TrashTracker - optymalizacja wywozu śmieci nigdy nie była przyjemniejsza

Dokumentacja projektowa

Dawid Radkowski, Bartosz Piekarczyk, Tomasz Jarząbek

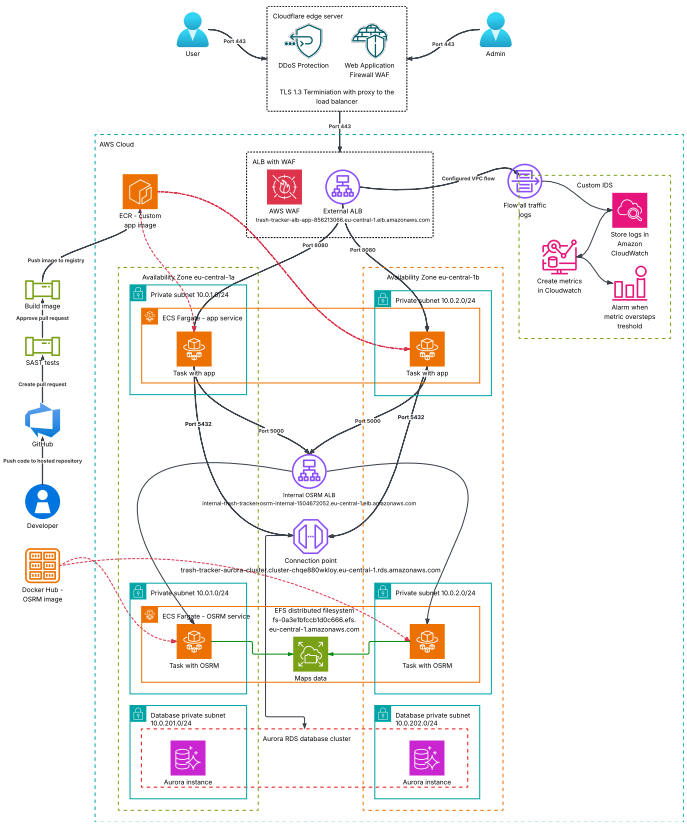
30.11.2025

Spis treści

1 Cel

Celem aplikacji jest wspomaganie procesu planowania tras przejazdu dla pojazdów odpowiedzialnych za odbiór odpadów komunalnych na terenie Wrocławia. Aplikacja umożliwia wyznaczenie najkrótszej możliwej trasy pomiędzy zestawem określonych punktów (np. lokalizacjami pojemników na odpady), z zachowaniem warunku odwiedzenia każdego punktu dokładnie raz oraz powrotu do punktu początkowego. Tego typu optymalizacja pozwala na znaczące ograniczenie czasu przejazdu oraz kosztów eksploatacyjnych pojazdów. W przyszłości aplikacja może być rozwijana o dodatkowe funkcjonalności, takie jak uwzględnianie dynamicznych warunków drogowych (np. korki, zamknięcia ulic), planowanie tras dla wielu pojazdów jednocześnie (rozszerzenie do problemu VRP – Vehicle Routing Problem), integrację z systemami GPS oraz analizę kosztów logistycznych. Możliwa jest również rozbudowa systemu o moduł raportowania i monitorowania efektywności realizowanych tras.

2 Architektura aplikacji



Rysunek 1: Infrastruktura aplikacji

Architektura opracowana z myślą o aplikacji, pozwala na wszechstronne i skuteczne dostosowanie do warunków, zarówno pod względem skalowalności, dostępności, jak i bezpieczeństwa. Jako chmurę,

zdecydowano się na Amazon Web Services, gdzie przy wykorzystaniu narzędzia Infrastructure as Code – Terraform, stworzono wszystkie zasoby. Rozklowowano je pomiędzy dwie AZ[1] tworząc w każdej z nich wymagane podsieci. Jak widać na schemacie, bramą wejścia do całej aplikacji jest Cloudflare, opisany w sekcji 4.

