Jose Manuel Ortega Candel

Bezpieczeństwo kontenerów w DevOps

Zabezpieczanie i monitorowanie kontenerów Docker

Helion

Tytuł oryginału: DevOps and Containers Security: Security and Monitoring in Docker Containers

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-7996-1

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/bekode.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/bekode
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Rozdział 4.

Wprowadzenie do bezpieczeństwa platformy Docker

W tym rozdziale opisane są dobre praktyki dotyczące bezpieczeństwa kontenerów oraz inne zagadnienia, m.in. funkcjonalności platformy Docker wykorzystywane w kontenerach (np. tryb uprzywilejowany). Przedstawiony jest rejestr Dockera i mechanizm sprawdzania wiarygodności umożliwiający bezpieczne umieszczanie obrazów w serwisie Docker Hub i w prywatnym rejestrze. Platforma Docker oferuje centralny rejestr publicznych obrazów. Jeżeli obraz nie powinien być publicznie dostępny, należy użyć prywatnego rejestru.

Za pomocą platformy Docker można bezpiecznie uruchamiać w odizolowanych środowiskach aplikacje wraz ze wszystkimi zależnościami i bibliotekami. Dzięki temu tworzenie aplikacji, testowanie ich i utrzymywanie jest mniej skomplikowane. Kompaktowe obrazy można uruchamiać w każdym środowisku.

Specjaliści IT powinni dokładnie poznać bezpieczeństwo kontenerów. Administratorzy zajmujący się zarządzaniem i zabezpieczaniem aplikacji powinni szybko zdobyć nowe umiejętności.

Struktura

- Zasady bezpieczeństwa platformy Docker.
- Dobre praktyki bezpieczeństwa.
- Kompetencje kontenerów.
- Wiarygodność kontenerów.
- Rejestr Dockera.

Cele

- Wyjaśnienie zasad bezpieczeństwa platformy Docker i dobrych praktyk.
- Omówienie kompetencji platformy Docker i weryfikacji wiarygodności kontenerów.
- Opis rejestru platformy Docker.
- Opis procesu tworzenia rejestru Dockera na lokalnym komputerze.

Zasady bezpieczeństwa platformy Docker

Z perspektywy bezpieczeństwa kontener Dockera wykorzystuje zasoby hosta, ale zawiera własne środowisko uruchomieniowe i zredukowaną przestrzeń użytkownika w systemie operacyjnym. Oznacza to, że kontener można chronić w bardzo podobny sposób jak hosta, bo w gruncie rzeczy jest to niemal rzeczywisty komputer.

Kontener nie ma dostępu do innych kontenerów ani do systemu operacyjnego hosta (z wyjątkiem woluminów, o ile nada się mu odpowiednie uprawnienia). Może komunikować się z innymi kontenerami poprzez sieć, jeżeli zostanie odpowiednio skonfigurowana.

Należy pamiętać, że kontener stanowi odizolowane środowisko, podobnie jak maszyna wirtualna, ale w mniejszym stopniu obciąża system operacyjny, jest prostszy w konfiguracji i w całkowicie bezpieczny sposób może korzystać z określonych katalogów hosta, jeżeli zajdzie taka potrzeba.

Mimo że procesy kontenera są odizolowane od procesów systemu operacyjnego, wykorzystują jego jądro. Platforma Docker stosuje różne techniki izolowania kontenerów od siebie i zabezpieczania ich przed sobą nawzajem. Najważniejsze są tu centralne funkcjonalności jądra systemu Linux, m.in. grupy sterowania i przestrzenie nazw. Zasoby systemu (pamięć, procesor, sieć) przydziela kontenerom mechanizm oparty na grupach sterujących, który gwarantuje, że każdy kontener wykorzystuje tylko przydzieloną mu ilość zasobów.

Opisane podejście nie oferuje takiego samego stopnia izolacji, jak maszyny wirtualne. Jeżeli haker zaatakuje maszynę wirtualną, jego szanse uzyskania dostępu do jądra bazowego systemu operacyjnego są niemal zerowe. Natomiast kontener jest wyodrębnioną instancją wykorzystującą wspólne jądro systemu, przez co haker ma większe możliwości prowadzenia ataków.

Pomimo opisanych technik izolacji z kontenera można uzyskać dostęp do ważnych elementów systemu operacyjnego, np. grup sterujących lub interfejsów jądra znajdujących się w katalogach */sys* i */proc*. Zatem haker może użyć kontenera do swoich celów. Ponieważ wszystkie kontenery wykorzystują ten sam system operacyjny hosta i tę samą przestrzeń nazw, haker po uzyskaniu dostępu administracyjnego do jednego kontenera może wykonywać operacje na jądrze systemu.

Proces platformy Docker, który zarządza kontenerami, jest uruchamiany z uprawnieniami administratora. Zatem użytkownik mający dostęp do tego procesu automatycznie posiada dostęp do wszystkich katalogów i możliwość korzystania z interfejsu REST API za pomocą protokołu HTTP. Dlatego dokumentacja do platformy Docker zaleca, aby dostęp do procesu miały tylko zaufane osoby.

Programiści korzystający z platformy Docker mają świadomość problemów z bezpieczeństwem i ich zdaniem stanowią one przeszkodę w stosowaniu tej technologii w środowiskach produkcyjnych. Dlatego najnowsze wersje platformy, oferujące fundamentalne techniki izolowania jądra systemu Linux, zostały wzbogacone we funkcjonalności AppArmor, SELinux i Seccomp pełniące funkcje blokad dostępu do zasobów jądra:

- **AppArmor** zarządza prawami dostępu kontenerów do systemu plików.
- **SELinux** jest zaawansowanym systemem zasad kontroli dostępu do zasobów jądra.
- **Seccomp** (Secure Computing Mode) monitoruje wywołania systemowe.

Oprócz powyższych funkcjonalności platforma Docker wykorzystuje tzw. kompetencje (ang. *capabilities*) ograniczające uprawnienia nadawane uruchamianym kontenerom.

Wątpliwości budzą również słabe punkty rejestru, w którym udostępniane są obrazy z aplikacjami. W zasadzie nie ma ograniczeń dotyczących tworzenia i publikowania obrazów w serwisie Docker Hub, co rodzi ryzyko wprowadzenia do systemu złośliwego kodu pobranego z obrazem. Dlatego przed wdrożeniem aplikacji należy się upewnić, że cały kod zawarty w obrazie pochodzi z zaufanego źródła.

Korzystanie z platformy Docker daje programistom, testerom i administratorom wiele korzyści. Programiści mogą skupić się na tworzeniu kodu i nie muszą się zajmować różnicami między środowiskami programistycznym i produkcyjnym. Ponieważ kontenery są bardzo proste w zarządzaniu, a ich główną cechą jest niewielkie obciążenie, które wprowadzają do systemu, bardzo dobrze nadają się do środowisk testowych. Administratorzy mogą łatwo wdrażać aplikacje bez konieczności stosowania maszyn wirtualnych.

Wirtualizacja oparta na kontenerach ma tę zaletę, że umożliwia izolowanie od siebie aplikacji o różnych wymaganiach bez korzystania z systemów operacyjnych gości wprowadzających dodatkowe obciążenie. Technologia kontenerowa wykorzystuje dwa podstawowe elementy jądra systemu Linux: grupy sterowania (ang. *control groups* lub *cgroups*) i przestrzenie nazw.

Za pomocą przestrzeni nazw można izolować od siebie procesy i punkty montażu. Dzięki temu procesy działające wewnątrz różnych kontenerów nie oddziałują na siebie nawzajem. Analogicznie punkty montażu istniejące w różnych kontenerach nie kolidują ze sobą.

Grupa sterowania jest to funkcjonalność jądra systemu ograniczająca wykorzystanie procesora i pamięci przez kontener. W efekcie kontener ma dostęp tylko do tych zasobów, których faktycznie potrzebuje.

Podatność głównego procesu platformy Docker na ataki

Korzystając z wirtualizacji oferowanej przez platformę Docker, łatwo jest zapomnieć o bezpieczeństwie uruchamianych kontenerów. Należy pamiętać, że platforma do normalnego funkcjonowania wymaga uprawnień administratora (*root*).

Główny proces platformy, zarządzający cyklem życia kontenerów, również wymaga uprawnień administratora. Z tego powodu jest podatny na ataki. Więcej informacji na ten temat jest dostępnych w oficjalnej dokumentacji pod adresem *https://docs.docker.com/engine/security*.

Główny proces platformy odpowiada za tworzenie kontenerów i zarządzanie nimi, tj. tworzenie systemów plików, przypisywanie adresów IP, kierowanie pakietów,

zarządzanie procesami i realizację wielu innych zadań wymagających uprawnień administratora. Dlatego bardzo ważne jest, aby proces ten uruchamiać za pomocą konta administratora.

Spośród wielu operacji, które można wykonywać na kontenerach, należy wyróżnić ich uruchamianie, zatrzymywanie i modyfikowanie ich konfiguracji. Za pomocą odpowiednich poleceń można również uzyskiwać poufne informacje, w tym hasła i certyfikaty.

