Kontenery: od chroot do Kubernetesa

Jak sztuczka z katalogiem głównym z 1979 roku, rosyjskie Virtuozzo, dwie łatki z Google i krótkie wystąpienie na konferencji Pythona złożyły się w sposób, w jaki w 2026 roku uruchamia się dużą część oprogramowania serwerowego — i dlaczego kontener wciąż nie jest maszyną wirtualną.

Ta strona opowiada historię, w której nikt nie wynalazł kontenera. Najpierw był chroot, narzędzie do budowania systemu; potem jaile FreeBSD, rosyjskie Virtuozzo i strefy Solarisa — sposoby na upchnięcie wielu klientów na jednym serwerze. Potem jądro Linuksa przez kilkanaście lat dostawało osobne klocki: przestrzenie nazw, grupy kontrolne, uprawnienia, filtry wywołań systemowych, a Google wkładał do niego kod, bo sam od dawna tak uruchamiał swoje usługi. W marcu 2013 roku mała firma od platformy hostingowej pokazała, jak te klocki zapakować w obraz, który da się wysłać i uruchomić gdziekolwiek — i od tego zaczęła się wojna formatów, wojna orkiestratorów i kilka sprzedaży firm. Osobno opisuję ucieczki z kontenerów, w tym najgłośniejszą, znalezioną przez dwóch Polaków w Nowy Rok 2019 roku, oraz to, co w październiku 2026 roku administrator musi zrobić, żeby jego klaster nie wypadł z łatek.

> Umiejscowienie w serii: strona należy do księgi „Serwerownia”, obok **Dystrybucje Linuksa: rodowód**, **Red Hat, RHEL i saga CentOS**, **Debian i Ubuntu** oraz **SUSE i openSUSE**. Hipernadzorcę, czyli drugą drogę do dzielenia serwera, opisuje strona **VMware ESX i ESXi**. Szczegóły jaili, VNET i bhyve są na stronie **FreeBSD**, strefy Solarisa i markę lx — na stronie **Solaris i SunOS**, WPAR i partycje HP-UX w szerszym tle — na stronie **Komercyjne UNIX-y: AIX, HP-UX, IRIX, SCO i Xenix**. Tabelę przestrzeni nazw jądra w kontekście całego Linuksa ma strona **Linux i GNU**, kontenery w Windows Server 2016 i obraz Nano — **Windows Server 2008–2025**, a Docker Desktop na zapleczu WSL 2 — **Windows Subsystem for Linux**. Monitor maszyn wirtualnych z Chromebooków, z którego wyrósł Firecracker, pojawia się na stronie **ChromeOS i ChromeOS Flex**; samo pojęcie wywołania systemowego i katalogu głównego — na stronie **UNIX**.

□□ Skróty użyte na tej stronie (82)

Skrót	Co znaczy
ACM	Association for Computing Machinery: stowarzyszenie informatyków, wydawca ACM Queue
AI	Artificial Intelligence: sztuczna inteligencja

Skrót	Co znaczy
AIX	UNIX IBM-a, z partycjami WPAR od wersji 6.1
AKS / EKS / GKE	usługi Kubernetesa w chmurach Azure, Amazon i Google
API	Application Programming Interface: interfejs programistyczny
AT&T	amerykański koncern telekomunikacyjny, właściciel Bell Labs i UNIX-a
AWS	Amazon Web Services: chmura Amazona
BPF	Berkeley Packet Filter: programy filtrujące w jądrze, używane przez seccomp
BSD	Berkeley Software Distribution: UNIX z Uniwersytetu Kalifornijskiego
BSI	Bitnami Secure Images: płatna oferta obrazów Bitnami
CFS	Completely Fair Scheduler: planista procesów Linuksa
CI	Continuous Integration: ciągła integracja
CNCF	Cloud Native Computing Foundation: fundacja Kubernetesa przy Linux Foundation
CRI	Container Runtime Interface: interfejs kubeleta do środowisk uruchomieniowych
CRI-O	środowisko uruchomieniowe dla Kubernetesa z rodziny Red Hata
CRIU	Checkpoint/Restore In Userspace: zapis i odtwarzanie stanu procesów
CTF	Capture The Flag: zawody w bezpieczeństwie komputerowym
CVE / CVSS	rejestr podatności i skala ich ważności od 0 do 10
D2IQ	nazwa Mesosphere od 2019 roku
DC/OS / DCOS	platforma Mesosphere zbudowana na Mesosie
DEB / RPM	formaty pakietów Debiana i Red Hata
EMC	producent pamięci masowych, w 2004 roku właściciel VMware
ESX / ESXi	hipernadzorca VMware
FTP	File Transfer Protocol: protokół przesyłania plików
GA	General Availability: ogólna dostępność wydania
GCP	Google Cloud Platform: chmura Google
GNU / GPL	projekt wolnego oprogramowania i jego licencja
HP / HP-UX	Hewlett-Packard i jego UNIX
IBM	International Business Machines
IP / IPv4 / IPv6	protokół internetowy i jego wersje; adres sieciowy
IPC	Inter-Process Communication: komunikacja międzyprocesowa
IRIX	UNIX firmy SGI
IT	Information Technology: informatyka

Skrót	Co znaczy
K8s / k8s	skrótowa nazwa Kubernetesa
KCD	Kubernetes Community Days: lokalne konferencje społeczności
KEP	Kubernetes Enhancement Proposal: dokument zmiany w Kubernetesie
KVM	Kernel-based Virtual Machine: hipernadzorca w jądrze Linuksa
LISA	Large Installation System Administration: konferencja administratorów
LPAR	Logical Partition: partycja logiczna w AIX i HP-UX
LTS	Long-Term Support: wydłużone wsparcie
LWN	LWN.net: serwis o jądrze Linuksa
LXC / LXD	Linux Containers i jego menedżer kontenerów systemowych
MB / MiB	megabajt dziesiętny i binarny
MIPT	Moskiewski Instytut Fizyko-Techniczny
NA	North America: tu KubeCon w Ameryce Północnej
NAT	Network Address Translation: wiele maszyn za jednym adresem
NGINX	serwer WWW; tu kontroler Ingress oparty na nim
OCI	Open Container Initiative: standard obrazu i środowiska kontenera
OS	Operating System: system operacyjny
OSCON	konferencja wolnego oprogramowania
PA-RISC	architektura procesorów HP 9000
PaaS	Platform as a Service: platforma aplikacyjna w chmurze
PDF	Portable Document Format
PID	Process ID: numer procesu; też przestrzeń nazw procesów
POSIX	norma interfejsu systemów uniksowych
QEMU	emulator i monitor maszyn wirtualnych
R&D	research and development: badania i rozwój
RAFT	algorytm uzgadniania stanu w klastrze
RAM	Random Access Memory: pamięć operacyjna
README	plik z opisem projektu
RELEASE	człon nazw wydań FreeBSD (4.0-RELEASE)
RHEL	Red Hat Enterprise Linux
SCCS	Source Code Control System: system kontroli wersji, z którego logów odczytano datę chroot w BSD

Skrót	Co znaczy
SCO	firma od UNIX-ów na PC, w tytule strony o komercyjnych UNIX-ach
SELinux	Security-Enhanced Linux: obowiązkowa kontrola dostępu w jądrze
SG24-7431-00	numer Redbooka IBM o WPAR
SIG	Special Interest Group: grupa robocza Kubernetesa
SUPERNAP	człon nazwy Switch SUPERNAP, jednego z założycieli CNCF
SUSE	niemiecka firma linuksowa, od 2020 właściciel Ranchera
UBI	Universal Base Image: obrazy bazowe Red Hata
UID / GID	identyfikator użytkownika i grupy
UNIX	system z Bell Labs i jego rodzina
US	United States: tu PyCon US
USD	dolar amerykański
UTS	UNIX Time-sharing System: przestrzeń nazwy hosta
VE	virtual environment: środowisko wirtualne Virtuozzo
VM	virtual machine: maszyna wirtualna; mikro-VM — jej lekka odmiana
VNET	wirtualny stos sieciowy jaili FreeBSD
VP	Vice President: wiceprezes
WPAR	Workload Partition: kontenery AIX 6.1
WSL	Windows Subsystem for Linux
YAML	format plików konfiguracyjnych, m.in. manifestów Kubernetesa

□□ Leksykon: nazwy i pojęcia z tej strony (53)

Nazwa	Co to jest
Bitnami	katalog gotowych obrazów i chartów; od IX 2025 wersjonowane obrazy płatne
bootc	narzędzie Red Hata do systemu w postaci obrazu kontenera
Borg	wewnętrzny system zarządzania kontenerami Google
Buildah	narzędzie z rodziny Podmana, w module container-tools
BuildKit	silnik budowania obrazów Dockera; luki z pakietu Leaky Vessels
CAP_SYS_ADMIN	uprawnienie nazywane w dokumentacji „the new root”
capabilities	podział władzy roota na osobne uprawnienia, od Linuksa 2.2
cgroups	grupy kontrolne: liczenie i ograniczanie zasobów grup procesów

Nazwa	Co to jest
chroot	wywołanie zmieniające procesowi katalog główny, od V7
containerd	warstwa między silnikiem Dockera a runc; projekt CNCF
CoreOS	firma od Container Linux, rkt i etcd, kupiona przez Red Hata w 2018
cri-dockerd	zewnętrzny adapter Docker Engine do Kubernetesa po 1.24
crun	alternatywne wobec runc środowisko niskiego poziomu
dockershim	adapter Dockera w kubelecie, usunięty w 1.24
dotCloud	firma PaaS, z której wyrósł Docker
etcd	rozproszony magazyn klucz–wartość klastra Kubernetesa
Firecracker	monitor mikro-maszyn wirtualnych AWS w Rust
Gateway API	API Kubernetesa zalecane zamiast Ingress NGINX
gVisor	jądro w Go działające jako proces; środowisko runsc
Helm	menedżer pakietów (chartów) Kubernetesa
Incus	rozwidlenie LXD w społeczności Linux Containers
Ingress NGINX	kontroler ruchu przychodzącego, bez utrzymania od III 2026
jail	izolowane środowisko FreeBSD od wersji 4.0
k3s	lekka dystrybucja Kubernetesa Rancher Labs
Kata Containers	kontenery w lekkich maszynach wirtualnych
kubelet	agent Kubernetesa na każdym węźle
libcontainer	biblioteka Dockera w Go, potem runc
lmctfy	otwarty stos kontenerów Google, porzucony
Mesos	menedżer klastra Apache, w Attic od X 2025
Mirantis	firma, która kupiła Docker Enterprise w 2019
Moby	otwarty kod silnika Dockera od 2017
Nomad	orkiestrator HashiCorp z 2015 roku
Omega	następczyni Borga w Google
OpenShift	platforma kontenerowa Red Hata na Kubernetesie
OpenVZ	otwarty rdzeń Virtuozzo na GPL
operator	program zarządzający usługą w Kubernetesie za człowieka
Pasta	mechanizm sieci kontenerów bezrootowych Podmana
pod	najmniejsza jednostka Kubernetesa, dawny „alloc” Borga
Podman	silnik kontenerów bez demona

Nazwa	Co to jest
przestrzeń nazw	osobny widok zasobu systemu dla grupy procesów
Quadlet	mechanizm Podmana zmieniony w wersji 6.0
Rancher	platforma zarządzania Kubernetesem, od 2020 w SUSE
rkt	środowisko kontenerów CoreOS, zarchiwizowane 2019
rootless	tryb bezrootowy: kontenery bez uprawnień roota
runc	referencyjne środowisko niskiego poziomu OCI
seccomp	filtr wywołań systemowych procesu
Skopeo	narzędzie z rodziny Podmana, w module container-tools
strefy	kontenery Solarisa 10 (Zones)
Swarm	orkiestrator wbudowany w Dockera od 1.12
systemd-nspawn	proste kontenery z systemd
Tiller	komponent wcześniejszych wersji Helma, usunięty w Helmie 3
Virtuozzo	komercyjne kontenery SWsoft, od 2002
VServer	Linux-VServer: kontenery na łatanym jądrze od 2001

> Zastrzeżenie zakresu: nie jest to podręcznik Dockera ani Kubernetesa. Opisuję, skąd się wzięły mechanizmy i produkty, kto za nimi stał i za ile, oraz co z tego wynika dla administratora w 2026 roku. Polecenia podaję tylko te, które cytują same źródła — dokumentacja Red Hata, Dockera, Microsoftu i Google; składni, której źródła nie podają, tu nie ma. Ucieczki z kontenerów opisuję wyłącznie jako historię: kto, kiedy, czego dotyczyło i co pomogło — bez technicznych szczegółów ataku.

Wprowadzenie: rzecz, której nie ma w jądrze

Kiedy administrator w 2026 roku mówi „kontener”, ma zwykle na myśli kilka rzeczy naraz: plik obrazu pobrany z rejestru, proces uruchomiony z tego obrazu, polecenie `docker run` albo `podman run`, a w większej firmie — pod w klastrze Kubernetesa. Wszystkie te rzeczy łączy jedno: proces działa na zwykłym jądrze gospodarza, ale widzi własny system plików, własną sieć, własne numery procesów i własną nazwę hosta, a jądro pilnuje, ile pamięci i procesora może zużyć. Nie ma w tym emulowanego sprzętu ani drugiego systemu operacyjnego. Jest jeden system, który część swoich procesów okłamuje co do tego, jak wygląda świat.

Najciekawsze w tej definicji jest to, czego w niej brakuje: w jądrze Linuksa nie ma obiektu „kontener”. Strona podręcznika `namespaces(7)` wylicza osiem typów przestrzeni nazw — Cgroup, IPC, Network, Mount, PID, Time, User i UTS. Obok są grupy kontrolne (cgroups), które liczą i ograniczają zasoby, uprawnienia (capabilities), które dzielą władzę roota na kawałki, i seccomp, który filtruje wywołania systemowe. Kontener to umowa, że uruchamiając proces, włączy się naraz większość tych mechanizmów. Słowo „container” w jądrze zresztą było — ale oznaczało grupę procesów i w 2007 roku usunięto je z nazwy jako „too generic”, zbyt ogólne. Ta historia wraca niżej.

Obiegowa opowieść o kontenerach brzmi mniej więcej tak: „Kontenery wymyślił Docker w 2013 roku; to lekka wirtualizacja, bezpieczna jak maszyna wirtualna; Google wymyślił Kubernetesa i dał go światu; Kubernetes porzucił Dockera w 2022 roku; Docker jest dziś płatny”. Każdy człon tego zdania jest w jakiejś części prawdziwy i w jakiejś części fałszywy. Rozbieram je w tabeli pod koniec strony, a po drodze pokazuję dokumenty: strony podręczników, artykuły LWN, komunikaty prasowe firm, wpisy z blogów projektów, prace naukowe Google i ogłoszenia z listy oss-security.

Jest w tej historii jeszcze jeden wątek, ważny w księdze „Serwerownia”: pieniądze. Kontenery przez ponad dekadę były produktem, zanim stały się standardem: SWsoft sprzedawał je od 2002 roku, Sun, IBM i HP — w swoich UNIX-ach. Docker, Inc. zrobił z nich projekt otwarty, sprzedał w 2019 roku biznes dla firm i zaczął pobierać opłaty za aplikację dla programistów; Google oddał Kubernetesa fundacji, a Red Hat zbudował na nim OpenShift i kupił CoreOS.

Wreszcie: tę stronę czyta administrator, dlatego rozdział o praktyce dotyczy tego, co trzeba zrobić teraz — od cyklu wydań Kubernetesa po koniec cgroups v1, Ingress NGINX i darmowych obrazów Bitnami.

Prehistoria: chroot, jaile i serwery wirtualne

chroot: 1979 czy 1982?

Najstarszy klocek kontenera to wywołanie systemowe, które zmienia procesowi katalog główny. W podręczniku **Version 7 Unix** z 1979 roku strona `chdir(2)` nosi tytuł „chdir, chroot − change default directory” i opisuje drugie wywołanie jednym zdaniem: „Chroot sets the root directory, the starting point for path names beginning with ‘/’.” Dalej stoi zastrzeżenie: „The call is restricted to the super-user.” Proces, któremu zmieniono katalog główny, widzi tylko poddrzewo systemu plików — wszystko, co leży wyżej, przestaje dla niego istnieć jako ścieżka.

Obiegowa wersja przypisuje chroot Billowi Joyowi i datuje go na 1982 rok. Ta data też ma źródło, i to dobre: praca Poula-Henninga Kampa i Roberta Watsona o jailach FreeBSD w przypisach powołuje się na Marshalla Kirka McKusicka, który według logów SCCS ustalił, że Joy dodał wywołanie chroot

do drzewa BSD **18 III 1982**, mniej więcej półtora roku przed wydaniem 4.2BSD. Źródła podają więc dwie daty, 1979 albo 1982[1] — i obie są prawdziwe, tyle że dla różnych drzew kodu: w Version 7 z AT&T wywołanie jest udokumentowane w 1979 roku, w drzewie Berkeley pojawia się trzy lata później. Zdanie „chroot wymyślił Bill Joy w 1982 roku” przewraca się więc na stronie podręcznika V7.

Przewraca się też drugie obiegowe zdanie — że chroot powstał jako zabezpieczenie. McKusick przypuszczał, że Joyowi służył do budowania 4.2BSD z katalogu odciętego od reszty systemu, czyli do pracy nad samym systemem. Serwery FTP, przy których chroot stał się klasycznym sposobem na zamknięcie anonimowego użytkownika w jednym katalogu, pojawiły się w drzewie BSD dopiero w styczniu 1983 roku. W żadnym ze źródeł z epoki nie znalazłem śladu, że ktoś projektował chroot jako granicę bezpieczeństwa. Odcinał widok systemu plików — i nic więcej: nie dotykał procesów, sieci ani władzy roota. Na pytanie, co zrobić z rootem, odpowiedziały dopiero jaile, dwadzieścia lat później.

