

**Nazwa Wydziału**

* + - 1. **Nazwa Katedry**
      2. **Katedra Inżynierii Oprogramowania**
      3. Zarządzanie projektami (IM)
      4. **Piotr Wrona**
      5. Nr albumu s25773
      6. **Tytuł pracy dyplomowej**
      7. “Procedura bezpieczeństwa dla ciągłej integracji oprogramowania oparta na DevSecOps.”
      8. Rodzaj pracy
      9. magisterska
      10. Imię i nazwisko promotora
      11. Dr. Hab. Inż. Piotr Habela

miejsce, miesiąc, rok obrony

**Spis treści:**

[1. Wstęp 3](#__RefHeading___Toc425_451261914)

[1.1. Cel pracy 3](#__RefHeading___Toc427_451261914)

[1.2. Założenia 3](#__RefHeading___Toc427_451261914_Copy_1)

[1.3. Praktyki DevSecOps 4](#__RefHeading___Toc437_451261914)

[1.3.1.1. Geneza 4](#__RefHeading___Toc138_3912164325)

[1.3.1.2. Cele DevSecOps 4](#__RefHeading___Toc140_3912164325)

[1.3.1.3. Manifest DevSecOps 5](#__RefHeading___Toc142_3912164325)

[1.3.1.4. Podejścia do DevSecOps 7](#__RefHeading___Toc144_3912164325)

[2. Rozwiązanie projektowe 8](#__RefHeading___Toc429_451261914)

[2.1. Rozwiązanie wejściowe 8](#__RefHeading___Toc149_3912164325)

[2.1.1. Wybór struktury aplikacji 8](#__RefHeading___Toc733_3420670368)

[2.1.2. Wybór środowiska aplikacji 9](#__RefHeading___Toc735_3420670368)

[2.1.3. Wybór bazy danych 9](#__RefHeading___Toc737_3420670368)

[2.1.4. Wybór modelu i architektury dla projektu 10](#__RefHeading___Toc739_3420670368)

[2.1.4.1. Modele infrastrukturalne 10](#__RefHeading___Toc431_451261914)

[2.1.4.2. Propozycja infrastruktury 10](#__RefHeading___Toc741_3420670368)

[2.1.5. Proces ciągłego dostarczania infrastruktury 13](#__RefHeading___Toc8667_1024472429)

[2.1.6. Wejściowy proces CI/CD dla projektu 15](#__RefHeading___Toc8669_1024472429)

[2.1.6.1. Cel projektowania i stosowania CI/CD 15](#__RefHeading___Toc57_1453835382)

[2.1.6.2. Propozycja procesu CI/CD dla projektu 16](#__RefHeading___Toc743_3420670368)

[2.2. Rozwiązanie wyjściowe 19](#__RefHeading___Toc151_3912164325)

[2.2.1. Planowanie bezpieczeństwa 19](#__RefHeading___Toc154_3912164325_Copy_1)

[2.2.1.1. Miary bezpieczeństwa procesu CI/CD 19](#__RefHeading___Toc2677_3420670368)

[2.2.1.2. Planowanie implementacji do procesu CI/CD 20](#__RefHeading___Toc2679_3420670368)

[2.2.1.3. Zarządzanie sekretami 21](#__RefHeading___Toc2681_3420670368)

[2.2.2. Implementacja DevSecOps w procesie CI/CD 22](#__RefHeading___Toc4350_3420670368)

[2.2.3. Zastosowanie narzędzi wspomagających DevSecOps 24](#__RefHeading___Toc160_3912164325)

[2.2.3.1. Zastosowanie narzędzia Sonarqube 24](#__RefHeading___Toc162_3912164325)

[2.2.3.2. Kontrola jakości kodu źródłowego 27](#__RefHeading___Toc3610_3420670368)

[2.2.3.3. Zastosowanie narzędzia Trivy 28](#__RefHeading___Toc8671_1024472429)

[2.2.4. Implementacja DevSecOps w infrastrukturze 35](#__RefHeading___Toc3612_3420670368)

[2.2.4.1. Implementacja DevSecOps w procesie dostarczania infrastruktury 35](#__RefHeading___Toc8673_1024472429)

[2.2.4.2. Zastosowanie narzędzia Checkov 36](#__RefHeading___Toc8675_1024472429)

[3. Wnioski z przeprowadzonego projektu 39](#__RefHeading___Toc425_451261914_Copy_1)

[3.1. Stopień spełnienia założeń projektowych 39](#__RefHeading___Toc169_3912164325)

[4. Spis rysunków 40](#__RefHeading___Toc1258_1495590498)

[5. Bibliography 41](#__RefHeading___Toc1677_3923969623)

# Wstęp

## Cel pracy

* + - 1. Celem pracy jest analiza procesu ciągłego dostarczania i eksploatacji oprogramowania, a następnie wyszczególnienie miejsc podatnych z punktu widzenia jego bezpieczeństwa. Zaproponowane zostaną rozwiązania na redukcję wrażliwości i docelowo zabezpieczenie kodu źródłowego. W niektórych sytuacjach przedstawione zostanie porównanie narzędzi, z wyszczególnieniem wad i zalet.
      2. Opracowanie ma pozwolić na jak najgłębsze przedstawienie zagadnienia oraz porządkuję zagadnienie DevSecOps. Dzięki temu, praca będzie przydatna dla osób zajmujących się procesami ciągłego dostarczania i eksploatacji oprogramowania jako uzupełnienie wiedzy, ale również będzie niezwykle przydatna dla osób poznających tematykę i chcących zaczerpnąć bazową wiedzę z zakresu bezpieczeństwa przy procesach CI/CD.

## Założenia

* + - 1. Spośród założeń projektowych pracy można wyróżnić:
* Zastosowanie aplikacji JavaScript, wraz z bazą danych – MongoDB. Stworzona platforma ma funkcjonalność wyświetlania video, indeksowanych w bazie danych.
* Zastosowanie technologii konteneryzacji do budowy aplikacji.
* Zastosowanie środowiska GitHub do tworzenia automatyzacji oraz jako repozytorium kodu źródłowego.
* Proces przewiduje manualne kroki, związane z pewną weryfikacją inżynierską – proces nie przewiduję pełnej automatyzacji.
* Proces nie jest kompletnie odzwierciedlony w postaci kodu źródłowego. Najistoniejsze z perspektywy projektu jest stworzenie procedury.
* Repozytorium przytoczone w źródle tworzy integralną część projektu.
* Opisy poddatności w przypadku zastosowania skanów bezpieczeństwa są pewną próbą. Kompleksowe raporty można znaleźć w źródłach.

## Praktyki DevSecOps

#### Geneza

* + - 1. DevSecOps to koncepcja, która łączy w sobie praktyki DevOps z bezpieczeństwem. Pomysł powstania DevSecOps pojawił się w odpowiedzi na zmieniające się wymagania rynku, które nakładały na organizacje IT konieczność szybszego wytwarzania oprogramowania, a jednocześnie zachowania wysokiego poziomu bezpieczeństwa.
      2. Tradycyjnie, bezpieczeństwo informatyczne traktowane było jako oddzielny proces, który przeprowadzany był dopiero po zakończeniu fazy rozwoju oprogramowania. Taki podejście jednak prowadziło do opóźnień i kosztów związanych z naprawą luk w zabezpieczeniach.
      3. W odpowiedzi na te wyzwania powstało podejście DevSecOps, które zakłada włączenie zabezpieczeń informatycznych już na początku cyklu życia aplikacji. Dzięki temu możliwe jest szybsze wykrywanie i naprawianie ewentualnych błędów związanych z bezpieczeństwem.
      4. Pojęcie DevSecOps zostało zapoczątkowane w 2012 roku przez Garta Allena, który na łamach swojego bloga opisał potrzebę zintegrowania bezpieczeństwa z praktykami DevOps. Od tamtej pory pojęcie to zyskało na popularności i stało się standardem w wielu organizacjach IT.
      5. Obecnie DevSecOps to nie tylko podejście do zarządzania bezpieczeństwem w procesie deweloperskim, ale także zbiór najlepszych praktyk i narzędzi, które pomagają organizacjom wdrażać te praktyki. Dzięki temu możliwe jest szybsze wytwarzanie oprogramowania przy jednoczesnym zachowaniu wysokiego poziomu bezpieczeństwa.

#### Cele DevSecOps

* + - 1. Głównym celem DevSecOps jest łączenie praktyk DevOps z bezpieczeństwem informatycznym w celu uzyskania szybszego i bardziej efektywnego wytwarzania oprogramowania o wysokim poziomie bezpieczeństwa. W ramach podejścia DevSecOps wdrażane są metody i narzędzia, które pozwalają na:

1. Wczesne wykrywanie błędów związanych z bezpieczeństwem - poprzez włączenie zabezpieczeń już na etapie tworzenia kodu, możliwe jest szybsze wykrywanie i usuwanie błędów związanych z bezpieczeństwem. Dzięki temu organizacje są w stanie zapobiec sytuacjom, w których luk w zabezpieczeniach zostają wykryte już na etapie produkcji, co znacznie zwiększa koszty naprawy.
2. Automatyzację procesów - DevSecOps zakłada automatyzację procesów związanych z bezpieczeństwem, takich jak testy bezpieczeństwa czy weryfikacja podatności. Dzięki temu możliwe jest szybsze wdrażanie zmian oraz uniknięcie błędów wynikających z ludzkiego czynnika.
3. Wdrażanie zabezpieczeń jako kod - DevSecOps zachęca do traktowania zabezpieczeń informatycznych jako kodu, który może być przechowywany w systemie kontroli wersji, a następnie wdrażany w sposób automatyczny. Dzięki temu możliwe jest szybsze i bardziej skuteczne wprowadzanie zmian związanych z bezpieczeństwem.
4. Kultura bezpieczeństwa - DevSecOps zakłada tworzenie kultury bezpieczeństwa w organizacji, w której każdy pracownik jest świadomy zagrożeń i zna podstawowe zasady bezpieczeństwa informatycznego. Dzięki temu możliwe jest zapobieganie incydentom związanym z cyberbezpieczeństwem oraz szybsze reagowanie w przypadku wystąpienia zagrożenia.
5. Integrację różnych zespołów - DevSecOps zachęca do integracji różnych zespołów, takich jak zespół programistów, zespół ds. bezpieczeństwa informatycznego czy zespół ds. operacji. Dzięki temu możliwe jest wspólne definiowanie wymagań oraz szybsze reagowanie na zmiany w otoczeniu.

#### Manifest DevSecOps

* + - 1. W celu poprawnego zrozumienia badanego zagadnienia konieczne jest przybliżenie koncepcji początkowej terminu DevSecOps. Został on podobnie jak koncept podejścia Agile zebrany w zbiór reguł, które tworzą manifest[[1]](#endnote-2). Jego twórcami są:
* Ian Allison
* Justin Tiplitsky
* Scott Kennedy
* Nigel Kersten
* Shannon Lietz
* Fabian Lim
* Michelle Nikulshin
* Christian Price
* Ravi Dhungel
* Kyle Rose
* Brandon Sherman
  + - 1. Ropatrzmy poszczególne tezy stawiane przez manifest:
      2. **„Leaning in** over Always Saying “No””
      3. Przez to hasło należy rozumieć pewną transformację podejściową członków zespołu z częstego mówienia „nie” do świadomego wkładu w rozwijanie kwestii bezpieczeństwa. Temat ten nie może być odkładany jako nieistotny, a kwestie związane z inicjatywami w tej materii muszą być wspierane.
      4. **„Data & Security Science** over Fear, Uncertainty and Doubt”
      5. DevSecOps musi być oparty na profesjonalnym podejściu, wspomaganym przez rzetelne raporty wykorzystywanych narzędzi niż podejściu chaotycznym i kierowanym przez strach.
      6. „**Open Contribution & Collaboration** over Security-Only Requirements”
      7. Tworzenie oprogramowania opiera się na integracji wielu produkty zewnętrznych oraz wewnętrznych. Dlatego, konieczna jest świadomość wszystkich elementów oraz jak je zintegrować aby stworzyć bezpieczny system.
      8. „**Consumable Security Services with APIs** over Mandated Security Controls & Paperwork”
      9. Priorytem powinno być stworzenie procesu który możemy wyrażać jako pewnego rodzaju usługę zapewniania bezpieczeństwa zamiast skupiać się na narzędziach służących wykonywaniu danej funkcjonalności.
      10. „**Business Driven Security Scores** over Rubber Stamp Security”
      11. Konieczne jest dostosowanie naszych wymogów dotyczących bezpieczeństwa do wymagań biznesu. Każda implementacja powinna być oceniana pod względem jej użyteczności, zamiast zwykłego ohczenia działania jako zrobione.
      12. **Red & Blue Team Exploit Testing** over Relying on Scans & Theoretical Vulnerabilities
      13. Tworzenie zespółów czerwony/niebiescy do nauki obrony naszego systemu. W zalożeniu podejście to odwzorowuję bardziej realistyczne przebieg potencjalnego ataku przy użyciu najbardziej aktualnych narzędzi penetracyjnych.
      14. **24x7 Proactive Security Monitoring** over Reacting after being Informed of an Incident
      15. Ciągłe skanowanie naszego systemu pozwala nam wykrywać zdarzenia potencjalnie niebezpieczne i przeciwdziałać im przed szkodą.
      16. **Shared Threat Intelligence** over Keeping Info to Ourselves
      17. Ciągła wymiana informacji jest kluczem do rozwijania świadomości i poszerzania wiedzy wszystkich członków zespołu. Takie działania pozwalają na szybszą reakcję i lepszą odpowiedź w przypadku sytuacji zagrożenia.
      18. **Compliance Operations** over Clipboards & Checklists
      19. Ciągła kontrola ponad ufność w checklisty.

#### Podejścia do DevSecOps

1. Integracja bezpieczeństwa z procesami DevOps - w tym podejściu DevSecOps jest traktowane jako naturalne rozszerzenie praktyk DevOps. Oznacza to, że bezpieczeństwo jest włączane w każdą fazę cyklu życia aplikacji, od planowania i projektowania po wdrożenie i utrzymanie. W ramach tego podejścia, zespoły DevOps i zespoły ds. bezpieczeństwa muszą ściśle ze sobą współpracować, aby zapewnić wytwarzanie oprogramowania o wysokim poziomie bezpieczeństwa.
2. Wdrażanie bezpieczeństwa jako kodu - w tym podejściu DevSecOps zachęca się do traktowania zabezpieczeń informatycznych jako kodu, który jest przechowywany w systemie kontroli wersji. Dzięki temu można stosować do zabezpieczeń te same praktyki, jakie stosuje się w procesie wytwarzania oprogramowania, takie jak testowanie automatyczne czy wdrażanie ciągłe. W ramach tego podejścia, zespoły ds. bezpieczeństwa mogą korzystać z narzędzi programistycznych i automatyzacji, co pozwala na szybsze wdrażanie zmian związanych z bezpieczeństwem.
3. Bezpieczeństwo jako kultura organizacyjna - w tym podejściu DevSecOps skupia się na kulturze organizacyjnej, w której każdy pracownik jest świadomy zagrożeń i zna podstawowe zasady bezpieczeństwa informatycznego. W ramach tego podejścia, zespoły ds. bezpieczeństwa skupiają się na edukacji i szkoleniach, aby zapewnić, że cała organizacja działa zgodnie z najlepszymi praktykami bezpieczeństwa. Dzięki temu możliwe jest zapobieganie incydentom związanym z cyberbezpieczeństwem oraz szybsze reagowanie w przypadku wystąpienia zagrożenia.

# Rozwiązanie projektowe

## Rozwiązanie wejściowe

### Wybór struktury aplikacji

* + - 1. W ostatnich latach, trend rozwijania aplikacji sukcesywnie przesuwa się z dużych monolitycznych aplikacji na aplikacje mikro serwisowe. Tabela poniżej opisuję główne różnice dla obu podejść.

|  |  |
| --- | --- |
| Podejście monoliticzne | Podejście mikroserwisowe |
| * Potencjalna awaria powoduję duże problemy funkcjonalnościowe i może sparaliżować cały system * Zmiany w aplikacji są trudne, ze względu na złożoność kodu źródłowego * Środowiska uruchumieniowe często wymagają dużych zasobów i są uruchamiane na dużych maszynach serwerowych | * Potencjalna awaria powoduję utratę pojedynczej funkcjonalności ale nie powoduję awarii całego systemu * Zmiany w aplikacji są proste, ze względu na rozdzielenie kodu źródłowego na mniejsze części * Środowisko nie wymaga dużo zasobów i jest to zazwyczaj kontener lub pod |

* + - 1. Rozpatrując przedstawione porównanie, łatwo można dostrzec przeważające zalety podejścia mikro serwisowego. Z tego powodu, projekt przewiduję zastosowanie konteneryzacji zarówno dla serwera www oraz serwera baz danych.
      2. W uproszczeniu konteneryzacja jest wydzieleniem małego środowiska uruchomieniowego, z izolacją zasobów, konfiguracji czy udostępnieniem osobnego interfejsu sieciowego. Kontener jest budowany na podstawie pliku „Dockerfile”, który zawiera predefiniowany obraz docker’a i uruchamia kod źródłowy na tak przygotowanym środowisku.

