

Wydanie IV

Serwer Ubuntu

Kompletny przewodnik
po Ubuntu Server 22.04

Jay LaCroix



Helion 

Packt>

Tytuł oryginału: Mastering Ubuntu Server: Explore the versatile, powerful Linux Server distribution Ubuntu 22.04 with this comprehensive guide, 4th Edition

Tłumaczenie: Magdalena A. Tkacz

ISBN: 978-83-8322-592-0

Copyright © Packt Publishing 2022. First published in the English language under the title 'Mastering Ubuntu Server - Fourth Edition – (9781803234243)'

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/serub4>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści |

O autorze	13
O recenzentach	13
Przedmowa	15
ROZDZIAŁ 1	
Wdrażanie serwera Ubuntu	21
Wymagania techniczne	22
Określanie roli serwera	22
Wybór urządzenia dla naszego serwera	24
Serwer fizyczny	24
Komputer stacjonarny	25
Laptop	25
Maszyna wirtualna	26
Prywatny serwer wirtualny	26
Raspberry Pi	27
Skąd wziąć nośnik instalacyjny?	27
Tworzenie rozruchowego dysku USB	30
Planowanie układu partycji	33
Instalacja serwera Ubuntu	35
Instalacja Ubuntu na Raspberry Pi	45
Podsumowanie	49
Dodatkowe samouczki	49
ROZDZIAŁ 2	
Zarządzanie użytkownikami i uprawnieniami	50
Po co nam użytkownicy i grupy?	51
Kiedy używać konta root?	52
Jak używać sudo do uruchomienia poleceń z podniesionymi uprawnieniami?	53
Tworzenie i usuwanie kont użytkowników	54
Używanie useradd	54
Korzystamy z adduser	56
Usuwanie kont użytkowników	58

Co jest w plikach /etc/passwd i /etc/shadow	60
Co jest w pliku /etc/passwd	60
Co jest w pliku /etc/shadow	62
Dostarczanie domyślnych plików konfiguracyjnych za pomocą /etc/skel	66
Przełączanie się pomiędzy kontami użytkowników	67
Zarządzanie grupami	69
Zarządzanie hasłami i zasady dotyczące haseł	73
Blokowanie i odblokowywanie kont użytkowników	73
Ustawianie informacji o wygaśnięciu hasła	74
Ustalanie zasad dotyczących haseł	76
Konfiguracja dostępu administratora za pomocą sudo	77
Ustawianie uprawnień na plikach i katalogach	81
Uprawnienia do odczytu	81
Zmiana uprawnień	85
Zmiana właściciela obiektów	87
Podsumowanie	88
Dodatkowe filmy związane z tematem	89
Lektura uzupełniająca	89

ROZDZIAŁ 3

Zarządzanie pakietami oprogramowania	90
Jak wygląda zarządzanie pakietami w systemie Linux	91
Różnice między pakietami Debiana i Snapa	92
Pakiety Debiana	93
Pakiety typu Snap	94
Instalowanie i odinstalowywanie oprogramowania	96
Zarządzanie pakietami Debiana za pomocą apt	97
Zarządzanie pakietami Snap za pomocą polecenia snap	100
Wyszukiwanie pakietów	102
Zarządzanie repozytoriami pakietów	104
Dodawanie dodatkowych repozytoriów	105
Dodawanie prywatnych archiwów pakietów	106
Tworzenie kopii zapasowej i przywracanie pakietów Debiana	108
Czyszczenie z osieroconych pakietów z użyciem apt	110
Korzystanie z aktualizacji wsparcia dla sprzętu	112
Podsumowanie	114
Filmy związane z tematem	115
Lektura uzupełniająca	115

ROZDZIAŁ 4

Nawigacja i podstawowe polecenia	116
Podstawowe polecenia systemu Linux	116
Struktura systemu plików w systemie Linux	121
Przeglądanie zawartości plików	124
Przeglądanie plików dziennika aplikacji	127
Podsumowanie	128
Odpowiedni film	128
Lektura uzupełniająca	128

ROZDZIAŁ 5

Zarządzanie plikami i katalogami	129
Kopiowanie, przenoszenie i zmiana nazw plików oraz katalogów	129
Edytowanie plików za pomocą edytorów tekstu nano i Vim	132
Edytowanie za pomocą nano	133
Edycja za pomocą Vim	135
Strumienie — wejściowy i wyjściowy	142
Używanie dowiązań symbolicznych i twardej	145
Podsumowanie	148
Dodatkowe filmy związane z tematem	149

Rozdział 6

Wydajna praca z wierszem poleceń	150
Powłoki w systemie Linux	150
O co chodzi z historią w powłoce Bash?	152
Kilka przydatnych sztuczek związanych z linią poleceń	154
Zrozumieć zmienne	158
Pisanie prostych skryptów	160
Łączenie wszystkiego w całość — piszemy skrypt wykonujący kopię zapasową z użyciem rsync	166
Podsumowanie	168
Dodatkowe filmy związane z tematem	169
Lektura uzupełniająca	169

ROZDZIAŁ 7

Procesy — kontrolowanie i zarządzanie	170
Zarządzanie zadaniami	170
Polecenie ps	174
Wyświetlanie uruchomionych procesów za pomocą ps	174
Opcje dla ps	175
Zmienianie priorytetu procesów	180

Radzenie sobie z nieprawidłowo działającymi procesami	185
Zarządzanie procesami systemowymi	187
Planowanie wykonywania zadań za pomocą polecenia cron	191
Podsumowanie	194
Dodatkowe filmy związane z tematem	194
Lektura uzupełniająca	194

ROZDZIAŁ 8

Monitorowanie zasobów systemu	195
Wyświetlanie wykorzystania dysku	195
Używanie df	196
Dokładniejsza analiza wykorzystania dysku	198
Monitorowanie wykorzystania pamięci	202
Jak wygląda zarządzanie pamięcią serwera	202
Zarządzanie obszarem wymiany	204
Czym są średnie obciążenia	207
Analiza wykorzystania zasobów za pomocą htop	211
Podsumowanie	214
Dodatkowe filmy związane z tematem	215
Lektura uzupełniająca	215

ROZDZIAŁ 9

Zarządzanie wolumenami pamięci masowej	216
Dodawanie dodatkowych wolumenów pamięci masowej	217
Formatowanie i partycjonowanie urządzeń pamięci masowej	220
Tworzenie partycji	221
Formatowanie partycji	224
Montowanie i odmontowywanie wolumenów	225
Do czego służy plik /etc/fstab	228
Co jest w pliku /etc/fstab	228
Dodawanie wpisu do pliku /etc/fstab	230
Tworzenie kopii zapasowych i przywracanie wolumenów	234
Wykorzystanie LVM	236
Zaczynamy pracę z LVM	237
Formatowanie wolumenów logicznych	241
Usuwanie wolumenów za pomocą LVM	243
Migawki LVM	244
Podsumowanie	246
Dodatkowe filmy związane z tematem	247
Lektura uzupełniająca	247

ROZDZIAŁ 10

Podłączanie serwera do sieci	248
Nadawanie serwerowi nazwy hosta	249
Zarządzanie interfejsami sieciowymi	252
Przypisywanie statycznych adresów IP	257
Jak działa rozwiązywanie nazw w systemie Linux	262
Zaczynamy pracę z OpenSSH	265
Instalacja OpenSSH	265
Wydawanie poleceń za pomocą OpenSSH	267
Wprowadzenie do zarządzania kluczami SSH	269
Generowanie kluczy publicznych i prywatnych	269
Kopiowanie klucza publicznego na zdalny serwer	270
Korzystanie z agenta SSH	271
Zmiana hasła klucza OpenSSH	272
Uproszczenie nawiązywania połączeń SSH za pomocą pliku konfiguracyjnego	273
Podsumowanie	275
Dodatkowe filmy związane z tematem	275
Lektura uzupełniająca	275

ROZDZIAŁ 11

Konfigurowanie usług sieciowych	276
Projekt przydziału adresów IP	276
Konfiguracja serwera DHCP do przydzielania adresów IP	281
Dodawanie serwera DNS	287
Konfigurowanie zewnętrznego serwera DNS za pomocą bind	288
Konfigurowanie serwera DNS do obsługi intranetu i dodawanie hostów	290
Konfigurowanie bramy internetowej	295
Podsumowanie	297
Lektura uzupełniająca	298

ROZDZIAŁ 12

Udostępnianie i przesyłanie plików	299
Uwagi dotyczące serwera plików	300
Udostępnianie plików użytkownikom systemu Windows za pomocą serwera Samba	301
Konfigurowanie udziałów NFS	307
Przesyłanie plików za pomocą rsync	312
Przesyłanie plików za pomocą SCP	316
Podsumowanie	318
Dodatkowe filmy związane z tematem	318
Lektura uzupełniająca	319

ROZDZIAŁ 13

Zarządzanie bazami danych	320
Przygotowania do założenia serwera bazy danych	321
Instalowanie MariaDB	323
Pliki konfiguracyjne MariaDB	326
Zarządzanie bazami danych MariaDB	328
Konfiguracja dodatkowego serwera bazy danych	335
Podsumowanie	340
Lektura uzupełniająca	341

ROZDZIAŁ 14

Udostępnianie serwisów internetowych	342
Instalacja i konfiguracja Apache	342
Instalacja dodatkowych modułów w Apache	349
Zabezpieczanie Apache za pomocą TLS	351
Instalacja i konfiguracja NGINX	357
Instalowanie i konfigurowanie Nextcloud	361
Podsumowanie	368
Dodatkowe filmy związane z tematem	368
Lektura uzupełniająca	368

ROZDZIAŁ 15

Automatyzacja konfigurowania serwerów — Ansible	369
Dlaczego zarządzanie konfiguracją jest potrzebne?	370
Dlaczego Ansible?	371
Tworzenie repozytorium Git	373
Zaczynamy pracę z Ansible	376
Co zrobić, by serwery wykonywały Twoje polecenia?	378
Konfiguracja pliku inwentarza i konfiguracja ustawień Ansible	379
Konfiguracja serwerów będących klientami	381
Łączenie wszystkiego w całość — automatyzacja wdrożenia serwera WWW	385
Używanie metody pull w Ansible	389
Podsumowanie	393
Dodatkowe filmy związane z tematem	393
Lektura uzupełniająca	394

ROZDZIAŁ 16	
Wirtualizacja	395
Wymagania wstępne i rozważania związane z wirtualizacją	396
Konfiguracja serwera maszyn wirtualnych	398
Tworzenie maszyn wirtualnych	404
Mostkowanie sieci maszyn wirtualnych	409
Uproszczenie tworzenia maszyn wirtualnych dzięki klonowaniu	413
Zarządzanie maszynami wirtualnymi za pomocą wiersza poleceń	415
Podsumowanie	417
Dodatkowe filmy związane z tematem	417
Lektura uzupełniająca	417
ROZDZIAŁ 17	
Korzystanie z kontenerów	418
Czym jest konteneryzacja?	418
Różnice między Dockerem a LXD	420
Instalacja Dockera	422
Zarządzanie kontenerami Dockera	423
Automatyzacja tworzenia obrazów Dockera za pomocą Dockerfiles	432
Zarządzanie kontenerami LXD	434
Podsumowanie	439
Dodatkowe filmy związane z tematem	440
Lektura uzupełniająca	440
ROZDZIAŁ 18	
Zestrajanie kontenerów	441
Zestrajanie kontenerów	442
Przygotowanie środowiska laboratoryjnego do przetestowania Kubernetes	444
Użycie MicroK8s	446
Instalowanie MicroK8s w systemie Linux	447
Instalowanie MicroK8s w macOS	448
Instalowanie MicroK8s w systemie Windows	449
Praca z MicroK8s	451
Konfigurowanie klastra Kubernetes	454
Ustawienia wstępne	456
Instalowanie Kubernetes	460
Wdrażanie kontenerów z użyciem Kubernetes	466
Podsumowanie	472
Dodatkowe filmy związane z tematem	473
Lektura uzupełniająca	473

ROZDZIAŁ 19

Wdrażanie Ubuntu w chmurze	474
Różnice między lokalną infrastrukturą a środowiskiem chmury obliczeniowej	475
Istotne rozważania przy braniu pod uwagę chmury obliczeniowej jako potencjalnego rozwiązania	476
Zapoznanie się z podstawowymi koncepcjami AWS	480
Tworzenie konta w AWS	484
Zakładanie konta w AWS	485
Wdrażanie podstawowych zabezpieczeń użytkowników	488
Wybór regionu	494
Wdrażanie Ubuntu jako instancji AWS EC2	496
Konfigurowanie roli IAM dla programu Session Manager	496
Tworzenie instancji serwera Ubuntu w AWS	499
Tworzenie i wdrażanie Ubuntu AMI	509
Automatyczne skalowanie wdrożeń Ubuntu EC2 poprzez Auto Scaling	512
Tworzenie szablonu uruchamiania	513
Tworzenie grup dla potrzeb automatycznego skalowania	514
Utrzymywanie kosztów na niskim poziomie, czyli jak oszczędzać pieniądze i podejmować opłacalne decyzje	520
Wyświetlanie informacji rozliczeniowych	521
Dodawanie alertu rozliczeniowego	521
Usuwanie niepotrzebnych kopii zapasowych	522
Uruchamianie instancji EC2 tylko wtedy, gdy jest potrzebna	522
Zatrzymywanie lub przerywanie działania niepotrzebnych instancji EC2	523
Więcej o chmurze — dodatkowe zasoby do poszerzenia wiedzy	524
Szkolenia i laboratoria online	524
Certyfikacja	524
Eksperymentuj i ucz się dalej	525
Dokumentacja AWS	525
Podsumowanie	525
Lektura uzupełniająca	526

ROZDZIAŁ 20

Automatyzacja wdrożeń w chmurze z użyciem Terraform	527
Dlaczego ważne jest, aby zautomatyzować budowanie swojej infrastruktury	528
Wprowadzenie do Terraform i jak może się on wpasować w Twój warsztat pracy	529
Instalowanie Terraform	532
Automatyzacja wdrożenia instancji EC2	537
Zarządzanie grupami zabezpieczeń za pomocą Terraform	543

Używanie Terraform do niszczenia nieużywanych zasobów	546
Ansible i Terraform jako kompletne rozwiązanie do automatyzacji wdrożeń	547
Podsumowanie	550
ROZDZIAŁ 21	
Zabezpieczanie serwera	551
Zmniejszanie powierzchni ataku	552
Czym są CVE i jak reagować, gdy się pojawią	557
Instalowanie aktualizacji zabezpieczeń	558
Automatyczne instalowanie poprawek za pomocą usługi Canonical Livepatch	562
Zabezpieczanie OpenSSH	564
Instalacja i konfiguracja Fail2ban	568
Najlepsze praktyki w zabezpieczaniu serwera baz danych MariaDB	572
Konfiguracja zapory sieciowej	575
Szyfrowanie i odszyfrowywanie dysków za pomocą LUKS	578
Blokowanie sudo	581
Podsumowanie	582
Lektura uzupełniająca	582
ROZDZIAŁ 22	
Rozwiązywanie problemów z serwerami Ubuntu	584
Ocena zasięgu problemu	585
Poszukiwanie źródła problemu	587
Przeglądanie dzienników systemowych	589
Śledzenie problemów z siecią	595
Rozwiązywanie problemów z zasobami	601
Diagnozowanie uszkodzonej pamięci RAM	605
Podsumowanie	607
Lektura uzupełniająca	608
ROZDZIAŁ 23	
Zapobieganie awariom	609
Zapobieganie awariom	609
Wykorzystanie Gita do zarządzania konfiguracją	612
Wdrożenie harmonogramu tworzenia kopii zapasowych	618
Odzyskiwanie z użyciem nośników startowych	621
Podsumowanie	623
Lektura uzupełniająca	624
Skorowidz	625

Procesy — kontrolowanie i zarządzanie

Na typowym serwerze linuksowym w dowolnym momencie może być uruchomionych ponad sto procesów. Są to zarówno procesy związane z usługami systemowymi, takimi jak **NTP** (ang. *Network Time Protocol*), jak i procesy, które dostarczają na zewnątrz informacje — takie jak serwer internetowy Apache. Jako administrator serwerów Ubuntu musisz umieć zarządzać tymi procesami, a także zarządzać zasobami dla nich dostępnymi. W tym rozdziale przyjrzymy się zarządzaniu procesami, w tym poleceniu `ps`, zarządzaniu komendami kontroli zadań i innymi kwestiami.