Jaile FreeBSD: „Confining the omnipotent root”

Tytuł pracy Kampa i Watsona dobrze opisuje problem, który rozwiązywali: „Jails: Confining the omnipotent root.” — jak zamknąć wszechmocnego roota. Jail miał zrobić to, czego chroot nie robił: odebrać rootowi wszechmoc poza wyznaczonym mu kawałkiem systemu. Zakończenie pracy mówi, że kod jaili „is included in the base system as part of FreeBSD 4.0-RELEASE, and fully documented in the jail(2) and jail(8) man-pages”, i odnotowuje „widespread deployment in particular as a vehicle for delivering ‘virtual private server’ services” — szerokie użycie zwłaszcza do sprzedawania wirtualnych serwerów prywatnych. FreeBSD 4.0 wyszło w marcu 2000 roku, trzynaście lat przed Dockerem. Dalsze dzieje jaili, z VNET włącznie, opisuje strona **FreeBSD**.

Zwróćmy uwagę na słowa „virtual private server”. Przez pierwszą dekadę kontenery nie były sposobem na pakowanie aplikacji, tylko na sprzedawanie kawałków serwera. Klient hostingu dostawał coś, co wyglądało jak własna maszyna z własnym rootem, a operator trzymał setki takich klientów na jednym fizycznym serwerze z jednym jądrem. Ta różnica w celu — konsolidacja zamiast dystrybucji oprogramowania — przewija się przez całą historię aż do 2013 roku.

Virtuozzo i OpenVZ: kontenery na łatanym jądrze

Najbogatszą udokumentowaną prehistorię linuksowych kontenerów ma projekt OpenVZ, który na swojej wiki prowadzi szczegółową oś czasu — to źródło „od producenta”, ale z datami do miesiąca. Według niej w **XI 1999** Aleksandr Tormasow zaproponował Siergiejowi Biełousowowi, przebywającemu w Singapurze, „container virtualization”. W **II 2000** przy Moskiewskim Instytucie Fizyko-Technicznym (MIPT) powstało biuro, w którym pięć osób pracowało nad makietą Virtuozzo.

W **VII 2000** ruszyła ograniczona publiczna beta: dwa serwery i, jak pisze wiki, „5000 VEs during summer” — pięć tysięcy środowisk wirtualnych w ciągu lata. We wrześniu 2000 roku Andriej Sawoczkin z Uniwersytetu Moskiewskiego zaczął pracę nad **User Beancounters**, mechanizmem rozliczania zasobów każdego środowiska — czymś, co w głównym jądrze Linuksa pojawi się jako cgroups dopiero osiem lat później.

W **I 2002** firma SWsoft wypuściła „Virtuozzo” dla Linuksa, a w lutym pierwszymi klientami zostały firmy z Doliny Krzemowej. Równolegle od grudnia 2001 roku trwały prace nad Virtuozzo dla Windows, wydanym w grudniu 2004 roku. W 2005 roku SWsoft założył projekt **OpenVZ**, żeby — jak pisze wiki — „release the core of Virtuozzo under GNU GPL”; w listopadzie 2006 roku OpenVZ dostał migrację na żywo.

OpenVZ miał jedną wadę, która zdecydowała o jego miejscu w historii: działał na mocno łatanym jądrze, poza głównym drzewem Linuksa. Ludzie z tego projektu zrobili jednak coś, co przetrwało dłużej niż sam produkt. W **VIII 2009** Parallels — tak nazywał się już wtedy SWsoft — był w pierwszej dziesiątce kontrybutorów jądra, a wiki wylicza pracę nad przestrzeniami nazw PID, IPC i sieci, „with the last one being the biggest”, przy czym ta ostatnia była największa. W lipcu 2011 roku Paweł Jemieljanow wysłał propozycję zapisu i odtwarzania stanu procesów w przestrzeni użytkownika, we wrześniu Cyrill Gorcunov zrobił pierwszy commit projektu **CRIU**, a w lipcu 2012 roku ukazała się wersja 0.1. W grudniu 2014 roku Parallels ogłosił połączenie OpenVZ i Parallels Cloud Server w jedną otwartą bazę kodu, a w 2015 roku opublikował źródła jądra opartego na RHEL 7 (3.10) i narzędzi. Firma nazywa się dziś Virtuozzo.

Moja ocena: OpenVZ jest brakującym ogniwem tej opowieści. Produkcyjne kontenery linuksowe istniały od 2002 roku, tylko nie w jądrze, które wszyscy mieli. Kiedy w styczniu 2008 roku ukazał się Linux 2.6.24 z przestrzenią nazw PID i zalążkiem sieciowej, serwis KernelNewbies wprost wskazał, że motywacją tych zmian były OpenVZ i Vserver.

Linux-VServer

Drugim takim projektem był **Linux-VServer**, który w 2001 roku założył Jacques Gélinas. Jego changelog opisuje cel jako „running several general purpose Linux server on a single box” — uruchomienie kilku ogólnych serwerów linuksowych na jednej maszynie. Podobnie jak OpenVZ, wymagał łatanego jądra. Źródłem tej informacji jest tekst Tima Hildreda z Red Hata, „The History of Containers”, z 28 VIII 2015 roku — blog producenta, dobry jako streszczenie, ale nie jako źródło pierwotne. Ten sam tekst odnotowuje m.in. łatki przestrzeni użytkowników Erica W. Biedermana z Red Hata z 2008 roku.

Komercyjne UNIX-y: strefy, WPAR i kontenery HP-UX

Słowo „container” w znaczeniu handlowym zawdzięczamy nie Dockerowi, tylko Sunowi. **15 II 2004** roku The Register (Ashlee Vance) pisał: „One of the major new additions to Solaris 10 is the N1 Grid Containers product”. Artykuł przedstawiał je jako odpowiedź Suna na partycje logiczne (LPAR) w AIX i HP-UX oraz na maszyny wirtualne VMware, należącego wtedy do EMC. Każda partycja miała mieć limity procesora, pamięci i pasma, a „users can create hundreds of partitions on top of a single instance of the operating system” — setki partycji na jednej instancji systemu. Klienci mogli je testować wcześniej w programie Software Express. Ta sama technika nazywała się kolejno „N1 Grid Containers”, „Solaris Containers” i „Zones”, czyli strefy. Sam Solaris 10 ze strefami ukazał się w styczniu 2005 roku — tę datę podaje strona **Solaris i SunOS**. Słowo „container” Sun używał w ofercie dziewięć lat przed Dockerem. Kulisy projektu „Kevlar” i pracę zespołu z konferencji LISA 2004 opisuje strona **Solaris i SunOS**.

Sun Ultra 1 z procesorem UltraSPARC i systemem Solaris 7. Strefy, sprzedawane jako N1 Grid Conta

Sun Ultra 1 z procesorem UltraSPARC i systemem Solaris 7. Strefy, sprzedawane jako N1 Grid Containers, przyszły do Solarisa dopiero w wersji 10. Źródło: oldcomputer.info, CC BY-NC-SA 3.0, autor: MCbx (oldcomputer.info).

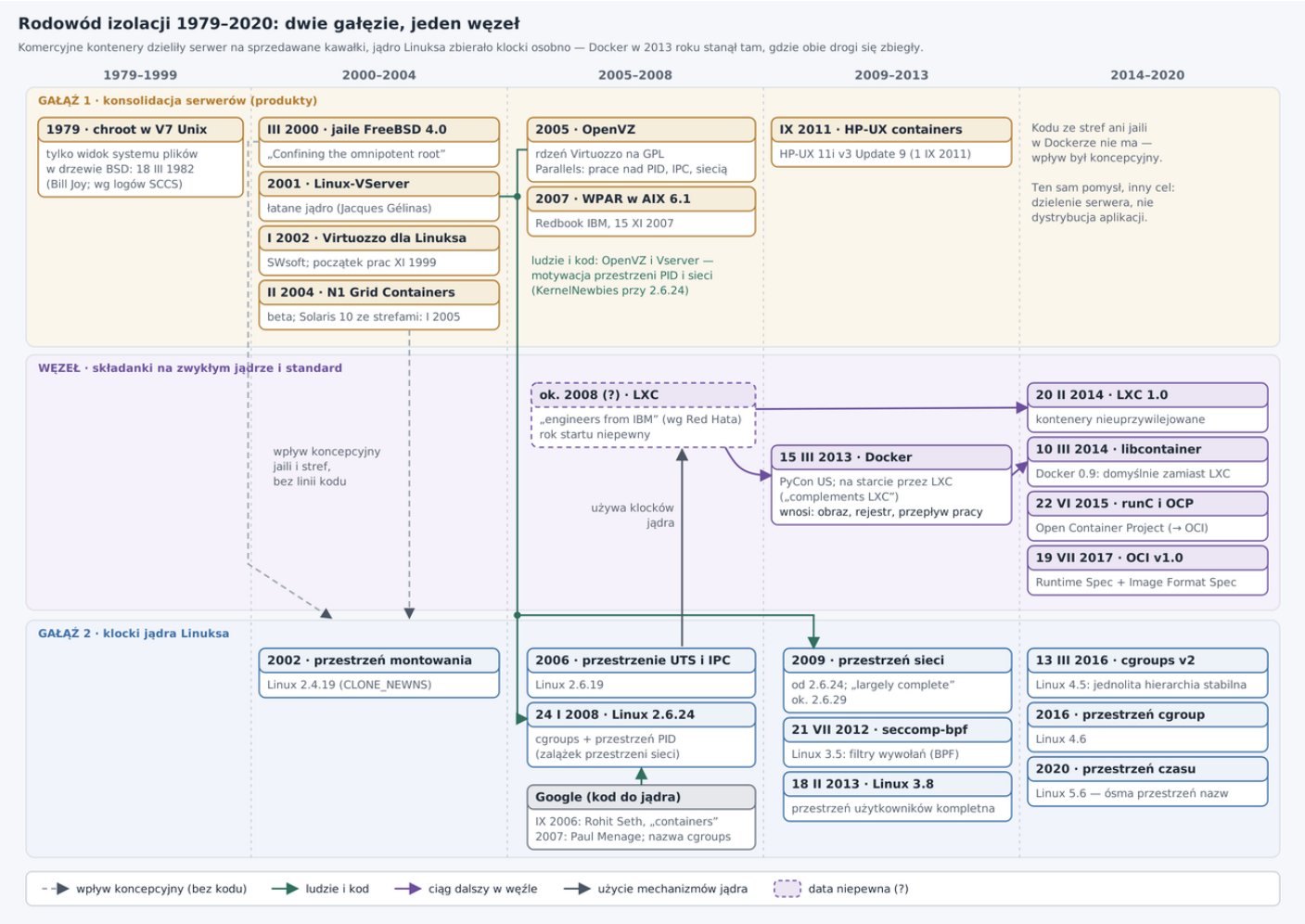
IBM odpowiedział w AIX 6.1. Redbook SG24-7431-00 „Introduction to Workload Partition Management in IBM AIX Version 6.1” z 15 XI 2007 roku przedstawia **Workload Partitions (WPAR)** jako „a set of completely new software-based system virtualization features introduced in IBM AIX Version 6.1”. Wśród cech wymienia zapis i odtwarzanie stanu bez modyfikacji aplikacji oraz „the capability to relocate a live application into a different AIX instance” — przenoszenie działającej aplikacji do innej instancji AIX. To, nad czym linuksowy świat pracował potem przez lata w CRIU, IBM sprzedawał w 2007 roku.

IBM Power Series 850 z procesorem PowerPC 604 (1995); według autora kolekcji maszyna pracowała

IBM Power Series 850 z procesorem PowerPC 604 (1995); według autora kolekcji maszyna pracowała prawdopodobnie pod AIX-em. Partycje WPAR przyszły do AIX-a dopiero w wersji 6.1, w 2007 roku. Źródło: oldcomputer.info, CC BY-NC-SA 3.0, autor: MCbx (oldcomputer.info).

HP dołączyło ostatnie. W HP-UX 11i v3 Update 9 z 1 IX 2011 roku dawne „HP-UX Secure Resource Partition” przemianowano na „HP-UX container”. Timothy Prickett Morgan w The Register opisał trzy typy: kontenery obciążeń (jedno jądro i jeden system plików), kontenery HP 9000 (aplikacje z procesorów PA-RISC uruchamiane przez emulację) i nowy kontener systemowy — „akin to a Solaris container or an AIX workload partition”, z „more of the look-and-feel of a virtual machine than a container typically does”, czyli bliższy w odczuciu maszynie wirtualnej niż typowy kontener.

Moja ocena: w latach 2004–2011 wszyscy trzej wielcy dostawcy komercyjnego UNIX-a sprzedawali „kontenery” jako konsolidację serwerów — wiele środowisk na jednej maszynie, czyli mniej sprzętu i mniej licencji. Nikt nie sprzedawał ich jako sposobu na pakowanie i przenoszenie aplikacji. Ta różnica jest kluczem do zrozumienia, co w 2013 roku zrobił Docker. Kodu ze stref czy jaili w Dockerze nie ma — wpływ był koncepcyjny, a nie przez linie kodu.



Rysunek 1. *Kontenery mają dwa rodowody, które przez dekadę biegły osobno: komercyjny, w którym dzielono serwer na sprzedawane kawałki, i jądrowy, w którym Linux kawałek po kawałku budował mechanizmy izolacji — a Docker w 2013 roku stanął w miejscu, gdzie się zbiegły.*

Klocki jądra Linuksa

Linux nie dostał kontenerów jednym ruchem, tylko kilka niezależnych mechanizmów, każdy z innym autorem i inną datą.

Przestrzenie nazw

Przestrzeń nazw daje procesowi własny widok jednego zasobu systemu. Michael Kerrisk w serii artykułów dla LWN „Namespaces in operation" (część pierwsza z 4 I 2013 roku) opisał kolejność, w jakiej pojawiały się w jądrze. Pierwsza była przestrzeń **montowania** (flaga `CLONE_NEWNS`) w Linuksie 2.4.19 z 2002 roku. W 2.6.19 (2006) doszły przestrzenie **UTS** (nazwa hosta i domeny) i **IPC** (komunikacja międzyprocesowa; kolejki komunikatów POSIX objęła dopiero od 2.6.30). W 2.6.24 przyszła przestrzeń **PID**, z wprost wskazaną motywacją: migracja kontenerów między gospodarzami z zachowaniem numerów procesów i własny proces init w każdym kontenerze.

Przestrzeń **sieci** ma dwie daty. Źródła podają, że pojawiła się w 2.6.24 (2008) albo w 2.6.29 (2009) [2] — Kerrisk wyjaśnia, że prace zaczęły się w 2.6.24, a implementacja była „largely complete”, w większości gotowa, około 2.6.29. Najdłużej trwała przestrzeń **użytkowników**: zaczęła się w 2.6.23, a ukończona została w 3.8, o którym Kerrisk pisze: „the implementation of user namespaces is now functionally complete”. Praca trwała około pięciu lat. Jej skutek był rewolucyjny: „unprivileged processes can create user namespaces in which they have full privileges” — zwykły proces może utworzyć przestrzeń, w której ma pełne uprawnienia. Linux 3.8 wyszedł 18 II 2013 roku, a KernelNewbies podsumował go jako „the ability for unprivileged users to use per-process namespaces safely”. Miesiąc później świat zobaczył Dockera.

Strona podręcznika `namespaces(7)` zamyka listę: osiem typów — Cgroup, IPC, Network, Mount, PID, Time, User i UTS. Przestrzeń cgroup doszła w Linuksie 4.6 (2016), przestrzeń czasu — w 5.6 (2020). Ta sama strona zawiera dwa zdania, które warto znać przed rozmową o bezpieczeństwie: „Creation of new namespaces… in most cases requires the CAP_SYS_ADMIN capability” oraz „User namespaces are the exception: since Linux 3.8, no privilege is required to create a user namespace.”

Przestrzeń nazw	Flaga	Jądro	Co izoluje
Mount	`CLONE_NEWNS`	2.4.19 (2002)	punkty montowania, czyli widok systemu plików
UTS	`CLONE_NEWUTS`	2.6.19 (2006)	nazwę hosta i domeny
IPC	`CLONE_NEWIPC`	2.6.19 (2006); kolejki POSIX od 2.6.30	komunikację międzyprocesową
PID	`CLONE_NEWPID`	2.6.24 (24 I 2008)	numery procesów, własny init
Network	`CLONE_NEWNET`	od 2.6.24, „largely complete” ok. 2.6.29[2]	interfejsy, adresy, tablice routingu
User	`CLONE_NEWUSER`	od 2.6.23, kompletna w 3.8 (18 II 2013)	identyfikatory użytkowników i grup, uprawnienia
Cgroup	—	4.6 (2016)	widok hierarchii grup kontrolnych
Time	—	5.6 (2020)	zegary systemowe

Grupy kontrolne: kiedy „container” było za ogólne

Druga połowa kontenera to rozliczanie zasobów. W tej części najważniejszym graczem był Google. We wrześniu 2006 roku Rohit Seth z Google wysłał pierwszą łatkę o nazwie „containers”, opisaną przez LWN. W maju 2007 roku rozwój prowadził już Paul Menage, również z adresem google.com; Jonathan Corbet w LWN z 29 V 2007 roku opisywał wersję „containers V10”, która przerabiała

istniejący mechanizm cpusets na ogólny „subsystem”. Definicja z tamtego tekstu jest dziś zabawna: „A "container" is a group of processes which shares a set of parameters used by one or more subsystems.” — kontener to grupa procesów dzielących parametry.

