections, który przedstawia statusy połączeń Apache, czy apache_cpu_time_ms_total, który mierzy zużycie CPU przez Apache.



Rysunek 15 - stan metryk apache_exportera

11. Implementacja Alert Managera

Alertmanager to narzędzie służące do zarządzania alertami w Prometheusie. Jego głównym zadaniem jest odbieranie powiadomień o alertach z Prometheusa, grupowanie ich, tłumaczenie na odpowiednie formaty oraz wysyłanie do zewnętrznych systemów powiadamiania, takich jak e-mail, Slack itp. Alertmanager umożliwia konfigurację reguł powiadamiania, zarządzanie stanem alertów i ich eskalacją w zależności od priorytetu.

Listing 11 - implementacja kontenera alertmanager

```
alertmanager:
  container_name: alertmanager
  image: prom/alertmanager
  depends_on:
    - prometheus
  ports:
    - '9093:9093'
  volumes:
    - ./alertmanager:/etc/alertmanager
  command:
    - '--config.file=/etc/alertmanager/alertmanager.yml'
  restart: always
```

W projekcie kontener **alertmanager** uruchamia oficjalny obraz **prom/alertmanager**, który jest odpowiedzialny za obsługę i zarządzanie alertami. Kontener zależy od uruchomienia **Prometheusa** i jest dostępny na porcie 9093, co pozwala na dostęp do interfejsu zarządzania alertami. W kontenerze

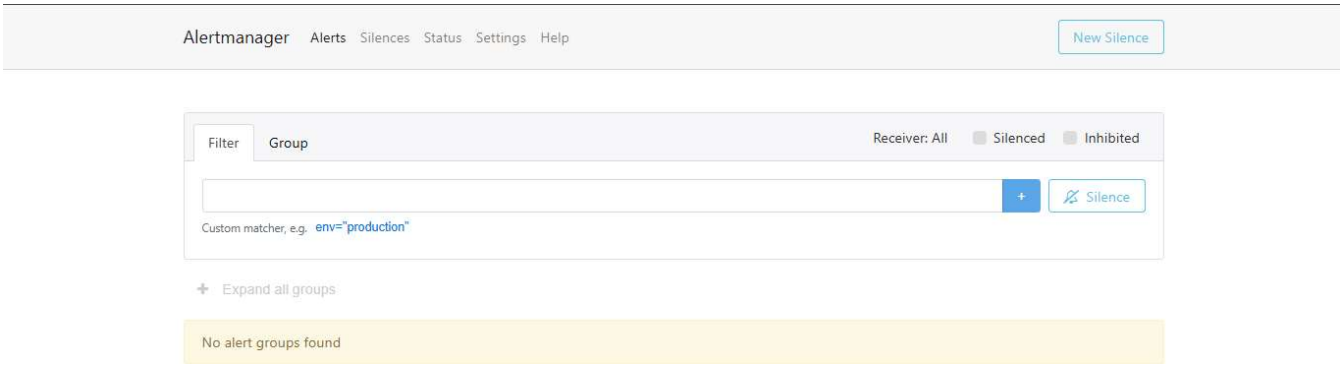
przekazywana jest konfiguracja **alertmanager.yml**, która definiuje szczegóły dotyczące powiadomień, takich jak odbiorcy, reguły grupowania alertów i inne ustawienia powiadamiania. Pliki konfiguracyjne alertmanagera są zamontowane jako wolumen z lokalnego katalogu **./alertmanager**, co umożliwia łatwą edycję i aktualizację konfiguracji. Kontener jest również ustawiony na automatyczny restart.

Listing 12 - plik konfiguracyjny alert managera (alertmanager.yml)

```yaml
global:
  resolve_timeout: 5m

route:
  receiver: 'email-notification'

receivers:
- name: 'email-notification'
  email_configs:
  - smarthost: 'smtp.gmail.com:587'
    from: 'krzysiek.kolodziej23@gmail.com'
    to: 'krzysiek.kolodziej23@gmail.com'
    auth_username: 'krzysiek.kolodziej23@gmail.com'
    auth_password: #########
    send_resolved: true
    require_tls: true
    headers:
      subject: 'Prometheus Mail Alerts'

inhibit_rules:
- source_match:
    severity: 'critical'
  target_match:
    severity: 'warning'
  equal: ['alertname', 'dev', 'instance']
```

Konfiguracja Alertmanagera określa, jak zarządzać powiadomieniami i ich dostarczaniem. **resolve_timeout** ustawione na 5 minut oznacza, że alerty zostaną uznane za rozwiązane po tym czasie, jeśli stan alertu przestanie występować. W sekcji **route** przypisano domyślny odbiornik powiadomień, którym jest "email-notification". To oznacza, że wszystkie alerty, które zostaną wygenerowane, będą wysyłane na przypisany adres e-mail. W sekcji **receivers** zdefiniowano odbiornik e-mail, który jest skonfigurowany do wysyłania powiadomień na adres **krzysiek.kolodziej23@gmail.com**. Powiadomienia będą wysyłane za pomocą serwera SMTP Gmaila, przy użyciu podanych danych uwierzytelniających. Dodatkowo, opcja **send_resolved** oznacza, że Alertmanager wyśle powiadomienie także wtedy, gdy alert zostanie rozwiązany, a **require_tls** zapewnia szyfrowane połączenie przy wysyłaniu maili. W sekcji **inhibit_rules** zdefiniowano regułę, która zapobiega wysyłaniu powiadomień o alertach o mniejszej wadze (np. o statusie "warning"), gdy istnieje już alert o wyższej wadze (np.

"critical") dotyczący tego samego zasobu. Reguła ta zapobiega nadmiernej liczbie powiadomień, jeśli dotyczą one tego samego problemu.