### Wybór środowiska aplikacji

* + - 1. Projekt przewiduje użycie aplikacji node.js jako środowiska uruchomieniowego. Wyróżnia się ono następującymi cechami:
* wieloplatformowa
* bardzo duża szybkość obsługi zapytań
* łatwość tworzenia zaawansowanych programów
* duża liczba framework’ów przyspieszających pisanie aplikację
  + - 2. Node.js wykorzystuję składnie języka JavaScript, który cechuję się elastycznością (możliwość zastosowania programowania funkcyjnego czy obiektowego) oraz dużą czytelnością (jest to język wysokopoziomowy).

### Wybór bazy danych

* + - 1. Aplikacja wykorzystuję bazę danych do przechowywania informacji o video, udostępnianym przez serwer www. Dla przytoczonego zastosowania, nie jest wymaga złożona relacyjność, a optymalne jest zestawienie klucz wartość. Wybór bazy padł na nierelacyjną bazę mongodb.
      2. Baza cechuję się:
* dużą liczbą obsługiwanych typów danych
* obsługą kursorów
* zapytaniami ad-hoc
* zapytaniami do zagnieżdżonych pól dokumentów
* indeksowaniem
* wsparciem dla agregacji danych
* możliwością składowania plików w bazie
* architekturą zaprojektowana z myślą o łatwej replikacji

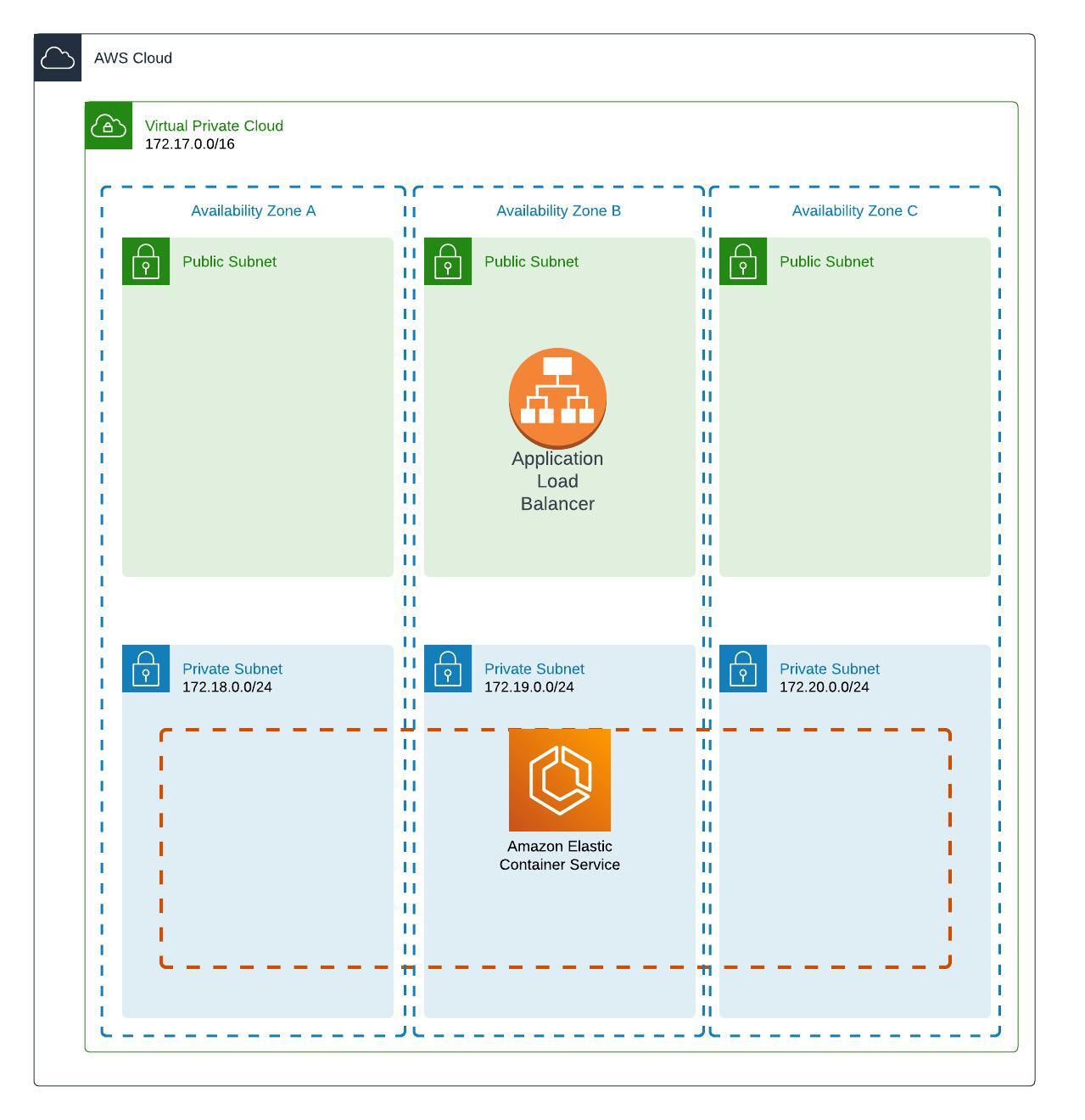
### Wybór modelu i architektury dla projektu

#### Modele infrastrukturalne

* + - 1. Główny podział infrastruktury współczesnych aplikacji opiera się na lokalizacji serwerów i dysponowaniem zakresem obowiązków dotyczących obsługi. Wyróżniamy:
* On-premises – infrastruktura zlokalizowana w przedsiębiorstwie. Pełna obsługa hardware’u przypada na zatrudnionych administratorów sieci.
* Cloud – w tym modelu przesuwamy odpowiedzialność za obsługę sprzętu na zewnętrznych dostawców. Ze względu na zakres obowiązków, w rozwiązaniach chmurowych możemy wymienić:
  + SaaS (Software as a Service) – model zakłada kompletne przeniesienie odpowiedzialności za sprzęt, infrastrukturę oraz aplikację na rzecz dostawcy.
  + PaaS (Platform as a Service) – polega na udostępnieniu przez dostawce gotowej infrastruktury oraz sprzętu do budowy aplikacji.
  + IaaS (Infrastructure as a Service) – w tym modelu zarówno budowa infrastruktury oraz aplikacji leży po stronie użytkownika końcowego. Dostawca odpowiada tylko za sprzęt.
    - 1. W praktyce zdarza się, że rolę się przeplatają a obok modelu IaaS stosujemy rozwiązania SaaS. Jednym z częstych przypadków jest stosowanie usług Elasticsearch do monitorowania naszej infrastruktury IaaS.

#### Propozycja infrastruktury

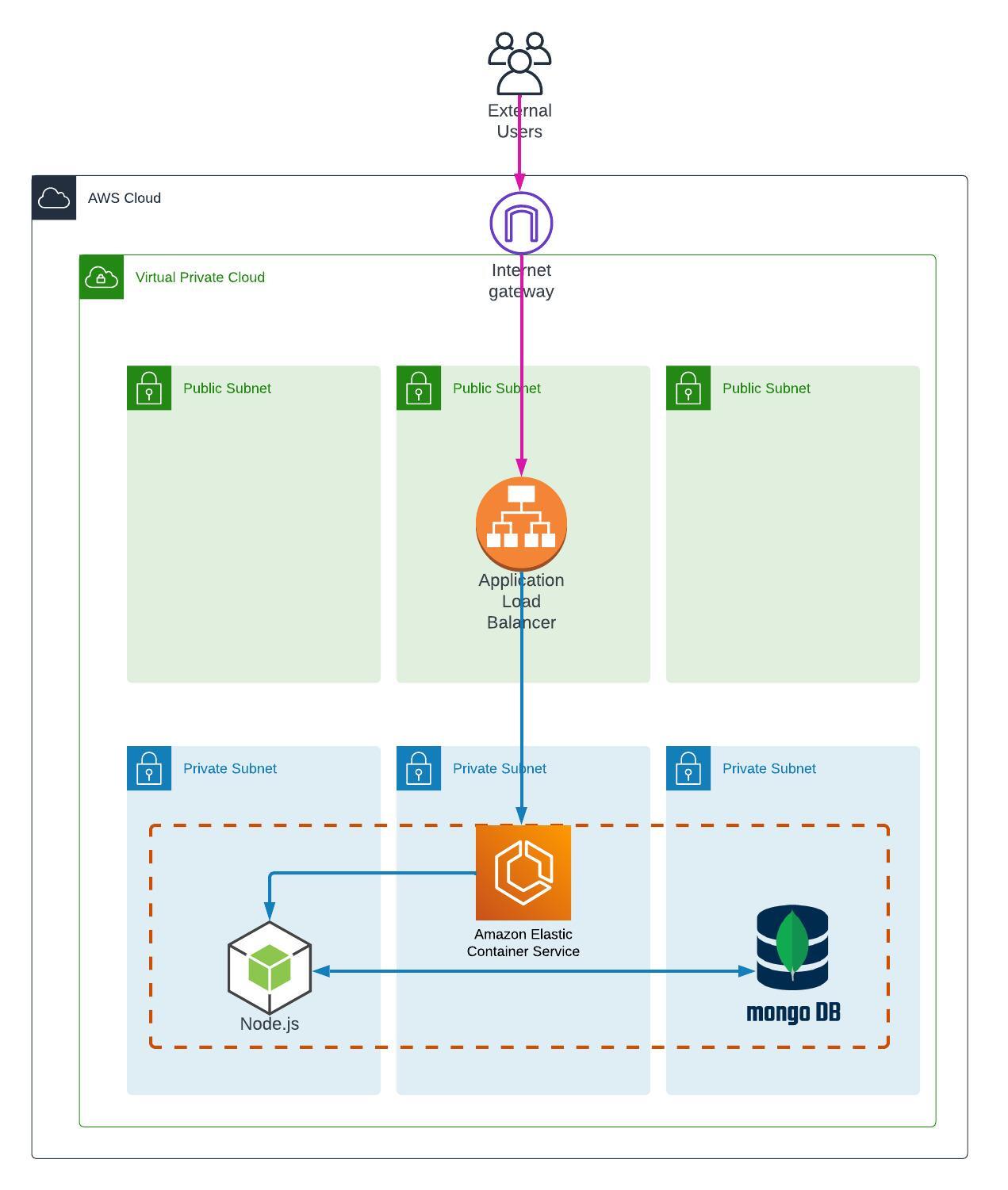
* + - 1. Na potrzeby rozwiązania projektowego, użyta została chmura AWS w modelu IaaS. Występują też pewne elementy zbliżvone do modelu PaaS, takie jak zastowanie AWS ECS, w podejściu fargate.
      2. Głównym logicznym podziałem naszej infrastruktury jest VPC (Virtual Private Cloud), który definiuję naszą sieć i zakres adresów. Architektura jest 2-tierowa i wyróżnia 3 podsieci publiczne oraz 3 podsieci prywatne. Cała sieć znajduję się w jednym regionie eu-central-1 (Frankfurt), co w pewien sposób powoduję podatność na awarię w obrębie regionu, natomiast pewną redundantność uzyskujemy poprzez wydzielenie 3 AZ (Avaliability zone): 1A, 1B, 1C.

Rysunek 1: Schemat infrastruktury projektu. Podział sieci. Źródło: zbiór autora

* + - 1. Każda podsieć publiczna ma maskę 255.255.255.0 która obejmuję 254 adresy. Dla naszego rozwiązania projektowe zakres możliwyhch destynacji jest nieproporocjonalnie duży, natomiast umożliwa to ewentualne dalsze rozwijanie infrastruktury czy mozliwości skalowania.
      2. Wejściem do naszego systemu jest IG (Internet Gateway) który przyjmuję cały ruch publiczny i kieruje go na ALB (Application Load Balancer). Odpowiada on za dystrubucję ruchu, terminację certyfikatu SSL oraz częściową obsługę błędów (np.: poprzez odpowiedź na błąd 404).

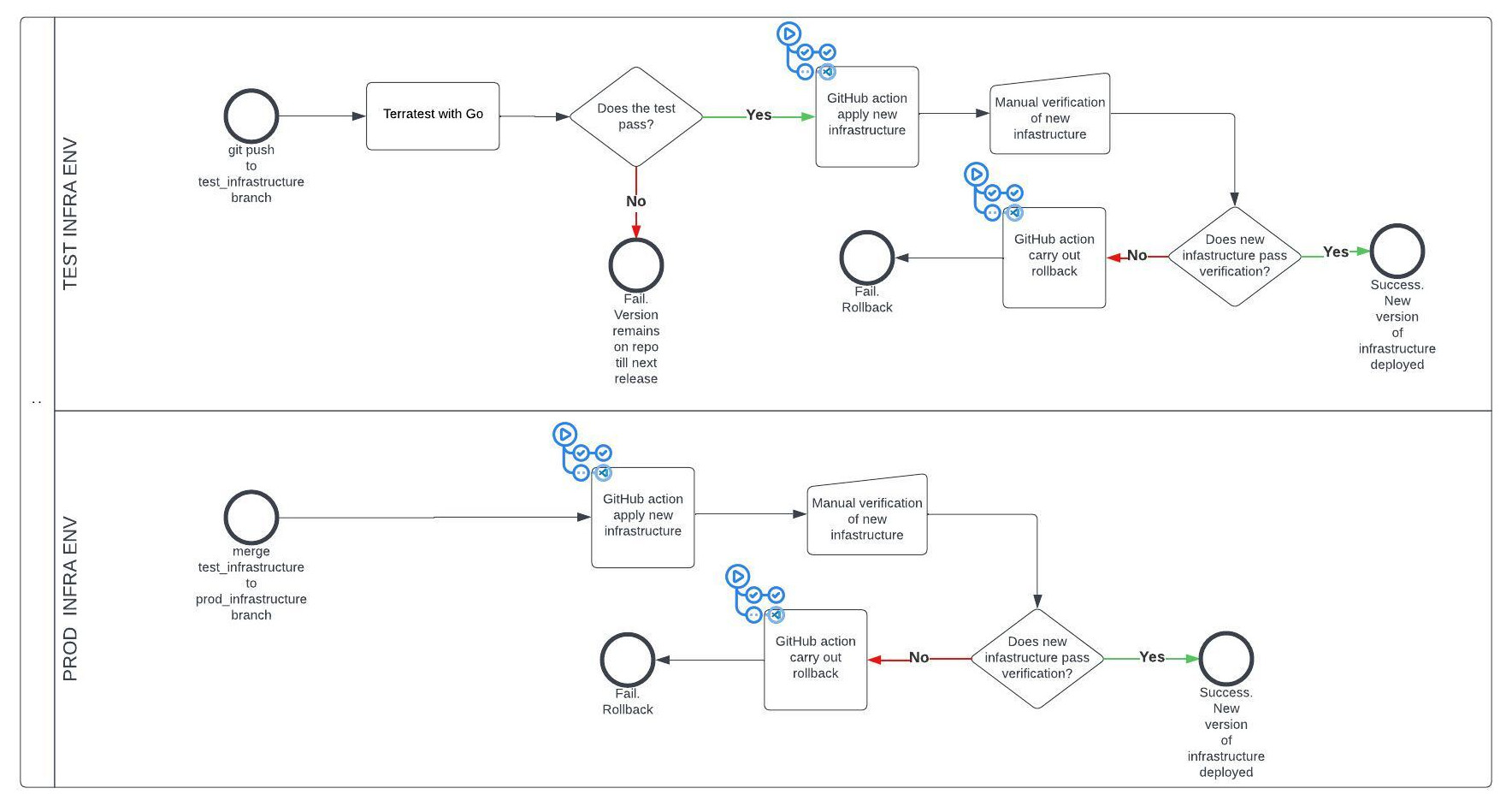
wysokie zapewnienie dostępności - 3 dostępne, redundatne strefy dostępu