Pięć miesięcy później, w artykule „Notes from a container” z 29 X 2007 roku, Corbet odnotował zmianę nazwy: „The original "containers" name was considered to be too generic” — i mechanizm stał się „control groups”, grupami kontrolnymi. W tym samym tekście padają zdania, które dobrze opisują ówczesny stan wiedzy: kontenery „lack some of the flexibility of full virtualization, but they tend to be quite a bit more efficient” i „in many ways, containers are harder to implement than virtualization is” — w wielu sprawach trudniej je zrobić niż wirtualizację. Corbet zauważył też drobiazg techniczny o dużej wymowie: w 32-bitowym słowie flag `CLONE_` zostały tylko dwa wolne bity. Przestrzenie nazw dosłownie kończyły miejsce.

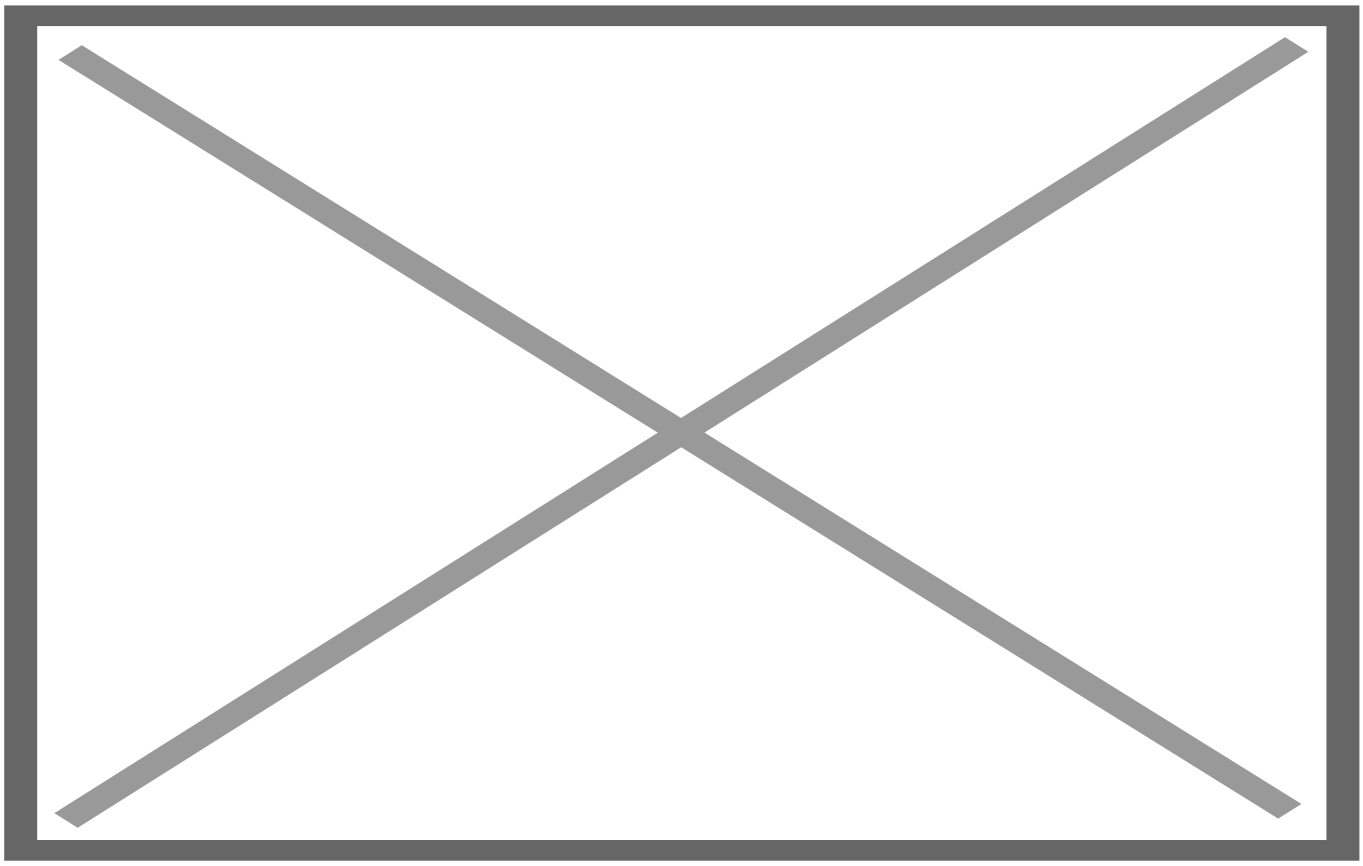
Grupy kontrolne weszły do Linuksa 2.6.24, wydanego **24 I 2008**. KernelNewbies opisuje je jako mechanizm, który pozwala „track and group processes into arbitrary "cgroups" and assign arbitrary state to those groups”, i wymienia pierwszych użytkowników: cpusets i grupowe szeregowanie CFS. W tym samym wydaniu przyszły przestrzeń PID i zalążek sieciowej — 2.6.24 jest więc pierwszym jądrem, w którym klocki kontenera leżą obok siebie.

Druga wersja, **cgroups v2** z jedną, ujednoliconą hierarchią, stała się stabilna w Linuksie 4.5 z 13 III 2016 roku. KernelNewbies: „the unified hierarchy is considered stable, and it's no longer hidden behind that developer flag”. Montuje się ją jako system plików `cgroup2`. Na starcie miała tylko kontrolery pamięci i wejścia-wyjścia, bez kontrolera procesora. Ta wersja wraca w rozdziale o praktyce — w 2026 roku przestała być opcją.

Uprawnienia i seccomp

Trzeci klocek jest najstarszy. Strona podręcznika `capabilities(7)`: „Starting with Linux 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled.” Od 2.2 władza roota jest więc zbiorem osobnych uprawnień, które kontener może dostać w części. Kłopot w tym, że jedno z nich obejmuje prawie wszystko. O `CAP_SYS_ADMIN` ta sama strona pisze, że „this capability is overloaded” i że „It can plausibly be called "the new root"” — można je zasadnie nazwać nowym rootem. Kto uruchamia kontener z tym uprawnieniem, oddaje mu dużą część gospodarza.

Czwarty klocek to filtr wywołań systemowych. W Linuksie 3.5 z 21 VII 2012 roku seccomp, według KernelNewbies, „has evolved into a filtering mechanism that allows processes to specify an arbitrary filter of system calls” — proces może wgrać program BPF, który decyduje, które wywołania wolno mu wykonać. Używała go piaskownica wtyczek w Chromium, a systemd dał administratorom dyrektywę `SystemCallFilter` w plikach unitów. Starszy, ścisły tryb seccomp istniał w jądrze wcześniej; autorów obu trybów w tej pracy nie sprawdzałem.



Animacja 1. *Składanie kontenera krok po kroku. Kontener nie jest obiektem jądra, tylko zwykłym procesem z włączonymi przestrzeniami nazw, limitami i filtrami — dlatego z zewnątrz pozostaje procesem gospodarza, a granica jest tak mocna jak jądro i runc.*

LXC i systemd-nspawn: pierwsze składanki

Kiedy klocki leżały już w jądrze, ktoś musiał je złożyć. Najbardziej znanym projektem, który robił to w przestrzeni użytkownika na zwykłym jądrze, było **LXC** (Linux Containers). Według Red Hata zaczęli je około 2008 roku „engineers from IBM”; nazwiska Daniela Lezcano i Serge’a Hallyna, które się przy tym podaje, znalazłem tylko w Wikipedii. Wersja 1.0 ukazała się **20 II 2014** roku. Ogłosił ją Stéphane Graber z Canonicala, który podsumował „10 months of work”, „60 or so contributors” i ponad tysiąc zmian. Najważniejsze nowości to kontenery nieuprzywilejowane („unprivileged containers”), możliwe dzięki przestrzeni użytkowników z Linuksa 3.8, oraz interfejs programistyczny w kilku językach. Graber zapowiadał „a stable release which we intend to support for a long time”.

Drugą składanką był **systemd-nspawn**. Jego strona podręcznika mówi o nim skromnie: „systemd-nspawn may be used to run a command or OS in a lightweight namespace container. In many ways it is similar to chroot(1), but more powerful since it virtualizes the file system hierarchy…” — ale w tym samym dokumencie stoi ostrzeżenie, które powinno wisieć nad każdym biurkiem: „This sandbox can easily be circumvented from within the container if user namespaces are not used.” oraz „untrusted code must always be run in a user namespace”. Twórcy narzędzia piszą wprost, że bez przestrzeni użytkowników jego piaskownicę łatwo obejść.

Dalsze losy LXC to osobna, krótka historia. Z LXC wyrósł **LXD**, menedżer kontenerów systemowych rozwijany przez Canonicala w ramach społeczności Linux Containers. **4 VII 2023** roku Graber napisał: „Canonical has requested that the LXD project be moved from LinuxContainers to Canonical's own infrastructure” — po ponad ośmiu latach w społeczności repozytorium, strona i CI przeszły do Canonicala. **7 VIII 2023** roku społeczność Linux Containers przyjęła **Incusa**, rozwidlenie LXD założone przez Aleksę Saraia po „Canonical's decision to take LXD away from Linux Containers”. Jego opiekunami zostali Aleksa Sarai, Christian Brauner, Serge Hallyn, Stéphane Graber i Tycho Andersen — jak pisze ogłoszenie, „the team that originally created LXD”.

Google: Borg, Omega i lmctfy

Dużą część kodu linuksowych kontenerów napisała firma, która przez dekadę nie mówiła publicznie, do czego go używa. W artykule „Borg, Omega, and Kubernetes” w ACM Queue (tom 14, nr 1, styczeń–luty 2016) Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer i John Wilkes piszą, że Google „contributed much of the container code to the Linux kernel” — po to, by oddzielić usługi interaktywne, w których liczy się czas odpowiedzi, od zadań wsadowych, które mogą poczekać. Kontenery linuksowe były w Google w użyciu na dużą skalę „for more than ten years”. Poprzednicy Borga, systemy Babysitter i Global Work Queue, „predated Linux control groups” — powstały, zanim w jądrze pojawiły się grupy kontrolne.

Pierwszy serwer produkcyjny Google z około 1999 roku — regał tanich pecetów połączonych w sieć ·

Pierwszy serwer produkcyjny Google z około 1999 roku — regał tanich pecetów połączonych w sieć — na wystawie w Computer History Museum, zdjęcie z 2006 roku. Źródło: Flickr, CC BY 2.0, autor: Steve Jurvetson.

Borg to według tego artykułu „the first unified container-management system developed at Google”, pierwszy wspólny system zarządzania kontenerami w firmie. Szczegółowo opisano go w 2015 roku: praca „Large-scale cluster management at Google with Borg” (Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune i John Wilkes, EuroSys 2015 w Bordeaux) mówi o systemie, który „runs hundreds of thousands of jobs” na klastrach liczących do dziesiątek tysięcy maszyn. Abstrakt przedstawia całość jako „lessons from a decade of operating the system”.

Pierwszy regał pamięci masowej Google, z obudową zbudowaną z klocków Duplo, w Huang Engineer

Pierwszy regał pamięci masowej Google, z obudową zbudowaną z klocków Duplo, w Huang Engineering Center na Uniwersytecie Stanforda. Źródło: Flickr, CC BY 2.0, autor: Steve Jurvetson.

Omega była następczynią, choć nie zastąpiła Borga. Opisuje ją praca „Omega: flexible, scalable schedulers for large compute clusters” (Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek i John Wilkes, EuroSys 2013 w Pradze, s. 351–364). Według artykułu z ACM Queue powstała dla poprawy inżynierii ekosystemu Borga i trzymała stan klastra w scentralizowanym magazynie transakcyjnym opartym na Paxosie. Oba systemy były, jak piszą autorzy, „purely Google-internal”

— wyłącznie wewnętrzne.

Z tego samego artykułu pochodzą dwa zdania, które warto zapamiętać, bo tłumaczą, dlaczego Kubernetes wygląda tak, a nie inaczej. Pierwsze: „In Borg, the outermost container is called a resource allocation, or alloc; in Kubernetes, it is called a pod.” Pod jest więc spadkobiercą borgowego „allocu”. Drugie dotyczy błędów: w Borgu zarządzano numerami portów, bo wszystkie zadania na maszynie dzieliły jej adres, a zadania rozróżniano numerowanymi indeksami. Kubernetes miał to naprawić, „aligning network identity (IP address) with application identity” — każdy pod dostaje własny adres IP, a zadania opisuje się etykietami. Autorzy podsumowują: „we have learned almost as many things not to do as ideas that are worth doing” — nauczyliśmy się prawie tyle samo rzeczy, których nie robić, co pomysłów wartych realizacji. Najogólniejszą myśl artykułu streszcza zdanie, że zarządzanie wokół kontenerów „shifts the ‘primary key’ of the data center from machine to application” — kluczem centrum danych przestaje być maszyna, a staje się aplikacja.

Google miał też własny, otwarty stos uruchamiania kontenerów: **lmctfy**, czyli „Let Me Contain That For You”. README projektu przedstawia go jako „the open source version of Google's container stack, which provides Linux application containers” — bibliotekę w C++ z narzędziem wiersza poleceń. Datę otwarcia (październik 2013) znalazłem tylko w Wikipedii. Ważniejsze jest zakończenie: „We are not actively developing lmctfy further and have moved our efforts to libcontainer”, „We have been collaborating with Docker over libcontainer” oraz „In future, we hope to replace the core of lmctfy with libcontainer.” Firma, która dała Linuksowi grupy kontrolne i miała dekadę doświadczenia z kontenerami, porzuciła własny stos na rzecz biblioteki napisanej przez Dockera.

dotCloud i Docker: „Until now: Docker”

Krótkie wystąpienie na PyCon 2013

Docker zadebiutował nie na konferencji o systemach, tylko na konferencji programistów Pythona. **15 III 2013** roku, w piątek, na PyCon US w Santa Clara Solomon Hykes z firmy dotCloud wygłosił krótkie wystąpienie (tzw. lightning talk) „The future of Linux Containers”. Opis w archiwum nagrań: „Solomon talks about Linux Containers which they use at dotCloud…” „but it's complicated so until now they haven't talked about it. Until now: Docker.” dotCloud był firmą od platformy hostingowej typu PaaS i sam uruchamiał w niej kontenery linuksowe; Docker był narzędziem wyjętym z jego wnętrza.

Udostępnienie kodu datuje się różnie: źródła podają 15 III 2013 (wystąpienie) albo okolice 17 III 2013 (publikacja projektu)[3]. Strona projektu, cytowana przez serwis Changelog, opisywała Dockera jako narzędzie, które „complements LXC with a high-level API which operates at the

process level” — uzupełnia LXC o interfejs wysokiego poziomu działający na poziomie procesów. Jako zastosowania wymieniała wdrożenia aplikacji webowych, klastry baz danych, ciągłe wdrażanie i prywatny PaaS. Zwróćmy uwagę na słowo „complements”: Docker w 2013 roku nie udawał, że wynalazł kontenery. Uruchamiał je przez LXC.

Co więc wniósł? Moja odpowiedź, oparta na tym, co opisują źródła: **obraz, rejestr i przepływ pracy**. Obraz — plik z całym systemem plików aplikacji, który buduje się raz i uruchamia gdziekolwiek. Rejestr — miejsce, z którego obraz się pobiera jednym poleceniem. Przepływ pracy — jedno narzędzie dla programisty i administratora. Klocki jądra istniały; Docker zrobił z nich sposób na dystrybucję oprogramowania, a nie na dzielenie serwera. To jest właśnie ta zmiana celu, o której była mowa przy Sunie i HP.

Docker, Inc. i liczby z października 2013

Tempo było takie, że po siedmiu miesiącach firma zmieniła nazwę na nazwę narzędzia. **29 X 2013** roku dotCloud stał się **Docker, Inc.**, a platforma PaaS — produktem pobocznym. Ben Golub, dyrektor generalny, tłumaczył to zdaniem „your best customers outgrow you” — najlepsi klienci wyrastają z ciebie. Komunikat firmy, cytowany przez Richarda Serotera w InfoQ, podawał liczby: „Over 140,000 containers downloaded”, ponad 6700 gwiazdek na GitHubie i 800 rozwidleń, „Almost 200 contributors, 92 percent of whom don't work for Docker, Inc.”, i ponad 150 projektów zbudowanych na silniku. To liczby producenta, ale skala tempa jest wyraźna: w siedem miesięcy niszowe narzędzie z wnętrza firmy hostingowej zebrało prawie dwustu współautorów, z których ponad dziewięciu na dziesięciu pracowało gdzie indziej.

Docker 0.9: rozwód z LXC

10 III 2014 roku, niecały rok po debiucie, Docker 0.9 przestał domyślnie korzystać z LXC. Solomon Hykes ogłosił nowy sterownik wykonawczy **libcontainer**, „a pure Go library which we developed to access the kernel's container APIs directly, without any other dependencies” — bibliotekę w czystym Go, która rozmawia z mechanizmami jądra bezpośrednio. „Docker out of the box can now manipulate namespaces, control groups, capabilities, apparmor profiles, network interfaces and firewalling rules… without depending on LXC”. Uzasadnienie zmiany domyślnego sterownika było praktyczne: „libcontainer delivered such a boost to stability that we decided to make it the default.” LXC zostało jako opcja; ogłoszenie podawało, jak do niego wrócić — uruchamiając demona poleceniem `docker -d -e lxc` (składnia z 2014 roku, dziś historyczna).