Jednym z ostatecznych celów twórców platformy Docker była możliwość uruchamiania jej głównego procesu bez uprawnień administratora, ale bez ograniczania funkcjonalności i delegowania operacji administracyjnych (np. na plikach systemowych lub sieci) do dedykowanych wątków z szerszymi uprawnieniami.

Jeżeli trzeba udostępnić port platformy Docker na zewnątrz (aby móc używać jej interfejsu API), należy to zrobić tak, aby mogli z niego korzystać tylko zaufani użytkownicy. Prosty sposób polega na zabezpieczeniu komunikacji protokołem HTTPS i certyfikatem. Niezbędna do tego celu konfiguracja jest opisana na stronie *https://docs.docker.com/articles/https*.

Dobre praktyki bezpieczeństwa

Poniższa lista zawiera dobre praktyki bezpiecznego korzystania z platformy Docker:

- Główny proces platformy należy uruchamiać na dedykowanym serwerze, odizolowanym od innych komputerów.
- Najlepszą opcją jest uruchamianie głównego procesu na komputerze z systemem Unix.
- Podczas konfigurowania katalogów hosta jako woluminów należy zachowywać szczególną ostrożność, ponieważ kontener może uzyskać do nich pełny dostęp i wykonywać w nich nieodwracalne operacje.
- Komunikację należy zabezpieczać za pomocą protokołu SSL.
- Wewnątrz kontenerów nie należy uruchamiać procesów z uprawnieniami administracyjnymi.
- Warto rozważyć użycie specjalnych funkcji zabezpieczeń, np. AppArmor lub SELinux.
- Kontenery, w odróżnieniu od maszyn wirtualnych, współdzielą jądro systemu hosta. Dlatego ważne jest, aby instalować najnowsze poprawki bezpieczeństwa.

Administratorzy systemu Linux powinni przestrzegać najlepszych zaleceń dotyczących bezpieczeństwa. Stosując wymienione niżej praktyki, można tworzyć bezpieczne usługi i kontenery:

- Nie uruchamiaj oprogramowania jako administrator.
- Blokuj uprawnienia `SETUID`.
- Usuwaj i dodawaj kompetencje do kontenerów za pomocą parametrów `--cap-drop` i `--cap-add`.
- Za pomocą parametru `--cap-add` dodawaj tylko te kompetencje, które są naprawdę potrzebne.
- W kontenerze uruchamiaj tylko jedną aplikację lub mikrousługę.
- Do współdzielenia poufnych danych nie używaj zmiennych środowiskowych ani nie uruchamiaj kontenerów w trybie uprzywilejowanym.
- Sprawdzaj, jacy użytkownicy mają dostęp do hosta z platformą Docker.
- Aktualizuj platformę Docker do najnowszych wersji, pozbawionych luk w bezpieczeństwie i oferujących nowe funkcjonalności.
- Aktualizuj jądro systemu Linux, w którym działa platforma Docker. Współdzielone przez kontenery jądro systemu jest jednym z najbardziej newralgicznych komponentów systemu. Dlatego bardzo ważne jest, aby instalować najnowsze poprawki, gdy tylko się pojawią.

W kolejnych podrozdziałach są dokładniej opisane niektóre dobre praktyki. Najpierw zajmijmy się domyślnym kontem użytkownika kontenera.

Uruchamianie kontenera jako nie-administratora

Domyślnie kontenery są uruchamiane z uprawnieniami administratora. Stosując opisane niżej polecenia, można się przekonać, że domyślnym użytkownikiem jest *root*.

Poniższe polecenie uruchamia kontener z systemem Alpine i uprawnieniami administratora:

```
$ docker run -v /bin:/host/bin --it --rm alpine sh
```

Rysunek 4.1 przedstawia wynik użycia tego polecenia.

Z perspektywy bezpieczeństwa ważne jest skonfigurowanie przestrzeni nazw, aby ograniczyć dostęp do kontenera. Nie jest to jednak dobra praktyka, gdyż silnik kontenera trzeba uruchamiać za pomocą konta administratora. Konieczne jest więc utworzenie konta dla każdego uruchamianego kontenera.

Domyślnie kontenery uruchamia się za pomocą konta administratora, więc takie uprawnienia obowiązują wewnątrz kontenera. Rozwiązaniem jest wskazanie

```
$  docker run -v /bin:/host/bin -it --rm alpine sh
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
e7c96db7181b: Pull complete
Digest: sha256:769fddc7cc2f0a1c35abb2f91432e8beecf83916c421420e6a6da9f8975464b6
Status: Downloaded newer image for alpine:latest
/ # whoami
root
/ # id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),
26(tape),27(video)
```

Rysunek 4.1. *Uruchomienie kontenera z uprawnieniami administratora*

w pliku *Dockerfile* podczas tworzenia obrazu, jakie konto ma być użyte do uruchomienia kontenera. Służą do tego następujące instrukcje:

```
RUN useradd <opcje>
USER <konto_użytkownika>
```

W pliku *Dockerfile* należy umieścić następujące informacje o koncie użytkownika:

```
FROM python:latest
RUN useradd -s /bin/bash unix_user
USER unix_user
ENTRYPOINT ["bin/bash"]
```

Obraz tworzy się za pomocą następującego polecenia:

```
$ docker image build -t python_image
```

Po uruchomieniu kontenera z opcją interaktywną (`-i`) można sprawdzić, czy konto użytkownika odpowiada temu, które zostało wskazane w pliku *Dockerfile*:

```
$ docker run -ti python_image
```

Rysunek 4.2 przedstawia zawartość pliku */etc/password* po użyciu powyższego polecenia.

```
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
unix_user:x:1000:1000::/home/unix_user:/bin/bash
```

Rysunek 4.2. *Zawartość pliku /etc/password*

Przeglądając plik */etc/password*, można sprawdzić, jakie konto zostało użyte wewnątrz kontenera.

Uruchamianie kontenera w trybie tylko do odczytu

Dobrą praktyką administrowania systemem Linux jest uruchamianie aplikacji z minimalnymi uprawnieniami. Zgodnie z nią kontener należy uruchamiać z parametrem `--read-only`.

Aby ochronić kontener przed atakiem hakera, należy ograniczyć możliwość zapisu w systemie plików. W tym celu należy uruchomić kontener za pomocą polecenia `docker run` i parametru `--read-only`:

```
$ docker run -d --read-only python sh
```

Rysunek 4.3 przedstawia wynik użycia powyższego polecenia.

```
$ docker run -it --read-only python sh
Unable to find image 'python:latest' locally
latest: Pulling from library/python
85b1f47fba49: Pull complete
ba6bd283713a: Pull complete
817c8cd48a09: Pull complete
47cc0ed96dc3: Pull complete
4a36819a59dc: Pull complete
db9a0221399f: Pull complete
7a511a7689b6: Pull complete
1223757f6914: Pull complete
Digest: sha256:59d8481f4b2d21f2ac6623e986b4e91fa704112df3e7d9dddbe7315d4
a153ef5
Status: Downloaded newer image for python:latest
# touch file
touch: cannot touch 'file': Read-only file system
```

Rysunek 4.3. *Uruchomienie kontenera w trybie tylko do odczytu*

Jeżeli kontener zostanie uruchomiony z powyższym parametrem, przy próbie utworzenia pliku pojawi się komunikat *cannot touch the file: read-only filesystem*.

Główny problem z parametrem `--read-only` polega na tym, że większość aplikacji musi tworzyć pliki, np. w katalogu */tmp*, i nie może działać w trybie tylko do odczytu. W takim przypadku pliki i katalogi, do których aplikacja musi mieć dostęp, należy montować jako woluminy.

Jeżeli kontener musi zapisywać dane w systemie plików, dzięki woluminowi można uniknąć błędów. Ponadto dane są zapisywane trwale i są dostępne nawet po zatrzymaniu kontenera. Zalecane jest stosowanie woluminów, jeżeli wykorzystywane są pliki tymczasowe.

Wolumin to osobny katalog w systemie plików kontenera, zarządzany bezpośrednio przez główny proces platformy Docker. Wolumin może być współdzielony z innymi kontenerami.

W poniższym przykładzie uruchamiany jest kontener zawierający bazę MySQL. Jest on skonfigurowany w trybie tylko do odczytu z wyjątkiem katalogów */var/lib/mysql* i */tmp*. Oznacza to, że kontener może zapisywać dane tylko w tych katalogach.

Kontener jest uruchamiany również z innymi parametrami, m.in. `MYSQL_ROOT_PAS` ↪`SWORD` oraz `-v` definiującym wolumin:

```
$ docker run --name mysql --read-only -v /var/lib/mysql -v /tmp -d -e
MYSQL_ROOT_PASSWORD=password mysql
```

Rysunek 4.4 przedstawia efekt próby utworzenia pliku wewnątrz kontenera uruchomionego w trybie tylko do odczytu.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                 NAMES
79b33d74c408        mysql               "docker-entrypoint.s…"   20 seconds ago      Up 19 seconds
3306/tcp, 33060/tcp   mysql
[node1] (local) root@192.168.0.38 ~
$ docker exec mysql touch /opt/filename
touch: cannot touch '/opt/filename': Read-only file system
```

Rysunek 4.4. *Uruchomienie kontenera z bazą MySQL i woluminem*

Po uruchomieniu kontenera przy próbie zapisania pliku w innym katalogu niż */tmp* pojawi się komunikat, że system plików działa w trybie tylko do odczytu.

W przypadku korzystania z woluminów można stosować parametr `ro` wskazujący ścieżkę do zadeklarowanego woluminu:

```
$ docker run -v $(pwd):/pwd:ro debian touch /pwd/x
```

Rysunek 4.5 przedstawia wynik użycia powyższego polecenia.

