**Politechnika Wrocławska**

**Wydział Informatyki i Telekomunikacji**

# Kierunek: Informatyczne Systemy Automatyki (ISA)

Specjalność: **Inteligentne systemy przemysłu 4.0 (ISP)**

## PRACA DYPLOMOWA

## MAGISTERSKA

**Porównanie wydajności systemów uruchamianych w kontenerach oraz natywnie**

Grzegorz Szuper

Opiekun pracy

**Dr inż. Jerzy Ramert-Greblicki**

Słowa kluczowe: wirtualizacja, konteneryzacja, Docker, Ec2, Ansible

### WROCŁAW 2025

Streszczenie

Tekst streszczenia po polsku

Abstract

Treść streszczenia po angielsku

**Spis treści**

[Kierunek: Informatyczne Systemy Automatyki (ISA) 1](#_Toc202291457)

[PRACA DYPLOMOWA 1](#_Toc202291458)

[MAGISTERSKA 1](#_Toc202291459)

[WROCŁAW 2025 1](#_Toc202291460)

[Streszczenie 2](#_Toc202291461)

[Abstract 2](#_Toc202291462)

[Wstęp 1](#_Toc202291463)

[Cel i motywacja 1](#_Toc202291464)

[Zakres badań 1](#_Toc202291465)

[Struktura pracy 1](#_Toc202291466)

[1. Podstawy teoretyczne 1](#_Toc202291467)

[1.1. Konteneryzacja vs wirtualizacja 1](#_Toc202291468)

[1.2. Mechanizmy izolacji (namespaces, cgroups) 1](#_Toc202291469)

[1.3. Docker i alternatywy (Podman, LXC) 2](#_Toc202291470)

[1.4. Infrastruktura jako kod (Terraform, Ansible, CI/CD) 2](#_Toc202291471)

[1.5. Chmura AWS – model IaaS (EC2, S3) 2](#_Toc202291472)

[2. Środowisko I narzędzia badawcze 2](#_Toc202291473)

[2.1. Konfiguracja platformy chmurowej 2](#_Toc202291474)

[2.2. Automatyzacja infrastruktury – Terraform 2](#_Toc202291475)

[2.3. Automatyzacja konfiguracji – Ansible 2](#_Toc202291476)

[2.4. Konfiguracje uruchomieniowe 2](#_Toc202291477)

[2.5. Narzędzia pomiarowe 2](#_Toc202291478)

[2.6. Automatyzacja CI/CD z wykorzystaniem GitHub 2](#_Toc202291479)

[3. Metodyka eksperymentów 2](#_Toc202291480)

[3.1. Założenia i hipotezy badawcze 2](#_Toc202291481)

[3.2. Scenariusze testowe 2](#_Toc202291482)

[3.2.1. CPU-bound – mnożenie macierzy (Python + NumPy) 2](#_Toc202291483)

[3.2.2. Baza danych – PostgreSQL + pgbench 2](#_Toc202291484)

[3.2.3. HTTP – Nginx + wrk 2](#_Toc202291485)

[3.2.4. I/O – fio (random/sequential) 2](#_Toc202291486)

[3.2.5. Memory – stress-ng / custom script 2](#_Toc202291487)

[3.3. Procedura wykonania testów 2](#_Toc202291488)

[3.4. Metryki i sposób analizy 2](#_Toc202291489)

[4. Wyniki eksperymentów 3](#_Toc202291490)

[4.1. Wyniki Cpu-bound 3](#_Toc202291491)

[4.2. Wyniki PostgresSQL 3](#_Toc202291492)

[4.3. Wyniki HTTP 3](#_Toc202291493)

[4.4. Wyniki I/O 3](#_Toc202291494)

[4.5. Wyniki Memory 3](#_Toc202291495)

[4.6. Zestawienie zbiorcze 3](#_Toc202291496)

[5. Analiza i dyskusje 3](#_Toc202291497)