Następnym punktem jest ALB[2], zintegrowany z samo-zarządzanym Web Application Firewall. Zasady wykrywania ataków zostały własnoręcznie utworzone, co opisano w sekcji 7. Dzięki zastosowaniu kontroli ruchu, jedynie ruch bezpośrednio z Cloudflare jest ackeptowany.

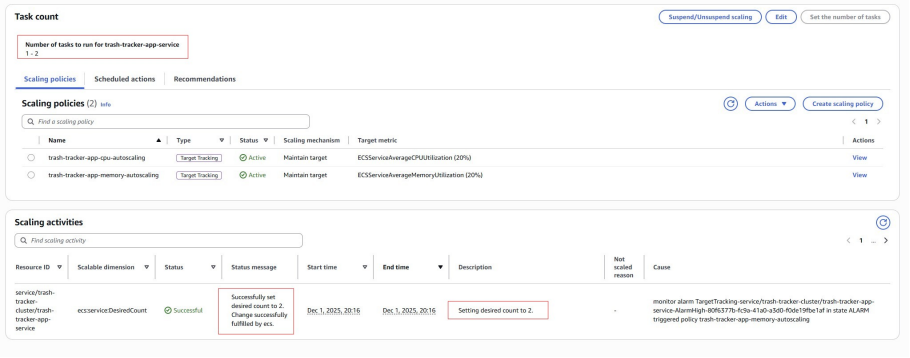
Jako IDS wykonano własną koncepcję nie obejmującą żadnej ze znanych typów zasobów takich jak IaaS czy PaaS. Wszystkie logi z VPC[3] są przesyłane do zasobu Cloudwatch, gdzie tworzone są na ich podstawie metryki. Następnie alarmy ustawione, aby monitorować przekroczenie konkretnych metryk, informują o potencjalnym ataku. Niestety napotkano pewne problemy z nieprzechwytywaniem pełnego ruchu, jednak nadanie minimalnych progów poskutkowało alarmowaniem o na przykład potencjalnym skakowaniu portów.

Główny ALB przekazuje ruch do zasobu ECS[4] Fargate, który służy jako samo-zarządzany klaster do hostowania kontenerów. Na podstawie Task Definition, będącego formą deklaratywnego opisu utrzymywanej jednostki, zasób ten pobiera wymagane obrazy, rozdziela zasoby i uruchamia kontenery. W przypadku aplikacji, obraz jest pobierany z ECR[5], w którym utrzymywana jest retencja i przeprowadzane są dodatkowe skany. Dodatkowo zastosowano skalowanie kontenerów, które w przypadku przekroczenia konkretnego zużycia pamięci lub CPU, tworzy automatycznie dodatkowe instancje.

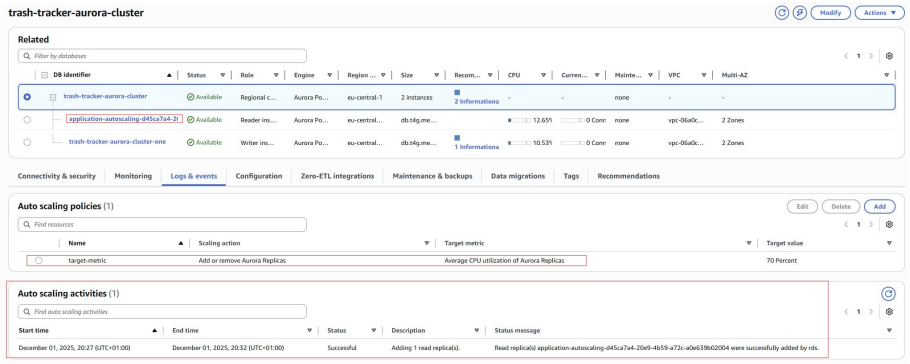
Następnie aplikacja komunikuje się z dwoma zasobami. Pierwszy to wewnętrzny ALB służący do rozdzielania ruchu pomiędzy kontenery OSRM[6], których obrazy są pobierane z Docker Hub. Aby sprostać wymaganiom dostępu do danych o mapach, utworzono rozproszony system plików, który jest montowany jako wolumen danych do każdego kontenera. Drugi zasób, z którym aplikacja może się komunikować, jest klaster bazy danych. Zastosowano bazę danych z silnikiem PostreSQL w systemie PaaS. Umożliwia ona utworzenie instancji głównej służącej do zapisu danych, oraz instancje z replikowanymi danymi do odczytu w innych AZ. Baza danych zapewnia automatyczne kopie zapasowe w postaci snapshot oraz auto skalowanie w momencie przekroczenia ustawionej granicy zużycia CPU.