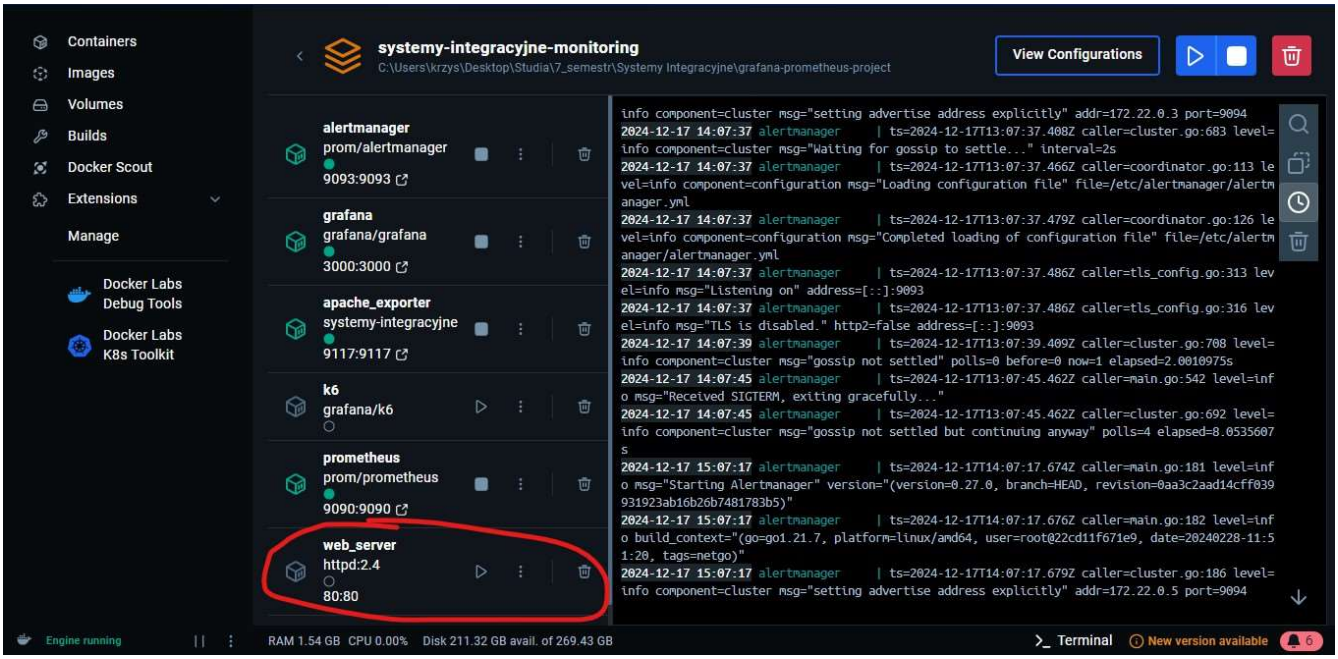


Rysunek 16 - strona alert manager

Na stronie **http://localhost:9093** wyświetlana jest głównie lista aktywnych alertów oraz stanów powiadomień. W przypadku, gdy żadne alerty nie zostały jeszcze uruchomione, strona jest pusta i nie pokazuje żadnych powiadomień. Alerty pojawią się dopiero, gdy spełnione zostaną określone warunki w monitorowanych usługach, takich jak na przykład spadek dostępności serwera Apache. W przypadku, gdy serwer Apache przestanie działać, zostanie uruchomiony odpowiedni alert, który pojawi się na stronie Alertmanagera. Przykład takiej sytuacji zostanie zaprezentowany w następnym rozdziale.

12. Przykład działania systemu powiadomień

W ramach projektu, system powiadomień generuje alerty w przypadku wystąpienia problemów, takich jak awaria serwera. Po skonfigurowaniu Alertmanagera do wysyłania powiadomień na adres e-mail, użytkownik jest informowany o krytycznych problemach w systemie, takich jak niedostępność serwera Apache. Zatrzymajmy serwer webowy:

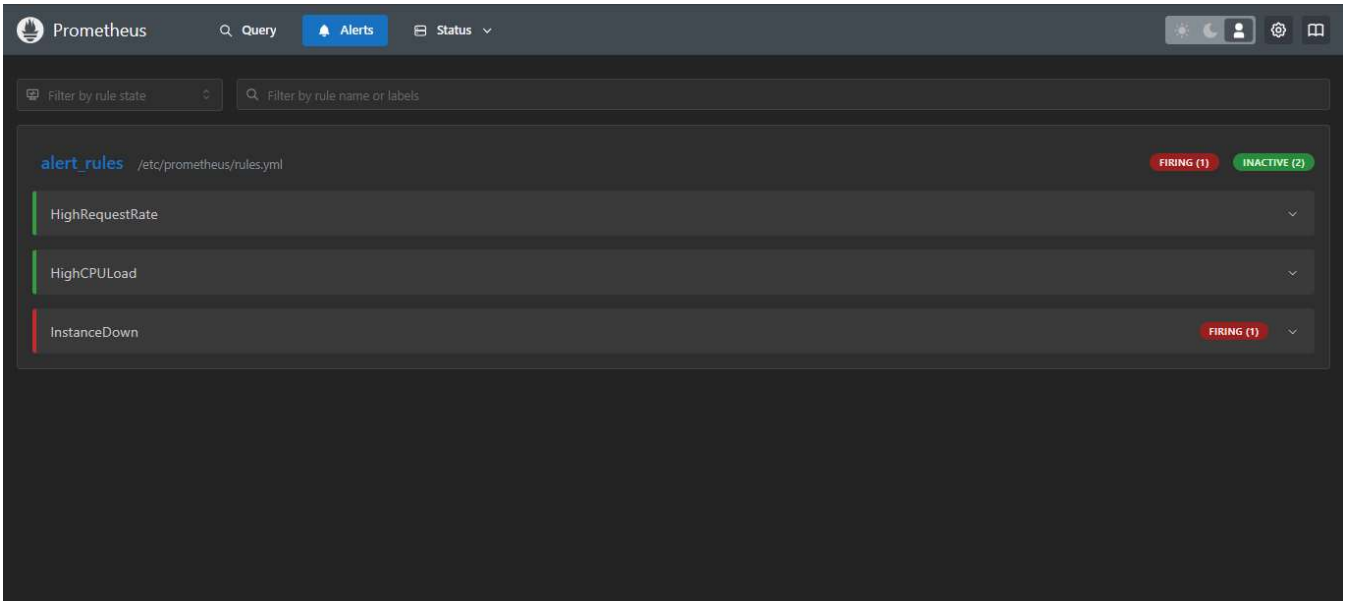


Rysunek 17 - zatrzymanie serwera webowego

Efekt zatrzymania serwera webowego możemy od razu zauważyć na Grafanie:

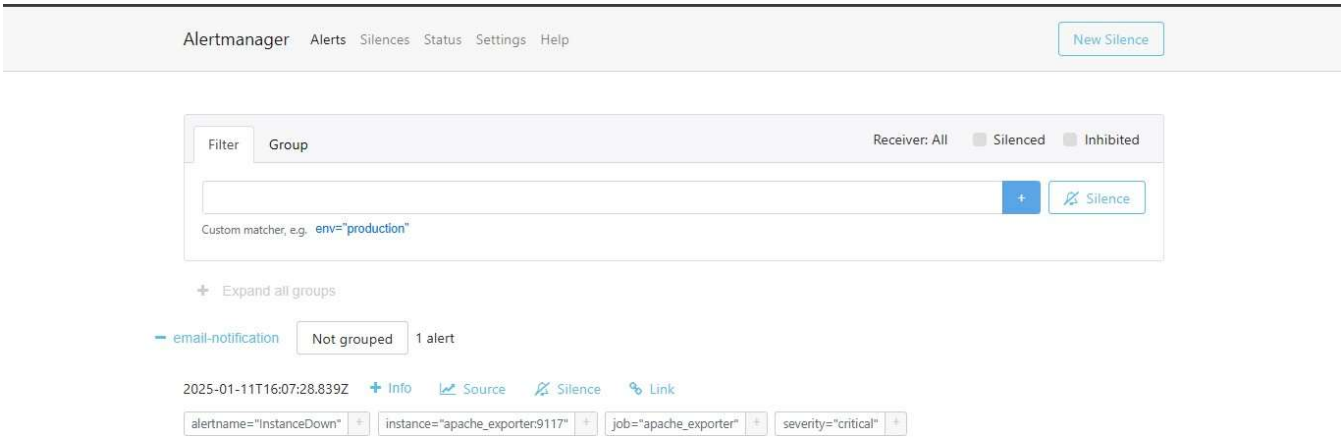
Rysunek 18 - efekt zatrzymania serwera webowego na Grafanie

Po upływie określonego czasu, w którym instancja serwera webowego nie działa, Prometheus decyduje się na zmianę stanu alertu w „Firing”:



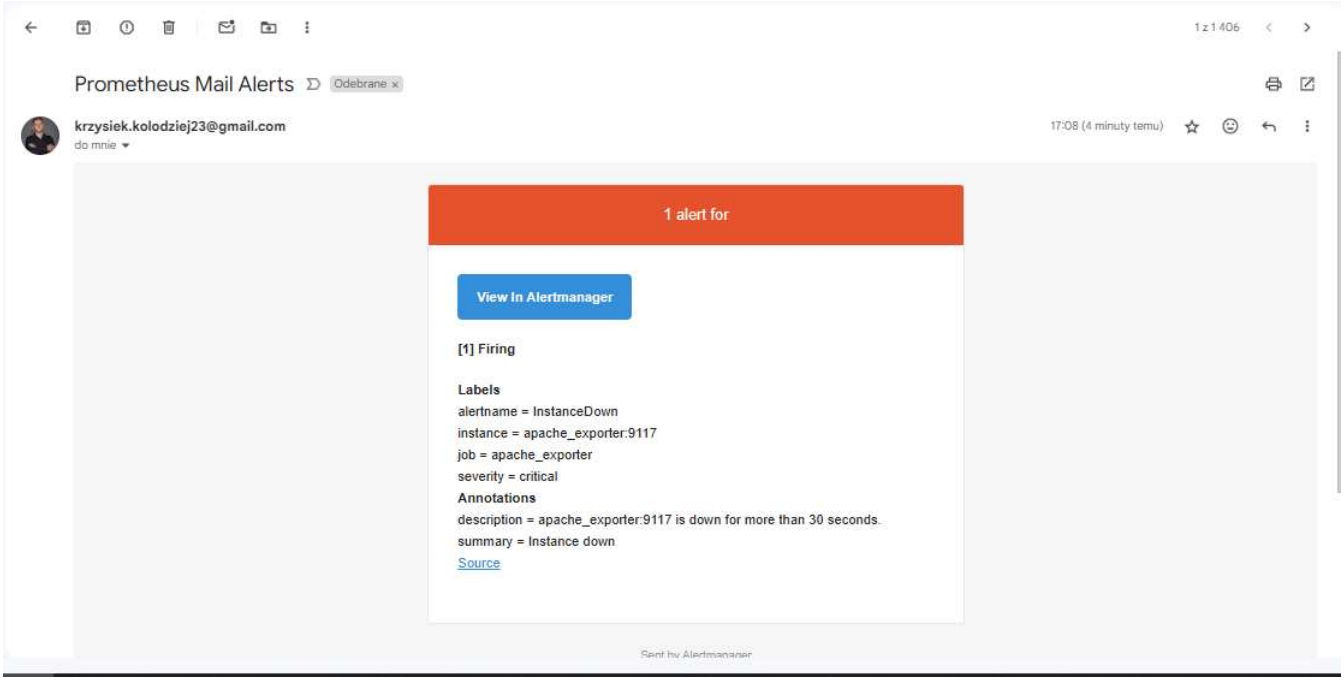
Rysunek 19 - stan uruchomionego alartu w Prometheusie