W miarę omawiania kolejnych pojęć będę omawiać następujące tematy:

- Zarządzanie zadaniami.
- Korzystanie z polecenia `ps`.
- Zmienianie priorytetu procesów.
- Radzenie sobie z niewłaściwie działającymi procesami.
- Zarządzanie procesami systemowymi.
- Planowanie zadań za pomocą `cron`.

Aby rozpocząć naszą przygodę z zarządzaniem procesami, przyjrzymy się najpierw zarządzaniu zadaniami. Nie tylko pomoże nam to lepiej zrozumieć ich ideę, ale także pozwoli lepiej zrozumieć działanie procesów w tle i procesów pierwszoplanowych.

Zarządzanie zadaniami

Do tej pory wszystko, co robiliśmy na powłoce, od momentu uruchomienia polecenia do jego zakończenia, było widoczne na ekranie. Instalowaliśmy aplikacje, uruchamialiśmy programy i wykonywaliśmy różne polecenia. Za każdym razem podczas wykonywania zadania „traciliśmy” na chwilę kontrolę nad powłoką — mogliśmy zacząć robienie czegoś nowego dopiero po zakończeniu wykonywania poprzedniego. Na przykład, gdy

chcieliśmy zainstalować pakiet *vim-nox* za pomocą polecenia `apt install`, musieliśmy patrzeć beczynn timer, jak `apt` pobiera pakiet i instaluje go za nas.

Podczas wykonywania zadania, gdy powłoka wykonuje dla nas zleczone jej zadania, nie pozwala nam na wydanie innego polecenia. Co prawda zawsze możemy na serwerze otworzyć nową powłokę i korzystać z wielozadaniowości, mając otwarte jednocześnie dwa okna i w każdym wykonując inne zadania. Ale nie jest to najbardziej efektywna metoda korzystania z wielozadaniowości podczas pracy z wierszem poleceń.

Zamiast tego możemy zdecydować o wykonywaniu procesu w tle, bez czekania na jego zakończenie, a w międzyczasie pracować nad czymś innym. Następnie możemy wywołać z powrotem proces, aby albo dalej z nim pracować, albo sprawdzić, czy jego wykonanie zakończyło się pomyślnie. Pomyśl o tym jako o czymś podobnym do środowiska pulpitu z oknami lub interfejsów użytkownika w systemach operacyjnych Windows lub macOS. Możesz pracować z aplikacją, zminimalizować ją, aby przestała być widoczna, a następnie zmaksymalizować, aby dalej z nią pracować. Generalnie chodzi o to samo, o co chodzi w przypadku działania procesu w tle w powłoce Linuksa.

Więc jak spowodować, by proces działał w tle, a jak, by działał na pierwszym planie? Może to być problematyczne do wyjaśnienia. Moim zdaniem najłatwiejszym sposobem na nauczenie się tego jest wypróbowanie, jak to działa. Najłatwiejszym przykładem, jaki przychodzi mi do głowy, jest (po raz kolejny) użycie edytora tekstu. Obiecuję, że tym razem użycie edytora tekstu w roli przykładu nie będzie nudne. Ten przykład jest akurat niezwykle przydatny i możliwe, że stanie się rutynowym elementem, z którego będziesz korzystał na co dzień w pracy. Aby wykonać to ćwiczenie, możesz użyć dowolnego, preferowanego przez siebie edytora tekstu działającego w wierszu poleceń, takiego jak Vim lub nano. Na serwerze Ubuntu nano jest zwykle zainstalowany domyślnie, więc już go masz, jeśli chcesz z niego korzystać. Jeśli wolisz używać Vim, nie obawiaj się i zainstaluj pakiet *vim-nox* (jeśli jeszcze tego nie zrobiłeś):

```
sudo apt install vim-nox
```

Możesz zainstalować Vim zamiast *vim-nox*, ale ja zawsze decyduję się na *vim-nox*, ponieważ ma on wbudowaną obsługę języków skryptowych.

Jednak nie ma problemu, jeśli chcesz używać dowolnego edytora tekstu, z którym czujesz się komfortowo. W poniższych przykładach będę używał nano, ale jeśli używasz Vim, po prostu zamień nazwę nano na vim za każdym razem, gdy ją zobaczysz.

Tak czy inaczej, aby zobaczyć działanie w tle w akcji, otwórz swój edytor tekstu. Możesz otworzyć jakiś plik lub po prostu rozpocząć pustą sesję. (Jeśli masz wątpliwości, wpisz `nano` i naciśnij `Enter`). Mając otwarty edytor tekstu, możemy w każdej chwili „zlecić mu”, by działał w tle, naciskając `Ctrl+Z` na klawiaturze.

Jeśli używasz Vim zamiast nano, możesz zlecić mu przejście do pracy w tle tylko wtedy, gdy jesteś w trybie poleceń, ponieważ tylko wtedy *Ctrl+Z* zostanie przekazane do powłoki.

Czy widziałeś, co się stało? Edytor znikł z ekranu i powróciłeś do powłoki, więc możesz teraz dalej wykonywać polecenia. Powinieneś zobaczyć na wyjściu komunikat podobny do poniższego:

```
[1]+ Stopped nano
```

Widzimy tutaj numer **zadania** (ang. *job*) naszego procesu, jego status, a następnie nazwę procesu. Nawet jeśli proces Twojego edytora tekstu pokazuje status Stopped, to nadal jest uruchomiony. Możesz to potwierdzić za pomocą następującego polecenia:

```
ps au | grep nano
```

W moim przypadku widzę, że proces nano działa z identyfikatorem procesu (PID) o numerze 43231:

```
jay 43231 0.0 0.1 5468 3632 pts/0 T 11:27 0:00 nano
```

W tym momencie mogę wykonywać dodatkowe polecenia, poruszać się po moim systemie plików i wykonywać inną pracę. Kiedy chcę przywrócić mój edytor tekstu, mogę użyć polecenia *fg*, aby przywrócić proces na pierwszy plan, co spowoduje wznowienie pracy edytora. Jeśli mam wiele procesów w tle, polecenie *fg* przywróci ten, z którym ostatnio pracowałem.

Użyłem polecenia *ps*, aby pokazać Ci, że proces nadal działa w tle, ale w rzeczywistości istnieje dedykowane polecenie do tego celu, a jest nim polecenie *jobs*.

Jeśli wydasz polecenie *jobs*, zobaczysz na wyjściu listę wszystkich procesów działających w tle (rysunek 7.1).



```
jay@ubuntu-server: ~
jay@ubuntu-server: $ jobs
[1]- Stopped nano plik1.txt
[2]+ Stopped nano plik2.txt
jay@ubuntu-server: $
```

Rysunek 7.1. Uruchamianie polecenia *jobs* po przestaniu do pracy w tle dwóch procesów nano

Na wyjściu widać, że korzystam z dwóch sesji nano, w jednej mam otwarty *plik1.txt*, a w drugiej *plik2.txt*. Gdybym wykonał polecenie *fg*, przywołałbym sesję nano, w której mam otwarty *plik2.txt*, ponieważ była to ostatnia sesja, w której pracowałem. To może, ale nie musi być ten plik, do którego edycji chcę wrócić. Ponieważ po lewej stronie

mam identyfikator zadania, to używając tego identyfikatora, mogę za pomocą polecenia `fg` przywołać określony proces działający w tle:

fg 1

Umiejętność przełączania procesów na działanie w tle może usprawnić wykonywanie przez Ciebie zadań w pracy. Na przykład założymy, że edytujesz plik konfiguracyjny dla aplikacji serwerowej, takiej jak Apache. Podczas edycji tego pliku konfiguracyjnego musisz sprawdzić dokumentację (stronę `man`) dla Apache, ponieważ zapomniałeś składni jakiegoś polecenia. Mógłbyś otworzyć nową powłokę i nową sesję SSH do serwera i przeglądać dokumentację w innym oknie. Ale jeśli otworzysz zbyt wiele powłok, może zrobić się zamieszanie. O wiele prościej jest przełączyć do działania w tle bieżącą sesję `nano`, przeczytać dokumentację, a następnie przywrócić na pierwszy plan proces poleceniem `fg` i kontynuować w nim pracę, i to wszystko w ramach jednej sesji SSH!

Aby proces działał w tle, nie musisz używać `Ctrl+Z`; właściwie to możesz ustawić proces, aby działał w tle od razu po uruchomieniu; wystarczy wpisać polecenie z symbolem *ampersand* (`&`) na końcu. Aby pokazać, jak to działa, użyję jako przykładu `htop`. Trzeba przyznać, że nie jest to najbardziej praktyczny przykład, ale działa, i wystarcza do pokazania, jak uruchomić proces od razu w tle.

Być może nie masz jeszcze zainstalowanego pakietu `htop` — w tym wypadku zachęcam do jego zainstalowania, a następnie uruchomienia go z symbolem `&`:

```
sudo apt install htop
htop &
```

Pierwsza komenda, jak już wiesz, instaluje pakiet `htop` na naszym serwerze. Drugim poleceniem uruchamiam `htop`, ale natychmiast przełączam go do działania w tle. Tym, co zobaczę, gdy będzie działał w tle, jest jego identyfikator zadania i identyfikator procesu (więcej na ten temat w następnym rozdziale). Teraz w każdej chwili mogę przywrócić `htop` na pierwszy plan za pomocą `fg`. Ponieważ właśnie przełączyłem go, by działał w tle, `fg` go przywróci, ponieważ uważa go za ostatni proces, który został przełączony do działania w tle. Jak już wiesz, aby go przywrócić, gdyby nie był ostatnim, mógłbym odwołać się do ID jego zadania za pomocą polecenia `fg`. Dobrze by było, gdybyś teraz poćwiczył używanie poleceń z symbolem `&`, a następnie przywracanie procesu na pierwszy plan. W przypadku `htop` użyteczne może być uruchomienie go, przełączenie do działania w tle, a następnie przywrócenie na pierwszy plan w momencie, gdy potrzebujesz sprawdzić wydajność swojego serwera.

Pamiętaj jednak, że po opuszczeniu powłoki wszystkie procesy działające w tle zostaną zamknięte. Jeśli masz niezapisaną pracę w edytorze tekstu, stracisz to, nad czym pracowałeś. Z tego powodu, jeśli używasz procesów w tle, możesz sprawdzić, czy masz jakieś oczekujące zadania, wykonując polecenie `jobs` (dosł. zadania), zanim się wylogujesz.

Prawdopodobnie zauważysz, że niektóre aplikacje bezproblemowo działają w tle, podczas gdy inne nie. W przypadku używania edytora tekstu i htop te aplikacje po przełączeniu na działanie w tle zostają wstrzymane (ang. *paused*), pozwalając nam na wykonywanie innych zadań i późniejszy powrót do tych poleceń. Jednak niektóre aplikacje niezależnie od tego, czy działają w tle, czy nie, mogą nadal regularnie wyświetlać diagnostyczny tekst w Twoim głównym oknie. Aby uzyskać jeszcze większą kontrolę nad sesjami Bash, możesz nauczyć się używać multipleksera, takiego jak *tmux* lub *screen*, aby umożliwić tym procesom uruchamianie się w ich własnych sesjach, tak by nie przeszkadzały Ci w pracy. Używanie programu takiego jak *tmux* wykracza poza zakres tej książki, ale jest to przydatne narzędzie, które warto poznać.

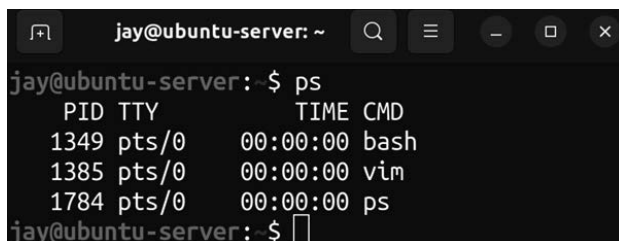
Możliwość ustawienia pracy procesu w tle i na pierwszym planie pozwala nam na bardziej efektywne zarządzanie zadaniami z linii poleceń i jest zdecydowanie przydatne. Teraz możemy szerzej spojrzeć na przeglądanie procesów na serwerze, w tym tych, które nie są przez nas uruchamiane ręcznie, jak to miało miejsce w przypadku edytora tekstu. W następnej sekcji przyjrzymy się poleceniu *ps*, które może pomóc w zrozumieniu, co tak naprawdę jest uruchomione na naszym serwerze.

Polecenie ps

Podczas zarządzania serwerem musisz rozumieć, jakie procesy są uruchomione i jak nimi zarządzać. W dalszej części tego rozdziału zajmiemy się uruchamianiem, zatrzymywaniem i monitorowaniem procesów. Jednak zanim przejdziemy do tych koncepcji, musisz najpierw być w stanie określić, co tak naprawdę jest uruchomione na serwerze. Do tego służy polecenie *ps*.

Wyświetlanie uruchomionych procesów za pomocą ps

Polecenie *ps* wywołane bez parametrów pokaże listę procesów uruchomionych przez użytkownika, który wywołał polecenie (rysunek 7.2).



```
jay@ubuntu-server: ~
jay@ubuntu-server:~$ ps
  PID TTY          TIME CMD
 1349 pts/0    00:00:00 bash
 1385 pts/0    00:00:00 vim
 1784 pts/0    00:00:00 ps
jay@ubuntu-server:~$
```

Rysunek 7.2. Wyjście polecenia *ps* po uruchomieniu jako zwykły użytkownik i bez żadnych opcji

Na rysunku 7.2 widać, że gdy uruchomiłem polecenie `ps` na koncie mojego użytkownika bez żadnych opcji, pokazało mi ono listę procesów, które uruchamiam jako ja sam. W tym przypadku mam otwartą sesję `vim` (działającą w tle), a w ostatnim wierszu widzimy również samo `ps`, które jest również pokazywane.

Po lewej stronie wyświetlonych informacji zobaczysz numer każdego z uruchomionych procesów. Jest to tzw. **identyfikator procesu** (ang. *process ID*, PID), o którym wspominaliśmy w podrozdziale „Zarządzanie zadaniami”. PID jest czymś, co naprawdę powinieneś znać, więc równie dobrze możemy go omówić teraz, zanim przejdziemy dalej.

