JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 413–420
Received: 7 September 2023
Accepted: 24 September 2023

Analysis of the impact of using containerization techniques on application performance in Python

Analiza wpływu wykorzystania technik konteneryzacji na wydajność aplikacji w języku Python

Kacper Chołody*, Sławomir Wojciech Przyłucki

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This article comprehensively evaluates the impact of two containerization environments, Docker and Podman, on the performance of Python applications. The paper characterizes the two tools and presents the differences in their architectures. The scope of the study covers three aspects. The first is a comparison of resource usage, such as CPU usage, RAM usage and execution time, during the calculation of the number π. The next step is to analyse the resource usage when sorting an ordered list. The final aspect of the research is a comparison of the start-up time of the container in both environments. The tests carried out allow the presence of a performance overhead in both containerization environments, with an average of 8%. In addition, it can be seen that there is better resource management in the case of the Podman tool and a more dynamic environment in the case of the Docker tool.
Keywords: containerization; performance comparison; Docker; Podman

Streszczenie

W niniejszym artykule dokonano kompleksowej oceny wpływu dwóch środowisk konteneryzacji, Dockera i Podmana, na wydajność aplikacji w języku Python. W pracy dokonano charakterystyki obu narzędzi oraz prezentacji różnic w ich architekturze. Zakres badań obejmuje trzy aspekty. Pierwszym z nich jest porównanie użycia zasobów, takich jak użycie procesora, pamięci RAM i czasu wykonania, w trakcie obliczeń liczby π. Kolejnym etapem jest analiza użycia zasobów podczas sortowania uporządkowanej listy. Ostatnim aspektem badań jest porównanie czasu startu kontenera w obu środowiskach. Przeprowadzone badania pozwalają na stwierdzenie występowania narzutu wydajności w obu środowiskach konteneryzacji, wynoszącego średnio 8%. Dodatkowo można zauważyć lepsze zarządzanie zasobami w przypadku narzędzia Podman oraz większą dynamikę środowiska w przypadku narzędzia Docker.

Słowa kluczowe: konteneryzacja; porównanie wydajności; Docker; Podman

*Corresponding author

Email address: kacper.cholody@pollub.edu.pl (K. Chołody)

1. Wstęp

Współczesna informatyka, zwłaszcza w kontekście wdrażania aplikacji i zarządzania nimi, znalazła się w epoce rewolucji. W miarę jak organizacje przechodzą od tradycyjnych, monolitycznych aplikacji w kierunku mikrousług i architektur opartych na kontenerach, konteneryzacja stała się jednym z kluczowych elementów tego przełomu. Jest to technologia, która pozwala na izolację aplikacji i ich zależności od infrastruktury, co umożliwia jednolite i niezawodne wdrażanie aplikacji na różnych platformach, niezależnie od systemu operacyjnego i środowiska.

Konteneryzacja umożliwia opakowanie aplikacji i wszystkich jej zależności, takich jak biblioteki, narzędzia i konfiguracje, w jednostkę zwaną kontenerem, która jest niezależna od środowiska, na którym jest uruchamiana. Dzięki temu możliwe jest uniknięcie problemów związanych z różnicami między środowiskami deweloperskimi, testowymi a produkcyjnymi, co jest częstym źródłem błędów podczas wdrażania aplikacji.

W tym kontekście narzędzia do konteneryzacji, takie jak Docker i Podman, odgrywają niezwykle ważną rolę. Są to platformy, które umożliwiają tworzenie, zarządzanie i uruchamianie kontenerów w sposób uproszczony. Dzięki nim programiści mogą pakować swoje aplikacje w kontenery i mieć pewność, że będą one działać tak samo na różnych maszynach i w różnych środowiskach. To ogromnie przyspiesza proces wdrażania aplikacji i pozwala zwiększyć niezawodność systemów.

Środowisko Docker, którego historia sięga 2013 roku, było jednym z pierwszych narzędzi do konteneryzacji, które zdobyło szeroką popularność. Jego prostota użycia i wsparcie dla kontenerów przyczyniły się do jego dominacji na rynku przez wiele lat.

Środowisko Podman natomiast, pomimo że młodsze, zdobyło znaczną popularność dzięki swojej elastyczności i filozofii "rootless", która pozwala uruchamiać kontenery bez konieczności uprawnień administratora. To narzędzie jest rozwijane jako część projektu OCI (ang. Open Container Initiative) i stanowi alternatywę dla narzędzia Docker, oferującą podobne możliwości, ale w bardziej modułowej i elastycznej formie.

