„dot.pl" - czasopismo Rejestru domeny .pl

Konteneryzacja - nowoczesna metoda utrzymywania usług

Mateusz Kozłowski
ORCID: https://orcid.org/0009-0005-6899-0244
E-mail: kontakt@matkoz2.dev

Received: 05.05.2024
Accepted: 25.05.2024
Published: 27.05.2024

Cite this article as:
M. Kozłowski, „Konteneryzacja - nowoczesna metoda utrzymywania usług"

DOT.PL, no. 1/ 2024, 10.60097/DOTPL/189321

Corresponding author:
Mateusz Kozłowski
E-mail: kontakt@matkoz2.dev

Streszczenie

Technologia konteneryzacji staje się coraz popularniejsza. Ewolucja na przestrzeni lat, wpłynęła na dojrzałość oraz łatwość korzystania z rozwiązania. Wzrost zainteresowania konteneryzacją spowodował, że narzędzie to staje się standardem na rynku. Artykuł ma na celu przybliżyć historię konteneryzacji oraz zaprezentować najważniejsze cechy, które wpłynęły na jej popularność.
Omówione zostaną również korzyści, które płyną z wykorzystania technologii konteneryzacji w dziedzinach tj. wytwarzanie oprogramowania oraz utrzymywanie aplikacji.

Słowa kluczowe: kontenery, docker, podman, bezpieczeństwo, izolacja, administracja aplikacjami

Abstract

Containerization technology is becoming increasingly popular. Evolving over the years, it has influenced the maturity and ease of use of the solution. The increased

interest in containerization has made this tool a standard on the market. This article aims to provide an overview of the history of containerization and present the key features that have influenced its popularity. It also discusses the benefits of using containerization technology in areas such as software development and application maintenance.

Keywords: containers, docker, podman, security, isolation, applications maintenance

Nazwy takie jak „Docker", „Podman" czy „Kubernetes", na przestrzeni ostatnich lat stały się coraz bardziej popularne. Wzrost zainteresowania konteneryzacją zawdzięczamy korzyściom, jakie za sobą niesie. Są to m.in. wygoda, łatwość korzystania, przyśpieszanie procesu wytwarzania oprogramowania oraz szeroka dostępność na główne systemy operacyjne, tj. Linux, MacOS lub Windows. Coraz więcej dostawców oprogramowania decyduje się na dostarczanie swojej aplikacji w kontenerze lub przynajmniej zapewnia taką opcję jako alternatywną do standardowego wdrożenia.

Sama idea konteneryzacji nie jest nowa. Pierwszym rozwiązaniem, które stało się podstawą do aktualnie znanych nam technologii, jest narzędzie *chroot*[1]. Zostało ono wprowadzone do systemu operacyjnego UNIX w 1979 roku. Narzędzie to pozwala na uruchomienie aplikacji w określonym katalogu, który z jej perspektywy staje się dla niej katalogiem głównym (z ang. *rootem - "/"*). Każdy kolejny proces potomny uruchomiony w ramach aplikacji, również zostaje uruchomiony ze zmienioną ścieżką główną. Korzystanie ze wspomnianego narzędzia wymaga od użytkownika dodatkowej wiedzy oraz doświadczenia. Przytoczona izolacja wymusza załączanie wszystkich zależności w katalogu docelowym, ponieważ uruchomiony proces nie ma dostępu do standardowych ścieżek w systemie operacyjnym hosta. Zazwyczaj sprowadza się to do konieczności odwzorowania struktury katalogów znajdujących się na hoście[2].

[1] Manual. Chroot-invocation, https://www.gnu.org/software/coreutils/manual/html_node/chroot-invocation.html dostęp: 20.05.2024.
[2] J. Bressers, *Is chroot a security feature?* https://www.rredhat.com/en/blog/chroot-security-feature, dostęp: 20.05.2024.

Kolejnym rozwiązaniem, będącym próbą stworzenia czegoś na wzór mechanizmu, który dziś znamy pod nazwą konteneryzacji, były *FreeBSD Jails*[3]. Narzędzie to zostało zaprezentowane w 2000 r. Był to mechanizm inspirowany ideą *chroot*, jednak autorzy postanowili pójść krok dalej i zaproponowali technologię, która nie ograniczała się jedynie do izolacji na poziomie plików, ale również przestrzeni użytkowników oraz sieci. Mechanizmy te gwarantowały separację pomiędzy aplikacjami uruchamianymi w ramach różnych *Jails*. Dzięki nowemu podejściu, a w zasadzie rozbudowie *chroot,* autorzy rozwiązania zaproponowali narzędzie pozwalające na bardziej szczegółową izolację oraz kontrolę aplikacji działającej w ramach środowiska *Jail*. Niestety rozwiązanie ograniczało się jedynie do systemu FreeBSD, przez co nie zdobyło tak dużej popularności, żeby stać się powszechnie wykorzystywanym[4].