```
$ docker run -v $(pwd):/pwd:ro debian touch /pwd/x
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
05d1a5232b46: Already exists
Digest: sha256:07fe888a6090482fc6e930c1282d1edf67998a39a09a0b339242fbfa2b602fff
Status: Downloaded newer image for debian:latest
touch: cannot touch '/pwd/x': Read-only file system
```

Rysunek 4.5. *Uruchomienie kontenera z woluminem w trybie tylko do odczytu*

W tym podrozdziale dowiedziałeś się, jak się uruchamia kontener i montuje wolumin w trybie tylko do odczytu.

Blokowanie uprawnień SETUID i SETGID

Bity `SETUID` (identyfikator użytkownika) i `SETGID` (identyfikator grupy) określają specjalne uprawnienia dostępu do katalogów i plików systemu operacyjnego dla użytkowników, którzy nie mają uprawnień administracyjnych.

Jednak powyższe bity mają ten mankament, że mogą zostać wykorzystane przez hakera. Dlatego dobrą praktyką jest wyłączenie uprawnień `SETUID` za pomocą odpowiedniej instrukcji w pliku *Dockerfile*. Poniższa instrukcja usuwa uprawnienia `SETUID` i `SETGID` z tworzonego obrazu:

```
RUN find / -perm +6000 -type f -exec chmod a-s {} ; || true
```

Powyższa instrukcja wyszukuje pliki wykonywalne i odbiera uprawnienia `SETUID` i `SETGID` wszystkim użytkownikom. Należy ją stosować ostrożnie, ponieważ zainstalowane oprogramowanie może potrzebować tych uprawnień do poprawnego działania w kontenerze.

Za pomocą poniższego polecenia można wyłączyć bity `SETUID` i `SETGID` podczas uruchamiania kontenera:

```
$ docker run -d --cap-drop SETGID --cap-drop SETUID <nazwa_kontenera>
```

Weryfikowanie wiarygodności obrazów

Za pomocą zmiennej środowiskowej `DOCKER_CONTENT_TRUST` można sprawdzać, czy obrazy pobierane z rejestru platformy Docker lub z serwisu Docker Hub są wiarygodne i podpisane. Jest to ochrona przed użyciem zainfekowanego obrazu. Aby użyć tej funkcjonalności, należy zdefiniować zmienną środowiskową za pomocą polecenia `export DOCKER_CONTENT_TRUST=1`.

Poniższe polecenie pobiera obraz z serwisu Docker Hub i sprawdza jego sumę kontrolną:

```
$ docker pull obraz@sha256:a25306f3850e1bd44541976aa7b5fd0a29be
```

Powyższe polecenie sprawdza zapisaną w manifeście obrazu sumę kontrolną SHA256. Manifest to plik zawierający metadane składowych części obrazu, m.in. listę wszystkich warstw. Po upewnieniu się, że manifest nie został zmieniony, można bezpiecznie pobrać obraz nawet za pomocą niezaufanego protokołu, jakim jest HTTP.

Ograniczanie wykorzystania zasobów

Domyślnie wszystkie kontenery równomiernie współdzielą zasoby hosta. Oznacza to, że nie ma priorytetów przydzielania zasobów kontenerom. Dlatego należy stosować oprogramowanie monitorujące infrastrukturę lub klaster kontenerów oraz dostarczające szczegółowych informacji o kontenerach, które mogą zagrażać stabilności kompletnej infrastruktury, blokować usługi i zakłócać działanie całego systemu kontenerów.

Jednym z rozwiązań tego problemu jest ograniczanie za pomocą poniższego polecenia zasobów wykorzystywanych przez każdy kontener:

```
$ docker run [opcje] [obraz] [polecenie] [argumenty]
```

Powyższe polecenie ma wiele parametrów, za pomocą których można zarówno ograniczać wykorzystywane zasoby, jak również nadawać kontenerom priorytety dostępu w odpowiednich momentach, aby spełnić wymagania różnych użytkowników.

Poniższe polecenie wyświetla informacje o opcjach dotyczących procesorów, urządzeń i pamięci:

```
$ docker run --help | grep 'cpu\|device\|memory'
```

Rysunek 4.6 przedstawia wynik użycia powyższego polecenia.

```
$ docker run --help | grep 'cpu\|device\|memory'
      --blkio-weight-device list       Block IO weight (relative device weight)
      --cpu-period int                 Limit CPU CFS (Completely Fair Scheduler)
      --cpu-quota int                  Limit CPU CFS (Completely Fair Scheduler)
      --cpu-rt-period int              Limit CPU real-time period in microseconds
      --cpu-rt-runtime int             Limit CPU real-time runtime in microseconds
  -c, --cpu-shares int                 CPU shares (relative weight)
      --cpus decimal                   Number of CPUs
      --cpuset-cpus string             CPUs in which to allow execution (0-3, 0,1)
      --cpuset-mems string             MEMs in which to allow execution (0-3, 0,1)
      --device list                    Add a host device to the container
      --device-cgroup-rule list        Add a rule to the cgroup allowed devices list
      --device-read-bps list           Limit read rate (bytes per second) from a
                                       device (default [])
      --device-read-iops list          Limit read rate (IO per second) from a
                                       device (default [])
      --device-write-bps list          Limit write rate (bytes per second) to a
                                       device (default [])
      --device-write-iops list         Limit write rate (IO per second) to a
                                       device (default [])
      --gpus gpu-request               GPU devices to add to the container
```

Rysunek 4.6. *Opcje ograniczające zasoby wykorzystywane przez kontener*

W tym podrozdziale dokonaliśmy przeglądu różnych opcji, za pomocą których można spełniać wymagania dotyczące wydajności, procesora, pamięci oraz prędkości odczytu i zapisu.

Kompetencje kontenera

Kontenery są domyślnie uruchamiane z określonymi uprawnieniami. Jedną z zalet systemu Linux jest możliwość precyzyjnego przydzielania uprawnień w zależności od sytuacji, dzięki czemu można korzystać z różnych funkcjonalności bez posiadania uprawnień administracyjnych.

Z perspektywy bezpieczeństwa w idealnym przypadku uprawnienia powinny być minimalne i jak najbardziej restrykcyjne. Innymi słowy: nie należy przydzielać szerszych uprawnień niż te, które są niezbędne do korzystania z potrzebnych funkcjonalności. Dzięki temu podatność kontenera na ataki będzie minimalna.

Za pomocą kompetencji zarządza się uprawnieniami dostępu procesu do jądra systemu i ogranicza możliwość wykonywania określonych operacji. Silnik platformy Docker wykorzystuje pewne kompetencje systemu Linux, które mają kluczowe znaczenie dla bezpieczeństwa.

Kompetencje umożliwiają stosowanie bardziej zaawansowanych strategii bezpieczeństwa na różnych poziomach uprawnień. Na stronie *http://man7.org/linux/man-pages/man7/capabilities.7.html* znajduje się opis kompetencji w systemie Linux.

Kompetencje są precyzyjnie określone, dzięki czemu można niezależnie przydzielać i odbierać uprawnienia superużytkownikowi. Poniższa lista zawiera przykładowe kompetencje:

- `CAP_SYSLOG` — modyfikowanie działania dziennika jądra.
- `CAP_NET_ADMIN` — modyfikowanie konfiguracji sieci.
- `CAP_SYS_MODULE` — zarządzanie modułami jądra.
- `CAP_SYS_RAWIO` — modyfikowanie pamięci jądra.
- `CAP_SYS_NICE` — modyfikowanie priorytetów procesów.
- `CAP_SYS_TIME` — modyfikowanie zegara systemowego.
- `CAP_SYS_TTY_CONFIG` — konfigurowanie urządzeń TTY.
- `CAP_AUDIT_CONTROL` — konfigurowanie podsystemu audytu.

Kompetencje są bardzo przydatne, ponieważ dzięki ich szczegółowości można wykonywać operacje systemowe z minimalnymi uprawnieniami bez konieczności korzystania z konta administratora. Kompetencje odgrywają ważną rolę w bezpieczeństwie wielu systemów. Są stosowane w środowiskach wirtualnych, takich jak platforma Docker, do zarządzania kontekstem bezpieczeństwa.

Główna zaleta kompetencji polega na tym, że nie trzeba nadawać procesom uprawnień superużytkownika, tylko uprawnienia niezbędne do wykonywania określonych operacji. Rysunek 4.7 przedstawia listę niektórych kompetencji wraz z opisem.

Nazwy wszystkich kompetencji mają prefiks `CAP_`. Na przykład kompetencja `CAP_`
`↪CHOWN` umożliwia modyfikowanie identyfikatorów UID i GID właściciela pliku.

Kompetencja	Opis
SETPCAP	Modyfikowanie uprawnień procesu.
MKNOD	Tworzenie plików specjalnych za pomocą polecenia mknod.
AUDIT_WRITE	Zapisywanie rekordów w dzienniku audytu jądra.
CHOWN	Wprowadzanie zmian w identyfikatorach UID i GID plików (polecenie chown).
NET_RAW	Stosowanie gniazd typu RAW i SOCKET.
DAC_OVERRIDE	Ominięcie ograniczeń odczytu i zapisu plików oraz sprawdzenie uprawnień.
FOWNER	Ominięcie kontroli uprawnień przy wykonywaniu operacji wymagających zgodnych identyfikatorów UID procesu i pliku.
FSETID	Pozostawienie bitów SETUID i SETGID bez zmian podczas modyfikowania plików.
KILL	Ominięcie kontroli uprawnień podczas wysyłania sygnałów.
SETGID	Wykonywanie dowolnych operacji na identyfikatorach GID procesów i liście dodatkowych identyfikatorów GID.
SETUID	Wykonywanie dowolnych operacji na identyfikatorach UID procesów.
NET_BIND_SERVICE	Wiązanie gniazd ze specjalnymi portami domenowymi (o numerach mniejszych od 1024).
SYS_CHROOT	Modyfikowanie katalogu administratora za pomocą polecenia chroot.
SETFCAP	Określanie uprawnień dostępu do plików.

Rysunek 4.7. *Kompetencje w systemie Linux*

Wyświetlenie wszystkich kompetencji

Pakiet `libcap` w systemie Linux zawiera polecenia i pliki binarne umożliwiające wyświetlanie kompetencji i zarządzanie nimi. Wśród nich można wyróżnić następujące polecenia:

- `getcap` — wyświetlenie kompetencji pliku.
- `setcap` — przydzielenie lub odebranie kompetencji plikowi.
- `getpcaps` — wyświetlenie kompetencji procesu.
- `capsh` — otwarcie interfejsu CLI do testowania i badania kompetencji.

Za pomocą polecenia `capsh` można otworzyć interfejs CLI umożliwiający testowanie i badanie domyślnych kompetencji:

- `CAP_AUDIT_WRITE` — zapisywanie dziennika jądra.
- `CAP_AUDIT_CONTROL` — konfigurowanie podsystemu audytu.
- `CAP_NET_ADMIN` — konfigurowanie sieci.
- `CAP_SETPCAP` — nadawanie uprawnień do sterowania procesami.

Rysunek 4.8 przedstawia przykład uruchomienia kontenera opartego na obrazie zawierającym środowisko Pythona oraz wyświetlenia kompetencji za pomocą polecenia `capsh --print`.