Każdy proces uruchomiony na serwerze ma przypisany PID, który odróżnia go od innych procesów w systemie. Możesz rozróżniać procesy po nazwach — jako `vim` lub `top` (lub pod jakimikolwiek innymi nazwami). Jednak serwer rozróżnia procesy po ich ID. Kiedy otwierasz program lub uruchamiasz proces, otrzymuje on PID od jądra. Podczas pracy nad zarządzaniem serwerem przekonasz się, że znajomość PID jest przydatna, zwłaszcza w przypadku poleceń, które omówimy w tym rozdziale. Jeśli chcesz na przykład zabić źle działający proces, typowym sposobem postępowania będzie znalezienie PID tego procesu, a następnie odwołanie się do niego, gdy wydajesz polecenie, by go zabić (jak to zrobić, pokażę w dalszej części rozdziału). W rzeczywistości PID jest bardziej złożony niż tylko reprezentujący go numer przypisany do uruchomionego procesu, ale na potrzeby tego rozdziału wystarczy, że zapamiętasz, iż proces jest identyfikowany jako numer.

Jeśli znasz nazwę procesu, możesz również użyć polecenia `pidof`, aby znaleźć PID procesu. Na przykład pokazałem Ci zrzut ekranu procesu `vim` uruchomionego z PID o wartości 1385. Możesz również odczytać jego PID, uruchamiając następujące polecenie:

```
pidof vim
```

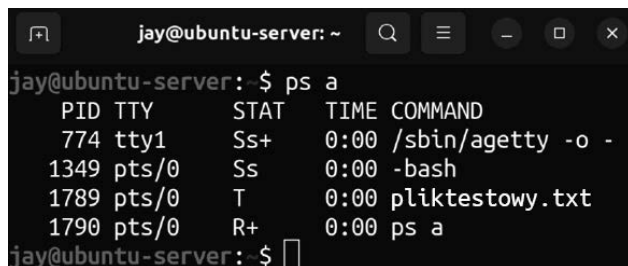
W odpowiedzi otrzymasz PID procesu bez konieczności używania polecenia `ps`.

Opcje dla `ps`

Pracując z poleceniem `ps`, warto znać kilka użytecznych opcji, które możesz dodać, aby zmienić sposób, w jaki `ps` wyświetla dane na wyjście. Jeśli użyjesz opcji `a`, zobaczysz więcej informacji niż dla samego `ps`:

```
ps a
```

Spowoduje to wyświetlenie danych wyjściowych pokazanych na rysunku 7.3.



```
jay@ubuntu-server: ~  
jay@ubuntu-server: ~$ ps a  
PID TTY STAT TIME COMMAND  
774 tty1 Ss+ 0:00 /sbin/agetty -o -  
1349 pts/0 Ss 0:00 -bash  
1789 pts/0 T 0:00 pliktestowy.txt  
1790 pts/0 R+ 0:00 ps a  
jay@ubuntu-server: ~$
```

Rysunek 7.3. Wynik po wydaniu polecenia `ps a`

Po użyciu polecenia `ps a` widzimy to samo co poprzednio, ale dodatkowo mamy informacje oraz nagłówki kolumn na górze. Teraz widzimy nagłówki dla PID, TTY, STAT, TIME i COMMAND. Dzięki temu możesz zobaczyć, że procesy `vim`, które mam uruchomione, edytują plik o nazwie `pliktestowy.txt`. Jest to świetna informacja, ponieważ jeśli miałbym otwartych więcej sesji `vim` i jedna z nich zachowywałaby się niewłaściwie, prawdopodobnie chciałbym wiedzieć, którą z nich konkretnie muszę zatrzymać.

Widzieliśmy już pola PID i COMMAND, choć nie widzieliśmy formalnego nagłówka u góry. PID został już omówiony, więc nie będę wchodził w dodatkowe szczegóły na ten temat. Pole COMMAND informuje nas o aktualnie wykonywanym poleceniu, co jest bardzo przydatne, jeśli chcemy się upewnić, że zarządzamy właściwym procesem, lub zobaczyć, co uruchamia konkretny użytkownik (wkrótce pokażę, jak wyświetlić procesy innych użytkowników).

Pole STAT jest czymś nowym; nie widzieliśmy go podczas uruchamiania `ps` bez opcji. Pole STAT to kod statusu procesu, który odnosi się do tego, w jakim stanie aktualnie znajduje się proces. Stanem tym może być permanentne uśpienie (D), nieaktywny (Z), zatrzymany (T), drzemka (S) oraz w kolejce do uruchomienia (R). Jest jeszcze stronicowanie (w), ale ponieważ nie jest już używane, nie widzę potrzeby, by je omawiać. Permanentne uśpienie to stan, w którym zasadniczo proces oczekuje na dane na wejście i nie może obsługiwać dodatkowych sygnałów (o sygnałach krótko powiem w dalszej części tego rozdziału). Proces nieaktywny (określany również jako *proces zombie*), to proces, który wykonał wszystko, co miał do wykonania, zakończył swoją pracę, ale czeka na proces macierzysty, aby zostać usuniętym. Nieaktywne procesy de facto nie działają — nie są uruchomione, ale jeszcze pozostają na liście procesów i normalnie powinny się same zakończyć. Jeśli taki proces widnieje stale na liście i nie zamyka się, może być kandydatem do użycia do niego polecenia `kill` (zabij), które omówimy później. Taki proces to na ogół proces, który został przełączony do pracy w tle — omówię to w następnym rozdziale. Przerzywalne uśpienie oznacza, że program jest beczynny: czeka na dane na wejściu, które go wybudzą.

Kolumna TTY mówi nam, do którego TTY dany proces jest podłączony. TTY odnosi się do **dalekopisu** (ang. *teletypewriter*), który jest terminem pochodzącym z innych czasów.

W przeszłości, w czasach wielkich komputerów mainframe, użytkownicy korzystali z takich komputerów za pomocą „terminali” — urządzeń składających się z monitora i klawiatury, podłączonych (przez kabel) do mainframe. Takie urządzenia mogły jedynie wyświetlać dane wyjściowe otrzymywane z komputera mainframe i odbierać dane wpisywane na klawiaturze. Teletypewriter był terminem stosowanym w odniesieniu do takich urządzeń. Oczywiście obecnie nie używamy takich maszyn, ale koncepcja działania jest podobna.

Na serwerze używamy klawiatury do wysyłania danych wejściowych do urządzenia, które następnie wyświetla nasze dane wyjściowe innemu urządzeniu. W tym przypadku urządzeniem wejściowym jest nasza klawiatura, a urządzeniem wyjściowym jest nasz ekran, który jest albo bezpośrednio podłączony do serwera, albo znajduje się na naszym komputerze połączonym z naszym serwerem poprzez usługę taką jak SSH. W systemie Linux większość procesów działa na TTY, który jest (praktycznie rzecz biorąc) terminalem. Przechwytuje on dane wejściowe i zarządza wyjściowymi, podobnie jak to miało miejsce w dalekopisie. Terminal jest metodą, którą wykorzystujemy do interakcji z serwerem.

Na rysunku 7.3 mamy proces działający na TTY o nazwie `tty1`, a pozostałe procesy działają na `pts/0`. TTY które widzimy, to rzeczywiste urządzenia terminalowe, a `pts` odnosi się do wirtualnego (pseudo-) urządzenia terminalowego. Nasz serwer jest w stanie uruchomić kilka sesji `tty`, zazwyczaj od jednej do siedmiu. Każda z nich może uruchamiać swoje własne programy i procesy. Aby to lepiej zrozumieć, spróbuj wcisnąć `Ctrl+Alt+` dowolny klawisz funkcyjny, od `F1` do `F7` (jeśli masz fizyczną klawiaturę podłączoną do fizycznego serwera). Za każdym razem powinieneś zobaczyć czysty ekran, każdy przeniesiony na inny terminal. Każdy z tych terminali działa niezależnie od innych. Każdy z Twoich klawiszy funkcyjnych reprezentuje konkretny TTY, więc naciskając `Ctrl+Alt+F6`, widzisz na ekranie to, co jest na TTY 6.

Zasadniczo możesz przełączać się między TTY od TTY 1 aż do TTY 7, każdy może mieć swoje własne uruchomione procesy. Jeśli ponownie uruchomisz `ps a`, zobaczysz, że wszystkie procesy uruchomione na tych TTY pokażą się na wyjściu jako sesje `tty`, takie jak `tty2` lub `tty4`. Procesy uruchamiane w emulatorze terminala otrzymają oznaczenie `pts`, ponieważ nie są uruchamiane w rzeczywistym TTY, ale raczej w pseudo-TTY.

To był dość długi wywód o czymś, co prowadzi do prostego rozróżnienia (TTY lub pseudo-TTY), ale z tą wiedzą powinieneś być w stanie odróżnić, który proces jest uruchomiony na serwerze, a który przez powłokę.

W ramach kontynuacji opisu kolumn z rysunku 7.3 spójrzmy na pole `TIME`. Pole to reprezentuje całkowity czas, przez jaki dany proces wykorzystywał procesor. Jednakże na zrzucie ekranu, który pokazałem, czas wynosi 0:00 dla każdego z procesów. Na początku może to być mylące. W moim przypadku procesy obsługujące `vim` były uruchomione

około 15 minut przed zrobieniem zrzutu ekranu, ale mimo to nadal pokazują czas wykorzystania 0:00. Powodem jest to, że to nie jest czas działania procesu, ale raczej czas, w którym proces aktywnie angażuje procesor. W przypadku `vim` każdy z procesów to po prostu bufor z otwartym plikiem. Dla porównania na maszynie z systemem Linux, na której piszę ten rozdział, jest proces o identyfikatorze ID 759 z czasem 92:51. PID 759 jest procesem mojego serwera *X* (ang. *X server*), który jest podstawą mojego **graficznego interfejsu użytkownika** (ang. *graphical user interface*, **GUI**) odpowiadającego za możliwość pracy w środowisku okienek. Jednak ten laptop pracuje obecnie od 6 dni i 22 godzin, co w przybliżeniu odpowiada 166 godzinom, a więc nie jest to ten sam czas, który PID 759 zgłasza w swoim wpisie `TIME`. Możemy zatem z tego wywnioskować, że mimo iż mój laptop działał przez 6 dni z rzędu, serwer *X* wykorzystał jedynie 92 godziny i 51 minut rzeczywistego czasu pracy procesora. Podsumowując, kolumna `TIME` odnosi się do czasu, w którym proces potrzebuje procesora, aby coś obliczyć, i niekoniecznie pokazywana w niej wartość jest równa temu, jak długo coś jest uruchomione lub jak długo proces graficzny jest wyświetlany na ekranie.

Wróćmy do polecenia `ps` i spójrzmy na kilka dodatkowych opcji. Najpierw zobaczymy, co otrzymamy, gdy do naszego poprzedniego przykładu dodamy opcję `u`, co daje nam następujące przykładowe polecenie:

```
ps au
```

Spowoduje to wygenerowanie informacji podobnych do pokazanych na rysunku 7.4.

```

jay@ubuntu-server: ~
┌───┴───┐
jay@ubuntu-server: ~$ ps au
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         774  0.0  0.0   6172   1116 tty1      Ss+  01:06   0:00 /sbin/agetty -o -p -- \u --no
jay       1349  0.0  0.2   8732   5324 pts/0    Ss   02:03   0:00 -bash
jay       1789  0.0  0.6  31480  12920 pts/0    T    03:20   0:00 pliktestowy.txt
jay       1793  0.0  0.0   10068   1544 pts/0    R+   03:22   0:00 ps au
jay@ubuntu-server: ~$

```

Rysunek 7.4. Wynik działania polecenia `ps au`

Kiedy uruchomisz to polecenie, powinieneś od razu zauważyć różnicę w stosunku do polecenia `ps`. Dzięki dodaniu tej opcji zobaczysz na liście procesy wraz z dodatkową informacją będącą identyfikatorami użytkowników, którzy je uruchomili. Uruchamiając polecenie z tą opcją, widzę procesy mojego użytkownika (`jay`), jak również jeden dla użytkownika `root`. Opcji `u` będziesz często używać, ponieważ przez większość czasu, kiedy zarządzasz serwerami, prawdopodobnie bardziej niż przyglądanie się procesom systemowym interesuje Cię obserwowanie, co kombinują użytkownicy Twojego serwera. Ale prawdopodobnie najczęstszym zastosowaniem polecenia `ps` jest jego następująca odmiana:

```
ps aux
```

Po dodaniu opcji `x` nie ograniczamy już naszego wyjścia do procesów w obrębie TTY (czy to natywnych, czy pseudo). W rezultacie zobaczymy dużo więcej procesów, w tym procesy systemowe, które nie są związane z procesem, który sami uruchomiliśmy. Bardzo proszę, wypróbuj to. W praktyce polecenie `ps aux` jest jednak najczęściej używane z `grep` do wyszukania konkretnego procesu lub ciągu znaków. Załóżmy, że chcesz zobaczyć listę wszystkich procesów roboczych `nginx`. Aby to zrobić, możesz wydać następujące polecenie:

```
ps aux | grep nginx
```

Tutaj wykonujemy tak jak poprzednio polecenie `ps aux`, ale przesyłamy wyjście do `grep`, gdzie szukamy tylko tych linii wyjścia, które zawierają ciąg `nginx`. W praktyce jest to sposób, w jaki zarówno ja, jak i inni administratorzy (na ile zauważyłem) często używamy `ps`. Używając `ps aux`, jesteś w stanie zobaczyć dużo więcej danych wyjściowych, a następnie możesz zawęzić wyniki za pomocą kryteriów wyszukiwania poprzez przesłanie strumienia (ang. *piping*) do `grep`. Jeśli jednak chcesz zobaczyć tylko te procesy, które zawierają określony ciąg znaków, możesz to również osiągnąć za pomocą polecenia:

```
ps u -C nginx
```

W ten sposób otrzymamy wynik zawierający listę procesów pasujących do `nginx` oraz związane z nimi szczegóły. Inną użyteczną odmianą polecenia `ps` jest posortowanie wyniku poprzez malejące posortowanie procesów z wykorzystaniem CPU jako kryterium:

```
ps aux --sort=-pcpu
```

Niestety, polecenie to pokazuje wiele danych wyjściowych i musielibyśmy przewijać je do góry, aby zobaczyć najważniejsze procesy. W zależności od ustawień Twojego terminala możesz nie mieć możliwości przewijania wyników do początkowych (lub przewijania w ogóle), więc poniższa komenda zawęzi je jeszcze bardziej:

```
ps aux --sort=-pcpu | head -n 5
```

Teraz wynik jest przydatny! W tym przykładzie używam polecenia `ps aux` z opcją `-sort`, sortując według procentowego wykorzystania procesora (`-pcpu`). Następnie przekazuję wyjście do polecenia `head`, gdzie instruuję je, aby pokazało mi tylko pięć linii (`-n 5`). W efekcie otrzymuję listę pięciu procesów, które zużyły najwięcej mocy procesora od czasu startu systemu. Mogę zrobić to samo, sprawdzając wykorzystanie pamięci:

```
ps aux --sort=-pmem | head -n 5
```

Jeśli chcesz sprawdzić, które procesy zachowują się niewłaściwie i używają nietypowej ilości pamięci lub procesora, te polecenia pomogą Ci zawęzić listę. Polecenie `ps` jest bardzo użytecznym poleceniem w zestawie narzędzi administratora. Nie krępuj się eksperymentować z nim, nie ograniczając się do przykładów, które podałem; możesz

zajrzeć na strony *man* polecenia `ps`, aby nauczyć się jeszcze więcej sztuczek. W drugiej sekcji strony *man* dla `ps` — pod *Examples* (przykłady) — znajdziesz nawet więcej ciekawych przykładów do wypróbowania.