[5.1. Porównanie natywnie vs kontener – omównie różnic 3](#_Toc202291498)

[5.2. Czyniki wpływające (OverlaFS, cgroups, warstwa sieciowa) 3](#_Toc202291499)

[5.3. Ogranieczenia badań (typ instalacji, wariancja chmury, liczba prób) 3](#_Toc202291500)

[6. Wnioski i dalsze kierunki badań 3](#_Toc202291501)

[6.1. Kluczowe wnioski 3](#_Toc202291502)

[6.2. Rekomendacje wdrożeniowe 3](#_Toc202291503)

[6.3. Pomysły na rozszerzenia (Kubernetes, różne fs, inne chmury) 3](#_Toc202291504)

[Zakończenie 3](#_Toc202291505)

[Bibliografia 3](#_Toc202291506)

[Spis rysunków 3](#_Toc202291507)

[Spis tabel 3](#_Toc202291508)

[Załącznik 3](#_Toc202291509)

Wstęp

Cel i motywacja

Zakres badań

Struktura pracy

# Podstawy teoretyczne

## Konteneryzacja vs wirtualizacja

Podejście do uruchamiania aplikacji uległo dużym przemianom – zaczynając od dedykowanych serwerów fizycznych, przez wirtualizacje, aż po współczesne mechanizmy konteneryzacyjne. Każde z tych podejść ma swoje zalety, ograniczenia oraz konkretne zastosowania.

Wirtualizacja polega na tworzeniu logicznych maszyn wirtualnych z wykorzystaniem hyperwizora, które znajdują się na jednym fizycznym hoście. Każda maszyna posiada oddzielny pełny system operacyjny, co zapewnia wysoki poziom izolacji, ale wiąże się z istotnym zużyciem zasobów takich jak procesor oraz pamięć operacyjna. Z uwagi na wczytywanie systemu operacyjnego maszyny wirtualne uruchamiają się powoli, a ich przenoszenie jest problematyczne.

Konteneryzacja jest modelem podobnym do maszyn wirtualnych, ale reprezenuje lżejsze podejście izolowania aplikacji na poziomie procesów. Aplikacje są uruchamiane z wykorzystaniem kontenerów, które współdzielą jądro systemu operacyjnego hosta i korzystają z mechanizmów namespaces i cgroups do izolowania, monitorowania oraz ograniczania zasobów systemowych wykorzystywanych przez każdy kontener. Zawierają jedynie aplikacje oraz jej zależności, a przez brak osobnego systemu operacyjnego, uruchamiają sie o wiele szybciej niż maszyny wirtualne oraz zużywają mniej zasobów. Z uwagi na swój mały rozmiar oraz szybkość uruchomienia mają bardzo dobrą skalowalność oraz prostotę podczas migracji miedzy środowiskami.

Podsumowując główna różnica między maszynami wirtualnymi a kontenerami jest na poziomie izolacji i narzutach systemowych. Wirtualne maszyny stusują izolacje aplikacji na poziomie systemu operacyjnego, a kontenery na poziomie procesów, dzięki czemu lepiej sprawdzają sie w sytuacjach w których zależy nam na szybkim uruchomieniu aplikacji oraz łatwości w przenoszeniu jej. Aktualnie kontenery nie wyparły maszyn wirtualnych, lecz uzupełniają je, ponieważ mogą być uruchamiane zarówno na fizycznych maszynach, jak i wirtualnych. Takie podejście jest wykorzystywane w platformach chmurowych np. AWS, gdzie kontenery są uruchamiane na instancjach EC2, które w rzeczywistości są maszynami wirtualnymi.

## Mechanizmy izolacji (namespaces, cgroups)

Jednym z kluczowych elementów umożliwiających działanie konteneryzacji w systemach uniksowych są wbudowane w jądro systemu Linux mechanizmy izolacji. Najważniejsze z nich to namespaces oraz control groups (cgroups), za pomocą których można stworzyć odizolowane środowisko wykonawcze, jednocześnie współdzieląc zasoby systemowe hosta.