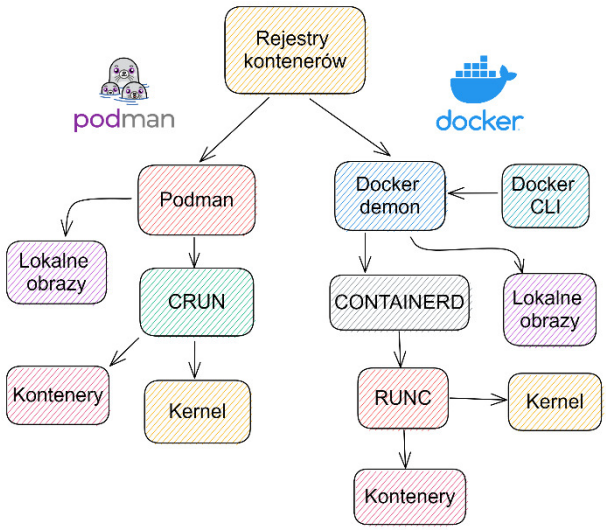
Wspólną cechą środowiska Docker i Podman jest to, że oba narzędzia mają ogromny wpływ na sposób, w jaki deweloperzy i administratorzy wdrażają

i zarządzają aplikacjami. Dzięki nim procesy te stają się bardziej skalowalne, niezawodne i bardziej zautomatyzowane, co ma kluczowe znaczenie w dzisiejszym dynamicznym świecie technologii informatycznych. Oba narzędzia również przyczyniły się do popularyzacji konteneryzacji jako standardowego podejścia do dostarczania i utrzymania oprogramowania, co stanowi kluczowy element transformacji cyfrowej organizacji na całym świecie.

2. Środowisko wirtualizacji Podman oraz Docker

Środowisko Docker jest jednym z najpopularniejszych narzędzi do konteneryzacji, ale przez ostatnie kilka lat mocno na popularności zyskały inne narzędzia, między innymi Podman. Podman jest to narzędzie do konteneryzacji i zarządzania kontenerami, które powstało jako alternatywa dla narzędzia Docker, oferując jednocześnie nowe podejście i szereg zalet w porównaniu do środowiska Docker, jednak jako młodsze nie jest aż tak rozbudowane, jak Docker.

Pomimo realizacji tych samych zadań, środowiska Docker i Podman różnią się znacząco pomiędzy sobą. Rysunek 1 przedstawia poglądową strukturę obu porównywanych środowisk konteneryzacji wraz z najważniejszymi wykorzystywanymi zależnościami.



Rysunek 1: Różnice w architekturze narzędzi Docker i Podman.

Architektura platform Docker i Podman różni się znacząco pod wieloma względami, wpływając na sposób, w jaki zarządzają i izolują kontenery oraz interakcje z nimi. Poniżej przedstawione są podstawowe różnice pomiędzy obydwiema architekturami:

- Architektura Docker [1]:
  - Demon i klient: Narzędzie Docker działa w oparciu o architekturę klient-serwer. Głównym komponentem jest Docker demon, który zarządza kontenerami, obrazami, sieciami itp. Klienci Docker komunikują się z demonem poprzez interfejs komend wiersza poleceń lub API REST.
  - Centralne zarządzanie: W środowisku Docker wszystkie operacje na kontenerach wykonywane są poprzez centralny demon. Wszystkie kontenery działające na danym hoście są zarządzane przez tego samego demona.
  - Jeden punkt awarii: Ponieważ Docker demon pełni kluczową funkcję w zarządzaniu kontenerami, awaria demona może wpłynąć na działanie wszystkich kontenerów na danym hoście.
- Architektura Podman [2]:
  - Bez demonów: W przeciwieństwie do środowiska Docker, Podman nie wymaga centralnego demona. Każdy kontener jest zarządzany przez oddzielny proces w przestrzeni użytkownika. Każdy kontener działa jako niezależny proces bez centralnego punktu zarządzania.
  - Zdecentralizowana architektura: Każdy kontener w środowisku Podman jest wykonywany jako osobny proces w przestrzeni użytkownika, co eliminuje potrzebę posiadania centralnego demona. To sprawia, że kontenery są bardziej izolowane
    i nie ma jednego punktu awarii dla wszystkich kontenerów.
  - Bez uprawnień administratora: W narzędziu Podman kontenery mogą działać w trybie "rootless", co oznacza, że nie wymagają uprawnień administratora. Każdy kontener jest wyizolowanym środowiskiem i może być zarządzany bez konieczności posiadania specjalnych uprawnień.
  - Mniejsza komunikacja: Bez potrzeby komunikacji z centralnym demonem, narzędzie Podman nie generuje dodatkowego ruchu sieciowego między klientem a demonem.
  - Większa izolacja: Ze względu na brak centralnego punktu zarządzania środowisko Podman dostarcza większej izolacji pomiędzy kontenerami.

3. Przegląd literatury

3.1. Porównanie wykorzystania procesora i pamięci RAM

W pracy [3] przedstawione jest porównanie wydajności dwóch konkurujących ze sobą rozwiązań dostarczających wirtualizację opartą o kontenery, to jest środowiska Docker i Podman. W tych badaniach porównanie wydajności opiera się na pomiarach użycia zasobów procesora, wykorzystania pamięci RAM oraz prędkości zapisu i odczytu danych z dysku. Do realizacji przeprowadzonych testów autor użył popularnych i dobrze znanych narzędzi do generowania syntetycznego obciążenia. W celu obciążenia procesora został wykorzystany Y-cruncher - program obliczający wartość liczby π przy użyciu wielu wątków. Na podstawie przedstawionych w pracy wyników można stwierdzić, że wykorzystanie konteneryzacji powoduje wymierny narzut na czas wykonania obliczeń dla liczby π, zarówno w przypadku narzędzia Docker, jak i Podman, a różnica czasu wykonania wynosi około 10%, w porównaniu do scenariusza testowego, w którym nie użyto rozwiązań wykorzystujących konteneryzację.