* + - 1. Kontenery aplikacji znajdują się w podsieciach prywatnych i są zarządzane przez serwis ECS (Elastic Container Service) w modelu Fargate. Podejście to cechuje:
* brak konieczności administracji serwerami
* mikroserwisowość
* zastępowalność
* łatwość skalowania
* skuteczniejsze wykorzystanie zasobów
* skalowalność horyzontalna

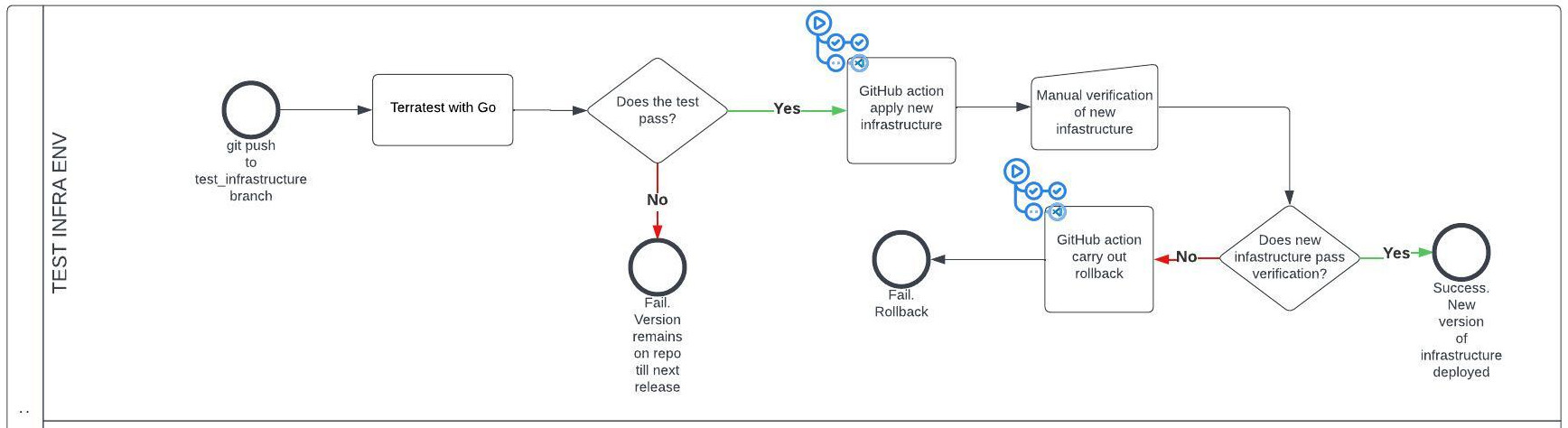
Rysunek 2: Schemat infrastruktury projektu. Wejście do aplikacji z perspektywy użytkownika zewnętrzenego. Źródło: zbiór autora

### Proces ciągłego dostarczania infrastruktury

* + - 1. W procesie dostarczania infrastruktury wyróżniamy środowiska: testowe oraz
      2. produkcyjne. Dla każdego zostały przydzielone branch’e: „test\_infrastructure” oraz „prod\_infrastructure”, których zawartość reflektuję w faktycznej infrastrukturze.

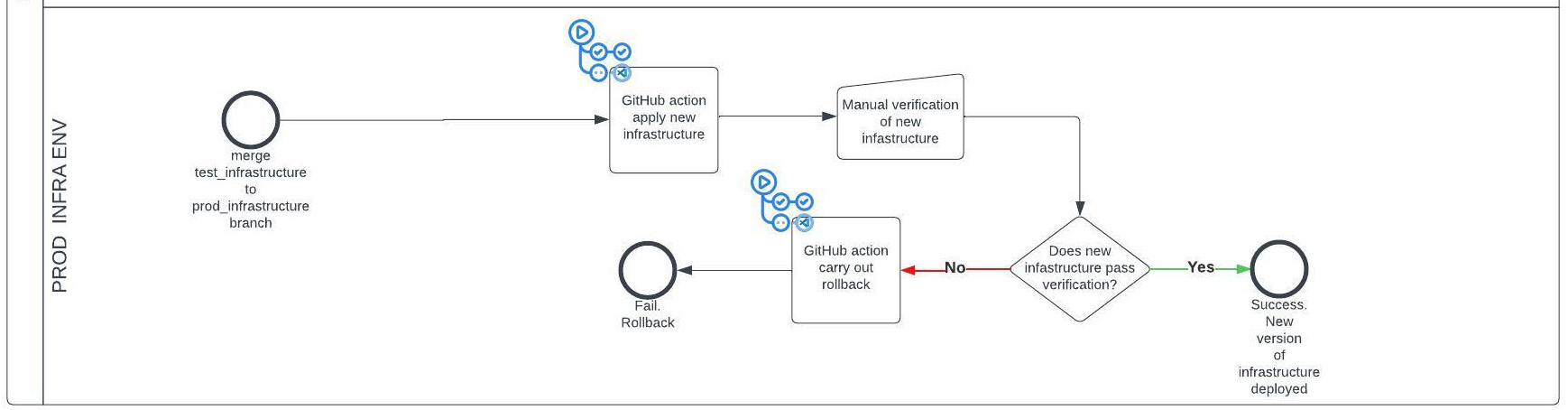
Rysunek 3: Schemat procesu dostarczania infrastruktury. Źródło: zbiór autora

* + - 1. Aby zainicjować cykl testowy, konieczne jest wypuszczenie kodu do odpowiedniej gałęzi. Rozpoczyna to tzw.: „Terratest”, który jest framework’em open-source napisany w języku Go. Umożliwia on automatyzację testowania infrastruktury w chmurze.
      2. Terratest umożliwia testowanie różnych aspektów infrastruktury w chmurze, w tym konfiguracji serwerów, dostępności usług, skalowalności i bezpieczeństwa. Dzięki temu można zwiększyć pewność, że wdrożenie działa zgodnie z oczekiwaniami.

Rysunek 4: Schemat procesu dostarczania infrasktruktury dla środowiska testowego.

Źródło: zbiór autora

* + - 1. W przypadku niepowodzenia, kod zostaje odrzucony i dalsze akcje nie są wyzwalane. W sytuacji odwrotnej, rozpoczyna się wykonanie akcji GitHub która aplikuję zmiany na środowisko testowe. Kolejnym krokiem jest manualna weryfikacja środowiska i podjęcie decyzji co do stabilności środowiska. W przypadku akceptacji – transkacja jest zakończona sukcesem, w przypadku jej braku- następuję rollback do poprzedniej wersji infrastruktury.
      2. Proces dostarczania aplikacji na środowisko produkcyjne jest poprzedzone pewnym ustalonym czasem testowania na niższym środowisku. Inicjacja procesu następuję przez połączenie zmian ze środowiska testowego z środowiskiem produkcyjnym.

Rysunek 5: Schemat proces dotarczania infrasktruktury dla środowiska produkcyjnego.   
Źródło: zbiór autora

* + - 1. Połączenie gałęzi rozpoczyna akcję GitHub wprowadzającą zmiany na środowisko produkcyjne. Cykl w tym biegu, nie uzwględnia już Terratest’u – przyjęto założenie stabilności regionu testowego. Zachowana natomiast jest manualna weryfikacja inżynierska, kończona przez decyzję dotyczącą stabilności systemy. W przypadku akceptacji – transkacja jest zakończona sukcesem, w przypadku jej braku- następuję rollback do poprzedniej wersji infrastruktury.

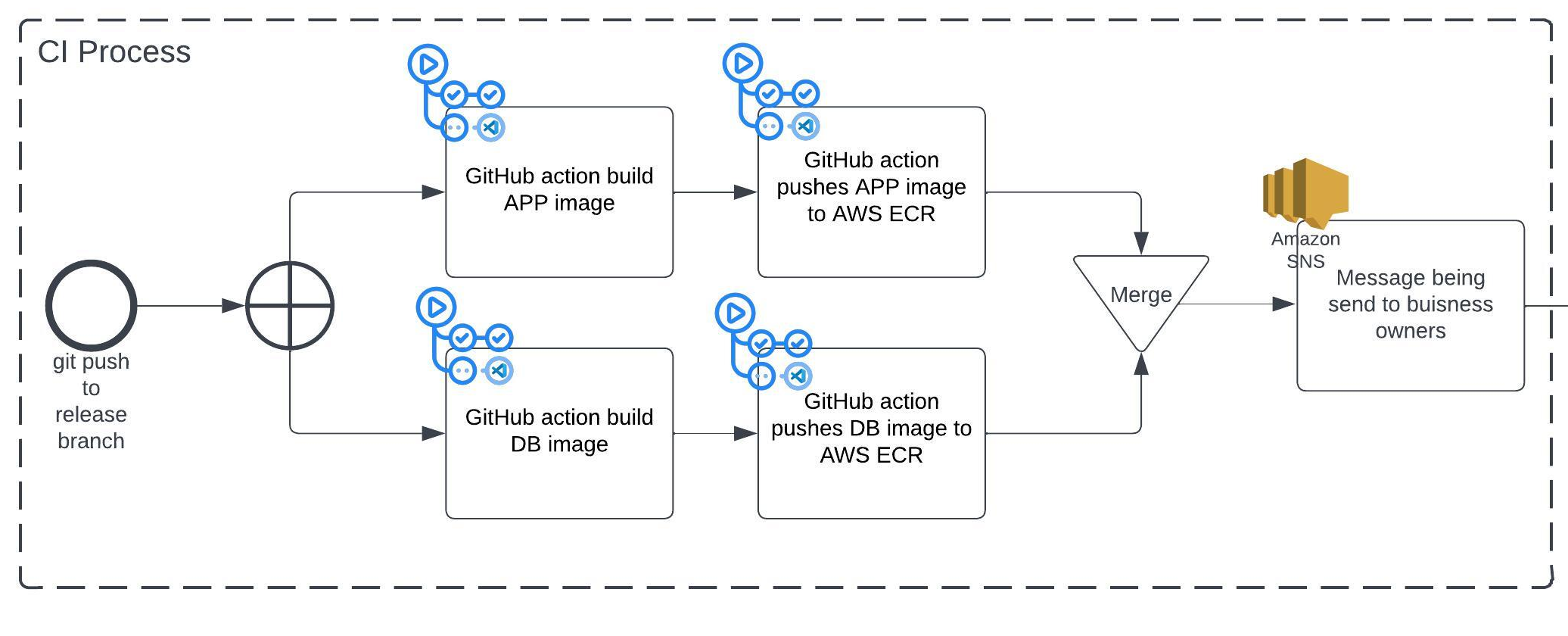
### Wejściowy proces CI/CD dla projektu

#### Cel projektowania i stosowania CI/CD

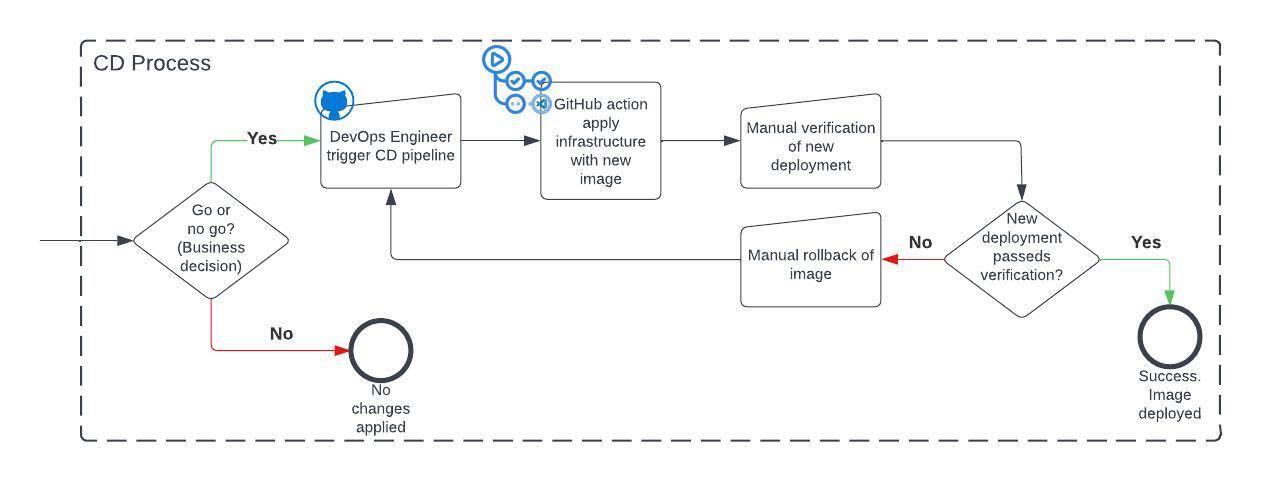
* + - 1. Trend tworzenia aplikacji na przestrzeni ostatnich lat zmienił się z typowego podejścia waterfall do podejścia agile. Podejście wywodzi się od koncepcji dostarczania MVP czyli produktu o małej wartości, w krótszych odstępach. Pozwala to na szybsze zebranie opinii klienta końcowego i zabezpiecza przedsiębiorstwo przed tworzeniem funkcjonalności niedostarczających wartości dodanej.
      2. Zmiana podejścia zrodziła również wiele wyzwań dla osób odpowiadających za cały strumień wartości i dostawę oprogramowania. Możemy wyróżnić kilka z nich:
* Integracja wielu pionów tj.: deweloperów, testerów, działu bezpieczeństwa informacji oraz biznesu
* Stworzenie oraz zarządzanie wieloma środowiskami, które muszą odpowiadać zarówno celom testerów oprogramowania czy działom operacji
* Wprowadzanie i dbanie o elementy bezpieczeństwa pomimo biznesowej presji dostarczania oprogramowania
  + - 1. Z tych wielu powodów konieczny jest prawidłowe zaplanowanie procesu CI/CD dla naszej aplikacji. CI czyli ciągła integracja to wszystkie czynności integrujące nasze repozytoria, aż do momentu decyzji biznesowej, która następnie inicjuję proces ciaŋłego wzdrażania, czyli CD.
      3. Podejście do schematu dostarczania oprogramowania jest bardzo róźne i zależy głównie od:
* Technologii wdrażanej aplikacji
* Poziomu unifikacji systemów i serwerów
* Strategii biznesowej firmy
* Ilości środowisk testowych i produkcyjnych
* Doświadczenia i testowania rozwiązań

#### Propozycja procesu CI/CD dla projektu

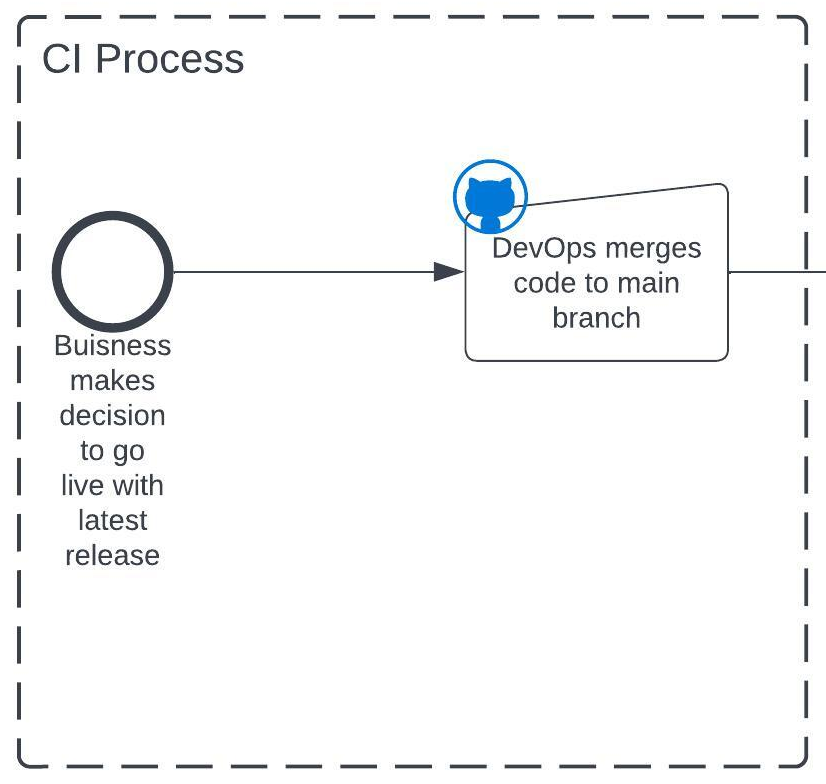
* + - 1. Projekt zakłada jedno środowisko testowe (TEST ENV) oraz jedno środowisko produkcyjne (PROD ENV). Należy nadmienić, że podejście jest uproszczone w stosunku do najczęstszych wzorów komercyjnych. Dla rozwiązań rynkowych przewiduję się zazwyczaj dwa środowiska testowe (np.: QA (Quality Assurance) oraz UAT (User Acceptance Tests)), jak również dwa środowiska produkcyjne, głównie dla celów przełączania w czasie awarii.
      2. Podejście konteneryzacyjne do aplikacji, centralizuję proces wokół repozytorium obrazów dockerowych. Wpływa to zarówno część integracyjną - obok udostępnia kodu, następuję udostępnienie obrazu, jak również część wdrożeniową – poprzez pobranie najnowszej wersji obrazu i zaaplikowanie na środowisku.

