



POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Nazwa Wydziału

Nazwa Katedry

Katedra Inżynierii Oprogramowania
Zarządzanie projektami (IM)

Piotr Wrona

Nr albumu s25773

Tytuł pracy dyplomowej

“Procedura bezpieczeństwa dla ciągłej integracji oprogramowania oparta na DevSecOps.”

Rodzaj pracy
magisterska

Imię i nazwisko promotora
Dr. Hab. Inż. Piotr Habela

miejsce, miesiąc, rok obrony

Spis Treści

Wstęp.....	4
Cel pracy.....	4
Założenia.....	4
Praktyki DevSecOps.....	6
Geneza.....	6
Cele DevSecOps.....	6
Manifest DevSecOps.....	6
Podejścia do DevSecOps.....	8
Rozwiązywanie projektowe.....	9
Rozwiązywanie wejściowe.....	9
Wybór struktury aplikacji.....	9
Wybór środowiska aplikacji.....	10
Wybór bazy danych.....	10
Wybór modelu i architektury dla projektu.....	11
Modele infrastrukturalne.....	11
Propozycja infrastruktury.....	11
Wejściowy proces CI/CD dla projektu.....	14
Cel projektowania i stosowania CI/CD.....	14
Propozycja procesu CI/CD dla projektu.....	15
Rozwiązywanie wyjściowe.....	18
Planowanie bezpieczeństwa.....	18
Miary bezpieczeństwa procesu CI/CD.....	18
Planowanie implementacji do procesu CI/CD.....	19
Zarządzanie sekretami.....	20
Udoskonalony proces CI/CD.....	21
Rola kodu źródłowego w bezpieczeństwie aplikacji.....	23
Zastosowanie SonarQube w celu zwiększenia bezpieczeństwa.....	23
Wprowadzenie bramki jakościowej dla kodu źródłowego.....	26
Bezpieczeństwo infrastruktury i implementacja rozwiązania.....	27
Bezpieczeństwo infrastruktury cloudowej.....	27
Skanowanie infrastruktury.....	27
Wnioski z przeprowadzonego projektu.....	28
Stopień spełnienia założeń projektowych.....	28

Wstęp

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 porządkuje 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

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 przewiduje pełnej automatyzacji.
- Proces nie jest kompletnie odzwierciedlony w postaci kodu źródłowego. Najistotniejsze 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

DevSecOps to koncepcja, która łączy w sobie praktyki DevOps z bezpieczeństwem. Pomyśl 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.

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 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 kodu - 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

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 manifest¹. 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 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.

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

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

„Open Contribution & Collaboration over Security-Only Requirements”

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.

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

Priorytet powinno być 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 zwykłego ohczenia działania jako zrobione.

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

Tworzenie zespołów czerwony/niebiescy do nauki obrony naszego systemu. W założeniu podejście to odwzorowuje bardziej realistyczne 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.

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 skupią 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

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
<ul style="list-style-type: none">Potencjalna awaria powoduje duże problemy funkcjonalnościowe i może sparaliżować cały systemZmiany w aplikacji są trudne, ze względu na złożoność kodu źródłowegoŚrodowiska 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 systemuZmiany w aplikacji są proste, ze względu na rozdzielenie kodu źródłowego na mniejsze częściŚrodowisko 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 przewiduję 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.

Wybór środowiska aplikacji

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 aplikacji

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

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 wartości. Wybór bazy padł na nierelacyjną bazę mongodb.

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 zaprojektowaną z myślą o łatwej replikacji

Wybór modelu i architektury dla projektu

Modele infrastrukturalne

Główny podział infrastruktury współczesnych aplikacji opiera się na lokalizacji serwerów i dysponowaniem zakresem obowiązków dotyczących obsługi. Wyróżniamy:

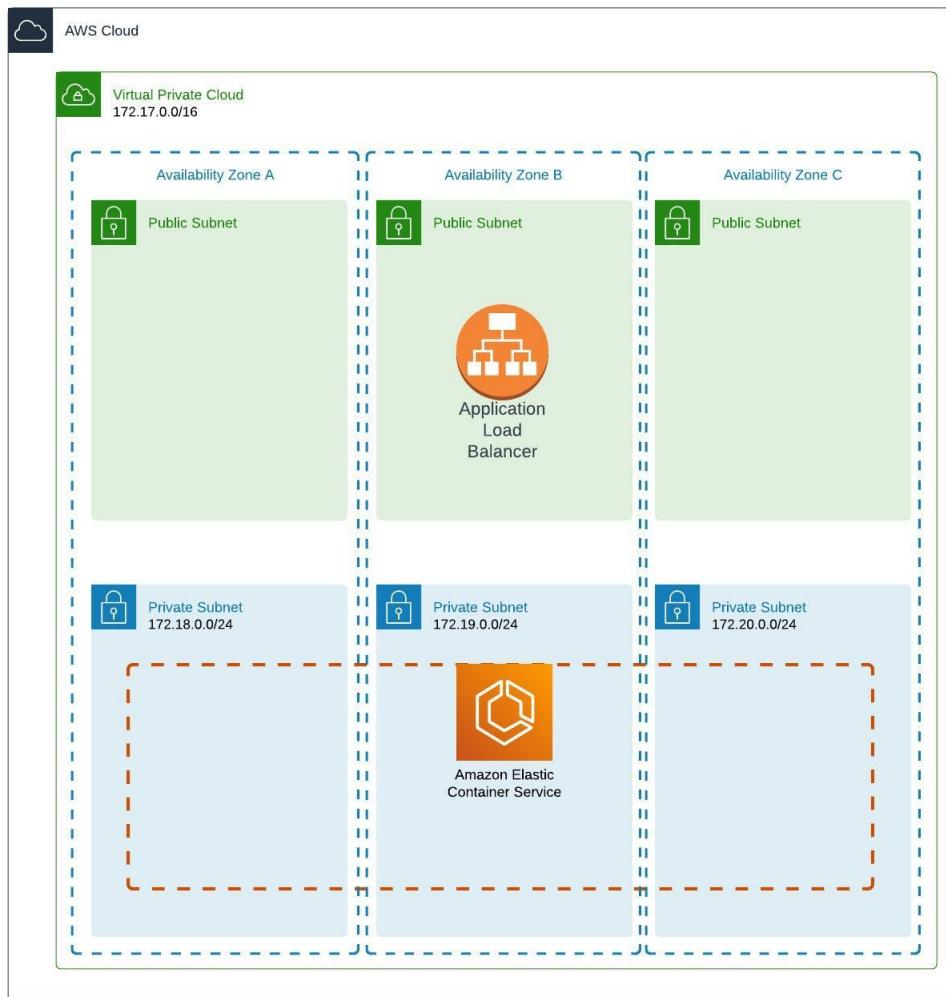
- On-premises – infrastruktura zlokalizowana w przedsiębiorstwie. Pełna obsługa hardware'u przypada na zatrudnionych administratorów sieci.
- Cloud – w tym modelu przesuwamy odpowiedzialność za obsługę sprzętu na zewnętrznych dostawców. Ze względu na zakres obowiązków, w rozwiązańach chmurowych możemy 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.