Następnym etapem popularyzacji konteneryzacji było zaprezentowanie w 2004 r., przez ówczesną firmę *Sun Microsystems* (aktualnie *Oracle Corporation*), technologii *Solaris Containers*[5]. Co do zasady było to rozwiązanie, podobne do wcześniej wspomnianego *FreeBSD Jails,* przy czym zawierało kilka ulepszeń, tj. możliwość alokacji zdefiniowanej ilości zasobów. W tym przypadku rozwiązanie również było ekskluzywne dla systemu Solaris, przez co nie zdobyło aż tak dużej popularności, żeby stać się standardem.

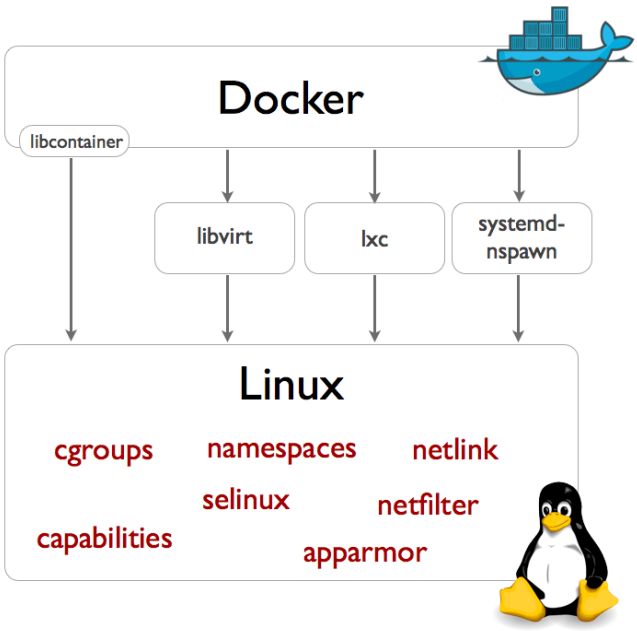
Przełom nastąpił w 2006 roku, kiedy to firma Google zaprezentowała rozwiązanie *Process Containers*, które w roku 2007 zostało nazwane *Control Groups* (*cgroups*) i pod tą nazwą znane jest do dziś. Inżynierowie firmy Google pokazali światu mechanizm, który umożliwiał przydzielanie zasobów sprzętowych oraz nadawanie priorytetów procesom. Następnie w 2008 r. rozwiązanie to zostało włączone do jądra Linuxa, przez co zyskało na popularności i pozwoliło na utworzenie *Linux Containers* (LXC)[6].

Linux Containers jest pierwszym mechanizmem, który przypomina kontenery w dzisiejszym tego słowa znaczeniu. Technologia ta została stworzona przez firmę Canonical i stała się powszechna oraz włączona do wielu popularnych dystrybucji

[3] Free BSD Handbook, https://docs.freebsd.org/en/books/handbook/jails, dostęp: 20.05.2024.
[4] K. Dabik, *Cyberprzestrzeń - zagrożenia i wyzwania*, [w:] M. Karpiuk (red.), *Cyberbezpieczeństwo – aspekty krajowe i międzynarodowe*, Warszawa 2024, s. 9 i n.
[5] Solaris containers, https://www.oracle.com/solaris/technologies/solaris-containers.html, dostęp: 20.05.2024.
[6] History containers, https://www.redhat.com/en/blog/history-containers, dostęp: 20.05.2024.

systemu Linux[7]. Technologia LXC realizowała to zadanie poprzez uruchomienie wielu procesów, które z perspektywy aplikacji działającej w środku, symulowały całkowicie niezależny od hosta system operacyjny. Z tego też względu rozwiązanie to jest określane mianem „lekkich maszyn wirtualnych”[8]. Kontenery LXC były mechanizmem stosunkowo zaawansowanym, przez co wymagały wiedzy od użytkownika. Fakt ten spowodował, że konteneryzacja LXC zyskała na popularności, ale tylko w kręgach osób zaznajomionych oraz doświadczonych w pracy z systemem Linux[9].

Po kolejnych kilku latach, w 2013 roku, podczas krótkiego wystąpienia w trakcie konferencji *PyCon,* Solomon Hykes przedstawia światu rozwiązanie Docker[10]. Technologia ta pierwotnie bazowała na LXC, jednak autorzy postanowili wprowadzić własną implementację w 2014 r. pod nazwą *libcontainer*[11], która dawała im większą kontrolę nad projektem.