System powiadomień wysyła maila na określoną pocztę:



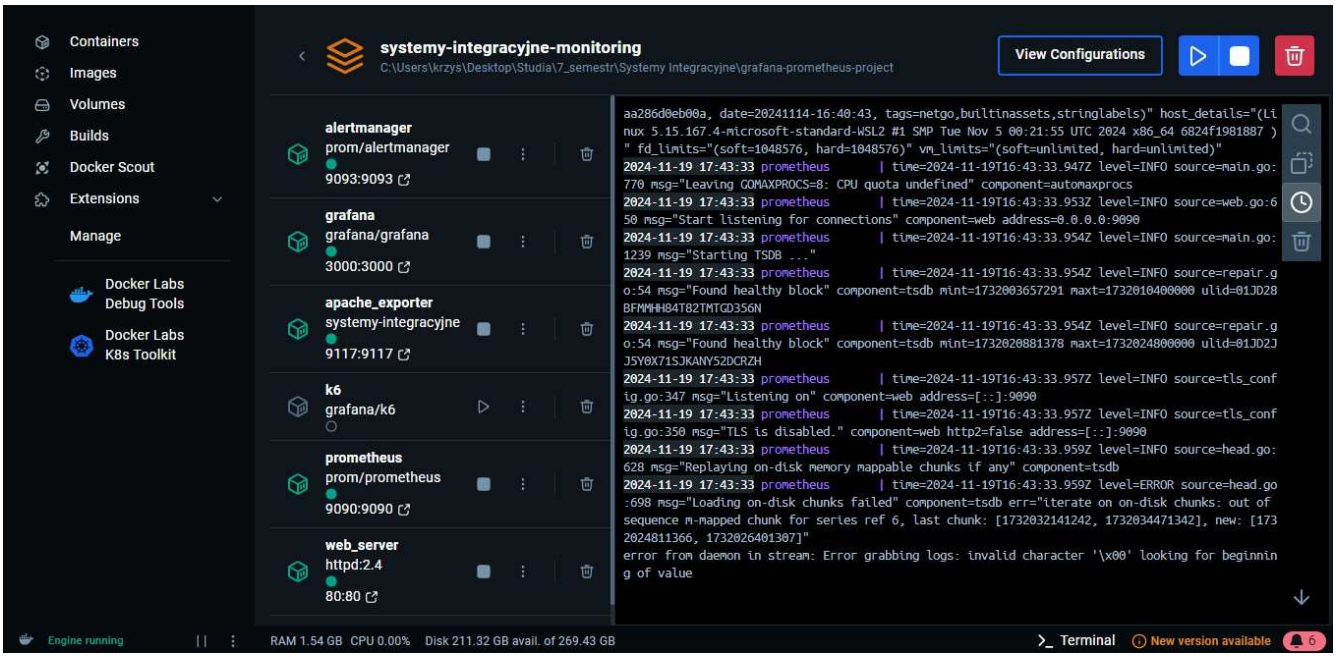
Rysunek 20 - wysłanie alertu przez alert managera na pocztę gmail

Wiadomość e-mail informuje o wystąpieniu krytycznego alertu **InstanceDown**, który wskazuje, że instancja **apache_exporter:9117** jest niedostępna przez ponad 30 sekund. W treści wiadomości zawarte są szczegóły dotyczące problemu, w tym nazwa instancji, typ alertu oraz czas jego trwania, co pozwala na szybsze zidentyfikowanie i rozwiązanie problemu.



Rysunek 21 - alert o wyłączonym serwerze webowym

Włączmy ponownie kontener z serwerem webowym:

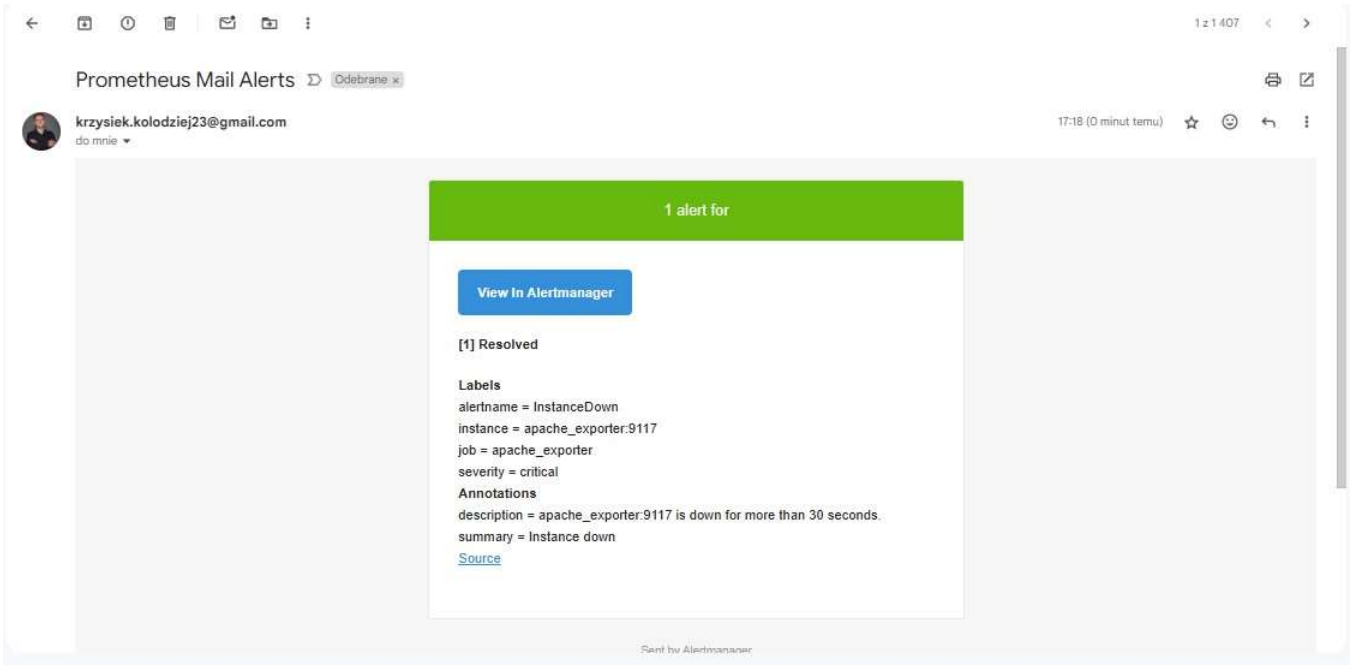


Rysunek 22 - zrestartowanie serwera webowego

Na Grafanie możemy zauważyć, że nasz serwer już działa:

Rysunek 23 - powrót działania serwera webowego na Grafanie

Po upływie określonego czasu, na pocztę dostajemy powiadomienie, że stan naszego serwera webowego jest już w porządku:



Rysunek 24 - powiadomienie o działającym serwerze webowym

13. Implementacja K6

K6 to narzędzie wykorzystywane do przeprowadzania testów wydajnościowych, które umożliwia symulację obciążenia na serwerze, generując dużą liczbę zapytań HTTP. W projekcie, kontener z k6 zostanie uruchomiony, aby przeprowadzić testy wydajnościowe na serwerze Apache, generując różne poziomy ruchu, aby sprawdzić, jak system reaguje na obciążenie.

Listing 13 - implementacja kontenera k6 (docker-compose.yml)

```
    k6:
    image: grafana/k6
    container_name: k6
    volumes:
      - ./k6:/scripts
    entrypoint: ["k6", "run", "--summary-trend-stats=med,p(95),p(99.9)",
"/scripts/load_test.js"]
    depends_on:
      - web_server
```

Kontener został zaimplementowany przy użyciu obrazu **grafana/k6**, a jego zadaniem jest uruchomienie skryptu **load_test.js** po starcie kontenera. Skrypt ten znajduje się w katalogu **./k6** na hoście i jest montowany do kontenera. Po uruchomieniu kontenera, k6 wykonuje testy, podczas których stopniowo zwiększa liczbę równoczesnych zapytań HTTP do serwera **web_server**. Kontener będzie działał do momentu zakończenia testu, a po jego zakończeniu automatycznie się zatrzyma.

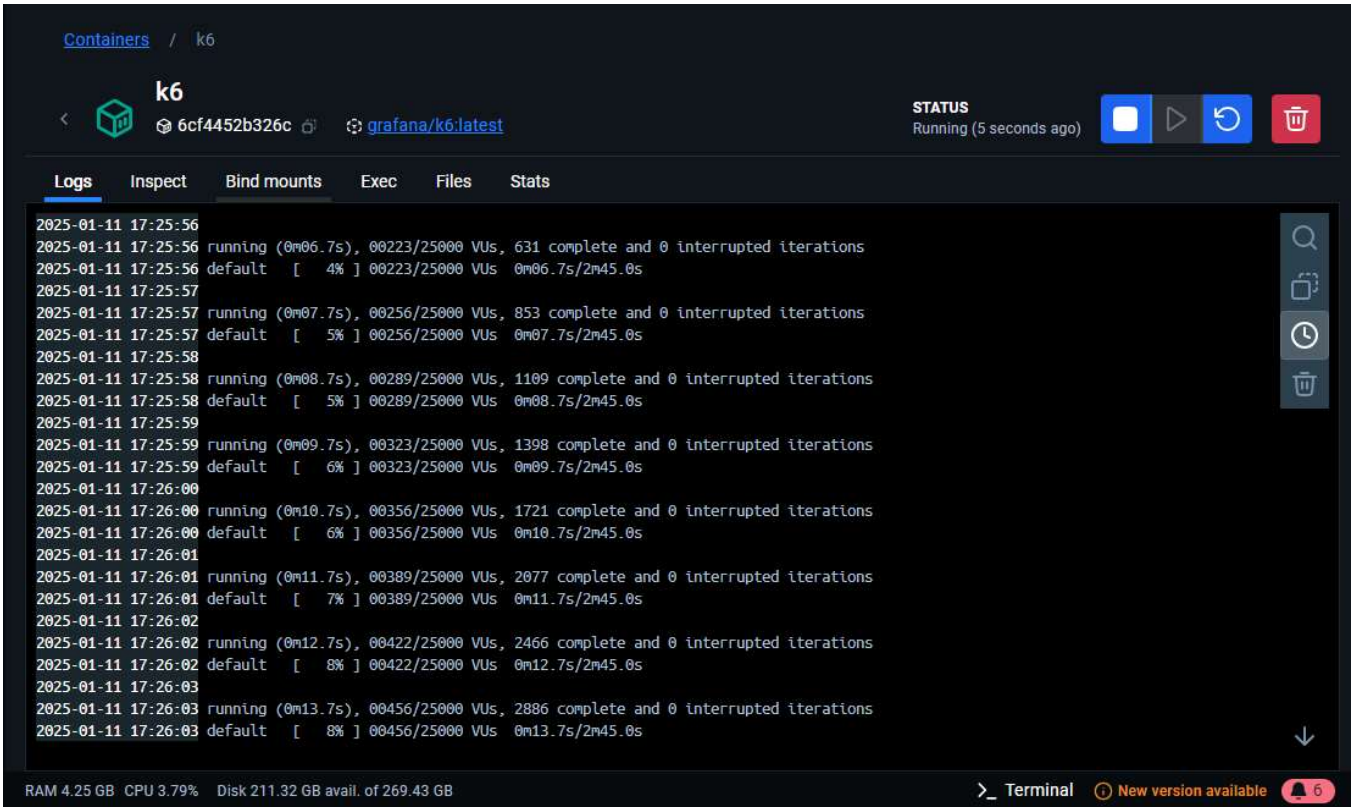
Listing 14 - implementacja testów wydajnościowych w k6 (load_test.js)

```
import http from 'k6/http';
import { sleep } from 'k6';

export let options = {
    stages: [
        { duration: '15s', target: 500 },
        { duration: '15s', target: 5000 },
        { duration: '15s', target: 25000 },
        { duration: '2m', target: 7500 },
    ],
};

export default function () {
    http.get('http://web_server/');
    sleep(1);
}
```