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 definiuję naszą sieć i zakres adresów. Architektura jest 2-tierowa i wyróżnia 3 podsieci publiczne oraz 3 podsieci prywatne. Cała sieć znajduje się w jednym regionie eu-central-1 (Frankfurt), co w pewien sposób powoduje podatność 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: autorstwo własne

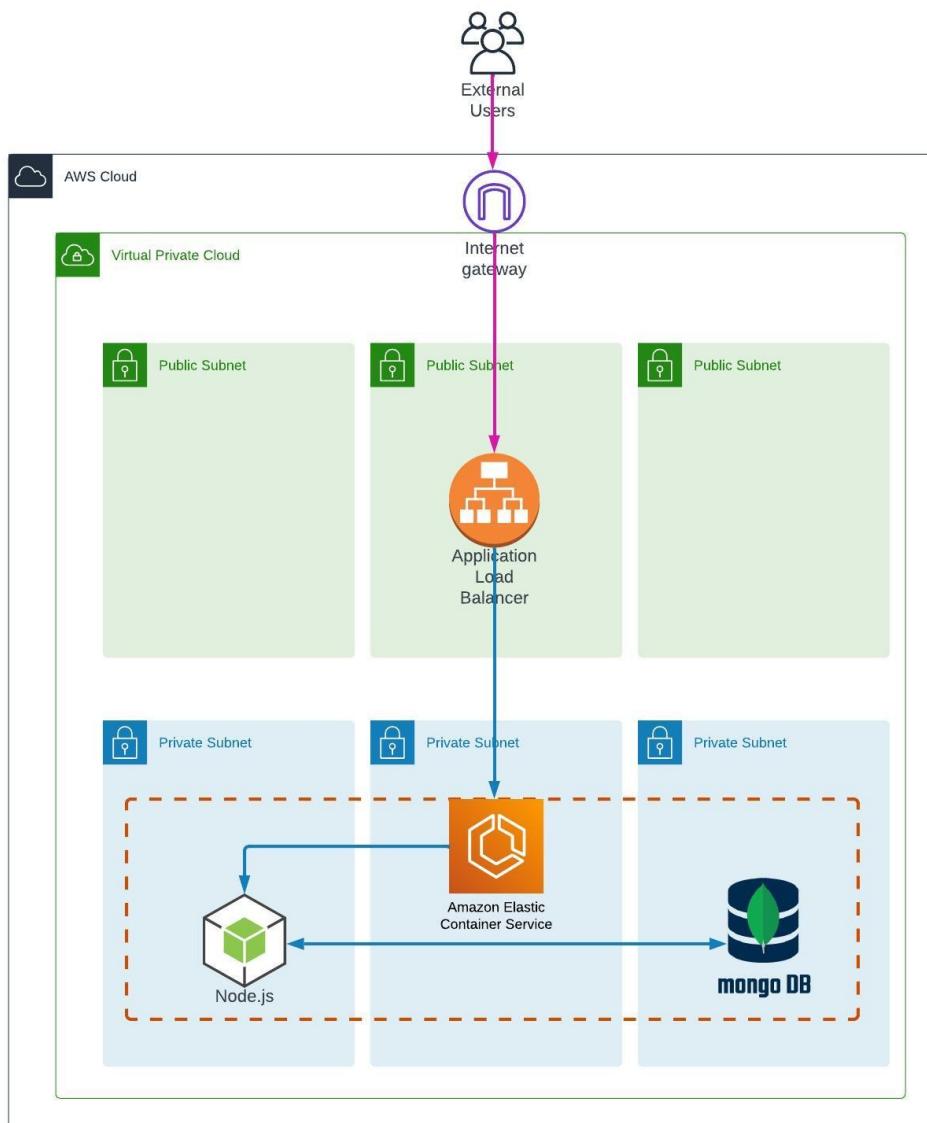
Każda podsieć publiczna ma maskę 255.255.255.0 która obejmuje 254 adresy. Dla naszego rozwiązania projektowe zakres możliwych destynacji jest nieproporcjonalnie duży, natomiast umożliwia to ewentualne dalsze rozwijanie infrastruktury czy możliwości skalowania.

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

wysokie zapewnienie dostępności - 3 dostępne, redundatne strefy dostępu

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

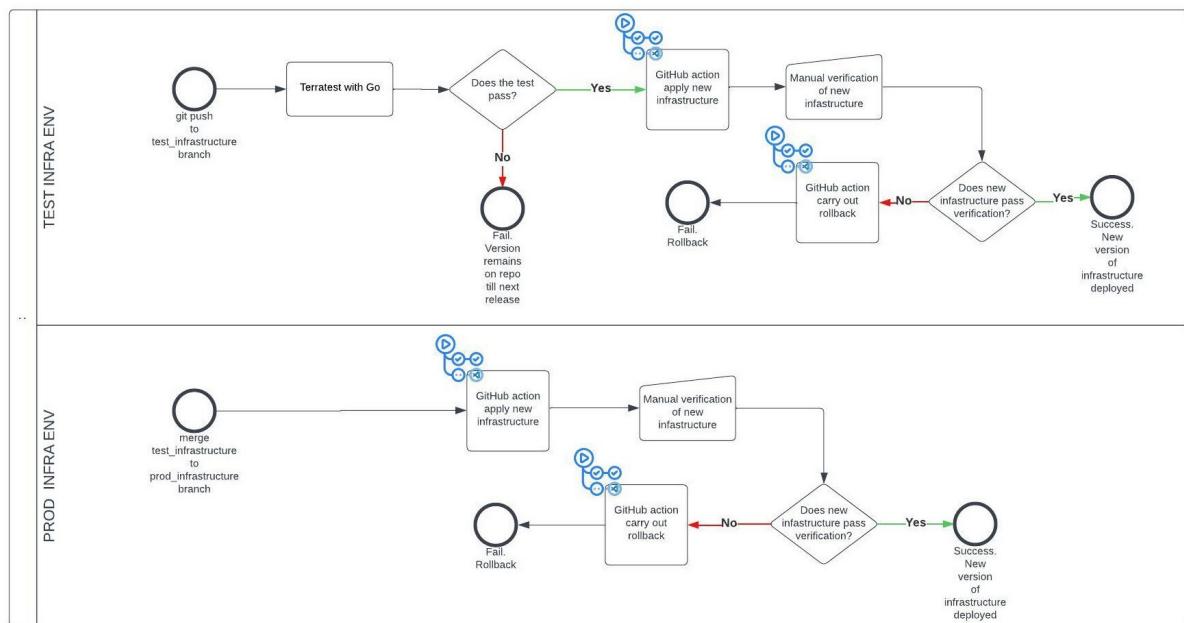


Rysunek 2: Schemat infrastruktury projektu. Wejście do aplikacji z perspektywy użytkownika zewnętrznego.