Docker przestał zależeć od cudzego projektu, a libcontainer zaczął przyciągać innych, z Google na czele; piętnaście miesięcy później, jako runC, trafił do fundacji.

Docker 1.0 i czerwcowy tydzień 2014

Wersja 1.0 ukazała się w poniedziałek **9 VI 2014** roku, na pierwszej konferencji DockerCon w San Francisco. Jack Clark z The Register pisał, że etykieta 1.0 miała oznaczać „feature completeness, backward compatibility and API stability to meet enterprise IT standards”, a do tego firma zapowiadała „complete documentation, training programs, professional services, and enterprise support”. Drobiazg dla miłośników prasy: ten sam tekst przekręca nazwisko twórcy na „Solomon Sykes”. Dzień później, na tej samej konferencji, Google ogłosił Kubernetesa — ale to już osobny rozdział.

Zakończenie DockerCon 2016 z ekipą Dockera (podpis autora zdjęcia). Pierwsza edycja tej konferen

Zakończenie DockerCon 2016 z ekipą Dockera (podpis autora zdjęcia). Pierwsza edycja tej konferencji, w czerwcu 2014 roku, przyniosła Dockera 1.0. Źródło: Flickr, CC BY-SA 2.0, autor: Stephen R. Walli.

Wojna formatów: CoreOS, rkt i OCI

„Fundamentally flawed”

Pierwszy poważny bunt przeciw Dockerowi wyszedł od firmy, która była jego entuzjastą. **CoreOS**, założony w 2013 roku, robił lekką dystrybucję Linuksa z automatycznymi aktualizacjami, znaną potem jako Container Linux. **1 XII 2014** roku ogłosił własne środowisko uruchomieniowe **Rocket** (później **rkt**) i specyfikację **App Container** (appc). Alex Polvi z CoreOS tłumaczył to Fredericowi Lardinois z TechCrunch w ostrych słowach: „We thought Docker would become a simple unit that we can all agree on”, ale Docker „is not becoming the simple composable building block we had envisioned”. Funkcje Dockera były według niego „all compiled into one monolithic binary running primarily as root on your server”, a jego model procesów — „fundamentally flawed”, z gruntu wadliwy. Tytuł artykułu TechCrunch powtarzał to ostatnie sformułowanie.

Open Container Initiative

22 VI 2015 roku Linux Foundation ogłosiła **Open Container Project**, wkrótce przemianowany na **Open Container Initiative** (OCI). Wśród członków byli m.in. AWS, Docker, Google, Microsoft, Red Hat i CoreOS. Docker wniósł format obrazu i kod środowiska uruchomieniowego: libcontainer stał się projektem **runC**. Opiekunami zostali ludzie z obu obozów — od appc (CoreOS, Red Hat) i od libcontainer. Zakres był celowo wąski: tylko poziom pojedynczego kontenera, bez orkiestracji. LWN odczytał całość jako ugodę Dockera z CoreOS.

Specyfikacje w wersji 1.0 ukazały się **19 VII 2017** roku: **Runtime Specification**, opisująca cykl życia kontenera, i **Image Format Specification**, opisująca obraz. Inicjatywa liczyła wtedy „over 40 member organizations”, a Chris Aniszczyk z Linux Foundation nazwał wydanie „a huge milestone”. Od tej chwili „obraz Dockera” jest w praktyce obrazem OCI, który uruchomi każde

zgodne środowisko, od runc po `runsc` z gVisora.

containerd, rkt i zakup CoreOS

Wewnątrz Dockera trwało równolegle rozbieranie monolitu, na który narzekał Polvi. **containerd**, według CNCF „born at Docker in 2014”, to warstwa, która siedzi między silnikiem Dockera a runc. W marcu 2017 roku Cloud Native Computing Foundation przyjęła do inkubacji jednocześnie containerd Dockera i rkt CoreOS-a. Dalsze losy obu projektów rozeszły się radykalnie. containerd **28 II 2019** roku uzyskał status „graduated” jako piąty projekt CNCF, po Kubernetesie, Prometheusie, Envoyu i CoreDNS. rkt CNCF zarchiwizowała **16 VIII 2019** roku z uzasadnieniem, że „end user adoption has severely declined” i „user adoption has trended away from rkt towards these other projects” — czyli ku containerd i CRI-O. Do tego dochodził spadek aktywności i niezałatane podatności.

CoreOS jako firma nie dotrwał do tej chwili samodzielnie. **30 I 2018** roku Red Hat ogłosił jego zakup za **250 mln USD**. Komunikat wymieniał produkty: Tectonic (Kubernetes dla firm), Container Linux („lightweight Linux distribution… that automates software updates”), etcd („the distributed data store for Kubernetes”) i rkt, który „helped drive” standard OCI. CoreOS był też określony jako „a leading contributor” Kubernetesa.

Moja ocena: CoreOS przegrał produktem i wygrał ideą. rkt zniknął, ale bunt z 2014 roku wymusił OCI, rozbicie Dockera na wymienne kawałki i świat, w którym środowiska uruchomieniowe są wymienne. A **etcd**, rozproszony magazyn klucz–wartość z CoreOS-a, jest dziś sercem każdego klastra Kubernetesa. Do tego dochodzi wzorzec operatorów, o którym niżej.

Moby: kiedy Docker przestał być nazwą kodu

18 IV 2017 roku na DockerCon w Austin Solomon Hykes ogłosił **Moby Project**: „a library of components, a framework for assembling them into custom container-based systems”. Porównanie z ogłoszenia: „Think of Moby as the ‘Lego Club’ of container systems” — Moby jest klubem klocków Lego dla systemów kontenerowych, przeznaczonym „for system builders… not for application developers”. Docker miał używać Moby jako „an open R&D lab”, a „all our open source collaboration will move to the Moby project”. Jednocześnie firma zapewniała: „Docker is, and will remain, a open source product… staying exactly the same from a user's perspective.” Wcześniej z Dockera wydzielono już runc, HyperKit, VPNKit, SwarmKit, InfraKit i containerd.

Dla administratora skutek jest głównie nazewniczy, ale potrafi zmylić. Od 2017 roku „Docker” to marka i produkt firmy, a „Moby” — otwarty kod silnika w repozytorium `moby/moby`. W Fedorze obok pakietu `docker` występował pakiet `moby-engine`, i to rozróżnienie miało znaczenie w biuletynie bezpieczeństwa — przykład niżej, przy CVE-2019-5736. HyperKit, hipernadzorca Docker Desktop na Macu, wywodzi się z bhyve przez xhyve; ten rodowód opisuje strona **FreeBSD**.

Kubernetes: sternik z Google

Pierwszy commit i ogłoszenie na DockerCon

Publiczne repozytorium Kubernetesa zaczyna się **6 VI 2014** roku. Autor wpisu „4 Years of K8s” na blogu projektu z 6 VI 2018 roku pisze w pierwszej osobie: „On June 6, 2014 I checked in the first commit of what would become the public repository for Kubernetes.” Wpis przypisuje się zwykle Joemu Bedzie, ale w pobranym tekście nazwiska nie było i tego nie potwierdziłem. Ten sam autor dodaje: „Kubernetes was built on ideas that had been proven out at Google over the previous ten years with Borg.” Wpis na dziesięciolecie projektu z 6 VI 2024 roku podaje rozmiar pierwszego commitu: „250 files and 47,501 lines of go, bash and markdown” — 250 plików i 47 501 linii.

Cztery dni później, **10 VI 2014** roku, drugiego dnia pierwszego DockerCon, Eric Brewer, wiceprezes Google ds. infrastruktury, wygłosił wystąpienie i opublikował wpis „An update on container support on Google Cloud Platform”. Padły w nim zdania: „Everything at Google, from Search to Gmail, is packaged and run in a Linux container.” oraz „Each week we launch more than 2 billion container instances across our global data centers.” — ponad dwa miliardy instancji kontenerów tygodniowo, według danych Google. Kubernetes był w tym wpisie „a lean yet powerful open-source container manager that deploys containers into a fleet of machines”. Nazwę wyjaśniono od razu: to greckie słowo oznaczające sternika (helmsman).

Scott Johnston, w 2024 roku dyrektor generalny Dockera, wspominał po dziesięciu latach z dokładnością do sekundy: „On June 10th, 2014, day 2 of the first ever DockerCon, 16:04 seconds into his keynote speech, Google VP of Infrastructure Eric Brewer announced that Google was releasing… Kubernetes.” Zapis „16:04” to minuty i sekundy: Kubernetesa ogłoszono w szesnastej minucie przemówienia. Docker 1.0 i Kubernetes zostały więc ogłoszone na tej samej konferencji, dzień po dniu. Johnston wylicza też, co jeszcze pokazano w tych dwóch dniach: według niego Apache Mesos, GearD Red Hata, Docker Libswarm i Tupperware Facebooka. „This triggered what later became known by some as ‘the Container Orchestration War.’” — tak zaczęła się wojna orkiestratorów.

Wersja 1.0 i nowa fundacja

21 VII 2015 roku Google ogłosił Kubernetesa 1.0, gotowego „for production”, i przekazał go nowo tworzonej **Cloud Native Computing Foundation** działającej pod Linux Foundation. TechCrunch wymienił założycieli obok Google: Box, Cisco, CoreOS, Cycle Computing, Docker, eBay, Goldman Sachs, IBM, Intel, Joyent, Mesosphere, Twitter, Switch SUPERNAP, Univa, VMware i Weaveworks. Ten sam tekst zawiera błąd, dobry jako przestroga przy korzystaniu z prasy: pisze, że Kubernetes został uruchomiony „last February”, w lutym, choć pierwszy commit i ogłoszenie przypadły na

czerwiec 2014 roku[4]. Że 1.0 ogłoszono na konferencji OSCON, często się powtarza, ale tego nie potwierdziłem.

Przekazanie było zapowiedzią, a formalności trwały dłużej. Kiedy **6 III 2018** roku Kubernetes jako pierwszy projekt CNCF uzyskał status „graduated” (ogłoszony na Open Source Leadership Summit w Sonomie), Chris Aniszczyk przypomniał, że komitet techniczny fundacji przyjął go jako pierwszy projekt „a little over two years ago” — czyli na początku 2016 roku. Źródła podają więc jako datę wejścia do CNCF lipiec 2015 (ogłoszenie przekazania) albo początek 2016 (przyjęcie przez komitet) [5].

Od stu do pięciu tysięcy węzłów

Kubernetes 1.0 nie był Borgiem. Timothy Prickett Morgan w The Next Platform z 15 IX 2015 roku, opierając się na wpisie o skalowalności napisanym przez Wojciecha Tyczyńskiego z Google, podawał praktyczny limit wersji 1.0: „about 100 nodes, and with about 30 Docker container pods per node”. Cel na kolejne wydania brzmiał „1,000 nodes and 30,000 pods”. Dla porównania komórka Borga liczyła średnio około 10 000 węzłów. Droga od stu węzłów do pięciu tysięcy była pracą grupy roboczej ds. skalowalności — o jej ludziach jest w Wątku polskim.

Artykuł z ACM Queue podkreśla: „Kubernetes is open source—a contrast to Borg and Omega, which were developed as purely Google-internal systems”. Napisano go od nowa, w Go i od pierwszego dnia na Dockerze. Motyw komercyjny — przyciągnięcie klientów do chmury Google — jest moim wnioskiem, nie zapisem dokumentów.

Wojna orkiestratorów (2014–2019) i jej ofiary

W latach 2014–2019 o to, kto będzie rozmieszczał kontenery na flocie maszyn, walczyło kilka projektów naraz. Każdy miał inne zalety; zwycięzca był jeden.

Apache Mesos wcześnie przyjęli „Twitter, Apple, Yelp, Uber, and Netflix”, jak przypomniał Jack Wallen w The New Stack w kwietniu 2021 roku. Firma Mesosphere zbudowała na nim komercyjny DC/OS. W 2019 roku Mesosphere „replatformed DCOS on Kubernetes” i zmieniła nazwę na D2IQ. W kwietniu 2021 roku w projekcie Apache odbyło się głosowanie o przeniesieniu Mesosa do Attic — „strych” fundacji, na który trafiają projekty wycofane — odwołane przez przewodniczącego Vinoda Kone. Odroczenie nie trwało długo: według strony Apache Attic projekt wycofano w sierpniu 2025 roku, a przeniesienie „completed in October 2025”.

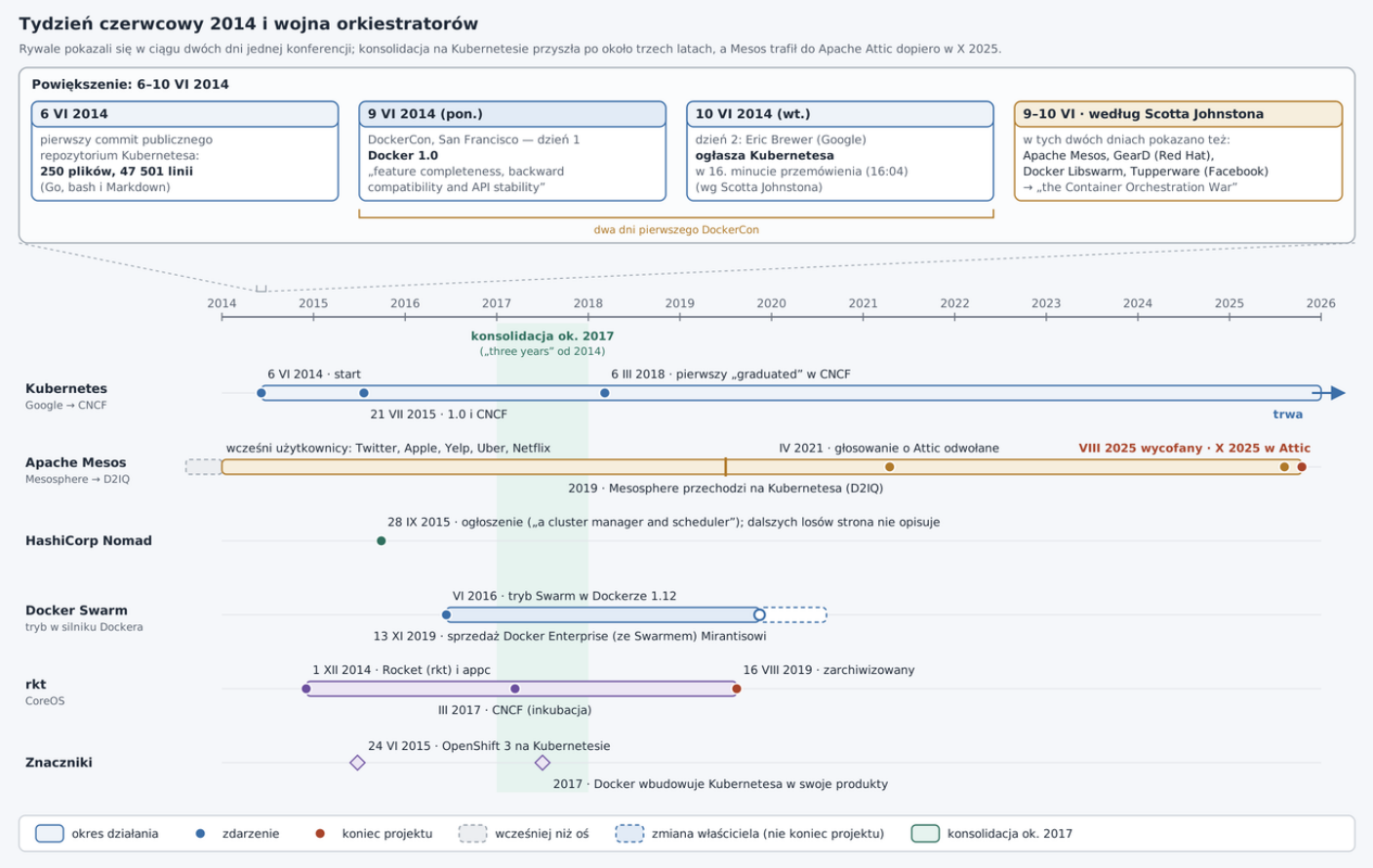
HashiCorp Nomad pojawił się **28 IX 2015** roku jako „a cluster manager and scheduler designed for microservices and batch workloads”. Armon Dadgar w ogłoszeniu wprost powoływał się na idee Borga i Omegi. Nomad był celowo prosty — „a single binary”, bez zewnętrznych usług — i

obsługiwał nie tylko Dockera, ale też QEMU, Javę i zwykłe procesy.

Najgroźniejszym przeciwnikiem był sam Docker. W czerwcu 2016 roku Docker 1.12 wbudował **tryb Swarm** w silnik: węzły dołączały do klastra, a silnik uruchamiał wbudowany rozproszony magazyn konfiguracji oparty na algorytmie RAFT. Hykes tłumaczył, że osobny magazyn konfiguracji był „a significant barrier to deployment for many users”. Josh Berkus z Red Hata pisał w LWN 13 VII 2016 roku, że Docker osiąga „near-parity on orchestration”, co „puts a lot of pressure on projects like Mesos and Kubernetes”.

Ta presja nie wystarczyła. Johnston w tekście na dziesięciolecie pisze: „Fast forward three years and the community had consolidated on Kubernetes for the orchestration layer and Docker (powered by containerd) for the container format…” i przyznaje: „In 2017, Docker integrated Kubernetes in its desktop and server products”. Docker sam wbudował zwycięzcę w swoje produkty. W listopadzie 2019 roku sprzedał biznes enterprise — razem ze Swarmem w wersji dla firm — Mirantisowi.