```
$ docker run --rm -it python sh -c 'apk add -U libcap; capsh --print'
Unable to find image 'python:latest' locally
latest: Pulling from library/python
85b1f47fba49: Pull complete
ba6bd283713a: Pull complete
817c8cd48a09: Pull complete
47cc0ed96dc3: Pull complete
4a36819a59dc: Pull complete
db9a0221399f: Pull complete
7a511a7689b6: Pull complete
1223757f6914: Pull complete
Digest: sha256:59d8481f4b2d21f2ac6623e986b4e91fa704112df3e7d9dddbe7315d4
a153ef5
Status: Downloaded newer image for python:latest
sh: 1: apk: not found
Current: = cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap
_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_
chroot,cap_mknod,cap_audit_write,cap_setfcap+eip
Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,
cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_s
```

Rysunek 4.8. *Wyświetlenie kompetencji kontenera*

W tym podrozdziale zostały opisane różne domyślne kompetencje kontenera.

Nadawanie i odbieranie kompetencji

Za pomocą poniższych poleceń i parametrów można nadawać i odbierać kontenerom różne kompetencje:

```
$ docker run --cap-add = {kompetencja}
$ docker run --cap-drop = {kompetencja}
```

Aby nadać wybraną kompetencję, należy użyć następujących poleceń:

```
$ docker run –rm -it –cap-add
$CAP alpine sh
```

Aby odebrać kompetencje administratorowi kontenera, można użyć następujących poleceń:

```
$ docker run –rm -it –cap-drop
$CAP alpine sh
```

Poniższe polecenia odbierają wszystkie kompetencje administracyjne, a następnie przydzielają indywidualne:

```
$ docker run –rm -it –cap-drop ALL –cap-add
$CAP alpine sh
```

W ramach ćwiczenia spróbuj usunąć kompetencję `CHOWN` wewnątrz kontenera, a następnie utworzyć konto użytkownika. Ta operacja zakończy się niepowodzeniem, ponieważ do jej wykonania jest potrzebna kompetencja `CHOWN`.

Poniższe polecenie zmienia właściciela pliku lub katalogu wewnątrz kontenera z systemem Ubuntu:

```
$ docker run –cap-add=ALL –cap-drop=CHOWN -ti ubuntu sh
```

Zmiana właściciela pliku lub katalogu za pomocą powyższego polecenia zakończy się niepowodzeniem i pojawi się komunikat *Operation not permitted* (niedozwolona operacja), jak na rysunku 4.9.

```
$ docker run --cap-add=ALL --cap-drop=CHOWN -ti ubuntu sh
# useradd test
useradd: failure while writing changes to /etc/shadow
# chown nobody /usr/share
chown: changing ownership of '/usr/share': Operation not permitted
```

Rysunek 4.9. *Odebranie kontenerowi kompetencji CHOWN*

W ten sam sposób można uruchomić kontener bazujący na obrazie zawierającym środowisko Pythona i odebrać kompetencję `CAPCHOWN`. W efekcie nie będzie można zmieniać właścicieli plików wewnątrz kontenera. Polecenie zwróci kod błędu, ponieważ konto administratora będzie niedostępne i zmiana właściciela pliku lub katalogu nie będzie możliwa.

Poniższe polecenie odbiera kompetencję `CHOWN` kontenerowi zawierającemu środowisko Pythona:

```
$ docker run -it –cap-drop CHOWN python /bin/bash
```

Rysunek 4.10 przedstawia wynik użycia powyższego polecenia.

Platforma Docker uruchamia kontenery z ograniczonymi kompetencjami. Domyślnie są nadawane następujące kompetencje: `chown`, `dac_override`, `fowner`, `kill`, `setgid`, `setuid`, `setpcap`, `net_bind_service`, `net_raw`, `sys_chroot`, `mknod`, `setfcap` i `audit_write`.

Można odebrać wszystkie domyślne kompetencje i sprawdzić, czy kontener przestanie działać. Poniższe polecenie otwiera powłokę bez domyślnie nadanych kompetencji:

```
$ docker run -ti –cap-drop=CHOWN –cap-drop=DAC_OVERRIDE
--cap-drop=FSETID –cap-drop=FOWNER –cap-drop=KILL –cap-drop=MKNOD
--cap-drop=NET_RAW –cap-drop=SETGID –cap-drop=SETUID
--cap-drop=SETFCAP –cap-drop=SETPCAP –cap-drop=NET_BIND_SERVICE
--cap-drop=SYS_CHROOT –cap-drop=AUDIT_WRITE ubuntu /bin/bash
```

```
$ docker run  -it --cap-drop CHOWN python /bin/bash
Unable to find image 'python:latest' locally
latest: Pulling from library/python
85b1f47fba49: Pull complete
ba6bd283713a: Pull complete
817c8cd48a09: Pull complete
47cc0ed96dc3: Pull complete
4a36819a59dc: Pull complete
db9a0221399f: Pull complete
7a511a7689b6: Pull complete
1223757f6914: Pull complete
Digest: sha256:59d8481f4b2d21f2ac6623e986b4e91fa704112df3e7d9dddbe7315d4
a153ef5
Status: Downloaded newer image for python:latest
root@ee7444d32500:/# chown nobody /usr/share/ca-certificates/
chown: changing ownership of '/usr/share/ca-certificates/': Operation no
t permitted
```

Rysunek 4.10. *Odebranie kompetencji CHOWN kontenerowi zawierającemu środowisko Pythona*

Zalecane jest również odbieranie uruchamianym kontenerom kompetencji `SETUID` i `SETGID`. Za zarządzanie identyfikatorami UID i GID jest odpowiedzialne jądro systemu Linux. Za pomocą wywołań systemowych można sprawdzić, czy żądane uprawnienia mogą zostać nadane.

Aby odebrać kompetencje `setuid` i `setgid` uruchamianemu kontenerowi, należy użyć następującego polecenia:

```
$ docker run -it --cap-drop SETGID --cap-drop SETUID python sh
```

Rysunek 4.11 przedstawia wynik użycia powyższego polecenia.

Przy próbie uzyskania powyższych kompetencji wewnątrz kontenera okaże się, że bity UID i GID są ustawione na 0.

Rysunek 4.12 przedstawia wynik wyświetlenia wszystkich kompetencji.

W tym podrozdziale dowiedziałeś się, że blokując bity UID i GID, można zwiększyć bezpieczeństwo kontenera i zapobiec uzyskaniu szerszych uprawnień.

Blokowanie polecenia ping w kontenerze

Aby zablokować polecenie `ping` w kontenerze zawierającym środowisko Pythona, należy użyć polecenia odbierającego kompetencję `NET_RAW`. Przy próbie użycia polecenia `ping` pojawi się komunikat *ping:Lacking privilege for raw socket* (brak uprawnień do surowego gniazda).