Rysunek 6: Schemat procesu CI dla środowiska testowego.   
Źródło: zbiór autora

1. Proces rozpoczyna się od wypuszczenia kodu na tzw.: „release branch”. Zaletami tworzenia dedykowanych gałęzi git dla dannych sprintów są:
   * Lepsza kontrola nad zmianami i możliwość prowadzenia dziennika zmian
   * Możliwość łatwego rollback’u do stabilnej wersji
2. Zmiana w repozytorium inicjuje równoległe przetwarzania dwóch pipelinów – jeden do tworzenia obrazu aplikacji oraz drugi do tworzenia kontenera bazy danych.
3. Każdy z nowo utworzonych obrazów zostaję otagowany mianem „latest” i przekazany do uprzednio utworzonych repozytoriów obrazów na chmurze AWS.
4. Zakończenie procesu ładowania nowych wersji aplikacji oraz bazy jest finalizowane poprzez przekazanie do kolejki Amazon SNS, notyfikacji mailowej dla biznesu.

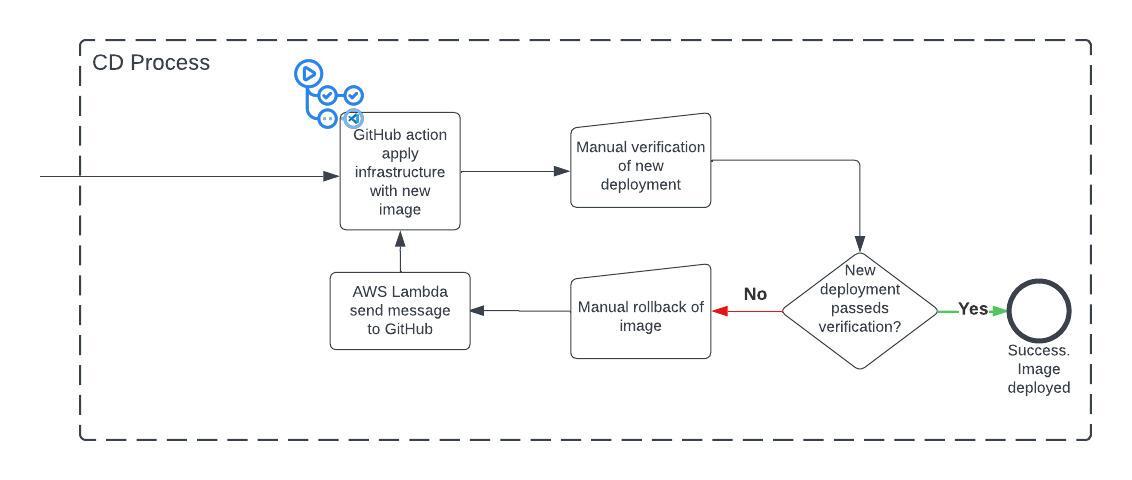
Rysunek 7: Schemat procesu CD dla środowiska testowego.   
Źródło: zbiór autora

1. Proces wdrożeniowy jest inicjowany poprzez decyzję biznesową. Na tym etapie występuję możliwość rezygnacji z nowego wydania. Taka akcja nie wymaga dodatkowych kroków, gdyż rollback następuję przy kolejnej budowie obrazów.
2. Automat aplikuję nowe rozwiązanie przy użyciu narzędzia terraform.
3. Inżynier DevOps przeprowadza weryfikację środowiska pod kątem stabilności i kompletności. Rekomendowane są w tym miejscu testy automatyczne wraz z obserwacją i finalizacja przez decyzję manualną.
4. W przypadku istotnych problemów i braku akceptacji inżyniera na emisję wydania, następuję rollback obrazu w repozytorium ECR.
5. Podobnie do poprzednich kroków z sekwencji, to inżynier DevOps inicucję aplikację poprzedniej wersji obrazów oraz weryfikację po procesie. Ze względu na działanie w sytuacji DR (disaster recovery) dla środowiska testowego, zbędna jest w tej sytuacji decyzja biznesowa.
6. W przypadku akceptacji nowego wydania, zmiana jest zamykana jako sukces.

Rysunek 8: Schemat procesu CI dla środowiska produkcyjnego.   
Źródło: Zbiór autora

* + - 1. Z perspektywy środowiska produkcyjnego, cześć integracyjna procesu ma zuważalnie mniejszy stopień złożoności.

1. Inicjacja sekwencji rozpoczyna się od decyzji biznesowej, która jest podejmowana zgodnie z harmonogramem wydań oprogramowania.
2. Inżynier DevOps na podstawie zarządzenia, łączy gałąź „release” z „main”, co inicjuję część wdrożeniową.

Rysunek 9: Schemat procesu CD dla środowiska produkcyjnego.   
Źródło: zbiór autora

1. Część wdrożeniowa polega na automatycznej inicjacji składni yaml która podmienia obraz repozytorium produkcyjnego na wersję przetestowaną na środowisku testowym.
2. Inżynier DevOps przeprowadza weryfikację środowiska pod kątem stabilności i kompletności. Rekomendowane są w tym miejscu testy automatyczne wraz z obserwacją i finalizacja przez decyzję manualną.
3. W przypadku istotnych problemów i braku akceptacji inżyniera na emisję wydania, następuję rollback obrazu w repozytorium ECR.
4. AWS Lambda wykrywa zmianę w repozytorium kodu i wysyła sygnał do GitHub w celu rozpoczęcia emisji cofniętego obrazu dockerowego.
5. Inżynier DevOps przeprowadza weryfikację środowiska pod kątem stabilności i kompletności. Rekomendowane są w tym miejscu testy automatyczne wraz z obserwacją i finalizacja przez decyzję manualną.
6. W przypadku akceptacji nowego wydania, zmiana jest zamykana jako sukces.

## Rozwiązanie wyjściowe

### Planowanie bezpieczeństwa

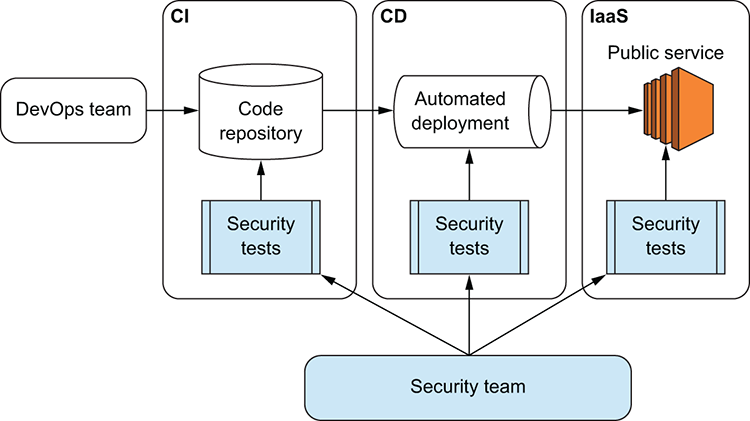
#### Miary bezpieczeństwa procesu CI/CD

* + - 1. W procesie CI/CD (Continuous Integration/Continuous Delivery) wyróżnia się wiele miar bezpieczeństwa, które pozwalają na zabezpieczenie procesu przed nieautoryzowanymi zmianami oraz zapewnienie, że system działa zgodnie z oczekiwaniami. Niektóre z tych miar to:

1. CI/CD pipeline security: polega na zabezpieczeniu procesu CI/CD poprzez wprowadzenie mechanizmów uwierzytelniania, autoryzacji i kontroli dostępu, aby zapewnić, że tylko upoważnione osoby mogą wprowadzać zmiany w kodzie i infrastrukturze.
2. Vulnerability management: dotyczy zarządzania podatnościami w oprogramowaniu i infrastrukturze, poprzez regularne skanowanie i analizowanie kodu oraz bazy danych pod kątem potencjalnych luk w zabezpieczeniach.
3. Code quality and testing: związane z zapewnieniem jakości kodu i testowania poprawności działania systemu. Obejmuje to przeprowadzanie testów jednostkowych, integracyjnych i funkcjonalnych, a także automatyzację testów i wprowadzenie procesów kontroli jakości kodu.
4. Compliance and regulation: dotyczy przestrzegania wymogów regulacji i standardów branżowych, takich jak GDPR czy ISO 27001. Wymaga to wprowadzenia odpowiednich procesów, procedur i narzędzi, które zapewnią zgodność z wymaganiami regulacyjnymi.
5. Incident management and response: związane z szybkim wykrywaniem i reagowaniem na incydenty związane z bezpieczeństwem systemu. Obejmuje to wypracowanie planów awaryjnych i procedur reagowania na incydenty, a także regularne szkolenia dla personelu w celu zwiększenia świadomości na temat zagrożeń i sposobów reagowania.
6. Access control and segregation: polega na wprowadzeniu zasad kontroli dostępu do zasobów i segregacji ról, aby zapobiec nieautoryzowanym modyfikacjom i zapewnić, że tylko upoważnione osoby mają dostęp do odpowiednich zasobów.
7. Security testing and penetration testing: polega na przeprowadzaniu testów penetracyjnych i testów bezpieczeństwa, aby wykryć słabości i podatności systemu. Testy te mogą obejmować symulacje ataków, testy penetracyjne i testy zgodności.
8. Encryption and data protection: dotyczy zabezpieczenia danych przechowywanych i przetwarzanych przez system. Obejmuje to wprowadzenie mechanizmów szyfrowania danych oraz zasad ich przechowywania i przetwarzania.
   * + 1. Wszystkie te miary mają na celu zabezpieczenie procesu CI/CD przed zagrożeniami i zapewnienie, że system działa zgodnie z oczekiwaniami.

#### Planowanie implementacji do procesu CI/CD

* + - 1. Wybranym podejściem implementacyjnym jest metoda TDS (Test Driven Security). Polega ona na określeniu spodziewanego stanu i wprowadzanie systemów kontroli spełnienia wymagań. TDS ma na celu zapewnienie bezpieczeństwa systemów poprzez wczesne wykrywanie luk w zabezpieczeniach i wprowadzanie odpowiednich poprawek już na etapie projektowania i implementacji systemu.
      2. Metoda ma następujące zalety:
* Idealnie sprawdza się w przypadku rozwiązań z zaimplementowanym procesem CI/CD (osiągamy ciągłe skanowanie kodu)
* Możliwość pomiaru procesu i wynikająca z tego łatwość do określenia miejsc do poprawy
* Brak możliwości wypuszczenia kodu wadliwego, z poddatnościami

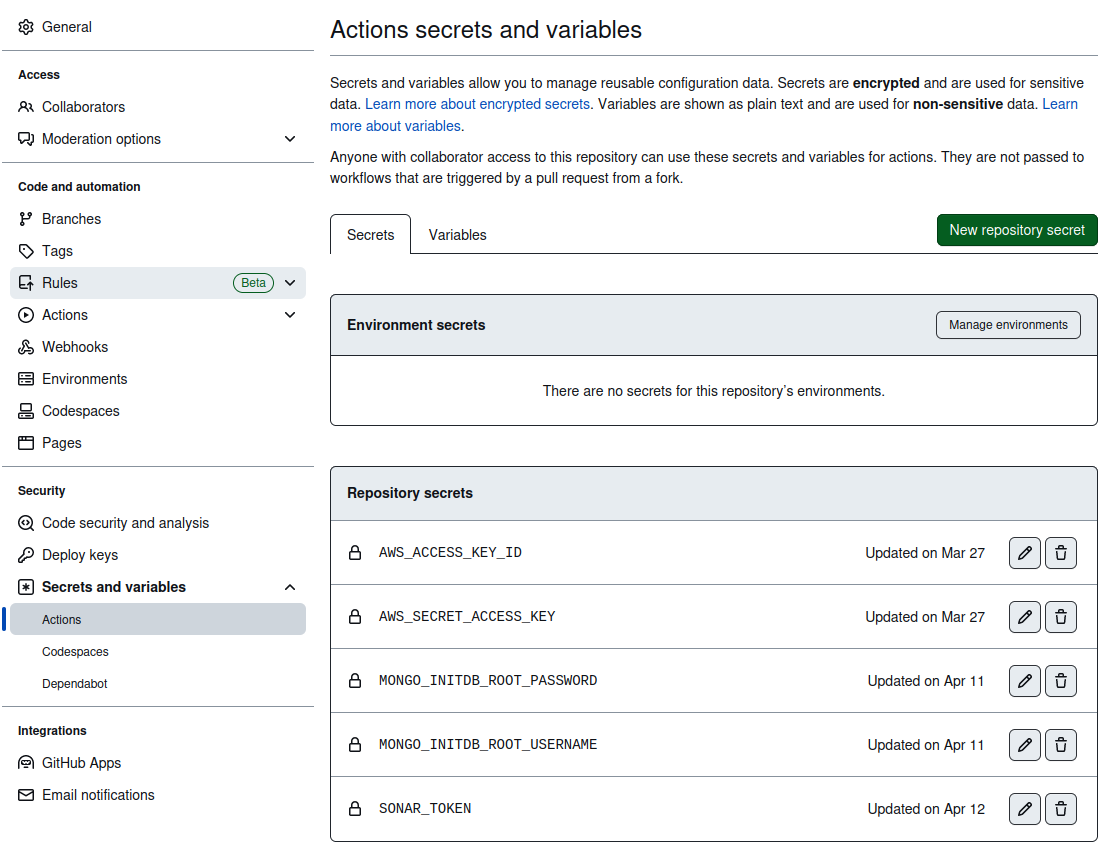
Rysunek 10: Schemat podejścia TDS (Test Driven Development)

Źródło: "Securing DevOps" - Julien Vehent

* + - 1. W metodzie TDS, podobnie jak w TDD (Test Driven Development), programista najpierw pisze testy bezpieczeństwa, które definiują, jakie zachowania i aktywności są bezpieczne, a jakie nie. Dopiero później programista implementuje kod, który będzie spełniał te wymagania. TDS opiera się na ciągłym testowaniu bezpieczeństwa systemu, tak aby wczesne wykrycie potencjalnych zagrożeń umożliwiło szybką reakcję i wprowadzenie poprawek.
      2. Metoda TDS umożliwia także monitorowanie systemu w czasie rzeczywistym i szybką reakcję na ewentualne naruszenia bezpieczeństwa. Dzięki temu, że testy bezpieczeństwa są automatyczne i przeprowadzane w sposób ciągły, metoda TDS przyczynia się do wczesnego wykrywania i usuwania luk w zabezpieczeniach, co minimalizuje ryzyko ataków na system.

#### Zarządzanie sekretami

* + - 1. Jednym z kluczowych zagadnień każdego projektu IT jest zarządzanie sekretami. Wyciek hasła lub klucza serwerowego jest otwarciem drzwi do podsieci.
      2. Rynek oferuję bardzo szeroki zakres narzędzi do zarządzania sekretami. Można tu wymienić między innymi CyberArk, AWS Secrets Manager czy Passowrd Vault od Hashicorp.
      3. W przypadku projektu zastosowano unifikację narzędzi. Podobnie jak w przypadku użycia GitHub Actions, zastosowane zostało również wbudowane narzędzie GitHub do przetrzymywania haseł.