Teraz gdy wiesz już, jak sprawdzać uruchomione procesy, w następnej części przyjrzymy się temu, jak zmienić priorytety procesów, aby procesor mógł się wydajniej zająć tymi, które są ważniejsze.

Zmienianie priorytetu procesów

Procesy w systemie Linux mogą być uruchamiane z priorytetem (który można zmienić), co daje niektórym procesom większy priorytet, a innym mniejszy. Daje też to Tobie, jako administratorowi, pełną swobodę, jeśli chodzi o zapewnienie, że najważniejsze procesy w systemie będą uruchamiane z priorytetem na odpowiednim poziomie. Do tego celu służą specjalne polecenia: `nice` i `renice`. Polecenia te pozwalają odpowiednio: uruchomić proces o określonym priorytecie lub zmienić priorytet już uruchomionego procesu.

W dzisiejszych czasach ręczna edycja priorytetu procesu jest czymś, co administratorzy robią rzadziej niż kiedyś. Procesor z 32 rdzeniami (lub jeszcze więcej) nie jest rzadkością, podobnie jak setki gigabajtów pamięci RAM. Dzisiejsze serwery są z pewnością bardziej wydajne niż kiedyś, a przy tym nie „cierpią” na niedostatek zasobów sprzętowych jak dawniej. Wiele serwerów (uruchomionych jako maszyny wirtualne) i kontenerów jest dedykowanych do pojedynczego zadania, więc dostrajanie priorytetów procesów straciło na znaczeniu. Jednak firmy przetwarzające dane i wykorzystujące funkcje uczenia głębokiego mogą znaleźć się w sytuacji, w której będą musiały dostosować niektóre parametry.

Niezależnie od tego, czy nadawanie priorytetów procesom jest czymś, z czego natychmiast skorzystasz, czy nie, dobrze by było rozumieć tę ideę — na wypadek gdybyś kiedyś potrzebował zwiększyć lub zmniejszyć priorytet jakiegoś procesu. Powróćmy do polecenia `ps`, tym razem z argumentem `-l`:

```
ps -l
```

Dane wyjściowe po wydaniu tego polecenia będą wyglądały tak jak na rysunku 7.5.

```
jay@ubuntu-server: ~
jay@ubuntu-server: $ ps -l
F S  UID      PID      PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    1349    1348  0  80   0  - 2183 do_wai pts/0        00:00:00 bash
0 T  1000    1789    1349  0  80   0  - 7870 do_sig pts/0        00:00:00 vim
0 R  1000    1794    1349  0  80   0  - 2517 -      pts/0        00:00:00 ps
jay@ubuntu-server: $
```

Rysunek 7.5. Wynik po wydaniu polecenia `ps -l`

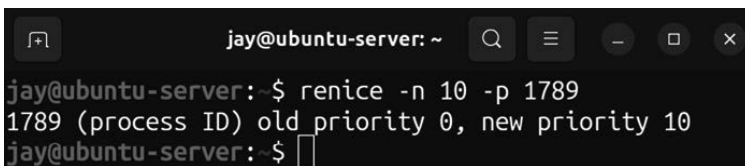
W wynikach, które pojawiły się po użyciu polecenia `ps -l`, możesz zobaczyć kolumny PRI i NI. PRI odnosi się do priorytetu (ang. *priority*), a NI odnosi się do wartości parametru określającego „uprzejmość” (ang. *niceness*), którą omówię bardziej szczegółowo w dalszej części tego rozdziału. W tym przykładzie każdy proces ma PRI równe 80 i NI równe 0. Nie modyfikowałem żadnej z tych wartości; są to wartości, które otrzymuję, gdy uruchamiam procesy bez żadnych specjalnych zmian. Wartość PRI równa 80 jest domyślną wartością początkową dla wszystkich procesów i będzie się zmieniać, gdy będziemy zwiększać lub zmniejszać wartość parametru uprzejmości.

Jak już wspominałem, mamy dedykowane komendy, które pozwalają nam zmieniać priorytety, mianowicie `nice` i `renice`. To, której z nich użyć, zależy od tego, czy proces jest już uruchomiony, czy nie. Jeśli chodzi o procesy wymienione na rysunku 7.5, trzeba użyć `renice`, aby zmienić ich priorytet, ponieważ wszystkie są już uruchomione. Jeśli chcielibyśmy uruchomić proces i nadać mu od początku określony priorytet, użyłbyśmy `nice`.

Na przykład zmodyfikujmy proces sesji `vim`, którą mam uruchomioną. Oczywiście, jest to trochę kiepski przykład, ponieważ `vim` nie jest bardzo ważnym procesem. W prawdziwym świecie nadawałbyś priorytety procesom, które są rzeczywiście ważne. W moim przypadku, ponieważ proces `vim` ma PID o wartości 1789, polecenie, które musiałbym uruchomić, aby zmienić „uprzejmość” procesu, wyglądałoby tak:

```
renice -n 10 -p 1789
```

Dane wyjściowe po wykonaniu tego polecenia będą wyglądały jak na rysunku 7.6.



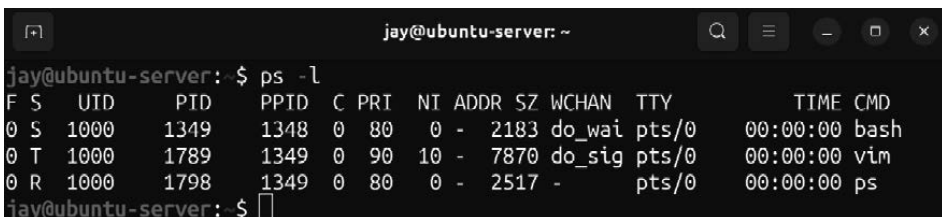
```

jay@ubuntu-server: ~
jay@ubuntu-server:~$ renice -n 10 -p 1789
1789 (process ID) old priority 0, new priority 10
jay@ubuntu-server:~$

```

Rysunek 7.6. Zmiana priorytetu procesu za pomocą `renice`

Jeśli uruchomisz ponownie `ps -l`, zobaczysz nową, ładną wartość dla `vim` (rysunek 7.7).



```

jay@ubuntu-server: $ ps -l
F S  UID    PID    PPID  C  PRI  NI ADDR  SZ  WCHAN  TTY      TIME CMD
0 S  1000   1349   1348  0   80   0 - 2183 do_wai pts/0    00:00:00 bash
0 T  1000   1789   1349  0   90  10 - 7870 do_sig pts/0    00:00:00 vim
0 R  1000   1798   1349  0   80   0 - 2517 -      pts/0    00:00:00 ps
jay@ubuntu-server: $

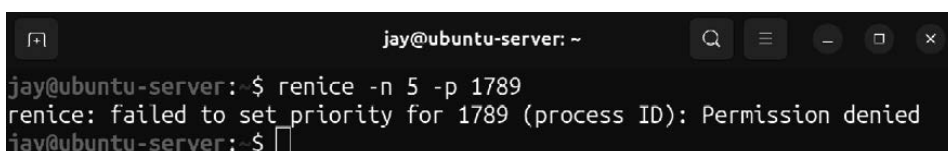
```

Rysunek 7.7. Wyjście polecenia `ps -l` po zmianie priorytetu procesu

Teraz dla vim mamy nową wartość uprzejmości, jest ona równa 10 i pojawia się pod NI, a wartość PRI wzrosła do 90. Od teraz ta instancja vim będzie działać z niższym priorytetem niż inne moje zadania: im wyższa wartość uprzejmości, tym niższy priorytet. Zauważ, że kiedy zmieniłem priorytet tym poleceniem, nie użyłem sudo. W tym przykładzie jest to w porządku, ponieważ zwiększam wartość uprzejmości procesu, a to jest dozwolone. Spróbuję jednak zmniejszyć wartość uprzejmości bez sudo, używając następującego polecenia:

```
renice -n 5 -p 1789
```

Jak widać na rysunku 7.8, to już się nie udaje.

A terminal window titled 'jay@ubuntu-server: ~' with search, menu, and window control icons. The terminal shows the command 'renice -n 5 -p 1789' being entered. The output is 'renice: failed to set priority for 1789 (process ID): Permission denied'. The prompt returns to '\$'.

Rysunek 7.8. Próba zmniejszenia priorytetu dla procesu

Moja próba zmniejszenia wartości uprzejmości z 10 do 5 została zablokowana. Gdybym był w stanie obniżyć uprzejmość, wtedy spowodowałbym, że mój proces zacząłby działać z wyższym priorytetem. Zamiast tego otrzymałem jednak komunikat o błędzie *Permission denied*. Tak więc widać, że zasadniczo użytkownicy mogą zwiększać uprzejmość swoich procesów, ale nie mogą jej zmniejszać, nawet dla procesów, które sami zainicjowali. Jeśli chcesz zmniejszyć wartość uprzejmości, musisz to zrobić, używając sudo. Więc zasadniczo, jeśli chcesz być „bardziej uprzejmy”, możesz to zrobić. Jeśli chcesz być „bardziej wredny”, będziesz potrzebował uprawnień użytkownika *root*. Na dodatek użytkownik nie może zmienić priorytetu procesu, którego nie jest właścicielem. Jeśli więc spróbujesz użyć *renice* do zmiany priorytetu zadania uruchomionego przez innego użytkownika, otrzymasz komunikat o błędzie: *Operation not permitted*.

W tym momencie wiesz już, jak zmienić priorytet uruchomionych procesów za pomocą *renice*. Teraz zobacz, jak za pomocą *nice* uruchomić nowy proces z określonym priorytetem. Spójrz na następujące polecenie:

```
nice -n 10 vim
```

Tutaj uruchamiamy nową instancję vim, ale od razu z priorytetem ustawionym na konkretną wartość. Jeśli chcemy później ponownie zmienić priorytet vim, będziemy musieli użyć *renice*. Jak wspomniałem wcześniej, *nice* służy do uruchomienia nowego procesu z określonym priorytetem, a *renice* służy do zmiany priorytetu wcześniej uruchomionego procesu. W tym przykładzie uruchomiliśmy vim i ustawiliśmy jego wartość uprzejmości na 10, używając jednego polecenia.

Zmiana priorytetu edytora tekstu takiego jak Vim może wydawać się dziwnym wyborem jako przykład i faktycznie tak jest. Ale eksperymentowanie z priorytetem procesu edytora Vim jest nieszkodliwe, ponieważ prawdopodobieństwo, że zmiana jego priorytetu doprowadzi do zatrzymania systemu, jest bardzo niewielkie. Nie przychodzi mi do głowy żadne praktyczne, użyteczne zastosowanie zmiany priorytetu czegoś takiego jak edytor tekstu. Najważniejsze jest jednak to, że *możesz* zmienić priorytet procesów działających na Twoim serwerze. Na prawdziwym serwerze możesz mieć ważny proces, który działa i generuje raport, a ten raport musi być dostarczony na czas. A może masz proces, który odpowiada za eksport danych, które muszą na czas dotrzeć do klienta. Tak więc, jeśli pomyślisz o tym szerzej, możesz zastąpić `vim` nazwą procesu, który jest rzeczywiście istotny dla Ciebie lub Twojej organizacji.

Być może zastanawiasz się, co oznacza *nice* w kontekście poleceń `nice` i `renice`. Liczba *nice* zasadniczo odnosi się do tego, na ile uprzejmy jest proces dla innych użytkowników. Im wyższa wartość *nice*, tym niższy priorytet. Tak więc proces o wartości uprzejmości 20 będzie bardziej uprzejmy niż ten o wartości 10. W tym przypadku procesy o wartości 20 mają niższy priorytet, a więc są bardziej uprzejme dla innych procesów w systemie. Parametr uprzejmość może przyjmować wartości z zakresu od -20 do 19. Proces o wartości uprzejmości -20 ma najwyższy możliwy priorytet, natomiast 19 to najniższy priorytet, jaki może przyjąć. Cały system jest bardziej skomplikowany niż ten prosty opis. Chociaż określam wartość uprzejmości jako priorytet procesu, w rzeczywistości nim nie jest. Wartość uprzejmości jest używana do obliczania rzeczywistego priorytetu. Ale na razie, jeśli uprościmy sprawę, przyjmując, że uprzejmość reprezentuje priorytet, przy czym niższa wartość uprzejmości oznacza wyższy priorytet, a wyższa wartość — niższy, będzie to dla nas wystarczające.

Jak dotąd używaliśmy poleceń `nice` i `renice` wraz z opcją `-n` do bezpośredniego ustawiania wartości *nice*. Interesujące może być jednak to, że można uprościć polecenie `renice` i pominąć opcję `-n`:

```
renice 10 42467
```

To polecenie ustawia wartość uprzejmości procesu na +10, podobnie jak w innych naszych przykładach. Możemy również użyć liczby ujemnej dla uprzejmości, jeśli chcemy zwiększyć priorytet procesu:

```
sudo renice -10 42467
```

Chociaż pominięcie opcji `-n` nie oszczędza nam zbyt wiele pisania na klawiaturze, to przynajmniej wiesz, że jest to możliwe. Dodatkowo w tym przykładzie musiałem użyć `sudo`, ponieważ zmniejszam wartość uprzejmości (więcej na ten temat później).

Jeśli chodzi o polecenie `nice`, możemy również pominąć opcję `-n`, ale polecenie działa w tym zakresie nieco inaczej. Nie zadziała następujące polecenie:

```
nice 15 vim
```

Składnia `nice` jest nieco inna, więc podanie jej bezpośrednio jako parametru liczby dodatniej nie zadziała tak jak w przypadku `renice`. Tutaj będziemy musieli dodać z przodu myślnik:

```
nice -15 vim
```

Patrząc na to polecenie, można pomyśleć, że ustawiamy wartość ujemną. W rzeczywistości tak nie jest. Ponieważ składnia jest inna w `nice`, użyta przez nas wartość `-15` daje w rezultacie dodatnią liczbę równą `15`. Potrzebowaliśmy myślnika przed wartością, aby poinformować polecenie `nice`, że wartość jest opcją, którą chcemy ustawić. Jeśli faktycznie chcemy użyć wartości ujemnej z `nice`, unikając jednocześnie opcji `-n`, musimy użyć dwóch myślników:

```
nice --10 vim
```

Różnica w składni między tymi dwoma poleceniami (z i bez opcji `-n`) jest moim zdaniem nieco myląca, więc polecam po prostu używanie opcji `-n` z `nice` i `renice`, ponieważ będzie to bardziej jednolite:

```
nice -n 10 vim
sudo nice -n -10 vim
renice -n 10 42467
sudo renice -n -10 42467
```

Przykłady te pokazują zarówno `nice`, jak i `renice` w wersji z użyciem opcji `-n`, gdy do ustawienia są zarówno wartości dodatnie, jak i ujemne parametru. Ponieważ opcja `-n` jest używana w ten sam sposób w obu poleceniach, może być łatwiejsza do zapamiętania niż skupianie się na detalach składni. W przypadku poleceń, które ustawiają wartość ujemną, użyłem `sudo`, ponieważ jak już wspomniałem wcześniej, tylko `root` może zmienić priorytet procesu lub go uruchomić z parametrem uprzejmości poniżej `0`. Jeśli tak czy inaczej spróbujesz to zrobić, otrzymasz błąd („nie można ustawić uprzejmości: odmowa dostępu”):

```
nice: cannot set niceness: Permission denied
```

Ten rodzaj ochrony jest dość ważny, ponieważ możesz mieć takich użytkowników, którym wydaje się, że ich procesy są najważniejsze, więc próbują nadać im priorytet, sięgając nawet po `-19`. W końcu lepiej, aby to administrator systemu podejmował decyzje, które procesy mogą mieć wartość uprzejmości na minusie.