W badaniach przeprowadzonych przez autorów pracy [4] skupiono się między innymi na porównaniu stopnia wykorzystania procesora i pamięci RAM przez

systemy lekkiej wirtualizacji, jak i wykorzystujące nadzorców wirtualizacji (ang. hipervisor). Z przedstawionych wyników można wywnioskować, że biorąc za punkt odniesienia rozwiązanie niewykorzystujące wirtualizacji, środowiska lekkiej wirtualizacji opartej na kontenerach, takie jak Docker i Podman, oferują wydajność na bardzo zbliżonym poziomie. Różnice zawierają się w przedziale od 0,1% do 3% na korzyść lub niekorzyść tych rozwiązań w zależności od użytego benchmarku. Podman oferuje minimalnie lepszą wydajność w zastosowaniach, w których obciążenie jest wysokie. Różnica w tych zastosowaniach w porównaniu do narzędzia Docker wynosi około 2 – 3%.

W pracy [5] autor do pomiarów wydajności użył narzędzia „sysbench", w połączeniu z pomiarem czasu potrzebnego na odpowiedź. Otrzymane wyniki są zbieżne z wynikami badań opisanymi wcześniej i nie wskazują, aby wydajność rozwiązań wykorzystujących wirtualizację opartą o kontenery odbiegała w zauważalnym stopniu od innych sposobów uruchamiania aplikacji.

W [6] autorzy zbadali, jaki narzut na zasoby systemowe powoduje wykorzystanie środowiska Docker. Otrzymane przez nich wyniki stanowią ciekawy aspekt badań w omawianym obszarze, ponieważ wykazały one, że narzut rozwiązania Docker na wielkość wykorzystania zasobów jest tym większy, im mniejsze jest wykonywane zadanie. Narzut w przypadku zadań wymagających niewielkiej części z puli ogólnie dostępnych zasobów systemowych wynosi prawie 10% całkowitego użycia procesora i maleje do około 3% w przypadku zadań, które już wymagają dużo większej części z dostępnych zasobów.

Na podstawie wyników badań zaprezentowanych przez autorów pracy [7] wynika, że ogólna wydajność aplikacji uruchomionych w środowisku Docker jest dużo większa niż pełnej wirtualizacji w badaniu 7-zip polegającym na kompresji algorytmem LZMA pliku o rozmiarze 10 GB. Docker osiągnął wynik prawie dwukrotnie lepszy od maszyny wirtualnej, skracając czas kompresji z około 36 sekund do około 19 sekund. Dodatkowo liczba instrukcji na sekundę wykonywanych przez kontener uruchomiony w środowisku Docker jest średnio 4-krotnie większa, a w skrajnym przypadku niemal 8-krotnie większa od maszyny wirtualnej.

Podobne wyniki zostały zaprezentowane przez autorów pracy [8]. Jednak w tym przypadku uzyskane przez nich różnice w wydajności aplikacji uruchomionej w środowisku Docker oraz na maszynie wirtualnej są zdecydowanie bardziej zbliżone do siebie. Według opublikowanych wyników, w przypadku narzędzia Docker autorom udało się uzyskać wyniki wydajności lepsze o około 2%.

Wyniki testów przedstawionych w pracy [9], potwierdzają, że Docker ma tendencję do alokowania zbyt dużej ilości zasobów przy mniejszym obciążeniu. Autorzy w swojej pracy zbadali, że narzut rozwiązania Docker zmniejsza się wraz z liczbą uruchomionych kontenerów, przez co efektywne obciążenie procesora może praktycznie nie wzrosnąć przy większej liczbie kontenerów.

3.2. Czas uruchomienia kontenera

Czas potrzebny na uruchomienie kontenera jest jednym z istotnych czynników wpływających na ogólną ocenę wydajności danego rozwiązania lekkiej wirtualizacji. Temat ten został poruszony przez autora pracy [5], który wykorzystał zestaw kontenerów i wykonał pomiary czasu, który upłynął od wydania polecenia uruchomienia kontenera, do momentu otrzymania odpowiedzi z kontenera. Z otrzymanych wyników można wywnioskować, że czas potrzebny na uruchomienie kontenera przez narzędzia Docker i Podman jest przynajmniej 4 – 5 krotnie większy w porównaniu do innych rozwiązań.

Należy podkreślić, że w badaniach przeprowadzonych przez autorów pracy [10], w porównaniu do klasycznego podejścia z wykorzystaniem maszyn wirtualnych i pełnej wirtualizacji, czas potrzebny na uruchomienie aplikacji zbudowanej z wykorzystaniem kontenerów jest mniejszy o cały rząd wielkości i wynosi około 2 sekund, zamiast około 11 – 12 sekund w przypadku maszyn wirtualnych.

4. Metodyka i scenariusze badań

4.1. Metodyka badań

Celem pracy jest przeprowadzenie analizy porównawczej dla dwóch środowisk konteneryzacji, to jest Docker i Podman. Porównanie będzie dotyczyło takich parametrów jak czas wykonania operacji, średnie użycie procesora przez proces oraz średnie wykorzystanie pamięci operacyjnej komputera. Badania będą podzielone na scenariusze badawcze i będą wykonywane w sposób sekwencyjny z zachowaniem odpowiedniego czasu potrzebnego na ustabilizowanie temperatury komputera po poprzednim badaniu. W przypadku, gdy nie będzie można stwierdzić znaczących różnic między badanymi środowiskami dla konkretnych parametrów, dla których wystąpiło znaczne podobieństwo, badania zostaną powtórzone z pominięciem środowiska wirtualizacji, tak, aby zbadać ewentualny wpływ samej lekkiej wirtualizacji z pominięciem poszczególnych środowisk. Otrzymane wyniki zostaną szczegółowo omówione i zaprezentowane w formie tabel oraz wykresów wraz z wykonaniem podstawowej analizy statystycznej.