Rysunek 11: Zarządzanie sekretami w GitHub  
 Żródło: zbiór autora

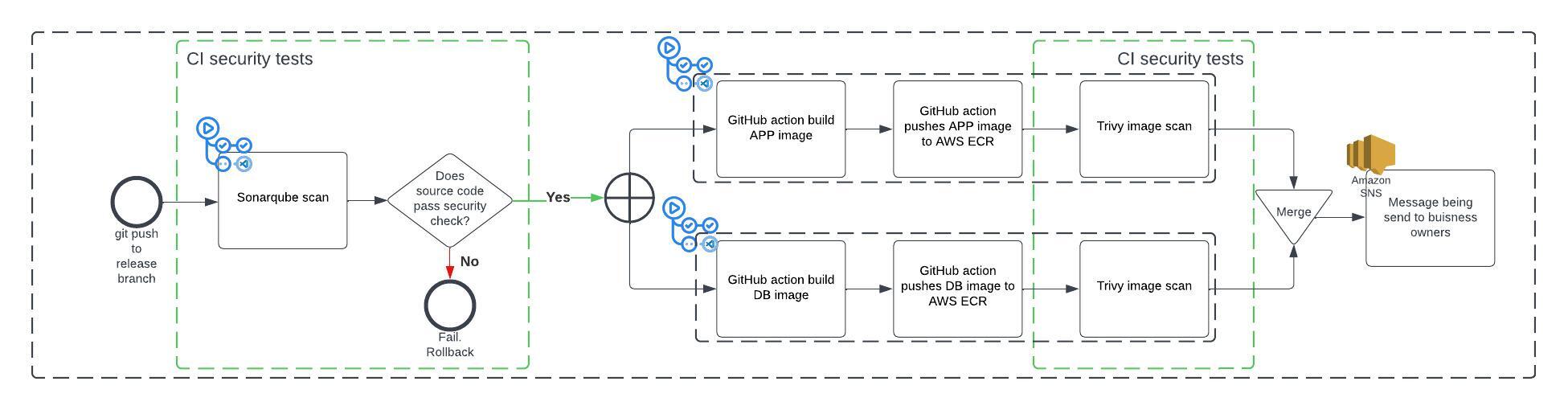
* + - 1. Projekt wymaga przechowywania danych kluczowych dla konta AWS oraz baz danych. Te pierwsze wymagane są do wykonywania zadań związanych z akcjami CI/CD, natomiast pozostałe są wstrzykiwane jako zmienne środowiskowe dla budowanych obrazów.
      2. Pomimo, że hasła znajdują się repozytorium publicznym, są one niedostępne dla nieupoważnionych osób. Kontrola dostępu odbywa się poprzez dodawanie/usuwanie użykowników projektowych oraz poprzez ich role.

### Implementacja DevSecOps w procesie CI/CD

Zgodnie z założeniem projektu, proces został zmodyfkowany o elementy ciągłej kontroli bezpieczeństwa. Ważne jest przedstawienie konstrukcji procesu już na tym etapie, dla lepszego zrozumienia pozostałych rozdziałów.

Wśród głównch podobieństw do procesu wejściowego możemy wymienić:

* Koncepcja podziału na branch „release” oraz „main”
* Rdzeń CI/CD oparty na budowie obrazów
* Budowa oparta na tych samych pipeline’ach, które są w pewnych wypadkach modyfikowane
* Rozdzielenie CI/CD decyzją sektora biznesowego

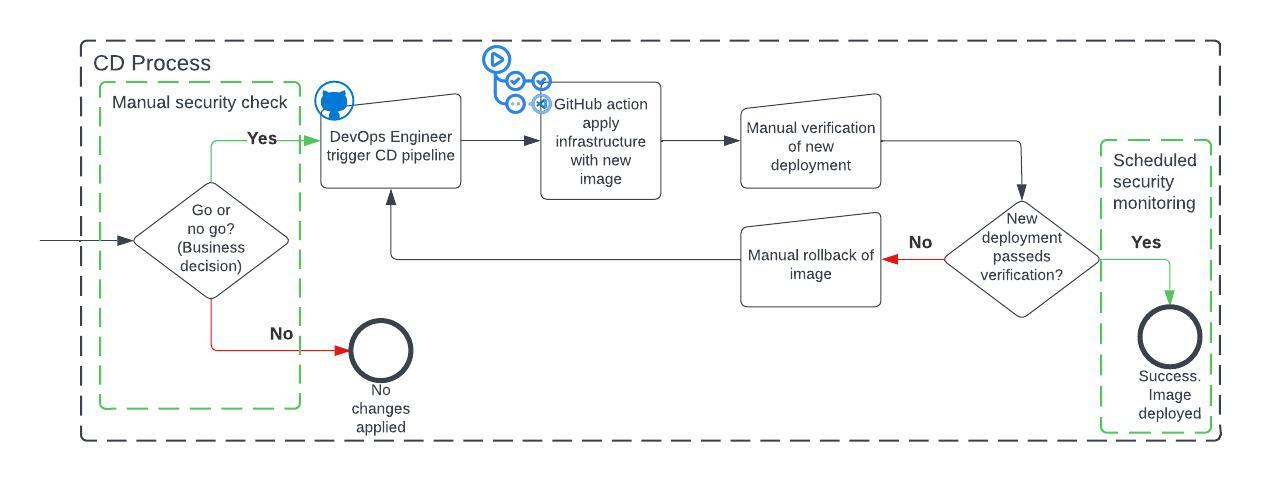
Rysunek 12: Implementacja DevSecOps dla procesu CI/CD. Środowisko testowe.  
Źródło: zbiór autora

1. Proces CI dla środowiska testowego w dalszym ciągu jest inicjowany przez wypuszczenie zmiany na release branch.
2. Zaraz po pojawieniu się kodu w repozytorium następuję główna zmiana w stosunku do poprzedniego procesu. Otóż, inicjowany jest pipeline, który odpowiada za sprawdzenie bezpieczeństwa kodu źródłowego za pośrednictwem narzędzia Sonarqube.
3. W przypadku nie spełnienia wymagań bramki jakościowej – proces jest przerywany, ale zmiana pozostaję w repozytorium. Jest ona nadpisywana przez następne wydanie kodu.
4. Przy pozytywnej ewaluacji, następuje budowa obrazów dockerowych. Budowa obrazów jest uzupełniona o tworzenie raportu poddatności obrazów kontenera przy użyciu narzędzia Trivy.
5. Raport jest wysyłany do repozytorium GitHub, natomiast obraz niezależne od wyniku jest zapisywany w AWS ECR i jest wymieniany przy kolejnej budowie obrazu.
6. Informacja o nowych obrazach oraz o wygenerowaniu nowego raportu jest kierowana do sektora binzesowego.

W przypadku procesu ciągłego dostarczania, zmiany są mniej znaczące i ograniczają się do do pewnych weryfikacji manualnych. W przypadku procesu integracji kodu z uwzględniem bezpieczeństwa, istnieją dwa dodatkowe zawory bezpieczeństwa. Pomimo faktu istnienia tych zabezpieczeń, zakładamy rozpoczęcie procesu dostarczenia nowego wydania, tylko pod warunkiem manualnego zweryfikowania raportów bezpieczeństwa. Proces biznesowy w tym miejscu jest podejmowany w uzgodnieniu z inżynierem.

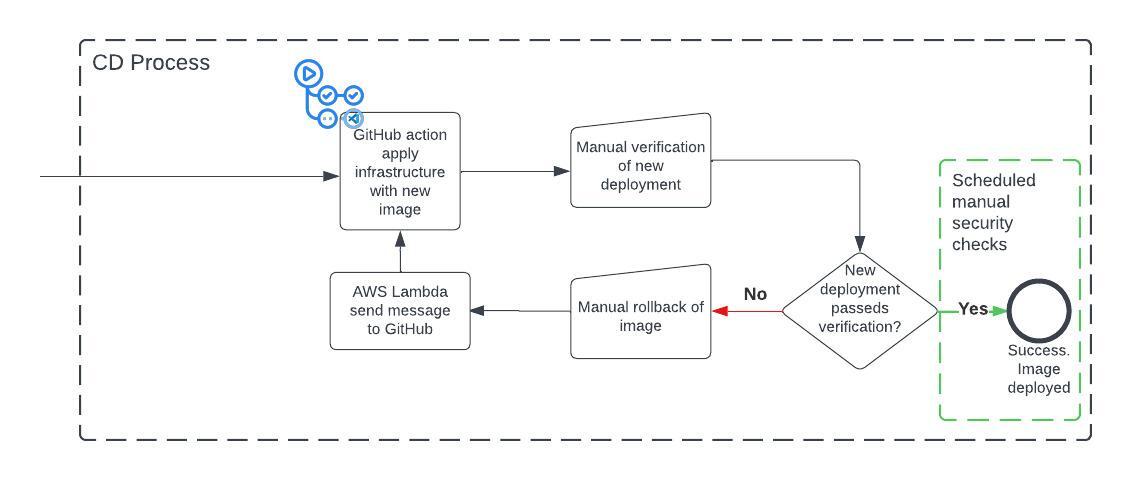
Manualna weryfikacja na poziomie etapy rozpoczączecia procesu ma następujące zalety:

* Zabezpieczenie przed ewentualnym błędem cyklu automatycznego
* Analiza raportu zwiększa świadomość i wiedzę w zakresie bezpieczeństwa
* Pozwala na wykrycie błędów funkcjonalnych aplikacji

Rysunek 13: Implementacja DevSecOps dla procesu CD. Środowisko testowe.

Źródło: zbiór autora

Obok manualnej weryfikacji inicjujący proces, konieczny jest proces ciągłej weryfikacji po zakończeniu deploymentu. Rekomendowane jest wyznaczenie rotowowalnej roli zespołowej, odpowiedzialnej za raportowanie podczas danego sprintu. Takie rozwiązanie uwzględnia inną kolejną istotnę funkcję polegająca na dzieleniu wiedzą i wzroście każdego członka zespołu.

Rysunek 14: Implementacja DevSecOps dla procesu CD. Środowisko produkcyjne.

Źródło: zbiór autora

Sytuacja wygląda innaczej w przypadku środowiska produkcyjnego. Dla tej części procesu, przyjęte zostało założenie, że powielony i użyty obraz został zweryfikowany na wcześniejszym etapie. W tym miejscu konieczne jest uzupełnienie jedynie o cześć stałej, zaplanowej weryfikacji manualnej, wykonywany nawet w przypadku wypuszczenia wersji na środowisko produkcyjne.

### Zastosowanie narzędzi wspomagających DevSecOps

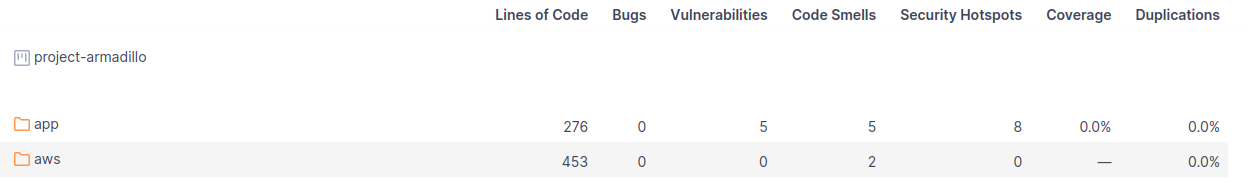
#### Zastosowanie narzędzia Sonarqube

Pomimo dążenia do mikroserwisów, kod źródłowy dzisiejszych aplikacji to nadal wiele wierszy obfitujących w funkcję zaczerpniętę z różnorodnych bibliotek. Okazuję się, że człowiek w swej prostocie nie jest w stanie zweryfikować całości rozwiązania, oraz jest podatny na błąd ludzki. W kontrze pojawia się koncepcja: „continuous security” która opiera się na automatycznym, maszynowym sprawdzaniu oraz recenzowaniu kodu źródłowego.

Narzędzie Sonarqube to platforma typu „open-source” stworzona przez firmę SonarSource. Cechuję się szczególną prostotą integracji z procesem CI/CD, funkcjonalną i przejrzystą wirtualizacją danych oraz wsparciem bardzo szerokiej gamy językow programowani.

Aby móc mierzyć proces usprawniania kodu źródłowe, konieczne jest zaznajomienie z metryki które owy opisują. W przypadku Sonarqube są to:

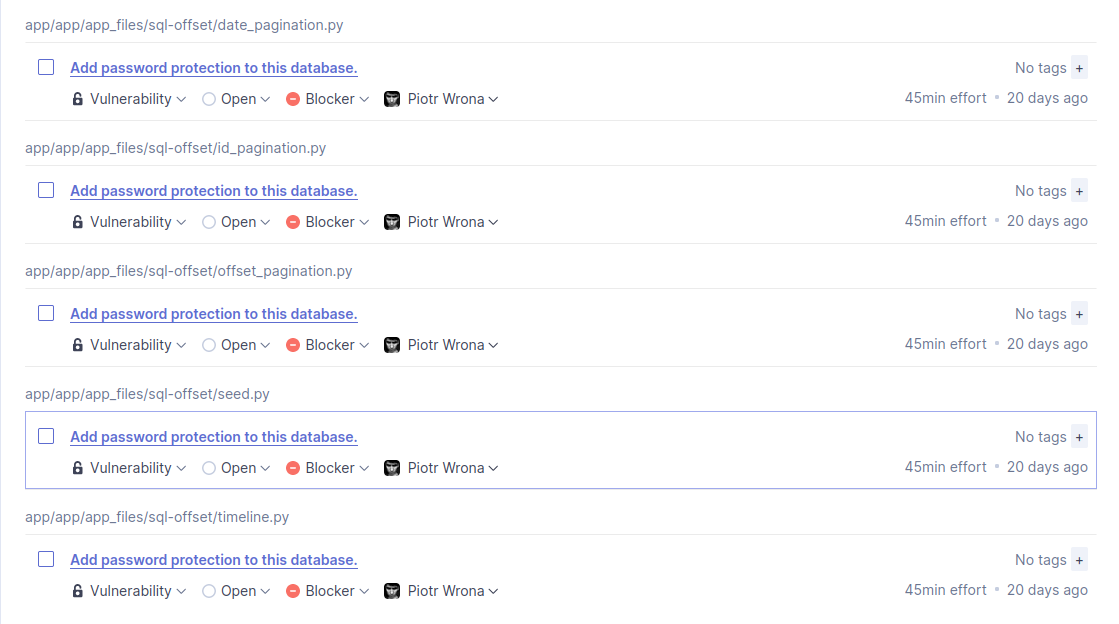
* + **Bugs** – błędy które mogą spowodować nagłą awarię aplikacji i muszą zostać rozwiązane natychmiastowo.
  + **Vulnerabillities** – są tą fragmenty kodu poddatne na ataki hackerów.
* **Code Smells** – fragmenty kodu które są mylące i mogą być trudne w utrzymaniu.
* **Security Hotspots** – wrażliwe na bezpieczeństwo fragmenty, które wymagają manualnej recenzji, bez znaczenia czy problem wystąpił jako poddatność.
* **Coverage** – miara określająca stopień pokrycia kodu przez testy.
* **Duplications –** określa ilość zduplikowanych fragmentów kodu.
  + - 1. Z punktu widzenia bezpieczeństwa kodu, istotne są głównie „Bugs”, „Vulnerabillities” oraz „Security Hotspots”. W praktyce należy wykorzystać możliwości, które dostarcza nam oprogramowanie i skupić się na raportach dotyczących wszystkich dostarczonych miar.

Rysunek 15: Przegląd miar narzędzia Sonarqube.

Źródło: https://sonarcloud.io

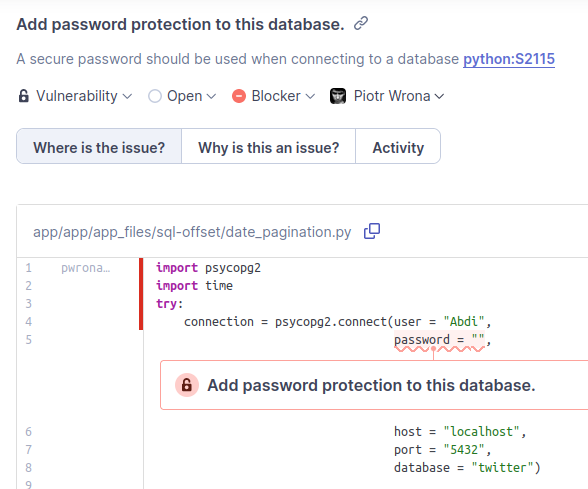
Wynik analizy wskazuję na pięć poddatności oraz osiem fragmentów o dużej wrażliwości, które powinny być niezwłocznie zbadane.

Głębszy raport wskazuję na naruszenie protokołów: OWASP A2, A3[[2]](#endnote-3) oraz CWE-521[[3]](#endnote-4). Pierwsze dwa kolejną odpowiadają za złamaną autentykację oraz udostępnienie danych wrażliwych. Natomiast ostatni, polega na złamaniu polityki dotyczącej wykorzystywania silnych haseł.