Jako administrator serwerów Ubuntu sam decydujesz o tym, które procesy powinny być uruchomione, w jaki sposób i z jakim priorytetem. Ty odpowiadasz za to, by system działał w odpowiedni sposób, a dostosowywanie priorytetów dla procesów jest częścią konfigurowania ustawień. Niezależnie od wszystkiego polecenia `nice` i `renice` to kolejne narzędzie w Twoim niezbędniku.

Radzenie sobie z nieprawidłowo działającymi procesami

Jeśli chodzi o polecenie `ps`, w tym momencie wiesz już, jak wyświetlić procesy uruchomione na Twoim serwerze, a także jak zawęzić wyjście według łańcucha lub wykorzystania zasobów. Ale co właściwie możesz zrobić z tą wiedzą? Jakkolwiek nie chcielibyśmy się do tego przyznać, czasami procesy uruchomione na serwerze ulegają awarii lub pracują niewłaściwie i trzeba je zrestartować. Jeśli jakiś proces nie chce się normalnie zamknąć, może być konieczne jego zabicie. W tym rozdziale przedstawiamy polecenia `kill` (zabij) i `killall` (zabij wszystko), którymi można to zrobić.

Polecenie `kill` przyjmuje jako argument identyfikator PID i próbuje elegancko zamknąć proces. W typowej sytuacji, gdy trzeba zakończyć proces, który nie chce się sam zamknąć, najpierw używa się polecenia `ps`, aby znaleźć PID winowajcy. Znając już PID, możesz spróbować wydać polecenie `kill` dla tego procesu. Na przykład, jeśli PID 31258 musi zostać zabity, możesz wydać następujące polecenie:

```
sudo kill 31258
```

Jeśli wszystko pójdzie dobrze, proces zostanie zakończony. Możesz go ponownie uruchomić lub zbadać przyczynę niepowodzenia, przeglądając jego logi.

Aby lepiej zrozumieć, co robi polecenie `kill`, najpierw musisz zrozumieć koncepcję **sygnałów** w Linuksie (zwanymi także *przerwaniami programowymi* lub *sygnałami przerwań*). Sygnały są używane zarówno przez administratorów, jak i programistów i mogą być wysyłane do procesu przez jądro, inny proces lub ręcznie za pomocą polecenia. Sygnał instruuje proces o zmianie żądania, a w niektórych przypadkach o całkowitym zakończeniu. Przykładem takiego sygnału jest `SIGHUP`, który informuje procesy, że ich terminal sterujący zakończył pracę. Może to nastąpić, gdy masz otwarty emulator terminala, z kilkoma uruchomionymi w nim procesami. Jeśli zamkniesz okno terminala (bez zatrzymywania uruchomionych procesów), zostanie wysłany do nich sygnał `SIGHUP`, który w zasadzie mówi im, że mają zakończyć pracę (zasadniczo oznacza to, że powłoka zakończyła pracę lub się zawiesiła).

Innymi przykładami są `SIGINT` (w przypadku gdy aplikacja działa na pierwszym planie i jest zatrzymywana przez naciśnięcie `Ctrl+C` na klawiaturze) i `SIGTERM`, który wysyłany do procesu, prosi go o eleganckie zakończenie pracy (ang. *terminate cleanly*). Jeszcze innym przykładem jest `SIGKILL`, który zmusza proces do natychmiastowego zakończenia pracy (ang. *terminate uncleanly*). Każdy sygnał jest reprezentowany przez nazwę, a także wartością liczbową, taką jak 15 dla `SIGTERM` i 9 dla `SIGKILL`. Omawianie każdego z sygnałów wykracza poza zakres tego rozdziału (zaawansowane zagadnienia związane z przerwaniami są przydatne głównie dla programistów), ale jeśli jesteś ciekaw, możesz przeczytać więcej na ten temat, zaglądając na stronę *man*.

```
man 7 signal
```

Na potrzeby zagadnień omawianych w tym rozdziale najbardziej interesują nas dwa typy sygnałów, są to sygnały SIGTERM(15) i SIGKILL(9). Kiedy chcemy zatrzymać proces, wysyłamy do niego jeden z tych sygnałów, a komenda `kill` pozwala nam właśnie to zrobić. Domyślnie polecenie `kill` wysyła sygnał 15 (SIGTERM), który mówi procesowi, aby elegancko zakończył działanie. Jeśli się to uda, proces zwolni swoją pamięć i zakończy pracę. W naszym poprzednim przykładzie polecenia `kill` wysłaliśmy do procesu sygnał 15, ponieważ nie sprecyzowaliśmy, jaki sygnał wysłać.

Zakończenie procesu za pomocą SIGKILL(9) jest uważane za skrajną ostateczność. Gdy wysyłasz sygnał 9 do procesu, jest to odpowiednik wyrwania spod niego dywanu lub wysadzania za pomocą laski dynamitu. Proces zostanie zamknięty na siłę, nie będzie miał możliwości i czasu na żadną reakcję, jest to więc jedna z tych rzeczy, których powinieneś unikać, chyba że wcześniej spróbowałeś już wszystkiego, co tylko przyszło Ci do głowy. Teoretycznie wysłanie sygnału 9 może spowodować uszkodzenie plików, problemy z pamięcią lub inne nieprzewidziane skutki. Jeśli chodzi o mnie, nigdy nie natknąłem się na długotrwałe uszkodzenie oprogramowania po użyciu tego sygnału, ale teoretycznie może się tak zdarzyć, więc zapewne będziesz chciał go użyć tylko w ekstremalnych przypadkach. Wysłanie takiego sygnału może być konieczne w przypadku źle działającego procesu (ze statusem *defunct*) lub **procesu zombie**, gdy nie chcą się one same zamknąć. Te procesy są w zasadzie już martwe i zazwyczaj czekają na to, by zostać zamknięte przez swoje procesy macierzyste.

Jeśli proces macierzysty nigdy nie spróbuje takiego procesu zamknąć, pozostanie on na liście procesów. Samo w sobie może i nie jest to wielkim problemem, ponieważ technicznie procesy te i tak nic nie robią. Ale jeśli ich obecność powoduje problemy i nie możesz ich zabić, możesz spróbować wysłać do nich sygnał SIGKILL. Nie powinno być nic złego w wyeliminowaniu procesu zombie, ale najpierw powinieneś dać mu szansę zakończyć działanie w normalny sposób.

Aby wysłać sygnał 9 do procesu, użyj opcji `-9` w poleceniu `kill`. Nie muszę chyba dodawać, że powinieneś się upewnić, że wykonujesz je wobec procesu o właściwym ID:

```
sudo kill -9 31258
```

Po takim poleceniu proces o PID 31258 po prostu zniknie bez śladu. Wszystko, czym się zajmował, zostanie natychmiast usunięte z pamięci. Jeśli z jakiegoś powodu proces nadal będzie działał (co zdarza się niezwykle rzadko), prawdopodobnie będziesz musiał zrestartować serwer, aby się go pozbyć — widziałem taką sytuację tylko w kilku bardzo rzadkich przypadkach. Przykładem takiego procesu jest proces zombie, czyli proces, który pojawia się na liście procesów, ale wysyłanie sygnałów nie ma na niego wpływu, ponieważ i tak nie zostanie mu przydzielony czas procesora. Jeśli `kill -9` nie spowoduje pozbycia się procesu, to nic innego tego nie zrobi.

Inną metodą zabicia procesu jest polecenie `killall`, które jest prawdopodobnie bezpieczniejsze niż polecenie `kill` (choćby z tego powodu, że jest mniejsza szansa na przypadkowe zabicie niewłaściwego procesu). Podobnie jak `kill`, `killall` pozwala wysłać `SIGTERM` do procesu, ale w przeciwieństwie do `kill`, można to zrobić, podając nazwę. Ponadto `killall` nie zabija tylko jednego procesu; zabija każdy proces pasujący do (podanej jako opcja) nazwy, który znajdzie. Aby użyć `killall`, po prostu wywołujesz `killall` wraz z nazwą procesu:

```
sudo killall myprocess
```

Podobnie jak w przypadku polecenia `kill`, możesz również wysłać do procesu sygnał 9:

```
sudo killall -9 myprocess
```

Przypomnę ponownie — używaj tego polecenia tylko w razie potrzeby. W praktyce prawdopodobnie nie będziesz bardzo często używał `killall -9` (jeśli w ogóle będzie Ci potrzebne), ponieważ rzadko zdarza się, aby wiele procesów pod tą samą nazwą przestało działać. Jeśli już musisz wysłać sygnał 9, to używaj, jeśli tylko możesz, polecenia `kill`.

Polecenia `kill` i `killall` mogą być niezwykle przydatne w sytuacji, gdy proces przestanie działać, ale są to polecenia, których — miejmy nadzieję — nie będziesz musiał używać zbyt często. Takie procesy mogą pojawić się w przypadkach, gdy aplikacje mają do czynienia z sytuacją, z którą nie mogą sobie poradzić. Więc jeśli okaże się, że ciągle musisz zabijać procesy, dobrym pomysłem może być albo sprawdzenie dostępności aktualizacji pakietu odpowiedzialnego za usługę, albo sprawdzenie serwera pod kątem problemów sprzętowych.

W następnym rozdziale przyjrzymy się działającym w tle procesom systemowym świadczącym usługi dla nas lub naszych użytkowników, takim jak proces serwera WWW lub serwera DHCP.

Zarządzanie procesami systemowymi

Procesy systemowe, znane również jako **demony** (ang. *daemons*), to programy, które działają w tle na serwerze i zazwyczaj są automatycznie uruchamiane podczas startu serwera. Zazwyczaj nie zarządzamy nimi bezpośrednio, ponieważ działają one w tle i wykonują swoje zadania, z naszym udziałem lub bez niego. Na przykład, jeśli nasz serwer jest serwerem DHCP i uruchamia proces `isc-dhcp-server`, proces ten będzie działał w tle, nasłuchując żądań DHCP i obsługując żądania przydzielania adresów IP, w miarę jak są zgłaszane. W większości przypadków, gdy instalujemy aplikację, która działa jako usługa, Ubuntu skonfiguruje ją tak, by uruchamiała się podczas uruchamiania serwera, więc nie musimy uruchamiać jej samodzielnie. Zakładając, że usługa

nie napotka problemu, będzie wykonywać swoją pracę, dopóki nie zdecydujemy, by ją zatrzymać. W systemie Linux usługi są zarządzane przez system inicjujący, zwany również PID 1, ponieważ system ten w Linuksie zawsze otrzymuje taki identyfikator. W ostatnich latach sposób zarządzania procesami w serwerze Ubuntu znacznie się zmienił. Obecnie Ubuntu używa systemd dla swojego systemu `init`, a jeszcze kilka lat temu był to `Upstart`. Ubuntu 16.04 było pierwszym wydaniem Ubuntu o długim okresie wsparcia mającym `systemd`, i jest on nadal jest używany w Ubuntu 22.04. Ponieważ `systemd` jest już od dłuższego czasu standardem, skupimy się na związanych z nim poleceniach, używanych do zarządzania naszymi usługami. Starsze systemy `init` są już „na wymarciu”.

W `systemd` usługi są znane jako **jednostki** (ang. *units*), ale w większości przypadków terminy „usługa”, „demon” i „jednostka” oznaczają w zasadzie to samo. Ponieważ zacząłem używać Linuksa ponad 20 lat temu, nadal z przyzwyczajenia używam nazwy „usługa” dla jednostek `systemd`. Aby pomóc nam zarządzać tymi „jednostkami”, `systemd` obsługuje polecenie `systemctl`, dzięki któremu możemy uruchamiać, zatrzymywać i przeglądać status [stan — przyp. tłum.] jednostek na serwerze. Aby pomóc to zilustrować, użyję jako przykładu `OpenSSH`. Nazwa jednostki jest bez znaczenia, ponieważ składnia polecenia `systemctl` jest taka sama niezależnie od nazwy jednostki, z którą wchodzimy w interakcję. Możesz użyć `systemctl` do uruchomienia, zatrzymania lub zrestartowania instancji serwera `Apache`, serwera bazy danych, a nawet użyć go do zrestartowania całego stosu sieciowego (ang. *networking stack*). Polecenie `systemctl`, bez żadnych opcji czy parametrów, domyślnie uruchamia się z opcją `list-units`, która zrzuca listę jednostek do powłoki. Może tam być trochę namieszane, ale jeśli znasz nazwę jednostki, którą chcesz znaleźć, możesz przekierować strumień wyjścia do `grep` i szukać konkretnego łańcucha. Przydaje się to również w sytuacji, gdy znasz część nazwy jednostki, a nie jej dokładną nazwę:

```
systemctl | grep ssh
```

Jeśli chcesz sprawdzić stan urządzenia, najlepszym sposobem jest użycie słowa kluczowego `status`, wtedy zobaczysz kilka bardzo użytecznych informacji dotyczących urządzenia. Są to informacje: czy urządzenie jest uruchomione, czy jest włączone (co oznacza, że jest skonfigurowane do uruchamiania podczas startu systemu), jak również ostatnie wpisy w dzienniku dla urządzenia:

```
systemctl status ssh
```

To polecenie da w efekcie coś w rodzaju informacji przedstawionych na rysunku 7.9.