Plik **load_test.js** definiuje etapy testu obciążeniowego. Na początku test generuje 500 zapytań na sekundę, a następnie stopniowo zwiększa tę liczbę do 5000, potem 25000, aby po osiągnięciu 7500 zapytań na sekundę przez 2 minuty, zakończyć test. W każdym cyklu k6 wykonuje zapytanie HTTP do serwera, po czym następuje krótka przerwa, aby symulować rzeczywiste warunki użytkowania serwera.

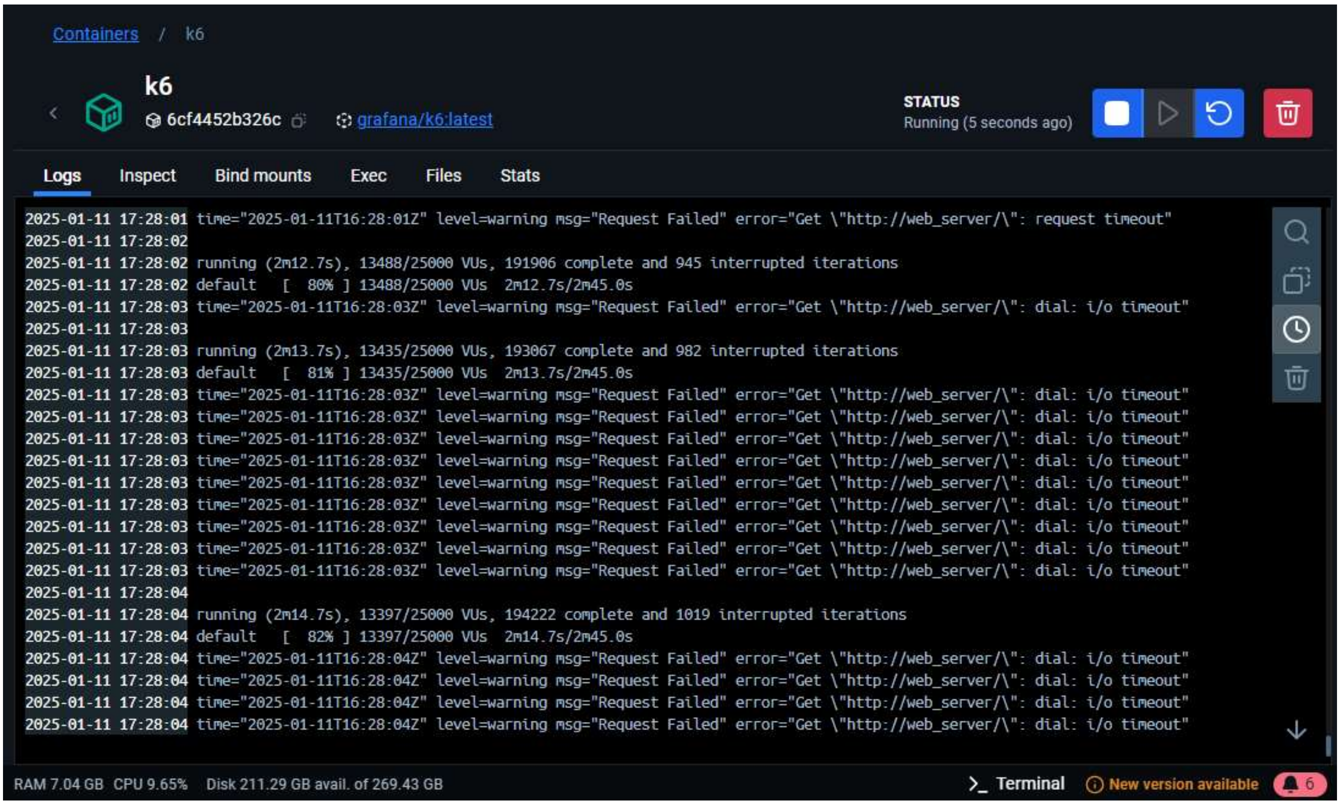
14. Przeprowadzenie testów wydajnościowych

Przeprowadzanie testów wydajnościowych jest kluczowym elementem oceny stabilności i efektywności systemu informatycznego, szczególnie w kontekście serwerów webowych. Testy te pozwalają na symulację dużego ruchu w sieci, co umożliwia określenie, jak system reaguje na różne poziomy obciążenia i identyfikację potencjalnych wąskich gardeł w jego wydajności. Na początku uruchamiamy kontener k6:



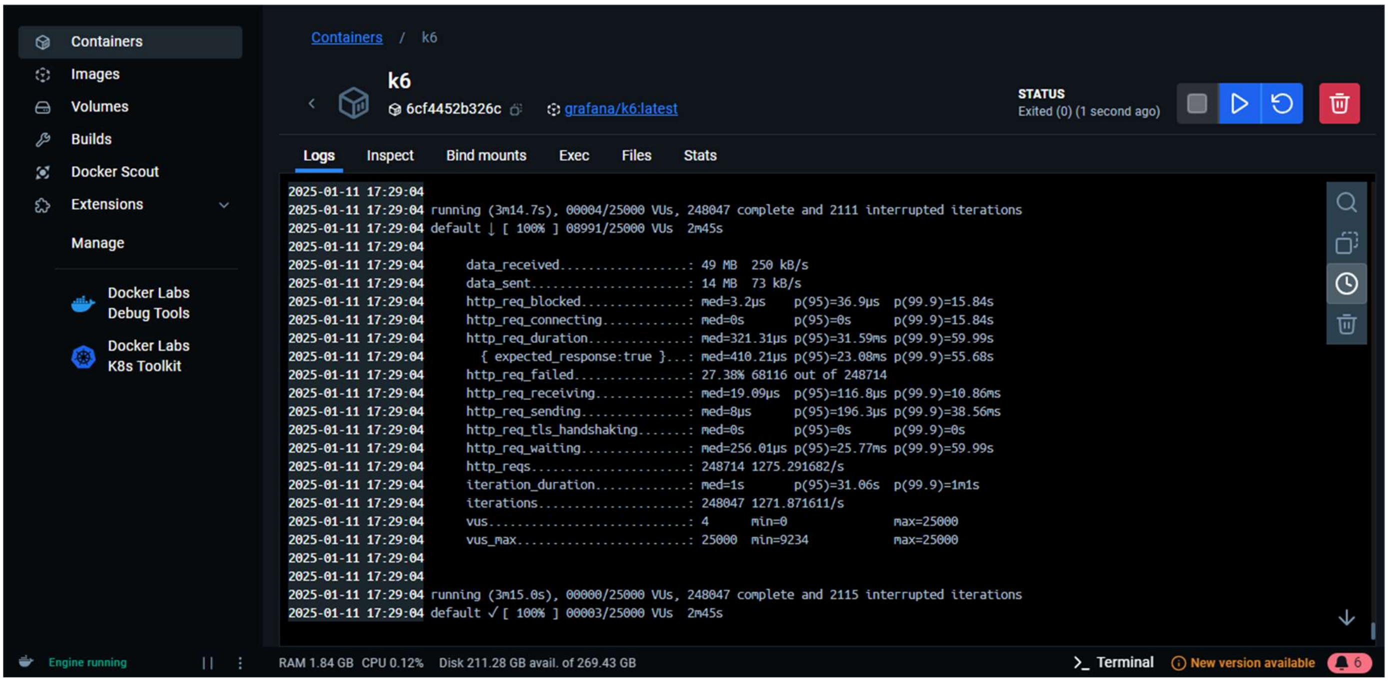
Rysunek 25 - uruchomienie kontenera k6

Większa ilość wysyłanych zapytań na raz powoduje, że nasz serwer notuje przestoje:



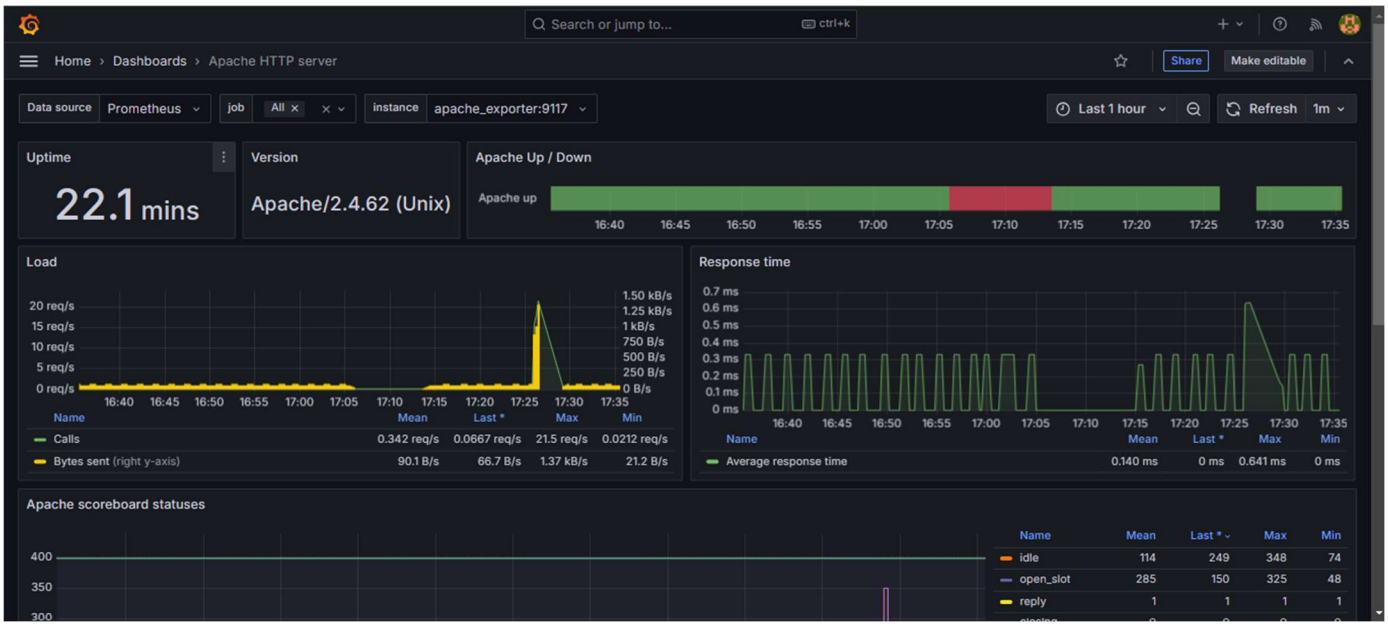
Rysunek 26 – przeciążenie serwera webowego

