



POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

**Katedra Inżynierii Oprogramowania
Zarządzanie projektami (IM)**

Inż. Piotr Wrona

Nr albumu s25773

Procedura bezpieczeństwa dla ciągłej integracji oprogramowania oparta na DevSecOps

Rodzaj pracy

Praca magisterska

Imię i nazwisko promotora

Dr. Hab. Inż. Piotr Habela

Warszawa, Lipiec, 2023

Streszczenie:

Praca dyplomowa jest wprowadzaniem do zagadnienia DevSecOps. Dyplomant we wstępie przybliży pojęcie i wyjaśni podstawy zagadnienia, niezbędne do prawidłowego rozumienia dalszej treści. W ramach wprowadzenia, przybliżony również zostaje cel i założenia pracy.

Dyplomant dla przeprowadzenia dalszej implementacji elementów DevSecOps, opracował projekt aplikacji NodeJS, wraz z infrastrukturą oraz procesem CI/CD. Każdy wybór dotyczący rozwiązań technologicznych został uzasadniony, a narzędzie/metodyka opisane. Proces CI/CD został zaplanowany, podzielony na części logiczne i zaprezentowany w postaci schematów w notacji BPMN.

W podrozdziale 2.2.1, zatytułowanym „Planowanie bezpieczeństwa”, następuje wybór metody implementacyjnej oraz przytoczone zostają miary określające bezpieczeństwo procesu. Po etapie planowania, dyplomant zaprojektował i opisał proces CI/CD z implementacją DevSecOps. Autor zaprezentował również poszczególne narzędzia wykorzystane w celu wprowadzania bezpieczeństwa oraz przedstawił wyniki ich zastosowania.

Finalnie, dyplomant zweryfikował stopień wykonania celów i założeń pracy oraz przedstawił wnioski dotyczące tematu pracy.

Słowa Kluczowe:

DevSecOps, cyberbezpieczeństwo, proces CI/CD, AWS, Cloud

Spis treści:

1. Wstęp.....	4
1.1. Cel pracy.....	4
1.2. Założenia.....	4
1.3. Praktyki DevSecOps.....	5
1.3.1.1. Geneza.....	5
1.3.1.2. Cele DevSecOps.....	5
1.3.1.3. Manifest DevSecOps.....	6
1.3.1.4. Podejścia do DevSecOps.....	8
2. Rozwiązanie projektowe.....	9
2.1. Rozwiązanie bazowe.....	9
2.1.1. Wybór struktury aplikacji.....	9
2.1.2. Wybór środowiska aplikacji.....	10
2.1.3. Wybór bazy danych.....	10
2.1.4. Wybór modelu i architektury dla projektu.....	11
2.1.4.1. Modele infrastrukturalne.....	11
2.1.4.2. Propozycja infrastruktury.....	11
2.1.5. Proces ciągłego dostarczania infrastruktury.....	14
2.1.6. Wejściowy proces CI/CD dla projektu.....	16
2.1.6.1. Cel projektowania i stosowania CI/CD.....	16
2.1.6.2. Propozycja procesu CI/CD dla projektu.....	17
2.2. Rozwiązanie z implementacją DevSecOps.....	21
2.2.1. Planowanie bezpieczeństwa.....	21
2.2.1.1. Miary bezpieczeństwa procesu CI/CD.....	21
2.2.1.2. Planowanie implementacji do procesu CI/D.....	22
2.2.1.3. Zarządzanie sekretami.....	23
2.2.2. Implementacja DevSecOps w procesie CI/CD.....	24
2.2.3. Zastosowanie narzędzi wspomagających DevSecOps.....	26
2.2.3.1. Zastosowanie narzędzia Sonarqube.....	26
2.2.3.2. Kontrola jakości kodu źródłowego.....	29
2.2.3.3. Zastosowanie narzędzia Trivy.....	30
2.2.4. Implementacja DevSecOps w infrastrukturze.....	36
2.2.4.1. Implementacja DevSecOps w procesie dostarczania infrastruktury.....	36
2.2.4.2. Zastosowanie narzędzia Checkov.....	37
3. Wnioski z przeprowadzonego projektu.....	40
3.1. Stopień spełnienia celów pracy.....	40
3.2. Stopień spełnienia założeń projektowych.....	40
3.3. Wnioski.....	41
3.3.1. DevSecOps nawykiem zamiast jednorazową implementacją.....	41
3.3.2. Wizualizacja procesów ponad pisemne raporty.....	41
3.3.3. Pełna automatyzacja nie zawsze jest optymalna.....	41
Bibliografia.....	42
Spis rysunków.....	43

1. Wstęp

1.1. Cel pracy

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.

Opracowanie ma pozwolić na jak najgłębsze przedstawienie zagadnienia oraz porządkując 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.

1.2. Założenia

Spośród założeń projektowych pracy można wyróżnić:

- Zastosowanie aplikacji JavaScript, zintegrowaną z bazą danych – MongoDB. Stworzona platforma pełni funkcję streamingu video.
- Zastosowanie technologii konteneryzacji.
- Zastosowanie środowiska GitHub do tworzenia automatyzacji oraz jako repozytorium kodu źródłowego.
- Stworzenie procesu w pełni automatycznego, niewymagającego czynnika ludzkiego.
- Kompletne odzwierciedlenie procesu w postaci kodu źródłowego.
- Repozytorium przytoczone w źródle tworzy integralną część projektu.
- Opisane podatności nie dotyczą wszystkich wrażliwości występujących w projekcie. Pełne raporty można znaleźć w źródłach.

1.3. Praktyki DevSecOps

1.3.1.1. *Geneza*

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.

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.

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.

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.

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.

1.3.1.2. *Cele DevSecOps*

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 luki 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.

1.3.1.3. Manifest DevSecOps

W celu poprawnego zrozumienia badanego zagadnienia konieczne jest przybliżenie genezy terminu DevSecOps. Został on podobnie jak koncept podejścia Agile zebrany w zbiór reguł, które tworzą manifest **[WEB01]**. 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

Ropatrzymy poszczególne tezy stawiane przez manifest:

„Leaning in over Always Saying “No””

Przez to hasło należy rozumieć transformację podejścia, polegającą na przejście z mówienia „nie”, do świadomego wkładu w rozwijanie kwestii bezpieczeństwa. Temat bezpieczeństwa nie może być odkładany na później, a kwestie związane z inicjatywami w tej materii muszą być wspierane.

„Data & Security Science over Fear, Uncertainty and Doubt”

DevSecOps musi być oparty na profesjonalnym podejściu, wspomaganym przez rzetelne raporty wykorzystywanych narzędzi, zamiast na podejściu chaotycznym, kierowanym przez strach.

„Open Contribution & Collaboration over Security-Only Requirements”

Tworzenie oprogramowania opiera się na integracji wielu produktów zewnętrznych oraz wewnętrznych. Konieczna jest świadomość wszystkich jej elementów oraz rozumienie ich integracji, tak aby stworzyć bezpieczny system.

„Consumable Security Services with APIs over Mandated Security Controls & Paperwork”

Priorytet to stworzenie procesu, który możemy wyrażać jako pewnego rodzaju usługę zapewniania bezpieczeństwa zamiast skupiać się na narzędziach służących wykonywaniu danej funkcjonalności.

„Business Driven Security Scores over Rubber Stamp Security”

Konieczne jest dostosowanie naszych wymogów dotyczących bezpieczeństwa do wymagań biznesu. Każda implementacja powinna być oceniana pod względem jej użyteczności, zamiast ślepego wykonywania założonych zadań.

Red & Blue Team Exploit Testing over Relying on Scans & Theoretical Vulnerabilities

Wykorzystanie techniki zespołów czerwony/niebiescy do nauki obrony systemu. W założeniu podejście to odwzorowuje bardziej realistyczny przebieg potencjalnego ataku przy użyciu najbardziej aktualnych narzędzi penetracyjnych.

24x7 Proactive Security Monitoring over Reacting after being Informed of an Incident

Ciągłe skanowanie naszego systemu pozwala nam wykrywać zdarzenia potencjalnie niebezpieczne i przeciwdziałać im przed szkodą.

Shared Threat Intelligence over Keeping Info to Ourselves

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.

Compliance Operations over Clipboards & Checklists

Ciągła kontrola ponad ufność w checklisty. Nawet najbardziej niezawodny proces automatyczny nie jest w stanie przewidzieć każdego problemu, a człowiek jest podatny na błąd ludzki.

1.3.1.4. *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.

2. Rozwiązanie projektowe

2.1. Rozwiązanie bazowe

2.1.1. Wybór struktury aplikacji

W ostatnich latach, trend rozwijania aplikacji sukcesywnie przesuwa się z dużych monolitycznych aplikacji na aplikacje mikro serwisowe. Tabela poniżej opisuje główne różnice obu podejść **[DEB01]**.

Podjęcie monolityczne	Podjęcie mikroserwisowe
<ul style="list-style-type: none">• Potencjalna awaria powoduje duże problemy funkcjonalne i może sparaliżować cały system• Zmiany w aplikacji są trudne, ze względu na złożoność kodu źródłowego• Środowiska uruchomieniowe często wymagają dużych zasobów i są uruchamiane na dużych maszynach serwerowych	<ul style="list-style-type: none">• Potencjalna awaria powoduje utratę pojedynczej funkcjonalności, ale nie powoduje 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

Rozpatrując przedstawione porównanie, łatwo można dostrzec przeważające zalety podejścia mikro serwisowego. Z tego powodu, projekt przewiduje zastosowanie konteneryzacji zarówno dla serwera www oraz serwera baz danych.

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.

2.1.2. Wybór środowiska aplikacji

Projekt przewiduje użycie node.js jako środowiska uruchomieniowego. Wyróżnia się ono następującymi cechami:

- wieloplatformowość
- bardzo duża szybkość obsługi zapytań
- łatwość tworzenia zaawansowanych programów
- duża liczba framework'ów przyspieszających pisanie aplikacji

Node.js wykorzystuje składnię języka JavaScript, który cechuje się elastycznością (możliwość zastosowania programowania funkcyjnego czy obiektowego) oraz dużą czytelnością (jest to język wysokopoziomowy).

2.1.3. Wybór bazy danych

Aplikacja wykorzystuje bazę danych do przechowywania informacji o video, udostępnianym przez serwer www. Dla przytoczonego zastosowania, nie jest wymagana złożona relacyjność, a optymalne jest zestawienie kluczy wartości. Wybór bazy padł na nierelacyjną bazę mongodb.

Baza cechuje 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 zaprojektowaną z myślą o łatwej replikacji

2.1.4. Wybór modelu i architektury dla projektu

2.1.4.1. Modele infrastrukturalne

W zależności od lokalizacji serwerów i zakresu obowiązków dotyczących ich obsługi, wyróżniamy **[WEB02]**:

- 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 dostawcę 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.