Skutecznie udało się wywołać autoskalowanie zarówno aplikacji, jak i bazy danych, co zostało ukazane na zrzutach 2 i 5.

[1] Availability Zone
[2] Application Load Balancer
[3] Virtual Private Cloud
[4] Elastic Container Service
[5] Elastic Container Registry
[6] Open Street Routing Machine

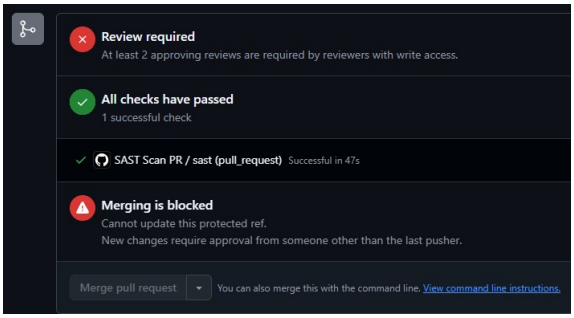


Rysunek 2: Skuteczne wyskalowanie aplikacji



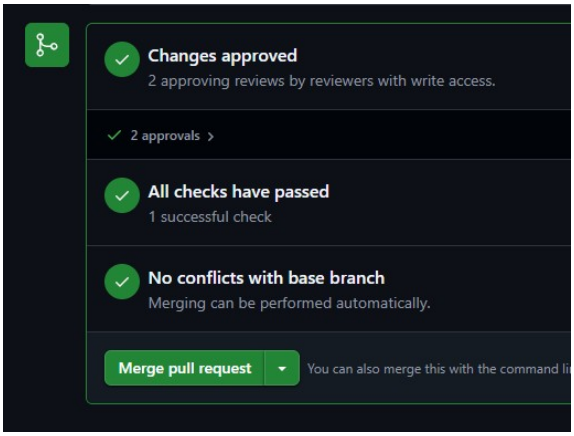
Rysunek 3: Skuteczne wyskalowanie bazy danych

Wdrożenie aplikacji, włączenie ze zbudowaniem obrazu aplikacji, przeprowadzeniem testów, oraz dokonaniem zmian w AWS, osiągnięte jest przy pomocy GitHub Actions. Główny pipeline, deploy.yaml, jest uruchamiany jedynie po zamknięciu PR[7] do branch prod. Do tego branch możliwe jest tylko wykonanie merge z brancha main, poprzez otworzenie PR, wykonanie testów bezpieczeństwa i zatwierdzenie zmian przez dwóch recenzentów. Ukazano to na poniższym zdjęciu. Testy SAST są wykonywane dla każdej aktualizacji PR do branch main i prod.



Rysunek 4: PR oczekuje na zatwierdzenie, SAST wykonany pomyślnie

[7]Pull request



Rysunek 5: Zezwolenie na merge PR

Dodatkowo w celu uniknięcia niepożądanych zmian na infrastrukturze, skonfigurowano wymagane zatwierdzenie Terraform Plan, które zatrzymuje pipeline. Po akceptacji, wygenerowany plan jest pobierany w etapie Apply i używano jako źródło zmian.

3 Stos technologiczny

- **Django** - website framework, który pozwala na kompleksowe tworzenie aplikacji internetowych. Nie polega na klasycznym modelu frontend - backend, lecz wypełnia wygenerowane pliki wizualne danymi zaciągniętymi z bazy danych. Zapewnia także przekierowania pomiędzy stronami i zarządzanie formularzami danych.

- **Django Rest Framework** - jest to framework odpowiedzialny za obsługę konkretnych zapytań dotyczących danych. Pozwala na dynamiczne dodawanie, usuwanie czy aktualizowanie punktów na stronie bez jej odświeżania.