```
$ docker run -it --cap-drop SETGID --cap-drop SETUID python sh
# cat /proc/self/status
Name:   cat
State:  R (running)
Tgid:   7
Ngid:   0
Pid:    7
PPid:   1
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 64
Groups:
NStgid: 7
NSpid:  7
NSpgid: 7
NSsid:  1
VmPeak:     6080 kB
```

Rysunek 4.11. *Odebranie kompetencji SETUID i SETGID kontenerowi zawierającemu środowisko Pythona*

```
# capsh --print
Current: = cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setpcap,cap_net_bind_serv
ice,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap+eip
Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setpcap,cap_net_bind_s
ervice,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
uid=0(root)
gid=0(root)
groups=
```

Rysunek 4.12. *Bity UID i GID po odebraniu kompetencji*

Poniższe polecenie odbiera kompetencję `net_raw` kontenerowi zawierającemu środowisko Pythona:

```
$ docker run --it --cap-drop NET_RAW python sh
```

Rysunek 4.13 przedstawia wynik użycia powyższego polecenia.

W powyższym przykładzie została zablokowana możliwość korzystania z gniazd typu `RAW` i `PACKET`. W ten sposób dzięki kompetencjom można precyzyjnie określać, jakie specjalne operacje mogą wykonywać użytkownicy i procesy.

Dobrą praktyką jest odbieranie kontenerowi wszystkich kompetencji i nadawanie mu tylko tych, których rzeczywiście potrzebuje. Służą do tego celu parametry `--cap-drop` i `--cap-add`.

Poniższy przykład pokazuje, że po odebraniu wszystkich kompetencji nie można użyć polecenia `ping` ani zmienić nazwy hosta. Za pomocą pokazanego polecenia

```
$ docker run -it --cap-drop NET_RAW python sh
Unable to find image 'python:latest' locally
latest: Pulling from library/python
85b1f47fba49: Pull complete
ba6bd283713a: Pull complete
817c8cd48a09: Pull complete
47cc0ed96dc3: Pull complete
4a36819a59dc: Pull complete
db9a0221399f: Pull complete
7a511a7689b6: Pull complete
1223757f6914: Pull complete
Digest: sha256:59d8481f4b2d21f2ac6623e986b4e91fa704112df3e7d9dddbe7315d4a
153ef5
Status: Downloaded newer image for python:latest
# ping 8.8.8.8
ping: Lacking privilege for raw socket.
```

Rysunek 4.13. *Zablokowanie polecenia ping w kontenerze zawierającym środowisko Pythona*

odbierane są wszystkie kompetencje kontenerowi zawierającemu system Alpine. Jak pokazuje rysunek 4.14, przy próbie użycia polecenia `ping` pojawia się komunikat o braku uprawnień.

```
$ docker run -ti --cap-drop=all alpine sh
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
e7c96db7181b: Pull complete
Digest: sha256:769fddc7cc2f0a1c35abb2f91432e8beecf83916c421420e6a6da9f8975464b6
Status: Downloaded newer image for alpine:latest
/ # ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
ping: permission denied (are you root?)
/ # hostname foo
hostname: sethostname: Operation not permitted
```

Rysunek 4.14. *Odebranie wszystkich kompetencji kontenerowi zawierającemu system Alpine*

W ten sposób za pomocą kompetencji można kontrolować wykonywanie przez użytkowników i przez procesy operacji wymagających szerszych uprawnień. Na przykład po odebraniu kontenerowi kompetencji `setuid` i `setgid` haker nie będzie mógł wykorzystać słabych punktów kontenera i uruchomić powłoki systemowej, ponieważ nie będzie miał uprawnień administratora.

Nadawanie kompetencji do zarządzania siecią

W następnym przykładzie użyjemy parametru `--cap-add=NET_ADMIN`, aby nadać kompetencje do konfigurowania i kontrolowania sieci. Dzięki temu będzie można wyłączyć interfejs sieciowy za pomocą polecenia `eth0 down`. W efekcie po użyciu polecenia `ping` pojawi się komunikat o niedostępności sieci.

Poniższe polecenie dodaje kompetencję `net_admin` wewnątrz kontenera zawierającego środowisko Pythona:

```
$ docker run -ti --cap-add=NET_ADMIN python sh -c "ip link set eth0 down"
```

Rysunek 4.15 przedstawia efekt użycia tego polecenia.

```
$ docker run -ti --cap-add=NET_ADMIN python sh
# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group def
ault
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
# ip link set eth0 down
# link
link: missing operand
Try 'link --help' for more information.
# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
7: eth0@if8: <BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
# ping 8.8.8.8
connect: Network is unreachable
```

Rysunek 4.15. *Dodanie kompetencji do zarządzania siecią*

Inna kompetencja, `NET_RAW`, ma istotny wpływ na bezpieczeństwo kontenera, ponieważ dotyczy generowania i wysyłania pakietów. Haker może ją wykorzystać do podszycia się pod innego użytkownika oraz do przeprowadzenia z kontenera ataku typu *man-in-the-middle*. Dlatego zalecane jest odbieranie tej kompetencji wraz z innymi, które nie są niezbędne do uruchomienia aplikacji kontenerowej. Kompetencja `NET_ADMIN` domyślnie nie jest aktywna, dlatego nie należy ufać obrazom, które ją wykorzystują, ponieważ oznacza to, że mają one pełną kontrolę nad siecią.

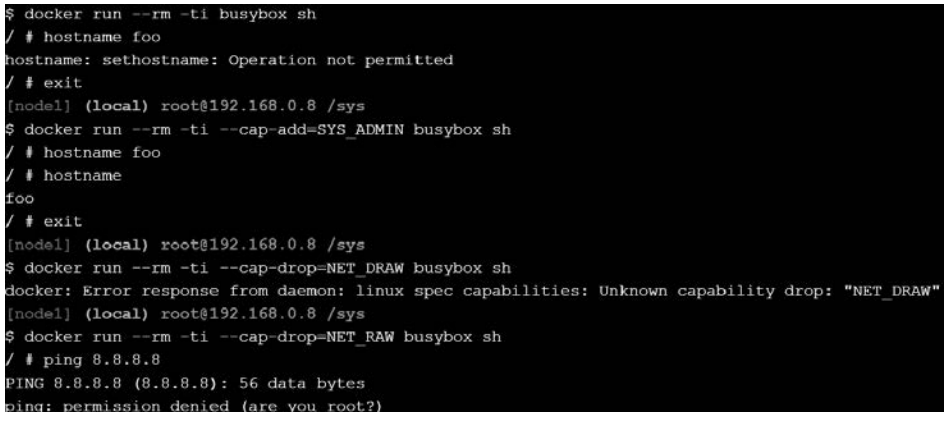
Polecenia pokazane na rysunku 4.16 odbierają kompetencję `NET_RAW` kontenerowi o nazwie *busybox*.

W tym podrozdziale zostało opisane zarządzanie siecią za pomocą kompetencji `NET_RAW` i `NET_ADMIN`.

Uruchamianie uprzywilejowanych kontenerów

W niektórych sytuacjach kontener musi mieć kompetencje do wykonywania operacji na jądrze systemu, które w normalnych warunkach nie są potrzebne. Obejmuje to m.in.: montowanie portu USB, modyfikowanie ustawień sieciowych i definiowanie urządzenia Unix. Poniższe polecenia podejmują próbę zmiany adresu MAC interfejsu eth0 kontenera:

```
$ docker run --rm -ti ubuntu /bin/bash
root@b328e3449da8:/# ip link ls
1: lo: <LOOPBACK,UP,LOWER_UP>mtu 65536 qdiscnoqueue state ...
```



Rysunek 4.16. *Odebranie kompetencji NET_RAW do zarządzania siecią*

```
link/loopback 00:00:00:00:00:00brd 00:00:00:00:00:00
9: eth0: <BROADCAST,UP,LOWER_UP>mtu 1500 qdiscnoqueue state ...
link/ether 02:42:0a:00:00:04brdff:ff:ff:ff:ff:ff
root@b328e3449da8:/# ip link set eth0 address 02:0a:03:0b:04:0c
RTNETLINK answers: Operation not permitted
```

Jak widać, jest to niedozwolona operacja i jądro systemu operacyjnego uniemożliwia jej wykonanie. Jeżeli jednak jest ona niezbędna do poprawnego działania kontenera, należy rozszerzyć jego uprawnienia. Najprościej można to zrobić, uruchamiając kontener z argumentem `--privileged=true`.

Poniższe polecenie uruchamia z pełnymi uprawnieniami kontener z systemem Ubuntu:

```
$ docker run -ti --rm --privileged=true ubuntu /bin/bash
```

Poniższe polecenia pokazują, że teraz można zmienić adres MAC:

```
root@88d9d17dc13c:/# ip link ls
1: lo: <LOOPBACK,UP,LOWER_UP>mtu 65536 qdiscnoqueue state ...
link/loopback 00:00:00:00:00:00brd 00:00:00:00:00:00
9: eth0: <BROADCAST,UP,LOWER_UP>mtu 1500 qdiscnoqueue state ...
link/ether 02:42:0a:00:00:04brdff:ff:ff:ff:ff:ff
root@88d9d17dc13c:/# ip link set eth0 address 02:0a:03:0b:04:0c
root@88d9d17dc13c:/# ip link ls
1: lo: <LOOPBACK,UP,LOWER_UP>mtu 65536 qdiscnoqueue state ...
link/loopback 00:00:00:00:00:00brd 00:00:00:00:00:00
9: eth0: <BROADCAST,UP,LOWER_UP>mtu 1500 qdiscnoqueue state ...
link/ether 02:0a:03:0b:04:0c brdff:ff:ff:ff:ff:ff
```

Kontener uprzywilejowany ma kompetencje do wykonywania operacji zazwyczaj dozwolonych tylko dla administratora. Poniższy przykład ilustruje tworzenie urządzenia podczas montowania obrazu dysku. Za pomocą poniższego polecenia jest uruchamiany kontener zawierający system Fedora z pełnymi uprawnieniami do powłoki `bash`:

```
$ docker run --privileged -i -t fedora /bin/bash
```

Rysunek 4.17 przedstawia wynik użycia tego polecenia.

Rysunek 4.17. *Montowanie obrazu dysku w kontenerze z pełnymi uprawnieniami*

Problem z argumentem `--privileged=true` polega na tym, że dzięki niemu kontener uzyskuje bardzo szerokie uprawnienia. W większości przypadków do wykonywania operacji na jądrze systemu potrzebna jest tylko jedna lub dwie kompetencje. Jak widać, uprzywilejowany kontener ma znacznie szerszy dostęp do sprzętu niż zwykły.

Wiarygodność kontenerów

Docker Content Trust (DCT) to funkcjonalność platformy Docker sprawdzająca, czy kontener jest bezpieczny, czy pochodzi z zaufanego źródła i czy można śledzić jego działanie.

Dzięki tej funkcjonalności programista może podpisać zawartość obrazu i skorzystać z wiarygodnego mechanizmu rozpowszechniania. Gdy użytkownik pobiera obraz z repozytorium, funkcjonalność DCT sprawdza podpis i pobiera certyfikat zawierający klucz publiczny, za pomocą którego może sprawdzić, czy obraz rzeczywiście pochodzi ze źródła widocznego w rejestrze.

W przypadku korzystania z serwisu Docker Hub włączenie funkcjonalności jest bardzo proste i polega na zdefiniowaniu następującej zmiennej środowiskowej:

```
$ export DOCKER_CONTENT_TRUST=1
```

Funkcjonalność DCT umożliwia użytkownikowi korzystanie tylko z tych etykiet obrazu, które zostały podpisane przed pobraniem. Aby ją włączyć, należy zdefiniować zmienną środowiskową `DOCKER_CONTENT_TRUST` lub uruchomić kontener z parametrem `--disable-content-trust=false`.

Wszystkie klucze są przechowywane po stronie klienta. Metadane zawierają tylko znacznik czasu, podpis, etykiety i obraz. Podczas pobierania obrazu z repozytorium sprawdzany jest podpis obrazu.

Rysunek 4.18 przedstawia pobieranie obrazu zawierającego środowisko Pythona przy włączonej funkcjonalności DCT.

Rysunek 4.18. *Pobieranie obrazu zawierającego środowisko Pythona przy włączonej funkcjonalności DCT*

Funkcjonalność DCT chroni kontener przed różnymi atakami, m.in.:

- **Umieszczeniem w obrazie szkodliwego kodu** — haker nie jest w stanie zmodyfikować oficjalnego obrazu i umieścić w nim szkodliwego kodu.
- **Powtarzanymi atakami** — tego typu ataki polegają na umieszczeniu starszej wersji aplikacji, która została złamana. Funkcjonalność DCT sprawdza integralność obrazu, wykorzystując znaczniki czasowe.
- **Zatwierdzeniem klucza** — jeżeli klucz zostanie wykradziony, funkcjonalność DCT tworzy nowy, który jest wykorzystywany do utworzenia nowego obrazu.

Podpisywanie obrazów

Teraz przyjrzymy się integralności obrazu i jego warstw. Każda warstwa zawiera sumę kontrolną SHA256, za pomocą której można sprawdzić, czy obraz nie został zmodyfikowany. Jeżeli jednak obraz zostanie pobrany przez niezabezpieczoną sieć lub opublikowany przez niezaufanego użytkownika, wtedy za pomocą sumy kontrolnej nie można wykryć zmian.

W pliku *Dockerfile* można umieścić dwie instrukcje, które podpisują obraz w chwili jego kompilacji oraz publikują go w repozytorium. Dzięki temu podczas pobierania obrazu można sprawdzić plik z podpisem.

Aby móc podpisywać obrazy, należy zdefiniować zmienne środowiskowe `DOCKER_CONTENT_TRUST` i `DOCKER_CONTENT_TRUST_SERVER`. Pierwsza służy do włączania i wyłączania funkcjonalności DCT. Przy włączonej funkcjonalności jest sprawdzana integralność obrazu z wykorzystaniem usługi uwierzytelniającej wskazanej w drugiej zmiennej.

Druga zmienna środowiskowa zawiera adres URL usługi uwierzytelniającej. Obrazy zazwyczaj pobiera się z serwisu Docker Hub lub repozytorium z wykorzystaniem usługi świadczonej przez firmę Docker.

Aby podpisać obraz, należy w pliku *Dockerfile* zdefiniować powyższe zmienne:

- `export DOCKER_CONTENT_TRUST=1`,
- `export DOCKER_CONTENT_TRUST_SERVER="http://notary.docker.io"`.

Po załadowaniu obrazu klient platformy Docker zwraca ciąg znaków zawierający sumę kontrolną. Suma ta jest następnie używana do weryfikowania obrazu podczas pobierania go.

Rysunek 4.19 przedstawia wynik pobrania obrazu Dockera zawierającego środowisko Pythona i sprawdzenia jego sumy kontrolnej.

```
$ docker pull python@sha256:35ff9f44818f8850f1d318aa69c2e7ba61d85e3b93283078c10e56e7d864c183
sha256:35ff9f44818f8850f1d318aa69c2e7ba61d85e3b93283078c10e56e7d864c183: Pulling from library/python
c5e155d5a1d1: Already exists
221d80d00ae9: Already exists
4250b3117dca: Already exists
3b7ca19181b2: Already exists
425d7b2a5bcc: Already exists
dc3049ff3f44: Pull complete
472a6afc6332: Pull complete
5f79c90f8d7c: Pull complete
1051ee813012: Pull complete
Digest: sha256:35ff9f44818f8850f1d318aa69c2e7ba61d85e3b93283078c10e56e7d864c183
Status: Downloaded newer image for python@sha256:35ff9f44818f8850f1d318aa69c2e7ba61d85e3b93283078c10e56e7d864c183
docker.io/library/python@sha256:35ff9f44818f8850f1d318aa69c2e7ba61d85e3b93283078c10e56e7d864c183
[node1] (local) root@192.168.0.28 ~
$ docker pull python@sha256:35ff9f44818f8850f1d318aa69c2e7ba61d85e3b93283078c10e56e7d864c1831
invalid checksum digest length
```

Rysunek 4.19. *Pobranie obrazu Dockera zawierającego środowisko Pythona i sprawdzenie jego sumy kontrolnej*

Gdy obraz jest pobierany w opisany sposób, platforma Docker sprawdza, czy wyliczona suma kontrolna jest zgodna z podaną. Aktualizacja obrazu powoduje zmianę jego sumy kontrolnej. W ten sposób zapobiega się modyfikacjom obrazu przez osoby trzecie, dzięki czemu można go bezpiecznie pobrać.

Jeżeli funkcjonalność DCT jest włączona, platforma Docker umożliwia pobieranie wyłącznie podpisanych obrazów. Przy próbie uruchomienia obrazu o niezgodnej sumie kontrolnej pojawi się komunikat o niedostępności danych uwierzytelniających dla *docker.io/<nazwa_obrazu>*.

Rysunek 4.20 przedstawia efekt pobrania obrazu z serwisu Docker Hub przy włączonej i wyłączonej funkcjonalności DCT.

W tym podrozdziale zostało opisane weryfikowanie wiarygodności obrazów za pomocą funkcjonalności DCT.