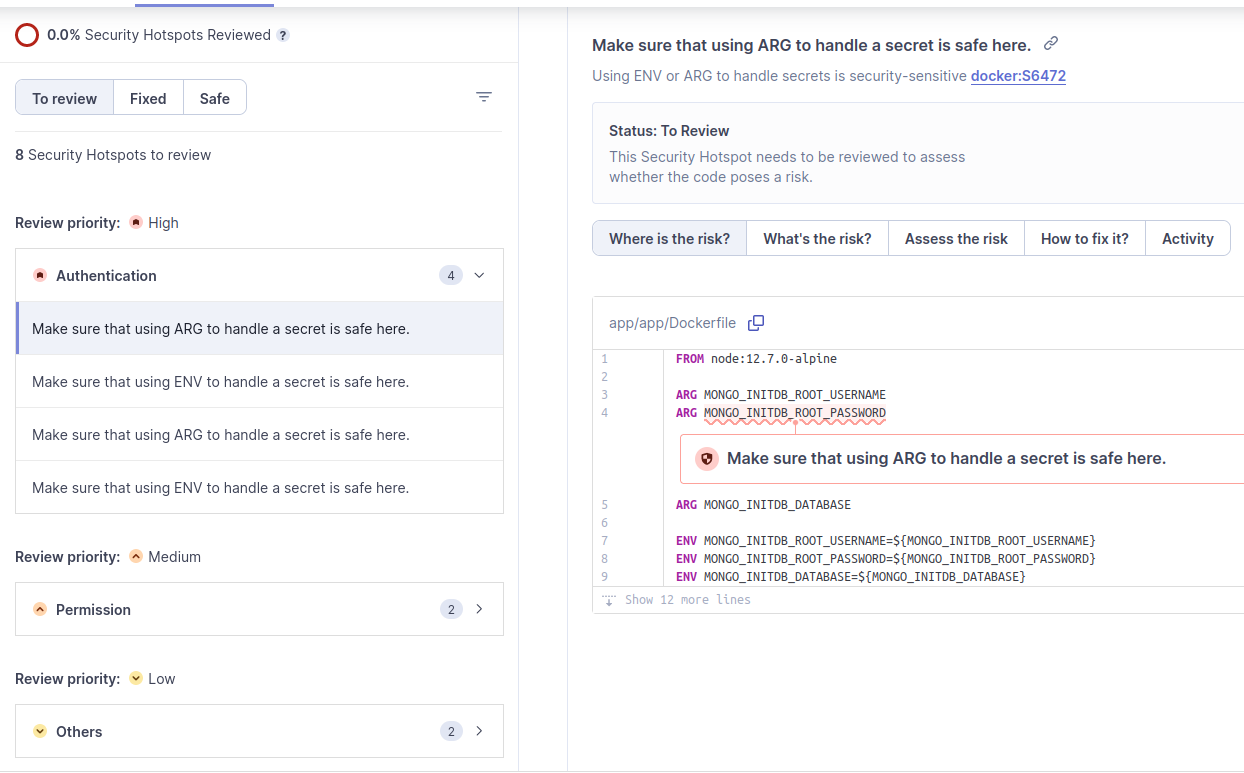


Rysunek 16: Spis poddatności kodu źródłowego

Źródło: https://sonarcloud.io

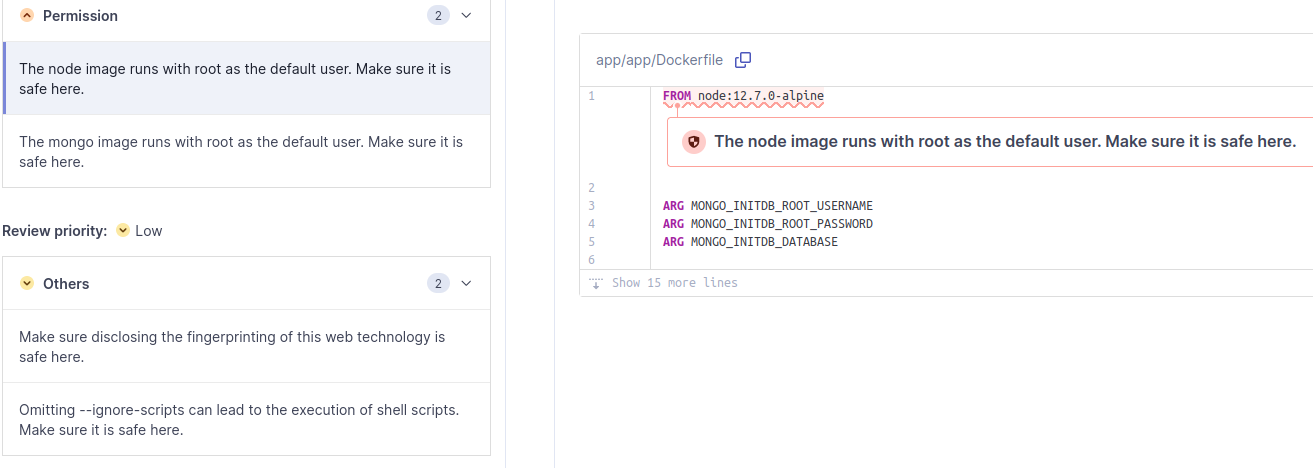
Rysunek 17: Szczegółowe przedstawienie poddatności  
Źródło: https://sonarcloud.io

* + - 1. Warto zwrócić uwagę na zakładki: „Where is the issue?”, „Why is this an issue?” i „Activy”, gdyż są bardzo pomocne dla lokalizacji oraz zrozumienia błędu. Dodatkowo, ostatnia zakładka pozwala na komunikację zespołową i informowaniu o podjętych akcjach dla rozwiązania problemu.
      2. Drugą miarą są „Security Hotspots”, czyli fragmenty potencjalne niebezpieczne i wymagające manualnej recenzji. Części te są podzielowe w zależności od ich priorytetu. W poniższym przypadku najwyższą wagę ma zabezpieczenie hasła do bazy danych. Rozwiązanie problemu zostało przedstawione w sekcji: „Zarządzanie sekretami”.



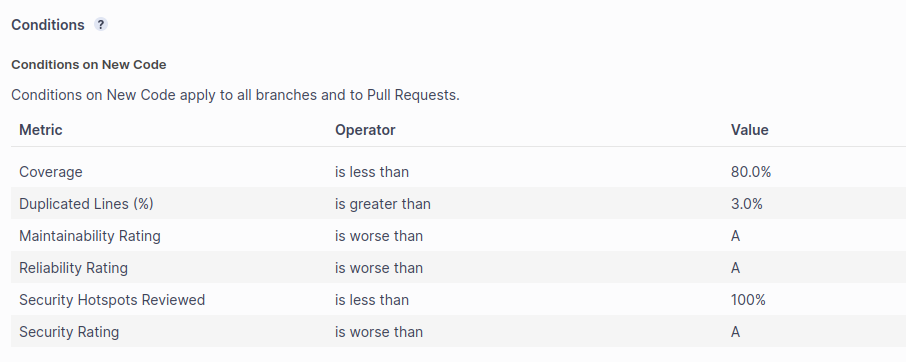
Rysunek 18: Spis "Security Hotspots" kodu żródłowego. Sekcja: priorytet wysoki.  
Źródło: https://sonarcloud.io

* + - 1. Raport wskazuję również na użycie konta root jako domyślnego użytkownika dla kontenerów ale przypisuje mu priorytet średni. Pomimo to, zagrożenie jest realne. W przypadku w którym hacker dostał się do sieci publicznej i zna adres IP, możliwe są próby wejścia na kontener. W przypadku sukcesu, włamywacz ma zdecydowanie szersze uprawnienia, które są szczególnie pomocne dla późniejszego opanowania kontenera bazy danych.

Rysunek 19: Spis "Security Hotspots" kodu żródłowego. Sekcja: priorytet średni/niski.  
Źródło: https://sonarcloud.io

#### Kontrola jakości kodu źródłowego

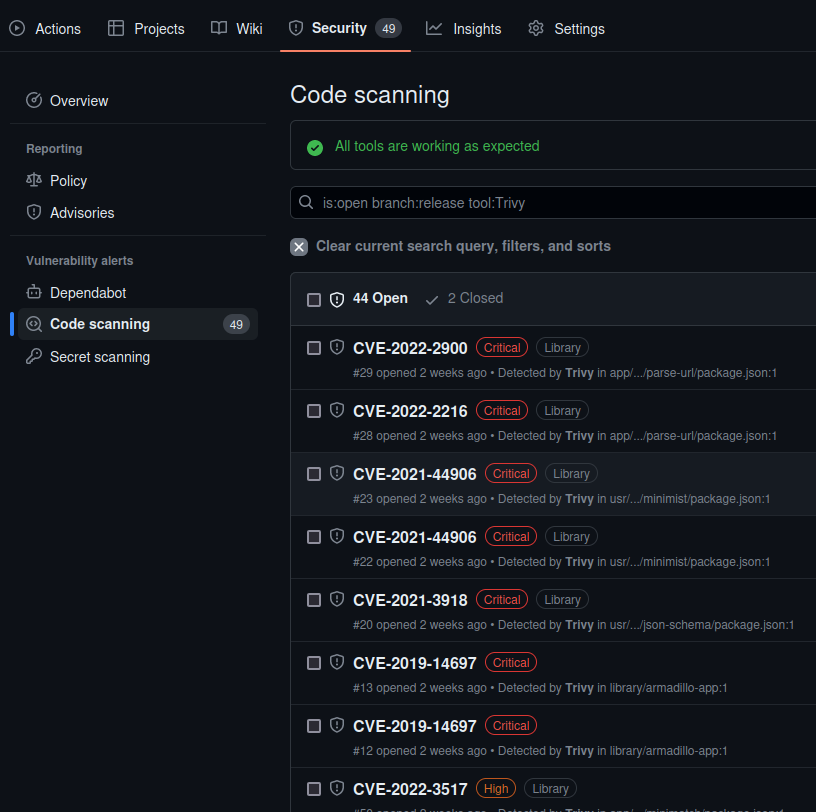
* + - 1. Podejście TDS wymaga wprowadzenia progów jakościowych, dlatego w tym podrozdziale przedstawione zostaną zalety stosowania bramki jakościowej SonarQube w procesie wytwarzania oprogramowania. Omówione zostaną również najważniejsze funkcjonalności tego narzędzia.
      2. Zalety stosowania bramek jakościowych:
* Definiowanie standardów jakości: Bramka jakościowa obejmuje ustalanie standardów jakości dla produktów lub usług. Określa się kryteria, wymagania i wytyczne dotyczące jakości, które muszą być spełnione.
* Ocena i weryfikacja: Bramka jakościowa przeprowadza ocenę i weryfikację produktów lub usług w celu sprawdzenia, czy spełniają ustalone standardy jakości. Może to obejmować przegląd dokumentów, testy, inspekcje, weryfikację spełnienia wymagań itp.
* Raportowanie i monitorowanie: Bramka jakościowa dostarcza raportów na temat oceny jakości oraz wyników weryfikacji. Monitoruje również postęp projektu pod kątem jakości, identyfikuje problemy i podejmuje działania naprawcze.
* Decyzje i akceptacja: Bramka jakościowa jest odpowiedzialna za podejmowanie decyzji dotyczących akceptacji lub odrzucenia produktów lub usług na podstawie oceny jakości. Decyzje te są podejmowane w oparciu o ustalone standardy jakości i wyniki weryfikacji.
* Doskonalenie procesu: Bramka jakościowa może również odgrywać rolę w doskonaleniu procesów projektowych i produkcyjnych. Poprzez analizę wyników i wniosków z oceny jakości, bramka jakościowa może wprowadzać ulepszenia i rekomendacje mające na celu poprawę jakości w przyszłych projektach.

Rysunek 20: Progi bramki jakościowej kodu żródłowego  
Źródło: https://sonarcloud.io

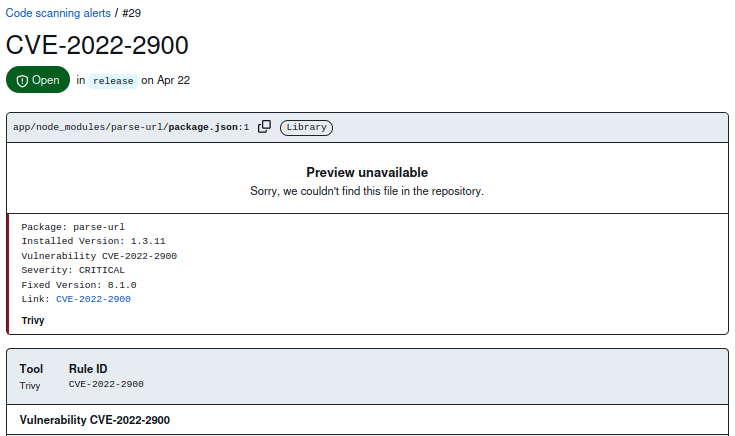
#### Zastosowanie narzędzia Trivy

* + - 1. W dzisiejszych czasach, wraz z rosnącą popularnością kontenerów, bezpieczeństwo staje się coraz bardziej kluczowym aspektem w kontekście zarządzania infrastrukturą IT. Jednym z elementów zwiększających bezpieczeństwo kontenerów jest wykorzystanie narzędzi do skanowania obrazów w celu wykrycia potencjalnych luk w zabezpieczeniach. Jednym z takich narzędzi jest Trivy - otwarte i łatwe w użyciu narzędzie do skanowania obrazów kontenerów pod kątem luk w zabezpieczeniach. W tym rozdziale omówimy, jak Trivy może zostać wykorzystany w celu zwiększenia bezpieczeństwa obrazów kontenerów, jakie są jego główne funkcje oraz jakie korzyści wynikają z jego stosowania.
      2. Zastosowania narzędzia Trivy:
* obrazów kontnera
* filesystem
* repozytorium GIT
* obrazy maszyn wirtualnych
* Kubernetes’a
* AWS
  + - 1. Głównym założenie Trivy jest analiza następijących niebezpieczeństw:

1. Podatności na ataki - Trivy skanuje obraz w celu wykrycia znanych podatności na ataki, które mogą umożliwić atakującemu zdalne wykonanie kodu lub uzyskanie dostępu do systemu.
2. Niezaktualizowane zależności - Trivy analizuje zależności używane przez obraz i sprawdza, czy są one zaktualizowane do najnowszych wersji. Nieaktualne zależności mogą stanowić potencjalne zagrożenie dla bezpieczeństwa.
3. Zainstalowane oprogramowanie - Trivy skanuje obraz w poszukiwaniu zainstalowanego oprogramowania, takiego jak biblioteki czy narzędzia deweloperskie, które mogą być potencjalnym celem ataku.
4. Znane podatności bezpieczeństwa - Trivy korzysta z różnych źródeł informacji, takich jak bazy danych podatności CVE (Common Vulnerabilities and Exposures), aby wykryć znane zagrożenia dla bezpieczeństwa.
5. Niezabezpieczone konfiguracje - Trivy sprawdza, czy obraz kontenera jest zabezpieczony poprzez weryfikację konfiguracji bezpieczeństwa, takich jak wyłączenie niepotrzebnych serwisów czy ustawienia uprawnień.
   * + 1. Implementacja projektowa polega na budowie tymczasowego kontenera i jego skanowaniu przy użyciu narzędzia Trivy. Proces jest wprzęgnięty w akcję Github, a raporty są gromadzone w zakładce Security na stronie głównej projektu. Dostęp do wyników jest kontrolowany poprzez licencję dostępowe.

Rysunek 21: Raport poddatności narzędzia Trivy  
Źródło: https://github.com/PiotrLotr/project-armadillo

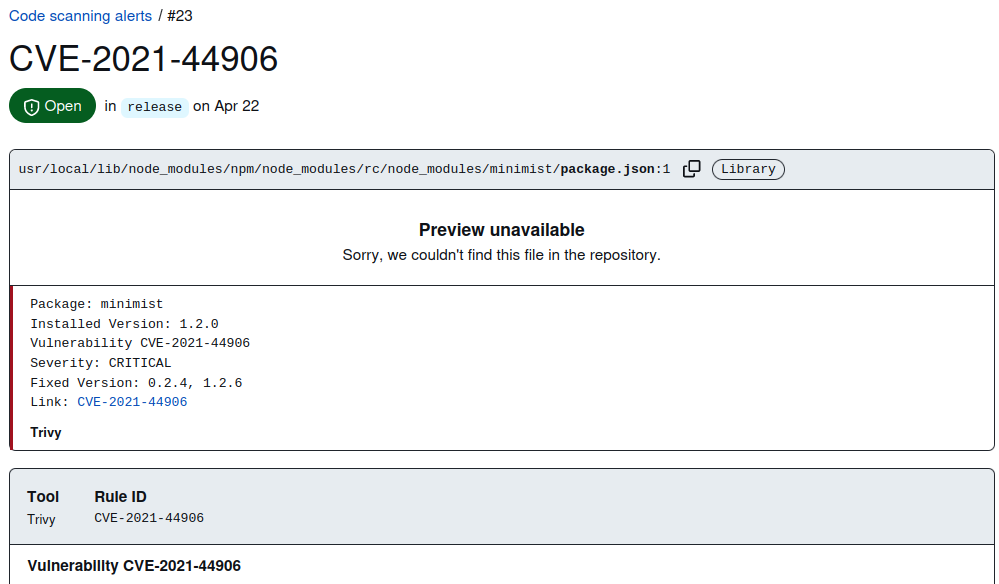
Na pierwszy rzut oka zauważalne są głównie wrażliwości z zakresu CVE (Common Vulnerabilities and Exposures). CVE to numery identyfikacyjne nadawane konkretnym podatnościom oprogramowania lub systemów informatycznych. Służą one do ujednolicenia sposobu identyfikacji i opisu zagrożeń, co ułatwia ich wymianę między różnymi podmiotami (np. producentami, administratorami systemów, badaczami bezpieczeństwa). Dzięki temu CVE pomaga w szybszym i skuteczniejszym reagowaniu na zagrożenia oraz w zapobieganiu ich wykorzystywaniu przez cyberprzestępców.