- **Docker** - wykorzystany do uruchomienia bazy danych zawierającej rekordy tras, punktów oraz dla działania aplikacji OSRM. Zastosowany w celu konteneryzacji usług projektu, co pozwoliło na ich niezależne uruchamianie, łatwe wdrażanie i jednakowe środowiska deweloperskie.

- **OSRM** - OpenSource Routing Machine - stosowana do wyznaczania tras wzdłuż faktycznych dróg w terenie, oparta na danych OpenStreetMap. Pozwalała na realistyczne i praktyczne odwzorowanie tras w terenie.

- **Folium** - biblioteka Pythona oparta na Leaflet.js używana w tworzeniu interaktywnych map na stronie wraz z zaznaczaniem punktów oraz kreśleniem obliczonych ścieżek w celu poprawnej i wygodnej wizualizacji.

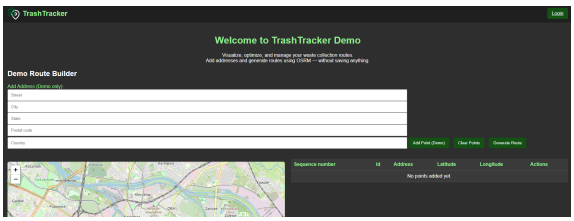
- **JavaScript** - wykorzystywany do obsługi dynamicznych elementów interfejsu użytkownika, umożliwiający interakcję z mapą, formularzami oraz elementami strony bez potrzeby przeładowywania widoku.

4 Cloudlfare

Cloudflare jest globalną siecią CDN oraz usługą bezpieczeństwa działającą jako warstwa pośrednia pomiędzy użytkownikiem a serwerem aplikacji, poprawiając zarówno wydajność, jak i ochronę serwisu. Jednym z jego kluczowych zadań jest zabezpieczanie aplikacji przed atakami DDoS poprzez filtrowanie ruchu jeszcze zanim dotrze on do właściwego serwera, analizę behawioralną zapytań oraz wykorzystanie globalnej infrastruktury do rozpraszania nadmiernego obciążenia. Dzięki temu nawet duże i nagłe wolumeny niepożądanego ruchu są skutecznie blokowane, a serwis pozostaje dostępny. Dodatkowo Cloudflare umożliwia korzystanie z zaufanej domeny publicznej, zapewniając szybkie i bezpieczne DNS, automatyczne certyfikaty TLS1.3, umożliwiające użycie szyfrowanego połączenia oraz gwarancję, że użytkownik łączy się z właściwym, zweryfikowanym źródłem.

5 Interfejs użytkownika

Aplikacja działa na domenie: `https://trash-tracker.pl/`
Interfejs graficzny, z którym użytkownik zachodzi w interakcję, składa się z kilku stron spełniających odpowiednie funkcje i zadania. Każda strona zawiera logo aplikacji wraz z nazwą, które służą jako odnośnik do strony głównej. Po prawej stronie widnieje przycisk Login, kierujący gościa na stronę logowania. Strona domowa jest zawarta na Rysunku 6.



Rysunek 6: Strona domowa aplikacji.

Strona główna pozwala gościom na przetestowanie głównej funkcji aplikacji w wersji ograniczonej, by zachęcić ich do utworzenia konta i odblokowania wszystkich możliwości. Gość może dodać 3 punkty na mapie i wytyczyć zoptymalizowaną ścieżkę między nimi. Przy każdym generowaniu ścieżki wyświetla się monit zachęcający do założenia konta w systemie.
Zakładka logowania pozwala na wprowadzenie danych uwierzytelniających oraz stworzenie nowego konta. Po zalogowaniu użytkownik przenoszony jest do swojej własnej strony, gdzie znajdują się

jego zapisane ścieżki z opcją stworzenia nowych. Nagłówek strony zostaje zmieniony, pojawiają sie dodatkowe informacje: obecnie zalogowany użytkownik oraz przyciski: zmiana hasła i wylogowanie. Wylogowanie użytkownika wysyła go do strony startowej.