Namespaces jest odpowiedzialne za logiczne oddzielenie potrzebnych aspektów systemu operacyjnego jak np. identyfikatory użytkowników, przestrzeń plików, sieci oraz procesów. Dzięki temu procesy, które działają w kontenerze widzą tylko swoją przestrzeń i nie mają dostępu procesów, systemów plików, interfejsów sieciowych innych kontenerów oraz hosta.

Control groups (cgroups) odpowiada za kontrole oraz ograniczanie wykorzystania zasobów systemowych przez procesy. Za pomocą jego można ustalić limity wykorzystywania przez kontener procesora, pamięci operacyjnej czy sieci, a także monitorowac aktualne zużycie tych zasobów.

Namespaces oraz cgroups są nieodzowną częścią konteneryzacji i dzięki nim uzyskujemy możliwość tworzenia w pełni odizolowanych środowisk, bez potrzeby stosowania maszyn wirtualnych. Z punktu widzenia aplikacji ich działanie jest niewidoczne i nie wymaga żadnych dodatkowych modyfikacji, co stanowi jedną z głównych zalet wykorzystania tego modelu

## Docker i alternatywy (Podman, LXC)

Docker to najpopularniejsze narzędzie do tworzenia, uruchamiania oraz zarządzania kontenerami. Zostało stworzone przez firmę Docker, inc w 2013 roku. Z uwagi na swoją innowacje narzędzie od razu zrewolucjonizowało podejście do konteneryzacji czyniąc je ogólno dostępne nie tylko dla inżynierów, ale także dla aministratorów, programistów i zwykłych użytkowników.

Głównym zadaniem Dockera jest uruchamianie aplikacji w kontenerach, czyli izolowanych środowiskach procesów z wykorzystaniem technologi wbudowanych w jądro Linux, takich jak namespaces oraz cgroups. Zanim pojawił się Docker konteneryzacją zajmowały się tylko duże firmy, a jej przeprowadzenie wymagało dużej wiedzy o systemie Linux. Docker wprowadzil intuicyjny interfejs, zintegrował narzędzia do budowania, wersjonowania oraz publikacji obrazów i kontenerów oraz uprościł ich zarządzanie.

Główne komponenty z jakich składa się Docker to:

- Docker Engine (dockerd) – główny demon odpowiedzialny za zarządzanie kontenerami, sieciami, wolumenami oraz obrazami. Demon (ang. deamon) to program który działa w tle systemu operacyjnego i odpowiada za wykonywanie zadań niezależnie od interfejsu użytkownika

- containerd – demon pośredni, który zarządza cyklem życia kontenerów. Odpowiada za ich tworzenie, wstrzymanie ich działania, uruchamianie czy usuwanie. Jest to projekt open source rozwijany przez Cloud Native Computing Foundation (CNCF) – organizacje non-profit wspierającą rozwój technologi konteneryzacyjnych i chmurowych.

- runc – niskopoziomowe narzędzie CLI, które tworzy i uruchamiakontenery na podstawie specyfikacji OCI (Open Container Initiative). OCI jest to standard definiujący format obrazów, kontenerów oraz sposób ich uruchamiania między różnymi narzędziami.

- Docker CLI (docker) – interfejs lini poleceń, który komunikuje się z głównym demonem (dockerd) i umożliwia użytkownikowi wykonywanie operacji na kontenerach w sposób prosty i spójny.

Pod względem architektury Docker stosuje podział warstwowy, w którym CLI komunikuje się z demonem (dockerd), ten odsyła zadania do contaired, a finalnie uruchomienie kontenera odbywa się przez runc. Jest to w pełni zgodne z zaleceniami specyfikacji OCI, która standaryzuje między innymi format obrazów oraz sposób ich uruchamiania.