Dlaczego wygrał Kubernetes? Moja ocena: nie dlatego, że był najprostszy — Swarm był prostszy, Nomad lżejszy. Wygrał, bo był neutralny (w fundacji od 2015 roku, a nie w rękach jednego dostawcy), rozszerzalny (własne typy zasobów i operatorzy pozwalały dopisać do niego cokolwiek) i wspierany przez wszystkich dostawców chmur naraz. Żaden z rywali nie miał wszystkich trzech cech jednocześnie.



Rysunek 2. *Wojna orkiestratorów zaczęła się w ciągu dwóch dni czerwca 2014 roku na jednej konferencji i w praktyce rozstrzygnęła się po trzech latach — choć ostatnia ofiara, Mesos, trafiła na*

strych fundacji Apache dopiero jesienią 2025 roku.

Docker, Inc.: jak zarabiać na standardzie

Docker zmienił branżę, a potem kilka razy zmieniał pomysł na to, jak na tym zarabiać. Danych o przychodach firmy nie mam, ale trzy daty dobrze pokazują zmianę kursu: sprzedaż części firmy, płatna aplikacja dla programistów i nowy cennik.

13 XI 2019: dwie firmy w jednej

13 XI 2019 roku Docker ogłosił sprzedaż biznesu enterprise firmie **Mirantis**; ceny nie ujawniono. Nowym dyrektorem generalnym został Scott Johnston, dotąd szef produktu, a firma dostała 35 mln USD od dotychczasowych inwestorów, Benchmark Capital i Insight Partners, w ramach „recapitalization”. Rob Bearden, cytowany w komunikacie, opisał problem tak: Docker był „two very distinct and different businesses: one an active developer business, and the other a growing enterprise business” — dwiema bardzo różnymi firmami, jedną dla programistów, drugą dla przedsiębiorstw. TechCrunch dopowiadał szczegóły: Mirantis, zatrudniający około 450 osób, przejmował około 300 pracowników Dockera i „about 700 new enterprise customers”. Autor tekstu oceniał, że cena była „nowhere near Docker's valuation during any of its last funding rounds” — daleko od wyceny z ostatnich rund finansowania; to opinia dziennikarza, nie liczba.

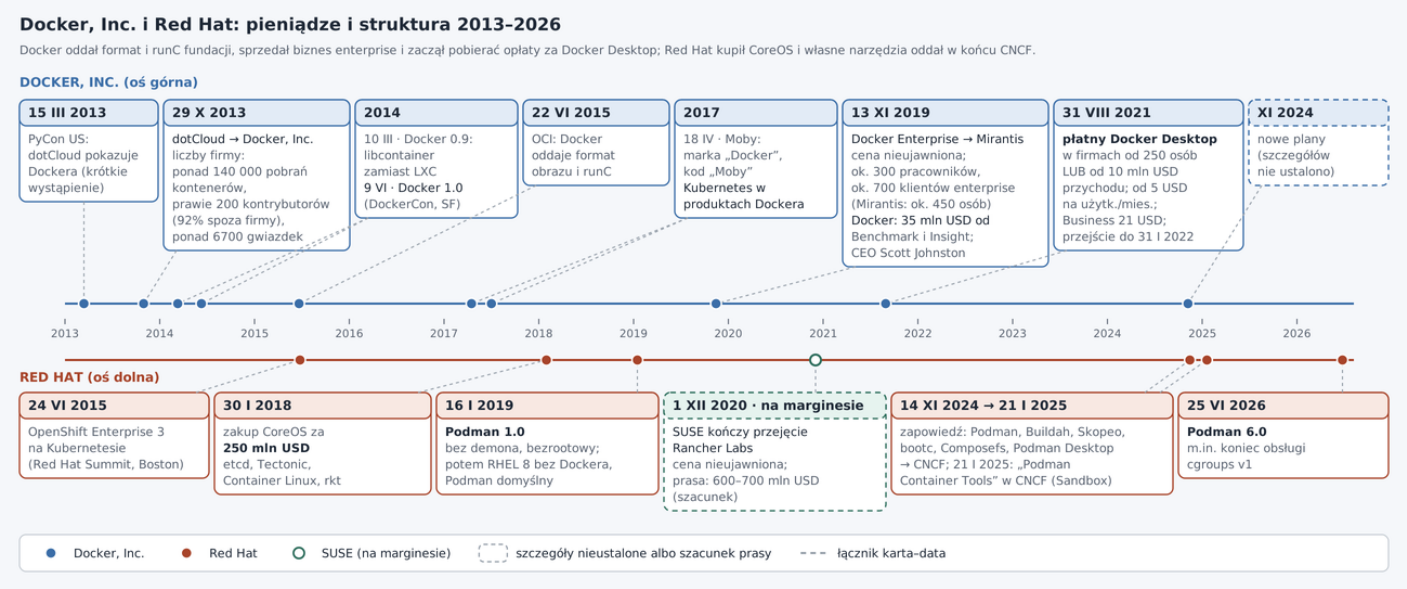
31 VIII 2021: płatny Docker Desktop

Druga część — biznes dla programistów — dostała model płatny. **31 VIII 2021** roku Docker zmienił warunki: „for use in larger businesses Docker Desktop will require a Pro, Team or Business paid subscription, starting at $5 per user per month. It remains free for small businesses (fewer than 250 employees AND less than $10 million in annual revenue), as well as for personal use, education and non-commercial open source projects.” Okres przejściowy trwał do 31 I 2022 roku, a plan Business kosztował 21 USD na użytkownika miesięcznie. Johnston uzasadniał: „This update enables our business to sustainably scale…”. W listopadzie 2024 roku Docker zmienił cennik ponownie; nowych planów w szczegółach nie sprawdzałem, więc ceny z 2021 roku traktuję jako historyczne.

Ważna dla administratora jest precyzja. Komunikat dotyczy **Docker Desktop** — aplikacji na Windows, macOS i Linuksa, która uruchamia silnik w maszynie wirtualnej. Silnika na serwerze komunikat nie wymienia. Licencji samego silnika (powszechnie podaje się Apache 2.0) w tej pracy nie sprawdziłem, więc nie powtarzam jej jako faktu — ale zdanie „Docker jest płatny” w odniesieniu do serwera z `dockerd` nie ma oparcia w warunkach z 2021 roku. Jak Docker Desktop działa na

Windows na zapleczu WSL 2, opisuje strona **Windows Subsystem for Linux**.

Moja ocena: orkiestrację przejął Kubernetes, sam silnik rozszedł się na containerd i runc oddane fundacjom, a jedyna płatna część oferty, o której mówią przytoczone źródła, to aplikacja dla programistów.



Rysunek 3. *Docker oddał standard, sprzedał biznes dla przedsiębiorstw i zaczął pobierać opłaty za aplikację dla programistów; Red Hat zbudował na tym samym standardzie platformę, kupił CoreOS i własne narzędzia oddał w końcu fundacji.*

Red Hat: Podman, OpenShift i świat bez demona

OpenShift 3: Kubernetes przed Kubernetesem 1.0

Red Hat związał się z Kubernetesem bardzo wcześnie. Jego platforma **OpenShift** działała jako publiczny PaaS (OpenShift Online) od listopada 2011 roku. **24 VI 2015** roku, na Red Hat Summit w Bostonie, Red Hat wydał **OpenShift Enterprise 3**, przebudowany na „Docker format Linux containers" i Kubernetesa, działający na RHEL 7. W komunikacie Red Hat przedstawiał się jako „leading contributor" obu projektów. Wersja komercyjna oparta na Kubernetesie ukazała się więc prawie miesiąc przed Kubernetesem 1.0. O OpenShifcie w kontekście całego portfela Red Hata, a także o obrazach UBI i trybie obrazowym RHEL opartym na bootc, piszę na stronie **Red Hat, RHEL i saga CentOS**.

Podman 1.0 i rozbicie Dockera na trzy narzędzia

Odpowiedzią Red Hata na „jeden monolityczny demon jako root” było rozbicie go na osobne narzędzia. **16 I 2019** roku ukazał się **Podman 1.0**, opisany w ogłoszeniu jako „a fully-featured container engine” z „a Docker-compatible command line to ease the transition” — wierszem poleceń zgodnym z Dockerem, żeby ułatwić przesiadkę. Podman działa bez demona. Kontenery bezrootowe, pody i obsługa plików YAML podów Kubernetesa były w nim od wersji 0.2; w 1.0 doszły m.in. `podman play kube`, opcja `--init` i `podman image sign`. Najważniejsze zdanie ogłoszenia: „Most Podman commands can be run as a regular user, without requiring additional privileges.”

Dwa dni później Sean Michael Kerner w eWeek cytował ludzi Red Hata. Scott McCarty: „Podman is set to be the default container engine for the single-node use case in Red Hat Enterprise Linux 8”. Dan Walsh uzasadniał architekturę: „Forcing all of your containers to run through a single daemon forces you to have a least common denominator” — zmuszanie wszystkich kontenerów do przejścia przez jednego demona sprowadza je do najniższego wspólnego mianownika. Mówił też o rozdzieleniu pracy na **CRI-O**, **Buildah** i **Podmana** — każde narzędzie z innym poziomem uprawnień. CRI-O jest środowiskiem uruchomieniowym dla Kubernetesa, a w RHEL 8 Podman, Buildah i trzecie narzędzie rodziny, **Skopeo**, trafiły do jednego modułu `container-tools`.

RHEL 8 bez Dockera

Dokumentacja RHEL 8 mówi to bez ogródek: „Red Hat removed the Docker container engine and the docker command from RHEL 8.” Docker pobrany z repozytoriów projektu „is unsupported in RHEL 8”. Dla przyzwyczajonych jest pakiet `podman-docker`: „every time you run a docker command, it actually runs a podman command” — każde wywołanie `docker` w rzeczywistości uruchamia `podman`, a gniazdo Dockera jest przekierowane na gniazdo Podmana. Polecenia instalacji podaję w rozdziale o praktyce. Środowiska uruchomieniowe niskiego poziomu są w dokumentacji dwa do wyboru: **runc** i **crun**.

Podman w CNCF

Przez kilka lat Podman był postrzegany jako „Docker od Red Hata”. To się zmieniło. **14 XI 2024** roku Red Hat ogłosił zamiar przekazania CNCF całej rodziny: bootc, Buildah, Composefs, Podman, Podman Desktop i Skopeo; podawał przy tym, że Podman Desktop był „downloaded over 1.5 million times”, a Podman miał 23,6 tysiąca gwiazdek na GitHubie — to liczby Red Hata. **21 I 2025** roku CNCF przyjęła „Podman Container Tools” (Podman, Buildah, Skopeo) na poziomie Sandbox, z opisem „manage[s] containers and images without requiring a daemon or root privileges”.

Ekosystem: Helm, operatorzy i dystrybucje

Sam Kubernetes to API i kilka demonów. Wokół niego wyrosły narzędzia, bez których mało kto używa go w praktyce, i dystrybucje, które składają go w produkt.

Helm, menedżer pakietów Kubernetesa, wyszedł w listopadzie 2015 roku na pierwszym KubeCon, przygotowany przez firmę Deis i wzorowany na Homebrew z macOS. W styczniu 2016 roku zespół Deis połączył siły z Google, Skippbox i Bitnami, od czerwca 2018 roku projekt jest w CNCF, a dziś ma tam status „graduated”. **Helm 3** ukazał się **13 XI 2019** roku — tego samego dnia, w którym Docker sprzedał biznes enterprise. Matt Fisher pisał w ogłoszeniu: „The most apparent change is the removal of Tiller.” Wpis nie podaje, dlaczego go usunięto, a obiegowe wyjaśnienie — względy bezpieczeństwa — zostawiam bez potwierdzenia.

Operatorzy to pomysł CoreOS-a. **3 XI 2016** roku Brandon Philips, dyrektor techniczny firmy, przedstawił „Operators” jako oprogramowanie, które zapisuje wiedzę inżyniera z instrukcji operacyjnych, żeby klaster sam wykonywał czynności, które dotąd robił człowiek. Pierwszymi dwoma operatorami, wydanymi jako otwarty kod, były etcd Operator i Prometheus Operator.

k3s Rancher Labs, ogłoszony **26 II 2019** roku, pokazał odwrotny kierunek: „a Kubernetes distribution for low-resource computing”, mieszczący się w „a less than 40MB binary” i działający w „less than 512MB of RAM”. Usunięto z niego kod przestarzały i domyślnie wyłączony, niedomyślne sterowniki pamięci masowej i kontrolery dopuszczania, a zamiast etcd można było użyć SQLite — „as a low memory alternative to etcd”. **1 XII 2020** roku SUSE zakończyło przejęcie Rancher Labs. Warunków nie ujawniono; SiliconANGLE pisał, że SUSE „reportedly paid $600 million to $700 million”, nie wskazując źródła szacunku — źródła podają więc kwotę od 600 do 700 mln USD, ale jest to szacunek prasy[6]. Melissa Di Donato z SUSE mówiła o „a union that brings together the best in Kubernetes management, Linux and edge computing”. Miejsce Ranchera w SUSE opisuje strona **SUSE i openSUSE**.

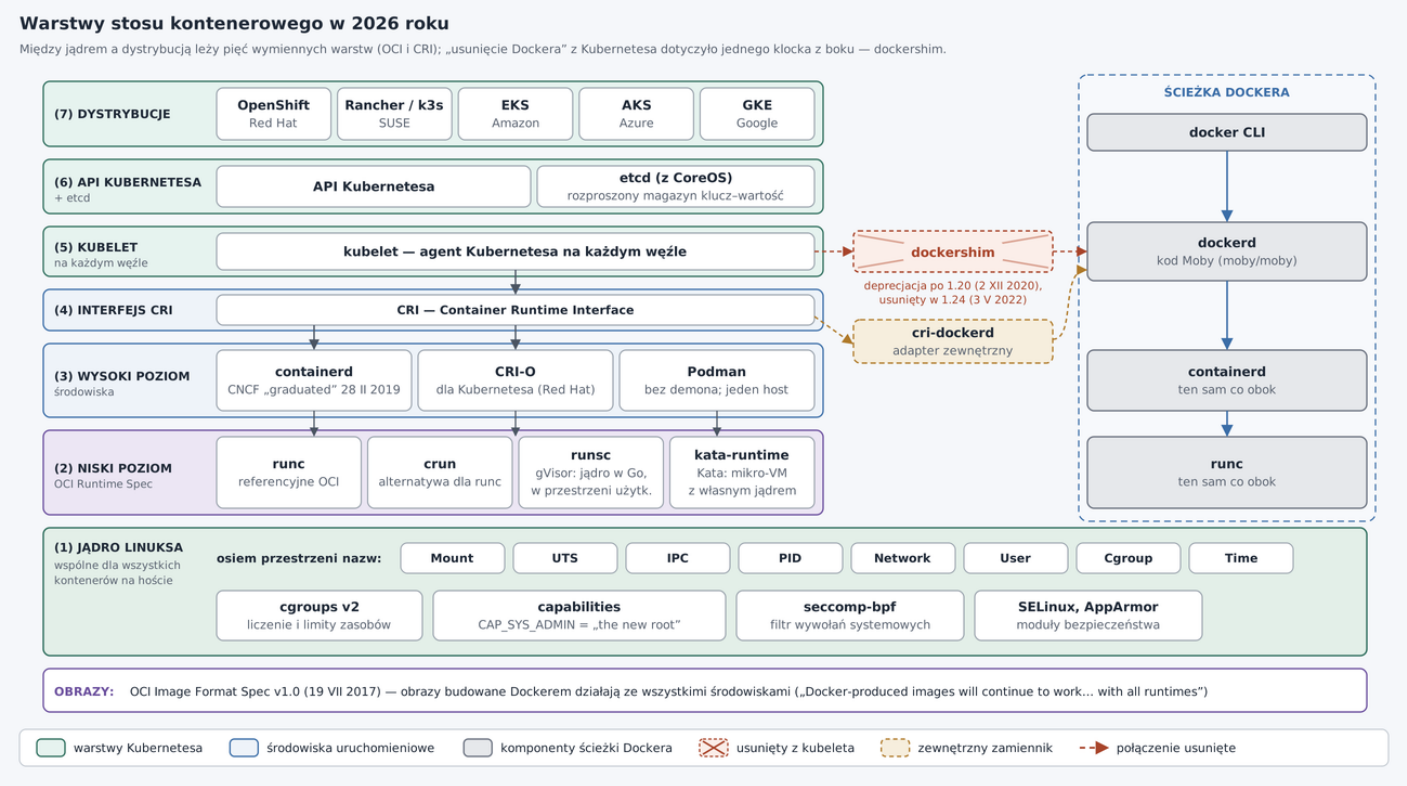
dockershim: „Don't Panic”

Głośna i często źle rozumiana decyzja Kubernetesa dotyczyła jednego adaptera. Kubernetes rozmawia ze środowiskiem uruchomieniowym przez interfejs **CRI** (Container Runtime Interface). Docker go nie obsługiwał, więc kubelet miał w sobie wbudowaną przejściówkę — **dockershim** — która tłumaczyła wywołania CRI na API Dockera. containerd i CRI-O mówiły w CRI bezpośrednio.

2 XII 2020 roku blog Kubernetesa opublikował wpis pod tytułem, który mówił wszystko: „Don't Panic: Kubernetes and Docker”. Treść: „Kubernetes is deprecating Docker as a container runtime after v1.20.” — i zaraz potem: „You do not need to panic. It's not as dramatic as it sounds.”

Wyjaśnienie: „Docker as an underlying runtime is being deprecated in favor of runtimes that use the Container Runtime Interface (CRI) created for Kubernetes.” Najważniejsze zdanie dla każdego, kto budował obrazy: „Docker-produced images will continue to work in your cluster with all runtimes, as they always have.” — obrazy zbudowane Dockerem działają dalej z każdym środowiskiem, jak zawsze.