```
$ export DOCKER_CONTENT_TRUST=1
[node1] (local) root@192.168.0.28 ~
$ docker pull jmortegac/linux_tweet_app:1.0
Error: remote trust data does not exist for docker.io/jmortegac/linux_tweet_app: notary.docker.io does not have trust
 data for docker.io/jmortegac/linux_tweet_app
[node1] (local) root@192.168.0.28 ~
$ export DOCKER_CONTENT_TRUST=0
[node1] (local) root@192.168.0.28 ~
$ docker pull jmortegac/linux_tweet_app:1.0
1.0: Pulling from jmortegac/linux_tweet_app
bc95e04b23c0: Pull complete
a21d9ee25fc3: Pull complete
9bda7d5afd39: Pull complete
e64d34b6ad71: Pull complete
a7e018a2b8ff: Pull complete
Digest: sha256:db9f2e75b91780c804c283081123fbe1b2b4080fe124eeb040f8ad1bfd70fe38
Status: Downloaded newer image for jmortegac/linux_tweet_app:1.0
docker.io/jmortegac/linux_tweet_app:1.0
```

Rysunek 4.20. *Pobieranie obrazu przy włączonej i wyłączonej funkcjonalności DCT*

Bezpieczne pobieranie obrazów z wykorzystaniem pliku Dockerfile

W większości przypadków, aby bezpiecznie pobrać obraz z wykorzystaniem pliku *Dockerfile*, należy dodać podpis weryfikujący. Rysunek 4.21 przedstawia plik *Dockerfile* obrazu zawierającego środowisko Pythona. W pliku jest wykorzystany klucz GPG weryfikujący podpis pobieranego obrazu.

```
ENV GPG_KEY E3FF2839C048B25C084DEBE9B26995E310250568
ENV PYTHON_VERSION 3.8.0a4

RUN set -ex \
        \
        && wget -O python.tar.xz "https://www.python.org/ftp/python/${PYTHON_VERSION%%[a-z]*}/Python-$PYTHON_VERSION.tar.xz" \
        && wget -O python.tar.xz.asc "https://www.python.org/ftp/python/${PYTHON_VERSION%%[a-z]*}/Python-$PYTHON_VERSION.tar.xz.asc" \
        && export GNUPGHOME="$(mktemp -d)" \
        && gpg --batch --keyserver ha.pool.sks-keyservers.net --recv-keys "$GPG_KEY" \
        && gpg --batch --verify python.tar.xz.asc python.tar.xz \
        && { command -v gpgconf > /dev/null && gpgconf --kill all || :; } \
        && rm -rf "$GNUPGHOME" python.tar.xz.asc \
        && mkdir -p /usr/src/python \
        && tar -xJC /usr/src/python --strip-components=1 -f python.tar.xz \
        && rm python.tar.xz \
        \
```

Rysunek 4.21. *Bezpieczne pobieranie obrazu z wykorzystaniem pliku Dockerfile*

W tym podrozdziale zostały opisane dobre praktyki pobierania obrazów z wykorzystaniem pliku *Dockerfile*.

Narzędzie notary do zarządzania obrazami

Narzędzie *notary* służy do bezpiecznego publikowania obrazów i zarządzania nimi. Podpisanie obrazu za pomocą tego narzędzia daje użytkownikowi pewność, że pobierany przez niego obraz jest wiarygodny.

Przeznaczeniem narzędzia *notary* jest zapewnienie wiarygodności obrazów pobieranych z publicznych i prywatnych repozytoriów, przenoszenie wiarygodności między użytkownikami i bezpieczne umieszczanie obrazów w repozytoriach z wykorzystaniem niezabezpieczonych kanałów.

Dzięki narzędziu *notary* pobieranie obrazów jest bezpieczniejszym i stabilniejszym procesem. Ułatwia ono publikowanie i weryfikowanie obrazów.

Narzędzie *notary* składa się z serwera i klienta. Na lokalnym komputerze instaluje się klienta, który zarządza kluczami i komunikacją z serwerem. Dokładne informacje na temat kompilowania i konfigurowania serwera uwierzytelniającego są dostępne na stronie *https://github.com/docker/notary*.

Oficjalny serwer narzędzia *notary* znajduje się pod adresem *https://notary.docker.io*. Skompilowane pliki binarne dla 64-bitowych systemów Linux i macOS są dostępne w serwisie GitHub pod adresem *https://github.com/theupdateframework/notary/releases*.

Rejestr Dockera

Platforma Docker oferuje mechanizm rozpowszechniania oprogramowania zwany *rejestrem*. Jest on bardzo istotny, ponieważ umożliwia pakowanie, wysyłanie, przechowywanie, odkrywanie i wykorzystywanie obrazów. Najbardziej znanym rejestrem jest serwis Docker Hub.

Czym jest rejestr?

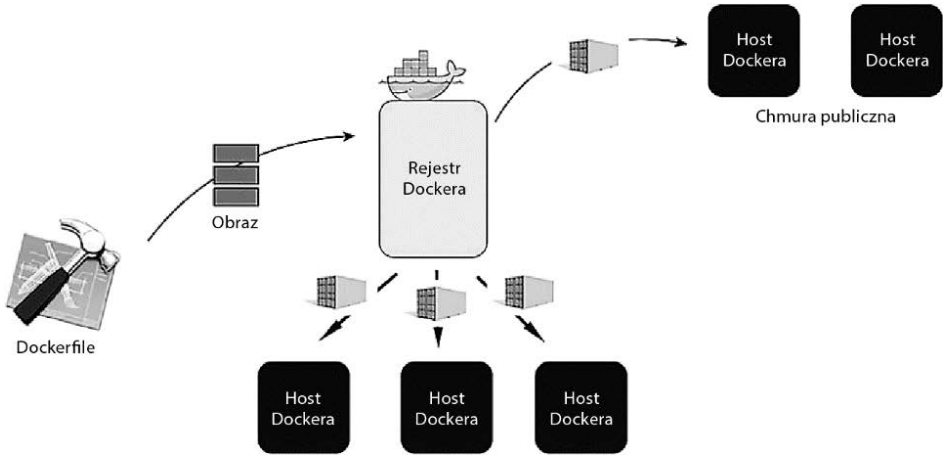
Rejestr odgrywa kluczową rolę podczas tworzenia środowiska kontenerowego i obrazów. Z własnego rejestru można pobierać tworzone samodzielnie obrazy, które nie powinny być publicznie dostępne. Dzięki rejestrowi umieszczonemu we własnej infrastrukturze oszczędza się pasmo, uzyskuje lepszy dostęp do obrazów i przyspiesza ich pobieranie. Dzisiaj może się to wydawać nieistotne, ale w klastrach, w których są udostępniane aktualizacje, jest to naprawdę ważne.

Rysunek 4.22 przedstawia proces tworzenia obrazu i umieszczania go w rejestrze.

Programiści pobierają obrazy z rejestru i tworzą kontenery, które wdrażają w publicznej chmurze lub na prywatnych firmowych serwerach.

Rejestr Dockera w serwisie Docker Hub

Docker Hub jest głównym rejestrem obrazów. Jest udostępniany jako usługa PaaS z różnymi opcjami abonamentowymi.



Rysunek 4.22. *Rejestr Dockera z punktu widzenia programisty*

Rejestr Dockera funkcjonuje podobnie jak repozytorium Git. Każdy obraz składa się z warstw. Platforma Docker kompilując obraz, rejestruje tylko różnice pomiędzy jego nową a poprzednią wersją. Dzięki temu proces tworzenia i rozpowszechniania obrazów jest o wiele efektywniejszy.

Aby móc przechowywać i rozpowszechniać własne obrazy, można zainstalować i uruchomić rejestr Dockera na własnym serwerze. Można to zrobić na kilka sposobów. W systemie Linux zawierającym pakiet rejestracyjny Dockera (np. Fedora lub Red Hat Enterprise) należy zainstalować pakiet i uruchomić usługę. W przypadku innych dystrybucji systemu należy uruchomić oficjalny kontener z rejestrem.

Rejestr Dockera jest otwartym oprogramowaniem. Można je zainstalować na dowolnym serwerze i wykorzystywać do przechowywania prywatnych obrazów. Uzyskuje się wtedy alternatywny rejestr dla serwisu Docker Hub, umożliwiający przechowywanie obrazów na własnym serwerze. W ten sposób można w lokalnym, kontrolowanym środowisku przechowywać prywatne obrazy przeznaczone na wewnętrzny użytek. Oficjalna wersja tej usługi jest dostępna w serwisie Docker Hub pod adresem *http://hub.docker.com/_/registry*.

Tworzenie lokalnego rejestru

Użyty w poniższym przykładzie obraz zawiera implementację rejestru Dockera HTTP API V2, który można stosować z platformą Docker w wersji 1.6 lub nowszej.

Aby zainstalować na własnym serwerze prywatny rejestr Dockera, należy wykonać następujące operacje:

1. Za pomocą poniższego polecenia uruchomić w trybie odłączonym kontener zawierający rejestr, udostępniający port nr 5000 i wykorzystujący adres IP hosta:

   ```
   $ docker run -d -p 5000:5000 --restart = always --name registry registry:2
   ```

   Rysunek 4.23 przedstawia efekt użycia tego polecenia.

Rysunek 4.23. *Pobranie kontenera w celu utworzenia lokalnego rejestru*

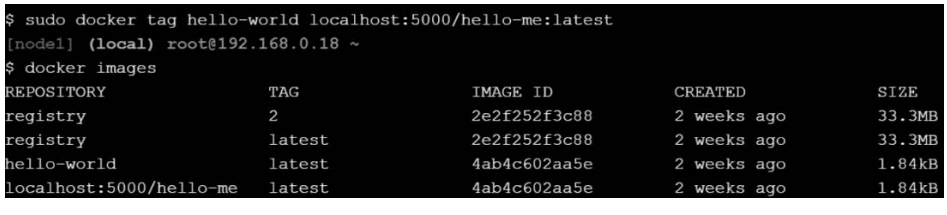
Powyższe polecenie uruchamia dostępny w serwisie Docker Hub obraz zawierający rejestr, udostępnia port TCP nr 5000, za pomocą którego klienci będą mogli korzystać z rejestru, i uruchamia kontener na pierwszym planie systemu. Aby przetestować kontener, załaduj i pobierz obraz z prywatnego repozytorium. Na przykład możesz pobrać obraz `hello-world` dostępny w serwisie Docker Hub:

```
$ sudo docker run --name myhello hello-world
```

2. Kolejną operacją jest przypisanie etykiety obrazowi `hello-world`. W tym celu należy użyć polecenia `docker tag`:

   ```
   $ docker tag hello-world localhost:5000/hello-me:latest
   ```

   Rysunek 4.24 przedstawia efekt użycia tego polecenia.



Rysunek 4.24. *Przypisanie etykiety obrazowi hello-world*

3. Następnie należy umieścić obraz w rejestrze za pomocą następującego polecenia:

   ```
   $ docker push localhost:5000/hello-me:latest
   ```

Rysunek 4.25 przedstawia efekt użycia tego polecenia.