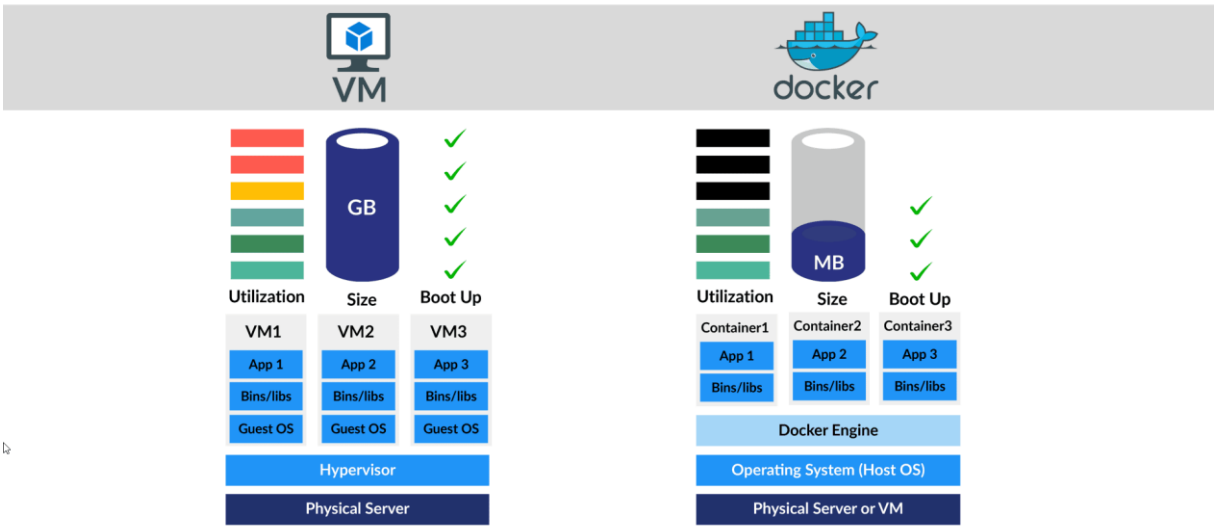


[7] D. Naprawa, *Docker vs LXC – czym to się różni?* https://szkoladockera.pl/czym-rozni-sie-docker-od-lxc, dostęp: 20.05.2024.
[8] Ibidem.
[9] R. Weber, *Legal safeguard for cloud computing*, [in:] A. Cheung, R. Weber (eds), *Privacy and Legal Issues in Cloud Computing*, Massachusetts 2016, s. 43 i n.
[10] The future of Linux Containers, https://www.youtube.com/watch?v=wW9CAH9nSLs, dostęp: 20.05.2024.
[11] Docker Desktop 0.9: Introducing Execution Drivers and libcontainer, https://www.docker.com/blog/docker-0-9-introducing-execution-drivers-and-libcontainer, dostęp: 20.05.2024.

Rysunek 1: *Schemat przedstawiający komunikację rozwiązania Docker z komponentami systemu, źródło: https://www.docker.com/blog/docker-0-9-introducing-execution-drivers-and-libcontainer (dostęp: 20.05.2024)*

Ogromną zaletą rozwiązania Docker była łatwość jego użytkowania. Postawiono na prostotę, która w poprzednich implementacjach konteneryzacji, tj. FreeBSD Jails lub LXC, nie była oczywista. Dzięki Dockerowi, uruchomienie kontenera sprowadzało się do wydania prostego polecenia w linii komend ***docker run NAZWA_OBRAZU KOMENDA***. Obrazami w tym przypadku są wcześniej przygotowane pliki, które zawierają w sobie docelową funkcjonalność.

To właśnie obraz uruchamiany jest w ramach konteneryzacji, co może w pewnym sensie kojarzyć się z pojęciem wirtualizacji. W przypadku kontenerów należy jednak pamiętać, że nie występuje emulacja jak przy standardowej wirtualizacji. Dostęp do zasób systemowych hosta jest izolowany przy pomocy specjalnych mechanizmów[12].