Usunięcie nastąpiło w **Kubernetesie 1.24 „Stargazer”**, wydanym **3 V 2022** roku z 46 usprawnieniami: „after its deprecation in v1.20, the dockershim component has been removed from the kubelet… you will need to either use one of the other supported runtimes (such as containerd or CRI-O) or use cri-dockerd if you are relying on Docker Engine”. Kto koniecznie chciał zostać przy Docker Engine na węzłach, mógł użyć zewnętrznego adaptera **cri-dockerd**. Zdanie „Kubernetes porzucił Dockera” broni się więc tylko częściowo: porzucił wbudowany adapter do silnika Dockera, nie format obrazów, który i tak od 2017 roku jest standardem OCI.



Rysunek 4. *Między jądrem a dystrybucją Kubernetesa leży pięć warstw, każda wymienna dzięki OCI i CRI — a „usunięcie Dockera” z Kubernetesa dotyczyło jednego klocka z boku, przejściówki, której reszta stosu nie potrzebowała.*

Bezpieczeństwo: ucieczki z kontenerów jako historia

Kontener dzieli z gospodarzem jądro, a często także programy, które nim zarządzają. Kiedy któryś z nich ma błąd, granica przestaje istnieć. Dwa takie przypadki są szczególnie dobrze udokumentowane i oba dotyczą tej samej warstwy: runc. Opisuję je jako historię — kto, kiedy, czego dotyczyły i co pomagało — bez szczegółów technicznych samego ataku.

CVE-2019-5736: Nowy Rok z Dragon Sector

Odkrywcami byli **Adam Iwaniuk** i **Borys Popławski** z polskiej drużyny CTF **Dragon Sector**. W swoim wpisie z 13 II 2019 roku piszą, że inspiracją było zadanie: „The inspiration to the following research was a CTF task called namespaces by _tsuro from the 35C3 CTF” — zadanie o przestrzeniach nazw autorstwa _tsuro z zawodów na kongresie 35C3. W drodze powrotnej z kongresu uznali, że polecenie `docker exec`, realizowane przez runc, działa w podobnym modelu dołączania procesu do przestrzeni nazw kontenera. **1 I 2019** roku mieli potwierdzenie luki i zgłosili ją zespołowi bezpieczeństwa Dockera. 2 stycznia sprawa trafiła do zespołu bezpieczeństwa OpenContainers, 3–5 stycznia trwała dyskusja o poprawce, a potem embargo do 11 lutego.

Ogłoszenie na liście oss-security wysłał **Aleksa Sarai**, opiekun runc z SUSE, pod tytułem „runc container breakout (all versions)”. Nagłówek wiadomości nosi datę 12 II 2019 roku, godzinę 00:05 strefy +1100 — w Europie był to jeszcze 11 lutego; źródła podają więc datę ujawnienia jako 11 albo 12 II 2019[7]. Łatki opublikowano 11 II 2019 roku o 15:00 czasu środkowoeuropejskiego, a kod demonstracyjny zapowiedziano zgodnie z regułami listy na 18 lutego, siedem dni po ujawnieniu.

Skutek, opisany w ogłoszeniu ogólnie: złośliwy kontener mógł podmienić program runc na gospodarzu i w ten sposób uzyskać wykonanie kodu jako root na samym gospodarzu. Warunkiem było uruchomienie czegoś jako root wewnątrz kontenera — przy tworzeniu kontenera z obrazu przygotowanego przez atakującego albo przy dołączaniu do kontenera, do którego atakujący miał prawo zapisu. Luka dotyczyła runc pod Dockerem, CRI-O, containerd i Kubernetesem; podatny był też Mesos, a podobną, bardziej zawiłą lukę Sarai znalazł w LXC. `systemd-nspawn` nie był podatny. Ocena CVSS: 7.2. Autorzy odkrycia piszą, że poważnie dotknięte były AWS i GCP.

Ogłoszenie zawiera też najcenniejszą dla administratora część: co pomagało. Domyślna polityka AppArmor **nie** blokowała ataku. SELinux w Fedorze chronił pakiety `docker` i `podman`, bo ich procesy miały inną etykietę — ale nie chronił pakietu `moby-engine`. Atak był „blocked through correct use of user namespaces” — blokowało go poprawne użycie przestrzeni nazw użytkowników, w której root gospodarza nie jest zmapowany do kontenera. Przyjęta poprawka sprawiała, że runc uruchamia się z własnej kopii w pamięci, więc zmiana pliku na dysku przestaje mu szkodzić.

Morał (moja ocena): o skutku tej luki zdecydowały dwie rzeczy, które administrator kontroluje — przestrzeń użytkowników i poprawnie skonfigurowany SELinux.

Leaky Vessels (2024)

Pięć lat później ta sama warstwa puściła ponownie. **31 I 2024** roku Rory McNamara z firmy Snyk opisał CVE-2024-21626 jako „an order of operations container breakout vulnerability in all versions of runc <=1.1.11” — błąd kolejności operacji, który przy złośliwym obrazie albo złośliwym procesie budowania mógł „result in container escape to the underlying host OS”. Poprawka przyszła w runc 1.1.12. W tym samym pakiecie, nazwanym „Leaky Vessels” (cieknące naczynia), opisano trzy luki w BuildKit: CVE-2024-23651, CVE-2024-23652 i CVE-2024-23653. Wniosek dla administratora (mój): runc i containerd są częścią gospodarza i łata się je jak jądro, a nie jak aplikację w kontenerze.

Piaskownice: gVisor, Kata Containers i Firecracker

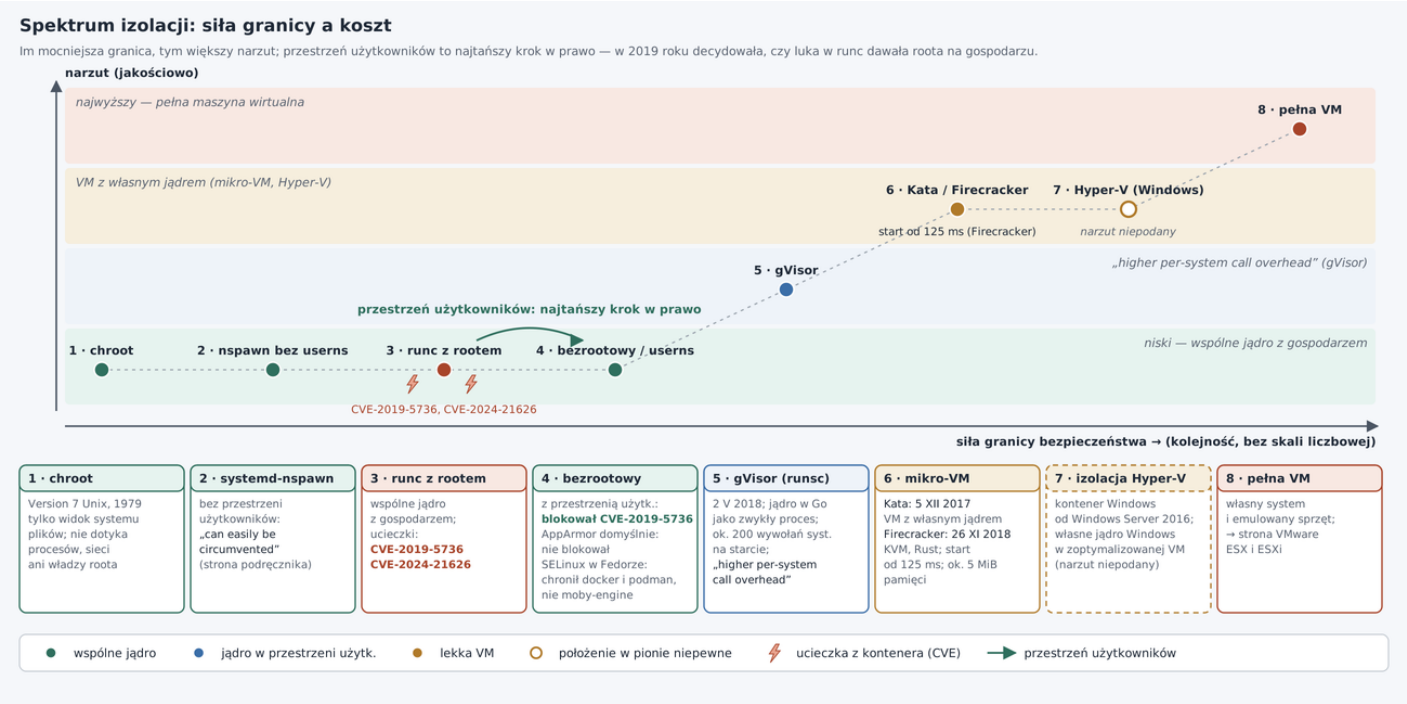
Skoro wspólne jądro jest problemem, rozwiązaniem jest go nie dzielić. Trzy projekty z lat 2017–2018 robią to na trzy sposoby.

gVisor Google udostępnił **2 V 2018** roku. Wpis Nicolasa Lacasse’a zaczyna się od śródtytułu „Traditional Linux containers are not sandboxes” — tradycyjne kontenery linuksowe nie są piaskownicami — i wyjaśnia: „the system surface exposed to containers is broad enough that many security experts don't recommend them for running untrusted or potentially malicious applications”. Rozwiązanie jest pomysłowe: „The core of gVisor is a kernel that runs as a normal, unprivileged process that supports most Linux system calls. This kernel is written in Go” — jądro napisane w Go, działające jako zwykły proces bez uprawnień. Na starcie obsługiwało około 200 wywołań systemowych. Koszt wpis podaje uczciwie: „higher per-system call overhead and application compatibility”. Środowisko `runsc` jest zgodne z OCI i wymienne z runc; przykład z ogłoszenia: `docker run --runtime=runsc hello-world`.

Kata Containers ogłosiła **5 XII 2017** roku OpenStack Foundation, jako projekt o niezależnym zarządzaniu. Hasło: „delivers the speed and performance of Linux containers with the workload isolation of virtual machines” — szybkość kontenerów z izolacją maszyn wirtualnych. Kata powstała z połączenia Intel Clear Containers i runV firmy Hyper, a wspierali ją m.in. Canonical, CoreOS, Google, Huawei, Mirantis, Red Hat, SUSE i Tencent.

Firecracker AWS przedstawił **26 XI 2018** roku; ogłoszenie napisał Jeff Barr. To monitor mikro-maszyn wirtualnych na KVM: „Firecracker is written in Rust”, mikro-VM startuje „in as little as 125 ms” i zajmuje około 5 MiB pamięci. Firecracker napędza AWS Lambda i Fargate. Punktem wyjścia był crosvm — monitor maszyn wirtualnych z Chromium OS, o którym jest mowa na stronie **ChromeOS i ChromeOS Flex**. Serwerowa usługa funkcji AWS ma więc w rodowodzie monitor maszyn wirtualnych z Chromebooków.

Na samym końcu spektrum jest pełna maszyna wirtualna z własnym systemem i emulowanym sprzętem — temat strony **VMware ESX i ESXi**. Między nią a chroot leży cała skala kompromisów, którą najlepiej pokazać na rysunku.



Rysunek 5. *Między chroot a pełną maszyną wirtualną jest kilka stopni izolacji, a im mocniejsza granica, tym większy narzut — przestrzeń użytkowników jest najtańszym krokiem w prawo i to ona w 2019 roku decydowała, czy luka w runc dawała roota na gospodarzu.*

Kontenery w Windows

Kontenery nie są wyłącznie linuksowe, choć Windows dołączył późno. **15 X 2014** roku Microsoft i Docker ogłosili partnerstwo: silnik Dockera „will work with the next release of Windows Server", rozwijany w otwartym projekcie Dockera z Microsoftem jako członkiem społeczności, a Docker Hub miał zostać zintegrowany z portalem Azure. Ben Golub zapowiadał: „Together, we will provide a framework for building multiplatform distributed applications". Następnym wydaniem był Windows Server 2016, w którym pojawiły się kontenery Windows Server i kontenery Hyper-V; szczegóły tej linii, łącznie z losem Nano Servera jako obrazu kontenera, są na stronie **Windows Server 2008–2025**.

Dokumentacja Microsoft Learn opisuje dwa tryby izolacji. W izolacji procesowej „containers share the same kernel with the host as well as each other" — kontenery dzielą jądro z gospodarzem i ze sobą, jak w Linuksie. W izolacji Hyper-V „each container runs inside of a highly optimized virtual machine and effectively gets its own kernel" — każdy kontener działa w zoptymalizowanej maszynie wirtualnej i ma w praktyce własne jądro. Domyślne ustawienia różnią się zależnie od systemu: „Windows containers running on Windows Server default to running with process isolation. Windows containers running on Windows 10 Pro and Enterprise default to running with Hyper-V isolation." Kontener Windows w trybie Hyper-V to w praktyce mała maszyna wirtualna — Microsoft zrobił w 2016 roku to, co świat Linuksa dostał jako Kata Containers pod koniec 2017 roku. To porównanie jest moje, daty pochodzą ze źródeł.

Typowa pułapka kontenerów Windows wynika z tego, że dzielą one jądro tylko z jądrem tej samej wersji. „Windows Server containers are blocked from starting when the build number between the container host and the container image are different” — w izolacji procesowej kontener nie wystartuje, gdy numer kompilacji gospodarza i obrazu się różni (wyjątkiem są obrazy Windows Server 2022 na Windows 11, w podglądzie). W izolacji Hyper-V „each Hyper-V isolated container has its own instance of the Windows kernel”, więc wersje mogą się różnić. Windows Server 2016 wymagał zgodności nawet rewizji (np. 14393.1914 wobec 14393.1944), od wersji 1809 już nie. Komunikat błędu warto znać na pamięć: „The operating system of the container does not match the operating system of the host. (0xc0370101)”. Według tabeli zgodności na gospodarzach Windows Server obrazy 2025 i 2022 działają w obu trybach, a 2019 i 2016 — tylko w Hyper-V.

Kubernetes nauczył się węzłów Windows w wersji 1.14 z 25 III 2019 roku, której ogłoszenie nosiło tytuł „Production-level support for Windows Nodes, Kubectl Updates, Persistent Local Volumes GA”. Jakiej minimalnej wersji Windows Server wymagało to wydanie, nie ustaliłem. Dokumentacja Microsoftu radzi przy mieszanych wersjach albo osobne klastry dla każdej wersji systemu, albo etykiety węzłów (np. `beta.kubernetes.io/osbuild`) i `nodeSelector`.

Skala w 2026 roku: liczby z datą i właścicielem

Liczb o kontenerach jest dużo i prawie wszystkie pochodzą od kogoś, kto je promuje. Podaję je z właścicielem.

Liczba	Kto podaje	Kiedy	Uwaga
ponad 2 mld instancji kontenerów tygodniowo w Google	Google (Eric Brewer)	VI 2014	dane własne firmy
71% firm z Fortune 100 używa kontenerów, ponad połowa — Kubernetesa	Redmonk wg CNCF	III 2018	przytoczone przez fundację Kubernetesa
55% zawodowych programistów używa Dockera codziennie	Stack Overflow Survey 2021 wg Dockera	VIII 2021	przytoczone przez producenta
Kubernetes w produkcji u 82% użytkowników kontenerów (2023: 66%)	CNCF, ankieta roczna za 2025	20 I 2026	ankieta wśród społeczności, próba stronnicza
66% organizacji hostujących modele generatywnej AI używa Kubernetesa do części lub całości inferencji	CNCF, j.w.	20 I 2026	j.w.

Liczba	Kto podaje	Kiedy	Uwaga
ponad 88 000 kontrybutorów z ponad 8000 firm w 44 krajach	projekt Kubernetes	VI 2024	liczby projektu

Ta sama ankieta CNCF z 2026 roku podaje, że 98% organizacji „adopted cloud native techniques”, a jako główną barierę — „Cultural changes with the development team” (47%). Udziałów poszczególnych środowisk uruchomieniowych (containerd, CRI-O, Docker) w 2026 roku nie znalazłem w żadnym wiarygodnym źródle z datą.

Praktyka administratora 2026

Polecenia podaję tylko te, które cytują dokumenty producentów; oceny i wnioski są oznaczone jako moje.

Cykl wydań Kubernetesa: rok i ani dnia dłużej

Projekt Kubernetes wydaje trzy wersje minor rocznie. Zmianę z czterech na trzy przyjęto w dokumencie KEP 23 IV 2021 roku i ogłoszono 20 VII 2021 roku; obowiązuje od wersji 1.22. Strona wydań mówi: „The Kubernetes project maintains release branches for the most recent three minor releases” oraz „Kubernetes 1.19 and newer receive approximately 1 year of patch support”. Rok wsparcia wprowadzono od 1.19, co ogłoszono wpisem „Increasing the Kubernetes Support Window to One Year” z 31 VIII 2020 roku.