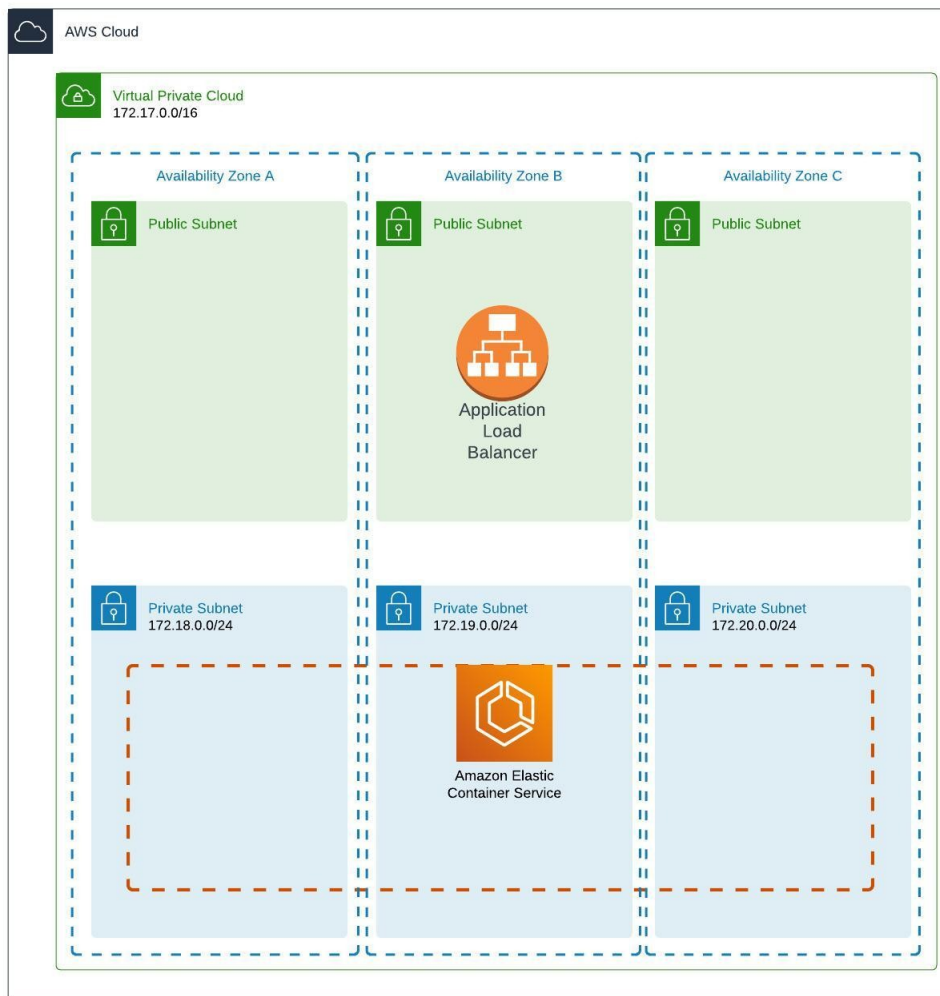
W praktyce zdarza się, że rolę się przeplatają a obok modelu IaaS stosujemy rozwiązania SaaS. Jednym z częstych przypadków jest stosowanie usług Elasticsearch do monitorowania naszej infrastruktury IaaS.

2.1.4.2. Propozycja infrastruktury

Na potrzeby rozwiązania projektowego, użyta została chmura AWS w modelu IaaS. Występują też pewne elementy zbliżone do modelu PaaS, takie jak zastowanie AWS ECS, w podejściu fargate.

Głównym logicznym podziałem naszej infrastruktury jest VPC (Virtual Private Cloud), który definiuje naszą sieć i zakres adresów. Architektura jest 2-tierowa i wyróżnia 3 podsieci publiczne oraz 3 podsieci prywatne. Sieć zlokalizowana jest w regionie eu-central-1 (Frankfurt).

Ograniczenie do jednej lokalizacji, sprawia, iż system jest podatny na awarię w obrębie regionu. Natomiast, pewną redundancję uzyskujemy poprzez wydzielenie 3 AZ (Availability zone): 1A, 1B, 1C.



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

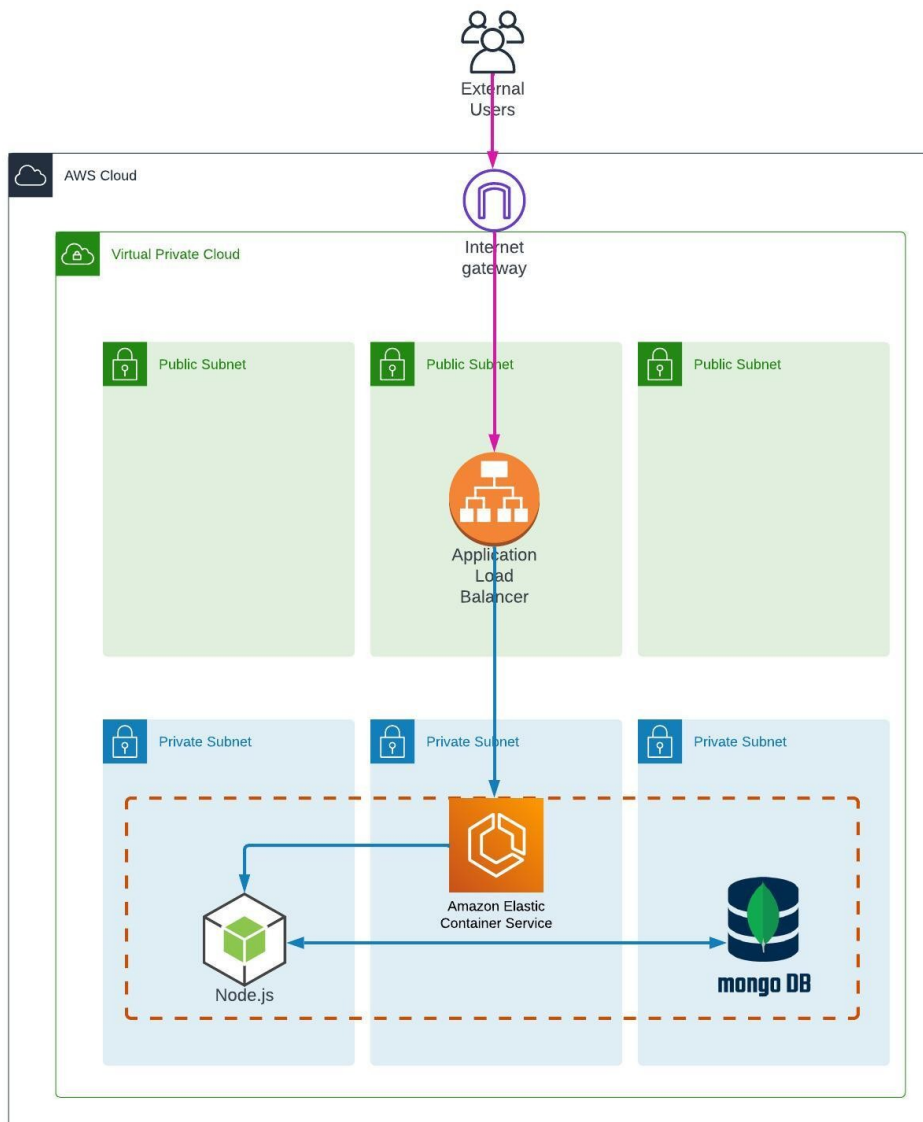
Każda podsieć publiczna ma maskę 255.255.255.0 która obejmują 254 adresy. Dla naszego rozwiązania projektowe zakres możliwych destynacji jest nieproporcjonalnie duży, natomiast umożliwia to ewentualne dalsze rozwijanie infrastruktury czy możliwości skalowania.

Wejściem do naszego systemu jest IG (Internet Gateway) który przyjmują cały ruch publiczny i kieruje go na ALB (Application Load Balancer). Odpowiada on za dystrybucję ruchu, terminując certyfikat SSL oraz odpowiada za częściową obsługę błędów (np.: poprzez odpowiedź na błąd 404).

Model wyróżnia się wysokim poziomie dostępności, poprzez zastosowanie 3 stref dostępowych. Jak już uprzednio wspomiano, dzięki takiemu działaniu osiągamy redundantność, a awaria jednego obszaru, nie wpływa na paraliż systemu.

Kontenery aplikacji znajdują się w podsieciach prywatnych i są zarządzane przez serwis ECS (Elastic Container Service) w modelu Fargate. Podejście to cechuje:

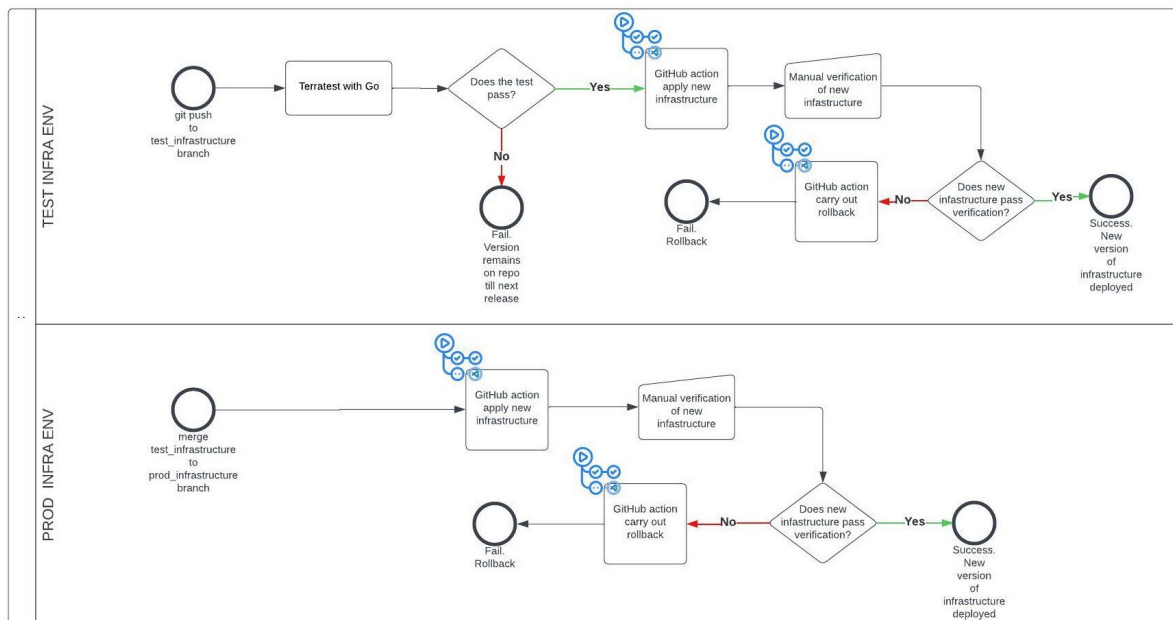
- brak konieczności administracji serwerami
- mikro serwisowość
- zastępowalność
- łatwość skalowania
- skuteczniejsze wykorzystanie zasobów
- skalowalność horyzontalna