4.2. Scenariusze badań

4.2.1. Scenariusz S1 – testowanie obciążenia procesora przy wykorzystaniu obliczeń liczby π

W tym scenariuszu zostaną przeprowadzone badania z wykorzystaniem wielokrotnych obliczeń liczby π. przy wykorzystaniu formuły Leibniza. Formuła Leibniza [11] lub też szereg Leibniza może zostać wykorzystany do obliczeń przybliżonej wartości liczby π. Wzór Leibniza opiera się na szeregu nieskończonym, który może być sumowany do dowolnego stopnia dokładności. Wykorzystanie powyższych obliczeń może być skutecznym sposobem na generowanie stałego obciąże-

nia procesora, przy założeniu, że liczba wyrazów ciągu do obliczenia jest stosunkowo duża.

Parametrami, które będą mierzone w trakcie wykonywania obliczeń, będą:

- Czas wykonywania obliczeń.
- Średnie użycie procesora.
- Średnie użycie pamięci operacyjnej.

Do zmierzenia użycia procesora oraz wykorzystania pamięci RAM, zostały wykorzystany program *btop* oraz moduł *psutil* z języka Python, natomiast do pomiaru czasu wykorzystano moduły *time* oraz *pyperf*.

4.2.2. Scenariusz S2 – testowanie wykorzystania pamięci operacyjnej poprzez sortowanie uporządkowanej listy

W tym scenariuszu przeprowadzone zostaną badania sortowania już uporządkowanej listy, z wykorzystaniem standardowej implementacji funkcji sortującej dostępnej w bibliotekach języka Python [12]. Głównym zadaniem skryptu testującego będzie alokacja znacznej części systemowej pamięci operacyjnej tak, aby zbadać implementację funkcjonalności środowisk konteneryzacji odpowiedzialnych za zarządzanie pamięcią.

4.2.3. Scenariusz S3 – testowanie czasu uruchomienia kontenera

Ten scenariusz polega na pomiarze czasu wymaganego na wykonanie tak zwanego zimnego startu kontenera. Do testów wybrany został standardowy kontener języka Python w wersji 3.11.4. Czas potrzebny na uruchomienie został zdefiniowany jako czas od momentu wydania polecenia do uruchomienia kontenera do momentu otrzymania odpowiedzi na komendę *print("Hello world!")*.

5. Wyniki badań

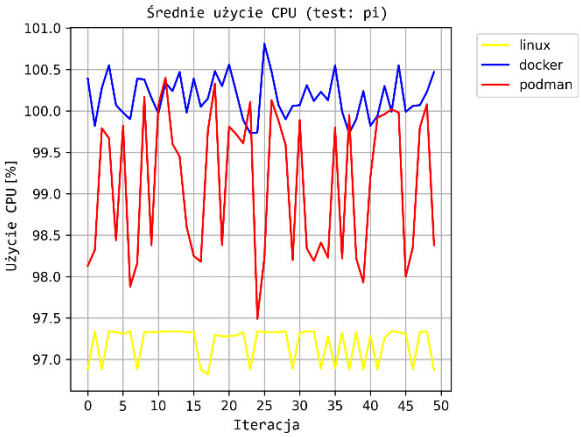
5.1. Wyniki dla scenariusza S1

Na Rysunku 2 przedstawiony został wykres średniego użycia procesora dla obliczeń liczby π. Wykres przedstawia pomiary z 50 uruchomień testów dla każdej z testowanych platform. Linia w kolorze żółtym oznacza procentowe zużycie procesora dla uruchomienia bez konteneryzacji, bezpośrednio w systemie Linux. Linia w kolorze czerwonym oznacza uruchomienie z wykorzystaniem platformy Podman. Linia w kolorze niebieskim oznacza uruchomienie na platformie Docker. Mniejsze użycie procesora oznacza lepszy wynik. Dokładniejsze wyniki oraz podstawowe wartości statystyczne zostały przedstawione w Tabeli 1.

Tabela 1: Wyniki zużycia CPU dla obliczeń liczby π

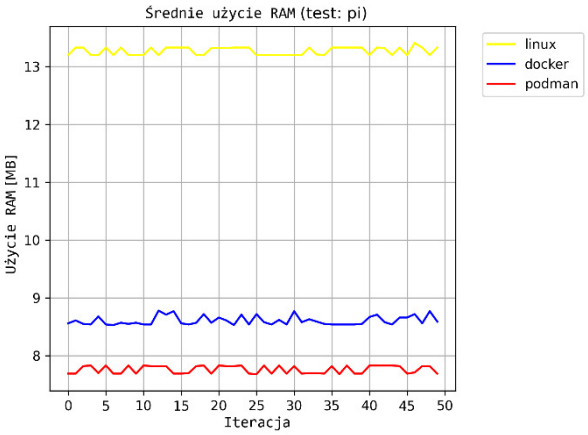
Platforma	Minimum (%)	Średnia (%)	Maksimum (%)	σ z próby
Linux	96,82	97,20	97,34	0,20
Docker	99,73	100,17	100,81	0,25
Podman	97,49	99,11	100,40	0,87