Źródło: autorstwo własne

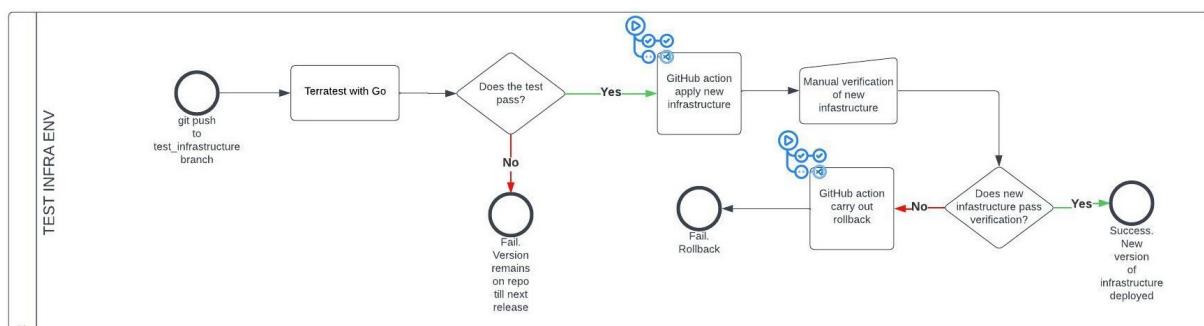
Proces ciągłego dostarczania infrastruktury

W procesie dostarczania infrastruktury wyróżniamy środowiska: testowe oraz produkcyjne. Dla każdego zostały przydzielone branch'e: „test_infrastructure” oraz „prod_infrastructure”, których zawartość reflektuje w faktycznej infrastrukturze.



Rysunek 3: Schemat procesu dostarczania infrastruktury. Źródło: autorstwo własne

Aby zainicjować cykl testowy, konieczne jest wypuszczenie kodu do odpowiedniej gałęzi. Rozpoczyna to tzw.: „Terratest”, który jest framework'm open-source 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 można zwiększyć pewność, że wdrożenie działa zgodnie z oczekiwaniemi.

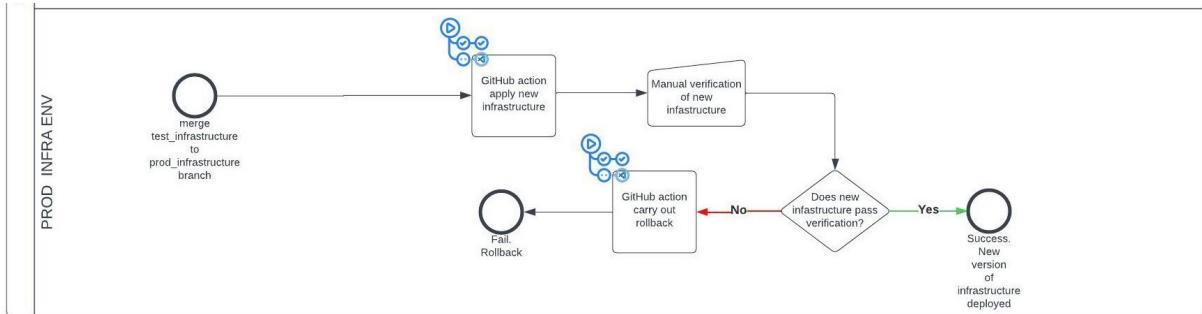


Rysunek 4: Schemat procesu dostarczania infrastruktury dla środowiska testowego.

Źródło: autorstwo własne

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 – transkacja 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 ze środowiskiem testowym z środowiskiem produkcyjnym.



Rysunek 5: Schemat procesu dostarczania infrastruktury dla środowiska produkcyjnego.
Źródło: autorstwo własne

Połączenie gałęzi rozpoczęta akcją 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 – transkacja jest zakończona sukcesem, w przypadku jej braku- następuje rollback do poprzedniej wersji infrastruktury.

Wejściowy proces CI/CD dla projektu

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 wzdrażania, czyli CD.

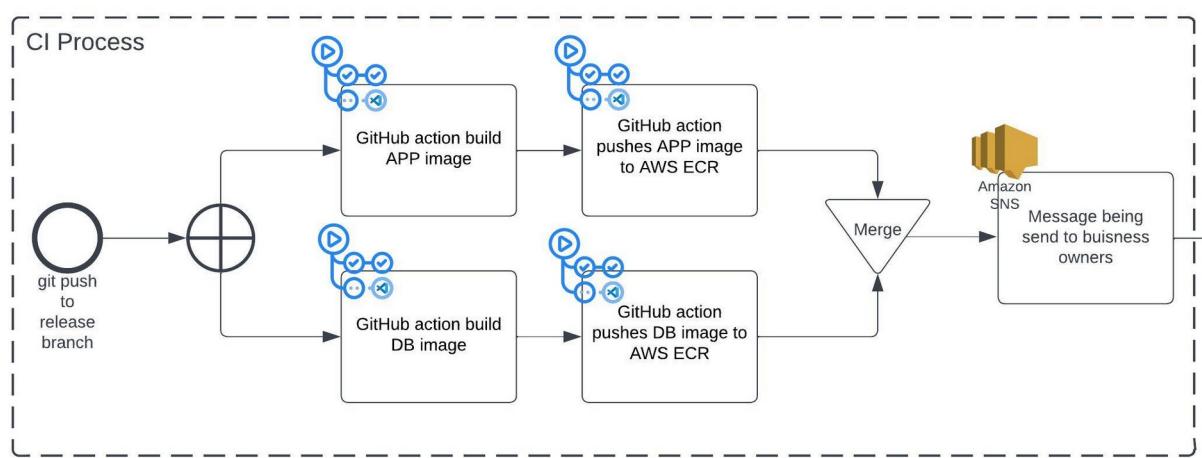
Podejście do schematu dostarczania oprogramowania jest bardzo różne i zależy głównie od:

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

Propozycja procesu CI/CD dla projektu

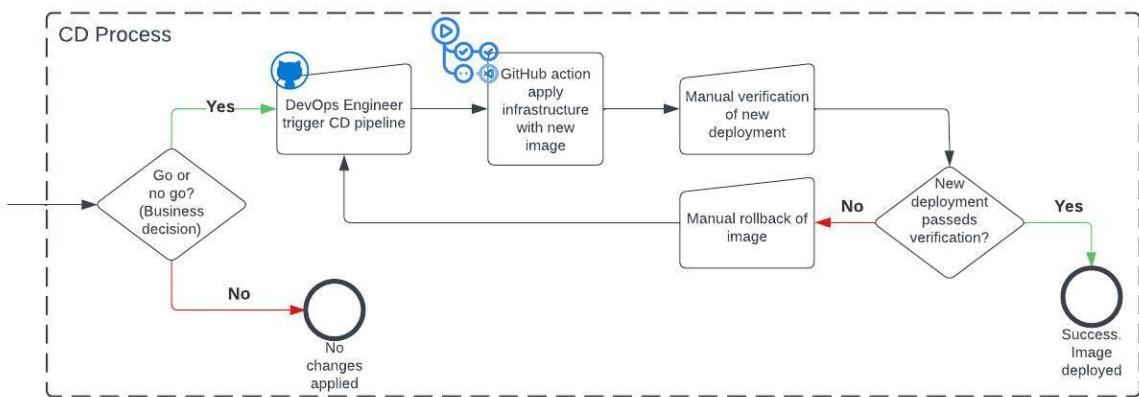
Projekt zakłada jedno środowisko testowe (TEST ENV) oraz jedno środowisko produkcyjne (PROD ENV). Należy nadmienić, że podejście jest uproszczone w stosunku do najczęstszych wzorów komercyjnych. Dla rozwiązań rynkowych przewiduję się zazwyczaj dwa środowiska testowe (np.: QA (Quality Assurance) oraz UAT (User Acceptance Tests)), jak również dwa środowiska produkcyjne, głównie dla celów przełączania w czasie awarii.

Podejście konteneryzacyjne od aplikacji, powoduje pewną centralizację procesu wokół repozytorium obrazów dockerowych. Wpływa to zarówno część integracyjną - obok udostępnienia kodu, następuje udostępnienie obrazu, jak również część wdrożeniową - poprzez pobranie najnowszej wersji obrazu i zaaplikowanie na środowisku.



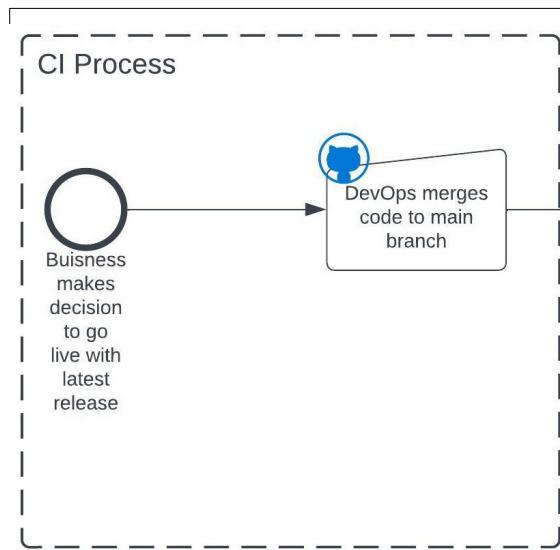
Rysunek 6: Schemat procesu CI dla środowiska testowego. Źródło: autorstwo własne

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 równolegle przetwarzanie dwóch pipelinów – 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: autorstwo własne

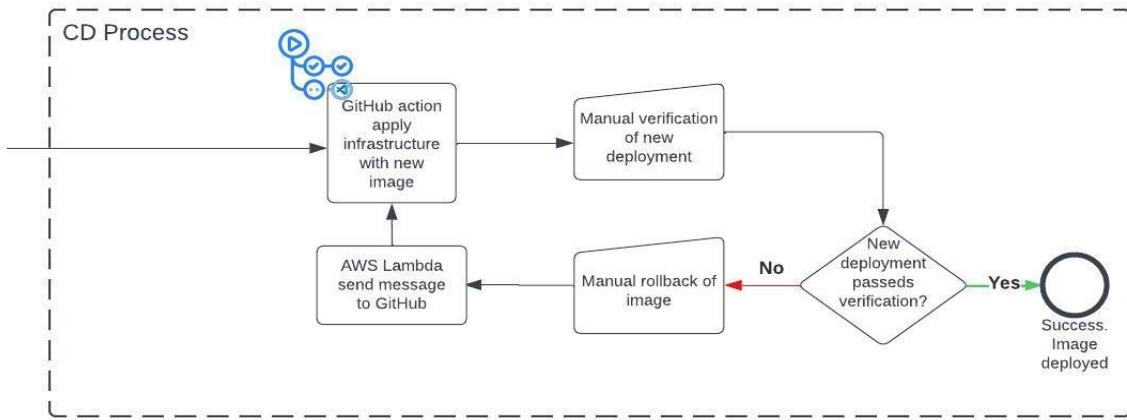
1. Proces wdrożeniowy jest inicjowany poprzez decyzję biznesową. Na tym etapie występuje możliwość rezygnacji z nowego wydania. Taka akcja nie wymaga dodatkowych kroków, gdyż rollback następuje przy kolejnej budowie obrazów.
2. Automat aplikuje nowe rozwiązanie przy użyciu narzędzia terraform.
3. Inżynier 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, 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: autorstwo własne

Z perspektywy środowiska produkcyjnego, część 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 inicjuje część wdrożeniową.



Rysunek 9: Schemat procesu CD dla środowiska produkcyjnego. Źródło: autorstwo własne

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, następuje 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.

Rozwiążanie wyjściowe

Planowanie bezpieczeństwa

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 oczekiwaniemi. 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.