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

2.1.5. Proces ciągłego dostarczania infrastruktury

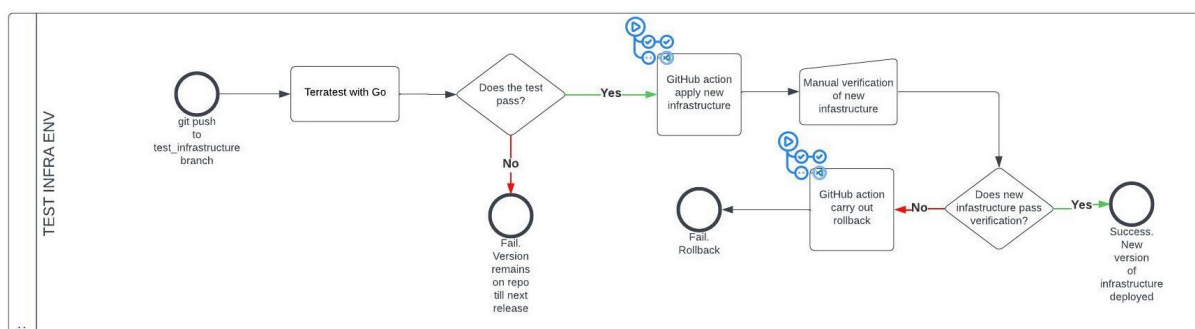
W procesie dostarczania infrastruktury wyróżniamy środowiska: testowe oraz produkcyjne. Dla każdego zostały przydzielone gałęzie, nazwane odpowiednio: „test_infrastructure” oraz „prod_infrastructure”.



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

Aby zainicjować cykl testowy, konieczne jest wypuszczenie kodu do odpowiedniej gałęzi. Rozpoczyna to tzw.: „Terratest”, który jest framework’iem open-source i jest napisany w języku Go. Umożliwia on automatyzację testowania infrastruktury w chmurze.

Terratest umożliwia testowanie różnych aspektów infrastruktury w chmurze, w tym konfiguracji serwerów, dostępności usług, skalowalności i bezpieczeństwa. Dzięki temu, uzyskujemy pewność, że wdrożenie działa zgodnie z oczekiwaniami.

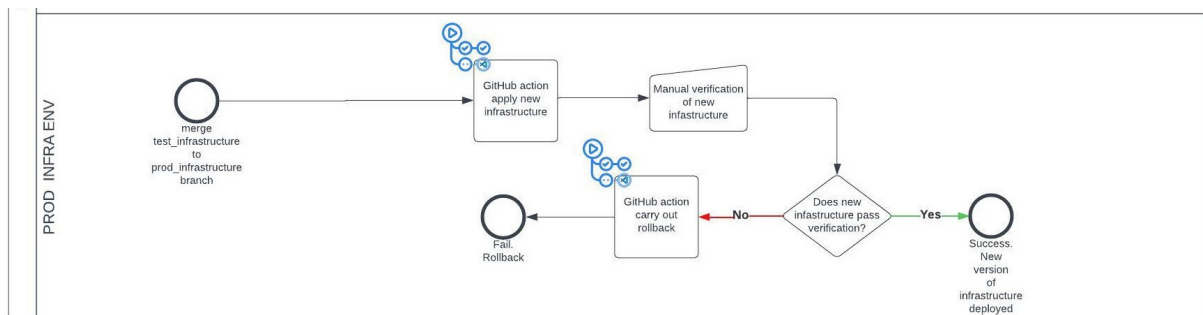


Rysunek 4: Schemat procesu dostarczania infrasktruktury dla środowiska testowego.

Źródło: zbiór autora

W przypadku niepowodzenia, kod zostaje odrzucony i dalsze akcje nie są wyzwalane. W sytuacji odwrotnej, rozpoczyna się wykonanie akcji GitHub która aplikuje zmiany na środowisko testowe. Kolejnym krokiem jest manualna weryfikacja środowiska i podjęcie decyzji co do stabilności środowiska. W przypadku akceptacji – transakcja jest zakończona sukcesem, w przypadku jej braku- następuje rollback do poprzedniej wersji infrastruktury.

Proces dostarczania aplikacji na środowisko produkcyjne jest poprzedzone pewnym ustalonym czasem testowania na niższym środowisku. Inicjacja procesu następuje przez połączenie zmian środowiska testowego z środowiskiem produkcyjnym.



Rysunek 5: Schemat proces dotarczenia infraskuktury dla środowiska produkcyjnego.

Źródło: zbiór autora

Połączenie gałęzi rozpoczyna akcję GitHub wprowadzającą zmiany na środowisko produkcyjne. Cykl w tym biegu, nie uwzględnia już Terratest'u – przyjęto założenie stabilności regionu testowego. Zachowana natomiast jest manualna weryfikacja inżynierska, kończona przez decyzję dotyczącą stabilności systemu. W przypadku akceptacji – transakcja jest zakończona sukcesem, w przypadku jej braku- następuje rollback do poprzedniej wersji infrastruktury.

2.1.6. Wejściowy proces CI/CD dla projektu

2.1.6.1. *Cel projektowania i stosowania CI/CD*

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.

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

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 inicjuje proces ciągłego wdrażania, czyli CD.

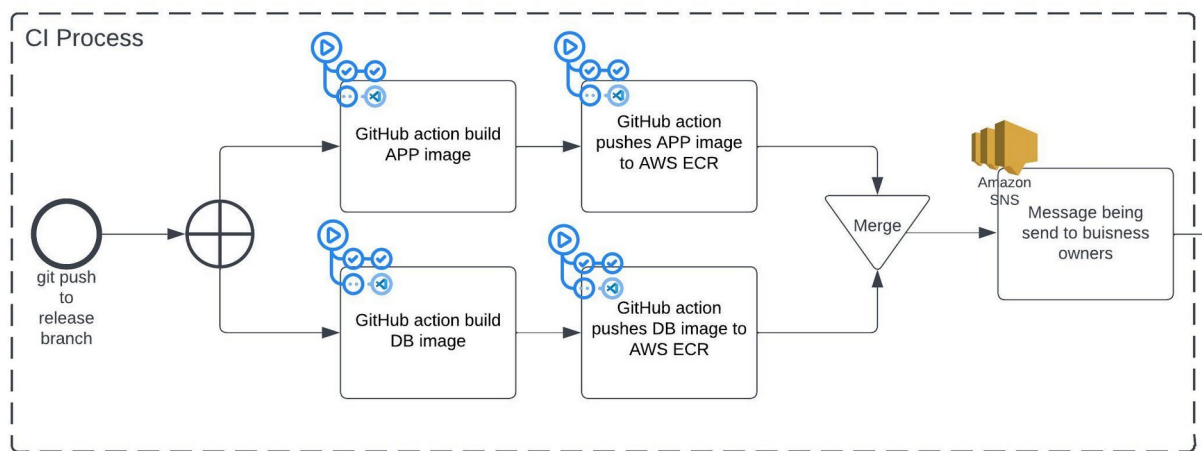
Podejście do schematu dostarczania oprogramowania jest bardzo różne i zależy głównie od:

- Technologii wdrażanej aplikacji
- Poziomu unifikacji systemów i serwerów
- Strategii biznesowej firmy
- Ilości środowisk testowych i produkcyjnych
- Doświadczenia i testowania rozwiązań

2.1.6.2. Propozycja procesu CI/CD dla projektu

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 przewidują 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.

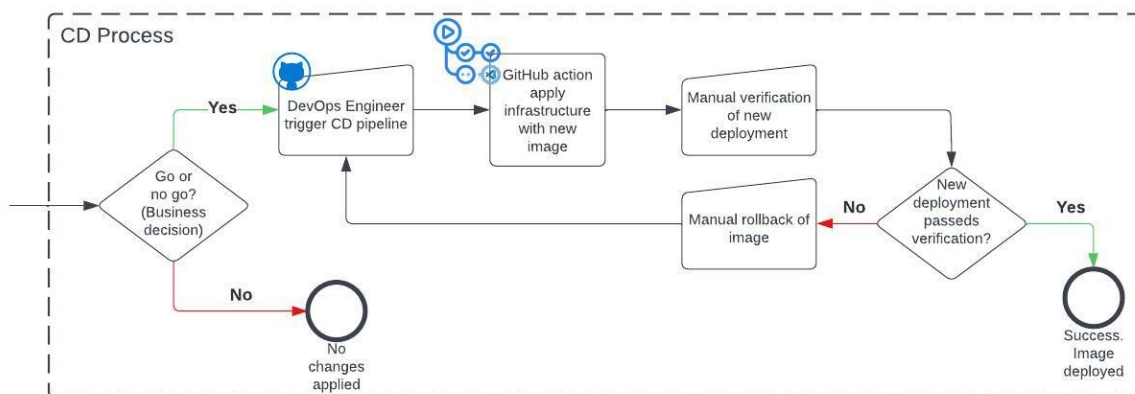
Podejście konteneryzacyjne do aplikacji, centralizują proces wokół repozytorium obrazów dockerowych. Wpływa to zarówno część integracyjną - obok udostępnia kodu, następuję udostępnienie obrazu, jak również część wdrożeniową – poprzez pobranie najnowszej wersji obrazu i zaaplikowanie na środowisku.



Rysunek 6: Schemat procesu CI dla środowiska testowego.