Współczesne wersje Dockera wspierają szereg dodatkowych funkcjonalności takich jak:

- budowanie obrazów kontenerów za pomocą plików Dockerfile, które opisują sposób utworzenia kontenera dla aplikacji krok po kroku

- orkiestracja kontenerów przy użyciu narzędzia Docker Swarm, za pomocą którego można zarządzać wieloma kontenerami, znajdującymi się na różnych hostach, w tym ich uruchamianie, skalowanie oraz aktualizajce. Jest to szczególnie istotne w środowiskach chmurowych i mikroserwisach, gdzie aplikacja może zawierać w sobie wiele współpracujących usług.

- zarządzanie sieciami i wolumenami, co umożliwia zdefiniowanie sposobu komunikacji między kontenerami oraz trwałe przechowywanie danych niezależnie od cyklu życia kontenera

-integracja z rejestrami obrazów kontenera takimi jak Docker Hub, który jest publicznym rejestrem kontenerów firmy Docker i pozwala na przechowywanie, udostępnianie i pobieranie gotowych obrazów oraz Amazon Elastic Container Registry (ECR) od firmy Amazon realizoway w usłudze AWS który jest zintegrowany z narzędziami takimi jak Amazon ECS oraz EKS, które służą do zarządzania i tworzenia kontenerów na platformie chmurowej AWS.

Docker ze względu na swoją powszechność w środowiskach produkcyjnych, szeroką dostepność materiałów technicznych i bardzo dobrą integracje z innymi narzędziami stosowanymi aktualnie w kontekście konteneryzacji jest najczęściej wybieranym rozwiązaniem, jednak istnieją również inny narzędzia, które zyskały popularnośc w określonych zastosowaniach. Przykładem jest Podman – rozwijany przez firme Red Hat, który umożliwia konteneryzacje bez potrzeby działania demona w tle. Inną alternatywą jest LXC (Linux Containers), które jest niskopoziomowym podejściem, opierającym się bezpośrednio na mechanizmach jądra Linux. Umożliwia ono uruchamianie kontenerów z własnym systemem init/systemd i odwzorowuje klasyczne środowisko systemu operacyjnego w sposób bardziej zbliżony do maszyn wirtualnych.

## Infrastruktura jako kod (Terraform, Ansible, CI/CD)

Koncepcja infrastruktury jako kod (infrastructure as Code, IaC) pojawiła wraz z wzrostem skomplikowania środowisk IT, popularności wykorzystywania chmury obliczeniowej oraz potrzeby automatyzacji zarządzania infrastrukturą. Zakłada ona zarządzanie zasobami infrastrukturalnymi za pomocą plików tekstowych, przeważnie w jezykach deklaratywnych lub skryptowych. Dzięki temu deklarowanie zasobów, ich tworzenie oraz odtwarzanie staje się powtarzalne oraz zautomatyzowane. Pozwala to wyeliminować ręczną konfiguracje każdego urządzenia i zminimalizować potencjalne błędy, które mogły by wystapić podczas instalacji zasobów oraz bardzo ułatwia szybkie skalowanie środowisk i testowanie zmian przed ich wdrożeniem.

Terraform jest to narzędzie typu open source, rozwijane przez firmę HashiCorp, które pozwala na definiowanie oraz zarządzanie infrastrukturą w sposób deklaratywny. Jest to obsługiwane poprzez pliki konfiguracyjne (.tf) w języku HCL (HashiCorp Configuration Language), które opisują żądane zasoby, takie jak np. maszyny wirtualne, serwery, bazy danych czy urządzenia sieciowe. Po napisaniu naszego pliku konfiguracyjnego, następuje zbudowanie naszego planu, terraform porównuje aktualną infrastrukture oraz stan pożądany i po zatwierdzeniu przez użytkownika proponowanych zmian, wykonuje tylko te działania, które są potrzebne do osiągnięcia zgodności. Terraform obsługuje wiele dostawców chmurowych oraz infrastrukturalnych, najpopularniejsi z nich to AWS, Azure, Google Cloud oraz VMware oraz Kubernetes, co umożliwia tworzenie środowisk z wykorzystaniem różnych platform za pomocą jednej technologii.