Na stronie tworzenia ścieżki dodaje się punkty na mapie za pomocą wprowadzania dokładnych adresów (ulica, numer, kod pocztowy, miasto, państwo). Widać na szczycie strony nazwę obecnie edytowanej trasy, pod nią miejsce do wprowadzania danych. Punkty dodawane są do tabeli poniżej oraz do mapy.

W tabeli dostępna jest również opcja usunięcia wybranego punktu. Wytypowane kolumny to: *ID, adres, szerokość geograficzna, długość geograficzna*. Powracając na stronę ścieżek użytkownika, widoczna jest nowa trasa na liście wymienionych wszystkich tras. Po naciśnięciu jej, użytkownik prowadzony jest na stronę zawierającą jej detale.

Widoczna jest nazwa trasy, a obok niej podany został szacowany czas przejazdu oraz całkowity dystans. Poniżej umieszczono interaktywną mapę z wyrysowanym przebiegiem trasy. Na trasie oznaczono punkty w kolorze zielonym, a punkt startowy wyróżniono kolorem czerwonym. Po odwiedzeniu ostatniego punktu trasa prowadzi z powrotem do miejsca startu.

Obok mapy znajduje się tabela identyczna jak ta dostępna w trybie edycji trasy. Pod mapą użytkownik ma możliwość wyboru jednej z trzech opcji. **Delete Route**, **Update Route** oraz **Optimize Route**. Pierwsza opcja powoduje całkowite usunięcie trasy wraz ze wszystkimi punktami z oknem z prośbą o potwierdzenie operacji. Druga opcja przenosi użytkownika do wcześniej opisanego widoku strony edycji. Trzecia natomiast odpowiada za prawidłowe uporządkowanie punktów zgodnie z ich kolejnością.

Interfejs administratora jest bardzo podobny do interfejsu użytkownika, jednak zawiera kilka istotnych różnic. Po zalogowaniu na konto administratora w widoku ścieżek pojawia się dodatkowy przycisk, który prowadzi do strony ze wszystkimi ścieżkami użytkowników. Administrator może je tam swobodnie przeglądać oraz usuwać.

Ponadto kliknięcie pola *User* w nagłówku przenosi do panelu administracyjnego Django. W tym miejscu administrator ma możliwość nadawania innym użytkownikom uprawnień administracyjnych, blokowania kont (co skutkuje ich czasową dezaktywacją), a także całkowitego usuwania użytkowników.

6 Skan aplikacji

Dodano skany kodu statyczne – SAST oraz dynamiczne, czyli DAST. Oba funkcjonują jako workflow w Github, ich kod znajduje się w plikach *.yml*. SAST uruchamia się automatycznie przy każdym Pull Request do main i służy jako jeden z warunków merge'a do main. Branch musi speł-

nić wymagania 3 skanerów kodu: Semgrep, Safety oraz Bandit – nie spełnienie wymogów skutkuje uniemożliwieniem scalenia. DAST uruchamiany jest ręcznie na działającej aplikacji pod wpisywanym adresem, a skanery dynamiczne to ZAP, SQLMap i Wapiti. Wszystkie raporty zapisywane są jako pliki do artefaktów dostępne do pobrania (jako bardziej przejrzysta opcja obejrzenia raportu) i wypisywane w konsoli w celu szybszego dostępu.

7 Testy bezpieczeństwa

W celu zweryfikowania poprawności działania zastosowanych reguł bezpieczeństwa, zarówno w Cloudflare, jak i w AWS WAF, przeprowadzono serię testów funkcjonalnych. Poniżej przedstawiono opis zainstalowanych reguł, sposób ich przetestowania oraz uzyskane rezultaty.

Reguły zainstalowane w Cloudflare:

Ograniczenie dostępu do użytkowników spoza sieci Politechniki Wrocławskiej

```
1 (not ip.src in {156.17.0.0/16})
```

Reguła ta wymusza wyświetlenie challenge'a dla wszystkich użytkowników łączących się spoza podsieci 156.17.0.0/16, czyli poza siecią Politechniki Wrocławskiej.