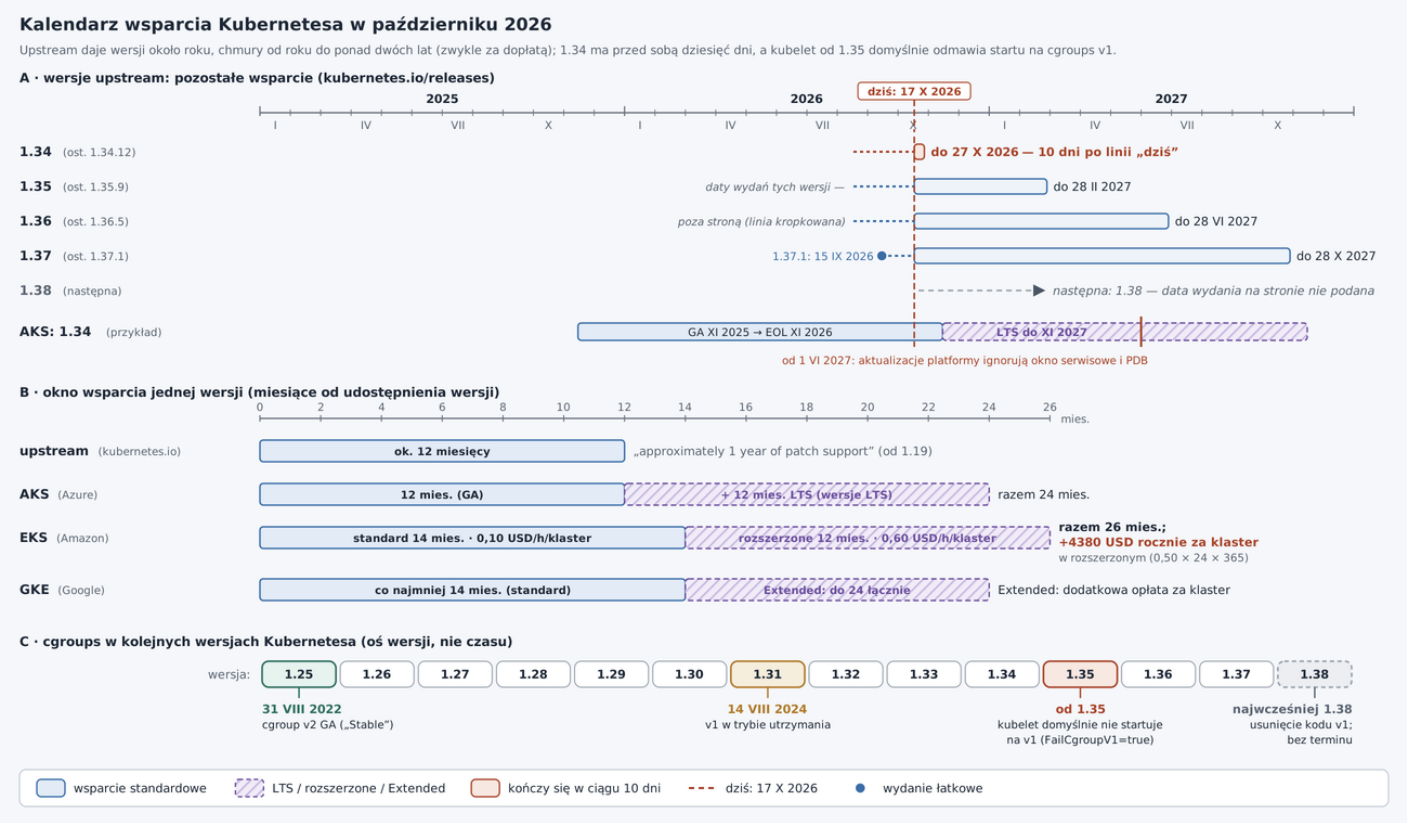
Wersja	Ostatnie wydanie łatkowe	Koniec wsparcia upstream
1.37	1.37.1 (15 IX 2026)	28 X 2027
1.36	1.36.5	28 VI 2027
1.35	1.35.9	28 II 2027
1.34	1.34.12	**27 X 2026**

Stan według kubernetes.io/releases w połowie października 2026 roku; następną wersją będzie 1.38. Wniosek (mój): klaster na wersji upstream trzeba podnosić o jedną wersję minor mniej więcej co cztery miesiące, żeby nie wypaść z łatek po około roku. Kto ma dziś 1.34, za kilka dni zostaje bez poprawek bezpieczeństwa.

Wsparcie wersji u dostawców chmurowych

Dostawcy usług zarządzanych wydłużają okno — zwykle za dopłatą.

- **Amazon EKS** (ogłoszenie z 1 IV 2024): standardowe wsparcie „continues for 14 months” od udostępnienia wersji w EKS, rozszerzone — „continues for 12 months”, razem do 26 miesięcy. Cena: „$0.10 per cluster per hour” w standardzie i „$0.60 per cluster per hour” w rozszerzonym, czyli 0,50 USD na godzinę więcej. Rachunek własny z cennika: 0,50 USD × 24 h × 365 dni = **4380 USD rocznie** dopłaty za jeden klaster pozostawiony na starej wersji.
- **Azure AKS**: „AKS follows a 12-month support policy for generally available (GA) Kubernetes versions”; obsługiwane są wersje N, N-1 i N-2, a dla N-3 jest „platform support” z „60-day grace period”, po którym AKS sam podnosi klaster. Wersje LTS mają „one year of Community Support and one year of Long Term Support (LTS)”. Według kalendarza Microsoftu 1.34 weszła do AKS w XI 2025, kończy się w XI 2026 i ma LTS do XI 2027; 1.35 — III 2026 do III 2027. Uwaga na przyszłość: od 1 VI 2027 aktualizacje inicjowane przez platformę „don't follow your maintenance window and can override a Pod Disruption Budget” — nie będą respektować okna serwisowego i mogą zignorować budżet zakłóceń podów.
- **Google GKE**: „at least 14 months of standard support”, a kanał Extended daje „up to a total of 24 months of support”, tylko z łatkami bezpieczeństwa, i „requires additional fee per cluster”. Kanały: Rapid, Regular, Stable, Extended oraz wycofywany No Channel.



Rysunek 6. *Upstream daje każdej wersji około roku, chmury — od roku do ponad dwóch lat, ale prawie zawsze za dopłatą; w połowie października 2026 roku wersja 1.34 ma przed sobą dziesięć dni, a kubelet od 1.35 domyślnie odmawia startu na gospodarzu z cgroups v1.*

cgroups v1 → v2: koniec okresu przejściowego

Druga wersja grup kontrolnych jest stabilna w jądrze od 2016 roku. W 2026 roku przestała być kwestią wyboru — i to z dwóch stron naraz.

Kubernetes. Obsługa cgroup v2 jest „Stable since Kubernetes v1.25” — ogłosił to wpis „Kubernetes 1.25: cgroup v2 graduates to GA” z 31 VIII 2022 roku. W 1.31 obsługa cgroup v1 „moved into maintenance mode” (wpis z 14 VIII 2024 roku). Dokument KEP-5573 „Remove cgroup v1 support” opisuje kolejny krok: w wersji 1.35 etap beta oznacza deprecjację, a bramka funkcji `FailCgroupV1` zmienia wartość domyślną z false na true — beta „will prevent kubelet from starting on a cgroup v1 node”, czyli kubelet nie uruchomi się na węźle z cgroups v1. Można ją przestawić z powrotem na false, czego KEP odradza. Samo usunięcie kodu nastąpi „no earlier than 1.38”, a „there is not yet a timeline for removal”. Blog Kubernetesa podsumował temat 6 X 2026 roku we wpisie „The Shift to cgroup v2 in Kubernetes: What You Need to Know”, którego treści nie sprawdzałem.

Podman. LWN 25 VI 2026 roku: „Version 6.0.0 of the Podman container-management tool has been released”, „There are many breaking changes”. Wśród zmian: wiele statycznych adresów IP, lepsza izolacja sieci „more compatible with Docker”, zmiany w Quadlecie i przepisana obsługa plików konfiguracyjnych. Strona zmiany „Podman6” w Fedorze (właściciele: Lokesh Mandvekar, Brent Baude, Mohan Boddu) mówi wprost: „Podman 6.0 will no longer support cgroups v1”. Usunięto też slirp4netns (domyślnym mechanizmem sieci dla trybu bezrootowego od Podmana 5.0 jest Pasta), BoltDB (domyślną bazą stanu od 4.8 jest SQLite) i opcję `--network-cmd-path`; Netavark porzuca iptables na rzecz nftables. Wymagane są Netavark, Pasta, cgroups v2 i SQLite. Najgroźniejsza pułapka: kto nadal ma stan w BoltDB, musi najpierw przejść na Podmana 5.8 i uruchomić go ponownie, bo inaczej „container state becomes unusable”. Pliki `containers.conf` mogą wymagać migracji. Docelowe wydanie Fedory strona zmiany podaje niespójnie: raz Fedorę 44, raz 45[8].

Wniosek (mój): w 2026 roku dwa najważniejsze narzędzia — kubelet od 1.35 i Podman 6 — przestają działać na gospodarzu z cgroups v1. To realny powód, żeby przejrzeć stare hosty i zaplanować ich migrację albo wymianę. Polecenia sprawdzającego wersję na hoście nie podaję, bo nie znalazłem go w czytanych źródłach — warto szukać go w dokumentacji Kubernetesa „About cgroup v2”.

Podman w RHEL: polecenia z dokumentacji

Dokumentacja RHEL 8 „Building, running, and managing containers” podaje polecenia instalacji. Cały zestaw narzędzi (Podman, Buildah, Skopeo) instaluje moduł:

```
# yum module install -y container-tools
```

Sam Podman:

```
# yum install podman -y
```

Zgodność z poleceniem `docker` (każde jego wywołanie uruchamia Podmana, a gniazdo Dockera jest przekierowane na gniazdo Podmana):

```
# yum install podman-docker
```

Do kontenerów bezrootowych dokumentacja podaje kilka zasad. Konfigurację trzeba zrobić jako root: „You need to become a root user to set up your RHEL system to allow non-root user accounts to use container tools”. Polecenie `useradd` samo ustawia zakresy podrzędnych identyfikatorów w `/etc/subuid` i `/etc/subgid`; dokumentacja ostrzega: „Never use UIDs or GIDs under 1000…”. Po ręcznej zmianie tych plików trzeba wykonać:

```
podman system migrate
```

I wreszcie zasada z tej samej dokumentacji: „Do not use su or su - commands because these commands do not set the correct environment variables.” — do pracy z kontenerami bezrootowymi nie przełączać się na użytkownika przez `su` ani `su -`. O RHEL jako całości, jego cyklu życia i trybie obrazowym jest strona **Red Hat, RHEL i saga CentOS**.

Docker w trybie bezrootowym

Dokumentacja Dockera opisuje tryb rootless, w którym „run the Docker daemon and containers as a non-root user” — demon i kontenery działają jako zwykły użytkownik, w przestrzeni nazw użytkownika. To co innego niż `userns-remap`, przy którym demon nadal działa jako root. Wymagania: programy `newuidmap` i `newgidmap` (pakiet `uidmap`) oraz „at least 65,536 subordinate UIDs/GIDs for the user” w `/etc/subuid` i `/etc/subgid`. Skrypt `dockerd-rootless-setuptool.sh` jest w `/usr/bin` w pakietach RPM i DEB od Dockera 20.10. Architektura s390x nie jest obsługiwana. Polecenia z dokumentacji:

```
sudo apt-get install -y docker-ce-rootless-extras
sudo systemctl disable --now docker.service docker.socket
dockerd-rootless-setuptool.sh install
sudo loginctl enable-linger testuser
systemctl --user start docker.service
```

Ostatnie polecenie działa analogicznie ze `stop` i `restart`. Tryb bezrootowy blokował CVE-2019-5736, ale nie czyni kontenera maszyną wirtualną.

Kontenery Windows w praktyce

Polecenia z dokumentacji Microsoftu pokazują wybór trybu izolacji przy uruchomieniu kontenera:

```
docker run -it --isolation=hyperv mcr.microsoft.com/windows/servercore:ltsc2019 cmd
docker run -it --isolation=process mcr.microsoft.com/windows/servercore:ltsc2019 cmd
```

Domyślną izolację Hyper-V ustawia się w pliku `daemon.json`:

```
{"exec-opts": ["isolation=hyperv"]}
```

Dwie rzeczy, o których trzeba pamiętać. Tag `latest` obrazów bazowych nie jest publikowany od 16 IV 2019 roku — zawsze trzeba podać tag wersji, np. `ltsc2019`. A w izolacji procesowej obraz musi pasować numerem kompilacji do gospodarza, inaczej kontener nie wystartuje z błędem 0xc0370101.

Pułapki 2025–2026

- **Ingress NGINX bez łatek.** Wpis z 11 XI 2025 roku zapowiedział wycofanie kontrolera `kubernetes/ingress-nginx`: „Best-effort maintenance will continue until March 2026”, potem bez wydań, poprawek i łatek bezpieczeństwa. Uzasadnienie: „To prioritize the safety and security of the ecosystem”. „Existing deployments of Ingress NGINX will continue to function” — istniejące wdrożenia działają dalej, ale w październiku 2026 roku każdy klaster z tym kontrolerem ma na brzegu komponent bez łatek od około siedmiu miesięcy. Zalecenie projektu: Gateway API albo inny kontroler Ingress z listy w dokumentacji.
- **Bitnami za opłatą.** Broadcom ogłosił 16 VII 2025 roku, że od 29 IX 2025 roku „access to most versioned images will move behind a paid subscription under Bitnami Secure Images (BSI), with only the :latest tags remaining free”, a „the bitnami organization will be deleted”. Stare obrazy przeniesiono do organizacji `bitnamilegacy`, gdzie są „unsupported, will no longer receive updates or patches”. Źródła obrazów i charty Helma pozostały dostępne do samodzielnego budowania. Opis pochodzi z bloga Dockera (30 VIII 2025), czyli konkurenta, który poleca własne obrazy. Każdy chart albo manifest wskazujący na wersjonowane obrazy `bitnami/…` wymaga przeglądu.
- **Limity Docker Hub.** Według dokumentacji z 27 III 2025 roku limit pobrań liczy się „per 6 hours”: dla niezalogowanych 100 (na adres IPv4 albo podsieć IPv6 /64), dla konta Personal 200. Wniosek (mój): wiele węzłów klastra za jednym NAT-em dzieli jeden limit — stąd sens lokalnego rejestru-lustra albo uwierzytelnionych pobrań.
- **Stary rejestr obrazów Kubernetesa.** „On the 3rd of April 2023, the old registry k8s.gcr.io will be frozen and no further images… will be pushed to the old registry”; nowy, wspólnotowy rejestr to `registry.k8s.io`. Manifesty z `k8s.gcr.io` wskazują na obrazy zamrożone w 2023 roku.

- **Wersja runc.** runc do 1.1.11 włącznie jest podatny na CVE-2024-21626; poprawka w 1.1.12. Sprawdza się to przy aktualizacji pakietów gospodarza, nie obrazów.
- **Licencja Docker Desktop.** W firmie od 250 osób albo od 10 mln USD rocznego przychodu Docker Desktop wymaga płatnej subskrypcji (od 31 VIII 2021).
- **Kubernetes 1.34** traci wsparcie upstream 27 X 2026 roku; **cgroups v1** — kubelet od 1.35 domyślnie nie startuje, Podman 6 nie obsługuje.

Mity i sprostowania

Człon opowieści	Werdykt	Dlaczego
„Kontenery wymyślił Docker w 2013 roku”	przewraca się	chroot w V7 (1979), jaile (2000), Virtuozzo (2002), strefy (2004–2005), cgroups (2008), LXC; Docker na starcie „complements LXC”. Wniósł obraz, rejestr i przepływ pracy.
„Docker był rewolucją”	broni się, ale nie w jądrze	Rewolucja w dystrybucji oprogramowania: 140 tysięcy pobrań i prawie 200 współautorów w siedem miesięcy; od 0.9 własny libcontainer.
„chroot wymyślił Bill Joy w 1982 roku dla bezpieczeństwa”	przewraca się	chroot jest w podręczniku V7 (1979); Joy dodał go do drzewa BSD 18 III 1982, według McKusicka raczej do budowania systemu; FTP przyszło w 1983.
„W jądrze Linuksa jest obiekt kontener”	przewraca się	Jest osiem przestrzeni nazw, cgroups, capabilities i seccomp; słowo „container” usunięto z nazwy cgroups jako „too generic”.
„Kontener to lekka, bezpieczna maszyna wirtualna”	przewraca się	Wspólne jądro; Google: „not sandboxes”; nspawn: „can easily be circumvented” bez przestrzeni użytkowników; CVE-2019-5736 i CVE-2024-21626. Lekkimi VM są Kata, Firecracker i izolacja Hyper-V.
„Rootless znaczy bezpieczne”	broni się częściowo	Usuwa klasę zagrożeń (blokował CVE-2019-5736), ale kontener dalej dzieli jądro, a tworzenie przestrzeni użytkowników bez uprawnień to od 3.8 nowa powierzchnia ataku (ocena).
„Jaile i strefy to przodkowie kodu Dockera”	przewraca się	Wpływ koncepcyjny, bez linii kodu; motywacją przestrzeni PID i sieci były OpenVZ i Vserver.

Człon opowieści	Werdykt	Dlaczego
„Google wymyślił Kubernetesa od zera”	broni się częściowo	Kod nowy i otwarty od 6 VI 2014, ale idee z Borga i Omegi; Google wcześniej dał jądru cgroups.
„Kubernetes to Borg wypuszczony na zewnątrz”	przewraca się	Borg i Omega są „purely Google-internal”; Kubernetes napisano od nowa, z IP na pod i etykietami zamiast numerów portów.
„Google dał Kubernetesa światu”	broni się	Przekazany CNCF przy 1.0 (21 VII 2015); interes chmury Google to moja ocena.
„Kubernetes od początku skalował się do tysięcy węzłów”	przewraca się	Przy 1.0 ok. 100 węzłów i 30 podów na węzeł; cel 1000.
„Kubernetes porzucił Dockera w 2022 roku”	broni się częściowo	Usunięto dockershim (1.24), nie obsługę obrazów: „Docker-produced images will continue to work”; Docker Engine nadal przez cri-dockerd.
„Docker wygrał wojnę orkiestratorów Swarmem”	przewraca się	Docker wbudował Kubernetesa w produkty w 2017 i sprzedał biznes enterprise w XI 2019.
„Docker jest płatny”	przewraca się w większości	Płatny jest Docker Desktop w firmach od 250 osób lub 10 mln USD przychodu; o silniku na serwerze warunki milczą.
„Podman to Docker bez demona od Red Hata”	broni się, ale niepełne	Bez demona i bezrootowy od 1.0, domyślny w RHEL 8; od 2025 w CNCF, więc już nie tylko Red Hata.
„Windows nie ma własnych kontenerów”	przewraca się	Natywne od Windows Server 2016, w dwóch trybach izolacji; Docker Desktop to osobny produkt.
„Kubernetes ma LTS na lata, jak RHEL”	przewraca się	Upstream: około roku dla trzech ostatnich wersji; dłużej tylko u dostawców, zwykle płatnie (EKS do 26 miesięcy, GKE do 24, AKS z LTS 2 lata).