W większości przypadków możesz sprawdzić stan jednostek bez konieczności posiadania dostępu do konta `root`, ale wtedy możesz nie mieć wglądu do wszystkich dostępnych informacji. Na zrzucie ekranu widać kilka wpisów w dzienniku dla usługi `ssh`, ale niektóre jednostki nie pokazują tych wpisów bez `sudo`. Jeśli chodzi o jednostkę `ssh`, wpisy dziennika widzimy podczas sprawdzania stanu zarówno z, jak i bez `sudo`.

```

jay@ubuntu-server: ~
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2022-05-09 01:06:46 UTC; 2h 4min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
   Process: 736 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 768 (sshd)
     Tasks: 1 (limit: 2241)
    Memory: 7.6M
       CPU: 90ms
   CGroup: /system.slice/ssh.service
           └─768 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

May 09 01:06:46 ubuntu-server systemd[1]: Starting OpenBSD Secure Shell server...
May 09 01:06:46 ubuntu-server sshd[768]: Server listening on 0.0.0.0 port 22.
May 09 01:06:46 ubuntu-server systemd[1]: Started OpenBSD Secure Shell server.
lines 1-16

```

Rysunek 7.9. Sprawdzanie stanu urządzenia za pomocą `systemctl`

Tym, co możesz jeszcze zauważyć na rzucie ekranu, jest to, że nazwa jednostki `ssh` to w rzeczywistości `ssh.service`, ale nie musisz dołączać części `.service` do nazwy, ponieważ jest to przyjmowane domyślnie. Czasami podczas przeglądania stanu procesu za pomocą `systemctl` wyjście może być skondensowane, aby zaoszczędzić miejsce na ekranie. Aby tego uniknąć i zobaczyć pełne wpisy dziennika, dodaj opcję `-l`:

```
systemctl status -l ssh
```

Inną rzeczą, na którą powinieneś zwrócić uwagę, jest sekcja `vendor preset` jednostki. Większość pakietów w Ubuntu, które zawierają plik serwisowy dla `systemd`, włącza go automatycznie, ale inne dystrybucje zazwyczaj tego nie robią — nie uruchamiają i nie włączają domyślnie jednostek (np. CentOS). W naszym przypadku, gdy chodzi o `ssh`, możesz zobaczyć, że `vendor preset` jest ustawiony na `enabled` (włączony). Oznacza to, że po zainstalowaniu pakietu `openssh-server` jednostka `ssh.service` zostanie automatycznie włączona. Możesz to potwierdzić, sprawdzając linię `Active`, która mówi nam, że jednostka jest uruchomiona (`active (running)`). Linia `Loaded` wyjaśnia, że jednostka jest `enabled` (włączona), więc wiemy, że następnym razem, gdy uruchomimy serwer, `ssh` zostanie automatycznie uruchomiony. Chociaż zazwyczaj w Ubuntu jednostki `systemd` są włączane i uruchamiane automatycznie podczas ich instalacji z pakietu, to nadal może być różnie. Kiedy instalujesz nowy pakiet, sprawdź stan jednostki, by być świadomym jej ustawień.

Uruchamianie i zatrzymywanie jednostki jest równie proste; wszystko, co musisz zrobić, to użyć z `systemctl` odpowiedniego słowa kluczowego: `start` lub `stop` — w zależności od oczekiwanego efektu:

```
sudo systemctl stop ssh
sudo systemctl start ssh
```


Mamy dodatkowe słowa kluczowe, takie jak `restart` (który za jednym zamachem wykonuje polecenia z obu poprzednich przykładów), a niektóre jednostki posiadają nawet funkcję `reload`, która pozwala na aktywowanie nowych ustawień konfiguracyjnych bez konieczności zamykania całej aplikacji. Przykładem, kiedy jest to przydatne, jest Apache, który jest serwerem udostępniającym strony internetowe dla lokalnych lub zewnętrznych użytkowników. Jeśli zatrzymasz serwer Apache, wszyscy użytkownicy zostaną odłączeni od udostępnianej przez niego witryny. Jeśli dodajesz nową stronę, możesz użyć `reload` zamiast `restart`, a wtedy uaktywnisz nową konfigurację, którą dodałeś, bez wpływu na już istniejące połączenia. Serwerowi Apache przyjrzymy się w rozdziale 14., „Udostępnianie serwisów internetowych”, więc nie przejmuj się nim zbyt w tej chwili. Jest to po prostu dobry przykład jednostki, która korzysta z tej dodatkowej funkcjonalności. Ponieważ nie wszystkie jednostki umożliwiają wykorzystanie opcji `reload`, więc dla pewności powinieneś sprawdzić dokumentację aplikacji, która udostępnia daną jednostkę.

Ponieważ w poprzednich przykładach wspominałem o uruchamianiu i zatrzymywaniu jednostki OpenSSH, ciekawostką jest to, że te działania nie powodują rozłączenia bieżących otwartych sesji SSH. Jeśli zatrzymasz usługę `ssh`, połączenia nie zostaną przerwane. Otwarte połączenia będą utrzymywane, a zatrzymanie SSH tylko zapobiegnie nawiązywaniu nowych połączeń. Dlatego SSH różni się od innych jednostek (takich jak Apache) tym, że istniejące połączenia nie są wyłączone podczas restartu jednostki.

Jak już wspominałem, jeśli chcesz, aby jednostka była automatycznie uruchamiana podczas startu serwera, musi być włączona (`enabled`). W większości przypadków jednostki są włączane automatycznie, ale jeśli znajdziesz taką, która nie jest, możesz ją włączyć za pomocą słowa kluczowego `enable`:

```
sudo systemctl enable ssh
```

Równie łatwo jest wyłączyć jednostkę:

```
sudo systemctl disable ssh
```

Możesz jednocześnie włączyć jednostkę i ją uruchomić:

```
sudo systemctl enable --now ssh
```

Argument `--now` informuje `systemctl`, by uruchomił jednostkę natychmiast po ustawieniu opcji `enabled` bez oczekiwania na restart serwera lub na uruchomienie jej argumentem `start` w osobnym poleceniu.

Nawet jeśli `systemd` jest używany głównie do zarządzania jednostkami, to w rzeczywistości jest to cała platforma, która zarządza wieloma rzeczami w systemie Linux, w tym rozwiązywaniem DNS, siecią i innymi. `systemd` obsługuje nawet logowanie, jak również zapewnia nam polecenie `journalctl`, którego możemy użyć do przeglądania informacji o logowaniu (jest to również powód, dla którego wyjście `systemctl status ssh` było w stanie pokazać nam wpisy w logu).

Omówiliśmy nieco logowanie w rozdziale 4., „Nawigacja i podstawowe polecenia”, a zrobimy to bardziej szczegółowo w rozdziale 22., „Rozwiązywanie problemów z serwerami Ubuntu” (który będzie zawierał również dalsze omówienie polecenia `journalctl`).

Na razie zrozum, że systemd jest dość obszerny, jeśli chodzi o liczbę rzeczy, którymi pomaga nam zarządzać. Jeśli jednak rozumiesz procesy uruchamiania, zatrzymywania, włączania, wyłączania i sprawdzania statusu jednostki, to na razie Ci to wystarczy.

Planowanie wykonywania zadań za pomocą polecenia `cron`

Wcześniej w tym rozdziale omówiliśmy uruchamianie procesów, umożliwianie im działania w tle oraz konfigurowanie uruchomienia wraz ze startem serwera. W niektórych przypadkach może zająć potrzeba wykonania zadania w określonym czasie, a nie ma potrzeby, by działało w tle. W tym miejscu pojawia się `cron`. Za pomocą polecenia `cron` można skonfigurować uruchamianie procesu, programu lub skryptu w określonym czasie z dokładnością do minuty. Każdy użytkownik może mieć swój własny zestaw konfiguracji `cron` (znany jako `crontab`) mogący wykonywać każde zadanie, które mógłby normalnie wykonywać użytkownik. Użytkownik `root` również dysponuje możliwością skonfigurowania `crontab`, z tym że pozwala on na wykonywanie zadań administracyjnych w obrębie całego systemu. Każdy `crontab` zawiera listę zadań `cron` (po jednym w każdej linii), którymi zajmiemy się za chwilę. Aby zobaczyć ustawienia `crontab` dla konta użytkownika, z którego aktualnie korzystasz, możesz użyć polecenia `crontab`:

```
crontab -l
```

Z opcją `-l` polecenie `crontab` pokaże Ci listę zadań dla użytkownika, który uruchomił polecenie. Jeśli wykonasz je jako `root`, zobaczysz `crontab` konta `root`. Jeśli wykonasz je jako użytkownik `jdoe`, zobaczysz `crontab` dla `jdoe`, i tak dalej. Jeśli chcesz zobaczyć `crontab` dla innego użytkownika niż Ty, możesz użyć opcji `-u` i określić użytkownika, ale będziesz musiał wykonać polecenie jako `root` (lub z `sudo`), aby zobaczyć `crontab` dla kogoś innego niż użytkownik, na którego koncie jesteś zalogowany:

```
sudo crontab -u jdoe -l
```

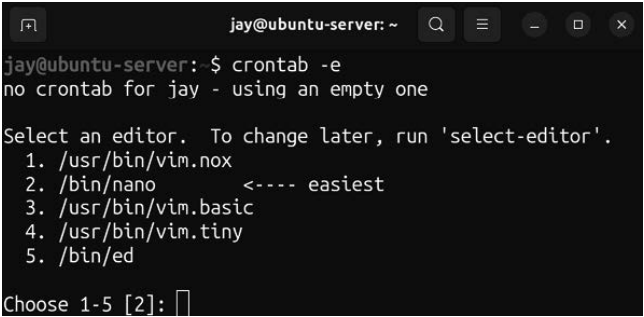
Domyślnie, dopóki nie stworzysz jednego lub więcej zadań, użytkownik nie będzie miał nic ustawione w `crontab`. Dlatego też prawdopodobnie zobaczysz komunikat o braku zadań, gdy sprawdzisz ustawienia dla swoich obecnych użytkowników:

```
no crontab for jdoe
```

Aby utworzyć zadanie cron, najpierw zaloguj się na to konto użytkownika, na którym ma być uruchamiane zadanie. Następnie wydaj polecenie:

```
crontab -e
```

Jeśli w systemie masz zainstalowanych więcej edytorów tekstu niż jeden, możesz zobaczyć wyjście podobne do przedstawionego na rysunku 7.10.



```
jay@ubuntu-server: ~
jay@ubuntu-server: $ crontab -e
no crontab for jay - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /usr/bin/vim.nox
 2. /bin/nano          <---- easiest
 3. /usr/bin/vim.basic
 4. /usr/bin/vim.tiny
 5. /bin/ed
Choose 1-5 [2]: 2
```

Rysunek 7.10. Wybieranie edytora, który ma być używany z poleceniem crontab

W tym przypadku po prostu naciskasz numer odpowiadający edytorowi tekstu, którego chcesz użyć podczas tworzenia *zadań* w cron. Aby ustawić zmienną środowiskową określającą konkretny edytor i móc edytować crontab za pomocą konkretnego edytora, możesz to zrobić następująco:

```
EDITOR=vim crontab -e
```

W tym przykładzie możesz zastąpić vim dowolnym edytorem tekstu, którego chcesz używać. W tym momencie powinieneś znaleźć się w edytorze tekstu z otwartym plikiem crontab. Domyślny plik crontab dla każdego użytkownika zawiera kilka pomocnych komentarzy, zawierających kilka użytecznych informacji na temat działania cron. Aby dodać nowe zadanie, należy przewinąć plik na sam dół (do miejsca zakończenia wszystkich komentarzy) i wstawić nową linię. Formatowanie wpisu jest tutaj bardzo specyficzne, a przykładowe komentarze w pliku dają Ci wskazówki co do struktury każdej linii. Chodzi konkretnie o tę część:

```
m h dom mon dow command
```

Każde zadanie w cron składa się z sześciu pól, a każde z nich jest oddzielone co najmniej jedną spacją lub tabulatorem. Jeśli użyjesz więcej niż jednej spacji lub tabulatora, cron jest wystarczająco inteligentny, aby poprawnie przetworzyć plik. W pierwszym polu mamy minutę, w której chcielibyśmy, aby zadanie się pojawiło. W drugim polu umieszczamy godzinę w formacie 24-godzinnym, od 0 do 23. Trzecie pole odpowiada za dzień miesiąca. W tym polu można umieścić 5 (5. dzień miesiąca), 23 (23. dzień miesiąca) itd. Czwarte pole odpowiada miesiącowi, np. 3 dla marca lub 12 dla grudnia. Piąte pole to dzień tygodnia, numerowany od 0 do 6, w formacie od niedzieli do soboty. Wreszcie w ostatnim polu znajduje się polecenie, które ma być wykonane. Kilka przykładowych linii crontab wygląda następująco:

```
3 0 * * 4 /usr/local/bin/cleanup.sh
* 0 * * /usr/bin/apt update
0 1 1 * * /usr/local/bin/run_report.sh
```

W pierwszym przykładzie skrypt *cleanup.sh*, znajdujący się w */usr/local/bin*, będzie uruchamiany o godzinie 12:03 w każdy czwartek. Możemy to odczytać, ponieważ kolumna minut ma ustawione 3, kolumna godzin 0 (północ), kolumna dni 4 (czwartek), a kolumna poleceń zawiera pełną ścieżkę do pliku */usr/local/bin/cleanup.sh*.

Co to znaczy, że polecenie zawiera **pełną ścieżkę**? Generalnie polecenie wpisane w formie pełnej ścieżki oznacza, że jest podana cała ścieżka do plików binarnych odpowiedzialnych za wykonanie polecenia. W drugim przykładzie mogliśmy po prostu wpisać dla polecenia `apt update` i to prawdopodobnie zadziałałoby dobrze. Jednak brak pełnej ścieżki jest uważany za zły zwyczaj w przypadku konfiguracji cron. Może tak być, że polecenie zadziała bez podania pełnej ścieżki, ale jest to uzależnione od tego, czy aplikacja zostanie znaleziona w ścieżkach zmiennych środowiskowych użytkownika, który ją wywołuje. Ponieważ różne serwery mogą być skonfigurowane w różny sposób, taka konfiguracja (bez pełnej ścieżki) może zadziałać lub nie — w zależności od tego, jak skonfigurowana jest powłoka. Jeśli wpiszesz pełną ścieżkę, zadanie powinno działać niezależnie od tego, jak skonfigurowana jest powłoka bazowa.

Jeśli nie wiesz, jaka jest pełna ścieżka polecenia, to możesz ją uzyskać, używając polecenia `which`. Polecenie to użyte z nazwą polecenia, które chcesz uruchomić, da Ci pełną ścieżkę dla polecenia, jeśli znajduje się ono w Twoim systemie.

Co do drugiego przykładu: uruchamiamy `/usr/bin/apt update`, aby zaktualizować indeks repozytorium naszego serwera każdego ranka o północy. Gwiazdki w każdym wierszu odnoszą się do dowolnej wartości, więc jeśli w kolumnie minut będą *, oznacza to po prostu, że to zadanie wykonywać się będzie co minutę. W zasadzie jedynym polem, które wymaga objaśnienia, jest pole godziny, które ustawiliśmy na 0, co reprezentuje godzinę 12:00.

W trzecim przykładzie uruchamiamy skrypt `/usr/local/bin/run_report.sh` pierwszego dnia każdego miesiąca o godzinie 01:00. Jak możesz zauważyć, ustawiliśmy trzecią kolumnę (**dzień w miesiącu**) na 1, co odpowiada 1 lutego, 1 marca itd. Zadanie to zostanie uruchomione każdego pierwszego dnia miesiąca, ale tylko o godzinie 01:00, ponieważ ustawiłem wartości tylko w pierwszej i drugiej kolumnie, a one reprezentują odpowiednio minutę i godzinę.

Kiedy skończysz edytować plik `crontab` użytkownika i zapiszesz go, cron zostanie zaktualizowany i od tego momentu będzie wykonywał zaplanowane zadania w wybranym przez Ciebie czasie. `Crontab` będzie wykonywany zgodnie z aktualnie ustawionym czasem i datą na Twoim serwerze, więc powinienes się upewnić, że ich ustawienia są poprawne, w przeciwnym razie Twoje zadania wykonają się w innym czasie, niż zakładałeś. Aktualną datę i czas na swoim serwerze możesz sprawdzić, po prostu wydając polecenie `date`.