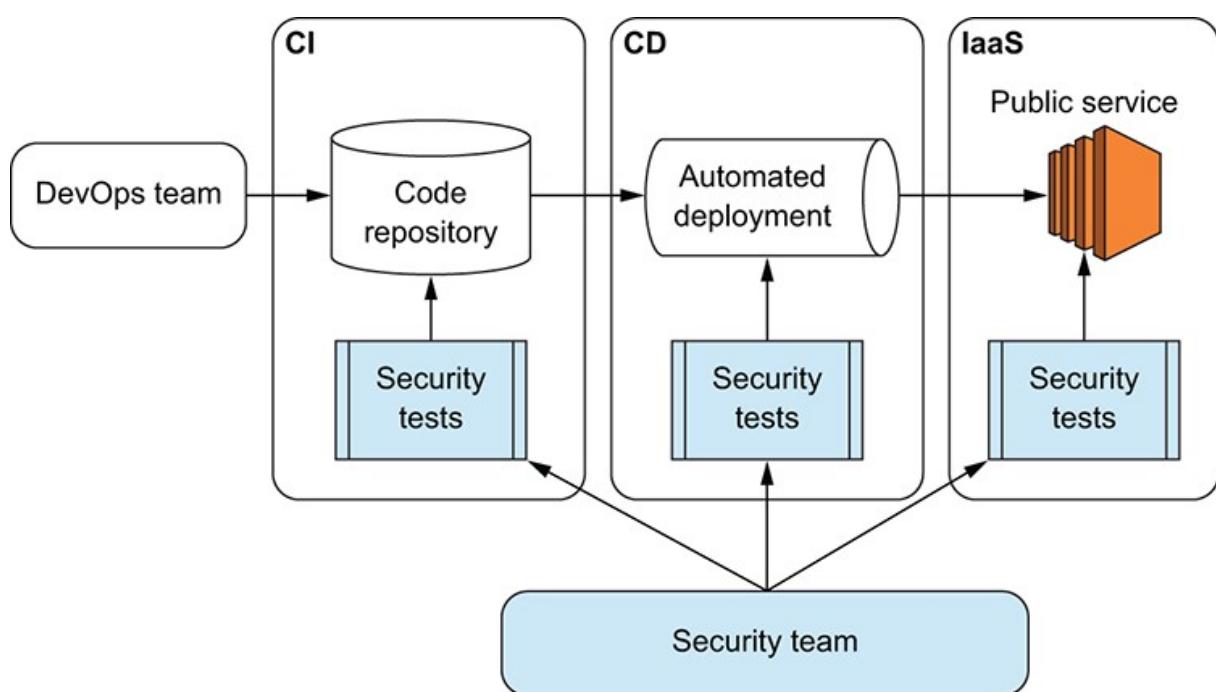
Wszystkie te miary mają na celu zabezpieczenie procesu CI/CD przed zagrożeniami i zapewnienie, że system działa zgodnie z oczekiwaniemi.

Planowanie implementacji do procesu CI/CD

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.

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



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.

Zarządzanie sekretami

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

Rynek oferuję 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 przetrzymywania haseł.

The screenshot shows the 'Actions secrets and variables' section of a GitHub repository settings page. On the left, there's a sidebar with various repository management options like General, Access, Collaborators, and Moderation options. Below that is a 'Code and automation' section with branches, tags, actions, webhooks, environments, codespaces, and pages. Under 'Security', there are code security analysis, deploy keys, and a 'Secrets and variables' section which is currently selected. Other sections include Actions, Codespaces, Dependabot, Integrations, GitHub Apps, and Email notifications. The main content area is titled 'Actions secrets and variables'. It contains two tabs: 'Secrets' (selected) and 'Variables'. A green button at the top right says 'New repository secret'. Below the tabs, there are two sections: 'Environment secrets' (which is empty) and 'Repository secrets'. The 'Repository secrets' section lists four secrets with their names, last updated times, and edit/delete icons:

Secret	Last Updated	Action
AWS_ACCESS_KEY_ID	Updated 2 weeks ago	
AWS_SECRET_ACCESS_KEY	Updated 2 weeks ago	
MONGO_INITDB_ROOT_PASSWORD	Updated 16 hours ago	
MONGO_INITDB_ROOT_USERNAME	Updated 16 hours ago	

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.

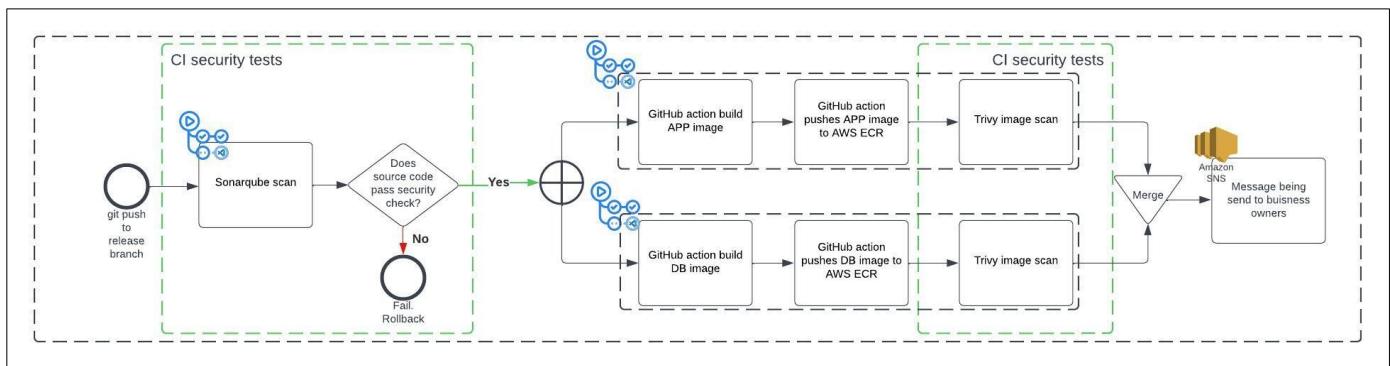
Pomimo, że hasła znajdują się repozytorium publicznym, są one niedostępne dla nieupoważnionych osób. Kontrola dostępu odbywa się poprzez dodawanie/usuwanie użytkowników projektowych oraz poprzez ich role.

Proces CI/CD wzbogacony o aspekty bezpieczeństwa

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.