Rysunek 22: Przedstawienie polityki CVE-2022-2900. Server-Side Request Forgery (SSRF)  
Źródło: https://github.com/PiotrLotr/project-armadillo

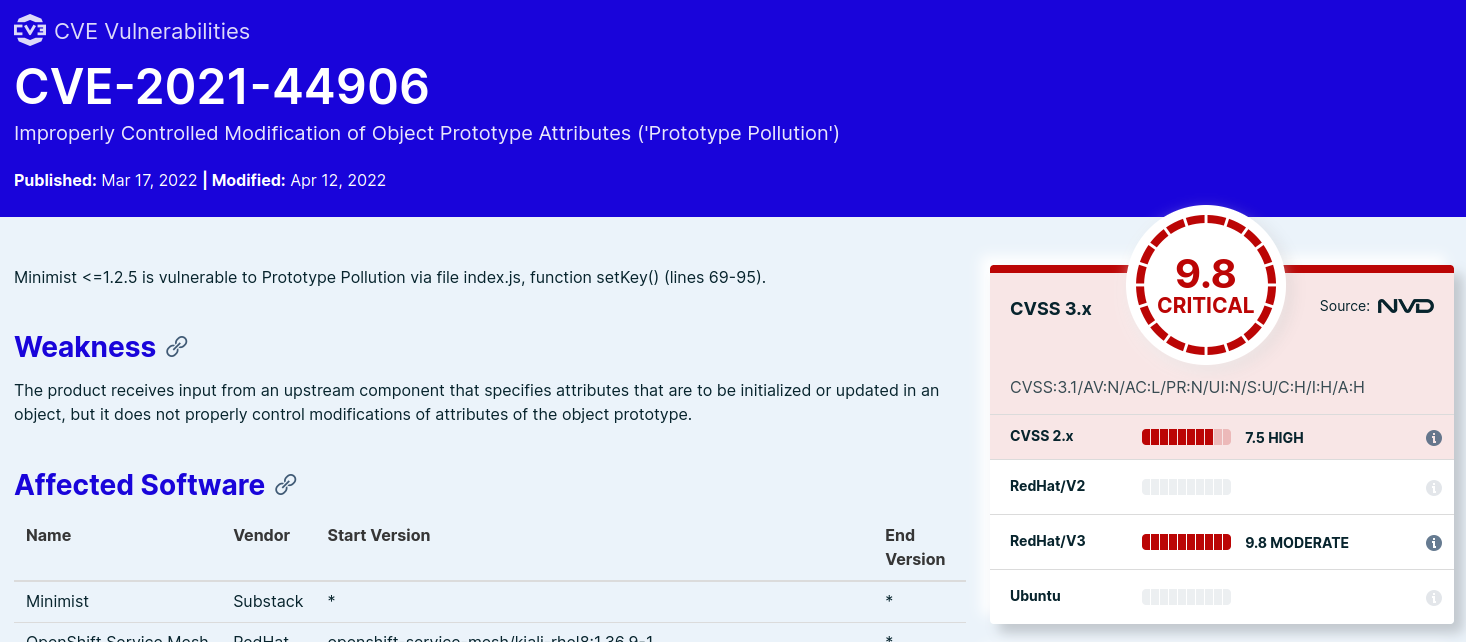
* + - 1. Jednym z pierwszych komunikatów jest naruszenie protokołu CVE-2022-2900. Dotyczy on naruszenia SSRF (Server Side Request Forgery).
      2. Atak SSRF (Server-Side Request Forgery) polega na wykorzystaniu nieodpowiednio zabezpieczonej aplikacji internetowej, aby zmusić serwer do wykonania niechcianych żądań sieciowych z jego perspektywy, zamiast z perspektywy użytkownika.

Rysunek 23: Przedstawienie polityki CVE-2022-2900. Server-Side Request Forgery (SSRF)  
Źródło: https://avd.aquasec.com/nvd/2022/cve-2022-2900/

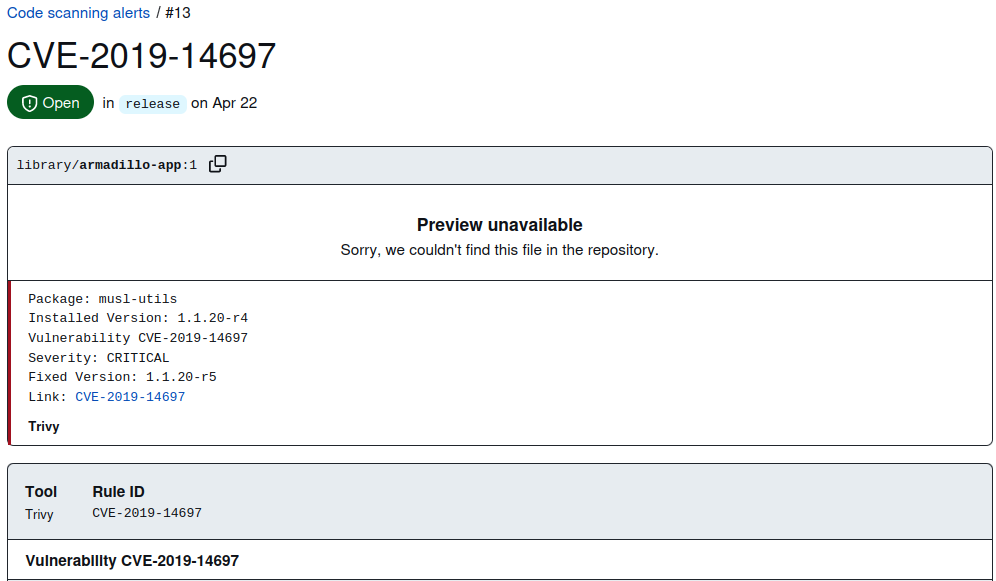
* + - 1. Atakujący może wprowadzić do aplikacji internetowej fałszywe żądanie HTTP, które zawiera szkodliwy kod. Serwer, wykonując to żądanie, może spowodować, że wykona się szkodliwe działanie, np. pobieranie poufnych informacji z innej usługi lub sieci wewnętrznej, przeprowadzanie ataków DDoS lub nawet uzyskanie pełnej kontroli nad serwerem.
      2. Atak SSRF jest szczególnie niebezpieczny, ponieważ może umożliwić atakującemu uzyskanie dostępu do wewnętrznych zasobów serwera, których normalnie nie powinien mieć dostępu. Aby zapobiec atakowi SSRF, ważne jest, aby aplikacje internetowe weryfikowały i filtrowały żądania HTTP, a także stosowały odpowiednie zabezpieczenia sieciowe.

Rysunek 24: Przedstawienie polityki CVE-2021-44906. Podatność „Prototype pollution”  
Źródło: https://github.com/PiotrLotr

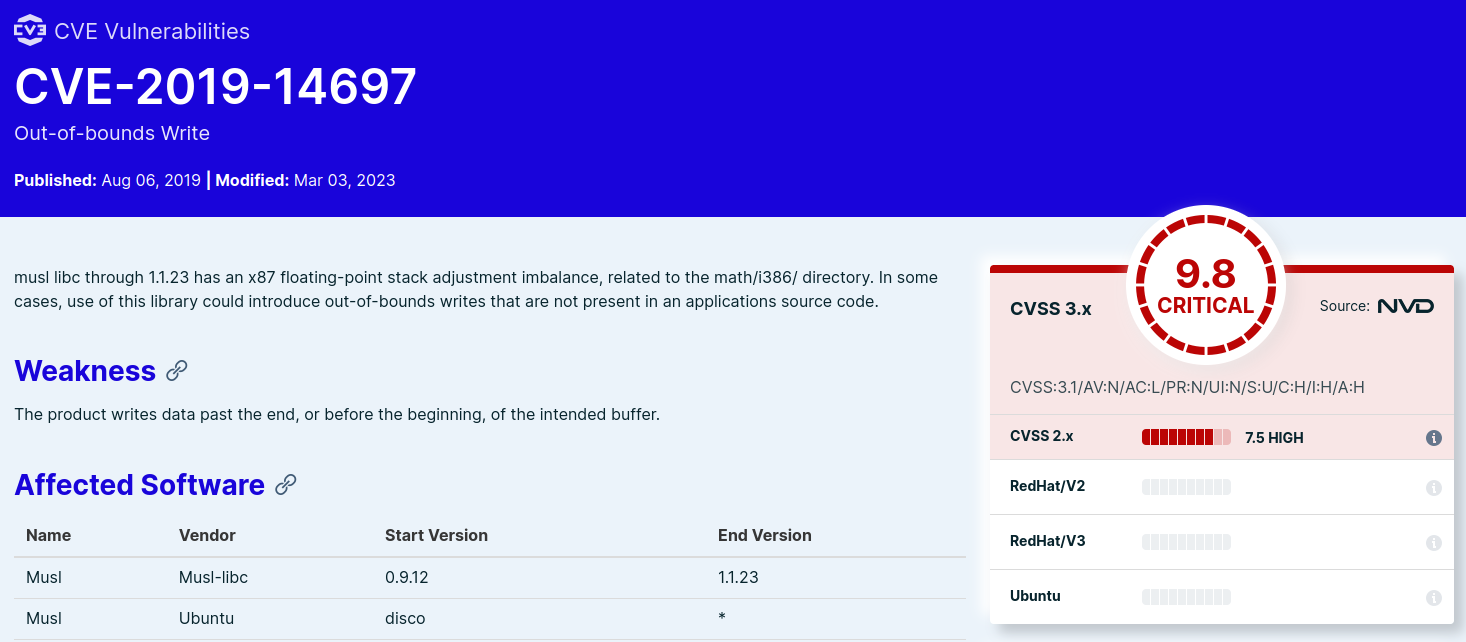
* + - 1. Kolejną podatnością jest: CVE-2021-44906, która jest oznaczona jako priorytet krytyczny. Alert oznacza wrażliwość typu: „Protototype poluttion” co w tłumaczeniu oznacza zanieczyszczenie prorotypu.
      2. Prototype pollution to atak typu wstrzykiwanie kodu, który polega na zmianie właściwości obiektów JavaScript poprzez modyfikację ich prototypów. Atakujący może użyć nieodpowiednio skonstruowanego kodu, aby wprowadzić niechciane zmiany do obiektów JavaScript, co może prowadzić do niespodziewanych i niepożądanych zachowań aplikacji.
      3. Zasadniczo, każdy obiekt JavaScript ma prototyp, który definiuje jego właściwości i metody. W przypadku ataku na prototyp, atakujący może wprowadzić niechciane zmiany do prototypu, które zostaną odzwierciedlone we wszystkich obiektach dziedziczących z tego prototypu.
      4. Atak na prototyp może mieć globalny wpływ na całą aplikację. Przykładem takiego ataku może być modyfikacja prototypu obiektu Array, aby dodać nieoczekiwane i szkodliwe metody. Jeśli aplikacja korzysta z tak z modyfikowanego obiektu Array, to atakujący może uzyskać nieuprawniony dostęp do wrażliwych danych lub naruszyć integralność aplikacji.

Rysunek 25: Przedstawienie polityki CVE-2021-44906. Podatność „Prototype pollution”  
Źródło: https://avd.aquasec.com/nvd/2021/cve-2021-44906/

* + - 1. Aby chronić się przed atakiem prototype pollution, ważne jest, aby aplikacje JavaScript filtrowały dane wejściowe, nie zaufały niewiarygodnym źródłom i używały odpowiednich metod do tworzenia i modyfikowania obiektów.
      2. Raport wskazuję również na złamanie poddatności CVE-2019-14697. Zagrożenie to określane jest jako: „Out-of-bounds Write”.

Rysunek 26: Przedstawienie polityki CVE-2019-14697. Podatność „Out-of-bounds Write”  
Źródło: https://github.com/PiotrLotr

* + - 1. Out-Of-Bounds Write (OOBW) to błąd programistyczny, który występuje, gdy program zapisuje dane poza zaalokowanym obszarem pamięci. W takim przypadku program może nadpisać lub uszkodzić ważne dane, co może prowadzić do nieprzewidywalnych zachowań programu, w tym do awarii systemu lub zagrożenia bezpieczeństwa.
      2. Przykładem OOBW może być sytuacja, gdy programista zapomni uwzględnić granice tablicy przy jej przetwarzaniu i zapisze wartość poza jej zadeklarowanym zakresem. Może to spowodować, że zostaną nadpisane dane w innych zmiennych lub wrażliwe informacje, takie jak dane uwierzytelniające, zostaną zmienione.

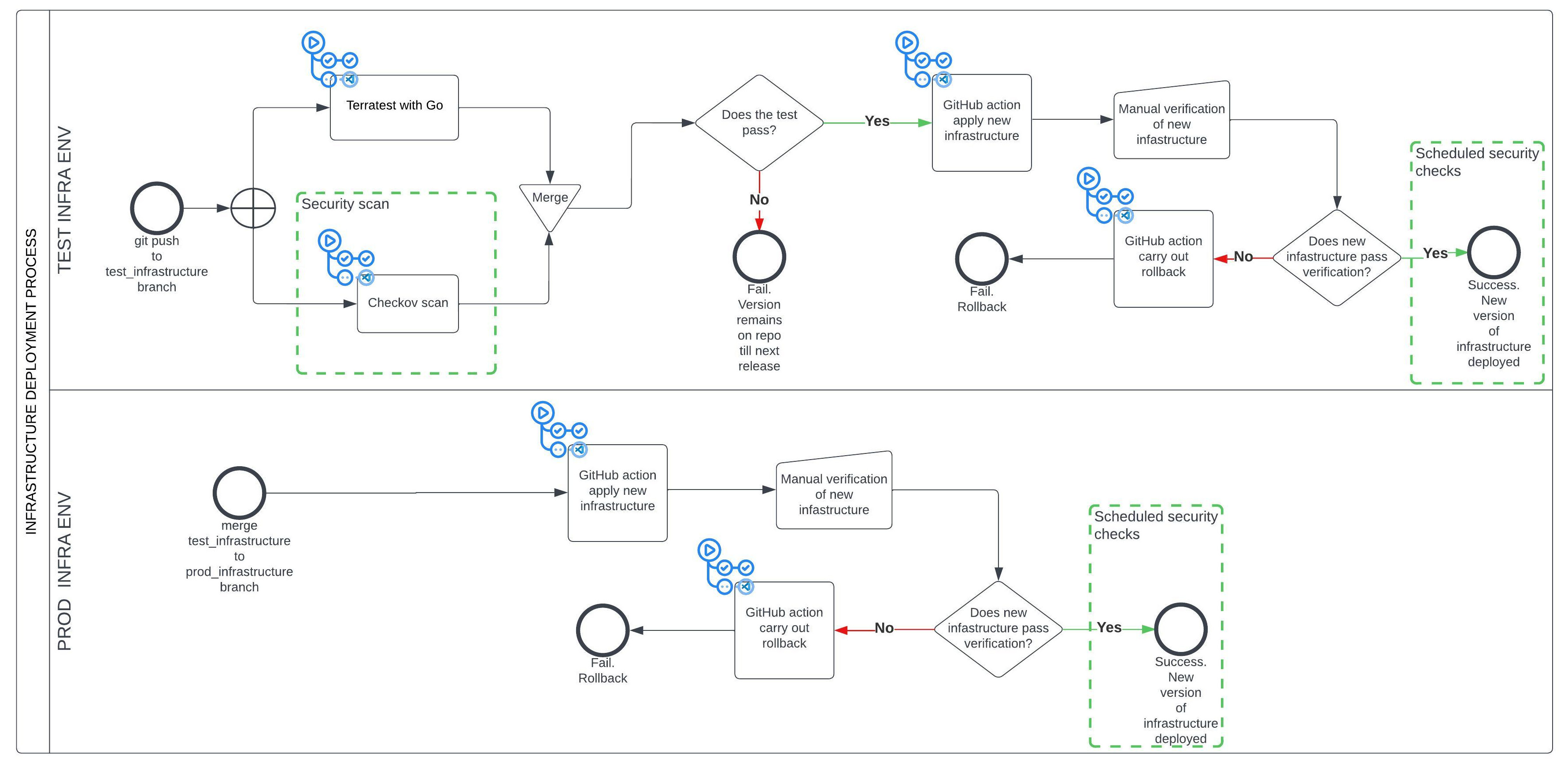
Rysunek 27: Przedstawienie polityki CVE-2019-14697. Podatność „Out-of-bounds Write”  
Źródło: https://avd.aquasec.com/nvd/2019/cve-2019-14697/

* + - 1. OOBW jest często wykorzystywany przez atakujących do wykonania tzw. ataków typu buffer overflow, w których złośliwy kod może nadpisać ważne dane lub wstrzyknąć kod do pamięci programu i tym samym uzyskać kontrolę nad systemem.
      2. Dlatego ważne jest, aby programiści starali się unikać OOBW poprzez zawsze uwzględnianie granic tablic i innych obszarów pamięci oraz stosowanie bezpiecznych funkcji bibliotecznych, które wykonują sprawdzenie granic. Ponadto, stosowanie mechanizmów kontroli błędów, takich jak ASLR (Address Space Layout Randomization) czy DEP (Data Execution Prevention), może pomóc w zapobieganiu wykorzystaniu błędów OOBW przez atakujących.