Rysunek 2: *Różnica pomiędzy wirtualizacją, a konteneryzacją. Źródło: https://k21academy.com/docker-kubernetes/docker-vs-virtual-machine/ (dostęp: 20.05.2024)*

Konteneryzacja z wykorzystaniem narzędzia Docker wymusza, choć są od tego odstępstwa, stosowanie zasady *Single Concern Principle*, która zmienia dotychczasowe

[12] D. Naprawa, *Historia konteneryzacji – czy było coś wcześniej przed Dockerem?*, https://www.youtube.com/watch?v=FAIZPF3Q80k, dostęp: 20.05.2024.

podejście do konteneryzacji w rozumieniu LXC. Jak wcześniej wspomniano, w ramach Linux Containers uruchomiano wiele procesów, celem zasymulowania całego systemu operacyjnego. Z kolei w Docker aplikacje powinny być rozbijane na poszczególne funkcjonalności tak, żeby jeden kontener odpowiadał jednemu procesowi lub inaczej mówiąc, był odpowiedzialny tylko za konkretną funkcjonalność. Zasada znacząco porządkuje oraz ułatwia zarządzanie aplikacją, a w szczególności jej zasobami, uprawnieniami oraz bezpieczeństwem. Podejście to ułatwia również proces aktualizacji, który ogranicza się jedynie do komponentu, który tego wymaga.

Kolejnym założeniem, na którym opiera się konteneryzacja w rozumieniu tego, co znamy pod nazwą Docker, to tworzenie aplikacji w taki sposób, żeby wszystkie zależności były zawarte w obrazie przygotowanym do uruchomienia. Ma to na celu uniknięcie problemu, który występował w przypadku korzystania z *chroot*. Wykorzystanie tego rozwiązania wymagało od użytkownika linkowania lub kopiowania zależności docelowego katalogu startowego.

Nowoczesne zasady konteneryzacji wymagają od twórców przestrzegania reguły niezmienności zawartości kontenera. Reguła ta polega na tym, że wszystkie dane, które powinny zostać zapisane, należy wynosić poza kontener, wykorzystując tzw. Volume[13]. Mechanizm należy traktować jako nieulotną przestrzeń dyskową, która gwarantuje nam, że najważniejsze dane nie zostaną utracone w przypadku usunięcie kontenera[14].

Stosując powyższe zasady (choć nie są to jedyne zasady dotyczące konteneryzacji), zarządzanie aplikacjami staje się wygodne oraz uporządkowane. Powyższe możliwości wynikające z wykorzystania konteneryzacji są jedynie częścią korzyści, które wpłynęły na sukces narzędzia. Wykorzystanie tej technologii pozwoliło również ujednolicić proces uruchamiania aplikacji poprzez rozwiązanie problemu ze zgodnością wersji oprogramowania zainstalowanego na systemie operacyjnym hosta.

Kolejnym ułatwieniem, które miało ogromny wpływ na popularność rozwiązania, było narzędzie Docker Desktop, które pozwalało zarządzać kontenerami z wykorzystaniem interfejsu graficznego. Rozwiązanie Docker jest dostępne na wszystkie kluczowe systemy

[13] Volumes, https://docs.docker.com/storage/volumes, dostęp: 20.05.2024.
[14] B. Ibryam, *Principles of container-based application design*, https://kubernetes.io/blog/2018/03/principles-of-container-app-design/, dostęp: 20.05.2024.

operacyjne tj. Linux, MacOS oraz Windows. Wszystko to powoduje, że uruchomienie oraz korzystanie z narzędzia nie wymaga specjalistycznej wiedzy.

Należy również pamiętać, że sam Docker to nie tylko narzędzie do zarządzania kontenerami. Firma uruchomiła wiele interesujących produktów, które przyciągają coraz to większe zainteresowanie, a są to m.in.:

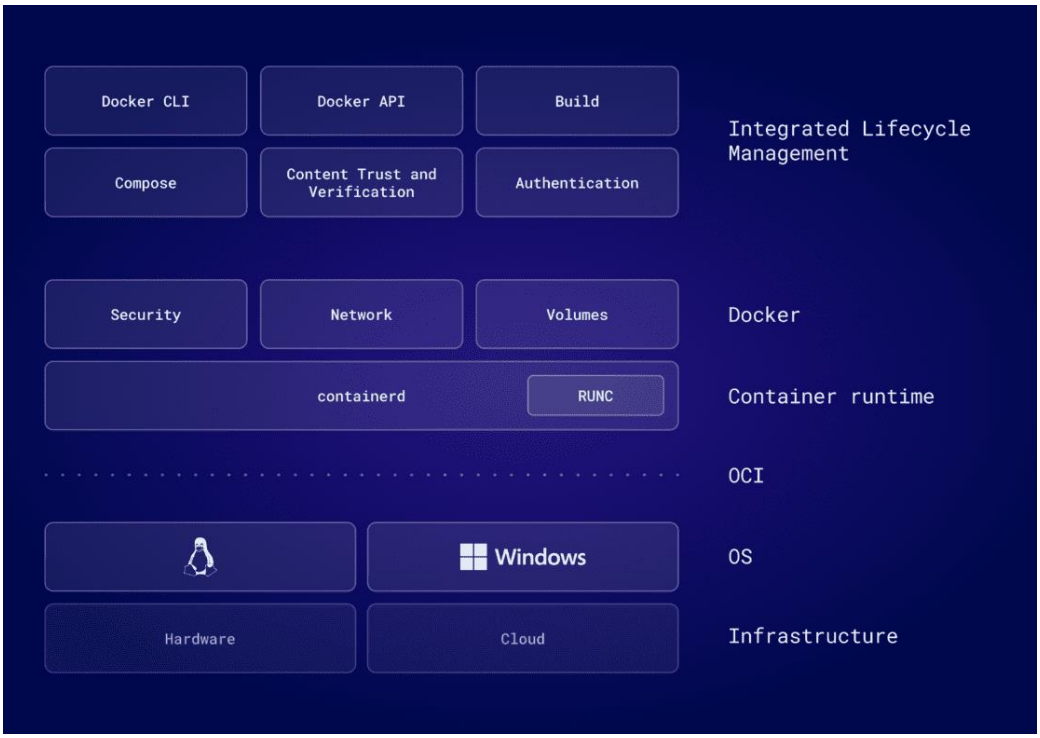
- Docker Hub[15] - centralny punkt dystrybucji obrazów, który umożliwia twórcom publikowanie swoich aplikacji
- Docker Scout[16] - skaner podatności w obrazach kontenerów;
- Docker Compose[17] - narzędzie pozwalające na zdefiniowanie aplikacji zawierającej w sobie wiele powiązanych ze sobą kontenerów, które opisane są z wykorzystaniem dedykowanej składni w pliku YAML.

Konteneryzacja znacząco wpłynęła na podejście do procesu wytwarzania oraz utrzymywania aplikacji. Wiele firm posiadających w swoich strukturach programistów, testerów oraz administratorów, zaobserwowało w tej technologii możliwość optymalizacji czasu, który był zajmowany przez liczne procedury, takie jak: testowanie oprogramowania oraz jego wdrażanie. W ten sposób, w kontekście konteneryzacji, zaczęto kierować się zasadą *Site Reliability Engineering*, która zakładała łączenie funkcji dotychczas traktowanych jako oddzielne, w obowiązkach należących do jednej osoby. Konteneryzacja sprawdziła się świetnie jako element uzupełniający to zadanie. Technologia pozwoliła rozszerzyć dotychczasową odpowiedzialność programisty o dodatkowe elementy tj. uruchomienie aplikacji i ewentualną obsługę błędów. Było to realne dzięki możliwościom oferowanym przez kontenery, a w szczególności przez fakt, że kontener funkcjonuje identycznie na dowolnym obsługiwanym hoście. Podejście to stanowczo skróciło proces obsługi i naprawy błędów, poprzez przybliżenie osób odpowiedzialnych za tworzenie kodu do jego obsługi w systemach docelowych.

Wraz ze wzrostem pojęcia konteneryzacji, rozpoczęto prace nad kolejnymi standardami, które miały na celu przyśpieszenie dotychczas stosowanych procesów. W ten sposób na rynku pojawiło się pojęcie *Cloud Native*, które miało na celu uporządkowanie kwestii

[15] Docker Hub, https://docs.docker.com/docker-hub, dostęp: 20.05.2024.
[16] Docker Scout, https://docs.docker.com/scout, dostęp: 20.05.2024.
[17] Docker Compose, https://docs.docker.com/compose, dostęp: 20.05.2024.

związanych z wytwarzaniem oraz utrzymywaniem aplikacji. Ważnym elementem składającym się na ww. zagadnienie było uniezależnienie się od konkretnego dostawcy usług. Wiele firm było przywiązane do konkretnego operatora, przez co migracja była czasochłonna lub wręcz niemożliwa. Z upływem czasu wymyślono podejście *Infrastructure as Code,* które zakładało, że cała infrastruktura aplikacji zostaje opisana w formie kodu. To z kolei miało zagwarantować brak przywiązania do konkretnego dostawcy, ponieważ co do zasady opis infrastruktury pozwala na jej automatyczne odtworzenie w identycznej formie u dowolnego operatora lub w modelu *on-premise*[18].