Na podstawie wykresu z Rysunku 6 oraz wyników przedstawionych w Tabeli 1 można stwierdzić, że najlepszy wynik w przeprowadzonym teście osiągnęło uruchomienie bez wykorzystania konteneryzacji, bezpośrednio w systemie Linux. Co warto zauważyć, w przypadku platformy Docker, generowane przez nią obciążenie przez znaczną część czasu wynosiło ponad 100%, oznacza to, że zadania w tym przypadku musiały czekać na wolny czas procesora. Należy jednocześnie podkreślić, że wyniki te dotyczą jednowątkowej aplikacji testowej.



Rysunek 2: Wykres przedstawiający średnie użycie procesora dla scenariusza S1.

Największe wahania wyników można zaobserwować w przypadku platformy Podman, a platformą generującą najbardziej jednorodne obciążenie był system Linux.



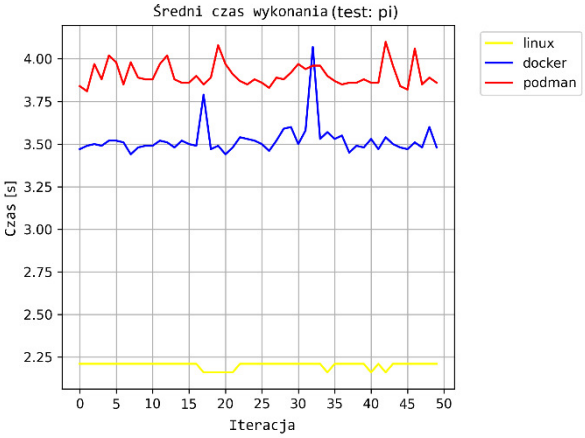
Rysunek 3: Wykres przedstawiający średnie użycie pamięci RAM dla scenariusza S1.

Tabela 2: Wyniki zużycia pamięci RAM dla obliczeń liczby π

Platforma	Minimum (MB)	Średnia (MB)	Maksimum (MB)	σ z próby
Linux	13,20	13,27	13,41	0,06
Docker	8,53	8,61	8,78	0,08
Podman	7,68	7,76	7,83	0,07

Na postawie wykresu z Rysunku 3 oraz danych przedstawionych w Tabeli 2 można stwierdzić, że najgorzej w trakcie przeprowadzonego testu wypadło uruchomienie bezpośrednio w systemie Linux, które zużywa o ponad 4 MB pamięci RAM więcej w porównaniu

do środowiska Docker oraz o ponad 5 MB więcej w porównaniu do platformy Podman. Różnica ta może być spowodowana, wykorzystaniem współdzielonych zasobów przez kontenery oraz lepszą optymalizacją współpracy z kernelem Linux. W przypadku tego testu najlepiej wypadło środowisko Podman.



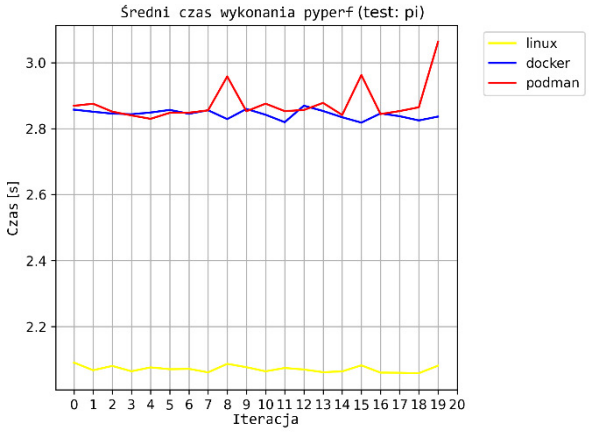
Rysunek 4: Wykres przedstawiający średni czas wykonania scenariusza S1.

Tabela 3: Wyniki dla czasu wykonania scenariusza S1

Platforma	Minimum (s)	Średnia (s)	Maksimum (s)	σ z próby
Linux	2,16	2,20	2,21	0,02
Docker	3,43	3,52	4,07	0,10
Podman	3,81	3,91	4,10	0,07

Analizując dane przedstawione na wykresie z Rysunku 4 oraz dane przedstawione w Tabeli 3 można zauważyć, że czas konieczny do wykonania scenariusza S1 jest znacząco dłuższy w przypadku rozwiązań wykorzystujących konteneryzację, niekiedy prawie dwukrotnie dłuższy niż czas realizacji tego samego zadania w systemie Linux. Najgorzej w tym teście wypada rozwiązanie Podman, może to być spowodowane tym, że jest to rozwiązanie niewykorzystujące stale działającego w tle demona, co pomimo zalet tego rozwiązania wymienionych w poprzednich rozdziałach, może wiązać się z dodatkowym czasem wymaganym do uruchomienia narzędzi konteneryzacji przy każdym starcie, zatrzymaniu
i modyfikacji kontenera.