Źródło: zbiór autora

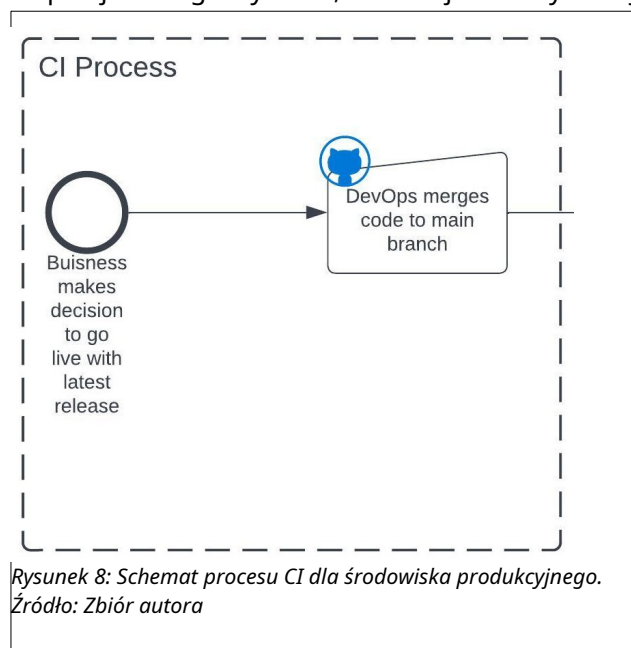
1. Proces rozpoczyna się od wypuszczenia kodu na tzw.: „release branch”. Zaletami tworzenia dedykowanych gałęzi git dla danych sprintów są:
 - Lepsza kontrola nad zmianami i możliwość prowadzenia dziennika zmian
 - Możliwość łatwego rollback’u do stabilnej wersji
2. Zmiana w repozytorium inicjuje równoległe przetwarzania dwóch pipeline’ów – jeden do tworzenia obrazu aplikacji oraz drugi do tworzenia kontenera bazy danych.
3. Każdy z nowo utworzonych obrazów zostaje 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 występuje możliwość rezygnacji z nowego wydania. Taka akcja nie wymaga dodatkowych kroków, gdyż rollback następuje 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 finalizacją przez decyzję manualną.
4. W przypadku istotnych problemów i braku akceptacji inżyniera na emisję wydania, następuje rollback obrazu w repozytorium ECR.
5. Podobnie do poprzednich kroków z sekwencji, to inżynier DevOps inicjuje 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.

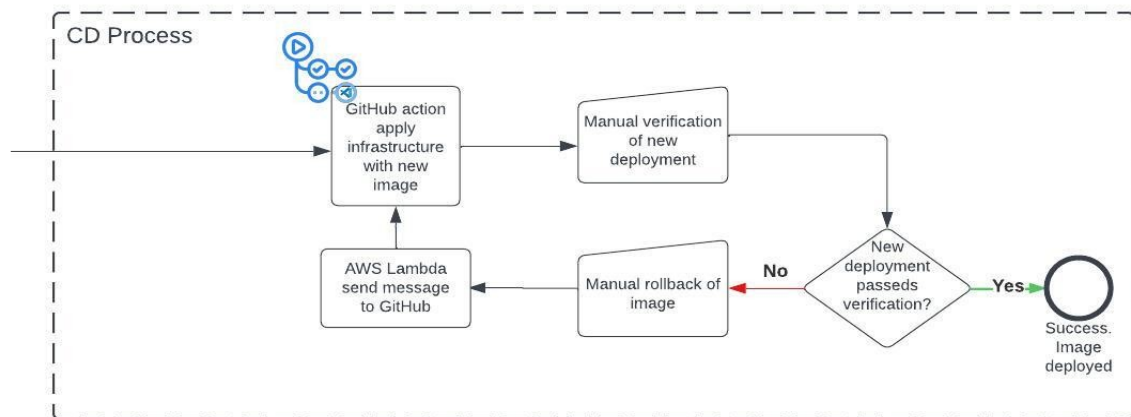


Rysunek 8: Schemat procesu CI dla środowiska produkcyjnego.

Źródło: Zbiór autora

Z perspektywy środowiska produkcyjnego, część integracyjna procesu ma zauważalnie 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 inicjuje 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 finalizacją 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 finalizacją przez decyzję manualną.
6. W przypadku akceptacji nowego wydania, zmiana jest zamykana jako sukces.

2.2. Rozwiązanie z implementacją DevSecOps

2.2.1. Planowanie bezpieczeństwa

2.2.1.1. *Miary bezpieczeństwa procesu CI/CD*

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.

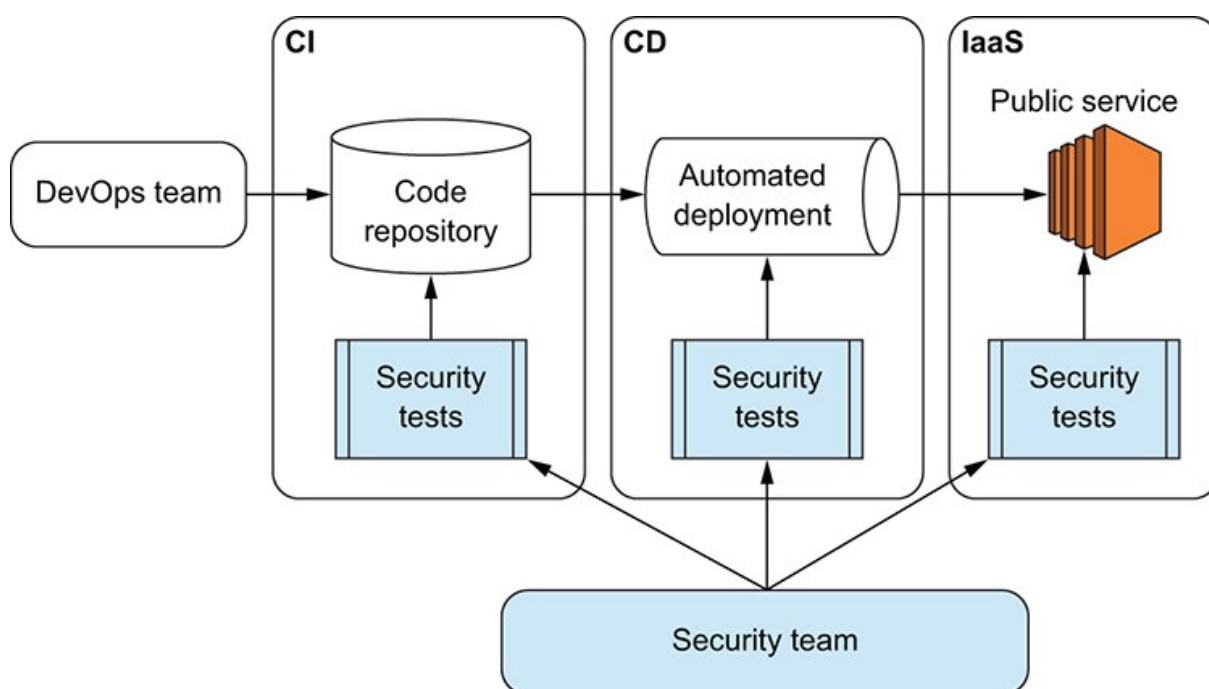
Wszystkie te miary mają na celu zabezpieczenie procesu CI/CD przed zagrożeniami i zapewnienie, że system działa zgodnie z oczekiwaniami.

2.2.1.2. Planowanie implementacji do procesu CI/D

Wybrany 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 [JEV01].

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 podatnościami



Rysunek 10: Schemat podejścia TDS (Test Driven Development)

Źródło: "Securing DevOps" - Julien Vehent

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.

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.

2.2.1.3. Zarządzanie sekretami

Jednym z kluczowych zagadnień każdego projektu IT jest zarządzanie sekretami. Wyciek hasła lub klucza serwerowego jest otwarciem drzwi do podsieci.

Rynek oferuje bardzo szeroki zakres narzędzi do zarządzania sekretami. Można tu wymienić między innymi CyberArk, AWS Secrets Manager czy Passowrd Vault od Hashicorp.

W przypadku projektu zastosowano unifikację narzędzi. Podobnie jak w przypadku użycia GitHub Actions, zastosowane zostało również wbudowane narzędzie GitHub do przechowywania haseł.

The screenshot displays the GitHub 'Actions secrets and variables' interface. On the left, the sidebar is categorized into 'General', 'Access', 'Code and automation', 'Security', and 'Integrations'. Under 'Security', the 'Secrets and variables' section is expanded, showing 'Actions' as the selected option. The main panel, titled 'Actions secrets and variables', provides an overview of secrets and variables. It includes a 'New repository secret' button and tabs for 'Secrets' and 'Variables'. Below these, there are two main sections: 'Environment secrets' (currently empty) and 'Repository secrets'. The 'Repository secrets' section contains a table with five entries, each showing the secret name, its last update date, and icons for editing or deleting the secret.

Repository secrets		
AWS_ACCESS_KEY_ID	Updated on Mar 27	[Edit] [Delete]
AWS_SECRET_ACCESS_KEY	Updated on Mar 27	[Edit] [Delete]
MONGO_INITDB_ROOT_PASSWORD	Updated on Apr 11	[Edit] [Delete]
MONGO_INITDB_ROOT_USERNAME	Updated on Apr 11	[Edit] [Delete]
SONAR_TOKEN	Updated on Apr 12	[Edit] [Delete]

Rysunek 11: Zarządzanie sekretami w GitHub
Źródło: <https://github.com/PiotrLotr/project-armadillo>

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.

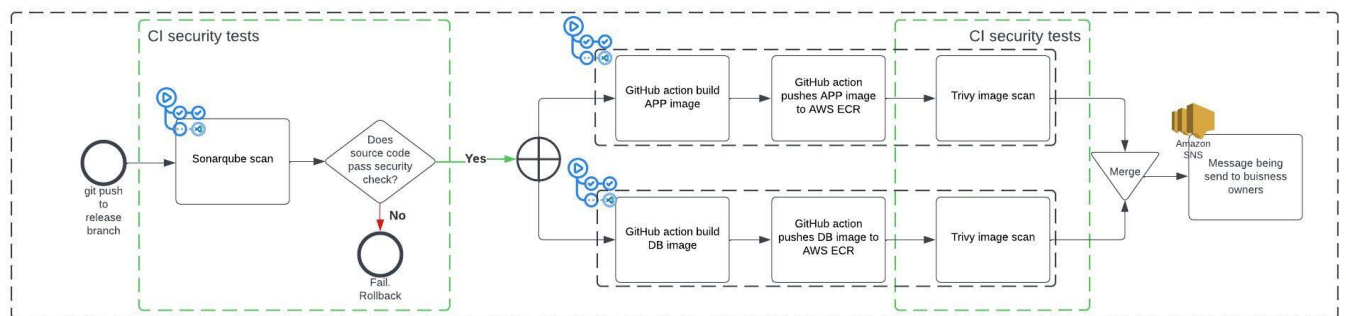
Pomimo, że hasła znajdują się repozytorium publicznym, są one niedostępne dla nieupoważnionych osób. Kontrola dostępu odbywa się poprzez dodawanie/usuwanie użytkowników projektowych oraz poprzez ich role.