### Implementacja DevSecOps w infrastrukturze

#### Implementacja DevSecOps w procesie dostarczania infrastruktury

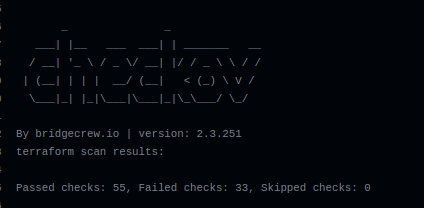
* + - 1. W przypadku procesu wyjściowego, zostaje utrzymana koncepcja dwóch gałęzi: testowej oraz produkcyjnej. Zasadniczą różnicą w procesie jest równoległe wykonanie Terratestu oraz nowego elementu – skanu przy użyciu narzędzia Checkov.

Rysunek 28: Implementacja DevSecOps do procesu dostarcza infrastruktury. Środowisko testowe i produkcyjne.  
Źródło: zbiór autora

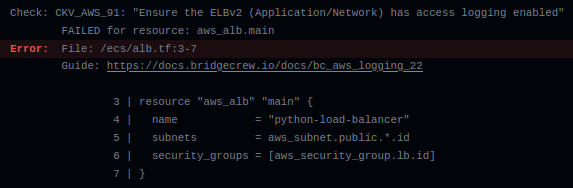
* + - 1. Kontunuacja procesu dotarczania infrastruktury następuję jedynie po złączeniu wyników równoległego przetwarzania, a wyniki są poddawane walidacji. W przypadku sukcesów w dla obu procesów – następuję automatyczny proces dostarczenia zmian. W przypadku porażki kod pozostaję w repozytorium do jego kolejnego nadpisania, natomiast procedura się kończy.
      2. Podobieństwa w stosunku do procesu dostarczania aplikacji, występuję w postaci pewnych zaplanowanych manualnych kontrolii bezpieczeństwa po zakończeniu procesu. Bezpiecznik występuję zarówno dla gałęzi „test”, jak i „prod”.

#### Zastosowanie narzędzia Checkov

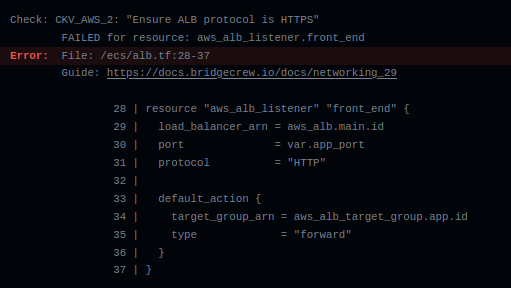
* + - 1. Checkov to narzędzie open-source do analizy infrastruktury w chmurze pod kątem zgodności z najlepszymi praktykami bezpieczeństwa i zalecaniami. Jest to framework zaprojektowany specjalnie dla inżynierów chmurowych, deweloperów i zespołów DevOps, którzy chcą zweryfikować, czy ich infrastruktura w chmurze jest zabezpieczona i zgodna z odpowiednimi standardami.
      2. Checkov automatycznie skanuje pliki konfiguracyjne takie jak Terraform, Kubernetes YAML i inne, analizuje je pod kątem różnych luk w zabezpieczeniach, nieprawidłowych ustawień i podatności. Wyniki analizy są prezentowane w formie raportów, które umożliwiają łatwe zidentyfikowanie i naprawę ewentualnych problemów. Dzięki Checkov, użytkownicy mogą skutecznie weryfikować bezpieczeństwo swojej infrastruktury w chmurze i dostosowywać ją do najlepszych praktyk.

Rysunek 29: Stosunek wyników pozytywnych/negatywnych w raporcie Checkov  
Źródło: https://github.com/PiotrLotr/project-armadillo

* + - 3. Pierwsze sprawdzenie infrastruktury, wykazało 33 nieudanych sprawdzeń, w stosunku to 55 które zostały przeprowadzone. Należy pamiętać, że nie wszystkie próby odnoszą się do bezpieczeństwa infrastruktury. Poniżej zostaną przytoczone niektóre z nich.

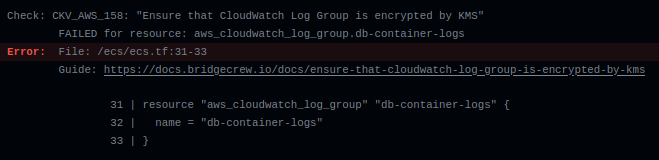
Rysunek 30: Przedstawienie polityki CKV\_AWS\_91.  
Źródło: https://github.com/PiotrLotr/project-armadillo

* + - 1. Polityki narzędzia Checkov są dokumentowane i indeksowane wzorem CKV\_AWS\_XX. Reguły te odpowiadają dobrym praktykom, opracowanych przez ekspertów Bridgecrew (obecnie część firmy Palo Alto Networks).
      2. Reguła CKV\_AWS\_91 informuję, iż tworzony przez Terraform zasób: ALB nie posiada parametru włączającego logowanie ruchu wejściowego. Taka informacja jest istotna z bardzo wielu powodów:
* Pozwala na monitoring błędów 5xx oraz 4xx oraz szybkie reagowanie na problemy
* Wykrywanie potencjalnych ataków DdoS
* Prowadzenie statyski dotyczącej najczęściej odwiedzanych adresów

Rysunek 31: Przedstawienie polityki CKV\_AWS\_2. Brak zapewnienia protokułu HTTPS

Źródło: https://github.com/PiotrLotr/project-armadillo

* + - 1. Reguła CKV\_AWS\_2 wymaga zapewnienia protokołu HTTPS. W przypadku HTTPS, dane są szyfrowane przy użyciu protokołu SSL (Secure Sockets Layer) lub jego następcy, protokołu TLS (Transport Layer Security), zanim są wysyłane przez sieć. Szyfrowanie danych sprawia, że są one praktycznie nieczytelne dla osób trzecich, które mogą przechwytywać przesyłane pakiety i zapobiega atakom typu: „Man in the middle”.

Rysunek 32: Przedstawienie polityki CKV\_AWS\_158. Brak zapewnienia szyfrowania kluczem KMS  
Źródło: https://github.com/PiotrLotr/project-armadillo

* + - 1. W przypadku reguły CKV\_AWS\_158, konieczne jest zapewnienie enkrypcji logów pochodzących z kontenera. Sugerowane jest użycie KMS (Key Management Service). Zapewnia on bezpieczne przechowywanie, generowanie, rotację i używanie kluczy kryptograficznych do szyfrowania danych w różnych usługach AWS i aplikacjach. Dzięki szyfryzacji, możemy atomizować dostęp do logów, tylko dla ról posiadających uprawnienia do klucza i jego użycia.

Figure 33: Przedstawienie polityki CKV2\_AWS\_28. Zastosowanie AWS WAF  
 Źródło: https://github.com/PiotrLotr/project-armadillo

* + - 1. Opisując bezpieczeństwo infrastruktury AWS, należy wsponieć o AWS WAF (Web Application Firewall). Służy do ochrony aplikacji internetowych przed atakami sieciowymi i złośliwym ruchem. AWS WAF działa jako warstwa ochronna między aplikacją internetową a klientami, analizując ruch sieciowy i blokując niepożądane lub niebezpieczne żądania.
      2. Główne cechy i funkcje AWS WAF obejmują:

1. Ochrona przed atakami typu OWASP Top 10: AWS WAF zapewnia zabezpieczenia przeciwko popularnym atakom, takim jak wstrzykiwanie SQL, przepełnienie bufora, ataki XSS (Cross-Site Scripting) i wiele innych. Działa na podstawie zestawu reguł, które można dostosować do specyficznych potrzeb aplikacji.
2. Blokowanie niepożądanego ruchu: AWS WAF umożliwia blokowanie niepożądanego ruchu sieciowego, takiego jak boty, ataki DDoS (Distributed Denial of Service) i inny złośliwy ruch. Można definiować reguły na podstawie adresów IP, podpisów ruchu i innych kryteriów, aby zidentyfikować i zablokować niebezpieczne żądania.
3. Reguły dostosowane do aplikacji: AWS WAF umożliwia dostosowywanie reguł do specyficznych potrzeb aplikacji. Można tworzyć reguły, które odpowiadają na unikalne scenariusze i zabezpieczają aplikację przed atakami specyficznymi dla danej dziedziny.
4. Integrowanie z innymi usługami AWS: AWS WAF jest łatwo zintegrowany z innymi usługami AWS, takimi jak Elastic Load Balancer (ELB), Amazon CloudFront, API Gateway i Application Load Balancer. Pozwala to na ochronę aplikacji w różnych punktach wejścia do systemu.
5. Ochrona przed atakami DDoS: AWS WAF oferuje również zaawansowaną ochronę przed atakami DDoS, zapewniając filtrowanie ruchu sieciowego, wykrywanie i blokowanie ataków DDoS w czasie rzeczywistym.

# Wnioski z przeprowadzonego projektu

## Stopień spełnienia założeń projektowych

# Spis rysunków

[Rysunek 1: Schemat infrastruktury projektu. Podział sieci. Źródło: zbiór autora 11](#__RefHeading___Toc741_3420670368)

[Rysunek 2: Schemat infrastruktury projektu. Wejście do aplikacji z perspektywy użytkownika zewnętrzenego. Źródło: zbiór autora 12](#__RefHeading___Toc741_3420670368)

[Rysunek 3: Schemat procesu dostarczania infrastruktury. Źródło: zbiór autora 13](#__RefHeading___Toc741_3420670368)

[Rysunek 4: Schemat procesu dostarczania infrasktruktury dla środowiska testowego. 13](#__RefHeading___Toc741_3420670368)

[Rysunek 5: Schemat proces dotarczania infrasktruktury dla środowiska produkcyjnego. Źródło: zbiór autora 14](#__RefHeading___Toc8667_1024472429)

[Rysunek 6: Schemat procesu CI dla środowiska testowego. Źródło: zbiór autora 16](#__RefHeading___Toc8667_1024472429)

[Rysunek 7: Schemat procesu CD dla środowiska testowego. Źródło: zbiór autora 17](#__RefHeading___Toc8667_1024472429)

[Rysunek 8: Schemat procesu CI dla środowiska produkcyjnego. Źródło: autorstwo własne 17](#__RefHeading___Toc8667_1024472429)

[Rysunek 9: Schemat procesu CD dla środowiska produkcyjnego. Źródło: zbiór autora 18](#__RefHeading___Toc8667_1024472429)

[Rysunek 10: Schemat podejścia TDS (Test Driven Development) 20](#__RefHeading___Toc8667_1024472429)

[Rysunek 11: Zarządzanie sekretami w GitHub  Żródło: zbiór autora 21](#__RefHeading___Toc743_3420670368)

[Rysunek 12: Implementacja DevSecOps dla procesu CI/CD. Środowisko testowe. Źródło: zbiór autora 22](#__RefHeading___Toc743_3420670368)

[Rysunek 13: Implementacja DevSecOps dla procesu CD. Środowisko testowe. 23](#__RefHeading___Toc743_3420670368)

[Rysunek 14: Implementacja DevSecOps dla procesu CD. Środowisko produkcyjne. 23](#__RefHeading___Toc743_3420670368)

[Rysunek 15: Przegląd miar narzędzia Sonarqube. 24](#__RefHeading___Toc743_3420670368)

[Rysunek 16: Spis poddatności kodu źródłowego 25](#__RefHeading___Toc743_3420670368)

[Rysunek 17: Szczegółowe przedstawienie poddatności Źródło: https://sonarcloud.io 25](#__RefHeading___Toc743_3420670368)

[Rysunek 18: Spis "Security Hotspots" kodu żródłowego. Sekcja: priorytet wysoki. Źródło: https://sonarcloud.io 26](#__RefHeading___Toc743_3420670368)

[Rysunek 19: Spis "Security Hotspots" kodu żródłowego. Sekcja: priorytet średni/niski. Źródło: https://sonarcloud.io 26](#__RefHeading___Toc2679_3420670368)

[Rysunek 20: Progi bramki jakościowej kodu żródłowego Źródło: https://sonarcloud.io 27](#__RefHeading___Toc2679_3420670368)

[Rysunek 21: Raport poddatności narzędzia Trivy Źródło: https://github.com/PiotrLotr/project-armadillo 29](#__RefHeading___Toc2681_3420670368)

[Rysunek 22: Przedstawienie polityki CVE-2022-2900. Server-Side Request Forgery (SSRF) Źródło: https://github.com/PiotrLotr/project-armadillo 30](#__RefHeading___Toc2681_3420670368)

[Rysunek 23: Przedstawienie polityki CVE-2022-2900. Server-Side Request Forgery (SSRF) Źródło: https://avd.aquasec.com/nvd/2022/cve-2022-2900/ 30](#__RefHeading___Toc4350_3420670368)

[Rysunek 24: Przedstawienie polityki CVE-2021-44906. Podatność „Prototype pollution” Źródło: https://github.com/PiotrLotr 31](#__RefHeading___Toc4350_3420670368)

[Rysunek 25: Przedstawienie polityki CVE-2021-44906. Podatność „Prototype pollution” Źródło: https://avd.aquasec.com/nvd/2021/cve-2021-44906/ 32](#__RefHeading___Toc4350_3420670368)

[Rysunek 26: Przedstawienie polityki CVE-2019-14697. Podatność „Out-of-bounds Write” Źródło: https://github.com/PiotrLotr 33](#__RefHeading___Toc4350_3420670368)

[Rysunek 27: Przedstawienie polityki CVE-2019-14697. Podatność „Out-of-bounds Write” Źródło: https://avd.aquasec.com/nvd/2019/cve-2019-14697/ 34](#__RefHeading___Toc4350_3420670368)

[Rysunek 28: Implementacja DevSecOps do procesu dostarcza infrastruktury. Środowisko testowe i produkcyjne. Źródło: zbiór autora 35](#__RefHeading___Toc4350_3420670368)

[Rysunek 29: Stosunek wyników pozytywnych/negatywnych w raporcie Checkov Źródło: https://github.com/PiotrLotr/project-armadillo 36](#benefits)

[Rysunek 30: Przedstawienie polityki CKV\_AWS\_91. Źródło: https://github.com/PiotrLotr/project-armadillo 36](#benefits)

[Rysunek 31: Przedstawienie polityki CKV\_AWS\_2. Brak zapewnienia protokułu HTTPS 37](#benefits)

[Rysunek 32: Przedstawienie polityki CKV\_AWS\_158. Brak zapewnienia szyfrowania kluczem KMS Źródło: https://github.com/PiotrLotr/project-armadillo 37](#benefits)

[Figure 33: Przedstawienie polityki CKV2\_AWS\_28. Zastosowanie AWS WAF  Źródło: https://github.com/PiotrLotr/project-armadillo 38](#benefits)

# Bibliography

1. https://www.devsecops.org [↑](#endnote-ref-2)
2. https://owasp.org/www-project-top-ten [↑](#endnote-ref-3)
3. https://cwe.mitre.org/data/definitions/521.html [↑](#endnote-ref-4)