Gwoli wprowadzenia należy wymienić główne podobieństwa do procesu wejściowego:

- 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 poprzez decyzję biznesową

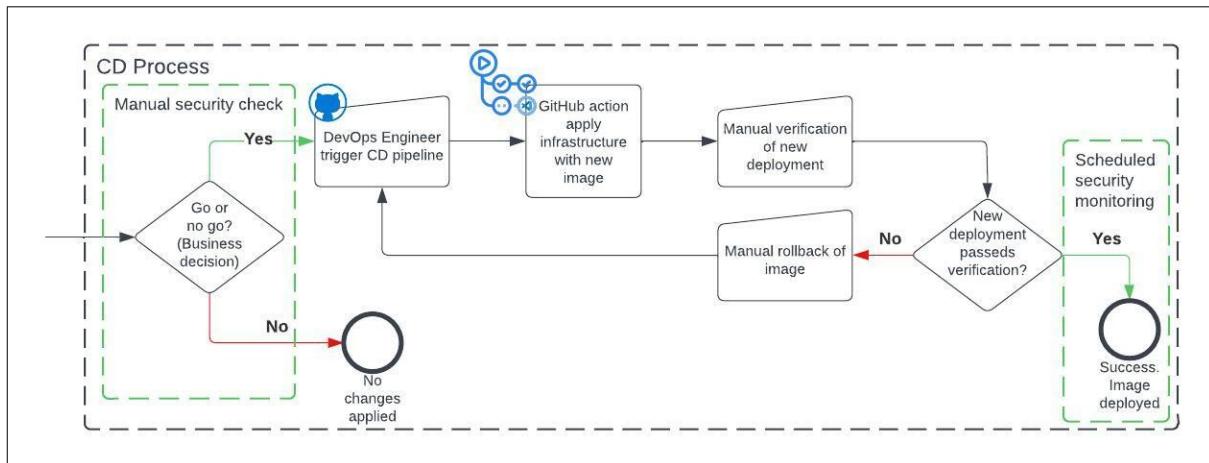


1. Proces CI dla środowiska testowego w dalszym ciągu jest inicjowany przez wypuszczenie zmiany na release branch.
2. Zaraz po pojawienniu się kodu w repozytorium następuje główna zmiana w stosunku do poprzedniego procesu. Otóż, inicjowany jest pipeline, który odpowiada za sprawdzenie bezpieczeństwa 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 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 podmieniany przy kolejnej budowie obrazu.
6. Informacja o nowych obrazach oraz o wygenerowaniu nowego raportu jest kierowany do binzesu.

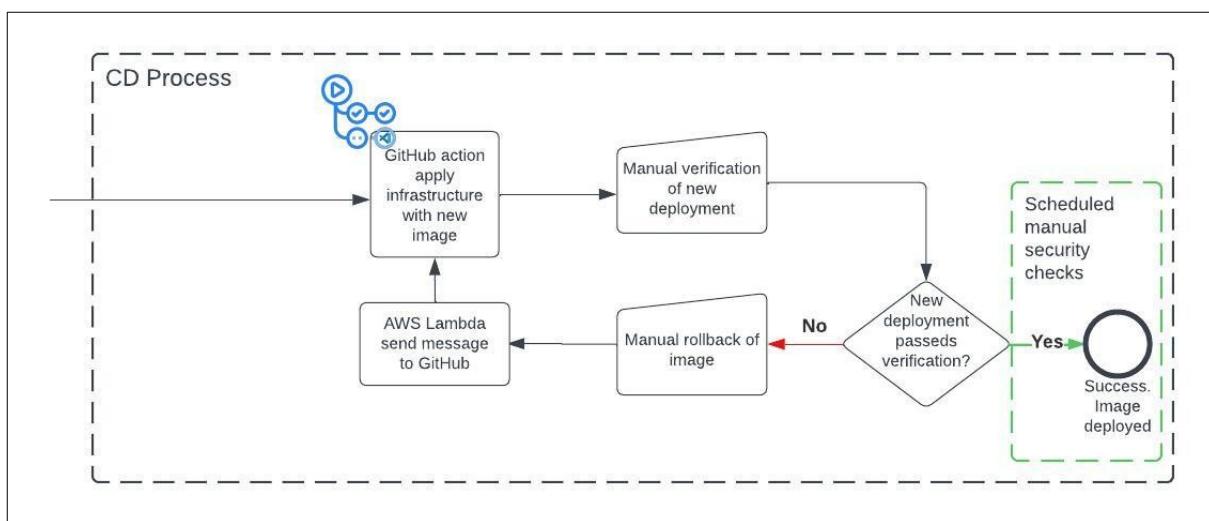
W przypadku procesu ciągłego dostarczania, zmiany są mniej znaczące i ograniczają się do pewnych weryfikacji manualnych. W przypadku procesu integracji kody z uwzględnieniem bezpieczeństwa, istnieją dwa dodatkowe zawory bezpieczeństwa. Pomimo faktu istnienia tych zabezpieczeń, zakładamy rozpoczęcie procesu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

- Zabezpieczenie przed ewentualnym błędem automatu
- Analiza raportu zwiększa świadomość i wiedzę w zakresie bezpieczeństwa
- Możliwość dostrzeżenia faktów, które z poziomu skryptu nie są istotne (automat ma pewne progi ustalone przez człowieka)
- Skutkiem ubocznym manualnej weryfikacji może być również wykrycie błędów funkcjonalnych aplikacji



Obok manualnej weryfikacji inicjujący proces, konieczny jest proces ciągłej weryfikacji po zakończeniu deploymentu. Rekomendowane jest wyznaczenie rotowalnej roli zespołowej, odpowiedzialnej za raportowanie podczas danego sprintu. Takie rozwiązanie uwzględnia inną kolejną istotną funkcję polegającą na dzieleniu wiedzy 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.

Rola kodu źródłowego w bezpieczeństwie aplikacji

Zastosowanie Sonarqube w celu zwiększenia bezpieczeństwa

Pomimo dążenia do mikroserwisów, kod źródłowy dzisiejszych aplikacji to nadal wiele wierszy obfitujących w funkcję zaczerpniętą 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 języków 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.
- **Vulnerabilities** – 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.

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.

	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%

Przedstawiony powyżej obraz pochodzi z sonarcloud.io, który spełnia rolę serwera sonarqube i pozwala na ich graficzną analizę.

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

app/app/app_files/sql-offset/date_pagination.py	<input type="checkbox"/> Add password protection to this database.	No tags +
	🔒 Vulnerability ▾ ○ Open ▾ ⚡ Blocker ▾ 📱 Piotr Wrona ▾	45min effort × 20 days ago
app/app/app_files/sql-offset/id_pagination.py	<input type="checkbox"/> Add password protection to this database.	No tags +
	🔒 Vulnerability ▾ ○ Open ▾ ⚡ Blocker ▾ 📱 Piotr Wrona ▾	45min effort × 20 days ago
app/app/app_files/sql-offset/offset_pagination.py	<input type="checkbox"/> Add password protection to this database.	No tags +
	🔒 Vulnerability ▾ ○ Open ▾ ⚡ Blocker ▾ 📱 Piotr Wrona ▾	45min effort × 20 days ago
app/app/app_files/sql-offset/seed.py	<input type="checkbox"/> Add password protection to this database.	No tags +
	🔒 Vulnerability ▾ ○ Open ▾ ⚡ Blocker ▾ 📱 Piotr Wrona ▾	45min effort × 20 days ago
app/app/app_files/sql-offset/timeline.py	<input type="checkbox"/> Add password protection to this database.	No tags +
	🔒 Vulnerability ▾ ○ Open ▾ ⚡ Blocker ▾ 📱 Piotr Wrona ▾	45min effort × 20 days ago

Głębszy raport wskazuję na naruszenie protokołów: OWASP A2, A3ⁱⁱ oraz CWE-521ⁱⁱⁱ. Pierwsze dwa kolejną odpowiadają za złamianą autentykację oraz udostępnienie danych wrażliwych. Natomiast ostatni, polega na złamaniu polityki dotyczącej wykorzystywania silnych haseł.