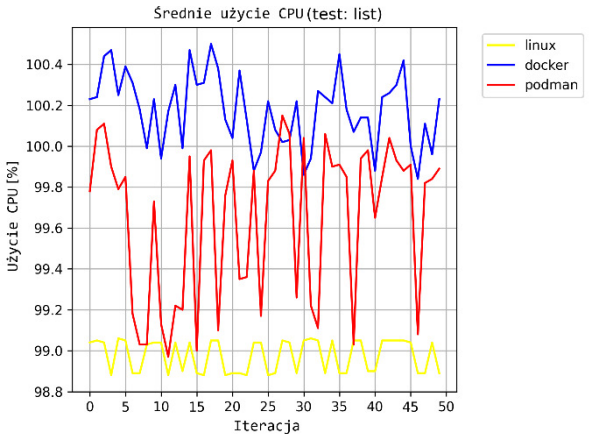
Wykres na Rysunku 5 przedstawia średni czas wykonania samych obliczeń liczby π, nie uwzględniając czasu potrzebnego na operacje związane z konteneryzacją. Pomiar czasu został wykonany przy pomocy modułu *pyperf* z języka Python. Po porównaniu wykresów z Rysunku 4 oraz Rysunku 5 można zauważyć, że czas potrzebny na zainicjowanie obliczeń w systemie Linux jest znacząco mniejszy niż czas potrzebny na uruchomienie oraz zatrzymanie kontenera zarówno w środowisku Docker, jak i Podman. W przybliżeniu czas potrzebny na operacje uruchomienia, zatrzymania i usunięcia kontenera w przypadku narzędzia Docker wynosi 0,68 sekundy, a przypadku narzędzia Podman 1,05 sekundy. Czas wykonywania samych obliczeń w przypadku rozwiązań Docker i Podman jest porównywalny.



Rysunek 5: Wykres przedstawiający średni czas wykonania obliczeń dla liczby π.

5.2. Wyniki dla scenariusza S2

W niniejszym podrozdziale przedstawione zostały wyniki badań testów zrealizowanych zgodnie ze scenariuszem S2.



Rysunek 6: Wykres przedstawiający średnie użycie procesora dla scenariusza S2.

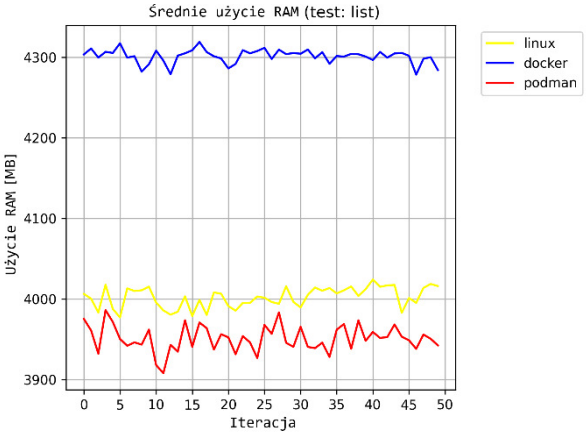
Na Rysunku 6 przedstawiony został wykres średniego obciążenia procesora dla obliczeń związanych z sortowaniem uporządkowanej listy.

Tabela 4: Wyniki zużycia CPU dla scenariusza S2

Platforma	Minimum (%)	Średnia (%)	Maksimum (%)	σ z próby
Linux	98,88	98,98	99,06	0,08
Docker	99,84	100,18	100,50	0,18
Podman	98,97	99,65	100,15	0,38

Analizując dane przedstawione na wykresie z Rysunku 6 oraz dane zgromadzone w Tabeli 4, można zauważyć podobieństwo do scenariusza S1. W tym przypadku również środowisko Docker okazało się tym, które bardziej obciąża procesor. Najbardziej zrównoważone obciążenie ponownie było generowane na platformie bez konteneryzacji wykorzystującej uruchomienie

bezpośrednio w systemie Linux. Największe wahania obciążenia dotyczą środowiska Podman.



Rysunek 7: Wykres przedstawiający średnie użycie pamięci RAM dla scenariusza S2.

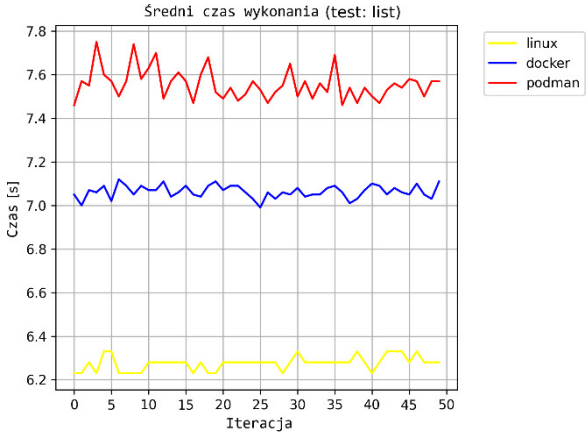
Tabela 5: Wyniki użycia pamięci RAM dla scenariusza S2

Platforma	Minimum (MB)	Średnia (MB)	Maksimum (MB)	σ z próby
Linux	3977,23	4002,14	4024,22	12,70
Docker	4278,47	4301,30	4318,95	8,61
Podman	3908,07	3951,17	3986,32	16,24

Dane przedstawione na wykresie z Rysunku 7 oraz dane zgromadzone w Tabeli 5 ukazują znaczne różnice w alokowaniu pamięci RAM przez narzędzia Docker i Podman. Środowisko Docker alokuje ponad 300 MB pamięci więcej w porównaniu do środowiska Podman przy wykonywaniu tego samego zadania. Prawdopodobnie ma to związek z problemem występującym w platformie Docker, polegającym na zbyt agresywnym alokowaniu zasobów przy stosunkowo małych zadaniach, zwłaszcza w warunkach małej liczby kontenerów i dostępności wolnych zasobów. Takie zachowanie zostało już zauważone przez autorów pracy [6] oraz [9].