```
$ sudo docker push localhost:5000/hello-me:latest
The push refers to repository [localhost:5000/hello-me]
428c97da766c: Pushed
latest: digest: sha256:1a6fd470b9ce10849be79e99529a88371dff60c60aab424c077007f6979b4812 size: 524
[node1] (local) root@192.168.0.18 ~
$ docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
registry                  latest              2e2f252f3c88        2 weeks ago         33.3MB
hello-world               latest              4ab4c602aa5e        2 weeks ago         1.84kB
localhost:5000/hello-me   latest              4ab4c602aa5e        2 weeks ago         1.84kB
```

Rysunek 4.25. *Umieszczenie obrazu hello-world w lokalnym rejestrze*

4. W kolejnym kroku należy sprawdzić, czy można pobrać obraz z rejestru. W tym celu należy najpierw usunąć bieżący obraz za pomocą polecenia `docker rm`, a następnie pobrać go z lokalnego rejestru:

   ```
   $ sudo docker rm myhello
   $ sudo docker rmi hello-world localhost:5000/hello-me:latest
   $ sudo docker pull localhost:5000/hello-me:latest
   ```

   Rysunek 4.26 przedstawia efekt użycia powyższych poleceń.

```
$ sudo docker rm myhello
myhello
[node1] (local) root@192.168.0.18 ~
$ sudo docker rmi hello-world localhost:5000/hello-me:latest
Untagged: hello-world:latest
Untagged: hello-world@sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Untagged: localhost:5000/hello-me:latest
Untagged: localhost:5000/hello-me@sha256:1a6fd470b9ce10849be79e99529a88371dff60c60aab424c077007f6979b4812
Deleted: sha256:4ab4c602aa5eed5528a6620ff18a1dc4faef0e1ab3a5eddeddb410714478c67f
Deleted: sha256:428c97da766c4c13b19088a471de6b622b038f3ae8efa10ec5a37d6d31a2df0b
[node1] (local) root@192.168.0.18 ~
$ sudo docker pull localhost:5000/hello-me:latest
latest: Pulling from hello-me
d1725b59e92d: Pull complete
Digest: sha256:1a6fd470b9ce10849be79e99529a88371dff60c60aab424c077007f6979b4812
Status: Downloaded newer image for localhost:5000/hello-me:latest
```

Rysunek 4.26. *Usunięcie obrazu hello-world i pobranie jego kopii z lokalnego rejestru*

5. Na koniec można zweryfikować pobrany obraz i uruchomić go:

   ```
   $ docker images
   $ docker run -it localhost:5000/hello-me
   ```

   Rysunek 4.27 przedstawia efekt użycia powyższych poleceń.

Krótko mówiąc: dzięki prywatnemu rejestrowi programiści mogą wysyłać i pobierać obrazy bez korzystania z publicznego rejestru.

```
$ docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
registry                  latest              2e2f252f3c88        2 weeks ago         33.3MB
localhost:5000/hello-me   latest              4ab4c602aa5e        2 weeks ago         1.84kB
ubuntu                    latest              cd6d8154f1e1        3 weeks ago         84.1MB
[node1] (local) root@192.168.0.18 ~
$ docker run -it local
localhost:5000/hello-me         localhost:5000/hello-me:latest
[node1] (local) root@192.168.0.18 ~
$ docker run -it localhost:5000/hello-me

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Rysunek 4.27. *Sprawdzenie i uruchomienie obrazu pobranego z lokalnego rejestru*

Podsumowanie

Bezpieczeństwo kontenerów to nie lada wyzwanie. Jest kilka aspektów, o których należy zawsze pamiętać. Przede wszystkim aby móc uruchamiać kontenery i aplikacje, należy wcześniej uruchamiać główny proces platformy Docker, co wymaga posiadania uprawnień administracyjnych. Inną kwestią jest elastyczność platformy, dzięki której można łatwo uruchamiać wiele instancji kontenerów, z których każdy może mieć zainstalowane inne poprawki bezpieczeństwa.

Platforma Docker, jak każde inne oprogramowanie, jest narażona na zagrożenia. Aby je zminimalizować, należy stosować dobre praktyki bezpieczeństwa i często weryfikować infrastrukturę pod kątem podatności na ataki. W następnym rozdziale będą opisane narzędzia do testowania bezpieczeństwa hosta platformy Docker.

Pytania

1. Jaki komponent platformy jest odpowiedzialny za tworzenie kontenerów, tj. tworzenie systemu plików, przypisywanie adresów IP, kierowanie pakietów, zarządzanie procesami i wykonywanie innych zadań wymagających uprawnień administracyjnych?
2. Jak się nazywa katalog wydzielony z systemu plików, współdzielony przez kontenery i zarządzany bezpośrednio przez główny proces platformy Docker?
3. Jakie bity określają uprawnienia użytkowników innych niż administratorzy do dostępu do plików i katalogów systemu operacyjnego?
4. Za pomocą jakiej zmiennej środowiskowej można sprawdzać, czy obrazy pobierane z rejestru Dockera lub serwisu Docker Hub są wiarygodne i podpisane?
5. Za pomocą jakiego narzędzia można zarządzać uprawnieniami procesów do dostępu do funkcji jądra oraz oddzielać uprawnienia administracyjne?

DevOps jest innowacyjną metodyką prowadzenia projektów, w której wyjątkowe znaczenie ma integracja zespołów programistów i administratorów systemów. Taki sposób rozwijania aplikacji wydaje się szczególnie atrakcyjny w odniesieniu do aplikacji kontenerowych. Technologia kontenerów i orkiestracji jest uważana za bardzo nowoczesną, jednak nawet w przypadku kontenerów Docker i klastrów Kubernetes kwestii bezpieczeństwa nie wolno lekceważyć. Podobnie jak w innych aplikacjach, zabezpieczanie zaczyna się podczas projektowania. O czym więc powinny pamiętać zespoły pracujące zgodnie z DevOps, aby zapewnić bezpieczeństwo swoich kontenerów?

W tej książce pokazano związek między metodyką DevOps a praktyką dotyczącą kontenerów Docker i klastrów Kubernetes z perspektywy bezpieczeństwa, monitoringu i zarządzania. Przedstawiono dobre praktyki tworzenia obrazów kontenerów Docker, a także zasady bezpieczeństwa hostów, na których są uruchamiane kontenery, i wszystkich komponentów. Poruszono takie zagadnienia jak statyczna analiza zagrożeń obrazów Docker, podpisywanie obrazów za pomocą Docker Content Trust oraz umieszczanie ich w rejestrze Docker. Opisano też techniki zabezpieczania platformy Kubernetes. Ponadto znalazł się tutaj opis narzędzi do zarządzania kontenerami i aplikacjami, jak również monitorowania aplikacji kontenerowych i tworzenia sieci w platformie Docker.

Najciekawsze zagadnienia:

- gruntowne wprowadzenie do metodyki DevOps
- czym są platformy kontenerowe: Docker, Kubernetes, Swarm, OpenShift
- zagrożenia kontenerów i obrazów
- narzędzia do audytu bezpieczeństwa i zabezpieczania aplikacji kontenerowych
- zarządzanie kontenerami i ich monitorowanie za pomocą narzędzi: cAdvisor, Sysdig, Portainer i Rancher

Jose Manuel Ortega Candel jest inżynierem oprogramowania i analitykiem bezpieczeństwa. Specjalizuje się w nowych technologiach i w bezpieczeństwie otwartego oprogramowania — koncentruje się na zabezpieczaniu kodu Pythona i stosowaniu metodyki DevOps.
Jest autorem publikacji w czasopismach branżowych. Słynie z entuzjastycznego nastawienia do nowych technologii.

Niezawodność DevOps to także bezpieczne wdrażanie kontenerów Docker!

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-7996-1
9 788328 379961

Cena: 79,00 zł