Rysunek 3: *Poglądowy rysunek zależności pomiędzy poszczególnymi komponentami składającymi się na uruchomienie kontenera. Źródło: https://www.docker.com/blog/containerd-vs-docker/ (dostęp: 20.05.2024)*

Popularyzacja technologii spowodowała również rozwój konkurencji w postaci rozwiązań alternatywnych tj. Podman, które aktualnie rozwijane jest przez firmę RedHat Inc. W przypadku tego narzędzia wprowadzono również kilka innych innowacji tj. konteneryzacja *rootless* oraz narzędzie *Podman Quadlet*. Wszystkie ww. narzędzia mają na celu

[18] *Infrastructure as Code (IaC)*. (2023, 11 30). Retrieved from https://glossary.cncf.io/infrastructure-as-code/, dostęp: 20.05.2024.

zwiększenie wygody użytkownika, zautomatyzowanie procesów oraz podniesienie bezpieczeństwa[19].

Kontenery rootless pozwoliły odejść od standardowego podejścia, które wymagało uruchomienia tzw. demona, który odpowiadał za zarządzeniem kontenerami. Zmiana ta znacznie wpłynęła na bezpieczeństwo konteneryzacji, ponieważ nie wymagała uruchamiania procesów z uprawnieniami administracyjnymi i w przypadku pojawienia się błędu umożliwiającego „wyjście" z kontenera, atakujący działał na uprawnieniach zwykłego użytkownika.

	Root Outside	**User Outside**

Root Inside	# whoami
	root
	
	# podman run -it ubi8 bash
	# whoami
	root
		$ whoami
		fatherlinux
		
		$ podman run -it ubi8 bash
		# whoami
		root

User Inside	# whoami
	root
	
	# podman run -itu sync ubi8 bash
	$ whoami
	sync
		$ whoami
		fatherlinux
		
		$ podman run -itu sync ubi8 bash
		$ whoami
		sync

Rysunek 4: *Porównanie uprawnień procesu kontenera w zależności od uprawnień, z jakimi został uruchomiony. Źródło: https://www.redhat.com/en/blog/understanding-root-inside-and-outside-container (dostęp: 20.05.2024)*

W skład narzędzia Podman wchodzi również komponent o nazwie *Quadlet*[20]. Rozwiązanie to ułatwia opisywanie kontenerów przy użyciu kodu. Mechanizm można porównać do rozwiązania Docker Compose, które pozwala opisywać wielokontenerowe aplikacje poprzez definiowanie ich w specjalnie przygotowanym pliku YAML. Wdrożenie tej funkcjonalności jest na potrzebę dostosowania się do zasady zawartej w Cloud Native, która zakłada, że uruchamiane aplikacje powinny być niezależne od dostawcy konkretnego usług cyfrowych i możliwe do bezproblemowej migracji. Poniżej znajduje się

[19] C. Banasiński, *Podstawowe pojęcia i podstawy prawne bezpieczeństwa w cyberprzes*trzeni, [w:] C. Banasiński (red.), *Cyberbezpieczeństwo*, Warszawa 2018, s. 27.
[20] Podman system, https://docs.podman.io/en/latest/markdown/podman-systemd.unit.5.html, dostęp: 20.05.2024.

przykład definicji prostego kontenera opisanego z wykorzystaniem rozwiązania Podman Quadlet.

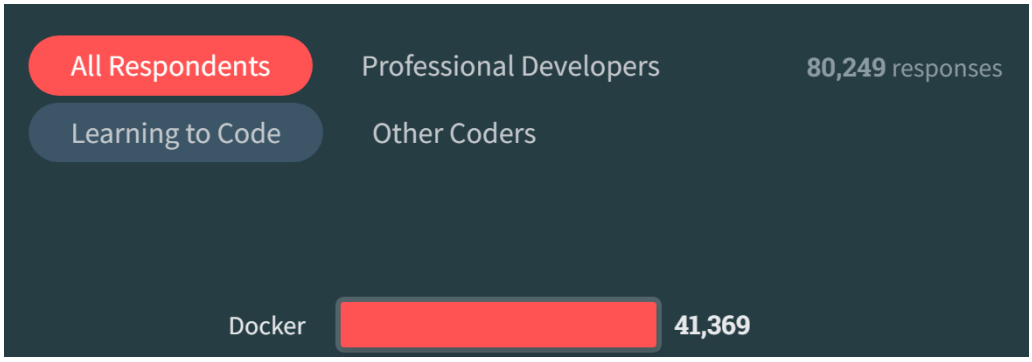
```
[Container]
ContainerName=busybox1
Image=docker.io/busybox
Exec=/bin/sh
```