2.2.2. Implementacja DevSecOps w procesie CI/CD

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

Wśród głównych 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

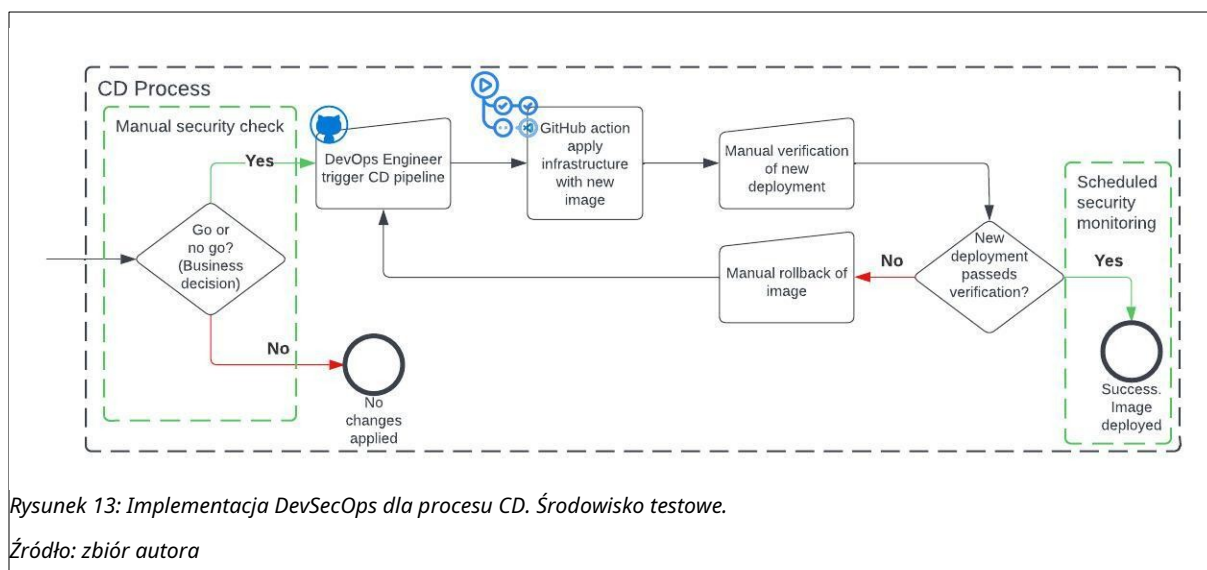


Rysunek 12: Implementacja DevSecOps dla procesu CI/CD. Środowisko testowe.

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 następuje 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 pozostaje w repozytorium. Jest ona nadpisywana przez następne wydanie kodu.
4. Przy pozytywnej ewaluacji, następuje budowa obrazów dockerowych. Budowa obrazów jest uzupełniona o tworzenie raportu podatności obrazów kontenera przy użyciu narzędzia Trivy.
5. Raport jest wysyłany do repozytorium GitHub, natomiast obraz niezależnie 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 biznesowego.

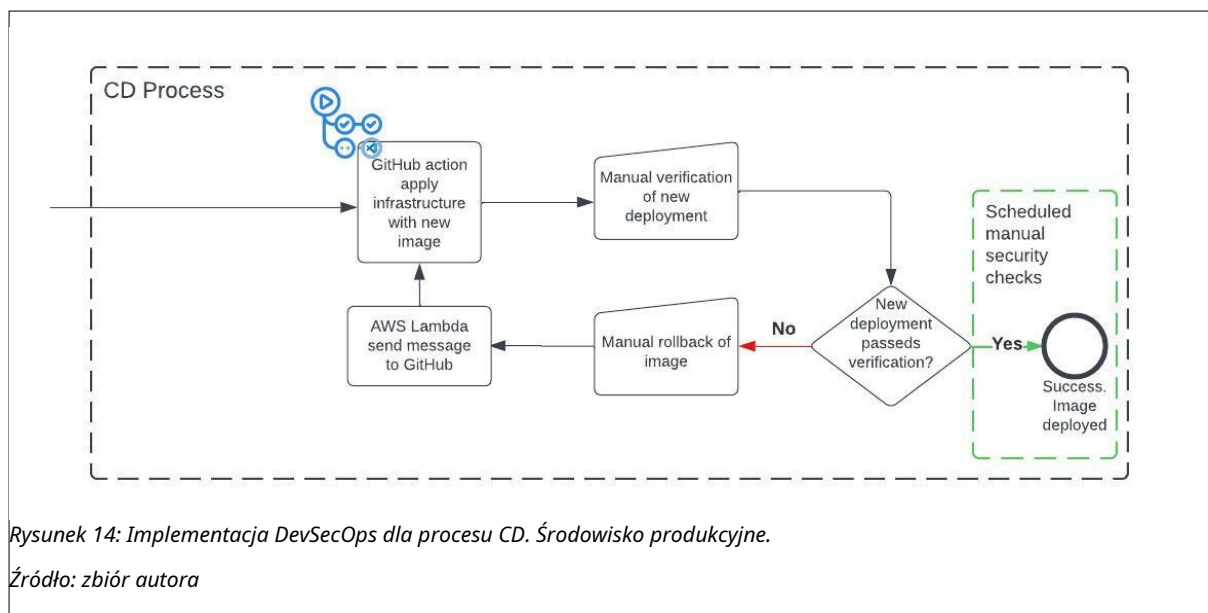
W przypadku procesu ciągłego dostarczania, zmiany są mniej znaczące i ograniczają się do pewnych weryfikacji manualnych. W przypadku procesu integracji kodu z uwzględnieniem bezpieczeństwa, istnieją dwa dodatkowe zawory bezpieczeństwa. Pomimo faktu istnienia tych zabezpieczeń, zakładamy rozpoczęcie procesu 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 rozpoczęcia 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

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 istotną funkcję polegającą na dzieleniu wiedzą i wzroście każdego członka zespołu.



Sytuacja wygląda inaczej 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 część stałej, zaplanowej weryfikacji manualnej, wykonywany nawet w przypadku wypuszczenia wersji na środowisko produkcyjne.

2.2.3. Zastosowanie narzędzi wspomagających DevSecOps

2.2.3.1. Zastosowanie narzędzia Sonarqube

Pomimo dążenia do mikroservisów, kod źródłowy dzisiejszych aplikacji to nadal wiele wierszy obfitujących w funkcję zaczerpniętą z różnorodnych bibliotek. Okazują 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. Cechują się szczególną prostotą integracji z procesem CI/CD, funkcjonalną i przejrzystą wirtualizacją danych oraz wsparciem bardzo szerokiej gamy języków programowania.

Aby móc mierzyć proces usprawniania kodu źródłowego, konieczne jest wybranie metryk, które ów proces będą opisywać **[SON01]**. W przypadku Sonarqube są to:

- **Bugs** – błędy które mogą spowodować nagłą awarię aplikacji i muszą zostać rozwiązane natychmiastowo.
- **Vulnerabilities** – są to fragmenty kodu podatne 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 podatność.
- **Coverage** – miara określająca stopień pokrycia kodu przez testy.
- **Duplications** – określa ilość zduplikowanych fragmentów kodu.

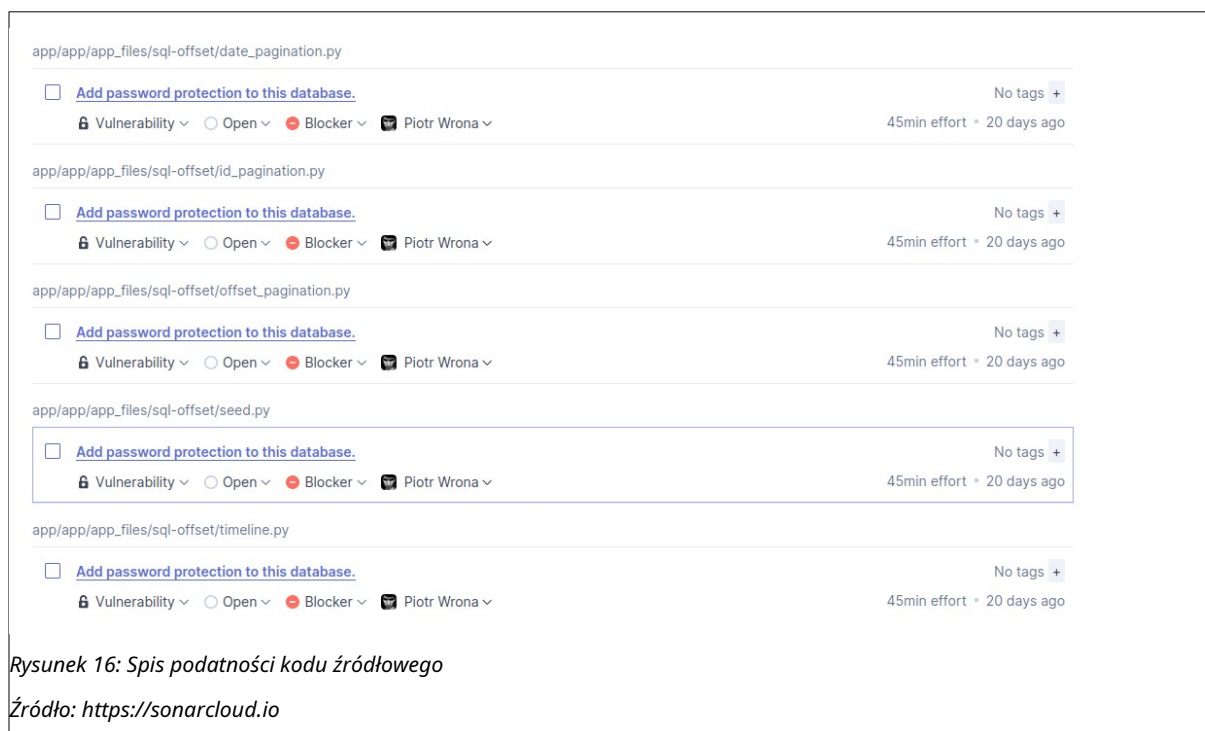
	Lines of Code	Bugs	Vulnerabilities	Code Smells	Security Hotspots	Coverage	Duplications
project-armadillo							
app	276	0	5	5	8	0.0%	0.0%
aws	453	0	0	2	0	—	0.0%

Rysunek 15: Przegląd miar narzędzia Sonarqube.