Na podstawie danych przedstawionych na wykresie z Rysunku 8 oraz danych zgromadzonych w Tabeli 6 można zauważyć, że wszystkie 3 testowane platformy posiadają różne czasy wykonania dla tego samego zadania. Różnica między czasem wykonania dla środowiska Docker oraz środowiska Podman jest tutaj dużo wyraźniejsza niż w przypadku scenariusza S1 i wynosi w tym przypadku średnio prawie 0,5 sekundy, co stanowi wynik gorszy o około 6,5%. Sama powtarzalność czasu wykonania ma zauważalnie mniejsze wahania w przypadku kontenerów Docker, który oferuje porównywalną powtarzalność jak uruchomienie natywne w systemie Linux.

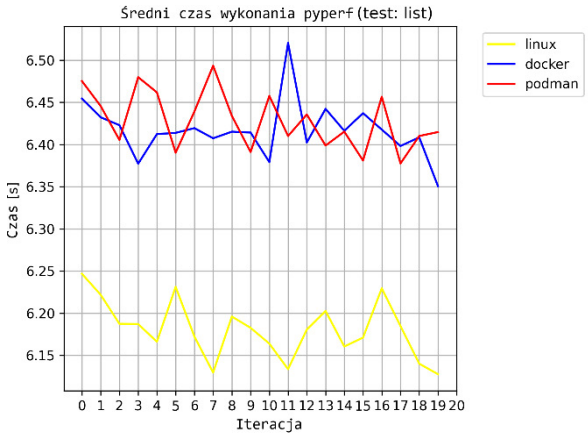
Po przeprowadzeniu badań czasu wykonania sortowania uporządkowanej listy za pomocą modułu *pyperf*, których wyniki zostały przedstawione na wykresie na Rysunku 9, można zauważyć, że sam czas trwania obliczeń dla kontenerów Docker i Podman nie różni się w sposób zauważalny oraz znaczący, jednak w porównaniu do uruchomienia natywnego można zaobserwować pewien spadek prędkości samych obliczeń, co świadczy o występowaniu pewnego rodzaju narzutu ze względu na konteneryzację.



Rysunek 8: Wykres przedstawiający średni czas wykonania scenariusza S2.

Tabela 6: Wyniki dla czasu wykonania scenariusza S2

Platforma	Minimum (s)	Średnia (s)	Maksimum (s)	σ z próby
Linux	6,23	6,28	6,33	0,03
Docker	7,00	7,06	7,11	0,03
Podman	7,46	7,55	7,75	0,07



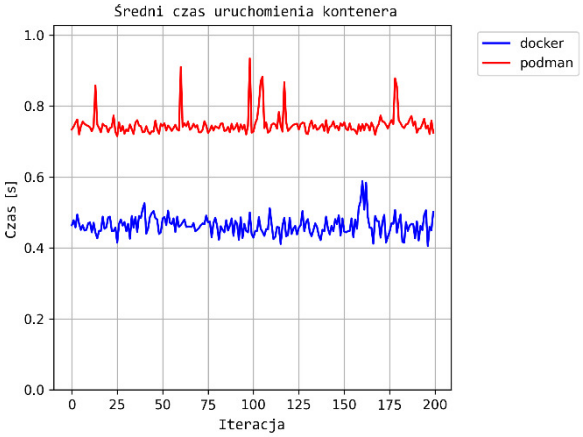
Rysunek 9: Wykres przedstawiający średni czas wykonania sortowania listy.

Narzut ten w przypadku czasu wykonywania obliczeń wewnątrz kontenera wynosi około 4% w porównaniu do uruchomienia natywnego w systemie Linux. Zauważa się także, że w przypadku kontenerów Podman występuje dodatkowy, wymierny narzut w postaci dodatkowego czasu potrzebnego na uruchomienie i zatrzymanie narzędzi konteneryzacji, które w przeciwieństwie do narzędzi ze środowiska Docker nie działają jako demon w tle.

5.3. Wyniki dla scenariusza S3

W niniejszym podrozdziale przedstawione zostały wyniki badań testów zrealizowanych zgodnie ze scenariuszem S3. Zestawiając ze sobą dane przedstawione na wykresie z Rysunku 10 oraz w Tabeli 7 z danymi zaprezentowanymi w scenariuszach S1 oraz S2 można zau-

ważyć, że narzędzie Podman posiada dodatkowy narzut do sumarycznego czasu wykonywania zadań w postaci czasu koniecznego na uruchomienie oraz zatrzymanie narzędzi konteneryzacji.



Rysunek 10: Wykres przedstawiający czas uruchomienia kontenera w scenariuszu S3.