Ostatecznie otrzymujemy raport:

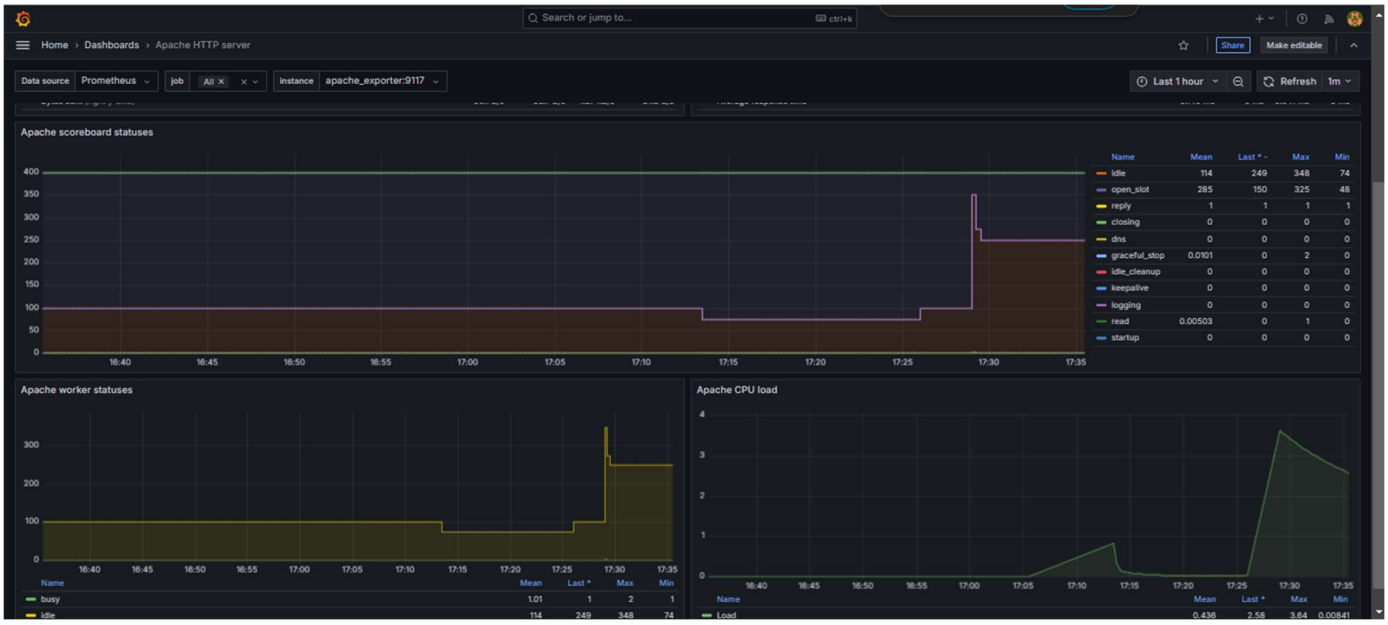


Rysunek 27 – raport z testów wydajnościowych

Stan serwera podczas testów wydajnościowych możemy również przeanalizować na Grafanie:



Rysunek 28 – stan serwera poczas testów wydajnościowych na Grafanie część 1



Rysunek 28 – stan serwera poczas testów wydajnościowych na Grafanie część 2

15. Podsumowanie

Zastosowanie narzędzi takich jak Prometheus, Grafana, Alertmanager i k6 do monitorowania i testowania wydajności serwera Apache w tym projekcie okazało się bardzo przydatne. System umożliwia nie tylko wizualizację metryk, ale także aktywne monitorowanie stanu infrastruktury, dzięki czemu możliwe jest szybkie reagowanie na potencjalne problemy za pomocą alertów. Alerty informują o krytycznych sytuacjach, jak np. zniknięcie instancji serwera, co pozwala na szybkie podjęcie działań naprawczych. Testy wydajnościowe przeprowadzone za pomocą k6 ujawniły, że serwer Apache ma trudności z obsługą dużego ruchu generowanego przez liczne zapytania w krótkim czasie, co prowadzi do jego przeciążenia i wyłączenia. Problemy te wystąpiły jednak dopiero przy ogromnych ilościach zapytań na raz, co jest raczej mało prawdopodobne w standardowych warunkach operacyjnych, chyba że w przypadku bardzo dużych projektów, z których korzystałoby wiele użytkowników jednocześnie. To pokazuje, jak istotne jest ciągłe monitorowanie i testowanie obciążenia, aby zapewnić stabilność i optymalną wydajność systemu w rzeczywistych warunkach produkcyjnych.

Bibliografia

- *Alertmanager*. (2025). Pobrano z lokalizacji https://prometheus.io/docs/alerting/latest/alertmanager/
- *Apache exporter*. (2025). Pobrano z lokalizacji https://github.com/Lusitaniae/apache_exporter
- *Oficjalna dokumentacja Grafany*. (2025). Pobrano z lokalizacji https://grafana.com/docs/
- *Oficjalna dokumentacja K6*. (2025). Pobrano z lokalizacji https://k6.io/
- *Oficjalna dokumentacja Prometheusa*. (2025). Pobrano z lokalizacji https://prometheus.io/docs/introduction/overview/