Ansible to narzędzie, które pozwala na automatyzajce i konfiguracje systemów oraz instalację oprogramowania i zarządzanie usługami. Jest rozwijane przez firmę Red Hat i opiera się na prostych oraz intuicyjnych plikach w formacie YAML. Plik konfiguracyjny (tzw. Playbook), określa jakie operacje mają zostać wykonane na danym serwerze lub grupie serwerów. Narzędzie jest głównie wykorzystywane do konfigurowania infrastruktury, w odróżnieniu do Terraform, którego zadaniem jest jej buduwanie. Przykładem typowych zadań do jakich możemy wykorzystać Ansible jest instalowanie aktualizacji, oprogramowania na wielu maszynach jednocześnie czy wdrażanie aplikacji oraz konfiguracja bazy danych. Działa on w sposób agentless, co oznacza, że nie jest konieczne instalowanie dodatkowego oprogramowania na maszynach, na których planujemy pracować za jego pomocą. Do komunikacji z docelowymi maszynami używa połączenia poprzez SSH lub w przypadku systemu Windows WinRm.

CI/CD (Continuous Integration / Continuous Deployment) jest to praktyka polegająca na ciągłej integracji oraz ciągłym wdrażaniu kodu i bardzo często współistnieje z tworzeniem oprogramowania w modelu infrasturktura jako kod. Jej celem jest pełna automatyzajca procesu tworzenia, testowania i wdrażania aplikacji. CI oznacza ciągłe, regularne scalanie zmian w kodzie go głównego repozytorium i automatyczne testowanie jego poprawności, tak aby na wczesnym etapie wykryć potencjalne błędy. CD natomiast odnosi się do wdrażania przetestowanego kodu na środowiska testowe lub produkcyjne w sposób automatyczny. Często wykorzystuje się tak zwane pipepline’y CI/CD, które są łańcuchem automatycznych kroków, które są wykonywane po wykryciu zmian w repozytorium. Przykładem pipeline’u może być pobranie kodu z repozytorium, uruchomienie testów jednostkowych czy wdrożenie aplikacji do środowiska testowego lub produkcyjnego. Każdy z etapów jest wykonywany automatycznie i według określonej kolejności. Przykładem popularnych narzędzi CI/CD jest GitHub Actions, Gitlab CI oraz Jenkins, które w bardzo przystępny sposób umożliwiają integracje kodu aplikacji z kodem infrastruktury, jednocześnie wspierając automatyzacje i zarządzanie środowiskiem, w którym jest ona uruchamiana.

## Chmura AWS – EC2, Security Groups, S3, IAM Roles

Amazon Web Services (AWS) to największy na świecie dostawca usług chmurowych, oferujący szeroki zakres narzędzi i rozwiązań – od maszyn wirtualnych i serwerów, przez bazy danych i urządzenia IoT, aż po usługi z obszaru sztucznej inteligencji (AI) i DevOps. AWS działa w modelu as-a-Service, co oznacza, że użytkownik może samodzielnie decydować, które warstwy środowiska IT chce obsługiwać samodzielnie, a które powierzyć dostawcy chmurowemu. Wyróżniamy trzy podstawowe modele usług chmurowych:

- IaaS (Infrastructure as a Service) – oznacza to, że użytkownik korzysta w chmurze głównie z maszyn wirtualnych, sieci, dysków, bez systemu operacyjnego i aplikacji.

- PaaS (Platform as a Service) – w tym modelu udostępniane są również środowiska uruchomieniowe, bazy danych oraz frameworki. Użytkownik poza chmurą skupia się głównie na aplikacji i jej danych.

- SaaS (Software as a Service) – to model chmury, w którym cała odpowiedzialność za uruchomianie, utrzymanie i aktualizację aplikacji leży po stronie dostawcy usług. Użytkownik otrzymuje gotowy system dostępny przez przeglądarkę lub interfejs API bez potrzeby instalacji, konfiguracji ani zarządzania infrastrukturą.