Najlepszym sposobem na opanowanie tworzenia zadań za pomocą cron jest (jak zawsze) praktyka. Drugie przykładowe zadanie cron dobrze nadaje się do eksperymentowania, ponieważ aktualizowanie indeksu repozytorium nie wyrządzi żadnej szkody w systemie.

Podsumowanie

W tym rozdziale dowiedziałeś się, jak zarządzać procesami. Zacząłeś od zapoznania się z poleceniem `ps`, którego można użyć do wyświetlenia listy aktualnie uruchomionych procesów. Dowiedziałeś się również trochę o zarządzaniu zadaniami, a także o zabijaniu procesów, które z tego czy innego powodu zachowują się niewłaściwie. Omówiłem również metody zmiany priorytetu dla procesu, co pozwala na uzyskanie pełnej kontroli nad tym, które procesy otrzymują więcej czasu na przetwarzanie danych, a także dowiedziałeś się, w jaki sposób można zaplanować uruchamianie zadań w określonym czasie i dniu za pomocą cron.

W rozdziale 8, „Monitorowanie zasobów systemu”, dowiesz się, jak można monitorować dostępne na serwerze zasoby. Dotyczy to sprawdzenia wykorzystania dysku, pamięci i przestrzeni wymiany. Zapoznasz się też z kilkoma narzędziami, które mogą sprawić, że zarządzanie zasobami stanie się proste.

Dodatkowe filmy związane z tematem

- Rozpoczęcie pracy z tmux (LearnLinuxTV):
<https://linux.video/tmux-guide>
- Procesy działające w tle i na pierwszym planie w systemie Linux (LearnLinuxTV):
<https://linux.video/bg-fg>

Lektura uzupełniająca

- Ham Vocke, *A Quick and Easy Guide to tmux*:
<https://learnlinux.link/tmux-article>
- Tmux Cheat Sheet & Quick Reference:
<https://learnlinux.link/tcs>
- crontab guru:
<https://learnlinux.link/ctg>

Skorowidz |

A

adres rozgłoszeniowy,
broadcast, 279
adresy IP
dla sieci kapsuł, 461
przydzielanie, 276, 281
statyczne, 257
agent SSH, 271
akcja, play, 381
aktualizacje
wsparcia dla sprzętu, HWE,
112
wydań stabilnych, 92
zabezpieczeń, 558
alias, 157
AMI, Amazon Machine Images,
509
tworzenie, 511
analiza
wykorzystania dysku, 198
wykorzystania zasobów, 211
źródła problemu, 587
Ansible, 371
automatyzacja
konfigurowania
serwerów, 369
wdrożenia serwera, 385
wdrożeń, 547
instalowanie, 376
konfiguracja ustawień, 379
metoda pull, 389
Apache
instalacja, 342
instalacja dodatkowych
modułów, 349
konfiguracja, 342
konfiguracja TLS, 351
APT, Advanced Package Tool, 97

Auto Scaling, 512
automatyczne
instalowanie poprawek, 562
skalowanie wdrożeń, 512,
514
automatyzacja
konfigurowania serwerów,
369
tworzenia obrazów
Dockera, 432
wdrożeń
Ansible, 547
instancji EC2, 537
Terraform, 547
w chmurze, 527
awarie, 609
AWS, Amazon Web Services,
474, 480
Auto Scaling, 480, 512
certyfikacja, 524
dodawanie alertu
rozliczeniowego, 521
dokumentacja, 525
ekran główny, 488
funkcja CloudFormation, 530
informacje rozliczeniowe,
521
instancja serwera Ubuntu,
499
klucz API dla Terraform,
534
konfigurowanie
użytkownika
administracyjnego, 492
narzędzie OpsWorks, 531
planowanie uruchamiania
instancji EC2, 522
przegląd, health check, 481
przerywanie działania
instancji EC2, 523

samouzdrawianie, auto
healing, 481
strona główna, 485
usługi, 482
usuwanie niepotrzebnych
kopii zapasowych, 522
wybór planu wsparcia, 486
wybór regionu, 494
zabezpieczenia użytkownik
ów, 488
zakładanie konta, 485

B

bazy danych, 320
blokowanie, 337
dostęp, 333
konfiguracja, 326
serwer, 321
serwer repliki, 335, 338
synchronizacja, 340
tworzenie, 331
tworzenie tabel, 333
uprawnienia, 333
usuwanie, 334
blokowanie sudo, 581
błąd standardowy, stderr, 160
brama, gateway, 295

C

chmura, 474
obliczeniowa, 475, 476
cron, 168
CSR, Certificate Signing
Request, 355
CVE, Common Vulnerabilities
and Exposures, 557

D

demony, daemons, 187
 apache2, 343
 smbd, 305

DHCP, Dynamic Host Control Protocol, 277

diagnozowanie uszkodzonej pamięci RAM, 605

DNS, Domain Name System, 262, 287

Docker, 420
 automatyzacja tworzenia obrazów, 432
 ENTRYPOINT, 425
 instalacja, 422
 instalacja Apache, 429
 kontener Apache, 430
 wyświetlanie listy zainstalowanych obrazów, 425
 zarządzanie kontenerami, 423

Dockerfiles, 432

dołączalne moduły
 uwierzytelniania, PAM, 76

dowiązania
 symboliczne, 145
 twarde, 145

dysk
 analiza wykorzystania, 198
 formatowanie i partycjonowanie, 220
 wyświetlanie wykorzystania, 195

dyski twarde
 NVME, 197, 219
 SATA, 197

dziennik
 aplikacji, application log, 591
 autoryzacji, authorization log, 592
 błędów, error log, 347
 dostępu, access log, 347
 systemowy, system log, 591, 593

dzierżawy statyczne, 280

E

EBS, Elastic Block Store, 482

EC2, Elastic Compute Cloud, 482
 dodawanie danych użytkownika, 505
 domyślna strona Apache, 509
 generowanie pary kluczy, 501
 okno główne, 500
 opcja Ubuntu, 500
 opcje dla szablonu startowego, 513
 opcje przechowywania, 504
 sprawdzanie stanu instancji, 506, 510
 tworzenie grupy autoskalowania, 514–518
 ustawienia instancji, 507
 opcji, 503
 roli IAM, 506
 sieci, 504
 wybór typu instancji, 501

edytor tekstu
 nano, 133
 Vim, 135

edytowanie plików, 132

EKS, Elastic Kubernetes Service, 483

elastyczna chmura obliczeniowa, EC2, 482

ELB, Elastic Load Balancer, 483

epoka Uniksa, Unix epoch, 64

F

Fail2ban
 instalacja i konfiguracja, 568

FHS, Filesystem Hierarchy Standard, 122, 226

Flannel, 464

format
 UNC, 306
 YAML, 381

formatowanie partycji, 224
 urządzeń pamięci masowej, 220
 wolumenów logicznych, 241

G

GCP, Google Cloud Platform, 475

generowanie pary kluczy SSH, 269

Git, 373
 zarządzanie konfiguracją, 612

GPT, GUID Partition Table, 222

graficzny interfejs użytkownika, GUI, 132, 178

grupa, 51, 69
 autoskalowania, auto scaling group, 514–518
 docelowa, target group, 517
 kontrolna, control group, 421
 podstawowa, primary group, 71
 pomocnicza, secondary group, 71
 wolumenów, volume group, 237
 zabezpieczeń, security group, 484, 543

H

harmonogram tworzenia kopii zapasowych, 618

hasła, 73
 okres ważności, 74
 ustalanie zasad, 76

Homebrew, 448

host wirtualny, virtual host, 344

HWE, hardware enablement, 112

- I
- IaC, Infrastructure as Code, 370, 530
 - IAM, Identity and Access Management, 483
 - IDE, Integrated Development Environment, 137
 - identyfikator
 - grupy, GID, 57
 - konta, account ID, 494
 - procesu, PID, 175
 - unikalny UUID, 229
 - użytkownika, UID, 57
 - informacje
 - o osieroconych pakietach, 110
 - o pakiecie, 103
 - infrastruktura jako kod, IaC, 370, 530
 - instalacja
 - aktualizacji zabezpieczeń, 558
 - Apache, 98, 342
 - certyfikatów TLS, 355
 - Dockera, 422
 - Fail2ban, 568
 - klucza GNU Privacy Guard, 106
 - Kubernetes, 460
 - MariaDB, 323
 - MicroK8s, 447
 - w macOS, 448
 - w Windows, 449
 - Nextcloud, 361
 - NGINX, 357
 - OpenSSH, 265
 - Terraform, 324
 - Ubuntu Server, 35
 - na Raspberry Pi, 45
 - pobieranie systemu, 27
 - tworzenie
 - rozdzielnego dysku USB, 30
 - wybór urządzenia, 24
 - wymagania techniczne, 22
 - instalowanie oprogramowania, 96
 - interfejsy sieciowe, 252
 - i-węzeł, inode, 145

J

 - jednostki, units, 188

K

 - kapsuły, 461
 - catalog
 - /etc/skel, 66
 - domowy, home directory, 117
 - macierzysty, document root, 344
 - roboczy, working directory, 117
 - catalogi
 - kopiowanie, 129
 - przenoszenie, 129
 - zmiana nazw, 129
 - klonowanie maszyny wirtualnej, 414
 - klucz
 - prywatny, 269
 - publiczny, 269, 270
 - klucze SSH, 269, 502
 - konfiguracja
 - Apache, 342
 - bramy internetowej, 295
 - dostępu administratora, 77
 - Fail2ban, 568
 - klastra Kubernetes, 454
 - LXD, 435
 - Nextcloud, 361
 - NGINX, 357
 - pliku inwentarza, 379
 - roli IAM, 496
 - serwera
 - bazy danych, 326, 335
 - będącego klientem, 381
 - DNS, 288, 290
 - KVM, 398
 - plików, 301
 - WWW, 385
 - TLS, 352
 - udziałów NFS, 307
 - urządzenia MFA, 491
 - usług sieciowych, 276
 - ustawień Ansible, 379
 - zapory sieciowej, 575
 - konsola
 - AWS Management Console, 485, 488
 - kontenery
 - zestranianie, 441, 442
 - konteneryzacja, containerization, 418
 - kopie zapasowe
 - plan tworzenia, 618
 - różnicowe, differential backups, 623
 - kopiowanie klucza publicznego, 270
 - Kubernetes, 444, *Patrz także* MicroK8s
 - inicjalizacja klastra, 462
 - instalowanie, 460
 - konfigurowanie klastra, 454
 - sprawdzanie stanu kapsuł, 463
 - ustawienia wstępne, 456
 - wdrażanie, 444
 - wdrażanie kontenerów, 466
 - KVM, Kernel-based VM, 396

L

 - licznik średniej CPU, 213
 - linia poleceń, 154
 - LUKS, Linux Unified Key Setup, 578
 - odszyfrowywanie, 578
 - szyfrowanie, 578
 - LVM, Logical Volume Manager, 216, 236
 - migawki, 244
 - usuwanie wolumenów, 243
 - LXD, 420
 - konfiguracja, 435
 - zarządzanie kontenerami, 434

Ł

łańcuch podatności,
vulnerability chain, 554

M

macierz RAID, 220
MariaDB
 instalacja, 323
 pliki konfiguracyjne, 326
 zabezpieczanie serwera, 572
 zarządzanie, 328
maszyna wirtualna, VM, 26, 377
 oparta na jądrze, KVM, 396
maszyny wirtualne
 klonowanie, 413
 mostkowanie, 409
 tworzenie, 404
 zarządzanie
 z programu virt-
 manager, 400
 z wiersza poleceń, 415
MBR, Master Boot Record, 222
mechanizm równoważenia
 obciążenia, 517
menedżer maszyn
 wirtualnych, VMM, 400
MFA, Multi-Factor
 Authentication, 489
 aktywowanie, 490
 konfigurowanie urządzenia,
 491
 wybór typu urządzenia, 490
MicroK8s, 446, 451
 instalowanie, 447
 w macOS, 448
 w Windows, 449
migawki LVM, LVM snapshots,
 244
monitorowanie
 pamięci masowej, 195
 wydajności serwera, 207
 wykorzystania pamięci, 202
montowanie wolumenów, 225
most, bridge, 409
mostkowanie maszyn
 wirtualnych, 409
Multipass, 452
MySQL, 321

N

narzędzie
 Clonezilla, 622
 cryptsetup, 579
 do pakietów, APT, 97
 htop, 211, 213
 iotop, 604
 Memtest86+, 605
 NCurses Disk Usage, 200
 OpenSSH, 265
 rsync, 312, 619, 623
 Uncomplicated Firewall,
 576
 virt-manager, 400
 dodawanie połączenia,
 401
 dodawanie puli pamięci
 ISO, 403
 mostkowanie maszyny
 wirtualnej, 411
 tworzenie maszyny
 wirtualnej, 405
nazwa hosta, 249
Nextcloud
 dodawanie nowego
 użytkownika, 367
 instalowanie, 361
 konfigurowanie, 361
 strona konfiguracyjna, 365,
 366
NFS, 307
NGINX
 instalacja, 357
 konfiguracja, 357
NodePort, 470
nośnik instalacyjny, 27, 621
notacja CIDR, 279

O

obiektowa pamięć masowa,
 483
obszar wymiany, swap, 204
odczyt, read, 83
odzyskiwanie, 621
OpenSSH, 265, 564
 instalacja, 265
 wydawanie poleceń, 267

operator AND, 156
opiekun pakietu, package
 maintainer, 91

P

pakiet
 apache2, 110, 343, 429
 bind, 288
 containerd, 457
 cryptsetup, 579
 dnstools, 290
 docker.io, 422
 dselect, 109
 git, 373, 613
 htop, 173, 382
 iotop, 603
 iproute2, 255
 isc-dhcp-server, 282
 kubeadm, 460
 kubectl, 461
 kubenet, 461
 libapache2-mod-php, 103
 libapache2-mod-php8.1,
 350, 363
 libapache2-modpython, 351
 libvirt, 398
 lvm2, 238
 mariadb-client-10.6, 325
 mariadb-server, 323, 336
 mysql_secure_installation,
 324
 mysql-server, 323
 net-tools, 255
 nfs-common, 310
 nfs-kernel-server, 308
 ntp, 599
 openssh-client, 316
 openssh-server, 97, 267
 samba, 301
 snap, 447
 tig, 616
 tmux, 157, 261
 traceroute, 596
 ufw, 576
 vim-nox, 135, 433
pakiety
 Debiana, 93
 polecenie apt, 97
 przywracanie, 108