Epilog

Przez prawie pół wieku kontener zmieniał przeznaczenie trzy razy. W 1979 roku był sztuczką z katalogiem głównym dla ludzi, którzy budowali system. Potem, przez dekadę z okładem, był sposobem na sprzedanie kawałka serwera: klientowi hostingu, działowi w korporacji z Solarisem, aplikacji przenoszonej między instancjami AIX. W 2013 roku Docker zrobił z niego sposób na wysłanie oprogramowania razem z całym jego światem, a Kubernetes rok później — jednostkę, w

której myśli centrum danych. Po drodze nikt niczego nie wynalazł od zera: Google oddał jądru grupy kontrolne, ludzie OpenVZ — dużą część przestrzeni nazw, Docker — format, który oddał fundacji, CoreOS — etcd i operatorów, Red Hat — narzędzia bez demona. Zmieniło się wszystko poza jednym: kontener nadal dzieli jądro z gospodarzem. Dlatego najważniejszą rzeczą, jaką administrator może zrobić w 2026 roku, nie jest wybór między Dockerem a Podmanem, tylko pilnowanie trzech dat — wersji Kubernetesa, wersji grup kontrolnych i wersji runc.

Ciekawostki zebrane

- Słowa „container” Sun używał w ofercie w 2004 roku („N1 Grid Containers”) — dziewięć lat przed Dockerem.
- W jądrze Linuksa „containers” było pierwotną nazwą cgroups; zmieniono ją, bo była „too generic”.
- W 2007 roku w 32-bitowych flagach `CLONE_` zostały tylko dwa wolne bity.
- Pierwszy commit Kubernetesa: 250 plików, 47 501 linii Go, basha i Markdownu.
- Kubernetesa ogłoszono w 16. minucie przemówienia Erica Brewera, dzień po Dockerze 1.0.
- Google porzucił własny lmctfy na rzecz libcontainer Dockera.
- Virtuozzo zaczęło się od rozmowy z Singapurem w XI 1999 roku i biura przy MIPT z pięcioma osobami.
- Firecracker, który napędza Lambdę, wyrósł z crosvm — monitora maszyn wirtualnych z Chromebooków.
- gVisor to jądro z interfejsem Linuksa napisane w Go, działające jako zwykły proces.
- TechCrunch w tekście o Kubernetesie 1.0 pomylił miesiąc jego narodzin, a The Register przy Dockerze 1.0 — nazwisko twórcy („Solomon Sykes”).
- k3s mieścił się w binarium poniżej 40 MB i 512 MB RAM, a etcd zastępował SQLite-em.
- Aleksa Sarai, który rozesłał ogłoszenie o CVE-2019-5736, cztery lata później założył Incusa.
- Helm 3 i sprzedaż Docker Enterprise przypadły na ten sam dzień, 13 XI 2019.

Oś czasu

Data	Wydarzenie
1979	Version 7 Unix: chroot w podręczniku `chdir(2)`
18 III 1982	Bill Joy dodaje chroot do drzewa BSD
XI 1999	Tormasow proponuje „container virtualization” — początek Virtuozzo
III 2000	FreeBSD 4.0 z jailami

Data	Wydarzenie
2001	Linux-VServer Jacques’a Gélinasa
I 2002	Virtuozzo dla Linuksa (SWsoft)
2002	Linux 2.4.19: przestrzeń nazw montowania
15 II 2004	Prasa o N1 Grid Containers w Solarisie 10
I 2005	Solaris 10 ze strefami
2005	OpenVZ: rdzeń Virtuozzo na GPL
IX 2006	Łatka „containers” Rohita Setha (Google)
29 X 2007	LWN: „containers” przemianowane na control groups
15 XI 2007	Redbook IBM o WPAR w AIX 6.1
24 I 2008	Linux 2.6.24: cgroups, przestrzeń PID, zalążek sieciowej
1 IX 2011	HP-UX 11i v3 Update 9: „HP-UX containers”
21 VII 2012	Linux 3.5: seccomp z filtrami BPF
18 II 2013	Linux 3.8: przestrzeń użytkowników kompletna
15 III 2013	Solomon Hykes pokazuje Dockera na PyCon US
29 X 2013	dotCloud zmienia nazwę na Docker, Inc.
20 II 2014	LXC 1.0
10 III 2014	Docker 0.9: libcontainer zamiast LXC
6 VI 2014	Pierwszy commit Kubernetesa
9–10 VI 2014	DockerCon: Docker 1.0, ogłoszenie Kubernetesa
15 X 2014	Partnerstwo Microsoftu i Dockera
1 XII 2014	CoreOS ogłasza Rocket i appc
22 VI 2015	Open Container Project; libcontainer staje się runC
24 VI 2015	OpenShift Enterprise 3 na Kubernetesie
21 VII 2015	Kubernetes 1.0; powstaje CNCF
28 IX 2015	HashiCorp Nomad
13 III 2016	Linux 4.5: cgroups v2 stabilne
VI 2016	Docker 1.12 z trybem Swarm
3 XI 2016	CoreOS przedstawia operatorów
18 IV 2017	Moby Project
19 VII 2017	OCI v1.0
5 XII 2017	Kata Containers

Data	Wydarzenie
30 I 2018	Red Hat kupuje CoreOS za 250 mln USD
6 III 2018	Kubernetes pierwszym absolwentem CNCF
2 V 2018	gVisor
26 XI 2018	Firecracker
1 I 2019	Iwaniuk i Popławski znajdują lukę w runc
16 I 2019	Podman 1.0
11 II 2019	Ujawnienie CVE-2019-5736
26 II 2019	k3s
25 III 2019	Kubernetes 1.14: produkcyjne węzły Windows
16 VIII 2019	CNCF archiwizuje rkt
13 XI 2019	Docker sprzedaje Enterprise Mirantisowi; Helm 3
1 XII 2020	SUSE kończy przejęcie Rancher Labs
2 XII 2020	Zapowiedź deprecjacji dockershim
31 VIII 2021	Płatny Docker Desktop w większych firmach
3 V 2022	Kubernetes 1.24: dockershim usunięty
31 VIII 2022	Kubernetes 1.25: cgroup v2 GA
3 IV 2023	Zamrożenie k8s.gcr.io
4 VII / 7 VIII 2023	LXD przechodzi do Canonicala; Incus
31 I 2024	Leaky Vessels (CVE-2024-21626)
1 IV 2024	Płatne rozszerzone wsparcie w EKS
21 I 2025	Podman Container Tools w CNCF
29 IX 2025	Koniec publicznego katalogu Bitnami
9 X 2025	KCD Warsaw 2025
X 2025	Mesos w Apache Attic
20 I 2026	CNCF: Kubernetes w produkcji u 82% użytkowników kontenerów
III 2026	Koniec utrzymania Ingress NGINX
25 VI 2026	Podman 6.0 bez cgroups v1
27 X 2026	Koniec wsparcia Kubernetesa 1.34

Wątek polski

Dragon Sector i luka w runc

Najgłośniejszą ucieczkę z kontenera w historii Dockera i Kubernetesa znaleźli dwaj Polacy. **Adam Iwaniuk** i **Borys Popławski** z drużyny CTF **Dragon Sector** — według serwisu CTFtime drużyny z Polski — potwierdzili lukę w runc 1 I 2019 roku, zainspirowani zadaniem z zawodów na kongresie 35C3, i opisali ją 13 II 2019 roku na blogu drużyny we wpisie „CVE-2019-5736: Escape from Docker and Kubernetes containers to root on host”. Historię luki opisuję w rozdziale o bezpieczeństwie. Drużyna organizowała też własne zawody — Dragon CTF 2019 odbył się w Warszawie.

Skalowalność Kubernetesa: Tyczyński i Matejczyk

Wojciech Tyczyński, według strony konferencji Infoshare Staff Software Engineer w Google i absolwent Uniwersytetu Warszawskiego, jest liderem grupy roboczej SIG Scalability Kubernetesa i pracuje przy Kubernetesie i GKE od 2015 roku, zajmując się skalowalnością, wydajnością i szeregowaniem. Na Infoshare 2019 mówił o „Kubernetes scalability - the path from 25 to 5000 nodes in a cluster”. To jego wpis o skalowalności Kubernetesa 1.0 cytował w 2015 roku The Next Platform: około 100 węzłów i 30 podów na węzeł przy wersji 1.0, cel — 1000 węzłów i 30 000 podów. Na KubeCon North America 2019 sesję wprowadzającą i pogłębioną SIG Scalability prowadzili Wojciech Tyczyński i **Mateusz Matejczyk**, obaj z Google. Ile z drogi od stu do pięciu tysięcy węzłów było pracą tej dwójki, a ile reszty grupy, źródła nie mówią; nie ustaliłem też, czy pracowali w warszawskim biurze Google.

KCD Warsaw 2025

9 X 2025 roku w Varso Tower w Warszawie odbyło się **KCD Warsaw 2025** (Kubernetes Community Days), wydarzenie „community organized” i „CNCF supported”. Raport przejrzystości opublikowany przez CNCF podaje 237 uczestników, 29 prelegentów i 7 krótkich wystąpień, a wśród organizatorów wymienia Chrisa Suszynskiego, Kasię Brzozowską, Tomasza Cholewę, Patrycję Węgrzynowicz, Macieja Gołaszewskiego, Wojciecha Kocjana, Piotra Stróża i Pawła Piwosza. Raport nazywa je „the first and largest CNCF cloud native event held in Poland”, a 4637,12 USD nadwyżki przeniesiono na kolejną edycję.

Allegro: z powrotem w serwerowni

Polski przykład z produkcji daje blog techniczny Allegro. We wpisie „Back to the data center in 2026 (just a little bit)” z 11 VIII 2026 roku Piotr Bakalarski opisuje usługę swojego zespołu działającą „in the Allegro data center, on a Kubernetes (k8s) cluster”, i incydent z pętlą restartów, który trwał prawie dwie godziny i kosztował około 1000 USD opłat za transfer danych.

Źródła i dalsze lektury

Przypisy

[1] 1979 (Version 7 Unix): strona podręcznika chdir(2) z V7; 18 III 1982 (drzewo BSD, Bill Joy, według logów SCCS i McKusicka): Kamp i Watson, „Jails: Confining the omnipotent root.”, przypisy. Obie daty dotyczą różnych drzew kodu.

[2] 2.6.24 (początek, „basic support for network namespaces”): KernelNewbies, Linux 2.6.24; 2.6.29 („largely complete”): Michael Kerrisk, LWN, „Namespaces in operation, part 1”.

[3] 15 III 2013 (wystąpienie): pyvideo.org, PyCon US 2013, nagranie; około 17 III 2013 (udostępnienie projektu): Changelog.

[4] Czerwiec 2014 (pierwszy commit 6 VI, ogłoszenie 10 VI): „4 Years of K8s”, blog Google Cloud Platform, 10 VI 2014; „launched last February”: TechCrunch, 21 VII 2015 — błąd prasy.

[5] Lipiec 2015 (przekazanie ogłoszone przy 1.0): TechCrunch; początek 2016 (przyjęcie przez komitet techniczny „a little over two years ago” od III 2018): CNCF, 6 III 2018.

[6] Warunki nieujawnione, 600–700 mln USD „reportedly”, bez wskazania źródła szacunku: SiliconANGLE, 1 XII 2020.

[7] 12 II 2019, 00:05 +1100 (nagłówek wiadomości): oss-security w archiwum seclists; 11 II 2019 (archiwum listy, koniec embarga): openwall.com, blog Dragon Sector.

[8] Fedora 44 albo 45 — obie wersje na tej samej stronie: Fedora, Changes/Podman6.

Prehistoria i jądro

- Kamp i Watson, praca o jailach; historia OpenVZ; Red Hat, „The History of Containers”

- Strony podręcznika: namespaces(7), capabilities(7), systemd-nspawn(1)
- LWN: process containers (2007), „Notes from a container”, Open Container Project, Docker 1.12, Podman 6.0
- KernelNewbies: 3.5, 3.8, 4.5; LXC 1.0; LXD do Canonicala; Incus
- UNIX-y: N1 Grid Containers, Redbook o WPAR, HP-UX containers

Google, Docker i Kubernetes

- Google: Borg, Omega, „Borg, Omega, and Kubernetes” (PDF), lmctfy, gVisor
- Docker: zmiana nazwy (InfoQ), 0.9, 1.0, Moby, sprzedaż Enterprise, TechCrunch o Mirantisie, warunki 2021, Johnston, 10 lat Kubernetesa
- Formaty i fundacje: CoreOS i Rocket, OCI v1.0, containerd, archiwizacja rkt, zakup CoreOS
- Kubernetes: 10 lat, „Don't Panic”, 1.24, cykl wydań, wydania, KEP-5573, cgroup v1 w 1.31, cgroup v2 (2026), skalowalność 1.0
- Orkiestratory i ekosystem: Nomad, Mesos 2021, Mesos w Attic, Helm 3, operatorzy, k3s

Red Hat, bezpieczeństwo, Windows i praktyka

- Red Hat: Podman 1.0, eWeek, kontenery w RHEL 8, OpenShift 3, Podman w CNCF, Fedora: Podman6
- Bezpieczeństwo: CVE-2019-5736 na oss-security, wpis Dragon Sector, Leaky Vessels, Kata Containers, Firecracker, Docker rootless
- Windows: Microsoft i Docker, 2014, tryby izolacji, zgodność wersji, Kubernetes 1.14
- Chmury i pułapki: EKS, AKS, GKE, Ingress NGINX, Bitnami, limity Docker Hub, k8s.gcr.io, ankieta CNCF 2026
- Polska: Dragon Sector w CTFtime, Wojciech Tyczyński na Infoshare, SIG Scalability, KubeCon NA 2019, raport KCD Warsaw 2025, grupa KCD Warsaw, Allegro Tech

Ilustracje

Diagramy i animacje są moje. Zdjęcia pochodzą ze zbiorów cyfrowych muzeów, kolekcji i serwisu Flickr, które udostępniają je jako domenę publiczną albo na licencjach Creative Commons; część

jest na licencjach niekomercyjnych (NC) lub bez utworów zależnych (ND) — ta wiki jest prywatna i niekomercyjna, obrazy pokazujemy bez zmian. Obrazy są wyświetlane bezpośrednio z serwerów instytucji i autorów; licencja i autor stoją w każdym podpisie. Materiałów objętych zastrzeżeniem praw nie osadzamy — odsyłamy do nich w źródłach.

Nota metodyczna: czego nie wiemy

- **Dzienne daty wydań jąder** 2.4.19, 2.6.19, 2.6.29, 4.6 i 5.6 — podaję same lata.
- **Początek LXC** (około 2008, IBM, nazwiska Lezcano i Hallyn) — tylko Red Hat i Wikipedia, bez źródła pierwotnego.
- **Autorzy seccomp** w trybie ścisłym i z filtrami BPF — niesprawdzeni.
- **Data otwarcia lmctfy** (X 2013) — tylko Wikipedia; autorstwo wpisu „4 Years of K8s” — niepotwierdzone.
- **OSCON jako miejsce ogłoszenia Kubernetesa 1.0**, pierwszy Docker Swarm i Libswarm z 2014 roku — niesprawdzone.
- **Licencja Docker Engine i Moby** oraz cennik Dockera po XI 2024 — niesprawdzone.
- **Domyślna wersja cgroups** w RHEL 8 i 9 oraz w Ubuntu, a także polecenie sprawdzające wersję ze źródła — nieustalone; treść wpisu z 6 X 2026 — znana tylko z tytułu.
- **Powód usunięcia Tillera** z Helma 3 — wpis go nie podaje.
- **Cena Ranchera** — tylko szacunek prasy; **udziały środowisk uruchomieniowych** w 2026 roku — brak liczby z datą.
- **Zgodność obrazów Windows Server 2022 na gospodarzu 2025** w izolacji procesowej i minimalna wersja Windows Server dla Kubernetesa 1.14 — nieustalone.
- **Kiedy Podman przeszedł** do organizacji GitHub `podman-container-tools` — nieustalone.
- **Polska**: prasy z epoki o Dockerze i wdrożeń bankowych na OpenShifcie nie znalazłem; miejsce pracy Tyczyńskiego i Matejczyka w Google — nieustalone.

Strona opiera się na dokumentach pierwotnych tam, gdzie były dostępne: stronach podręczników, wpisach na blogach projektów, komunikatach prasowych firm, pracach naukowych i ogłoszeniach z list dyskusyjnych. Liczby o skali (Google, CNCF, Docker, Red Hat) pochodzą od podmiotów, które technologię promują, i tak je oznaczam.

Revision #7
Created 2026-10-17 13:31:57 UTC by Tomasz Konieczny
Updated 2026-11-01 17:34:36 UTC by Tomasz Konieczny