Amazon Elastic Compute Cloud (EC2) to usługa chmurowa, umożliwiająca uruchamianie i zarządzanie maszynami wirtualnymi, tak zwanymi instancjami EC2. Oprócz samego środowiska, EC2 zapewnia również zarządzanie wirtualnymi dyskami (Amazon EBS), grupami zabezpieczeń, adresami publicznymi oraz metadanymi instancji. Podczas tworzenia instancji, użytkownik sam decyduje o konfiguracji pod względem liczby rdzeni procesora, pamięci operacyjnej, rodzaju dysku, systemie operacyjnym czy regionu geograficznego, w którym instancja zostanie utworzona. Większość instancji EC2 stosuje technologię wirtualizacji Nitro – czyli łączy elementy sprzętowe z oprogramowaniem odpowiedzialnym za zarządzanie maszynami wirtualnymi. Zapewnia również minimalny narzut systemowy oraz wysoki poziom izolacji między klientami. AWS udostępnia również specjalne instancje „bare-metal”, które pomijają etap wirtualizacji i zapewniają bezpośredni dostęp do fizycznych zasobów serwera, bez warstwy hyperwizora. Jest to rozwiązanie, które umożliwia posiadanie pełnej kontroli nad sprzętem, ale ze względu na koszt i ograniczoną dostępność, jest rzadziej stosowane w projektach edukacyjnych i badawczych.

Dostęp sieciowy do instancji EC2 jest kontrolowany za pomocą Security Groups, które są logiczną zaporą ogniową w postaci wirtualnego interfejsu sieciowego. Dzięki nim, można zdefiniować reguły ruchu sieciowego przychodzącego oraz wychodzącego a następnie przypisać je do np. konkretnych instancji czy sieci prywatnych. Bezpieczeństwo każdego systemu chmurowego wymaga, dobrze skonfigurowanych Security Groups, ponieważ nie zabezpieczenie lub zbyt szerokie reguly mogły by doprowadzić do podatności na ataki sieciowe.

Amazon Simple Storage Service (S3) to usługa trwałego i skalowalnego przechowywania obiektów. Dane są przechowywane, w tak zwanych bucketach i identyfikowane unikalnymi kluczami. Usługa zapewnia wysoką trwałość danych i dostęp w skali globalnej. S3 jest najczęściej wykorzystywane jako repozytoria testów, statycznych plików oraz backupów.

IAM Roles (IAM - Identity and Access Management) jest to mechanizm przypisywania ról do instancji EC2. Role umożliwiają instancją dostęp do wybranych usług AWS bez konieczności przechowywania klczy dostępowych na maszynie. Przykładowo instancja z przypisaną rolą IAM może bezpieczne przesyłąć dane do S3 lub jeśli ma zainstalowany AWS System Manager Agent (SSM) oraz przypisaną odpowiednią rolę, możliwy jest dostęp do niej bez wykorzystania portu SSH ani korzystania z klucza dostępu.

# Środowisko badawcze

## Środowisko kontenerowe (Docker)

W ramach części eksperymentalnej zastosowano platformę Docker do uruchamiania aplikacji w izolowanych, lekkich kontenerach. Wybór tego narzędzia wynikał z jego powszechności w środowiskach produkcyjnych, szerokiej dostepności materiałów technicznych i bardzo dobrej integracji z innymi narzędziami wykorzystywanym w projekcie, takim jak Terraform, Ansible oraz GitHub.

Kontenery były uruchamiane na instancjach EC2 z systemem Amazon Linux 2, a ich konfiguracja została w pełni zautomatyzowana za pomocą narzędzia Ansible. W związku z problemami podczas lokalnego uruchamiania Ansible na systemie Windows, które były spowodowane brakiem bibliotek oraz wsparcia dla SSH, również wykorzystano kontenery z obrazem williamyeh/ansible:alpine3. Jest to lekki obraz bazujący na systemie Alpine Linux, który zawiera w sobie pełne, gotowe środowisko do korzystania z Ansible, co umożliwiło wygodne korzystanie z playbooków w systemie Windows, bez potrzeby instalowania zależności lokalnie.