Add password protection to this database. 🔒

A secure password should be used when connecting to a database [python:S2115](#)

🔒 Vulnerability ▾ ○ Open ▾ ⚡ Blocker ▾ 📱 Piotr Wrona ▾

Where is the issue? Why is this an issue? Activity

app/app/app_files/sql-offset/date_pagination.py 🔎

```
1  pwrona...
2
3
4
5      import psycopg2
6      import time
7      try:
8          connection = psycopg2.connect(user = "Abdi",
9                                         password = "",
```

🔒 Add password protection to this database.

```
6
7
8
9          host = "localhost",
10         port = "5432",
11         database = "twitter")
```

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.

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”.

The screenshot shows a security review interface. On the left, there's a summary of 8 Security Hotspots to review, categorized by priority: High, Medium, and Low. The High priority section is expanded, showing a specific hotspot related to handling secrets in a Dockerfile. The Dockerfile code is displayed on the right, with line 1 highlighted in red. A callout box points to this line with the message: "Make sure that using ARG to handle a secret is safe here." The Dockerfile code snippet is as follows:

```

FROM node:12.7.0-alpine
ARG MONGO_INITDB_ROOT_USERNAME
ARG MONGO_INITDB_ROOT_PASSWORD
ARG MONGO_INITDB_DATABASE
ENV MONGO_INITDB_ROOT_USERNAME=${MONGO_INITDB_ROOT_USERNAME}
ENV MONGO_INITDB_ROOT_PASSWORD=${MONGO_INITDB_ROOT_PASSWORD}
ENV MONGO_INITDB_DATABASE=${MONGO_INITDB_DATABASE}

```

Below the Dockerfile, there's a link to "Show 12 more lines".

Raport wskazuje 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 uprawnień, które są szczególnie pomocne dla późniejszego opanowania kontenera bazy danych.

The screenshot shows another part of the security review interface. On the left, the "Permission" section is expanded, showing a hotspot about running as root. The Dockerfile code on the right has line 1 highlighted in red. A callout box points to this line with the message: "The node image runs with root as the default user. Make sure it is safe here." The Dockerfile code snippet is as follows:

```

FROM node:12.7.0-alpine

```

Below the Dockerfile, there's a link to "Show 15 more lines".

Wprowadzenie bramki jakościowej dla kodu źródłowego

Podejście TDS wymaga wprowadzenia progów jakościowych, dlatego w tym podrozdziale przedstawione zostaną zalety stosowania bramki jakościowej SonarQube w procesie tworzenia 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

Zastosowanie Trivy w celu zwiększenia bezpieczeństwa obrazów

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.

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. Nieaktualizowane 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 wprzegnięty w akcję Github, a raporty są gromadzone w zakładce Security na stronie głównej projektu. Dostęp do wyników jest kontrolowany poprzez licencję dostępowe.

The screenshot shows the GitHub Security tab with the 'Code scanning' section selected. The sidebar on the left lists various security-related features: Actions, Projects, Wiki, Security (49), Insights, Settings, Overview, Reporting, Policy, Advisories, Vulnerability alerts, Dependabot, Code scanning (49), and Secret scanning. The main area displays the results of the code scanning. It shows a summary: 'All tools are working as expected'. A search bar contains the query 'is:open branch:release tool:Trivy'. Below this, there is a link to 'Clear current search query, filters, and sorts'. The list of vulnerabilities is as follows:

- CVE-2022-2900** (Critical) Library: #29 opened 2 weeks ago • Detected by Trivy in app/.../parse-url/package.json:1
- CVE-2022-2216** (Critical) Library: #28 opened 2 weeks ago • Detected by Trivy in app/.../parse-url/package.json:1
- CVE-2021-44906** (Critical) Library: #23 opened 2 weeks ago • Detected by Trivy in usr/.../minimist/package.json:1
- CVE-2021-44906** (Critical) Library: #22 opened 2 weeks ago • Detected by Trivy in usr/.../minimist/package.json:1
- CVE-2021-3918** (Critical) Library: #20 opened 2 weeks ago • Detected by Trivy in usr/.../json-schema/package.json:1
- CVE-2019-14697** (Critical)
- CVE-2019-14697** (Critical)
- CVE-2022-3517** (High) Library

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 cyberprzestępco.

Code scanning alerts / #29

CVE-2022-2900

[Open](#) in `release` 4 days ago

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

[Show more ▾](#)

Jednym z pierwszych komunikatów jest naruszenie protokołu CVE-2022-2900. Dotyczy on naruszenia SSRF (Server Side Request Forgery) i tłumaczone jest jako: „Podrabianie żądań po stronie serwera”.

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

CVSS 2.x

RedHat/V2

RedHat/V3

Ubuntu

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.

The screenshot shows a GitHub code scanning alert for a vulnerability. At the top, it says "Code scanning alerts / #23" and "CVE-2021-44906". Below that is a green button with a shield icon labeled "Open" and a note "in release 2 weeks ago". The main content area shows a file path "usr/local/lib/node_modules/npm/node_modules/rc/node_modules/minimist/package.json:1" with a preview link. A message "Preview unavailable" and "Sorry, we couldn't find this file in the repository." follows. Below this, detailed information about the vulnerability is listed:

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

Below the table, the word "Trivy" is mentioned. At the bottom, there's a section titled "Vulnerability CVE-2021-44906" with a table showing "Tool" (Trivy) and "Rule ID" (CVE-2021-44906).

Kolejną podatnością jest: CVE-2021-44906, która jest oznaczona jako priorytet krytyczny. Alert oznacza wrażliwość typu: „Prototype pollution” co w tłumaczeniu oznacza zanieczyszczenie prorotypu.

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 Vulnerabilities

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 Mock	RedHat	openshift/service-mock@v1.26.0-1	*



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

CVSS 2.x
CVSS:2.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

RedHat/V2
CVSS:2.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

RedHat/V3
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

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

Source: NVD

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 wskazuję również na złamanie poddatności CVE-2019-14697. Dalsza inwestygacja rozwija ten enigmatyczny skrót o nazwę: „Out-of-bounds Write”.