Rysunek 5: *Definicja kontenera z wykorzystaniem Podman Quadlet, opracowanie własne.*

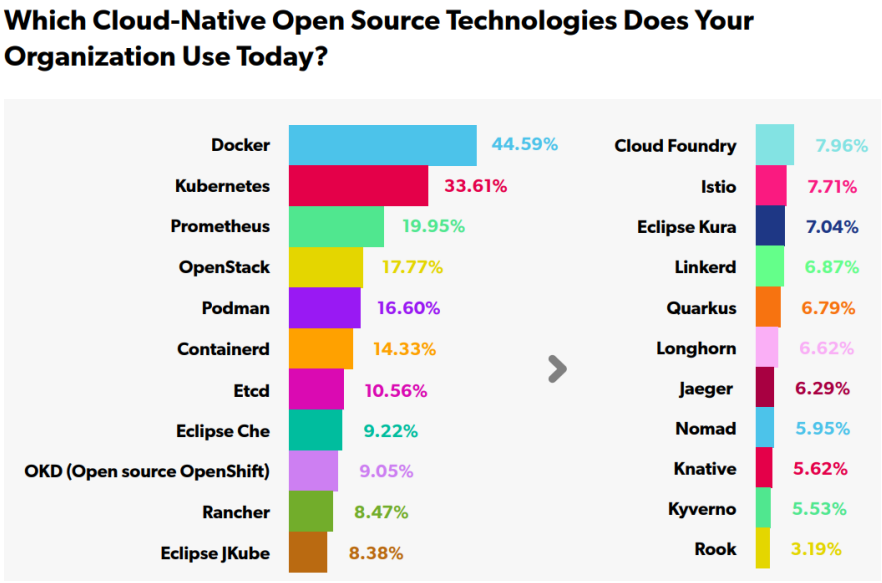
Dzięki zaangażowaniu dużych firm w technologię konteneryzacji, na rynku pojawiają się kolejne narzędzia ułatwiające pracę w tym środowisku. Giganci technologiczni sami wdrażają kontenery, w swoich strukturach. Są przez to zmotywowani do rozwijania narzędzi, które im to ułatwią. W ten sposób powstało wiele interesujących projektów, które zostały udostępnione bezpłatnie dla całej społeczności. Przykładowymi rozwiązaniami, które zostały stworzone przez duże przedsiębiorstwa i są stosowane przez użytkowników na skalę globalną są: Kubernetes[21], Google Distroless[22], OKD[23], Fedora CoreOS[24]. Wszystkie te narzędzia są rozwijane i mają na celu popularyzację technologii konteneryzacji.

Najnowsze badania pokazują, że technologia konteneryzacji jest aktualnie bardzo popularna. Została pozytywnie przyjęta przez osoby zajmujące się zagadnieniami z dziedziny informatyki. Zainteresowanie technologią sugeruje, że na przestrzeni kolejnych lat konteneryzacja stanie się standardem w procesie utrzymywania aplikacji. W przypadku przestrzegania dobrych praktyk związanych z tworzeniem kontenerów, użytkownicy powinni zaobserwować poprawę wydajności oprogramowania oraz zwiększyć wygodę jego utrzymywania. Trend ten powinien również przełożyć się na ogólną redukcję kosztów, jak i poprawę bezpieczeństwa uruchamianych aplikacji.

[21] Production-Grade Container Orchestration, https://kubernetes.io, dostęp: 20.05.2024.
[22] Google container tools, https://github.com/GoogleContainerTools/distroless, dostęp: 20.05.2024.
[23] The Community Distribution of Kubernetes that powers Red Hat OpenShift, https://www.okd.io, dostęp: 20.05.2024.
[24] Fedora CoreOS Documentation, https://docs.fedoraproject.org/en-US/fedora-coreos, dostęp: 20.05.2024.



Rysunek 6: *Wyniki ankiety przeprowadzonej przez firmę Stack Exchange Inc. wskazującej zainteresowanie technologią Docker w 2023 roku, źródło: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-other-tools (dostęp: 20.05.2024)*



Rysunek 7: *Wyniki ankiety przeprowadzonej przez firmę Perforce Software, Inc. wskazującej zainteresowanie technologią Docker w 2023 roku. W ankiecie wzięło udział 2046 osób. źródło: https://www.openlogic.com/resources/state-of-open-source-report (dostęp: 20.05.2024)*