Weryfikacja: Test wykonano, nawiązując połączenie z sieci zewnętrznej, niepowiązanej z PWr. Użytkownik został prawidłowo przekierowany do strony weryfikacyjnej (challenge), co potwierdziło skuteczność działania reguły.

Ograniczenie dostępu do panelu administratora z krajów innych niż Polska

```
1 (http.request.uri.path contains "/admin" and ip.src.country ne "PL")
```

Reguła blokuje dostęp do zasobów, w których ścieżka URI zawiera fragment „/admin", jeżeli połączenie pochodzi spoza terytorium Polski.

Weryfikacja: Próba wejścia na adres zawierający „/admin" została przeprowadzona przy użyciu zagranicznego adresu IP. Dostęp został zablokowany zgodnie z założeniami.

Blokowanie ruchu z wybranych krajów: Białoruś, Chiny, Indie, Iran, Rosja

```
1 (ip.src.country in {"BY" "CN" "IN" "IR" "RU"})
```

Reguła ta odrzuca cały ruch przychodzący z wymienionych państw.

Weryfikacja: Test polegał na próbie wejścia na stronę przy wykorzystaniu połączenia VPN zlokalizowanego w Chinach. Połączenie zostało całkowicie zablokowane, co potwierdziło poprawne działanie mechanizmu filtrującego.

Reguły zainstalowane na AWS WAF:

1. **Reguła limitująca liczbę zapytań z pojedynczego adresu IP**

Reguła monitoruje liczbę zapytań pochodzących z pojedynczego adresu IP w oknie czasowym wynoszącym 300 sekund i blokuje źródła generujące ponad 20 żądań (jest to wartość testowa)
Weryfikacja: Test wykonano poprzez wysłanie serii automatycznych zapytań (ponad skonfigurowany limit) z jednego adresu IP. Po przekroczeniu progu 20 żądań w ciągu 5 minut WAF prawidłowo zablokował dalszy ruch, co zostało odnotowane zarówno w logach CloudWatch, jak i w podglądzie zapytań próbkujących.
Powyższe rozwiązanie jest implementacją monitorowania sesji w opisywanej aplikacji. Reguła monitoruje sesję określonego adresu IP oraz blokuje go po przekroczeniu określonej liczby zapytań.
2. **Reguła blokująca SQL injection** W celu sprawdzenia podatności aplikacji na ataki typu SQL Injection wykonano zapytanie HTTP POST przy użyciu narzędzia curl:

```
curl -v -X POST "https://trash-tracker.pl/accounts/login/" -H "Content-Type:
    ↪ application/x-www-form-urlencoded" --data "username=dawid' OR '1' = '1 &
    ↪ password=dawid123&csrfmiddlewaretoken=3
    ↪ H9ccqBd9vpJEsFM0hEJMN1G8blkOJgcW6secQTR3SCcAlWYkOOUz7Ik4oKeuH9U"
```

Cel testu: Sprawdzenie, czy WAF wykryje klasyczne próby wstrzyknięcia kodu SQL, które mogą prowadzić do nieautoryzowanego dostępu.
Rezultat: WAF wykrył SQL injection
3. **Reguła blokująca XSS**
W ramach testów podatności na ataki XSS wprowadzono następujący fragment kodu:

```
<script> test XSS; <\script>
```

Rezultat: Wprowadzone znaczniki <script> powinny zostać wykryte przez WAF a ruch nie dopuszczony w głąb infrastruktury.

8 Podsumowanie

Projekt TrashTracker stanowi kompleksową aplikację webową wspomagającą optymalizację tras pojazdów zajmujących się odbiorem odpadów komunalnych na terenie Wrocławia. Celem systemu było zredukowanie kosztów eksploatacyjnych, czasu przejazdu oraz wpływu środowiskowego śmieciarek, poprzez inteligentne wyznaczanie tras pomiędzy punktami odbioru odpadów. Podsumowując, TrashTracker to nowoczesne, zintegrowane rozwiązanie dla miast chcących zarządzać odpadami w sposób inteligentny, ekonomiczny i ekologiczny.