Tabela 7: Wyniki dla czasu uruchomienia kontenera w scenariuszu S3

Platforma	Minimum (s)	Średnia (s)	Maksimum (s)	σ z populacji
Docker	0,405	0,463	0,588	0,026
Podman	0,715	0,747	0,934	0,031

Narzut ten wynosi od 0,3 sekundy do 0,7 sekundy w testowanych scenariuszach. W przypadku statycznych zadań, gdzie główną problematykę stanowi wykonywanie zadań w obrębie jednego kontenera, wpływ dodatkowego narzutu może być pominięty, ponieważ będzie stanowił niewielką część sumarycznego czasu działania kontenera, tak w przypadku dynamicznych środowisk, gdzie kontenery są uruchamiane i zatrzymywane z dużą częstotliwością, może stanowić przyczynę problemów z wydajnością i responsywnością, o ile nie zostanie w prawidłowy sposób rozwiązany.

6. Podsumowanie

Analizując wyniki zgromadzone w trakcie badań nad scenariuszami, można dostrzec pewne wzorce w działaniu obu porównywanych rozwiązań w dziedzinie konteneryzacji. Środowisko Podman w testowanych scenariuszach osiąga lepsze wyniki w kwestii ogólnie pojętego zarządzania zasobami. Użycie procesora oraz alokacja pamięci operacyjnej przez to rozwiązanie wypadają na ogół lepiej w porównaniu do narzędzia Docker. Różnica nie jest duża, ponieważ wynosi od 1% do maksymalnie 4% w przypadku wykorzystania CPU na korzyść środowiska Podman, jednak wynik ten jest powtarzalny i widoczny w wielu testach. Narzędzie Docker charakteryzuje się natomiast lepszą dynamiką działania, czas potrzebny na start i zatrzymanie kontenerów jest krótszy, co daje przewagę temu narzędziu od 0,3 sekundy do 0,7 sekundy według przeprowadzonych testów. Tutaj również różnica wyników jest niewielka, jednak może mieć znaczenie w pewnych zastosowaniach, wymagających wielokrotnego uruchamiania i zatrzymywania wielu kontenerów.

Pomimo występującego niewielkiego narzutu wydajności w zakresie 7 - 9%, wykorzystanie środowisk konteneryzacji, takich jak Docker i Podman, oferuje ogromne korzyści związane z wygodą, skalowalnością i zarządzaniem aplikacjami.

Warto również podkreślić, że różnice w wydajności na poziomie 7 - 9% są zazwyczaj akceptowalne w większości przypadków, zwłaszcza jeśli wziąć pod uwagę wszystkie korzyści związane z konteneryzacją. Ostatecznie, wybór między Dockerem a Podmanem może zależeć od indywidualnych potrzeb i preferencji, ale obie te platformy pozostają niezwykle przydatnymi narzędziami
w dziedzinie wdrażania i zarządzania aplikacjami.

Literatura

[1] S.Shah, N. Khandhar, Docker - The Future of Virtualization, International Journal of Research and Analytical Reviews (IJRAR) 6(2) (2019) 164 - 167.

[2] D.Walsh, Podman in Action, Manning Publications, (2023), ISBN: 9781633439689.

[3] M. Kjellstedt, Performance Evaluation of deploying microservices using Docker and Podman, thesis, Umeå University, (2020) 13 – 16.

[4] S. Giallorenzo, J. Mauro, M. G. Poulsen, F. Siroky, Virtualization Costs: Benchmarking Containers and Virtual Machines Against Bare-Metal, SN Computer Science 2(404) (2021) 11 – 15, https://doi.org/10.1007/s42979-021-00781-8.

[5] A. Subil, On the Use of Containers in High Performance Computing, 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), (2020) 284 - 293, https://doi.org/10.1109/CLOUD49709.2020.00048.

[6] E. Casalicchio, V. Percibali, Measuring Docker Performance: What a mess!!!, Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, (2017) 11 – 16, https://doi.org/10.1145/3053600.3053605.

[7] M. A. Potdar, G. D. Narayan, S. Kengond, M. M. Mulla, Performance Evaluation of Docker Container and Virtual Machine, Third International Conference on Computing and Network Communications (CoCoNet'19), (2019) 1419 – 1428, https://doi.org/10.1016/j.procs.2020.04.152.

[8] R. R. Yadav, G. T. E. Sousa, A. R. G. Callou, Performance Comparison Between Virtual Machines And Docker Containers, IEEE Latin America Transactions 16(8) (2018) 2282 – 2288, https://doi.org/10.1109/TLA.2018.8528247.

[9] C. MinSu, L. HwaMin, L. Kiyeol, A performance comparison of linux containers and virtual machines using Docker and KVM, Cluster Computing 22(1) (2019) 1765 – 1775, https://doi.org/10.1007/s10586-017-1511-2.

[10] B. B. Rad, J. H. Bhatti, M. Ahmadi, An Introduction to Docker and Analysis of its Performance, International Journal of Computer Science and Network Security (IJCSNS) 17(3) (2017) 228 – 234.

[11] Formuła Leibniza do obliczeń liczby π, https://en.wikipedia.org/wiki/Leibniz_formula_for_%CF%80, [06.09.2023].

[12] Sorting HOW TO - Python Documentation, https://docs.python.org/3/howto/sorting.html, [06.09.2023]