Do celów testowych każda instancja EC2, posiadała dedykowany kontener, dopasowany do typu przeprowadzanego testu:

- kontener benchmark (CPU + RAM) został wykorzystany do przeprowadzenia testów zużycia zasobów systemowych podczas wykonywania obliczeń arytmetycznych. Jest on oparty na obrazie amazonlinux:2. Za jego pomocą, wewnątrz kontenera uruchamiany był autorski skrypt benchmark.py, który wykonywał mnożenie macierzy i rejestrował czas wykonania oraz zużycie zasobów, takich jak procesor i pamięc RAM. Wyniki testów były zapisywane do pliku, a następnie przesyłane do zasobnika Amazon S3.

- kontener Postgres (baza danych) użyto do przeprowadzenia pomiaru wydajności operacji bazodanowych, korzystając z oficjalnego kontenera postgres:13. Został uruchomiony z podstawową konfiguracją środowiskową oraz mapowaniem portu 5433.

- dopracować ten kawałek ! I/O

- kontener nginx posłużył do wykonania testów wydajnościowych HTTP. Serwował on statyczną stronę testową, która znajdowała się w lokalnym katalogu – zamontowanym do domyślnego katalogu serwera. Serwer został wystawiony na porcie 80. Obciążenie generowano z poziomu osobnej instancji-klienta przy użyciu narzędzia wrk.

## Konfiguracja platformy chmurowej

W ramach środowiska testowego wykorzystano platformę Amazon Web Service (AWS) działającą w modelu IaaS (infrastructure as a Service). Operacje były uruchamiane na instancjach EC2, które stanowiły bazę dla kontenerowych oraz natywnych wersji testów. Instancje były zarządzane z wykorzystaniem narzędzia Terraform, które umożliwiło definiowanie infrastruktury w plikach konfiguracyjnych HCL. Dzięki temu udało się zachować pełną powtarzalność podczas uruchamianych testów, a konfiguracja była przejrzysta oraz skalowalna. Wszystkie instancje zostały stworzone w regionie eu-west-3a (Paris). Wybór tego regionu wynikał z jego geograficznej bliskości, co przełożyło się na niższe opóźnienia sieciowe oraz większą responywność usług w czasie rzeczywistym. Do dostępu SSH, który odpowiadał za łączenie się z instancjami, bez konieczności stosowania haseł, wykorzystana utworzoną parę kluczy o nazwie thesis-key.

### Rodzaje i konfiguracja instancji EC2

### Dostęp i bezpieczeństwo

## Automatyzacja infrastruktury – Terraform

## Automatyzacja konfiguracji – Ansible

## Konfiguracje uruchomieniowe

## Narzędzia pomiarowe

## Automatyzacja CI/CD z wykorzystaniem GitHub

W ramach realizacji projektu utworzone dedykowane repozytorium o nazwie „Praca-magisterska” na platformie GitHub, które odpowiadało za centralne miejsce zarządzania kodem, konfiguracją infrastruktury oraz skryptami automatyzującymi testy. Cała struktura została podzielona na trzy główne katalogi: infra, ansible oraz tests. Każdy z nich odpowiada za inny etap realizacji projektu, od definiowania infrastruktury, przez automatyzacje konfiguracji, aż po uruchomienie testów wydajnościowych.