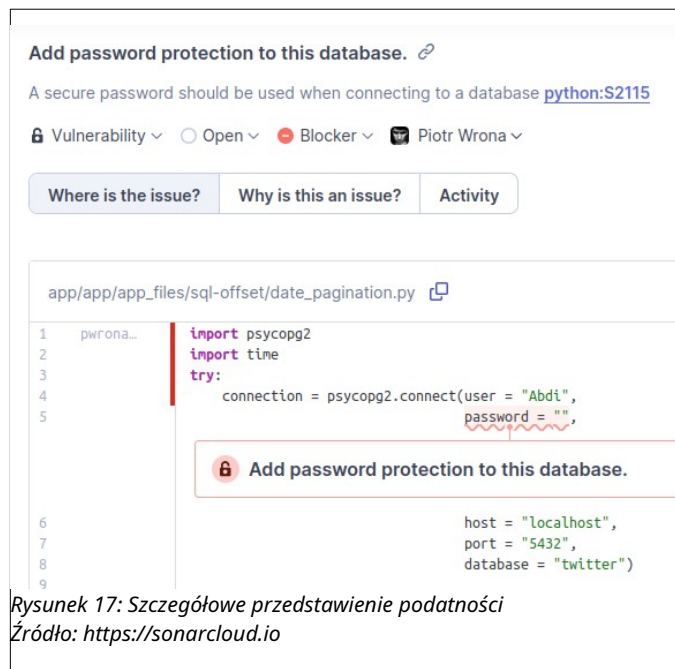
Źródło: <https://sonarcloud.io>

Z punktu widzenia bezpieczeństwa kodu, istotne są głównie „Bugs”, „Vulnerabilities” oraz „Security Hotspots”. W praktyce należy wykorzystać możliwości, które dostarcza nam oprogramowanie i skupić się na raportach dotyczących wszystkich dostarczonych miar.

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



Głębszy raport wskazując na naruszenie protokołów: OWASP A2, A3 oraz CWE-521 **[OWA01]**. Pierwsze dwa kolejną odpowiadają za złamaną autentykację oraz udostępnienie danych wrażliwych. Natomiast ostatni, polega na złamaniu polityki dotyczącej wykorzystywania silnych haseł.



Warto zwrócić uwagę na zakładki: „Where is the issue?“, „Why is this an issue?“ i „Activity“, gdyż są bardzo pomocne dla lokalizacji oraz zrozumienia błędu. Dodatkowo, ostatnia zakładka pozwala na komunikację zespołową i informowaniu o podjętych akcjach dla rozwiązania problemu.

Drugą miarą są „Security Hotspots”, czyli fragmenty potencjalnie niebezpieczne i wymagające manualnej recenzji. Części te są podzielone w zależności od ich priorytetu. W poniższym przypadku najwyższą wagę ma zabezpieczenie hasła do bazy danych. Rozwiązanie problemu zostało przedstawione w sekcji: „Zarządzanie sekretami”.

0.0% Security Hotspots Reviewed ?

To review Fixed Safe

8 Security Hotspots to review

Review priority: High

Authentication 4

Make sure that using ARG to handle a secret is safe here.

Make sure that using ENV to handle a secret is safe here.

Make sure that using ARG to handle a secret is safe here.

Make sure that using ENV to handle a secret is safe here.

Review priority: Medium

Permission 2

Review priority: Low

Others 2

Make sure that using ARG to handle a secret is safe here.

Using ENV or ARG to handle secrets is security-sensitive [docker:S6472](#)

Status: To Review

This Security Hotspot needs to be reviewed to assess whether the code poses a risk.

Where is the risk? What's the risk? Assess the risk How to fix it? Activity

app/app/Dockerfile

```
1 FROM node:12.7.0-alpine
2
3 ARG MONGO_INITDB_ROOT_USERNAME
4 ARG MONGO_INITDB_ROOT_PASSWORD
5
6 ARG MONGO_INITDB_DATABASE
7
8 ENV MONGO_INITDB_ROOT_USERNAME=${MONGO_INITDB_ROOT_USERNAME}
9 ENV MONGO_INITDB_ROOT_PASSWORD=${MONGO_INITDB_ROOT_PASSWORD}
10 ENV MONGO_INITDB_DATABASE=${MONGO_INITDB_DATABASE}
```

Show 12 more lines

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

Raport wskazuje również na użycie konta root jako domyślnego użytkownika dla kontenerów, natomiast 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.

Permission 2

The node image runs with root as the default user. Make sure it is safe here.

The mongo image runs with root as the default user. Make sure it is safe here.

Review priority: Low

Others 2

Make sure disclosing the fingerprinting of this web technology is safe here.

Omitting --ignore-scripts can lead to the execution of shell scripts. Make sure it is safe here.

The node image runs with root as the default user. Make sure it is safe here.

app/app/Dockerfile

```
1 FROM node:12.7.0-alpine
2
3 ARG MONGO_INITDB_ROOT_USERNAME
4 ARG MONGO_INITDB_ROOT_PASSWORD
5 ARG MONGO_INITDB_DATABASE
6
```

Show 15 more lines

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

2.2.3.2. Kontrola jakości kodu źródłowego

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.

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.

Conditions ?		
Conditions on New Code		
Conditions on New Code apply to all branches and to Pull Requests.		
Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

Rysunek 20: Progi bramki jakościowej kodu źródłowego
Źródło: <https://sonarcloud.io>

2.2.3.3. *Zastosowanie narzędzia Trivy*

W dzisiejszych czasach, wraz z rosnącą popularnością kontenerów, ich 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 **[TRV01]**. 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.

Zastosowania narzędzia Trivy:

- obrazów kontnera
- filesystem
- repozytorium GIT
- obrazy maszyn wirtualnych
- Kubernetes'a
- AWS

Głównym założeniem Trivy jest analiza następujących 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.

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.

Overview

Reporting

Policy

Advisories

Vulnerability alerts

Dependabot

Code scanning 49

Secret scanning

Code scanning

✓ All tools are working as expected

Q is:open branch:release

49 Open

2 Closed

CVE-2022-2900

Critical

Library

#29 opened 2 months ago • Detected by Trivy in app/.../parse-url/package.json:1

CVE-2022-2216

Critical

Library

#28 opened 2 months ago • Detected by Trivy in app/.../parse-url/package.json:1

CVE-2021-44906

Critical

Library

#23 opened 2 months ago • Detected by Trivy in usr/.../minimist/package.json:1

CVE-2021-44906

Critical

Library

#22 opened 2 months ago • Detected by Trivy in usr/.../minimist/package.json:1

CVE-2021-3918

Critical

Library

#20 opened 2 months ago • Detected by Trivy in usr/.../json-schema/package.json:1

CVE-2019-14697

Critical

#13 opened 2 months ago • Detected by Trivy in library/armadillo-app:1

CVE-2019-14697

Critical

#12 opened 2 months ago • Detected by Trivy in library/armadillo-app:1

CVE-2022-3517

High

Library


#50 opened 2 months ago • Detected by Trivy in app/.../minimatch/package.json:1

Rysunek 21: Raport podatności 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) [CVE01]. Dzięki temu CVE pomaga w szybszym i skuteczniejszym reagowaniu na zagrożenia oraz w zapobieganiu ich wykorzystywaniu przez cyberprzestępców.

CVE-2022-2900

 Open in release on Apr 22

app/node_modules/parse-url/package.json:1
Library

Preview unavailable
Sorry, we couldn't find this file in the repository.

Package: parse-url
Installed Version: 1.3.11
Vulnerability CVE-2022-2900
Severity: CRITICAL
Fixed Version: 8.1.0
Link: CVE-2022-2900
Trivy

Tool	Rule ID
Trivy	CVE-2022-2900


Vulnerability CVE-2022-2900

Rysunek 22: Przedstawienie polityki CVE-2022-2900. Server-Side Request Forgery (SSRF)

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

Jednym z pierwszych komunikatów jest naruszenie protokołu CVE-2022-2900 [CVE02]. Dotyczy on naruszenia SSRF (Server Side Request Forgery).

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.

 CVE Vulnerabilities

CVE-2022-2900

Server-Side Request Forgery (SSRF)

Published: Sep 14, 2022 | Modified: Sep 16, 2022

Server-Side Request Forgery (SSRF) in GitHub repository ionicabizau/parse-url prior to 8.1.0.

Weakness

The web server receives a URL or similar request from an upstream component and retrieves the contents of this URL, but it does not sufficiently ensure that the request is being sent to the expected destination.

Affected Software

Name	Vendor	Start Version	End Version
Parse-url	Parse-url_project	*	*

CVSS 3.x

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/

CVSS 2.x

RedHat/V2

RedHat/V3

Ubuntu

9.1
CRITICAL

Rysunek 23: Przedstawienie polityki CVE-2022-2900. Server-Side Request Forgery (SSRF)

Źródło: <https://avd.aquasec.com/nvd/2022/cve-2022-2900/>

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.

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.

Code scanning alerts / #23

CVE-2021-44906

Open in release on Apr 22

usr/local/lib/node_modules/npm/node_modules/rc/node_modules/minimist/package.json:1 Library

Preview unavailable
Sorry, we couldn't find this file in the repository.

Package: minimist
Installed Version: 1.2.0
Vulnerability CVE-2021-44906
Severity: CRITICAL
Fixed Version: 0.2.4, 1.2.6
Link: CVE-2021-44906
Trivy

Tool	Rule ID
Trivy	CVE-2021-44906

Vulnerability CVE-2021-44906

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

Kolejną podatnością jest: CVE-2021-44906 **[CVE03]** która jest oznaczona jako priorytet krytyczny. Alert oznacza wrażliwość typu: „Prototype pollution” co w tłumaczeniu oznacza zanieczyszczenie prototypu.

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.

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.

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.

CVE-2021-44906

Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution')

Published: Mar 17, 2022 | Modified: Apr 12, 2022

Minimist <=1.2.5 is vulnerable to Prototype Pollution via file index.js, function setKey() (lines 69-95).

Weakness

The product receives input from an upstream component that specifies attributes that are to be initialized or updated in an object, but it does not properly control modifications of attributes of the object prototype.

Affected Software

Name	Vendor	Start Version	End Version
Minimist	Substack	*	*

OpenShift Service Mesh RedHat openshift-service-mesh/olm-tha19:1.26.0-1 *

Rysunek 25: Przedstawienie polityki CVE-2021-44906. Podatność „Prototype pollution”

Źródło: <https://avd.aquasec.com/nvd/2021/cve-2021-44906/>

CVSS 3.x

9.8
CRITICAL

Source: NVD

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

CVSS 2.x

7.5 HIGH

RedHat/V2

RedHat/V3

9.8 MODERATE

Ubuntu

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.

Raport wskazują również na złamanie podatności CVE-2019-14697 [CVE04]. Zagrożenie to określone jest jako: „Out-of-bounds Write”.

[Code scanning alerts](#) / #13

CVE-2019-14697

Open in release on Apr 22

library/armadillo-app:1

Preview unavailable

Sorry, we couldn't find this file in the repository.

Package: musl-utils
Installed Version: 1.1.20-r4
Vulnerability CVE-2019-14697
Severity: CRITICAL
Fixed Version: 1.1.20-r5
Link: [CVE-2019-14697](#)
Trivy

Tool
Rule ID

Trivy
CVE-2019-14697

Vulnerability CVE-2019-14697

Rysunek 26: Przedstawienie polityki CVE-2019-14697. Podatność „Out-of-bounds Write”

Źródło: <https://github.com/PiotrLotr>

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.