Konteneryzacja oraz technologie powstające wokół niej pozwalają uprościć oraz przyśpieszyć wiele procesów, które do tej pory były czasochłonne. Kontenery pozwalają ujednolicić pracę programistów poprzez zagwarantowanie spójności środowiska uruchomieniowego. Kolejny etap, który staje się łatwiejszy dzięki ww. technologii to testy aplikacji. Dzięki spójności środowiska, gotowa aplikacja może trafić do testera, który po prostu ją uruchomi bez konieczności rozwiązywania problemów z zależnościami. Konteneryzacja pozwala również, ze względu na swoje założenia, przyśpieszyć proces

wdrożenia aplikacji, który co do zasady sprowadza się do pobrania obrazu, przygotowania konfiguracji oraz uruchomienia aplikacji. Dodatkowo, warty uwagi jest fakt, że konteneryzacja zapewnia izolację aplikacji od hosta, co przekłada się na zwiększone bezpieczeństwo całego środowiska.

Konteneryzacja w każdym roku zyskuje na popularności i wydaje się, że trend ten będzie utrzymany. Jest to zjawisko pozytywne, ponieważ technologia ta rozwiązuje wiele problemów i wpływa korzystnie na wiele aspektów utrzymywania aplikacji, przez co stanowi świetną alternatywę dla standardowego podejścia wytwarzania oraz utrzymywania aplikacji, jakie było znane dotychczas.

Bibliografia

J. Bressers, *Is chroot a security feature?,* https://www.redhat.com/en/blog/chroot-security-feature, dostęp: 20.05.2024.

C. Banasiński, *Podstawowe pojęcia I podstawy prawne bezpieczeństwa w cyberprzestrzeni*, [w:] C. Banasiński (red.), *Cyberbezpieczeństwo*, Warszawa 2018.

K. Dabik, *Cyberprzestrzeń - zagrożenia i wyzwania*, [w:] M. Karpiuk (red.), *Cyberbezpieczeństwo – aspekty krajowe i międzynarodowe*, Warszawa 2024.

Docker Compose, https://docs.docker.com/compose, dostęp: 20.05.2024.

Docker Desktop 0.9: Introducing Execution Drivers and libcontainer, https://www.docker.com/blog/docker-0-9-introducing-execution-drivers-and-libcontainer, dostęp: 20.05.2024.

Docker Hub, https://docs.docker.com/docker-hub, dostęp: 20.05.2024.

Docker Scout, https://docs.docker.com/scout, dostęp: 20.05.2024.

Fedora CoreOS Documentation, https://docs.fedoraproject.org/en-US/fedora-coreos, dostęp: 20.05.2024.

Free BSD Handbook, https://docs.freebsd.org/en/books/handbook/jails, dostęp: 20.05.2024.

Google container tools, https://github.com/GoogleContainerTools/distroless, dostęp: 20.05.2024.

History containers, https://www.redhat.com/en/blog/history-containers, dostęp: 20.05.2024

B. Ibryam, *Principles of container-based application design*, https://kubernetes.io/blog/2018/03/principles-of-container-app-design/, dostęp: 20.05.2024.

Infrastructure as Code (IaC), https://glossary.cncf.io/infrastructure-as-code/, dostęp: 20.05.2024.

LXC containers. (n.d.), https://ubuntu.com/server/docs/lxc-containers, dostęp: 20.05.2024.

Manual. Chroot-invocation, https://www.gnu.org/software/coreutils/manual/html_node/chroot-invocation.html dostęp: 20.05.2024.

D. Naprawa, *Docker vs LXC – czym to się różni?*, https://szkoladockera.pl/czym-rozni-sie-docker-od-lxc, dostęp: 20.05.2024.

D. Naprawa, *Historia konteneryzacji – czy było coś wcześniej przed Dockerem*, https://www.youtube.com/watch?v=FAIZPF3Q80k, dostęp: 20.05.2024.

Podman system, https://docs.podman.io/en/latest/markdown/podman-systemd.unit.5.html, dostęp: 20.05.2024.

Production-Grade Container Orchestration, https://kubernetes.io, dostęp: 20.05.2024.

Solaris containers, https://www.oracle.com/solaris/technologies/solaris-containers.html, dostęp: 20.05.2024.

The Community Distribution of Kubernetes that powers Red Hat OpenShift, https://www.okd.io, dostęp: 20.05.2024.
The future of Linux Containers, https://www.youtube.com/watch?v=wW9CAH9nSLs, dostęp: 20.05.2024.
R. Weber, *Legal safeguard for cloud computing*, [in:] A. Cheung, R. Weber (eds.), *Privacy and Legal Issues in Cloud Computing*, Massachusetts 2016.
Volumes, https://docs.docker.com/storage/volumes, dostęp: 20.05.2024.

Wykaz rysunków