- instalowanie, 96
- odinstalowywanie, 96
- prywatne archiwa, 106
- repozytoria, 104
- wyszukiwanie, 102
- osierocone, 110
- RPM, 93
- Snap, 94
 - polecenie snap, 100
- uniwersalne, universal packages, 94
- PAM, Pluggable Authentication Module, 76
- pamięć
 - Error Correction Code, 25
 - masowa, 216
 - dodawanie wolumenów, 217
 - monitorowanie
 - wykorzystania, 202
 - RAM, 203
- partycja wymiany, swap, 35
- partycjonowanie
 - dysku, 33
 - urządzeń pamięci masowej, 220
- pętla
 - while, 164
 - zwrotna, loopback, 250
- pisanie skryptu, 160
- plan
 - odzyskiwania po awarii, 609
 - Terraform, 540
- plik
 - /etc/fstab, 228, 458
 - dodawanie wpisu, 230
 - /etc/hosts, 251
 - /etc/passwd, 60
 - /etc/shadow, 60, 62
 - config, 273
 - Dockerfile, 432
 - konfiguracyjny Terraform, 537
 - strefy, zone file, 288, 290
 - strefy głównej, 291
 - wymiany, swap file, 35, 205
- pliki
 - .tf, 537
 - dziennika, 127, 589
 - edytowanie, 132
 - konfiguracyjne, 66
 - konfiguracyjne MariaDB, 326
 - kopiowanie, 129
 - przeglądanie zawartości, 124
 - przenoszenie, 129
 - przesyłanie, 312, 316
 - udostępnianie, 301
 - zmiana nazw, 129
- podatności i zagrożenia, CVE, 557
- podpowłoka, sub-shell, 167
- podsieć, subnet, 259, 278
- polecenia systemu Linux, 116
- polecenie
 - adduser, 56
 - alias, 157, 158
 - ansible-pull, 389, 391
 - apt, 93, 97
 - apt dist-upgrade, 560
 - blkid, 229, 230
 - cat, 70, 124, 143
 - cd, 118, 129
 - chage, 74, 75
 - chgrp, 87
 - chmod, 85, 86, 161, 206
 - chown, 87
 - commit git, 616
 - cp, 129–131
 - cron, 191
 - crontab, 191, 192
 - curl, 469
 - df, 196, 198, 601
 - dig, 598
 - dmesg, 594
 - dnf, 93
 - docker, 423
 - dpkg, 109
 - du, 199, 602
 - egrep, 397
 - export, 539
 - fallocate, 206
 - fdisk, 218–221, 224
 - fg, 172, 173
 - find, 144, 145
 - git checkout, 617
 - git push, 616
 - git status, 616
 - gpasswd, 72
 - grep, 125, 127, 144
 - groupadd, 71
 - groupdel, 71
 - history, 152
 - host, 598
 - hostname, 250
 - hostnamectl, 250
 - htop, 604
 - ifconfig, 255
 - install, 157
 - iotop, 603
 - ip, 253
 - jobs, 172
 - journalctl, 589, 590
 - kill, 185, 187
 - killall, 185, 187
 - kubeadm init, 461
 - kubectl, 469
 - kubectl get nodes, 465
 - less, 125, 592
 - ln, 146, 147
 - ls, 65, 119, 129, 147
 - lsblk, 219
 - lspci, 599
 - lvconvert, 245
 - lvcreate, 240, 245
 - lvdisplay, 241
 - lvextend, 242, 243
 - lvremove, 243, 246
 - lxd init, 435
 - mariadb, 153
 - microk8s, 452
 - mkdir, 226, 231
 - mkfs.ext4, 241
 - mkswap, 206
 - more, 125
 - mount, 227
 - mv, 131
 - myvol1, 245
 - nano, 134, 161, 231, 297
 - ncdu, 602
 - netplan, 261, 410
 - nice, 180–183
 - opcje, 175
 - passwd, 73
 - ps, 172, 174
 - pvsdisplay, 239
 - pwd, 117, 118
 - read, 160

- połączenie
 - renice, 180–183
 - resize2fs, 242, 243
 - restart, 411
 - rm, 60, 119
 - rsync, 166, 313, 620
 - scp, 316
 - search, 102
 - show, 103
 - snap, 100, 447
 - ss, 339, 553
 - ssh, 564
 - ssh-agent, 272
 - ssh-copy-id, 271, 273
 - ssh-keygen, 272
 - su, 67
 - sudo, 53, 77
 - sudo !, 155
 - swapon, 205, 206
 - systemctl, 188–190, 267, 289, 293
 - tail, 127, 286, 591
 - tee, 297
 - terraform apply, 542
 - terraform destroy, 546
 - terraform init, 539
 - terraform plan, 540
 - tig, 617
 - touch, 120, 129, 602
 - traceroute, 596
 - update, 109
 - uptime, 208
 - useradd, 54
 - userdel, 58
 - usermod, 71, 78
 - vgcreate, 240
 - vgextend, 243
 - vgremove, 244
 - virsh, 415
 - visudo, 78, 581
 - which scp, 316
 - zless, 594
 - połączenia SSH, 273
 - powierzchnia ataku, attack surface, 552
 - powłoka Bash, 151
 - funkcja historii, 152
 - PPA, Personal Package Archive, 106
 - priorytet procesu, 180
 - problemy
 - ocena zasięgu, 585
 - poszukiwanie źródła, 587
 - z pamięcią RAM, 605
 - z siecią, 595
 - z trasowaniem, 595
 - z zasobami, 601
 - ze znalezieniem przyczyny, 589
 - proces zombie, 186
 - procesy
 - działające nieprawidłowo, 185
 - systemowe, 187
 - zmienianie priorytetów, 180
 - program Etcher, 31
 - protokół
 - LDAP, 51
 - SMB, 300
 - prywatne archiwum pakietów, PPA, 106
 - prywatny serwer wirtualny, VPS, 26, 348
 - przeglądanie
 - dzienników systemowych, 589
 - tablicy tras, 596
 - przekierowanie portów, port redirection, 428
 - przełączanie się pomiędzy kontami, 67
 - przydzielanie adresów IP, 276, 281
 - przywracanie pakietów Debiana, 108
- ## Q
- QEMU, Quick Emulator, 396
- ## R
- rekord nazwy kanonicznej, 293
 - repozytoria pakietów, 104
 - dodawanie, 105
 - repozytorium Ansible, 374
 - rezerwacja DHCP, DHCP reservation, 257
 - rola IAM
 - tworzenie, 496
 - zasada
 - AmazonSSMFullAccess, 498
 - root, 52, 77
 - Route 53, 483
 - rsync
 - przesyłanie plików, 312
- ## S
- S3, Simple Storage Service, 483
 - scenariusz, playbook, 381
 - scenariusze konfiguracji, 376
 - SCP, Secure Copy, 316
 - przesyłanie plików, 316
 - serwer
 - bazy danych MariaDB, 321, 335
 - buforujący nazwy, 288
 - DHCP, 277
 - przydzielanie adresów IP, 281
 - DNS, 263, 287
 - konfigurowanie, 288, 290
 - obsługa intranetu, 290
 - zewnętrzny, 288
 - KVM
 - konfiguracja, 398
 - lustrzany, mirror, 91
 - plików, 300
 - pośredniczący, proxy, 357
 - Samba
 - udostępnianie plików, 301
 - Ubuntu
 - instalacja, 21, 35
 - jako serwer fizyczny, 24
 - na komputerze
 - stacjonarnym, 25
 - na laptopie, 25
 - na maszynie wirtualnej, 26
 - na Raspberry Pi, 27, 45
 - określanie roli, 22
 - w chmurze, 26
 - WWW, 385

serwery
 nadawanie nazwy hosta, 249
 rozwiązywanie problemów, 584
 zabezpieczenia, 551
 serwis Docker Hub, 422
 sesja instancji, 508
 Session Manager
 konfigurowanie roli IAM, 496
 shebang, 161
 skrypt
 Bash, 161
 wykonujący
 kopię zapasową, 166
 powłoki, shell script, 57
 SOA, Start of Authority, 292
 sprawdzanie poddrzew, subtree checking, 309
 SSL, Secure Sockets Layer, 351
 standardowe
 wejście, stdin, 142, 160
 wyjście, stdout, 142, 160
 strefy
 dostępności, availability zones, 495
 lokalne, local zones, 495
 strumienie, 142
 sygnały, 185
 system plików, file system, 121
 ext4, 225
 standard FHS, 122
 Systems Manager, 496
 szablony
 maszyn wirtualnych, 413
 uruchamiania, launch template, 513
 szybki emulator, QEMU, 396
 szyfrowanie, 578
 w spoczynku, 23

Ś

średnie obciążenia, load average, 207

T

tablica tras, 596
 tajne klucze dostępu, secret access key, 539
 Terraform
 automatyzacja wdrożenia instancji EC2, 537
 automatyzacja wdrożeń, 547
 inicjalizacja, 540
 instalowanie, 532
 niszczenie nieużywanych zasobów, 546
 tworzenie użytkownika IAM, 535
 zarządzanie grupami zabezpieczeń, 543
 TLS, Transport Layer Security, 351
 trasowanie, routing, 595
 tryb ściągania, pull, 372, 389
 tworzenie
 AMI, 511
 grupy autoskalowania, 514–518
 instancji serwera Ubuntu, 499
 kont użytkowników, 54
 konta w AWS, 484
 kopii zapasowych, 234, 618
 maszyn wirtualnych dzięki klonowaniu, 413
 maszyny wirtualnej, 404
 partycji, 221
 repozytorium Git, 373
 roli IAM, 497
 szablonu uruchamiania, 513
 Ubuntu AMI, 509

U

Ubuntu AMI, 509
 udziały NFS, 307
 UNC, Universal Naming Convention, 306

uprawnienia
 do odczytu, 81
 zmienianie, 85
 usługa
 Canonical Livepatch, 562
 CloudFront, 495
 EC2, 500
 ECS, 422
 IAM, 489
 libvirtd, 398
 NodePort, 470
 Session Manager, 496
 ssh, 590
 usługi
 dla NFS, Services for NFS, 301
 sieciowe, 276
 usuwanie kont użytkowników, 58
 uwierzytelnianie
 przez gniazda systemu Unix, 324
 wieloskładnikowe, MFA, 489
 użytkownicy, 51

V

virt-manager, 400
 dodawanie połączenia, 401
 dodawanie puli pamięci ISO, 403
 mostkowanie maszyny wirtualnej, 411
 tworzenie maszyny wirtualnej, 405
 VirtualBox, 397
 VM, Virtual Machine, 26, 396
 VMM, Virtual Machine Manager, 400
 VoIP, Voice over IP, 277
 VPC, Virtual Private Cloud, 482
 VPS, Virtual Private Server, 26, 67, 348, 475

W

wdrażanie
 klastra Kubernetes, 444
 kontenerów, 466
 Ubuntu, 35
 AMI, 509
 automatyczne
 skalowanie, 512
 EC2, 496
 w chmurze, 474
 wiersz poleceń, 118, 150
 zarządzanie maszynami
 wirtualnymi, 415
 wirtualizacja, 395
 wirtualna chmura prywatna,
 VPC, 482
 wirtualny serwer prywatny,
 VPS, 26, 67, 348, 475
 właściciel obiektów
 zmienianie, 87
 wolumen
 fizyczny, physical volume,
 237
 kopia zapasowa, 234
 logiczny, logical volume,
 238
 formatowanie, 241
 montowanie i
 odmontowywanie, 225
 przywracanie, 234
 wydajność serwera, 207
 wydanie
 bez wsparcia
 długoterminowego, 29
 ze wsparciem
 długoterminowym, 28

wyszukiwanie pakietów, 102
 wyświetlanie bieżącego
 katalogu, 117

Z

zabezpieczanie
 OpenSSH, 564
 serwera, 551
 serwera MariaDB, 572
 zabezpieczenia, 558
 zadanie, job, 172
 planowanie, 191
 zapis, write, 83
 zapobieganie awariom, 609
 zaporą sieciową, firewall, 575
 zarządzanie
 bazami danych, 320
 bazami danych MariaDB,
 328
 grupami, 69
 grupami zabezpieczeń, 543
 hasłami, 73
 interfejsami sieciowymi,
 252
 kluczami SSH, 269
 konfiguracją, 370, 612
 kontenerami Dockera, 423
 kontenerami LXD, 434
 obszarem wymiany, 204
 pakietami, 90
 Debiana, 97
 Snap, 100
 pamięcią serwera, 202
 plikami i katalogami, 129
 procesami systemowymi,
 187
 repozytoriami pakietów,
 104
 wolumenami
 pamięci masowej, 216
 zadaniami, 170
 zasada najmniejszych
 przywilejów, 491, 556, 610
 zasady dotyczące hasel, 76
 zasilacz awaryjny, UPS, 25
 zestrzajanie kontenerów,
 container orchestration, 441
 zintegrowane środowisko
 programistyczne, IDE, 137
 zmiana
 hasła klucza, 272
 hasła SSH, 273
 priorytetu procesu, 180
 uprawnień, 85
 właściciela obiektów, 87
 zmienna wyjściowa, output
 variable, 544
 zmienne, 158
 znak
 dolara (\$), 159
 kratki (#), 161
 procentu (%), 330
 ukośnika (/), 196
 zwrot z inwestycji, ROI, 477

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Ubuntu Server: wszystko, czego oczekujesz od najlepszych serwerów!

Ubuntu Server zdobył popularność i uznanie. To zrozumiałe, pozwala bowiem na uzyskanie wysokiej elastyczności i wydajności przy niewielkich kosztach. Od czasu pierwszego wydania w 2004 roku Ubuntu został wzbogacony o potężne i nowoczesne funkcje dla administratorów. Dziś jest najczęściej wdrażaną dystrybucją Linuksa w sieci. System ten jest używany przez organizacje o różnej wielkości i zasobności, w tym przez najbardziej znane korporacje.

To książka przeznaczona dla czytelników o średnich lub zaawansowanych umiejętnościach postępowania się systemem Linux. Dzięki niej pogłębisz wiedzę o systemach linuksowych i zarządzaniu serwerami działającymi pod kontrolą systemu Ubuntu w rzeczywistych wdrożeniach produkcyjnych. Dowiesz się, od czego zacząć instalację systemu, a następnie jak wprowadzać gotowe rozwiązania w środowisku produkcyjnym. Poznasz w ten sposób narzędzia do administrowania pracą sieci zarówno w małym biurze, jak i w centrum danych. Nauczysz się wdrażać usługi sieciowe, w tym DHCP i DNS. Zobaczysz również, jak skonteneryzować aplikacje za pomocą LXD, aby zmaksymalizować wydajność, i jak budować klastry Kubernetes. W tym wydaniu zaprezentowano Ubuntu w wersji 22.04 LTS, w której zastosowano najnowsze technologie oparte na Linuksie.

Najciekawsze zagadnienia:

- instalacja systemu Ubuntu Server na fizycznych serwerach i na Raspberry Pi
- wdrażanie aplikacji we własnych kontenerach i skalowanie infrastruktury
- automatyzacja wdrożeń i ich konfiguracja
- konteneryzacja aplikacji z wykorzystaniem LXD
- najlepsze praktyki i techniki rozwiązywania problemów

Jay LaCroix jest nauczycielem, inżynierem i ekspertem w dziedzinie serwerów Linuksa. Jest też autorem setek filmów instruktażowych, które zdobyły ogromne uznanie i popularność. W wolnym czasie gra w gry retro i ćwiczy sztuki walki.

	KOD KORZYŚCI Sięgnij po więcej! ▶	
 helion.pl	ISBN 978-83-8322-592-0	
 HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	 9 788383 225920	
Cena: 129,00 zł		

Packty