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.

CVE-2019-14697
Out-of-bounds Write
Published: Aug 06, 2019 | Modified: Mar 03, 2023

musl libc through 1.1.23 has an x87 floating-point stack adjustment imbalance, related to the math/lib386/ directory. In some cases, use of this library could introduce out-of-bounds writes that are not present in an applications source code.

Weakness
The product writes data past the end, or before the beginning, of the intended buffer.

Affected Software

Name	Vendor	Start Version	End Version
Musl	Musl-libc	0.9.12	1.1.23
Musl	Ubuntu	disco	*

CVSS 3.x
9.8 CRITICAL
Source: NVD
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

CVSS 2.x
7.5 HIGH

RedHat/V2
RedHat/V3
Ubuntu

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

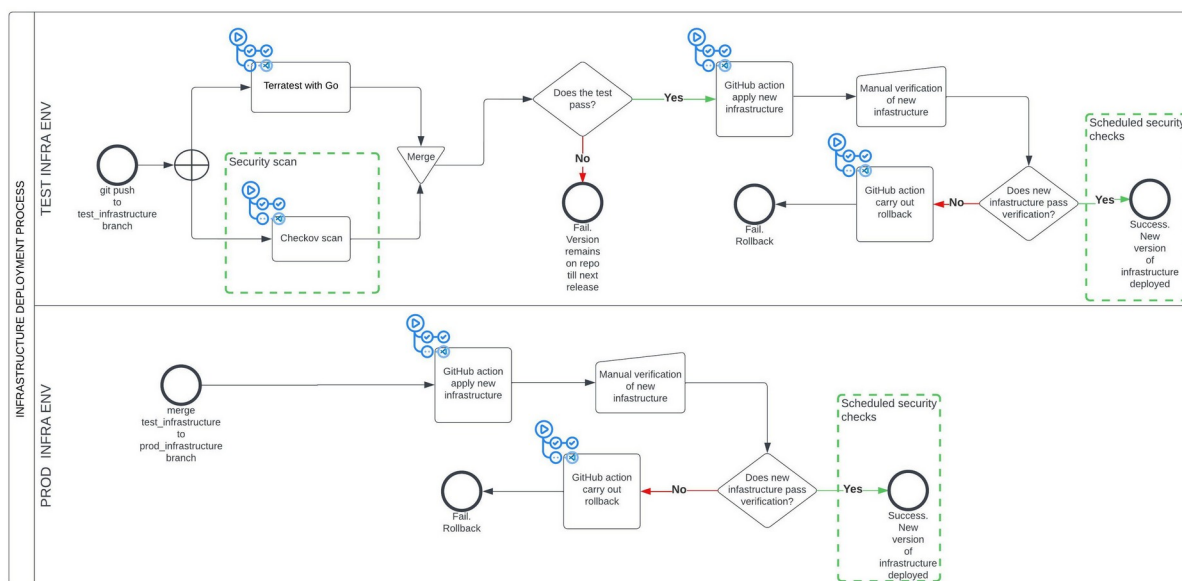
OOBW jest często wykorzystywany przez atakujących do wykonania tzw. ataków typu buffer overflow, w których złośliwy kod może nadpisać ważne dane lub wstrzyknąć kod do pamięci programu i tym samym uzyskać kontrolę nad systemem.

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.

2.2.4. Implementacja DevSecOps w infrastrukturze

2.2.4.1. Implementacja DevSecOps w procesie dostarczania infrastruktury

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

Kontynuacja procesu dostarczania 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 pozostają w repozytorium do jego kolejnego nadpisania, natomiast procedura się kończy.

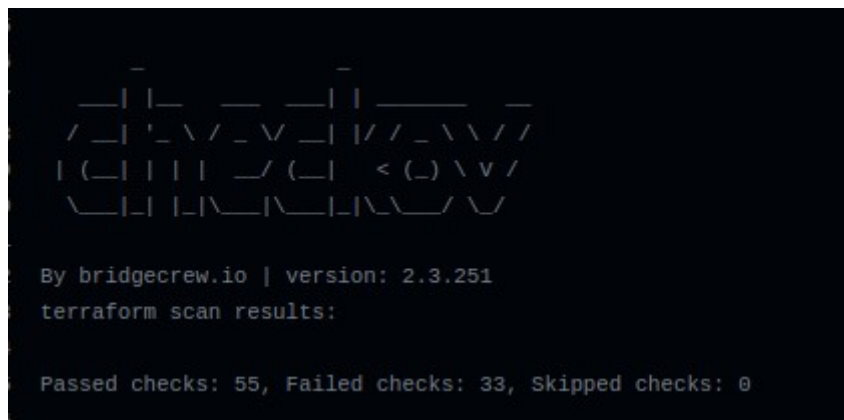
Podobieństwa w stosunku do procesu dostarczania aplikacji, występuję w postaci pewnych zaplanowanych manualnych kontroli bezpieczeństwa po zakończeniu procesu. Bezpiecznik występuję zarówno dla gałęzi „test”, jak i „prod”.

2.2.4.2.

Zastosowanie narzędzia Checkov

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 [CHK01].

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.



```

  _ _ _ _ _
 / _ \ ' _ \ / _ \ / _ \ / _ \ / _ \ / _ \ / _ \ /
 | ( _ | | | _ \ ( _ | < ( _ ) \ \ /
 \ _ | | | \ _ | \ _ | \ _ | \ _ | \ _ | \ _ | \ _ |

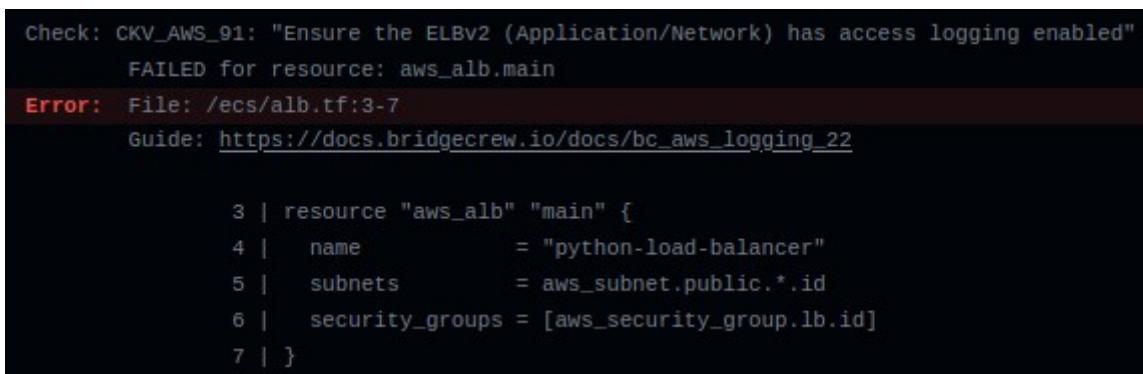
By bridgecrew.io | version: 2.3.251
terraform scan results:

Passed checks: 55, Failed checks: 33, Skipped checks: 0

```

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

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.



```

Check: CKV_AWS_91: "Ensure the ELBv2 (Application/Network) has access logging enabled"
      FAILED for resource: aws_alb.main
Error: File: /ecs/alb.tf:3-7
      Guide: https://docs.bridgecrew.io/docs/bc\_aws\_logging\_22

3 | resource "aws_alb" "main" {
4 |     name           = "python-load-balancer"
5 |     subnets        = aws_subnet.public.*.id
6 |     security_groups = [aws_security_group.lb.id]
7 | }

```

Rysunek 30: Przedstawienie polityki CKV_AWS_91.
Źródło: <https://github.com/PiotrLotr/project-armadillo>

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).

Reguła CKV_AWS_91 **[CKV01]** informuje, 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 statystyki dotyczącej najczęściej odwiedzanych adresów

```
Check: CKV_AWS_2: "Ensure ALB protocol is HTTPS"
      FAILED for resource: aws_alb_listener.front_end
Error: File: /ecs/alb.tf:28-37
      Guide: https://docs.bridgecrew.io/docs/networking\_29

28 | resource "aws_alb_listener" "front_end" {
29 |   load_balancer_arn = aws_alb.main.id
30 |   port              = var.app_port
31 |   protocol          = "HTTP"
32 |
33 |   default_action {
34 |     target_group_arn = aws_alb_target_group.app.id
35 |     type              = "forward"
36 |   }
37 | }
```

Rysunek 31: Przedstawienie polityki CKV_AWS_2. Brak zapewnienia protokołu HTTPS

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

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 **[CKV01]**. Szyfrowanie danych sprawia, że są one praktycznie nieczytelne dla osób trzecich, które mogą przechwytywać przesyłane pakiety i zapobiega atakom typu: „Man in the middle”.

```
Check: CKV_AWS_158: "Ensure that CloudWatch Log Group is encrypted by KMS"
      FAILED for resource: aws_cloudwatch_log_group.db-container-logs
Error: File: /ecs/ecs.tf:31-33
      Guide: https://docs.bridgecrew.io/docs/ensure-that-cloudwatch-log-group-is-encrypted-by-kms

31 | resource "aws_cloudwatch_log_group" "db-container-logs" {
32 |   name = "db-container-logs"
33 | }
```

Rysunek 32: Przedstawienie polityki CKV_AWS_158. Brak zapewnienia szyfrowania kluczem KMS

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

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 **[CKV01]**. Dzięki szyfryzacji, możemy atomizować dostęp do logów, tylko dla ról posiadających uprawnienia do klucza i jego użycia.

```
Check: CKV2_AWS_28: "Ensure public facing ALB are protected by WAF"
      FAILED for resource: aws_alb.main
Error: File: /ecs/alb.tf:3-7
      Guide: https://docs.bridgecrew.io/docs/ensure-public-facing-alb-are-protected-by-waf

3 | resource "aws_alb" "main" {
4 |   name           = "python-load-balancer"
5 |   subnets       = aws_subnet.public.*.id
6 |   security_groups = [aws_security_group.lb.id]
7 | }
```

Figure 33: Przedstawienie polityki CKV2_AWS_28. Zastosowanie AWS WAF

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

Opisując bezpieczeństwo infrastruktury AWS, należy wspomnieć 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.

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.