Code scanning alerts / #13

CVE-2019-14697

 Open in release 2 weeks ago

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

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 Vulnerabilities

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/i386/ 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	*

9.8 CRITICAL Source: NVD

CVSS 3.x: 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		<small>i</small>
RedHat/V3		<small>i</small>
Ubuntu		<small>i</small>

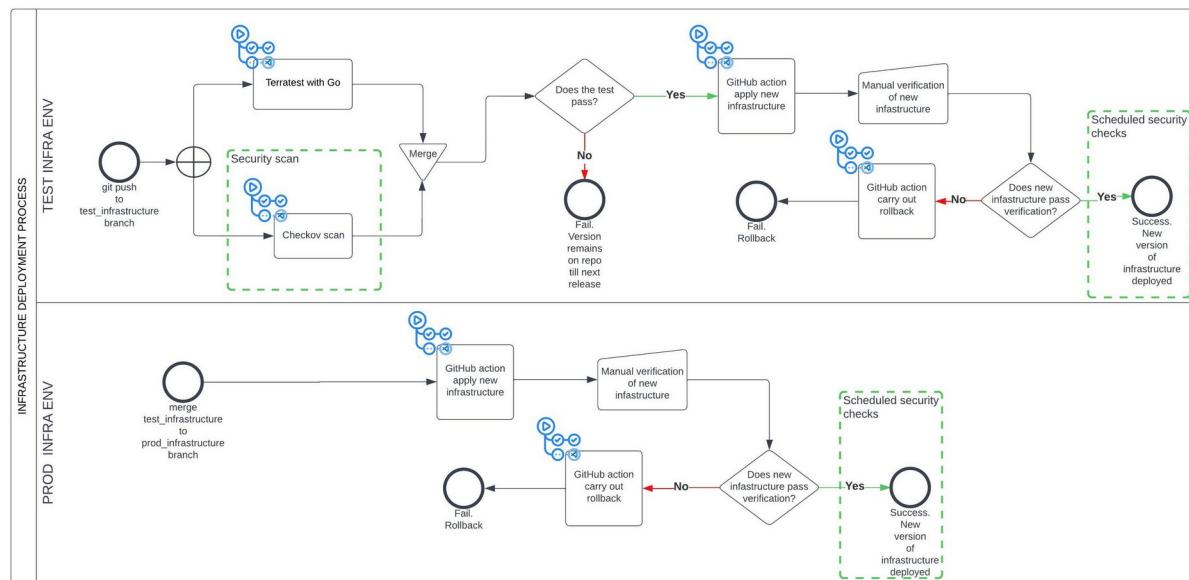
OOBW jest często wykorzystywany przez atakujących do wykonania tzw. ataków typu buffer overflow, w których złośliwy kod może nadpisać ważne dane lub wstrzyknąć kod do pamięci programu i tym samym uzyskać kontrolę nad systemem.

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.

Bezpieczeństwo infrastruktury i implementacja rozwiązania

Proces dostarczania infrastruktury wzbogacony o automatyczne bezpieczeństwo

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.



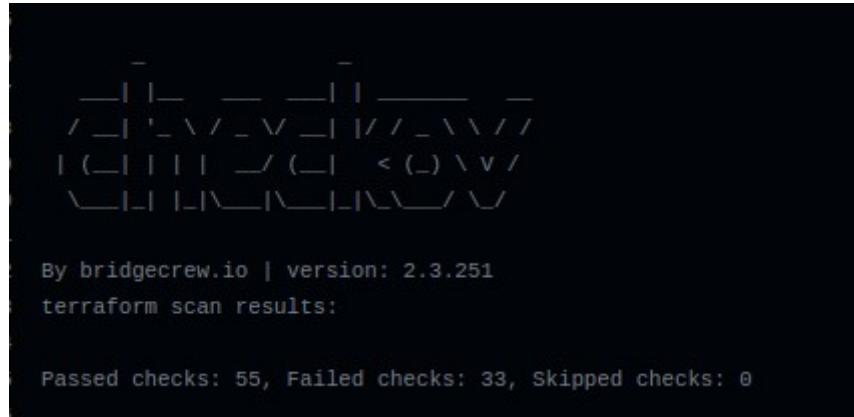
Kontynuacja procesu dostarczania infrastruktury następuje jedynie po złączeniu wyników równoległego przetwarzania, a wyniki są poddawane walidacji. W przypadku sukcesów w dla obu procesów – następuje automatyczny proces dostarczenia zmian. W przypadku porażki kod pozostaje w repozytorium do jego kolejnego nadpisania, natomiast procedura się kończy.

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

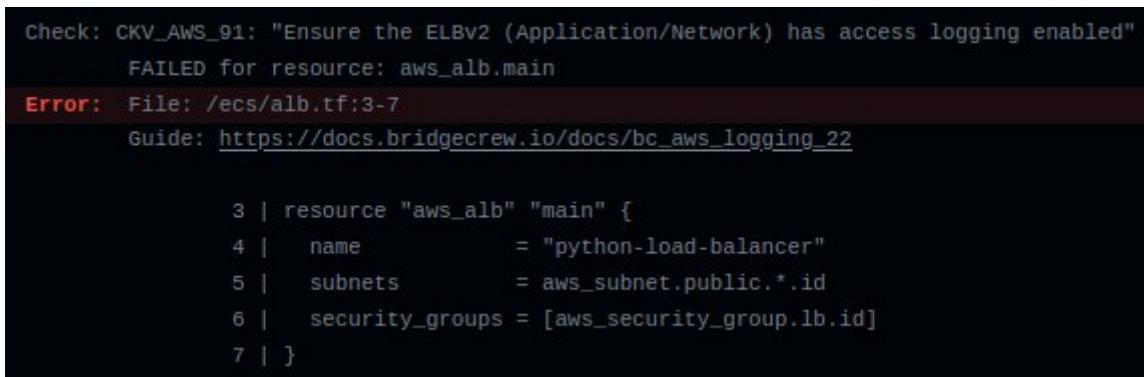
Zastosowanie narzędzia Checkov do zwiększenia bezpieczeństwa infrastruktury

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.

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.



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



Polityki narzędzi 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 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 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 | }
```

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”.

```
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 | }
```

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 szyfrowaniu, 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 | }
```

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 niepożdanego ruchu: AWS WAF umożliwia blokowanie niepożdanego 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

Table of Figures

- i <https://www.devsecops.org>
- ii <https://owasp.org/www-project-top-ten>
- iii <https://cwe.mitre.org/data/definitions/521.html>