W katalogu infra/ zostały umieszczone pliki Terraform odpowiadające za definiowanie infrastuktury w chmurze AWS. Struktura została podzielona na podkatalogi, w których znajdują się pliki za pomocą których zostały utworzone instancje, wykorzystywane pod konkretne rodzaje testów:

* benchmark/ - instancje do tesów CPU i pamięci RAM
* http/ - środowisko do testów wydajniościowych serwera HTTP (nginx)
* io/ - definicje zasobów wykorzystanych w testach operacji dyskowych (fio)
* postgres/ - maszyny przygotowane do uruchomienia bazy danych PostgreSQL i przeprowadzenia testów zapytań

Dodatkowo folder infra/ zawiera pliki ogólne:

* main.tf, security.tf, iam.tf, keypair.tf, za pomocą któróch zostały zadeklarowanie definicje sieci, połaczenia z Terraform, uprawnień, kluczy i ról IAM,
* terraform.tfvars, variables.tf – w których zostały zapisane główne parametry infrastruktury
* thesis-key.pub – publiczny klucz SSH, używany do dostępu do instancji EC2

Folder ansible/ zawiera podkatalogi, w których zostały umieszczone zestawy playbooków oraz plików konfiguracyjnych, wykorzystanych do automatycznego konfigurowania środowiska testowego na utworzonych instancjach EC2. Struktura folderów została podzielona zgodnie z planem testów:

* benchmark/ - playbooki wykorzystane do instalacji zależności oraz uruchomienia kontenera testowego mierzącego użycie CPU oraz pamięci RAM za pomocą pliku benchmark.py
* http/ - zawiera pliki odpowiedzialne za przygotowanie środowiska do testów wydajnościowych HTTP
* io/ - zestaw playbooków za pomocą których wykonano testy operacji dyskowych, w środowisku natywnym oraz kontenerowym z użyciem narzędzia „fio”
* postgres/ - automatyzacja instalacji i uruchomienia instancji PostgreSQL w wersji natywnej i dockerowej, zawierająca konfiguracje środowiska i mapowanie portów

W każdym z katalogów tematycznych, zostal dodany plik: inventory.ini, w którym zdefiniowane są adresy IP instancji na których są wykonywane playbooki oraz klucz SSH (thesis-key.pem), wykorzystywany do zdalnego logowania się i wykonywania poleceń przez Ansible.

Folder tests/ - TUTAJ UZUPEŁNIC JAK POPRAWIE TESTY !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

# Metodyka eksperymentów

## Założenia i hipotezy badawcze

## Procedura wykonania testów

-opisać że aws sam w sobie nie jest natywny

## Scenariusze testowe

### CPU-bound – mnożenie macierzy (Python + NumPy)

### Baza danych – PostgreSQL + pgbench

### HTTP – Nginx + wrk

### I/O – fio (random/sequential)

### Memory – stress-ng / custom script

## Procedura wykonania testów

* Liczba powtórzeń, warm-up, zbieranie testów do s3

## Metryki i sposób analizy

- czas wykonania, wykorzystanie CPU/RAM, IOPS, RPS, latency

- statystyka: średnia, σ, 95 % CI, test t / Mann-Whitney

# Wyniki eksperymentów

## Wyniki Cpu-bound

## Wyniki PostgresSQL

## Wyniki HTTP

## Wyniki I/O

## Wyniki Memory

## Zestawienie zbiorcze

# Analiza i dyskusje

## Porównanie natywnie vs kontener – omównie różnic

## Czyniki wpływające (OverlaFS, cgroups, warstwa sieciowa)

## Ogranieczenia badań (typ instalacji, wariancja chmury, liczba prób)

# Wnioski i dalsze kierunki badań

## Kluczowe wnioski

## Rekomendacje wdrożeniowe

## Pomysły na rozszerzenia (Kubernetes, różne fs, inne chmury)

Zakończenie

Bibliografia

|  |  |
| --- | --- |
| [1] | J. Nowak, „The title of the article,” *Journal Name,* pp. 20-24, 2016. |
| [2] | P. Babington, The title of the book, Wydawca, 2008. |
| [3] | P. Eston, „The title of the chapter,” w *The title of the book*, City, Springer, 2009,  pp. 122-144. |
| [4] | „Nazwa strony,” 2017. [Online]. Available: www.strona.pl. |

Spis rysunków

Spis tabel

Załącznik