3. Wnioski z przeprowadzonego projektu

3.1. Stopień spełnienia celów pracy

Analizując cele pracy, można wyciągnąć następujące wnioski:

- Opracowano i zaproponowano proces ciągłego dostarczania aplikacji, zgodnie z założeniami pracy. Następnie, procedura została przeanalizowana i z sukcesem wyszczególniono miejsca podatne z perspektywy bezpieczeństwa.
- Opracowano proces alternatywny, z implementacją DevSecOps.
- Zaproponowano narzędzia redukujące wrażliwości i zabezpieczające kod źródłowy.
- Nie wykonano celu jakim było porównanie narzędzi eliminujących poszczególne wrażliwości.
- Spełniono cel dydaktyczny, jakim było wprowadzenie do zagadnienia DevSecOps. Zachowano balans, dzięki które utrzymano wartość merytoryczną zarówno dla początkujących inżynierów DevOps, jak i tych bardziej zaawansowanych.

3.2. Stopień spełnienia założeń projektowych

Analizując wstępne założenia projektowe, można wyciągnąć następujące wnioski:

- Zastosowano aplikację JavaScript oraz bazę danych MongoDB. Platforma spełnia podstawową funkcjonalność polegającą na streamingu video
- Stworzone rozwiązanie oparte jest na podejściu konteneracyjnym, zgodnie z założeniem.
- Zastosowano środowisko GitHub jako centrum tworzenia automatyzacji.
- Proces przewiduje manualne kroki, związane z weryfikacją inżynierską, co skutkuje niespełnieniem wymogu pełnej automatyzacji. Decyzja o zastosowaniu czynnika ludzkiego została uzasadniona w pracy.
- Proces nie jest kompletnie odzwierciedlony w postaci kodu źródłowego. Priorytetem stało się stworzenie procedury. Stopień odzwierciedlenia procesu w postaci kodu pozostają wysoki.

3.3. Wnioski

3.3.1. DevSecOps nawykiem zamiast jednorazową implementacją

Rozwiązanie projektowe posiada pewien stan początkowy w postaci uprzednio przygotowanej procedury, do której implementowane są elementy bezpieczeństwa. Sytuacja taka odpowiada punktowi wyjścia wielu dużych korporacji, gdzie wprowadzanie odpowiednich procedur bezpieczeństwa jest narzucane „z góry”.

Problem tkwi w fakcie, iż odpowiednia strategia i wdrażanie bezpieczeństwa powinno być wprowadzane już na etapie tworzenia procesów podstawowych. Takie postępowanie pozwala przede wszystkim lepiej zaplanować proces wyjściowy, pozwala zabezpieczyć aplikację w zdecydowanie wcześniejszym etapie punkcie w czasie, zwiększa świadomość pracowników i tworzy w nich nawyk tworzenia bezpieczeństwa.

3.3.2. Wizualizacja procesów ponad pisemne raporty

Koncepcja pisemnych raportów dotyczących każdego utworzonego procesu jest częsta w pracy dzisiejszych przedsiębiorstw. Skutkuje to stertą stron dokumentacji, często ciężkich w odnalezieniu i zrozumieniu. Ma to też skutki pośrednie w postaci zwiększonej demotywacji pracowników oraz docelowo nieprzestrzeganie procedur.

W pracy zastosowano zarówno notację BPMN do wizualizacji procesu oraz jego pisemną reprezentację. Porównując te dwa podejścia, można wywnioskować, iż prezentacja graficzna cechuje się większą czytelnością, łatwością zrozumienia, kompaktowością treści. W rozumienia autora, podejście takie przyspiesza również zapamiętywanie oraz szybciej się je dokumentuje.

3.3.3. Pełna automatyzacja nie zawsze jest optymalna

Dążenie do pełnej automatyzacji jest powszechne, głównie z powodu redukcji kosztów, eliminacji czynnika i błędów ludzkich oraz powtarzalności. Pomimo wielu zauważalnych zalet, istnieją również pewne przeciwwskazania unikania manualnych akcji.

W zaprezentowanym projekcie, każdy proces zawiera kroki związane z manualną weryfikacją. Podejście takie wynika z faktu, iż każdy automat porusza się w obrębie pewnych zdefiniowanych reguł i brakuje mu analizy abstrakcyjnej. Istotne jest z całą pewnością zachowanie bramek jakościowych i autonomicznych sprawdzeń, natomiast aby otrzymać optymalny stopień weryfikacji i kontroli jakości, konieczny jest czynnik ludzki.

Bibliografia

- [CKV01]** Prisma Cloud. Opis wrażliwości CKV.
<https://www.checkov.io/5.Policy%20Index/serverless.html> (pobrano dnia: 13/06/2023)
- [CVE01]** Korporacja Mitre. Opis programu CVE.
<https://www.cve.org/About/Overview> (pobrano dnia: 13/06/2023)
- [CVE02]** Korporacja Mitre. Opis podatności CVE-2022-2900.
<https://avd.aquasec.com/nvd/2022/cve-2022-2900/> (pobrano dnia: 13/06/2023)
- [CVE03]** Korporacja Mitre. Opis podatności CVE-2021-44906.
<https://avd.aquasec.com/nvd/2021/cve-2021-44906/> (pobrano dnia: 13/06/2023)
- [CVE04]** Korporacja Mitre. Opis podatności CVE-2019-14697.
<https://avd.aquasec.com/nvd/2019/cve-2019-14697/> (pobrano dnia: 13/06/2023)
- [DEB01]** G.Kim, J.Humble, P.Debois, J.Willis, N.Forsgren.:
DevOps. Światowej klasy zwinnosć, niezawodność i bezpieczeństwo w Twojej organizacji.
Wydanie 2. Wyd.: OnePress, 2023. ISBN 978-83-283-9686-9.
- [JEV01]** J.Vehent.: Securing DevOps. Security in the Cloud.
Wyd.: Manning, 2018. ISBN 9781617294136.
- [SON01]** SonarSource. Dokumentacja narzędzia Sonarqube.
<https://docs.sonarqube.org/latest/> (pobrano dnia: 13/06/2023)
- [TRV01]** Aqua. Dokumentacja narzędzia Trivy.
<https://aquasecurity.github.io/trivy/v0.42/> (pobrano dnia: 13/06/2023)
- [WEB01]** G.Kim, J.Humble, P.Debois, J.Willis, N.Forsgren. Manifest DevSecOps.
<https://www.devsecops.org/> (pobrano dnia: 13/06/2023)
- [WEB02]** Cloud Box, Chmurzasty Blog. Przegląd modeli Cloud.
<https://cloud-box.pl/iaas-paas-saas-faas/> (pobrano dnia: 13/06/2023)

Spis rysunków

Rysunek 1: Schemat infrastruktury projektu. Podział sieci. Źródło: zbiór autora.....	12
Rysunek 2: Schemat infrastruktury projektu. Wejście do aplikacji z perspektywy użytkownika zewnętrznego. Źródło: zbiór autora.....	13
Rysunek 3: Schemat procesu dostarczania infrastruktury. Źródło: zbiór autora.....	14
Rysunek 4: Schemat procesu dostarczania infrastruktury dla środowiska testowego.....	14
Rysunek 5: Schemat procesu dotarczania infrastruktury dla środowiska produkcyjnego. Źródło: zbiór autora.....	15
Rysunek 6: Schemat procesu CI dla środowiska testowego. Źródło: zbiór autora.....	17
Rysunek 7: Schemat procesu CD dla środowiska testowego. Źródło: zbiór autora.....	18
Rysunek 8: Schemat procesu CI dla środowiska produkcyjnego. Źródło: Zbiór autora.....	19
Rysunek 9: Schemat procesu CD dla środowiska produkcyjnego. Źródło: zbiór autora.....	19
Rysunek 10: Schemat podejścia TDS (Test Driven Development).....	22
Rysunek 11: Zarządzanie sekretami w GitHub Źródło: https://github.com/PiotrLotr/project-armadillo	23
Rysunek 12: Implementacja DevSecOps dla procesu CI/CD. Środowisko testowe. Źródło: zbiór autora.....	24
Rysunek 13: Implementacja DevSecOps dla procesu CD. Środowisko testowe.....	25
Rysunek 14: Implementacja DevSecOps dla procesu CD. Środowisko produkcyjne.....	25
Rysunek 15: Przegląd miar narzędzia Sonarqube.....	26
Rysunek 16: Spis podatności kodu źródłowego.....	27
Rysunek 17: Szczegółowe przedstawienie podatności Źródło: https://sonarcloud.io	27
Rysunek 18: Spis "Security Hotspots" kodu źródłowego. Sekcja: priorytet wysoki. Źródło: https://sonarcloud.io	28
Rysunek 19: Spis "Security Hotspots" kodu źródłowego. Sekcja: priorytet średni/niski. Źródło: https://sonarcloud.io	28
Rysunek 20: Progi bramki jakościowej kodu źródłowego Źródło: https://sonarcloud.io	29
Rysunek 21: Raport podatności narzędzia Trivy.....	31
Rysunek 22: Przedstawienie polityki CVE-2022-2900. Server-Side Request Forgery (SSRF) Źródło: https://github.com/PiotrLotr/project-armadillo	32
Rysunek 23: Przedstawienie polityki CVE-2022-2900. Server-Side Request Forgery (SSRF) Źródło: https://avd.aquasec.com/nvd/2022/cve-2022-2900/	32
Rysunek 24: Przedstawienie polityki CVE-2021-44906. Podatność „Prototype pollution” Źródło: https://github.com/PiotrLotr	33
Rysunek 25: Przedstawienie polityki CVE-2021-44906. Podatność „Prototype pollution” Źródło: https://avd.aquasec.com/nvd/2021/cve-2021-44906/	34
Rysunek 26: Przedstawienie polityki CVE-2019-14697. Podatność „Out-of-bounds Write” Źródło: https://github.com/PiotrLotr	34
Rysunek 27: Przedstawienie polityki CVE-2019-14697. Podatność „Out-of-bounds Write” Źródło: https://avd.aquasec.com/nvd/2019/cve-2019-14697/	35
Rysunek 28: Implementacja DevSecOps do procesu dostarcza infrastruktury. Środowisko testowe i produkcyjne. Źródło: zbiór autora.....	36
Rysunek 29: Stosunek wyników pozytywnych/negatywnych w raporcie Checkov Źródło: https://github.com/PiotrLotr/project-armadillo	37
Rysunek 30: Przedstawienie polityki CKV_AWS_91. Źródło: https://github.com/PiotrLotr/project-armadillo	37
Rysunek 31: Przedstawienie polityki CKV_AWS_2. Brak zapewnienia protokołu HTTPS.....	38

Rysunek 32: Przedstawienie polityki CKV_AWS_158. Brak zapewnienia szyfrowania kluczem KMS	
Źródło: https://github.com/PiotrLotr/project-armadillo	38
Figure 33: Przedstawienie polityki CKV2_AWS_28. Zastosowanie AWS WAF Źródło:	
https://github.com/PiotrLotr/project-armadillo	39